

## File 1 - utils.py:

```

1: (0)         """Utilities to create and set the config.
2: (0)         The main class exported by this module is :class:`ManimConfig`. This class
3: (0)         contains all configuration options, including frame geometry (e.g. frame
4: (0)         height/width, frame rate), output (e.g. directories, logging), styling
5: (0)         (e.g. background color, transparency), and general behavior (e.g. writing a
6: (0)         movie vs writing a single frame).
7: (0)         See :doc:`/guides/configuration` for an introduction to Manim's configuration
8: (0)         system.
9: (0)
10: (0)        """
11: (0)        from __future__ import annotations
12: (0)        import argparse
13: (0)        import configparser
14: (0)        import copy
15: (0)        import errno
16: (0)        import logging
17: (0)        import os
18: (0)        import re
19: (0)        import sys
20: (0)        from collections.abc import Mapping, MutableMapping
21: (0)        from pathlib import Path
22: (0)        from typing import TYPE_CHECKING, Any, ClassVar, Iterable, Iterator, NoReturn
23: (0)        import numpy as np
24: (0)        from manim import constants
25: (0)        from manim.constants import RendererType
26: (0)        from manim.utils.color import ManimColor
27: (0)        from manim.utils.tex import TexTemplate
28: (0)        if TYPE_CHECKING:
29: (4)            from enum import EnumMeta
30: (4)            from typing_extensions import Self
31: (4)            from manim.typing import StrPath, Vector3D
32: (0)        __all__ = ["config_file_paths", "make_config_parser", "ManimConfig",
33: (0)        "ManimFrame"]
34: (0)    This
35: (4)    function returns the locations in which these files are searched for. In
36: (4)    the
37: (4)    user-wide config file, and the folder-wide config file.
38: (4)    The library-wide config file determines manim's default behavior. The
39: (4)    user-wide config file is stored in the user's home folder, and determines
40: (4)    the behavior of manim whenever the user invokes it from anywhere in the
41: (4)    system. The folder-wide config file only affects scenes that are in the
42: (4)    same folder. The latter two files are optional.
43: (4)    These files, if they exist, are meant to loaded into a single
44: (4)    :class:`configparser.ConfigParser` object, and then processed by
45: (4)    :class:`ManimConfig`.
46: (4)    Returns
47: (4)    -----
48: (4)    List[:class:`Path`]
49: (8)    List of paths which may contain ``.cfg`` files, in ascending order of
50: (8)    precedence.
51: (4)    See Also
52: (4)    -----
53: (4)    :func:`make_config_parser`, :meth:`ManimConfig.digest_file`,
54: (4)    :meth:`ManimConfig.digest_parser`
55: (4)    Notes
56: (4)    -----
57: (4)    The location of the user-wide config file is OS-specific.
58: (4)    """
59: (4)    library_wide = Path.resolve(Path(__file__).parent / "default.cfg")
60: (8)    if sys.platform.startswith("win32"):
61: (8)        user_wide = Path.home() / "AppData" / "Roaming" / "Manim" /
62: (8)        "manim.cfg"
63: (4)    else:
64: (8)        user_wide = Path.home() / ".config" / "manim" / "manim.cfg"

```

```

63: (4)             folder_wide = Path("manim.cfg")
64: (4)             return [library_wide, user_wide, folder_wide]
65: (0)             def make_config_parser(
66: (4)                 custom_file: StrPath | None = None,
67: (0)             ) -> configparser.ConfigParser:
68: (4)                 """Make a :class:`ConfigParser` object and load any ``.cfg`` files.
69: (4)                 The user-wide file, if it exists, overrides the library-wide file. The
70: (4)                 folder-wide file, if it exists, overrides the other two.
71: (4)                 The folder-wide file can be ignored by passing ``custom_file``. However,
72: (4)                 the user-wide and library-wide config files cannot be ignored.
73: (4)             Parameters
74: (4)             -----
75: (4)             custom_file
76: (8)                 Path to a custom config file. If used, the folder-wide file in the
77: (8)                 relevant directory will be ignored, if it exists. If None, the
78: (8)                 folder-wide file will be used, if it exists.
79: (4)             Returns
80: (4)             -----
81: (4)             :class:`ConfigParser`
82: (8)                 A parser containing the config options found in the .cfg files that
83: (8)                 were found. It is guaranteed to contain at least the config options
84: (8)                 found in the library-wide file.
85: (4)             See Also
86: (4)             -----
87: (4)             :func:`config_file_paths`
88: (4)             """
89: (4)             library_wide, user_wide, folder_wide = config_file_paths()
90: (4)             parser = configparser.ConfigParser()
91: (4)             with library_wide.open() as file:
92: (8)                 parser.read_file(file) # necessary file
93: (4)             other_files = [user_wide, Path(custom_file) if custom_file else
94: (4)                 folder_wide]
95: (4)
96: (0)             return parser
97: (4)             def determine_quality(qual: str) -> str:
98: (8)                 for quality, values in constants.QUALITIES.items():
99: (12)                     if values["flag"] is not None and values["flag"] == qual:
100: (4)                         return quality
101: (0)             return qual
102: (4)             class ManimConfig(MutableMapping):
103: (4)                 """Dict-like class storing all config options.
104: (4)                 The global ``config`` object is an instance of this class, and acts as a
105: (4)                 single source of truth for all of the library's customizable behavior.
106: (4)                 The global ``config`` object is capable of digesting different types of
107: (4)                 sources and converting them into a uniform interface. These sources are
108: (4)                 (in ascending order of precedence): configuration files, command line
109: (4)                 arguments, and programmatic changes. Regardless of how the user chooses
110: (4)                 to
111: (4)                 set a config option, she can access its current value using
112: (4)                 :class:`ManimConfig`'s attributes and properties.
113: (4)             Notes
114: (4)             -----
115: (4)                 Each config option is implemented as a property of this class.
116: (4)                 Each config option can be set via a config file, using the full name of
117: (4)                 the
118: (4)                 property. If a config option has an associated CLI flag, then the flag is
119: (4)                 equal to the full name of the property. Those that admit an alternative
120: (4)                 flag or no flag at all are documented in the individual property's
121: (4)                 docstring.
122: (4)             Examples
123: (4)             -----
124: (4)                 We use a copy of the global configuration object in the following
125: (4)                 examples for the sake of demonstration; you can skip these lines
126: (4)                 and just import ``config`` directly if you actually want to modify
127: (4)                 the configuration:
128: (4)                 .. code-block:: pycon
129: (8)                     >>> from manim import config as global_config
130: (8)                     >>> config = global_config.copy()
131: (4)             Each config option allows for dict syntax and attribute syntax. For

```

```

129: (4)           example, the following two lines are equivalent,
130: (4)           .. code-block:: pycon
131: (8)             >>> from manim import WHITE
132: (8)             >>> config.background_color = WHITE
133: (8)             >>> config["background_color"] = WHITE
134: (4)           The former is preferred; the latter is provided mostly for backwards
135: (4)           compatibility.
136: (4)           The config options are designed to keep internal consistency. For
example,
137: (4)           setting ``frame_y_radius`` will affect ``frame_height``:
138: (4)           .. code-block:: pycon
139: (8)             >>> config.frame_height
140: (8)             8.0
141: (8)             >>> config.frame_y_radius = 5.0
142: (8)             >>> config.frame_height
143: (8)             10.0
144: (4)           There are many ways of interacting with config options. Take for example
145: (4)           the config option ``background_color``. There are three ways to change
it:
146: (4)           via a config file, via CLI flags, or programmatically.
147: (4)           To set the background color via a config file, save the following
148: (4)           ```manim.cfg`` file with the following contents.
149: (4)           .. code-block::
150: (7)             [CLI]
151: (7)             background_color = WHITE
152: (4)           In order to have this ```.cfg`` file apply to a manim scene, it needs to be
153: (4)           placed in the same directory as the script,
154: (4)           .. code-block:: bash
155: (10)             project/
156: (10)               └── scene.py
157: (10)               └── manim.cfg
158: (4)           Now, when the user executes
159: (4)           .. code-block:: bash
160: (8)             manim scene.py
161: (4)           the background of the scene will be set to ```WHITE``. This applies
regardless
162: (4)           of where the manim command is invoked from.
163: (4)           Command line arguments override ```.cfg`` files. In the previous example,
164: (4)           executing
165: (4)           .. code-block:: bash
166: (8)             manim scene.py -c BLUE
167: (4)           will set the background color to BLUE, regardless of the contents of
168: (4)           ```manim.cfg``.
169: (4)           Finally, any programmatic changes made within the scene script itself will
170: (4)           override the command line arguments. For example, if ```scene.py``

contains
171: (4)           the following
172: (4)           .. code-block:: python
173: (8)             from manim import *
174: (8)             config.background_color = RED
175: (8)             class MyScene(Scene): ...
176: (4)           the background color will be set to RED, regardless of the contents of
177: (4)           ```manim.cfg`` or the CLI arguments used when invoking manim.
178: (4)
179: (4)           """
180: (8)             _OPTS = {
181: (8)               "assets_dir",
182: (8)               "background_color",
183: (8)               "background_opacity",
184: (8)               "custom_folders",
185: (8)               "disable_caching",
186: (8)               "disable_caching_warning",
187: (8)               "dry_run",
188: (8)               "enable_wireframe",
189: (8)               "ffmpeg_loglevel",
190: (8)               "ffmpeg_executable",
191: (8)               "format",
192: (8)               "flush_cache",
193: (8)               "frame_height",
194: (8)               "frame_rate",

```

```

194: (8)                 "frame_width",
195: (8)                 "frame_x_radius",
196: (8)                 "frame_y_radius",
197: (8)                 "from_animation_number",
198: (8)                 "images_dir",
199: (8)                 "input_file",
200: (8)                 "media_embed",
201: (8)                 "media_width",
202: (8)                 "log_dir",
203: (8)                 "log_to_file",
204: (8)                 "max_files_cached",
205: (8)                 "media_dir",
206: (8)                 "movie_file_extension",
207: (8)                 "notify_outdated_version",
208: (8)                 "output_file",
209: (8)                 "partial_movie_dir",
210: (8)                 "pixel_height",
211: (8)                 "pixel_width",
212: (8)                 "plugins",
213: (8)                 "preview",
214: (8)                 "progress_bar",
215: (8)                 "quality",
216: (8)                 "save_as_gif",
217: (8)                 "save_sections",
218: (8)                 "save_last_frame",
219: (8)                 "save_pngs",
220: (8)                 "scene_names",
221: (8)                 "show_in_file_browser",
222: (8)                 "tex_dir",
223: (8)                 "tex_template",
224: (8)                 "tex_template_file",
225: (8)                 "text_dir",
226: (8)                 "upto_animation_number",
227: (8)                 "renderer",
228: (8)                 "enable_gui",
229: (8)                 "gui_location",
230: (8)                 "use_projection_fill_shaders",
231: (8)                 "use_projection_stroke_shaders",
232: (8)                 "verbosity",
233: (8)                 "video_dir",
234: (8)                 "sections_dir",
235: (8)                 "fullscreen",
236: (8)                 "window_position",
237: (8)                 "window_size",
238: (8)                 "window_monitor",
239: (8)                 "write_all",
240: (8)                 "write_to_movie",
241: (8)                 "zero_pad",
242: (8)                 "force_window",
243: (8)                 "no_latex_cleanup",
244: (8)                 "preview_command",
245: (4)
246: (4)             }
247: (8)         def __init__(self) -> None:
248: (4)             self._d: dict[str, Any | None] = {k: None for k in self._OPTS}
249: (8)         def __iter__(self) -> Iterator[str]:
250: (4)             return iter(self._d)
251: (8)         def __len__(self) -> int:
252: (4)             return len(self._d)
253: (8)         def __contains__(self, key: object) -> bool:
254: (12)             try:
255: (12)                 self.__getitem__(key)
256: (8)                     return True
257: (12)             except AttributeError:
258: (4)                 return False
259: (8)         def __getitem__(self, key: str) -> Any:
260: (4)             return getattr(self, key)
261: (8)         def __setitem__(self, key: str, val: Any) -> None:
262: (4)             setattr(ManimConfig, key).fset(self, val) # fset is the property's
263: (8)             setter

```

```

262: (4)             def update(self, obj: ManimConfig | dict[str, Any]) -> None: # type:
ignore[override]
263: (8)                 """Digest the options found in another :class:`ManimConfig` or in a
dict.
264: (8)                 Similar to :meth:`dict.update`, replaces the values of this object
with
265: (8)                 those of ``obj``.
266: (8)                 Parameters
267: (8)                 -----
268: (8)                 obj
269: (12)                     The object to copy values from.
270: (8)                 Returns
271: (8)                 -----
272: (8)                 None
273: (8)                 Raises
274: (8)                 -----
275: (8)                 :class:`AttributeError`
276: (12)                     If ``obj`` is a dict but contains keys that do not belong to any
config options.
277: (12)
278: (8)                 See Also
279: (8)                 -----
280: (8)                 :meth:`~ManimConfig.digest_file`, :meth:`~ManimConfig.digest_args`,
281: (8)                 :meth:`~ManimConfig.digest_parser`
282: (8)                 """
283: (8)                 if isinstance(obj, ManimConfig):
284: (12)                     self._d.update(obj._d)
285: (12)                     if obj.tex_template:
286: (16)                         self.tex_template = obj.tex_template
287: (8)                 elif isinstance(obj, dict):
288: (12)                     _dict = {k: v for k, v in obj.items() if k in self._d}
289: (12)                     for k, v in _dict.items():
290: (16)                         self[k] = v
291: (12)                     _dict = {k: v for k, v in obj.items() if k not in self._d}
292: (12)                     for k, v in _dict.items():
293: (16)                         self[k] = v
294: (4)                 def __delitem__(self, key: str) -> NoReturn:
295: (8)                     raise AttributeError("'ManimConfig' object does not support item
deletion")
296: (4)
297: (8)                 def __delattr__(self, key: str) -> NoReturn:
298: (8)                     raise AttributeError("'ManimConfig' object does not support item
deletion")
299: (4)                 def copy(self) -> Self:
300: (8)                     """Deepcopy the contents of this ManimConfig.
301: (8)                     Returns
302: (8)                     -----
303: (12)                     :class:`ManimConfig`
304: (8)                     A copy of this object containing no shared references.
305: (8)                 See Also
306: (8)                 -----
307: (8)                 :func:`tempconfig`
308: (8)                 Notes
309: (8)                 -----
310: (8)                 This is the main mechanism behind :func:`tempconfig`.
311: (8)                 """
312: (4)                 def __copy__(self) -> Self:
313: (8)                     """See ManimConfig.copy()."""
314: (8)                     return copy.deepcopy(self)
315: (4)                 def __deepcopy__(self, memo: dict[str, Any]) -> Self:
316: (8)                     """See ManimConfig.copy()."""
317: (8)                     c = type(self)()
318: (8)                     c._d = copy.deepcopy(self._d, memo)
319: (8)                     return c
320: (4)                 def _set_from_list(self, key: str, val: Any, values: list[Any]) -> None:
321: (8)                     """Set ``key`` to ``val`` if ``val`` is contained in ``values``."""
322: (8)                     if val in values:
323: (12)                         self._d[key] = val
324: (8)                     else:
325: (12)                         raise ValueError(f"attempted to set {key} to {val}; must be in

```

```

{values}")
326: (4)         def _set_from_enum(self, key: str, enum_value: Any, enum_class: EnumMeta)
-> None:
327: (8)             """Set ``key`` to the enum object with value ``enum_value`` in the
given
328: (8)             ``enum_class``.
329: (8)             Tests::
330: (12)                 >>> from enum import Enum
331: (12)                 >>> class Fruit(Enum):
332: (12)                     ...     APPLE = 1
333: (12)                     ...     BANANA = 2
334: (12)                     ...     CANTALOUPE = 3
335: (12)                 >>> test_config = ManimConfig()
336: (12)                 >>> test_config._set_from_enum("fruit", 1, Fruit)
337: (12)                 >>> test_config._d['fruit']
338: (12)                 <Fruit.APPLE: 1>
339: (12)                 >>> test_config._set_from_enum("fruit", Fruit.BANANA, Fruit)
340: (12)                 >>> test_config._d['fruit']
341: (12)                 <Fruit.BANANA: 2>
342: (12)                 >>> test_config._set_from_enum("fruit", 42, Fruit)
343: (12)                 Traceback (most recent call last):
344: (12)                 ...
345: (12)                 ValueError: 42 is not a valid Fruit
346: (8)             """
347: (8)             self._d[key] = enum_class(enum_value)
348: (4)         def _set_boolean(self, key: str, val: Any) -> None:
349: (8)             """Set ``key`` to ``val`` if ``val`` is Boolean."""
350: (8)             if val in [True, False]:
351: (12)                 self._d[key] = val
352: (8)             else:
353: (12)                 raise ValueError(f"{key} must be boolean")
354: (4)         def _set_tuple(self, key: str, val: tuple[Any]) -> None:
355: (8)             if isinstance(val, tuple):
356: (12)                 self._d[key] = val
357: (8)             else:
358: (12)                 raise ValueError(f"{key} must be tuple")
359: (4)         def _set_str(self, key: str, val: Any) -> None:
360: (8)             """Set ``key`` to ``val`` if ``val`` is a string."""
361: (8)             if isinstance(val, str):
362: (12)                 self._d[key] = val
363: (8)             elif not val:
364: (12)                 self._d[key] = ""
365: (8)             else:
366: (12)                 raise ValueError(f"{key} must be str or falsy value")
367: (4)         def _set_between(self, key: str, val: float, lo: float, hi: float) ->
None:
368: (8)             """Set ``key`` to ``val`` if lo <= val <= hi."""
369: (8)             if lo <= val <= hi:
370: (12)                 self._d[key] = val
371: (8)             else:
372: (12)                 raise ValueError(f"{key} must be {lo} <= {key} <= {hi}")
373: (4)         def _set_int_between(self, key: str, val: int, lo: int, hi: int) -> None:
374: (8)             """Set ``key`` to ``val`` if lo <= val <= hi."""
375: (8)             if lo <= val <= hi:
376: (12)                 self._d[key] = val
377: (8)             else:
378: (12)                 raise ValueError(
379: (16)                     f"{key} must be an integer such that {lo} <= {key} <= {hi}",
380: (12)                     )
381: (4)         def _set_pos_number(self, key: str, val: int, allow_inf: bool) -> None:
382: (8)             """Set ``key`` to ``val`` if ``val`` is a positive integer."""
383: (8)             if isinstance(val, int) and val > -1:
384: (12)                 self._d[key] = val
385: (8)             elif allow_inf and val in [-1, float("inf")]:
386: (12)                 self._d[key] = float("inf")
387: (8)             else:
388: (12)                 raise ValueError(
389: (16)                     f"{key} must be a non-negative integer (use -1 for infinity)",
390: (12)                     )

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
391: (4)             def __repr__(self) -> str:
392: (8)               rep = ""
393: (8)               for k, v in sorted(self._d.items(), key=lambda x: x[0]):
394: (12)                 rep += f"{k}: {v}, "
395: (8)               return rep
396: (4)             def digest_parser(self, parser: configparser.ConfigParser) -> Self:
397: (8)               """Process the config options present in a :class:`ConfigParser` object.
398: (8)               This method processes arbitrary parsers, not only those read from a single file, whereas :meth:`~ManimConfig.digest_file` can only process one
399: (8)               file at a time.
400: (8)             Parameters
401: (8)               -----
402: (8)             parser
403: (8)               An object reflecting the contents of one or many ``.cfg`` files.
404: (12)               In
405: (12)                 have
406: (12)               particular, it may reflect the contents of multiple files that
407: (8)               been parsed in a cascading fashion.
408: (8)             Returns
409: (8)               -----
410: (8)             self : :class:`ManimConfig`
411: (12)               This object, after processing the contents of ``parser``.
412: (8)             See Also
413: (8)               -----
414: (8)             :func:`make_config_parser`, :meth:`~ManimConfig.digest_file`,
415: (8)             :meth:`~ManimConfig.digest_args`,
416: (8)             Notes
417: (8)               -----
418: (8)             If there are multiple ``.cfg`` files to process, it is always more efficient to parse them into a single :class:`ConfigParser` object first, and then call this function once (instead of calling :meth:`~ManimConfig.digest_file` multiple times).
419: (8)             Examples
420: (8)               -----
421: (8)             To digest the config options set in two files, first create a ConfigParser and parse both files and then digest the parser:
422: (8)               ..
423: (8)                 code-block:: python
424: (8)                   parser = configparser.ConfigParser()
425: (8)                   parser.read([file1, file2])
426: (12)                   config = ManimConfig().digest_parser(parser)
427: (12)               In fact, the global ``config`` object is initialized like so:
428: (12)               ..
429: (8)                 code-block:: python
430: (8)                   parser = make_config_parser()
431: (12)                   config = ManimConfig().digest_parser(parser)
432: (12)               """
433: (8)               self._parser = parser
434: (8)               for key in [
435: (12)                 "notify_outdated_version",
436: (12)                 "write_to_movie",
437: (12)                 "save_last_frame",
438: (12)                 "write_all",
439: (12)                 "save_pngs",
440: (12)                 "save_as_gif",
441: (12)                 "save_sections",
442: (12)                 "preview",
443: (12)                 "show_in_file_browser",
444: (12)                 "log_to_file",
445: (12)                 "disable_caching",
446: (12)                 "disable_caching_warning",
447: (12)                 "flush_cache",
448: (12)                 "custom_folders",
449: (12)                 "enable_gui",
450: (12)                 "fullscreen",
451: (12)                 "use_projection_fill_shaders",
452: (12)                 "use_projection_stroke_shaders",
453: (12)                 "enable_wireframe",
454: (12)                 "force_window",
455: (12)

```

```

456: (12)           "no_latex_cleanup",
457: (8)            ]:
458: (12)            setattr(self, key, parser["CLI"].getboolean(key, fallback=False))
459: (8)            for key in [
460: (12)              "from_animation_number",
461: (12)              "upto_animation_number",
462: (12)              "max_files_cached",
463: (12)              "pixel_height",
464: (12)              "pixel_width",
465: (12)              "window_monitor",
466: (12)              "zero_pad",
467: (8)            ]:
468: (12)              setattr(self, key, parser["CLI"].getint(key))
469: (8)            for key in [
470: (12)              "assets_dir",
471: (12)              "verbosity",
472: (12)              "media_dir",
473: (12)              "log_dir",
474: (12)              "video_dir",
475: (12)              "sections_dir",
476: (12)              "images_dir",
477: (12)              "text_dir",
478: (12)              "tex_dir",
479: (12)              "partial_movie_dir",
480: (12)              "input_file",
481: (12)              "output_file",
482: (12)              "movie_file_extension",
483: (12)              "background_color",
484: (12)              "renderer",
485: (12)              "window_position",
486: (8)            ]:
487: (12)              setattr(self, key, parser["CLI"].get(key, fallback="", raw=True))
488: (8)            for key in [
489: (12)              "background_opacity",
490: (12)              "frame_rate",
491: (8)            ]:
492: (12)              setattr(self, key, parser["CLI"].getfloat(key))
493: (8)            gui_location = tuple(
494: (12)              map(int, re.split(r";|\-", parser["CLI"]["gui_location"])))
495: (8)
496: (8)            )
497: (8)            setattr(self, "gui_location", gui_location)
498: (12)            window_size = parser["CLI"][
499: (8)              "window_size"
500: (8)            ] # if not "default", get a tuple of the position
501: (12)            if window_size != "default":
502: (8)              window_size = tuple(map(int, re.split(r";|\-", window_size)))
503: (8)            setattr(self, "window_size", window_size)
504: (8)            plugins = parser["CLI"].get("plugins", fallback="", raw=True)
505: (12)            if plugins == "":
506: (8)              plugins = []
507: (12)            else:
508: (8)              plugins = plugins.split(",")
509: (8)            self.plugins = plugins
510: (8)            self["frame_height"] = parser["CLI"].getfloat("frame_height", 8.0)
511: (8)            width = parser["CLI"].getfloat("frame_width", None)
512: (12)            if width is None:
513: (8)              self["frame_width"] = self["frame_height"] * self["aspect_ratio"]
514: (12)            else:
515: (8)              self["frame_width"] = width
516: (8)            val = parser["CLI"].get("tex_template_file")
517: (12)            if val:
518: (8)              self.tex_template_file = val
519: (8)            val = parser["CLI"].get("progress_bar")
520: (12)            if val:
521: (8)              setattr(self, "progress_bar", val)
522: (8)            val = parser["ffmpeg"].get("loglevel")
523: (12)            if val:
524: (8)              self.ffmpeg_loglevel = val

```

```

525: (8)                     setattr(self, "ffmpeg_executable", val)
526: (8)             try:
527: (12)                 val = parser["jupyter"].getboolean("media_embed")
528: (8)             except ValueError:
529: (12)                 val = None
530: (8)             setattr(self, "media_embed", val)
531: (8)             val = parser["jupyter"].get("media_width")
532: (8)             if val:
533: (12)                 setattr(self, "media_width", val)
534: (8)             val = parser["CLI"].get("quality", fallback="", raw=True)
535: (8)             if val:
536: (12)                 self.quality = _determine_quality(val)
537: (8)         return self
538: (4)     def digest_args(self, args: argparse.Namespace) -> Self:
539: (8)         """Process the config options present in CLI arguments.
540: (8)         Parameters
541: (8)         -----
542: (8)         args
543: (12)             An object returned by :func:`.main_utils.parse_args()` .
544: (8)         Returns
545: (8)         -----
546: (8)         self : :class:`ManimConfig`
547: (12)             This object, after processing the contents of ``parser``.
548: (8)         See Also
549: (8)         -----
550: (8)         :func:`.main_utils.parse_args()`, :meth:`~.ManimConfig.digest_parser` ,
551: (8)         :meth:`~.ManimConfig.digest_file` 
552: (8)         Notes
553: (8)         -----
554: (8)         If ``args.config_file`` is a non-empty string, ``ManimConfig`` tries
to digest the
555: (8)         contents of said file with :meth:`~ManimConfig.digest_file` before
556: (8)         digesting any other CLI arguments.
557: (8)         """
558: (8)         if args.file.suffix == ".cfg":
559: (12)             args.config_file = args.file
560: (8)         if str(args.file) == "-":
561: (12)             self.input_file = args.file
562: (8)         if args.config_file:
563: (12)             self.digest_file(args.config_file)
564: (8)         if not self.input_file:
565: (12)             self.input_file = Path(args.file).absolute()
566: (8)         self.scene_names = args.scene_names if args.scene_names is not None
else []
567: (8)         self.output_file = args.output_file
568: (8)         for key in [
569: (12)             "notify_outdated_version",
570: (12)             "preview",
571: (12)             "show_in_file_browser",
572: (12)             "write_to_movie",
573: (12)             "save_last_frame",
574: (12)             "save_pngs",
575: (12)             "save_as_gif",
576: (12)             "save_sections",
577: (12)             "write_all",
578: (12)             "disable_caching",
579: (12)             "format",
580: (12)             "flush_cache",
581: (12)             "progress_bar",
582: (12)             "transparent",
583: (12)             "scene_names",
584: (12)             "verbosity",
585: (12)             "renderer",
586: (12)             "background_color",
587: (12)             "enable_gui",
588: (12)             "fullscreen",
589: (12)             "use_projection_fill_shaders",
590: (12)             "use_projection_stroke_shaders",
591: (12)             "zero_pad",

```

```

592: (12)                                "enable_wireframe",
593: (12)                                "force_window",
594: (12)                                "dry_run",
595: (12)                                "no_latex_cleanup",
596: (12)                                "preview_command",
597: (8) ]:
598: (12)        if hasattr(args, key):
599: (16)            attr = getattr(args, key)
600: (16)            if attr is not None:
601: (20)                self[key] = attr
602: (8) for key in [
603: (12)    "media_dir", # always set this one first
604: (12)    "log_dir",
605: (12)    "log_to_file", # always set this one last
606: (8) ]:
607: (12)        if hasattr(args, key):
608: (16)            attr = getattr(args, key)
609: (16)            if attr is not None:
610: (20)                self[key] = attr
611: (8) if self["save_last_frame"]:
612: (12)    self["write_to_movie"] = False
613: (8) nflag = args.from_animation_number
614: (8) if nflag:
615: (12)    self.from_animation_number = nflag[0]
616: (12)    try:
617: (16)        self.upto_animation_number = nflag[1]
618: (12)    except Exception:
619: (16)        logging.getLogger("manim").info(
620: (20)            f"No end scene number specified in -n option. Rendering
from {nflag[0]} onwards...", )
621: (16)        self.quality = _determine_quality(getattr(args, "quality", None))
622: (8) rflag = args.resolution
623: (8) if rflag:
624: (8)    self.pixel_width = int(rflag[0])
625: (12)    self.pixel_height = int(rflag[1])
626: (12) fps = args.frame_rate
627: (8) if fps:
628: (8)    self.frame_rate = float(fps)
629: (12) if args.custom_folders:
630: (8)    for opt in [
631: (12)        "media_dir",
632: (16)        "video_dir",
633: (16)        "sections_dir",
634: (16)        "images_dir",
635: (16)        "text_dir",
636: (16)        "tex_dir",
637: (16)        "log_dir",
638: (16)        "partial_movie_dir",
639: (16)    ]:
640: (12)        self[opt] = self._parser["custom_folders"].get(opt, raw=True)
641: (16) if hasattr(args, "media_dir") and args.media_dir:
642: (12)    self.media_dir = args.media_dir
643: (16) if args.tex_template:
644: (8)    self.tex_template = TexTemplate.from_file(args.tex_template)
645: (12) if (
646: (8)        self.renderer == RendererType.OPENGL
647: (12)        and getattr(args, "write_to_movie") is None
648: (12)    ):
649: (8)        self["write_to_movie"] = False
650: (12) if getattr(args, "gui_location") is not None:
651: (8)        self.gui_location = args.gui_location
652: (12) return self
653: (8) def digest_file(self, filename: StrPath) -> Self:
654: (4)    """Process the config options present in a ``.cfg`` file.
655: (8)    This method processes a single ``.cfg`` file, whereas
656: (8)    :meth:`~ManimConfig.digest_parser` can process arbitrary parsers,
657: (8)    built
658: (8)    perhaps from multiple ``.cfg`` files.

```

```

659: (8)             Parameters
660: (8)             -----
661: (8)             filename
662: (12)            Path to the ``.cfg`` file.
663: (8)             Returns
664: (8)             -----
665: (8)             self : :class:`ManimConfig`
666: (12)            This object, after processing the contents of ``filename``.
667: (8)             See Also
668: (8)             -----
669: (8)             :meth:`~ManimConfig.digest_file`, :meth:`~ManimConfig.digest_args`,
670: (8)             :func:`make_config_parser`
671: (8)             Notes
672: (8)             -----
673: (8)             If there are multiple ``.cfg`` files to process, it is always more
674: (8)             efficient to parse them into a single :class:`ConfigParser` object
675: (8)             first and digesting them with one call to
676: (8)             :meth:`~ManimConfig.digest_parser`, instead of calling this method
677: (8)             multiple times.
678: (8)             """
679: (8)             if not Path(filename).is_file():
680: (12)               raise FileNotFoundError(
681: (16)                 errno.ENOENT,
682: (16)                 "Error: --config_file could not find a valid config file.",
683: (16)                 str(filename),
684: (12)               )
685: (8)             return self.digest_parser(make_config_parser(filename))
686: (4)             @property
687: (4)             def preview(self) -> bool:
688: (8)               """Whether to play the rendered movie (-p)."""
689: (8)               return self._d["preview"] or self._d["enable_gui"]
690: (4)             @preview.setter
691: (4)             def preview(self, value: bool) -> None:
692: (8)               self._set_boolean("preview", value)
693: (4)             @property
694: (4)             def show_in_file_browser(self) -> bool:
695: (8)               """Whether to show the output file in the file browser (-f)."""
696: (8)               return self._d["show_in_file_browser"]
697: (4)             @show_in_file_browser.setter
698: (4)             def show_in_file_browser(self, value: bool) -> None:
699: (8)               self._set_boolean("show_in_file_browser", value)
700: (4)             @property
701: (4)             def progress_bar(self) -> str:
702: (8)               """Whether to show progress bars while rendering animations."""
703: (8)               return self._d["progress_bar"]
704: (4)             @progress_bar.setter
705: (4)             def progress_bar(self, value: str) -> None:
706: (8)               self._set_from_list("progress_bar", value, ["none", "display",
707: "leave"])
707: (4)             @property
708: (4)             def log_to_file(self) -> bool:
709: (8)               """Whether to save logs to a file."""
710: (8)               return self._d["log_to_file"]
711: (4)             @log_to_file.setter
712: (4)             def log_to_file(self, value: bool) -> None:
713: (8)               self._set_boolean("log_to_file", value)
714: (4)             @property
715: (4)             def notify_outdated_version(self) -> bool:
716: (8)               """Whether to notify if there is a version update available."""
717: (8)               return self._d["notify_outdated_version"]
718: (4)             @notify_outdated_version.setter
719: (4)             def notify_outdated_version(self, value: bool) -> None:
720: (8)               self._set_boolean("notify_outdated_version", value)
721: (4)             @property
722: (4)             def write_to_movie(self) -> bool:
723: (8)               """Whether to render the scene to a movie file (-w)."""
724: (8)               return self._d["write_to_movie"]
725: (4)             @write_to_movie.setter
726: (4)             def write_to_movie(self, value: bool) -> None:

```

```

727: (8)                     self._set_boolean("write_to_movie", value)
728: (4) @property
729: (4) def save_last_frame(self) -> bool:
730: (8)     """Whether to save the last frame of the scene as an image file (-s)."""
731: (8)     return self._d["save_last_frame"]
732: (4) @save_last_frame.setter
733: (4) def save_last_frame(self, value: bool) -> None:
734: (8)     self._set_boolean("save_last_frame", value)
735: (4) @property
736: (4) def write_all(self) -> bool:
737: (8)     """Whether to render all scenes in the input file (-a)."""
738: (8)     return self._d["write_all"]
739: (4) @write_all.setter
740: (4) def write_all(self, value: bool) -> None:
741: (8)     self._set_boolean("write_all", value)
742: (4) @property
743: (4) def save_pngs(self) -> bool:
744: (8)     """Whether to save all frames in the scene as images files (-g)."""
745: (8)     return self._d["save_pngs"]
746: (4) @save_pngs.setter
747: (4) def save_pngs(self, value: bool) -> None:
748: (8)     self._set_boolean("save_pngs", value)
749: (4) @property
750: (4) def save_as_gif(self) -> bool:
751: (8)     """Whether to save the rendered scene in .gif format (-i)."""
752: (8)     return self._d["save_as_gif"]
753: (4) @save_as_gif.setter
754: (4) def save_as_gif(self, value: bool) -> None:
755: (8)     self._set_boolean("save_as_gif", value)
756: (4) @property
757: (4) def save_sections(self) -> bool:
758: (8)     """Whether to save single videos for each section in addition to the
movie file."""
759: (8)     return self._d["save_sections"]
760: (4) @save_sections.setter
761: (4) def save_sections(self, value: bool) -> None:
762: (8)     self._set_boolean("save_sections", value)
763: (4) @property
764: (4) def enable_wireframe(self) -> bool:
765: (8)     """Whether to enable wireframe debugging mode in opengl."""
766: (8)     return self._d["enable_wireframe"]
767: (4) @enable_wireframe.setter
768: (4) def enable_wireframe(self, value: bool) -> None:
769: (8)     self._set_boolean("enable_wireframe", value)
770: (4) @property
771: (4) def force_window(self) -> bool:
772: (8)     """Whether to force window when using the opengl renderer."""
773: (8)     return self._d["force_window"]
774: (4) @force_window.setter
775: (4) def force_window(self, value: bool) -> None:
776: (8)     self._set_boolean("force_window", value)
777: (4) @property
778: (4) def no_latex_cleanup(self) -> bool:
779: (8)     """Prevents deletion of .aux, .dvi, and .log files produced by Tex and
MathTex."""
780: (8)     return self._d["no_latex_cleanup"]
781: (4) @no_latex_cleanup.setter
782: (4) def no_latex_cleanup(self, value: bool) -> None:
783: (8)     self._set_boolean("no_latex_cleanup", value)
784: (4) @property
785: (4) def preview_command(self) -> str:
786: (8)     return self._d["preview_command"]
787: (4) @preview_command.setter
788: (4) def preview_command(self, value: str) -> None:
789: (8)     self._set_str("preview_command", value)
790: (4) @property
791: (4) def verbosity(self) -> str:
792: (8)     """Logger verbosity; "DEBUG", "INFO", "WARNING", "ERROR", or

```

```

"CRITICAL" (-v)."""
793: (8)             return self._d["verbosity"]
794: (4) @verbosity.setter
795: (4) def verbosity(self, val: str) -> None:
796: (8)     self._set_from_list(
797: (12)         "verbosity",
798: (12)         val,
799: (12)         ["DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"],
800: (8)     )
801: (8)     logging.getLogger("manim").setLevel(val)
802: (4) @property
803: (4) def format(self) -> str:
804: (8)     """File format; "png", "gif", "mp4", "webm" or "mov"."""
805: (8)     return self._d["format"]
806: (4) @format.setter
807: (4) def format(self, val: str) -> None:
808: (8)     self._set_from_list(
809: (12)         "format",
810: (12)         val,
811: (12)         [None, "png", "gif", "mp4", "mov", "webm"],
812: (8)     )
813: (8)     if self.format == "webm":
814: (12)         logging.getLogger("manim").warning(
815: (16)             "Output format set as webm, this can be slower than other
formats",
816: (12)         )
817: (4) @property
818: (4) def ffmpeg_loglevel(self) -> str:
819: (8)     """Verbosity level of ffmpeg (no flag)."""
820: (8)     return self._d["ffmpeg_loglevel"]
821: (4) @ffmpeg_loglevel.setter
822: (4) def ffmpeg_loglevel(self, val: str) -> None:
823: (8)     self._set_from_list(
824: (12)         "ffmpeg_loglevel",
825: (12)         val,
826: (12)         ["DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"],
827: (8)     )
828: (4) @property
829: (4) def ffmpeg_executable(self) -> str:
830: (8)     """Custom path to the ffmpeg executable."""
831: (8)     return self._d["ffmpeg_executable"]
832: (4) @ffmpeg_executable.setter
833: (4) def ffmpeg_executable(self, value: str) -> None:
834: (8)     self._set_str("ffmpeg_executable", value)
835: (4) @property
836: (4) def media_embed(self) -> bool:
837: (8)     """Whether to embed videos in Jupyter notebook."""
838: (8)     return self._d["media_embed"]
839: (4) @media_embed.setter
840: (4) def media_embed(self, value: bool) -> None:
841: (8)     self._set_boolean("media_embed", value)
842: (4) @property
843: (4) def media_width(self) -> str:
844: (8)     """Media width in Jupyter notebook."""
845: (8)     return self._d["media_width"]
846: (4) @media_width.setter
847: (4) def media_width(self, value: str) -> None:
848: (8)     self._set_str("media_width", value)
849: (4) @property
850: (4) def pixel_width(self) -> int:
851: (8)     """Frame width in pixels (--resolution, -r)."""
852: (8)     return self._d["pixel_width"]
853: (4) @pixel_width.setter
854: (4) def pixel_width(self, value: int) -> None:
855: (8)     self._set_pos_number("pixel_width", value, False)
856: (4) @property
857: (4) def pixel_height(self) -> int:
858: (8)     """Frame height in pixels (--resolution, -r)."""
859: (8)     return self._d["pixel_height"]

```

```

860: (4) @pixel_height.setter
861: (4)     def pixel_height(self, value: int) -> None:
862: (8)         self._set_pos_number("pixel_height", value, False)
863: (4) @property
864: (4)     def aspect_ratio(self) -> int:
865: (8)         """Aspect ratio (width / height) in pixels (--resolution, -r)."""
866: (8)         return self._d["pixel_width"] / self._d["pixel_height"]
867: (4) @property
868: (4)     def frame_height(self) -> float:
869: (8)         """Frame height in logical units (no flag)."""
870: (8)         return self._d["frame_height"]
871: (4) @frame_height.setter
872: (4)     def frame_height(self, value: float) -> None:
873: (8)         self._d.__setitem__("frame_height", value)
874: (4) @property
875: (4)     def frame_width(self) -> float:
876: (8)         """Frame width in logical units (no flag)."""
877: (8)         return self._d["frame_width"]
878: (4) @frame_width.setter
879: (4)     def frame_width(self, value: float) -> None:
880: (8)         self._d.__setitem__("frame_width", value)
881: (4) @property
882: (4)     def frame_y_radius(self) -> float:
883: (8)         """Half the frame height (no flag)."""
884: (8)         return self._d["frame_height"] / 2
885: (4) @frame_y_radius.setter
886: (4)     def frame_y_radius(self, value: float) -> None:
887: (8)         self._d.__setitem__("frame_y_radius", value) or self._d.__setitem__(
888: (12)             "frame_height", 2 * value
889: (8)         )
890: (4) @property
891: (4)     def frame_x_radius(self) -> float:
892: (8)         """Half the frame width (no flag)."""
893: (8)         return self._d["frame_width"] / 2
894: (4) @frame_x_radius.setter
895: (4)     def frame_x_radius(self, value: float) -> None:
896: (8)         self._d.__setitem__("frame_x_radius", value) or self._d.__setitem__(
897: (12)             "frame_width", 2 * value
898: (8)         )
899: (4) @property
900: (4)     def top(self) -> Vector3D:
901: (8)         """Coordinate at the center top of the frame."""
902: (8)         return self.frame_y_radius * constants.UP
903: (4) @property
904: (4)     def bottom(self) -> Vector3D:
905: (8)         """Coordinate at the center bottom of the frame."""
906: (8)         return self.frame_y_radius * constants.DOWN
907: (4) @property
908: (4)     def left_side(self) -> Vector3D:
909: (8)         """Coordinate at the middle left of the frame."""
910: (8)         return self.frame_x_radius * constants.LEFT
911: (4) @property
912: (4)     def right_side(self) -> Vector3D:
913: (8)         """Coordinate at the middle right of the frame."""
914: (8)         return self.frame_x_radius * constants.RIGHT
915: (4) @property
916: (4)     def frame_rate(self) -> float:
917: (8)         """Frame rate in frames per second."""
918: (8)         return self._d["frame_rate"]
919: (4) @frame_rate.setter
920: (4)     def frame_rate(self, value: float) -> None:
921: (8)         self._d.__setitem__("frame_rate", value)
922: (4) @property
923: (4)     def background_color(self) -> ManimColor:
924: (8)         """Background color of the scene (-c)."""
925: (8)         return self._d["background_color"]
926: (4) @background_color.setter
927: (4)     def background_color(self, value: Any) -> None:
928: (8)         self._d.__setitem__("background_color", ManimColor(value))

```

```

929: (4) @property
930: (4)     def from_animation_number(self) -> int:
931: (8)         """Start rendering animations at this number (-n)."""
932: (8)         return self._d["from_animation_number"]
933: (4)     @from_animation_number.setter
934: (4)     def from_animation_number(self, value: int) -> None:
935: (8)         self._d.__setitem__("from_animation_number", value)
936: (4)     @property
937: (4)     def upto_animation_number(self) -> int:
938: (8)         """Stop rendering animations at this number. Use -1 to avoid skipping
(-n)."""
939: (8)     return self._d["upto_animation_number"]
940: (4)     @upto_animation_number.setter
941: (4)     def upto_animation_number(self, value: int) -> None:
942: (8)         self._set_pos_number("upto_animation_number", value, True)
943: (4)     @property
944: (4)     def max_files_cached(self) -> int:
945: (8)         """Maximum number of files cached. Use -1 for infinity (no flag)."""
946: (8)         return self._d["max_files_cached"]
947: (4)     @max_files_cached.setter
948: (4)     def max_files_cached(self, value: int) -> None:
949: (8)         self._set_pos_number("max_files_cached", value, True)
950: (4)     @property
951: (4)     def window_monitor(self) -> int:
952: (8)         """The monitor on which the scene will be rendered."""
953: (8)         return self._d["window_monitor"]
954: (4)     @window_monitor.setter
955: (4)     def window_monitor(self, value: int) -> None:
956: (8)         self._set_pos_number("window_monitor", value, True)
957: (4)     @property
958: (4)     def flush_cache(self) -> bool:
959: (8)         """Whether to delete all the cached partial movie files."""
960: (8)         return self._d["flush_cache"]
961: (4)     @flush_cache.setter
962: (4)     def flush_cache(self, value: bool) -> None:
963: (8)         self._set_boolean("flush_cache", value)
964: (4)     @property
965: (4)     def disable_caching(self) -> bool:
966: (8)         """Whether to use scene caching."""
967: (8)         return self._d["disable_caching"]
968: (4)     @disable_caching.setter
969: (4)     def disable_caching(self, value: bool) -> None:
970: (8)         self._set_boolean("disable_caching", value)
971: (4)     @property
972: (4)     def disable_caching_warning(self) -> bool:
973: (8)         """Whether a warning is raised if there are too much submobjects to
hash."""
974: (8)         return self._d["disable_caching_warning"]
975: (4)     @disable_caching_warning.setter
976: (4)     def disable_caching_warning(self, value: bool) -> None:
977: (8)         self._set_boolean("disable_caching_warning", value)
978: (4)     @property
979: (4)     def movie_file_extension(self) -> str:
980: (8)         """Either .mp4, .webm or .mov."""
981: (8)         return self._d["movie_file_extension"]
982: (4)     @movie_file_extension.setter
983: (4)     def movie_file_extension(self, value: str) -> None:
984: (8)         self._set_from_list("movie_file_extension", value, [".mp4", ".mov",
".webm"])
985: (4)     @property
986: (4)     def background_opacity(self) -> float:
987: (8)         """A number between 0.0 (fully transparent) and 1.0 (fully opaque)."""
988: (8)         return self._d["background_opacity"]
989: (4)     @background_opacity.setter
990: (4)     def background_opacity(self, value: float) -> None:
991: (8)         self._set_between("background_opacity", value, 0, 1)
992: (4)     @property
993: (4)     def frame_size(self) -> tuple[int, int]:
994: (8)         """Tuple with (pixel width, pixel height) (no flag)."""

```

```

995: (8)             return (self._d["pixel_width"], self._d["pixel_height"])
996: (4)             @frame_size.setter
997: (4)             def frame_size(self, value: tuple[int, int]) -> None:
998: (8)                 self._d.__setitem__("pixel_width", value[0]) or self._d.__setitem__(
999: (12)                   "pixel_height", value[1]
1000: (8)
1001: (4)             )
1002: (4)             @property
1003: (8)             def quality(self) -> str | None:
1004: (8)                 """Video quality (-q)."""
1005: (8)                 keys = ["pixel_width", "pixel_height", "frame_rate"]
1006: (8)                 q = {k: self[k] for k in keys}
1007: (12)                 for qual in constants.QUALITIES:
1008: (16)                     if all(q[k] == constants.QUALITIES[qual][k] for k in keys):
1009: (8)                         return qual
1010: (4)
1011: (4)             return None
1012: (8)             @quality.setter
1013: (8)             def quality(self, value: str | None) -> None:
1014: (8)                 if value is None:
1015: (12)                     return
1016: (8)                 if value not in constants.QUALITIES:
1017: (8)                     raise KeyError(f"quality must be one of
1018: ({list(constants.QUALITIES.keys())})")
1019: (4)                 q = constants.QUALITIES[value]
1020: (8)                 self.frame_size = q["pixel_width"], q["pixel_height"]
1021: (8)                 self.frame_rate = q["frame_rate"]
1022: (4)             @property
1023: (4)             def transparent(self) -> bool:
1024: (8)                 """Whether the background opacity is 0.0 (-t)."""
1025: (8)                 return self._d["background_opacity"] == 0.0
1026: (4)             @transparent.setter
1027: (4)             def transparent(self, value: bool) -> None:
1028: (8)                 self._d["background_opacity"] = float(not value)
1029: (8)                 self.resolve_movie_file_extension(value)
1030: (4)             @property
1031: (4)             def dry_run(self) -> bool:
1032: (8)                 """Whether dry run is enabled."""
1033: (8)                 return self._d["dry_run"]
1034: (8)             @dry_run.setter
1035: (12)             def dry_run(self, val: bool) -> None:
1036: (12)                 self._d["dry_run"] = val
1037: (12)                 if val:
1038: (12)                     self.write_to_movie = False
1039: (4)                     self.write_all = False
1040: (12)                     self.save_last_frame = False
1041: (8)                     self.format = None
1042: (8)             @property
1043: (4)             def renderer(self) -> RendererType:
1044: (12)                 """The currently active renderer.
1045: (12)                 Populated with one of the available renderers in
1046: (12)                 :class:`.RendererType` .
1047: (12)             Tests::
1048: (12)                 >>> test_config = ManimConfig()
1049: (12)                 >>> test_config.renderer is None # a new ManimConfig is
1050: (12)                 unpopulated
1051: (12)                 True
1052: (12)                 >>> test_config.renderer = 'opengl'
1053: (12)                 >>> test_config.renderer
1054: (8)                 <RendererType.OPENGL: 'opengl'>
1055: (12)                 >>> test_config.renderer = 42
1056: (12)                 Traceback (most recent call last):
1057: (12)                 ...
1058: (12)                 ValueError: 42 is not a valid RendererType
1059: (4)                 Check that capitalization of renderer types is irrelevant::
1060: (4)                 >>> test_config.renderer = 'OpenGL'
1061: (4)                 >>> test_config.renderer = 'cAir0'
1062: (4)                 """
1063: (8)                 return self._d["renderer"]
1064: (4)             @renderer.setter
1065: (4)             def renderer(self, value: str | RendererType) -> None:

```

```

1061: (8)             """The setter of the renderer property.
1062: (8)             Takes care of switching inheritance bases using the
1063: (8)             :class:`.ConvertToOpenGL` metaclass.
1064: (8)
1065: (8)             if isinstance(value, str):
1066: (12)                 value = value.lower()
1067: (8)             renderer = RendererType(value)
1068: (8)             try:
1069: (12)                 from manim.mobject.opengl.opengl_compatibility import
1070: (12)                     from manim.mobject.opengl.opengl_mobject import OpenGLMobject
1071: (12)                     from manim.mobject.opengl.vectorized_mobject import
1072: (12)                         from ..mobject.mobject import Mobject
1073: (12)                         from ..mobject.types.vectorized_mobject import VMobject
1074: (12)                         for cls in ConvertToOpenGL._converted_classes:
1075: (16)                             if renderer == RendererType.OPENGL:
1076: (20)                                 conversion_dict = {
1077: (24)                                     Mobject: OpenGLMobject,
1078: (24)                                     VMobject: OpenGLMobject,
1079: (20)                             }
1080: (16)                         else:
1081: (20)                             conversion_dict = {
1082: (24)                                 OpenGLMobject: Mobject,
1083: (24)                                 OpenGLMobject: VMobject,
1084: (20)                             }
1085: (16)                         cls.__bases__ = tuple(
1086: (20)                             conversion_dict.get(base, base) for base in cls.__bases__
1087: (16)                         )
1088: (8)             except ImportError:
1089: (12)                 pass
1090: (8)             self._set_from_enum("renderer", renderer, RendererType)
1091: (4)             @property
1092: (4)             def media_dir(self) -> str:
1093: (8)                 """Main output directory. See :meth:`ManimConfig.get_dir`."""
1094: (8)                 return self._d["media_dir"]
1095: (4)             @media_dir.setter
1096: (4)             def media_dir(self, value: str | Path) -> None:
1097: (8)                 self._set_dir("media_dir", value)
1098: (4)             @property
1099: (4)             def window_position(self) -> str:
1100: (8)                 """Set the position of preview window. You can use directions, e.g.
UL/DR/ORIGIN/LEFT...or the position(pixel) of the upper left corner of the window, e.g.
'960,540'."""
1101: (8)                 return self._d["window_position"]
1102: (4)             @window_position.setter
1103: (4)             def window_position(self, value: str) -> None:
1104: (8)                 self._d.__setitem__("window_position", value)
1105: (4)             @property
1106: (4)             def window_size(self) -> str:
1107: (8)                 """The size of the opengl window. 'default' to automatically scale the
window based on the display monitor."""
1108: (8)                 return self._d["window_size"]
1109: (4)             @window_size.setter
1110: (4)             def window_size(self, value: str) -> None:
1111: (8)                 self._d.__setitem__("window_size", value)
1112: (4)             def resolve_movie_file_extension(self, is_transparent: bool) -> None:
1113: (8)                 if is_transparent:
1114: (12)                     self.movie_file_extension = ".webm" if self.format == "webm" else
".mov"
1115: (8)                     elif self.format == "webm":
1116: (12)                         self.movie_file_extension = ".webm"
1117: (8)
1118: (12)                         elif self.format == "mov":
1119: (8)                             self.movie_file_extension = ".mov"
1120: (12)                         else:
1121: (4)                             self.movie_file_extension = ".mp4"
1122: (4)             @property
1123: (8)             def enable_gui(self) -> bool:
1124: (8)                 """Enable GUI interaction."""

```

```

1124: (8)                                return self._d["enable_gui"]
1125: (4) @enable_gui.setter
1126: (4) def enable_gui(self, value: bool) -> None:
1127: (8)     self._set_boolean("enable_gui", value)
1128: (4) @property
1129: (4) def gui_location(self) -> tuple[Any]:
1130: (8)     """Enable GUI interaction."""
1131: (8)     return self._d["gui_location"]
1132: (4) @gui_location.setter
1133: (4) def gui_location(self, value: tuple[Any]) -> None:
1134: (8)     self._set_tuple("gui_location", value)
1135: (4) @property
1136: (4) def fullscreen(self) -> bool:
1137: (8)     """Expand the window to its maximum possible size."""
1138: (8)     return self._d["fullscreen"]
1139: (4) @fullscreen.setter
1140: (4) def fullscreen(self, value: bool) -> None:
1141: (8)     self._set_boolean("fullscreen", value)
1142: (4) @property
1143: (4) def use_projection_fill_shaders(self) -> bool:
1144: (8)     """Use shaders for OpenGLMobject fill which are compatible with
transformation matrices."""
1145: (8)     return self._d["use_projection_fill_shaders"]
1146: (4) @use_projection_fill_shaders.setter
1147: (4) def use_projection_fill_shaders(self, value: bool) -> None:
1148: (8)     self._set_boolean("use_projection_fill_shaders", value)
1149: (4) @property
1150: (4) def use_projection_stroke_shaders(self) -> bool:
1151: (8)     """Use shaders for OpenGLMobject stroke which are compatible with
transformation matrices."""
1152: (8)     return self._d["use_projection_stroke_shaders"]
1153: (4) @use_projection_stroke_shaders.setter
1154: (4) def use_projection_stroke_shaders(self, value: bool) -> None:
1155: (8)     self._set_boolean("use_projection_stroke_shaders", value)
1156: (4) @property
1157: (4) def zero_pad(self) -> int:
1158: (8)     """PNG zero padding. A number between 0 (no zero padding) and 9 (9
columns minimum)."""
1159: (8)     return self._d["zero_pad"]
1160: (4) @zero_pad.setter
1161: (4) def zero_pad(self, value: int) -> None:
1162: (8)     self._set_int_between("zero_pad", value, 0, 9)
1163: (4) def get_dir(self, key: str, **kwargs: Any) -> Path:
1164: (8)     """Resolve a config option that stores a directory.
Config options that store directories may depend on one another. This
method is used to provide the actual directory to the end user.
Parameters
-----
key
    The config option to be resolved. Must be an option ending in
    ``'_dir'``, for example ``'media_dir'`` or ``'video_dir'``.
kwargs
    Any strings to be used when resolving the directory.
Returns
-----
:class:`pathlib.Path`
    Path to the requested directory. If the path resolves to the
empty
    string, return ``None`` instead.
Raises
-----
:class:`KeyError`
    When ``key`` is not a config option that stores a directory and
    thus :meth:`~ManimConfig.get_dir` is not appropriate; or when
    ``key`` is appropriate but there is not enough information to
    resolve the directory.
Notes
-----
Standard :meth:`str.format` syntax is used to resolve the paths so the

```

```

1189: (8)           paths may contain arbitrary placeholders using f-string notation.
1190: (8)           However, these will require ``kwargs`` to contain the required values.
1191: (8)           Examples
1192: (8)
1193: (8)           -----
1194: (8)           The value of ``config.tex_dir`` is ``'{media_dir}/Tex``` by default,
In          i.e. it is a subfolder of wherever ``config.media_dir`` is located.

1195: (8)           order to get the *actual* directory, use :meth:`~ManimConfig.get_dir`.
1196: (8)           .. code-block:: pycon
1197: (12)             >>> from manim import config as globalconfig
1198: (12)             >>> config = globalconfig.copy()
1199: (12)             >>> config.tex_dir
1200: (12)             '{media_dir}/Tex'
1201: (12)             >>> config.media_dir
1202: (12)             './media'
1203: (12)             >>> config.get_dir("tex_dir").as_posix()
1204: (12)             'media/Tex'

1205: (8)           Resolving directories is done in a lazy way, at the last possible
1206: (8)           moment, to reflect any changes in other config options:
1207: (8)           .. code-block:: pycon
1208: (12)             >>> config.media_dir = "my_media_dir"
1209: (12)             >>> config.get_dir("tex_dir").as_posix()
1210: (12)             'my_media_dir/Tex'

1211: (8)           Some directories depend on information that is not available to
1212: (8)           :class:`ManimConfig`. For example, the default value of `video_dir`
1213: (8)           includes the name of the input file and the video quality
1214: (8)           (e.g. 480p15). This informamtion has to be supplied via ``kwargs``:
1215: (8)           .. code-block:: pycon
1216: (12)             >>> config.video_dir
1217: (12)             '{media_dir}/videos/{module_name}/{quality}'
1218: (12)             >>> config.get_dir("video_dir")
1219: (12)             Traceback (most recent call last):
1220: (12)             KeyError: 'video_dir {media_dir}/videos/{module_name}/{quality}'

requires the following keyword arguments: module_name'
1221: (12)             >>> config.get_dir("video_dir", module_name="myfile").as_posix()
1222: (12)             'my_media_dir/videos/myfile/1080p60'

Note the quality does not need to be passed as keyword argument since
1223: (8)           :class:`ManimConfig` does store information about quality.
1224: (8)
1225: (8)           Directories may be recursively defined. For example, the config
option
1226: (8)           ``partial_movie_dir`` depends on ``video_dir``, which in turn depends
1227: (8)           on ``media_dir``:
1228: (8)           .. code-block:: pycon
1229: (12)             >>> config.partial_movie_dir
1230: (12)             '{video_dir}/partial_movie_files/{scene_name}'
1231: (12)             >>> config.get_dir("partial_movie_dir")
1232: (12)             Traceback (most recent call last):
1233: (12)             KeyError: 'partial_movie_dir'

{video_dir}/partial_movie_files/{scene_name} requires the following keyword arguments: scene_name'
1234: (12)             >>> config.get_dir(
1235: (12)               ...     "partial_movie_dir", module_name="myfile",
scene_name="myscene"
1236: (12)               ... ).as_posix()
1237: (12)               'my_media_dir/videos/myfile/1080p60/partial_movie_files/myscene'

Standard f-string syntax is used. Arbitrary names can be used when
1238: (8)
1239: (8)           defining directories, as long as the corresponding values are passed
to
1240: (8)           :meth:`ManimConfig.get_dir` via ``kwargs``.
1241: (8)           .. code-block:: pycon
1242: (12)             >>> config.media_dir = "{dir1}/{dir2}"
1243: (12)             >>> config.get_dir("media_dir")
1244: (12)             Traceback (most recent call last):
1245: (12)             KeyError: 'media_dir {dir1}/{dir2} requires the following keyword
arguments: dir1'

1246: (12)             >>> config.get_dir("media_dir", dir1="foo", dir2="bar").as_posix()
1247: (12)             'foo/bar'
1248: (12)             >>> config.media_dir = "./media"
1249: (12)             >>> config.get_dir("media_dir").as_posix()
1250: (12)             'media'

```

```

1251: (8)             """
1252: (8)         dirs = [
1253: (12)             "assets_dir",
1254: (12)             "media_dir",
1255: (12)             "video_dir",
1256: (12)             "sections_dir",
1257: (12)             "images_dir",
1258: (12)             "text_dir",
1259: (12)             "tex_dir",
1260: (12)             "log_dir",
1261: (12)             "input_file",
1262: (12)             "output_file",
1263: (12)             "partial_movie_dir",
1264: (8)         ]
1265: (8)     if key not in dirs:
1266: (12)         raise KeyError(
1267: (16)             "must pass one of "
1268: (16)             "{media,video,images,text,tex,log}_dir "
1269: (16)             "or {input,output}_file",
1270: (12)         )
1271: (8)     dirs.remove(key) # a path cannot contain itself
1272: (8)     all_args = {k: self._d[k] for k in dirs}
1273: (8)     all_args.update(kwargs)
1274: (8)     all_args["quality"] = f"{self.pixel_height}p{self.frame_rate:g}"
1275: (8)     path = self._d[key]
1276: (8)     while "{" in path:
1277: (12)         try:
1278: (16)             path = path.format(**all_args)
1279: (12)         except KeyError as exc:
1280: (16)             raise KeyError(
1281: (20)                 f"{key} {self._d[key]} requires the following "
1282: (20)                 "+ keyword arguments: "
1283: (20)                 "+ ".join(exc.args),
1284: (16)             ) from exc
1285: (8)     return Path(path) if path else None
1286: (4) def _set_dir(self, key: str, val: str | Path) -> None:
1287: (8)     if isinstance(val, Path):
1288: (12)         self._d.__setitem__(key, str(val))
1289: (8)     else:
1290: (12)         self._d.__setitem__(key, val)
1291: (4) @property
1292: (4) def assets_dir(self) -> str:
1293: (8)     """Directory to locate video assets (no flag)."""
1294: (8)     return self._d["assets_dir"]
1295: (4) @assets_dir.setter
1296: (4) def assets_dir(self, value: str | Path) -> None:
1297: (8)     self._set_dir("assets_dir", value)
1298: (4) @property
1299: (4) def log_dir(self) -> str:
1300: (8)     """Directory to place logs. See :meth:`ManimConfig.get_dir`."""
1301: (8)     return self._d["log_dir"]
1302: (4) @log_dir.setter
1303: (4) def log_dir(self, value: str | Path) -> None:
1304: (8)     self._set_dir("log_dir", value)
1305: (4) @property
1306: (4) def video_dir(self) -> str:
1307: (8)     """Directory to place videos (no flag). See
:meth:`ManimConfig.get_dir`."""
1308: (8)     return self._d["video_dir"]
1309: (4) @video_dir.setter
1310: (4) def video_dir(self, value: str | Path) -> None:
1311: (8)     self._set_dir("video_dir", value)
1312: (4) @property
1313: (4) def sections_dir(self) -> str:
1314: (8)     """Directory to place section videos (no flag). See
:meth:`ManimConfig.get_dir`."""
1315: (8)     return self._d["sections_dir"]
1316: (4) @sections_dir.setter
1317: (4) def sections_dir(self, value: str | Path) -> None:

```

```

1318: (8)             self._set_dir("sections_dir", value)
1319: (4)             @property
1320: (4)             def images_dir(self) -> str:
1321: (8)                 """Directory to place images (no flag). See
:meth:`ManimConfig.get_dir`."""
1322: (8)             return self._d["images_dir"]
1323: (4)             @images_dir.setter
1324: (4)             def images_dir(self, value: str | Path) -> None:
1325: (8)                 self._set_dir("images_dir", value)
1326: (4)             @property
1327: (4)             def text_dir(self) -> str:
1328: (8)                 """Directory to place text (no flag). See
:meth:`ManimConfig.get_dir`."""
1329: (8)                 return self._d["text_dir"]
1330: (4)                 @text_dir.setter
1331: (4)                 def text_dir(self, value: str | Path) -> None:
1332: (8)                     self._set_dir("text_dir", value)
1333: (4)                     @property
1334: (4)                     def tex_dir(self) -> str:
1335: (8)                         """Directory to place tex (no flag). See
:meth:`ManimConfig.get_dir`."""
1336: (8)                         return self._d["tex_dir"]
1337: (4)                         @tex_dir.setter
1338: (4)                         def tex_dir(self, value: str | Path) -> None:
1339: (8)                             self._set_dir("tex_dir", value)
1340: (4)                         @property
1341: (4)                         def partial_movie_dir(self) -> str:
1342: (8)                             """Directory to place partial movie files (no flag). See
:meth:`ManimConfig.get_dir`."""
1343: (8)                             return self._d["partial_movie_dir"]
1344: (4)                             @partial_movie_dir.setter
1345: (4)                             def partial_movie_dir(self, value: str | Path) -> None:
1346: (8)                                 self._set_dir("partial_movie_dir", value)
1347: (4)                                 @property
1348: (4)                                 def custom_folders(self) -> str:
1349: (8)                                     """Whether to use custom folder output."""
1350: (8)                                     return self._d["custom_folders"]
1351: (4)                                     @custom_folders.setter
1352: (4)                                     def custom_folders(self, value: str | Path) -> None:
1353: (8)                                         self._set_dir("custom_folders", value)
1354: (4)                                         @property
1355: (4)                                         def input_file(self) -> str:
1356: (8)                                             """Input file name."""
1357: (8)                                             return self._d["input_file"]
1358: (4)                                             @input_file.setter
1359: (4)                                             def input_file(self, value: str | Path) -> None:
1360: (8)                                                 self._set_dir("input_file", value)
1361: (4)                                                 @property
1362: (4)                                                 def output_file(self) -> str:
1363: (8)                                                     """Output file name (-o)."""
1364: (8)                                                     return self._d["output_file"]
1365: (4)                                                     @output_file.setter
1366: (4)                                                     def output_file(self, value: str | Path) -> None:
1367: (8)                                                         self._set_dir("output_file", value)
1368: (4)                                                         @property
1369: (4)                                                         def scene_names(self) -> list[str]:
1370: (8)                                                             """Scenes to play from file."""
1371: (8)                                                             return self._d["scene_names"]
1372: (4)                                                             @scene_names.setter
1373: (4)                                                             def scene_names(self, value: list[str]) -> None:
1374: (8)                                                                 self._d.__setitem__("scene_names", value)
1375: (4)                                                               @property
1376: (4)                                                               def tex_template(self) -> TexTemplate:
1377: (8)                                                               """Template used when rendering Tex. See :class:`.TexTemplate`."""
1378: (8)                                                               if not hasattr(self, "_tex_template") or not self._tex_template:
1379: (12)                                                               fn = self._d["tex_template_file"]
1380: (12)                                                               if fn:
1381: (16)                                                               self._tex_template = TexTemplate.from_file(fn)
1382: (12)                                                               else:

```

```

1383: (16)                     self._tex_template = TexTemplate()
1384: (8)                     return self._tex_template
1385: (4) @tex_template.setter
1386: (4)     def tex_template(self, val: TexTemplate) -> None:
1387: (8)         if isinstance(val, TexTemplate):
1388: (12)             self._tex_template = val
1389: (4)     @property
1390: (4)     def tex_template_file(self) -> Path:
1391: (8)         """File to read Tex template from (no flag). See
:class:`.TexTemplate`."""
1392: (8)         return self._d["tex_template_file"]
1393: (4) @tex_template_file.setter
1394: (4)     def tex_template_file(self, val: str) -> None:
1395: (8)         if val:
1396: (12)             if not os.access(val, os.R_OK):
1397: (16)                 logging.getLogger("manim").warning(
1398: (20)                     f"Custom TeX template {val} not found or not readable.",
1399: (16)
1400: (12)
1401: (16)             self._d["tex_template_file"] = Path(val)
1402: (8)         else:
1403: (12)             self._d["tex_template_file"] = val # actually set the falsy value
1404: (4) @property
1405: (4)     def plugins(self) -> list[str]:
1406: (8)         """List of plugins to enable."""
1407: (8)         return self._d["plugins"]
1408: (4) @plugins.setter
1409: (4)     def plugins(self, value: list[str]):
1410: (8)         self._d["plugins"] = value
1411: (0) class ManimFrame(Mapping):
1412: (4)     _OPTS: ClassVar[set[str]] = {
1413: (8)         "pixel_width",
1414: (8)         "pixel_height",
1415: (8)         "aspect_ratio",
1416: (8)         "frame_height",
1417: (8)         "frame_width",
1418: (8)         "frame_y_radius",
1419: (8)         "frame_x_radius",
1420: (8)         "top",
1421: (8)         "bottom",
1422: (8)         "left_side",
1423: (8)         "right_side",
1424: (4)
1425: (4)     _CONSTANTS: ClassVar[dict[str, Vector3D]] = {
1426: (8)         "UP": np.array((0.0, 1.0, 0.0)),
1427: (8)         "DOWN": np.array((0.0, -1.0, 0.0)),
1428: (8)         "RIGHT": np.array((1.0, 0.0, 0.0)),
1429: (8)         "LEFT": np.array((-1.0, 0.0, 0.0)),
1430: (8)         "IN": np.array((0.0, 0.0, -1.0)),
1431: (8)         "OUT": np.array((0.0, 0.0, 1.0)),
1432: (8)         "ORIGIN": np.array((0.0, 0.0, 0.0)),
1433: (8)         "X_AXIS": np.array((1.0, 0.0, 0.0)),
1434: (8)         "Y_AXIS": np.array((0.0, 1.0, 0.0)),
1435: (8)         "Z_AXIS": np.array((0.0, 0.0, 1.0)),
1436: (8)         "UL": np.array((-1.0, 1.0, 0.0)),
1437: (8)         "UR": np.array((1.0, 1.0, 0.0)),
1438: (8)         "DL": np.array((-1.0, -1.0, 0.0)),
1439: (8)         "DR": np.array((1.0, -1.0, 0.0)),
1440: (4)
1441: (4)     _c: ManimConfig
1442: (4)     def __init__(self, c: ManimConfig) -> None:
1443: (8)         if not isinstance(c, ManimConfig):
1444: (12)             raise TypeError("argument must be instance of 'ManimConfig'")
1445: (8)         self._dict__["_c"] = c
1446: (4)     def __getitem__(self, key: str | int) -> Any:
1447: (8)         if key in self._OPTS:
1448: (12)             return self._c[key]
1449: (8)         elif key in self._CONSTANTS:
1450: (12)             return self._CONSTANTS[key]

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

1451: (8)             else:
1452: (12)             raise KeyError(key)
1453: (4)             def __iter__(self) -> Iterable[str]:
1454: (8)             return iter(list(self._OPTS) + list(self._CONSTANTS))
1455: (4)             def __len__(self) -> int:
1456: (8)             return len(self._OPTS)
1457: (4)             def __setattr__(self, attr: Any, val: Any) -> NoReturn:
1458: (8)                 raise TypeError("'ManimFrame' object does not support item
assignment")
1459: (4)             def __setitem__(self, key: Any, val: Any) -> NoReturn:
1460: (8)                 raise TypeError("'ManimFrame' object does not support item
assignment")
1461: (4)             def __delitem__(self, key: Any) -> NoReturn:
1462: (8)                 raise TypeError("'ManimFrame' object does not support item deletion")
1463: (0)             for opt in list(ManimFrame._OPTS) + list(ManimFrame._CONSTANTS):
1464: (4)                 setattr(ManimFrame, opt, property(lambda self, o=opt: self[o]))

```

---

File 2 - fading.py:

```

1: (0)             """Fading in and out of view.
2: (0)             .. manim:: Fading
3: (4)             class Fading(Scene):
4: (8)                 def construct(self):
5: (12)                     tex_in = Tex("Fade", "In").scale(3)
6: (12)                     tex_out = Tex("Fade", "Out").scale(3)
7: (12)                     self.play(FadeIn(tex_in, shift=DOWN, scale=0.66))
8: (12)                     self.play(ReplacementTransform(tex_in, tex_out))
9: (12)                     self.play(FadeOut(tex_out, shift=DOWN * 2, scale=1.5))
10: (0)                 """
11: (0)                 from __future__ import annotations
12: (0)                 __all__ = [
13: (4)                     "FadeOut",
14: (4)                     "FadeIn",
15: (0)                 ]
16: (0)                 import numpy as np
17: (0)                 from manim.mobject.opengl.opengl_mobject import OpenGLMobject
18: (0)                 from ..animation.transform import Transform
19: (0)                 from ..constants import ORIGIN
20: (0)                 from ..mobject.mobject import Group, Mobject
21: (0)                 from ..scene.scene import Scene
22: (0)                 class _Fade(Transform):
23: (4)                     """Fade :class:`~.Mobject`'s in or out.
24: (4)                     Parameters
25: (4)                     -----
26: (4)                     mobjects
27: (8)                         The mobjects to be faded.
28: (4)                     shift
29: (8)                         The vector by which the mobject shifts while being faded.
30: (4)                     target_position
31: (8)                         The position to/from which the mobject moves while being faded in. In
case
32: (8)                         another mobject is given as target position, its center is used.
33: (4)                     scale
34: (8)                         The factor by which the mobject is scaled initially before being
rescaling to
35: (8)                         its original size while being faded in.
36: (4)                     """
37: (4)                     def __init__(
38: (8)                         self,
39: (8)                         *mobjects: Mobject,
40: (8)                         shift: np.ndarray | None = None,
41: (8)                         target_position: np.ndarray | Mobject | None = None,
42: (8)                         scale: float = 1,
43: (8)                         **kwargs,
44: (4)                     ) -> None:
45: (8)                         if not mobjects:
46: (12)                             raise ValueError("At least one mobject must be passed.")

```

```

47: (8)                     if len(mobjects) == 1:
48: (12)                   mobject = mobjects[0]
49: (8)                   else:
50: (12)                     mobject = Group(*mobjects)
51: (8)                   self.point_target = False
52: (8)                   if shift is None:
53: (12)                     if target_position is not None:
54: (16)                       if isinstance(target_position, (Mobject, OpenGLMobject)):
55: (20)                         target_position = target_position.get_center()
56: (16)                         shift = target_position - mobject.get_center()
57: (16)                         self.point_target = True
58: (12)                     else:
59: (16)                       shift = ORIGIN
60: (8)                     self.shift_vector = shift
61: (8)                     self.scale_factor = scale
62: (8)                     super().__init__(mobject, **kwargs)
63: (4) def _create_faded_mobject(self, fadeIn: bool) -> Mobject:
64: (8)     """Create a faded, shifted and scaled copy of the mobject.
65: (8)     Parameters
66: (8)     -----
67: (8)     fadeIn
68: (12)         Whether the faded mobject is used to fade in.
69: (8)     Returns
70: (8)     -----
71: (8)     Mobject
72: (12)         The faded, shifted and scaled copy of the mobject.
73: (8)     """
74: (8)     faded_mobject = self.mobject.copy()
75: (8)     faded_mobject.fade(1)
76: (8)     direction_modifier = -1 if fadeIn and not self.point_target else 1
77: (8)     faded_mobject.shift(self.shift_vector * direction_modifier)
78: (8)     faded_mobject.scale(self.scale_factor)
79: (8)     return faded_mobject
80: (0) class FadeIn(_Fade):
81: (4)     """Fade in :class:`~.Mobject`'s.
82: (4)     Parameters
83: (4)     -----
84: (4)     mobjects
85: (8)         The mobjects to be faded in.
86: (4)     shift
87: (8)         The vector by which the mobject shifts while being faded in.
88: (4)     target_position
89: (8)         The position from which the mobject starts while being faded in. In
90: (8)             another mobject is given as target position, its center is used.
91: (4)     scale
92: (8)         The factor by which the mobject is scaled initially before being
93: (8)             its original size while being faded in.
94: (4)     Examples
95: (4)     -----
96: (4)     .. manim :: FadeInExample
97: (8)         class FadeInExample(Scene):
98: (12)             def construct(self):
99: (16)                 dot = Dot(UP * 2 + LEFT)
100: (16)                 self.add(dot)
101: (16)                 tex = Tex(
102: (20)                     "FadeIn with ", "shift ", " or target\\_position", " and
scale"
103: (16)                     ).scale(1)
104: (16)                     animations = [
105: (20)                         FadeIn(tex[0]),
106: (20)                         FadeIn(tex[1], shift=DOWN),
107: (20)                         FadeIn(tex[2], target_position=dot),
108: (20)                         FadeIn(tex[3], scale=1.5),
109: (16)                     ]
110: (16)                     self.play(AnimationGroup(*animations, lag_ratio=0.5))
111: (4)             """
112: (4)             def __init__(self, *mobjects: Mobject, **kwargs) -> None:

```

```

113: (8)             super().__init__(*mobjects, introducer=True, **kwargs)
114: (4)             def create_target(self):
115: (8)                 return self.mobject
116: (4)             def create_starting_mobject(self):
117: (8)                 return self._create_faded_mobject(fadeIn=True)
118: (0)             class FadeOut(_Fade):
119: (4)                 """Fade out :class:`~.Mobject` s.
120: (4)                 Parameters
121: (4)                 -----
122: (4)                 mobjects
123: (8)                     The mobjects to be faded out.
124: (4)                 shift
125: (8)                     The vector by which the mobject shifts while being faded out.
126: (4)                 target_position
127: (8)                     The position to which the mobject moves while being faded out. In case
another
128: (8)                         mobject is given as target position, its center is used.
129: (4)                 scale
130: (8)                     The factor by which the mobject is scaled while being faded out.
131: (4)             Examples
132: (4)             -----
133: (4)             .. manim :: FadeInExample
134: (8)             class FadeInExample(Scene):
135: (12)                 def construct(self):
136: (16)                     dot = Dot(UP * 2 + LEFT)
137: (16)                     self.add(dot)
138: (16)                     tex = Tex(
139: (20)                         "FadeOut with ", "shift ", " or target\\_position", " and
scale"
140: (16)                     ).scale(1)
141: (16)                     animations = [
142: (20)                         FadeOut(tex[0]),
143: (20)                         FadeOut(tex[1], shift=DOWN),
144: (20)                         FadeOut(tex[2], target_position=dot),
145: (20)                         FadeOut(tex[3], scale=0.5),
146: (16)                     ]
147: (16)                     self.play(AnimationGroup(*animations, lag_ratio=0.5))
148: (4)             """
149: (4)             def __init__(self, *mobjects: Mobject, **kwargs) -> None:
150: (8)                 super().__init__(*mobjects, remover=True, **kwargs)
151: (4)             def create_target(self):
152: (8)                 return self._create_faded_mobject(fadeIn=False)
153: (4)             def clean_up_from_scene(self, scene: Scene = None) -> None:
154: (8)                 super().clean_up_from_scene(scene)
155: (8)                 self.interpolate(0)
-----
```

**File 3 - growing.py:**

```

1: (0)             """Animations that introduce mobjects to scene by growing them from points.
2: (0)             .. manim:: Growing
3: (4)             class Growing(Scene):
4: (8)                 def construct(self):
5: (12)                     square = Square()
6: (12)                     circle = Circle()
7: (12)                     triangle = Triangle()
8: (12)                     arrow = Arrow(LEFT, RIGHT)
9: (12)                     star = Star()
10: (12)                     VGroup(square, circle,
triangle).set_x(0).arrange(buff=1.5).set_y(2)
11: (12)                     VGroup(arrow,
star).move_to(DOWN).set_x(0).arrange(buff=1.5).set_y(-2)
12: (12)                     self.play(GrowFromPoint(square, ORIGIN))
13: (12)                     self.play(GrowFromCenter(circle))
14: (12)                     self.play(GrowFromEdge(triangle, DOWN))
15: (12)                     self.play(GrowArrow(arrow))
16: (12)                     self.play(SpinInFromNothing(star))
17: (0)             """
-----
```

```

18: (0)         from __future__ import annotations
19: (0)         __all__ = [
20: (4)             "GrowFromPoint",
21: (4)             "GrowFromCenter",
22: (4)             "GrowFromEdge",
23: (4)             "GrowArrow",
24: (4)             "SpinInFromNothing",
25: (0)
26: (0)         import typing
27: (0)         import numpy as np
28: (0)         from ..animation.transform import Transform
29: (0)         from ..constants import PI
30: (0)         from ..utils.paths import spiral_path
31: (0)         if typing.TYPE_CHECKING:
32: (4)             from manim.mobject.geometry.line import Arrow
33: (4)             from ..mobject.mobject import Mobject
34: (0)         class GrowFromPoint(Transform):
35: (4)             """Introduce an :class:`~.Mobject` by growing it from a point.
36: (4)             Parameters
37: (4)             -----
38: (4)             mobject
39: (8)                 The mobjects to be introduced.
40: (4)             point
41: (8)                 The point from which the mobject grows.
42: (4)             point_color
43: (8)                 Initial color of the mobject before growing to its full size. Leave
empty to match mobject's color.
44: (4)             Examples
45: (4)             -----
46: (4)             .. manim :: GrowFromPointExample
47: (8)                 class GrowFromPointExample(Scene):
48: (12)                     def construct(self):
49: (16)                         dot = Dot(3 * UR, color=GREEN)
50: (16)                         squares = [Square() for _ in range(4)]
51: (16)                         VGroup(*squares).set_x(0).arrange(buff=1)
52: (16)                         self.add(dot)
53: (16)                         self.play(GrowFromPoint(squares[0], ORIGIN))
54: (16)                         self.play(GrowFromPoint(squares[1], [-2, 2, 0]))
55: (16)                         self.play(GrowFromPoint(squares[2], [3, -2, 0], RED))
56: (16)                         self.play(GrowFromPoint(squares[3], dot, dot.get_color()))
57: (4)
58: (4)             def __init__(
59: (8)                 self, mobject: Mobject, point: np.ndarray, point_color: str = None,
**kwargs
60: (4)             ) -> None:
61: (8)                 self.point = point
62: (8)                 self.point_color = point_color
63: (8)                 super().__init__(mobject, introducer=True, **kwargs)
64: (4)             def create_target(self) -> Mobject:
65: (8)                 return self.mobject
66: (4)             def create_starting_mobject(self) -> Mobject:
67: (8)                 start = super().create_starting_mobject()
68: (8)                 start.scale(0)
69: (8)                 start.move_to(self.point)
70: (8)                 if self.point_color:
71: (12)                     start.set_color(self.point_color)
72: (8)                 return start
73: (0)         class GrowFromCenter(GrowFromPoint):
74: (4)             """Introduce an :class:`~.Mobject` by growing it from its center.
75: (4)             Parameters
76: (4)             -----
77: (4)             mobject
78: (8)                 The mobjects to be introduced.
79: (4)             point_color
80: (8)                 Initial color of the mobject before growing to its full size. Leave
empty to match mobject's color.
81: (4)             Examples
82: (4)             -----
83: (4)             .. manim :: GrowFromCenterExample

```

```

84: (8)             class GrowFromCenterExample(Scene):
85: (12)             def construct(self):
86: (16)                 squares = [Square() for _ in range(2)]
87: (16)                 VGroup(*squares).set_x(0).arrange(buff=2)
88: (16)                 self.play(GrowFromCenter(squares[0]))
89: (16)                 self.play(GrowFromCenter(squares[1], point_color=RED))
90: (4)             """
91: (4)             def __init__(self, mobject: Mobject, point_color: str = None, **kwargs) ->
None:
92: (8)                 point = mobject.get_center()
93: (8)                 super().__init__(mobject, point, point_color=point_color, **kwargs)
94: (0)             class GrowFromEdge(GrowFromPoint):
95: (4)                 """Introduce an :class:`~.Mobject` by growing it from one of its bounding
box edges.
96: (4)             Parameters
97: (4)             -----
98: (4)             mobject
99: (8)                 The mobjects to be introduced.
100: (4)             edge
101: (8)                 The direction to seek bounding box edge of mobject.
102: (4)             point_color
103: (8)                 Initial color of the mobject before growing to its full size. Leave
empty to match mobject's color.
104: (4)             Examples
105: (4)             -----
106: (4)             .. manim :: GrowFromEdgeExample
107: (8)             class GrowFromEdgeExample(Scene):
108: (12)             def construct(self):
109: (16)                 squares = [Square() for _ in range(4)]
110: (16)                 VGroup(*squares).set_x(0).arrange(buff=1)
111: (16)                 self.play(GrowFromEdge(squares[0], DOWN))
112: (16)                 self.play(GrowFromEdge(squares[1], RIGHT))
113: (16)                 self.play(GrowFromEdge(squares[2], UR))
114: (16)                 self.play(GrowFromEdge(squares[3], UP, point_color=RED))
115: (4)             """
116: (4)             def __init__(
117: (8)                 self, mobject: Mobject, edge: np.ndarray, point_color: str = None,
**kwargs
118: (4)             ) -> None:
119: (8)                 point = mobject.get_critical_point(edge)
120: (8)                 super().__init__(mobject, point, point_color=point_color, **kwargs)
121: (0)             class GrowArrow(GrowFromPoint):
122: (4)                 """Introduce an :class:`~.Arrow` by growing it from its start toward its
tip.
123: (4)             Parameters
124: (4)             -----
125: (4)             arrow
126: (8)                 The arrow to be introduced.
127: (4)             point_color
128: (8)                 Initial color of the arrow before growing to its full size. Leave
empty to match arrow's color.
129: (4)             Examples
130: (4)             -----
131: (4)             .. manim :: GrowArrowExample
132: (8)             class GrowArrowExample(Scene):
133: (12)             def construct(self):
134: (16)                 arrows = [Arrow(2 * LEFT, 2 * RIGHT), Arrow(2 * DR, 2 * UL)]
135: (16)                 VGroup(*arrows).set_x(0).arrange(buff=2)
136: (16)                 self.play(GrowArrow(arrows[0]))
137: (16)                 self.play(GrowArrow(arrows[1], point_color=RED))
138: (4)             """
139: (4)             def __init__(self, arrow: Arrow, point_color: str = None, **kwargs) ->
None:
140: (8)                 point = arrow.get_start()
141: (8)                 super().__init__(arrow, point, point_color=point_color, **kwargs)
142: (4)             def create_starting_mobject(self) -> Mobject:
143: (8)                 start_arrow = self.mobject.copy()
144: (8)                 start_arrow.scale(0, scale_tips=True, about_point=self.point)
145: (8)                 if self.point_color:

```

```

146: (12)                      start_arrow.set_color(self.point_color)
147: (8)                       return start_arrow
148: (0)  class SpinInFromNothing(GrowFromCenter):
149: (4)    """Introduce an :class:`~.Mobject` spinning and growing it from its
center.
150: (4)    Parameters
151: (4)    -----
152: (4)    mobject
153: (8)      The mobjects to be introduced.
154: (4)    angle
155: (8)      The amount of spinning before mobject reaches its full size. E.g. 2*PI
means
156: (8)      that the object will do one full spin before being fully introduced.
157: (4)    point_color
158: (8)      Initial color of the mobject before growing to its full size. Leave
empty to match mobject's color.
159: (4)  Examples
160: (4)  -----
161: (4)  .. manim :: SpinInFromNothingExample
162: (8)  class SpinInFromNothingExample(Scene):
163: (12)    def construct(self):
164: (16)      squares = [Square() for _ in range(3)]
165: (16)      VGroup(*squares).set_x(0).arrange(buff=2)
166: (16)      self.play(SpinInFromNothing(squares[0]))
167: (16)      self.play(SpinInFromNothing(squares[1], angle=2 * PI))
168: (16)      self.play(SpinInFromNothing(squares[2], point_color=RED))
169: (4)    """
170: (4)    def __init__(
171: (8)        self, mobject: Mobject, angle: float = PI / 2, point_color: str =
None, **kwargs
172: (4)    ) -> None:
173: (8)        self.angle = angle
174: (8)        super().__init__(
175: (12)            mobject, path_func=spiral_path(angle), point_color=point_color,
**kwargs
176: (8)        )

```

-----

File 4 - `__init__.py`:

```

1: (0)          from __future__ import annotations
2: (0)          from importlib.metadata import version
3: (0)          __version__ = version(__name__)
4: (0)          from .config import *
5: (0)          from .utils.commands import *
6: (0)          import numpy as np
7: (0)          from .animation.animation import *
8: (0)          from .animation.changing import *
9: (0)          from .animation.composition import *
10: (0)         from .animation.creation import *
11: (0)         from .animation.fading import *
12: (0)         from .animation.growing import *
13: (0)         from .animation.indication import *
14: (0)         from .animation.movement import *
15: (0)         from .animation.numbers import *
16: (0)         from .animation.rotation import *
17: (0)         from .animation.specialized import *
18: (0)         from .animation.speedmodifier import *
19: (0)         from .animation.transform import *
20: (0)         from .animation.transform_matching_parts import *
21: (0)         from .animation.updaters.mobject_update_utils import *
22: (0)         from .animation.updaters.update import *
23: (0)         from .camera.camera import *
24: (0)         from .camera.mapping_camera import *
25: (0)         from .camera.moving_camera import *
26: (0)         from .camera.multi_camera import *
27: (0)         from .camera.three_d_camera import *
28: (0)         from .constants import *

```

```

29: (0)         from .mobject.frame import *
30: (0)         from .mobject.geometry.arc import *
31: (0)         from .mobject.geometry.boolean_ops import *
32: (0)         from .mobject.geometry.labeled import *
33: (0)         from .mobject.geometry.line import *
34: (0)         from .mobject.geometry.polygram import *
35: (0)         from .mobject.geometry.shape_matchers import *
36: (0)         from .mobject.geometry.tips import *
37: (0)         from .mobject.graph import *
38: (0)         from .mobject.graphing.coordinate_systems import *
39: (0)         from .mobject.graphing.functions import *
40: (0)         from .mobject.graphing.number_line import *
41: (0)         from .mobject.graphing.probability import *
42: (0)         from .mobject.graphing.scale import *
43: (0)         from .mobject.logo import *
44: (0)         from .mobject.matrix import *
45: (0)         from .mobject.mobject import *
46: (0)         from .mobject.opengl.dot_cloud import *
47: (0)         from .mobject.opengl.opengl_point_cloud_mobject import *
48: (0)         from .mobject.svg.brace import *
49: (0)         from .mobject.svg.svg_mobject import *
50: (0)         from .mobject.table import *
51: (0)         from .mobject.text.code_mobject import *
52: (0)         from .mobject.text.numbers import *
53: (0)         from .mobject.text.tex_mobject import *
54: (0)         from .mobject.text.text_mobject import *
55: (0)         from .mobject.three_d.polyhedra import *
56: (0)         from .mobject.three_d.three_d_utils import *
57: (0)         from .mobject.three_d.three_dimensions import *
58: (0)         from .mobject.types.image_mobject import *
59: (0)         from .mobject.types.point_cloud_mobject import *
60: (0)         from .mobject.types.vectorized_mobject import *
61: (0)         from .mobject.value_tracker import *
62: (0)         from .mobject.vector_field import *
63: (0)         from .renderer.cairo_renderer import *
64: (0)         from .scene.moving_camera_scene import *
65: (0)         from .scene.scene import *
66: (0)         from .scene.scene_file_writer import *
67: (0)         from .scene.section import *
68: (0)         from .scene.three_d_scene import *
69: (0)         from .scene.vector_space_scene import *
70: (0)         from .scene.zoomed_scene import *
71: (0)         from .utils import color, rate_functions, unit
72: (0)         from .utils.bezier import *
73: (0)         from .utils.color import *
74: (0)         from .utils.config_ops import *
75: (0)         from .utils.debug import *
76: (0)         from .utils.file_ops import *
77: (0)         from .utils.images import *
78: (0)         from .utils.iterables import *
79: (0)         from .utils.paths import *
80: (0)         from .utils.rate_functions import *
81: (0)         from .utils.simple_functions import *
82: (0)         from .utils.sounds import *
83: (0)         from .utils.space_ops import *
84: (0)         from .utils.tex import *
85: (0)         from .utils.tex_templates import *
86: (0)     try:
87: (4)         from IPython import get_ipython
88: (4)         from .utils.ipython_magic import ManimMagic
89: (0)     except ImportError:
90: (4)         pass
91: (0)     else:
92: (4)         ipy = get_ipython()
93: (4)         if ipy is not None:
94: (8)             ipy.register_magics(ManimMagic)
95: (0)         from .plugins import *

```

-----

## File 5 - \_\_main\_\_.py:

```

1: (0)          from __future__ import annotations
2: (0)          import click
3: (0)          import cloup
4: (0)          from . import __version__, cli_ctx_settings, console
5: (0)          from .cli.cfg.group import cfg
6: (0)          from .cli.checkhealth.commands import checkhealth
7: (0)          from .cli.default_group import DefaultGroup
8: (0)          from .cli.init.commands import init
9: (0)          from .cli.plugins.commands import plugins
10: (0)         from .cli.render.commands import render
11: (0)         from .constants import EPILOG
12: (0)         def show_splash(ctx, param, value):
13: (4)             if value:
14: (8)                 console.print(f"Manim Community [green]v{__version__}[/green]\n")
15: (0)         def print_version_and_exit(ctx, param, value):
16: (4)             show_splash(ctx, param, value)
17: (4)             if value:
18: (8)                 ctx.exit()
19: (0)         @cloup.group(
20: (4)             context_settings=cli_ctx_settings,
21: (4)             cls=DefaultGroup,
22: (4)             default="render",
23: (4)             no_args_is_help=True,
24: (4)             help="Animation engine for explanatory math videos.",
25: (4)             epilog="See 'manim <command>' to read about a specific subcommand.\n\n"
26: (4)             "Note: the subcommand 'manim render' is called if no other subcommand "
27: (4)             "is specified. Run 'manim render --help' if you would like to know what
the "
28: (4)             f"--ql' or '-p' flags do, for example.\n\n{EPILOG}",
29: (0)         )
30: (0)         @cloup.option(
31: (4)             "--version",
32: (4)             is_flag=True,
33: (4)             help="Show version and exit.",
34: (4)             callback=print_version_and_exit,
35: (4)             is_eager=True,
36: (4)             expose_value=False,
37: (0)         )
38: (0)         @click.option(
39: (4)             "--show-splash/--hide-splash",
40: (4)             is_flag=True,
41: (4)             default=True,
42: (4)             help="Print splash message with version information.",
43: (4)             callback=show_splash,
44: (4)             is_eager=True,
45: (4)             expose_value=False,
46: (0)         )
47: (0)         @cloup.pass_context
48: (0)         def main(ctx):
49: (4)             """The entry point for manim."""
50: (4)             pass
51: (0)             main.add_command(checkhealth)
52: (0)             main.add_command(cfg)
53: (0)             main.add_command(plugins)
54: (0)             main.add_command(init)
55: (0)             main.add_command(render)
56: (0)             if __name__ == "__main__":
57: (4)                 main()

```

## -----

## File 6 - changing.py:

```

1: (0)         """Animation of a mobject boundary and tracing of points."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = ["AnimatedBoundary", "TracedPath"]

```

```

4: (0)         from typing import Callable
5: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
6: (0)         from manim.mobject.types.vectorized_mobject import VGroup, VMobject
7: (0)         from manim.utils.color import (
8: (4)             BLUE_B,
9: (4)             BLUE_D,
10: (4)             BLUE_E,
11: (4)             GREY_BROWN,
12: (4)             WHITE,
13: (4)             ParsableManimColor,
14: (0)
15: (0)     )
16: (0)     from manim.utils.rate_functions import smooth
17: (4)     class AnimatedBoundary(VGroup):
18: (4)         """Boundary of a :class:`.VMobject` with animated color change.
19: (4)         Examples
20: (4)         -----
21: (8)         .. manim:: AnimatedBoundaryExample
22: (12)             class AnimatedBoundaryExample(Scene):
23: (16)                 def construct(self):
24: (16)                     text = Text("So shiny!")
25: (44)                     boundary = AnimatedBoundary(text, colors=[RED, GREEN, BLUE],
26: (16)                                     cycle_rate=3)
27: (16)                     self.add(text, boundary)
28: (16)                     self.wait(2)
28: (4)
29: (4)         def __init__(
30: (8)             self,
31: (8)             vmobject,
32: (8)             colors=[BLUE_D, BLUE_B, BLUE_E, GREY_BROWN],
33: (8)             max_stroke_width=3,
34: (8)             cycle_rate=0.5,
35: (8)             back_and_forth=True,
36: (8)             draw_rate_func=smooth,
37: (8)             fade_rate_func=smooth,
38: (8)             **kwargs,
39: (4)         ):
40: (8)             super().__init__(**kwargs)
41: (8)             self.colors = colors
42: (8)             self.max_stroke_width = max_stroke_width
43: (8)             self.cycle_rate = cycle_rate
44: (8)             self.back_and_forth = back_and_forth
45: (8)             self.draw_rate_func = draw_rate_func
46: (8)             self.fade_rate_func = fade_rate_func
47: (8)             self.vmobject = vmobject
48: (8)             self.boundary_copies = [
49: (12)                 vmobject.copy().set_style(stroke_width=0, fill_opacity=0) for x in
range(2)
50: (8)             ]
51: (8)             self.add(*self.boundary_copies)
52: (8)             self.total_time = 0
53: (8)             self.add_updater(lambda m, dt: self.update_boundary_copies(dt))
54: (4)         def update_boundary_copies(self, dt):
55: (8)             time = self.total_time * self.cycle_rate
56: (8)             growing, fading = self.boundary_copies
57: (8)             colors = self.colors
58: (8)             msw = self.max_stroke_width
59: (8)             vmobject = self.vmobject
60: (8)             index = int(time % len(colors))
61: (8)             alpha = time % 1
62: (8)             draw_alpha = self.draw_rate_func(alpha)
63: (8)             fade_alpha = self.fade_rate_func(alpha)
64: (8)             if self.back_and_forth and int(time) % 2 == 1:
65: (12)                 bounds = (1 - draw_alpha, 1)
66: (8)             else:
67: (12)                 bounds = (0, draw_alpha)
68: (8)             self.full_family_become_partial(growing, vmobject, *bounds)
69: (8)             growing.set_stroke(colors[index], width=msw)
70: (8)             if time >= 1:
71: (12)                 self.full_family_become_partial(fading, vmobject, 0, 1)

```

```

72: (12)                               fading.set_stroke(color=colors[index - 1], width=(1 - fade_alpha))
* msw)
73: (8)                               self.total_time += dt
74: (4)                               def full_family_become_partial(self, mob1, mob2, a, b):
75: (8)                                   family1 = mob1.family_members_with_points()
76: (8)                                   family2 = mob2.family_members_with_points()
77: (8)                                   for sm1, sm2 in zip(family1, family2):
78: (12)                                       sm1.pointwise_become_partial(sm2, a, b)
79: (8)                                   return self
80: (0)                               class TracedPath(VMobject, metaclass=ConvertToOpenGL):
81: (4)                                   """Traces the path of a point returned by a function call.
82: (4)                                   Parameters
83: (4)                                   -----
84: (4)                                   traced_point_func
85: (8)                                       The function to be traced.
86: (4)                                   stroke_width
87: (8)                                       The width of the trace.
88: (4)                                   stroke_color
89: (8)                                       The color of the trace.
90: (4)                                   dissipating_time
91: (8)                                       The time taken for the path to dissipate. Default set to ``None`` which disables dissipation.
92: (8)                                   Examples
93: (4)                                   -----
94: (4)                                   .. manim:: TracedPathExample
95: (4)                                       class TracedPathExample(Scene):
96: (8)                                           def construct(self):
97: (12)                                               circ = Circle(color=RED).shift(4*LEFT)
98: (16)                                               dot = Dot(color=RED).move_to(circ.get_start())
99: (16)                                               rolling_circle = VGroup(circ, dot)
100: (16)                                              trace = TracedPath(circ.get_start())
101: (16)                                              rolling_circle.add_updater(lambda m: m.rotate(-0.3))
102: (16)                                              self.add(trace, rolling_circle)
103: (16)                                              self.play(rolling_circle.animate.shift(8*RIGHT), run_time=4,
104: (16)                                              rate_func=linear)
stroke_opacity=[0, 1])
105: (4)                               .. manim:: DissipatingPathExample
106: (8)                                       class DissipatingPathExample(Scene):
107: (12)                                         def construct(self):
108: (16)                                             a = Dot(RIGHT * 2)
109: (16)                                             b = TracedPath(a.get_center, dissipating_time=0.5,
self.add(a, b)
self.play(a.animate(path_arc=PI / 4).shift(LEFT * 2))
self.play(a.animate(path_arc=-PI / 4).shift(LEFT * 2))
self.wait())
110: (16)
111: (16)
112: (16)
113: (16)
114: (4)
115: (4)                               """
116: (8)                               def __init__(
117: (8)                                   self,
118: (8)                                   traced_point_func: Callable,
119: (8)                                   stroke_width: float = 2,
120: (8)                                   stroke_color: ParsableManimColor | None = WHITE,
121: (8)                                   dissipating_time: float | None = None,
**kwargs,
122: (4)):
123: (8)                                   super().__init__(stroke_color=stroke_color, stroke_width=stroke_width,
**kwargs)
124: (8)                                   self.traced_point_func = traced_point_func
125: (8)                                   self.dissipating_time = dissipating_time
126: (8)                                   self.time = 1 if self.dissipating_time else None
127: (8)                                   self.add_updater(self.update_path)
128: (4)                               def update_path(self, mob, dt):
129: (8)                                   new_point = self.traced_point_func()
130: (8)                                   if not self.has_points():
131: (12)                                       self.start_new_path(new_point)
132: (8)                                       self.add_line_to(new_point)
133: (8)                                   if self.dissipating_time:
134: (12)                                       self.time += dt
135: (12)                                       if self.time - 1 > self.dissipating_time:
136: (16)                                           nppcc = self.n_points_per_curve

```

137: (16)

self.set\_points(self.points[nppcc:])

-----  
File 7 - creation.py:

```

1: (0)          r"""Animate the display or removal of a mobject from a scene.
2: (0)          .. manim:: CreationModule
3: (4)          :hide_source:
4: (4)          from manim import ManimBanner
5: (4)          class CreationModule(Scene):
6: (8)          def construct(self):
7: (12)          s1 = Square()
8: (12)          s2 = Square()
9: (12)          s3 = Square()
10: (12)         s4 = Square()
11: (12)         VGroup(s1, s2, s3, s4).set_x(0).arrange(buff=1.9).shift(UP)
12: (12)         s5 = Square()
13: (12)         s6 = Square()
14: (12)         s7 = Square()
15: (12)         VGroup(s5, s6, s7).set_x(0).arrange(buff=2.6).shift(2 * DOWN)
16: (12)         t1 = Text("Write", font_size=24).next_to(s1, UP)
17: (12)         t2 = Text("AddTextLetterByLetter", font_size=24).next_to(s2, UP)
18: (12)         t3 = Text("Create", font_size=24).next_to(s3, UP)
19: (12)         t4 = Text("Uncreate", font_size=24).next_to(s4, UP)
20: (12)         t5 = Text("DrawBorderThenFill", font_size=24).next_to(s5, UP)
21: (12)         t6 = Text("ShowIncreasingSubsets", font_size=22).next_to(s6, UP)
22: (12)         t7 = Text("ShowSubmobjectsOneByOne", font_size=22).next_to(s7, UP)
23: (12)         self.add(s1, s2, s3, s4, s5, s6, s7, t1, t2, t3, t4, t5, t6, t7)
24: (12)         texts = [Text("manim", font_size=29), Text("manim", font_size=29)]
25: (12)         texts[0].move_to(s1.get_center())
26: (12)         texts[1].move_to(s2.get_center())
27: (12)         self.add(*texts)
28: (12)         objs = [ManimBanner().scale(0.25) for _ in range(5)]
29: (12)         objs[0].move_to(s3.get_center())
30: (12)         objs[1].move_to(s4.get_center())
31: (12)         objs[2].move_to(s5.get_center())
32: (12)         objs[3].move_to(s6.get_center())
33: (12)         objs[4].move_to(s7.get_center())
34: (12)         self.add(*objs)
35: (12)         self.play(
36: (16)             Write(texts[0]),
37: (16)             AddTextLetterByLetter(texts[1]),
38: (16)             Create(objs[0]),
39: (16)             Uncreate(objs[1]),
40: (16)             DrawBorderThenFill(objs[2]),
41: (16)             ShowIncreasingSubsets(objs[3]),
42: (16)             ShowSubmobjectsOneByOne(objs[4]),
43: (16)             run_time=3,
44: (12)         )
45: (12)         self.wait()
46: (0)         """
47: (0)         from __future__ import annotations
48: (0)         __all__ = [
49: (4)             "Create",
50: (4)             "Uncreate",
51: (4)             "DrawBorderThenFill",
52: (4)             "Write",
53: (4)             "Unwrite",
54: (4)             "ShowPartial",
55: (4)             "ShowIncreasingSubsets",
56: (4)             "SpiralIn",
57: (4)             "AddTextLetterByLetter",
58: (4)             "RemoveTextLetterByLetter",
59: (4)             "ShowSubmobjectsOneByOne",
60: (4)             "AddTextWordByWord",
61: (0)         ]
62: (0)         import itertools as it
63: (0)         from typing import TYPE_CHECKING, Callable, Iterable, Sequence

```

```

64: (0)         import numpy as np
65: (0)         if TYPE_CHECKING:
66: (4)             from manim.mobject.text.text_mobject import Text
67: (0)             from manim.mobject.opengl.opengl_surface import OpenGLSurface
68: (0)             from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLMobject
69: (0)             from manim.utils.color import ManimColor
70: (0)             from .. import config
71: (0)             from ..animation.animation import Animation
72: (0)             from ..animation.composition import Succession
73: (0)             from ..constants import TAU
74: (0)             from ..mobject.mobject import Group, Mobject
75: (0)             from ..mobject.types.vectorized_mobject import VMobject
76: (0)             from ..utils.bezier import integer_interpolate
77: (0)             from ..utils.rate_functions import double_smooth, linear
78: (0)         class ShowPartial(Animation):
79: (4)             """Abstract class for Animations that show the VMobject partially.
80: (4)             Raises
81: (4)             -----
82: (4)             :class:`TypeError`
83: (8)                 If ``mobject`` is not an instance of :class:`~.VMobject`.
84: (4)             See Also
85: (4)             -----
86: (4)             :class:`Create`, :class:`~.ShowPassingFlash`
87: (4)             """
88: (4)         def __init__(
89: (8)             self,
90: (8)             mobject: VMobject | OpenGLMobject | OpenGLSurface | None,
91: (8)             **kwargs,
92: (4)         ):
93: (8)             pointwise = getattr(mobject, "pointwise_become_partial", None)
94: (8)             if not callable(pointwise):
95: (12)                 raise NotImplementedError("This animation is not defined for this
Mobject.")
96: (8)             super().__init__(mobject, **kwargs)
97: (4)         def interpolate_submobject(
98: (8)             self,
99: (8)             submobject: Mobject,
100: (8)             starting_submobject: Mobject,
101: (8)             alpha: float,
102: (4)         ) -> None:
103: (8)             submobject.pointwise_become_partial(
104: (12)                 starting_submobject, *self._get_bounds(alpha)
105: (8)             )
106: (4)             def _get_bounds(self, alpha: float) -> None:
107: (8)                 raise NotImplementedError("Please use Create or ShowPassingFlash")
108: (0)         class Create(ShowPartial):
109: (4)             """Incrementally show a VMobject.
110: (4)             Parameters
111: (4)             -----
112: (4)             mobject
113: (8)                 The VMobject to animate.
114: (4)             Raises
115: (4)             -----
116: (4)             :class:`TypeError`
117: (8)                 If ``mobject`` is not an instance of :class:`~.VMobject`.
118: (4)             Examples
119: (4)             -----
120: (4)             .. manim:: CreateScene
121: (8)                 class CreateScene(Scene):
122: (12)                     def construct(self):
123: (16)                         self.play(Create(Square()))
124: (4)             See Also
125: (4)             -----
126: (4)             :class:`~.ShowPassingFlash`
127: (4)             """
128: (4)             def __init__(
129: (8)                 self,
130: (8)                 mobject: VMobject | OpenGLMobject | OpenGLSurface,
131: (8)                 lag_ratio: float = 1.0,

```

```

132: (8)             introducer: bool = True,
133: (8)             **kwargs,
134: (4)         ) -> None:
135: (8)             super().__init__(mobject, lag_ratio=lag_ratio, introducer=introducer,
**kwargs)
136: (4)             def _get_bounds(self, alpha: float) -> tuple[int, float]:
137: (8)                 return (0, alpha)
138: (0)         class Uncreate(Create):
139: (4)             """Like :class:`Create` but in reverse.
140: (4)             Examples
141: (4)             -----
142: (4)             .. manim:: ShowUncreate
143: (8)                 class ShowUncreate(Scene):
144: (12)                     def construct(self):
145: (16)                         self.play(Uncreate(Square()))
146: (4)             See Also
147: (4)             -----
148: (4)             :class:`Create`
149: (4)             """
150: (4)             def __init__(
151: (8)                 self,
152: (8)                 mobject: VMobject | OpenGLMobject,
153: (8)                 reverse_rate_function: bool = True,
154: (8)                 remover: bool = True,
155: (8)                 **kwargs,
156: (4)             ) -> None:
157: (8)                 super().__init__(
158: (12)                     mobject,
159: (12)                     reverse_rate_function=reverse_rate_function,
160: (12)                     introducer=False,
161: (12)                     remover=remover,
162: (12)                     **kwargs,
163: (8)                 )
164: (0)         class DrawBorderThenFill(Animation):
165: (4)             """Draw the border first and then show the fill.
166: (4)             Examples
167: (4)             -----
168: (4)             .. manim:: ShowDrawBorderThenFill
169: (8)                 class ShowDrawBorderThenFill(Scene):
170: (12)                     def construct(self):
171: (16)                         self.play(DrawBorderThenFill(Square(fill_opacity=1,
fill_color=ORANGE)))
172: (4)             """
173: (4)             def __init__(
174: (8)                 self,
175: (8)                 vmobject: VMobject | OpenGLMobject,
176: (8)                 run_time: float = 2,
177: (8)                 rate_func: Callable[[float], float] = double_smooth,
178: (8)                 stroke_width: float = 2,
179: (8)                 stroke_color: str = None,
180: (8)                 draw_border_animation_config: dict = {}, # what does this dict
accept?
181: (8)                 fill_animation_config: dict = {},
182: (8)                 introducer: bool = True,
183: (8)                 **kwargs,
184: (4)             ) -> None:
185: (8)                 self._typecheck_input(vmobject)
186: (8)                 super().__init__(
187: (12)                     vmobject,
188: (12)                     run_time=run_time,
189: (12)                     introducer=introducer,
190: (12)                     rate_func=rate_func,
191: (12)                     **kwargs,
192: (8)                 )
193: (8)                 self.stroke_width = stroke_width
194: (8)                 self.stroke_color = stroke_color
195: (8)                 self.draw_border_animation_config = draw_border_animation_config
196: (8)                 self.fill_animation_config = fill_animation_config
197: (8)                 self.outline = self.get_outline()

```

```

198: (4)             def _typecheck_input(self, vmobject: VMobject | OpenGLVMobject) -> None:
199: (8)                 if not isinstance(vmobject, (VMobject, OpenGLVMobject)):
200: (12)                     raise TypeError("DrawBorderThenFill only works for vectorized
Mobjects")
201: (4)             def begin(self) -> None:
202: (8)                 self.outline = self.get_outline()
203: (8)                 super().begin()
204: (4)             def get_outline(self) -> Mobject:
205: (8)                 outline = self.mobject.copy()
206: (8)                 outline.set_fill(opacity=0)
207: (8)                 for sm in outline.family_members_with_points():
208: (12)                     sm.set_stroke(color=self.get_stroke_color(sm),
width=self.stroke_width)
209: (8)             return outline
210: (4)             def get_stroke_color(self, vmobject: VMobject | OpenGLVMobject) ->
ManimColor:
211: (8)                 if self.stroke_color:
212: (12)                     return self.stroke_color
213: (8)                 elif vmobject.get_stroke_width() > 0:
214: (12)                     return vmobject.get_stroke_color()
215: (8)                 return vmobject.get_color()
216: (4)             def get_all_mobjects(self) -> Sequence[Mobject]:
217: (8)                 return [*super().get_all_mobjects(), self.outline]
218: (4)             def interpolate_submobject(
219: (8)                 self,
220: (8)                 submobject: Mobject,
221: (8)                 starting_submobject: Mobject,
222: (8)                 outline,
223: (8)                 alpha: float,
224: (4)             ) -> None: # FIXME: not matching the parent class? What is outline doing
here?
225: (8)                 index: int
226: (8)                 subalpha: int
227: (8)                 index, subalpha = integer_interpolate(0, 2, alpha)
228: (8)                 if index == 0:
229: (12)                     submobject.pointwise_become_partial(outline, 0, subalpha)
230: (12)                     submobject.match_style(outline)
231: (8)                 else:
232: (12)                     submobject.interpolate(outline, starting_submobject, subalpha)
233: (0)             class Write(DrawBorderThenFill):
234: (4)                 """Simulate hand-writing a :class:`~.Text` or hand-drawing a
:class:`~.VMobject`.
235: (4)             Examples
236: (4)             -----
237: (4)             .. manim:: ShowWrite
238: (8)                 class ShowWrite(Scene):
239: (12)                     def construct(self):
240: (16)                         self.play(Write(Text("Hello", font_size=144)))
241: (4)             .. manim:: ShowWriteReversed
242: (8)                 class ShowWriteReversed(Scene):
243: (12)                     def construct(self):
244: (16)                         self.play(Write(Text("Hello", font_size=144), reverse=True,
remover=False))
245: (4)             Tests
246: (4)             -----
247: (4)             Check that creating empty :class:`.Write` animations works::
248: (8)                 >>> from manim import Write, Text
249: (8)                 >>> Write(Text(''))
250: (8)                 Write(Text(''))
251: (4)                 """
252: (4)                 def __init__(
253: (8)                     self,
254: (8)                     vmobject: VMobject | OpenGLVMobject,
255: (8)                     rate_func: Callable[[float], float] = linear,
256: (8)                     reverse: bool = False,
257: (8)                     **kwargs,
258: (4)                 ) -> None:
259: (8)                     run_time: float | None = kwargs.pop("run_time", None)
260: (8)                     lag_ratio: float | None = kwargs.pop("lag_ratio", None)

```

```

261: (8)             run_time, lag_ratio = self._set_default_config_from_length(
262: (12)             vmobject,
263: (12)             run_time,
264: (12)             lag_ratio,
265: (8)         )
266: (8)         self.reverse = reverse
267: (8)         if "remover" not in kwargs:
268: (12)             kwargs["remover"] = reverse
269: (8)         super().__init__(
270: (12)             vmobject,
271: (12)             rate_func=rate_func,
272: (12)             run_time=run_time,
273: (12)             lag_ratio=lag_ratio,
274: (12)             introducer=not reverse,
275: (12)             **kwargs,
276: (8)         )
277: (4)     def _set_default_config_from_length(
278: (8)         self,
279: (8)         vmobject: VMobject | OpenGLVMobject,
280: (8)         run_time: float | None,
281: (8)         lag_ratio: float | None,
282: (4)     ) -> tuple[float, float]:
283: (8)         length = len(vmobject.family_members_with_points())
284: (8)         if run_time is None:
285: (12)             if length < 15:
286: (16)                 run_time = 1
287: (12)             else:
288: (16)                 run_time = 2
289: (8)         if lag_ratio is None:
290: (12)             lag_ratio = min(4.0 / max(1.0, length), 0.2)
291: (8)         return run_time, lag_ratio
292: (4)     def reverse_submobjects(self) -> None:
293: (8)         self.mobject.invert(recursive=True)
294: (4)     def begin(self) -> None:
295: (8)         if self.reverse:
296: (12)             self.reverse_submobjects()
297: (8)         super().begin()
298: (4)     def finish(self) -> None:
299: (8)         super().finish()
300: (8)         if self.reverse:
301: (12)             self.reverse_submobjects()
302: (0) class Unwrite(Write):
303: (4)     """Simulate erasing by hand a :class:`~.Text` or a :class:`~.VMobject` .
304: (4)     Parameters
305: (4)     -----
306: (4)     reverse
307: (8)         Set True to have the animation start erasing from the last submobject
first.
308: (4)     Examples
309: (4)     -----
310: (4)     .. manim :: UnwriteReverseTrue
311: (8)         class UnwriteReverseTrue(Scene):
312: (12)             def construct(self):
313: (16)                 text = Tex("Alice and Bob").scale(3)
314: (16)                 self.add(text)
315: (16)                 self.play(Unwrite(text))
316: (4)     .. manim:: UnwriteReverseFalse
317: (8)         class UnwriteReverseFalse(Scene):
318: (12)             def construct(self):
319: (16)                 text = Tex("Alice and Bob").scale(3)
320: (16)                 self.add(text)
321: (16)                 self.play(Unwrite(text, reverse=False))
322: (4)     """
323: (4)     def __init__(
324: (8)         self,
325: (8)         vmobject: VMobject,
326: (8)         rate_func: Callable[[float], float] = linear,
327: (8)         reverse: bool = True,
328: (8)         **kwargs,

```

```

329: (4)             ) -> None:
330: (8)             run_time: float | None = kwargs.pop("run_time", None)
331: (8)             lag_ratio: float | None = kwargs.pop("lag_ratio", None)
332: (8)             run_time, lag_ratio = self._set_default_config_from_length(
333: (12)             vmobject,
334: (12)             run_time,
335: (12)             lag_ratio,
336: (8)         )
337: (8)         super().__init__(
338: (12)             vmobject,
339: (12)             run_time=run_time,
340: (12)             lag_ratio=lag_ratio,
341: (12)             reverse_rate_function=True,
342: (12)             reverse=reverse,
343: (12)             **kwargs,
344: (8)         )
345: (0)     class SpiralIn(Animation):
346: (4)         """Create the Mobject with sub-Mobjects flying in on spiral trajectories.
347: (4)         Parameters
348: (4)         -----
349: (4)         shapes
350: (8)             The Mobject on which to be operated.
351: (4)         scale_factor
352: (8)             The factor used for scaling the effect.
353: (4)         fade_in_fraction
354: (8)             Fractional duration of initial fade-in of sub-Mobjects as they fly
inward.
355: (4)     Examples
356: (4)     -----
357: (4)     .. manim :: SpiralInExample
358: (8)         class SpiralInExample(Scene):
359: (12)             def construct(self):
360: (16)                 pi = MathTex(r"\pi").scale(7)
361: (16)                 pi.shift(2.25 * LEFT + 1.5 * UP)
362: (16)                 circle = Circle(color=GREEN_C, fill_opacity=1).shift(LEFT)
363: (16)                 square = Square(color=BLUE_D, fill_opacity=1).shift(UP)
364: (16)                 shapes = VGroup(pi, circle, square)
365: (16)                 self.play(SpiralIn(shapes))
366: (4)
367: (4)             def __init__(
368: (8)                 self,
369: (8)                 shapes: Mobject,
370: (8)                 scale_factor: float = 8,
371: (8)                 fade_in_fraction=0.3,
372: (8)                 **kwargs,
373: (4)             ) -> None:
374: (8)                 self.shapes = shapes.copy()
375: (8)                 self.scale_factor = scale_factor
376: (8)                 self.shape_center = shapes.get_center()
377: (8)                 self.fade_in_fraction = fade_in_fraction
378: (8)                 for shape in shapes:
379: (12)                     shape.final_position = shape.get_center()
380: (12)                     shape.initial_position = (
381: (16)                         shape.final_position
382: (16)                         + (shape.final_position - self.shape_center) *
self.scale_factor
383: (12)
384: (12)                     )
385: (12)                     shape.move_to(shape.initial_position)
386: (8)                     shape.save_state()
387: (4)             super().__init__(shapes, introducer=True, **kwargs)
388: (8)             def interpolate_mobject(self, alpha: float) -> None:
389: (8)                 alpha = self.rate_func(alpha)
390: (12)                 for original_shape, shape in zip(self.shapes, self.mobject):
391: (12)                     shape.restore()
392: (12)                     fill_opacity = original_shape.get_fill_opacity()
393: (12)                     stroke_opacity = original_shape.get_stroke_opacity()
394: (16)                     new_fill_opacity = min(
395: (16)                         fill_opacity, alpha * fill_opacity / self.fade_in_fraction
)

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
396: (12)           new_stroke_opacity = min(
397: (16)             stroke_opacity, alpha * stroke_opacity / self.fade_in_fraction
398: (12)         )
399: (12)           shape.shift((shape.final_position - shape.initial_position) *
alpha)
400: (12)             shape.rotate(TAU * alpha, about_point=self.shape_center)
401: (12)             shape.rotate(-TAU * alpha, about_point=shape.get_center_of_mass())
402: (12)             shape.set_fill(opacity=new_fill_opacity)
403: (12)             shape.set_stroke(opacity=new_stroke_opacity)
404: (0) class ShowIncreasingSubsets(Animation):
405: (4)     """Show one submobject at a time, leaving all previous ones displayed on
screen.
406: (4)     Examples
407: (4)     -----
408: (4)     .. manim:: ShowIncreasingSubsetsScene
409: (8)       class ShowIncreasingSubsetsScene(Scene):
410: (12)         def construct(self):
411: (16)             p = VGroup(Dot(), Square(), Triangle())
412: (16)             self.add(p)
413: (16)             self.play(ShowIncreasingSubsets(p))
414: (16)             self.wait()
415: (4)     """
416: (4)     def __init__(
417: (8)         self,
418: (8)         group: Mobject,
419: (8)         suspend_mobject_updating: bool = False,
420: (8)         int_func: Callable[[np.ndarray], np.ndarray] = np.floor,
421: (8)         reverse_rate_function=False,
422: (8)         **kwargs,
423: (4)     ) -> None:
424: (8)         self.all_submobs = list(group.submobjects)
425: (8)         self.int_func = int_func
426: (8)         for mobj in self.all_submobs:
427: (12)             mobj.set_opacity(0)
428: (8)         super().__init__(
429: (12)             group,
430: (12)             suspend_mobject_updating=suspend_mobject_updating,
431: (12)             reverse_rate_function=reverse_rate_function,
432: (12)             **kwargs,
433: (8)         )
434: (4)     def interpolate_mobject(self, alpha: float) -> None:
435: (8)         n_submobs = len(self.all_submobs)
436: (8)         value = (
437: (12)             1 - self.rate_func(alpha)
438: (12)             if self.reverse_rate_function
439: (12)             else self.rate_func(alpha)
440: (8)         )
441: (8)         index = int(self.int_func(value * n_submobs))
442: (8)         self.update_submobject_list(index)
443: (4)     def update_submobject_list(self, index: int) -> None:
444: (8)         for mobj in self.all_submobs[:index]:
445: (12)             mobj.set_opacity(1)
446: (8)         for mobj in self.all_submobs[index:]:
447: (12)             mobj.set_opacity(0)
448: (0) class AddTextLetterByLetter(ShowIncreasingSubsets):
449: (4)     """Show a :class:`~.Text` letter by letter on the scene.
450: (4)     Parameters
451: (4)     -----
452: (4)     time_per_char
453: (8)         Frequency of appearance of the letters.
454: (4)     .. tip::
455: (8)         This is currently only possible for class:`~.Text` and not for
class:`~.MathTex`
456: (4)     """
457: (4)     def __init__(
458: (8)         self,
459: (8)         text: Text,
460: (8)         suspend_mobject_updating: bool = False,
461: (8)         int_func: Callable[[np.ndarray], np.ndarray] = np.ceil,

```

```

462: (8)                     rate_func: Callable[[float], float] = linear,
463: (8)                     time_per_char: float = 0.1,
464: (8)                     run_time: float | None = None,
465: (8)                     reverse_rate_function=False,
466: (8)                     introducer=True,
467: (8)                     **kwargs,
468: (4)             ) -> None:
469: (8)                 self.time_per_char = time_per_char
470: (8)                 if not text.family_members_with_points():
471: (12)                     raise ValueError(
472: (16)                         f"The text mobject {text} does not seem to contain any
characters."
473: (12)
474: (8)
475: (12)             len(text)
476: (8)         super().__init__(
477: (12)             text,
478: (12)             suspend_mobject_updating=suspend_mobject_updating,
479: (12)             int_func=int_func,
480: (12)             rate_func=rate_func,
481: (12)             run_time=run_time,
482: (12)             reverse_rate_function=reverse_rate_function,
483: (12)             introducer=introducer,
484: (12)             **kwargs,
485: (8)
486: (0)     class RemoveTextLetterByLetter(AddTextLetterByLetter):
487: (4)         """Remove a :class:`~.Text` letter by letter from the scene.
488: (4)         Parameters
489: (4)         -----
490: (4)         time_per_char
491: (8)             Frequency of appearance of the letters.
492: (4)         .. tip::
493: (8)             This is currently only possible for class:`~.Text` and not for
class:`~.MathTex`
494: (4)
495: (4)     def __init__(
496: (8)         self,
497: (8)         text: Text,
498: (8)         suspend_mobject_updating: bool = False,
499: (8)         int_func: Callable[[np.ndarray], np.ndarray] = np.ceil,
500: (8)         rate_func: Callable[[float], float] = linear,
501: (8)         time_per_char: float = 0.1,
502: (8)         run_time: float | None = None,
503: (8)         reverse_rate_function=True,
504: (8)         introducer=False,
505: (8)         remover=True,
506: (8)         **kwargs,
507: (4)     ) -> None:
508: (8)         super().__init__(
509: (12)             text,
510: (12)             suspend_mobject_updating=suspend_mobject_updating,
511: (12)             int_func=int_func,
512: (12)             rate_func=rate_func,
513: (12)             time_per_char=time_per_char,
514: (12)             run_time=run_time,
515: (12)             reverse_rate_function=reverse_rate_function,
516: (12)             introducer=introducer,
517: (12)             remover=remover,
518: (12)             **kwargs,
519: (8)
520: (0)     class ShowSubmobjectsOneByOne(ShowIncreasingSubsets):
521: (4)         """Show one submobject at a time, removing all previously displayed ones
from screen."""
522: (4)     def __init__(
523: (8)         self,
524: (8)         group: Iterable[Mobject],
525: (8)         int_func: Callable[[np.ndarray], np.ndarray] = np.ceil,
526: (8)         **kwargs,

```

```

527: (4)             ) -> None:
528: (8)             new_group = Group(*group)
529: (8)             super().__init__(new_group, int_func=int_func, **kwargs)
530: (4)             def update_submobject_list(self, index: int) -> None:
531: (8)             current_submobjects = self.all_submobs[:index]
532: (8)             for mob in current_submobjects[:-1]:
533: (12)                 mob.set_opacity(0)
534: (8)             if len(current_submobjects) > 0:
535: (12)                 current_submobjects[-1].set_opacity(1)
536: (0)             class AddTextWordByWord(Succession):
537: (4)                 """Show a :class:`~.Text` word by word on the scene. Note: currently
broken."""
538: (4)                 def __init__(
539: (8)                     self,
540: (8)                     text_mobject: Text,
541: (8)                     run_time: float = None,
542: (8)                     time_per_char: float = 0.06,
543: (8)                     **kwargs,
544: (4)                 ) -> None:
545: (8)                     self.time_per_char = time_per_char
546: (8)                     tpc = self.time_per_char
547: (8)                     anims = it.chain(
548: (12)                         *
549: (16)                         [
550: (20)                             ShowIncreasingSubsets(word, run_time=tpc * len(word)),
551: (20)                             Animation(word, run_time=0.005 * len(word) ** 1.5),
552: (16)                         ]
553: (16)                         for word in text_mobject
554: (12)                     )
555: (8)                 )
556: (8)             super().__init__(*anims, **kwargs)

```

-----

File 8 - `__init__.py`:

1: (0)

File 9 - `__init__.py`:

```

1: (0)             """Set the global config and logger."""
2: (0)             from __future__ import annotations
3: (0)             import logging
4: (0)             from contextlib import contextmanager
5: (0)             from typing import Any, Generator
6: (0)             from .cli_colors import parse_cli_ctx
7: (0)             from .logger_utils import make_logger
8: (0)             from .utils import ManimConfig, ManimFrame, make_config_parser
9: (0)             __all__ = [
10: (4)                 "logger",
11: (4)                 "console",
12: (4)                 "error_console",
13: (4)                 "config",
14: (4)                 "frame",
15: (4)                 "tempconfig",
16: (4)                 "cli_ctx_settings",
17: (0)             ]
18: (0)             parser = make_config_parser()
19: (0)             logger, console, error_console = make_logger(
20: (4)                 parser["logger"],
21: (4)                 parser["CLI"]["verbosity"],
22: (0)             )
23: (0)             cli_ctx_settings = parse_cli_ctx(parser["CLI_CTX"])
24: (0)             logging.getLogger("PIL").setLevel(logging.INFO)
25: (0)             logging.getLogger("matplotlib").setLevel(logging.INFO)
26: (0)             config = ManimConfig().digest_parser(parser)
27: (0)             frame = ManimFrame(config)

```

```

28: (0)          @contextmanager
29: (0)          def tempconfig(temp: ManimConfig | dict[str, Any]) -> Generator[None, None,
None]:
30: (4)              """Context manager that temporarily modifies the global ``config`` object.
31: (4)              Inside the ``with`` statement, the modified config will be used. After
32: (4)              context manager exits, the config will be restored to its original state.
33: (4)              Parameters
34: (4)                  -----
35: (4)          temp
36: (8)              Object whose keys will be used to temporarily update the global
37: (8)              ``config``.
38: (4)          Examples
39: (4)          -----
40: (4)          Use ``with tempconfig({...})`` to temporarily change the default values of
41: (4)          certain config options.
42: (4)          .. code-block:: pycon
43: (7)              >>> config["frame_height"]
44: (7)              8.0
45: (7)              >>> with tempconfig({"frame_height": 100.0}):
46: (7)                  ...     print(config["frame_height"])
47: (7)                  ...
48: (7)                  100.0
49: (7)              >>> config["frame_height"]
50: (7)              8.0
51: (4)          """
52: (4)          global config
53: (4)          original = config.copy()
54: (4)          temp = {k: v for k, v in temp.items() if k in original}
55: (4)          config.update(temp)
56: (4)          try:
57: (8)              yield
58: (4)          finally:
59: (8)              config.update(original) # update, not assignment!

```

-----

## File 10 - animation.py:

```

1: (0)          """Animate mobjects."""
2: (0)          from __future__ import annotations
3: (0)          from manim.mobject.opengl.opengl_mobject import OpenGLMobject
4: (0)          from .. import config, logger
5: (0)          from ..constants import RendererType
6: (0)          from ..mobject import mobject
7: (0)          from ..mobject.mobject import Mobject
8: (0)          from ..mobject.opengl import opengl_mobject
9: (0)          from ..utils.rate_functions import linear, smooth
10: (0)         __all__ = ["Animation", "Wait", "override_animation"]
11: (0)          from copy import deepcopy
12: (0)          from typing import TYPE_CHECKING, Callable, Iterable, Sequence
13: (0)          from typing_extensions import Self
14: (0)          if TYPE_CHECKING:
15: (4)              from manim.scene.scene import Scene
16: (0)          DEFAULT_ANIMATION_RUN_TIME: float = 1.0
17: (0)          DEFAULT_ANIMATION_LAG_RATIO: float = 0.0
18: (0)          class Animation:
19: (4)              """An animation.
20: (4)              Animations have a fixed time span.
21: (4)              Parameters
22: (4)                  -----
23: (4)          mobject
24: (8)              The mobject to be animated. This is not required for all types of
animations.
25: (4)          lag_ratio
26: (8)              Defines the delay after which the animation is applied to submobjects.
27: (8)              is relative to the duration of the animation.
28: (8)              This does not influence the total
29: (8)              runtime of the animation. Instead the runtime of individual animations

```

```

is
30: (8)           adjusted so that the complete animation has the defined run time.
31: (4)           run_time
32: (8)           The duration of the animation in seconds.
33: (4)           rate_func
34: (8)           The function defining the animation progress based on the relative
runtime (see :mod:`~.rate_functions`).
35: (8)           For example ``rate_func(0.5)`` is the proportion of the animation that
is done
36: (8)           after half of the animations run time.
37: (4)           reverse_rate_function
38: (8)           Reverses the rate function of the animation. Setting
``reverse_rate_function``
39: (8)           does not have any effect on ``remover`` or ``introducer``. These need
to be
40: (8)           set explicitly if an introducer-animation should be turned into a
remover one
41: (8)           and vice versa.
name
42: (4)           The name of the animation. This gets displayed while rendering the
43: (8)           Defaults to <class-name>(<Mobject-name>).
remover
44: (4)           Whether the given mobject should be removed from the scene after this
45: (4)           animation.
46: (8)           suspend_mobject_updating
47: (4)           Whether updaters of the mobject should be suspended during the
48: (8)           animation.
49: (4)           .. NOTE::
50: (8)           In the current implementation of this class, the specified rate
function is applied
51: (8)           to
52: (8)           :class:`.Animation`
53: (8)           rate function
54: (8)           instead
55: (8)           of just ``alpha``).
56: (4)           Examples
57: (4)           -----
58: (4)           .. manim:: LagRatios
59: (8)           class LagRatios(Scene):
60: (12)           def construct(self):
61: (16)           ratios = [0, 0.1, 0.5, 1, 2] # demonstrated lag_ratios
62: (16)           group = VGroup(*[Dot() for _ in
range(4)]).arrange_submobjects()
63: (16)           groups = VGroup(*[group.copy() for _ in
ratios]).arrange_submobjects(buff=1)
64: (16)           self.add(groups)
65: (16)           self.add(Text("lag_ratio = ", font_size=36).next_to(groups,
UP, buff=1.5))
66: (16)
67: (20)           for group, ratio in zip(groups, ratios):
68: (16)               self.add(Text(str(ratio), font_size=36).next_to(group,
UP))
69: (20)               self.play(AnimationGroup(*[
70: (20)                   group.animate(lag_ratio=ratio, run_time=1.5).shift(DOWN *
2)
71: (16)                   for group, ratio in zip(groups, ratios)
72: (16)               ]))
73: (20)               self.play(groups.animate(run_time=1, lag_ratio=0.1).shift(UP *
2))
74: (4)           """
75: (8)           def __new__(
76: (8)               cls,
77: (8)               mobject=None,
78: (8)               *args,
use_override=True,

```

```

79: (8)                         **kwargs,
80: (4)                     ) -> Self:
81: (8)                     if isinstance(mobject, Mobject) and use_override:
82: (12)                         func = mobject.animation_override_for(cls)
83: (12)                         if func is not None:
84: (16)                             anim = func(mobject, *args, **kwargs)
85: (16)                             logger.debug(
86: (20)                                 f"The {cls.__name__} animation has been overridden for"
87: (20)                                 f"{type(mobject).__name__} mobjects. use_override = False"
88: (20)                                 f" can "
89: (20)                                 f" be used as keyword argument to prevent animation"
90: (20)                                 overriding."
91: (8)                         )
92: (16)                     return anim
93: (8)                 return super().__new__(cls)
94: (4)             def __init__(
95: (8)                 self,
96: (8)                 mobject: Mobject | None,
97: (8)                 lag_ratio: float = DEFAULT_ANIMATION_LAG_RATIO,
98: (8)                 run_time: float = DEFAULT_ANIMATION_RUN_TIME,
99: (8)                 rate_func: Callable[[float], float] = smooth,
100: (8)                reverse_rate_function: bool = False,
101: (8)                name: str = None,
102: (8)                remover: bool = False, # remove a mobject from the screen?
103: (8)                suspend_mobject_updating: bool = True,
104: (8)                introducer: bool = False,
105: (8)                *,
106: (4)                _on_finish: Callable[[], None] = lambda _: None,
107: (8)                **kwargs,
108: (8)             ) -> None:
109: (8)                 self._typecheck_input(mobject)
110: (8)                 self.run_time: float = run_time
111: (8)                 self.rate_func: Callable[[float], float] = rate_func
112: (8)                 self.reverse_rate_function: bool = reverse_rate_function
113: (8)                 self.name: str | None = name
114: (8)                 self.remover: bool = remover
115: (8)                 self.introducer: bool = introducer
116: (8)                 self.suspend_mobject_updating: bool = suspend_mobject_updating
117: (8)                 self.lag_ratio: float = lag_ratio
118: (12)                 self._on_finish: Callable[[Scene], None] = _on_finish
119: (12)                 if config["renderer"] == RendererType.OPENGL:
120: (16)                     self.starting_mobject: OpenGLMobject = OpenGLMobject()
121: (12)                     self.mobject: OpenGLMobject = (
122: (8)                         mobject if mobject is not None else OpenGLMobject()
123: (12)                     )
124: (12)                 else:
125: (8)                     self.starting_mobject: Mobject = Mobject()
126: (12)                     self.mobject: Mobject = mobject if mobject is not None else
127: (8)             if kwargs:
128: (12)                 logger.debug("Animation received extra kwargs: %s", kwargs)
129: (16)             if hasattr(self, "CONFIG"):
130: (20)                 logger.error(
131: (20)                     (
132: (16)                         "CONFIG has been removed from ManimCommunity.",
133: (12)                         "Please use keyword arguments instead.",
134: (12)                     ),
135: (8)                 )
136: (12)             def _typecheck_input(self, mobject: Mobject | None) -> None:
137: (8)                 if mobject is None:
138: (12)                     logger.debug("Animation with empty mobject")
139: (12)                 elif not isinstance(mobject, (Mobject, OpenGLMobject)):
140: (8)                     raise TypeError("Animation only works on Mobjects")
141: (12)             def __str__(self) -> str:
142: (8)                 if self.name:
143: (4)                     return self.name

```

```
                     return f"{self.__class__.__name__}({str(self.mobject)})"
```

```
                     def __repr__(self) -> str:
```

```

144: (8)             return str(self)
145: (4)             def begin(self) -> None:
146: (8)                 """Begin the animation.
147: (8)                 This method is called right as an animation is being played. As much
148: (8)                 initialization as possible, especially any mobject copying, should
live in this
149: (8)                 method.
150: (8)                 """
151: (8)                 if self.run_time <= 0:
152: (12)                     raise ValueError(
153: (16)                         f"{self} has a run_time of <= 0 seconds, this cannot be
rendered correctly. "
154: (16)                         "Please set the run_time to be positive"
155: (12)                     )
156: (8)                     self.starting_mobject = self.create_starting_mobject()
157: (8)                     if self.suspend_mobject_updating:
158: (12)                         self.mobject.suspend_updating()
159: (8)                         self.interpolate(0)
160: (4)             def finish(self) -> None:
161: (8)                 """Finish the animation.
162: (8)                 This method gets called when the animation is over.
163: (8)                 """
164: (8)                 self.interpolate(1)
165: (8)                 if self.suspend_mobject_updating and self.mobject is not None:
166: (12)                     self.mobject.resume_updating()
167: (4)             def clean_up_from_scene(self, scene: Scene) -> None:
168: (8)                 """Clean up the :class:`~.Scene` after finishing the animation.
169: (8)                 This includes to :meth:`~.Scene.remove` the Animation's
170: (8)                 :class:`~.Mobject` if the animation is a remover.
171: (8)                 Parameters
172: (8)                 -----
173: (8)                 scene
174: (12)                     The scene the animation should be cleaned up from.
175: (8)                     """
176: (8)                     self._on_finish(scene)
177: (8)                     if self.is_remover():
178: (12)                         scene.remove(self.mobject)
179: (4)             def _setup_scene(self, scene: Scene) -> None:
180: (8)                 """Setup up the :class:`~.Scene` before starting the animation.
181: (8)                 This includes to :meth:`~.Scene.add` the Animation's
182: (8)                 :class:`~.Mobject` if the animation is an introducer.
183: (8)                 Parameters
184: (8)                 -----
185: (8)                 scene
186: (12)                     The scene the animation should be cleaned up from.
187: (8)                     """
188: (8)                     if scene is None:
189: (12)                         return
190: (8)                     if (
191: (12)                         self.is_introducer()
192: (12)                         and self.mobject not in scene.get_mobject_family_members()
193: (8)                     ):
194: (12)                         scene.add(self.mobject)
195: (4)             def create_starting_mobject(self) -> Mobject:
196: (8)                 return self.mobject.copy()
197: (4)             def get_all_mobjects(self) -> Sequence[Mobject]:
198: (8)                 """Get all mobjects involved in the animation.
199: (8)                 Ordering must match the ordering of arguments to
interpolate_submobject
200: (8)                     Returns
201: (8)                     -----
202: (8)                     Sequence[Mobject]
203: (12)                     The sequence of mobjects.
204: (8)                     """
205: (8)                     return self.mobject, self.starting_mobject
206: (4)             def get_all_families_zipped(self) -> Iterable[tuple]:
207: (8)                 if config["renderer"] == RendererType.OPENGL:
208: (12)                     return zip(*[mob.get_family() for mob in self.get_all_mobjects()])
209: (8)                     return zip(

```

```

210: (12)                         *(mob.family_members_with_points() for mob in
self.get_all_mobjects())
211: (8)                           )
212: (4)                           def update_mobjects(self, dt: float) -> None:
213: (8)                           """
214: (8)                               Updates things like starting_mobject, and (for
215: (8)                               Transforms) target_mobject. Note, since typically
216: (8)                               (always?) self.mobject will have its updating
217: (8)                               suspended during the animation, this will do
218: (8)                               nothing to self.mobject.
219: (8)                           """
220: (8)                           for mob in self.get_all_mobjects_to_update():
221: (12)                             mob.update(dt)
222: (4)                           def get_all_mobjects_to_update(self) -> list[Mobject]:
223: (8)                           """Get all mobjects to be updated during the animation.
224: (8)                           Returns
225: (8)                           -----
226: (8)                           List[Mobject]
227: (12)                             The list of mobjects to be updated during the animation.
228: (8)                           """
229: (8)                           return list(filter(lambda m: m is not self.mobject,
self.get_all_mobjects()))
230: (4)                           def copy(self) -> Animation:
231: (8)                           """Create a copy of the animation.
232: (8)                           Returns
233: (8)                           -----
234: (8)                           Animation
235: (12)                             A copy of ``self``
236: (8)                           """
237: (8)                           return deepcopy(self)
238: (4)                           def interpolate(self, alpha: float) -> None:
239: (8)                           """Set the animation progress.
240: (8)                           This method gets called for every frame during an animation.
241: (8)                           Parameters
242: (8)                           -----
243: (8)                           alpha
244: (12)                             The relative time to set the animation to, 0 meaning the start, 1
meaning
245: (12)
246: (8)
247: (8)                           self.interpolate_mobject(alpha)
248: (4)                           def interpolate_mobject(self, alpha: float) -> None:
249: (8)                           """Interpolates the mobject of the :class:`Animation` based on alpha
value.
250: (8)
251: (8)
252: (8)
253: (12)                             Parameters
254: (12)                             alpha
255: (12)                             A float between 0 and 1 expressing the ratio to which the
animation
256: (8)                             is completed. For example, alpha-values of 0, 0.5, and 1
correspond
257: (8)                             to the animation being completed 0%, 50%, and 100%, respectively.
258: (8)
259: (12)
260: (12)
261: (4)                           families = list(self.get_all_families_zipped())
262: (8)                           for i, mobs in enumerate(families):
263: (8)                             sub_alpha = self.get_sub_alpha(alpha, i, len(families))
264: (8)                             self.interpolate_submobject(*mobs, sub_alpha)
265: (8)
266: (4)                           def interpolate_submobject(
267: (8)                             self,
268: (8)                             submobject: Mobject,
269: (8)                             starting_submobject: Mobject,
270: (8)                             alpha: float,
271: (8)                           ) -> Animation:
272: (8)                             pass
273: (4)                           def get_sub_alpha(self, alpha: float, index: int, num_submobjects: int) ->
float:
274: (8)                           """Get the animation progress of any submobjects subanimation.
275: (8)                           Parameters
276: (8)                           -----

```

```

272: (8)                               alpha
273: (12)                             The overall animation progress
274: (8)                               index
275: (12)                             The index of the subanimation.
276: (8)                               num_subobjects
277: (12)                             The total count of subanimations.
278: (8)                               Returns
279: (8)                               -----
280: (8)                               float
281: (12)                             The progress of the subanimation.
282: (8)                               """
283: (8)                               lag_ratio = self.lag_ratio
284: (8)                               full_length = (num_subobjects - 1) * lag_ratio + 1
285: (8)                               value = alpha * full_length
286: (8)                               lower = index * lag_ratio
287: (8)                               if self.reverse_rate_function:
288: (12)                                 return self.rate_func(1 - (value - lower))
289: (8)                               else:
290: (12)                                 return self.rate_func(value - lower)
291: (4)                               def set_run_time(self, run_time: float) -> Animation:
292: (8)                               """Set the run time of the animation.
293: (8)                               Parameters
294: (8)                               -----
295: (8)                               run_time
296: (12)                             The new time the animation should take in seconds.
297: (8)                               .. note::
298: (12)                             The run_time of an animation should not be changed while it is
already
299: (12)                               running.
300: (8)                               Returns
301: (8)                               -----
302: (8)                               Animation
303: (12)                             ``self``
304: (8)                               """
305: (8)                               self.run_time = run_time
306: (8)                               return self
307: (4)                               def get_run_time(self) -> float:
308: (8)                               """Get the run time of the animation.
309: (8)                               Returns
310: (8)                               -----
311: (8)                               float
312: (12)                             The time the animation takes in seconds.
313: (8)                               """
314: (8)                               return self.run_time
315: (4)                               def set_rate_func(
316: (8)                               self,
317: (8)                               rate_func: Callable[[float], float],
318: (4) ) -> Animation:
319: (8)                               """Set the rate function of the animation.
320: (8)                               Parameters
321: (8)                               -----
322: (8)                               rate_func
323: (12)                             The new function defining the animation progress based on the
relative runtime (see :mod:`~.rate_functions`).
324: (12)                               Returns
325: (8)                               -----
326: (8)                               Animation
327: (8)                             ``self``
328: (12)                               """
329: (8)                               self.rate_func = rate_func
330: (8)                               return self
331: (8)                               def get_rate_func(
332: (4)                               self,
333: (8) ) -> Callable[[float], float]:
334: (4)                               """Get the rate function of the animation.
335: (8)                               Returns
336: (8)                               -----
337: (8)                               Callable[[float], float]
338: (8)                               The rate function of the animation.
339: (12)

```

```

340: (8)             """
341: (8)             return self.rate_func
342: (4)             def set_name(self, name: str) -> Animation:
343: (8)                 """Set the name of the animation.
344: (8)                 Parameters
345: (8)                 -----
346: (8)                 name
347: (12)                The new name of the animation.
348: (8)                 Returns
349: (8)                 -----
350: (8)                 Animation
351: (12)                ``self``
352: (8)                 """
353: (8)                 self.name = name
354: (8)                 return self
355: (4)             def is_remover(self) -> bool:
356: (8)                 """Test if the animation is a remover.
357: (8)                 Returns
358: (8)                 -----
359: (8)                 bool
360: (12)                ``True`` if the animation is a remover, ``False`` otherwise.
361: (8)                 """
362: (8)                 return self.remover
363: (4)             def is_introducer(self) -> bool:
364: (8)                 """Test if the animation is an introducer.
365: (8)                 Returns
366: (8)                 -----
367: (8)                 bool
368: (12)                ``True`` if the animation is an introducer, ``False`` otherwise.
369: (8)                 """
370: (8)                 return self.introducer
371: (0)             def prepare_animation(
372: (4)                 anim: Animation | mobject._AnimationBuilder,
373: (0)             ) -> Animation:
374: (4)                 r"""Returns either an unchanged animation, or the animation built
375: (4)                 from a passed animation factory.
376: (4)                 Examples
377: (4)                 -----
378: (4)                 :::
379: (8)                     >>> from manim import Square, FadeIn
380: (8)                     >>> s = Square()
381: (8)                     >>> prepare_animation(FadeIn(s))
382: (8)                     FadeIn(Square)
383: (4)                     :::
384: (8)                     >>> prepare_animation(s.animate.scale(2).rotate(42))
385: (8)                     _MethodAnimation(Square)
386: (4)                     :::
387: (8)                     >>> prepare_animation(42)
388: (8)                     Traceback (most recent call last):
389: (8)                         ...
390: (8)                         TypeError: Object 42 cannot be converted to an animation
391: (4)                         """
392: (4)             if isinstance(anim, mobject._AnimationBuilder):
393: (8)                 return anim.build()
394: (4)             if isinstance(anim, opengl_mobject._AnimationBuilder):
395: (8)                 return anim.build()
396: (4)             if isinstance(anim, Animation):
397: (8)                 return anim
398: (4)             raise TypeError(f"Object {anim} cannot be converted to an animation")
399: (0)         class Wait(Animation):
400: (4)             """A "no operation" animation.
401: (4)             Parameters
402: (4)             -----
403: (4)             run_time
404: (8)                 The amount of time that should pass.
405: (4)             stop_condition
406: (8)                 A function without positional arguments that evaluates to a boolean.
407: (8)                 The function is evaluated after every new frame has been rendered.
408: (8)                 Playing the animation stops after the return value is truthy, or

```

```

409: (8)           after the specified ``run_time`` has passed.
410: (4)           frozen_frame
411: (8)           Controls whether or not the wait animation is static, i.e.,
corresponds
412: (8)           to a frozen frame. If ``False`` is passed, the render loop still
413: (8)           progresses through the animation as usual and (among other things)
414: (8)           continues to call updater functions. If ``None`` (the default value),
415: (8)           the :meth:`.Scene.play` call tries to determine whether the Wait call
416: (8)           can be static or not itself via :meth:`.Scene.should_mobjects_update`.
417: (4)           kwargs
418: (8)           Keyword arguments to be passed to the parent class,
:class:`.Animation`.
419: (4) """
420: (4)     def __init__(
421: (8)         self,
422: (8)         run_time: float = 1,
423: (8)         stop_condition: Callable[[], bool] | None = None,
424: (8)         frozen_frame: bool | None = None,
425: (8)         rate_func: Callable[[float], float] = linear,
426: (8)         **kwargs,
427: (4)     ):
428: (8)         if stop_condition and frozen_frame:
429: (12)             raise ValueError("A static Wait animation cannot have a stop
condition.")
430: (8)         self.duration: float = run_time
431: (8)         self.stop_condition = stop_condition
432: (8)         self.is_static_wait: bool = frozen_frame
433: (8)         super().__init__(None, run_time=run_time, rate_func=rate_func,
**kwargs)
434: (8)         self.mobject.shader_wrapper_list = []
435: (4)     def begin(self) -> None:
436: (8)         pass
437: (4)     def finish(self) -> None:
438: (8)         pass
439: (4)     def clean_up_from_scene(self, scene: Scene) -> None:
440: (8)         pass
441: (4)     def update_mobjects(self, dt: float) -> None:
442: (8)         pass
443: (4)     def interpolate(self, alpha: float) -> None:
444: (8)         pass
445: (0)     def override_animation(
446: (4)         animation_class: type[Animation],
447: (0)     ) -> Callable[[Callable], Callable]:
448: (4)         """Decorator used to mark methods as overrides for specific
:class:`~.Animation` types.
449: (4)         Should only be used to decorate methods of classes derived from
:class:`~.Mobject`.
450: (4) defined
451: (4)         ``Animation`` overrides get inherited to subclasses of the ``Mobject`` who
them. They don't override subclasses of the ``Animation`` they override.
See Also
-----
:meth:`~.Mobject.add_animation_override`
Parameters
-----
animation_class
    The animation to be overridden.
Returns
-----
Callable[[Callable], Callable]
    The actual decorator. This marks the method as overriding an
animation.
463: (4) Examples
464: (4) -----
465: (4) .. manim:: OverrideAnimationExample
466: (8)     class MySquare(Square):
467: (12)         @override_animation(FadeIn)
468: (12)         def _fade_in_override(self, **kwargs):
469: (16)             return Create(self, **kwargs)

```

```

470: (8)             class OverrideAnimationExample(Scene):
471: (12)             def construct(self):
472: (16)                 self.play(FadeIn(MySquare()))
473: (4)
474: (4)             """
475: (8)             def decorator(func):
476: (8)                 func._override_animation = animation_class
477: (4)             return func
478: (4)         return decorator

```

---

File 11 - indication.py:

```

1: (0)             """Animations drawing attention to particular mobjects.
2: (0)             Examples
3: (0)             -----
4: (0)             .. manim:: Indications
5: (4)             class Indications(Scene):
6: (8)                 def construct(self):
7: (12)                     indications =
[ApplyWave,Circumscribe,Flash,FocusOn,Indicate>ShowPassingFlash,Wiggle]
8: (12)                     names = [Tex(i.__name__).scale(3) for i in indications]
9: (12)                     self.add(names[0])
10: (12)                     for i in range(len(names)):
11: (16)                         if indications[i] is Flash:
12: (20)                             self.play(Flash(UP))
13: (16)                         elif indications[i] is ShowPassingFlash:
14: (20)                             self.play(ShowPassingFlash(Underline(names[i])))
15: (16)                         else:
16: (20)                             self.play(indications[i](names[i]))
17: (16)                     self.play(AnimationGroup(
18: (20)                         FadeOut(names[i], shift=UP*1.5),
19: (20)                         FadeIn(names[(i+1)%len(names)], shift=UP*1.5),
20: (16)                     ))
21: (0)
22: (0)             """
23: (4)             __all__ = [
24: (4)                 "FocusOn",
25: (4)                 "Indicate",
26: (4)                 "Flash",
27: (4)                 "ShowPassingFlash",
28: (4)                 "ShowPassingFlashWithThinningStrokeWidth",
29: (4)                 "ApplyWave",
30: (4)                 "Circumscribe",
31: (4)                 "Wiggle",
32: (0)
33: (0)             from typing import Callable, Iterable, Optional, Tuple, Type, Union
34: (0)             import numpy as np
35: (0)             from manim.mobject.geometry.arc import Circle, Dot
36: (0)             from manim.mobject.geometry.line import Line
37: (0)             from manim.mobject.geometry.polygram import Rectangle
38: (0)             from manim.mobject.geometry.shape_matchers import SurroundingRectangle
39: (0)             from manim.scene.scene import Scene
40: (0)             from .. import config
41: (0)             from ..animation.animation import Animation
42: (0)             from ..animation.composition import AnimationGroup, Succession
43: (0)             from ..animation.creation import Create, ShowPartial, Uncreate
44: (0)             from ..animation.fading import FadeIn, FadeOut
45: (0)             from ..animation.movement import Homotopy
46: (0)             from ..animation.transform import Transform
47: (0)             from ..constants import *
48: (0)             from ..mobject.mobject import Mobject
49: (0)             from ..mobject.types.vectorized_mobject import VGroup, VMobject
50: (0)             from ..utils.bezier import interpolate, inverse_interpolate
51: (0)             from ..utils.color import GREY, YELLOW, ParsableManimColor
52: (0)             from ..utils.deprecation import deprecated
53: (0)             from ..utils.rate_functions import smooth, there_and_back, wiggle
54: (0)             from ..utils.space_ops import normalize
55: (4)             class FocusOn(Transform):
56: (4)                 """Shrink a spotlight to a position.

```

```

56: (4)                      Parameters
57: (4)                      -----
58: (4)                      focus_point
59: (8)                        The point at which to shrink the spotlight. If it is a
:class:`.~Mobject` its center will be used.
60: (4)                      opacity
61: (8)                        The opacity of the spotlight.
62: (4)                      color
63: (8)                        The color of the spotlight.
64: (4)                      run_time
65: (8)                        The duration of the animation.
66: (4)                      kwargs
67: (8)                        Additional arguments to be passed to the :class:`~.Succession`  

constructor
68: (4)                      Examples
69: (4)                      -----
70: (4)                      .. manim:: UsingFocusOn
71: (8)                        class UsingFocusOn(Scene):
72: (12)                          def construct(self):
73: (16)                            dot = Dot(color=YELLOW).shift(DOWN)
74: (16)                            self.add(Tex("Focusing on the dot below:"), dot)
75: (16)                            self.play(FocusOn(dot))
76: (16)                            self.wait()
77: (4)                          """
78: (4)                      def __init__(
79: (8)                        self,
80: (8)                          focus_point: Union[np.ndarray, Mobject],
81: (8)                          opacity: float = 0.2,
82: (8)                          color: str = GREY,
83: (8)                          run_time: float = 2,
84: (8)                          **kwargs
85: (4)                      ) -> None:
86: (8)                        self.focus_point = focus_point
87: (8)                        self.color = color
88: (8)                        self.opacity = opacity
89: (8)                        remover = True
90: (8)                        starting_dot = Dot(
91: (12)                          radius=config["frame_x_radius"] + config["frame_y_radius"],
92: (12)                          stroke_width=0,
93: (12)                          fill_color=self.color,
94: (12)                          fill_opacity=0,
95: (8)                      )
96: (8)                        super().__init__(starting_dot, run_time=run_time, remover=remover,
**kwargs)
97: (4)                      def create_target(self) -> Dot:
98: (8)                        little_dot = Dot(radius=0)
99: (8)                        little_dot.set_fill(self.color, opacity=self.opacity)
100: (8)                        little_dot.add_updater(lambda d: d.move_to(self.focus_point))
101: (8)                        return little_dot
102: (0)                      class Indicate(Transform):
103: (4)                        """Indicate a Mobject by temporarily resizing and recoloring it.
104: (4)                        Parameters
105: (4)                        -----
106: (4)                        mobject
107: (8)                          The mobject to indicate.
108: (4)                        scale_factor
109: (8)                          The factor by which the mobject will be temporally scaled
110: (4)                        color
111: (8)                          The color the mobject temporally takes.
112: (4)                        rate_func
113: (8)                          The function defining the animation progress at every point in time.
114: (4)                        kwargs
115: (8)                          Additional arguments to be passed to the :class:`~.Succession`  

constructor
116: (4)                      Examples
117: (4)                      -----
118: (4)                      .. manim:: UsingIndicate
119: (8)                        class UsingIndicate(Scene):
120: (12)                          def construct(self):

```

```

121: (16)                     tex = Tex("Indicate").scale(3)
122: (16)                     self.play(Indicate(tex))
123: (16)                     self.wait()
124: (4) """
125: (4)     def __init__(
126: (8)         self,
127: (8)         mobject: Mobject,
128: (8)         scale_factor: float = 1.2,
129: (8)         color: str = YELLOW,
130: (8)         rate_func: Callable[[float, Optional[float]], np.ndarray] =
131: (8)             there_and_back,
132: (8)             **kwargs
133: (8)         ) -> None:
134: (8)             self.color = color
135: (8)             self.scale_factor = scale_factor
136: (4)             super().__init__(mobject, rate_func=rate_func, **kwargs)
137: (8)     def create_target(self) -> Mobject:
138: (8)         target = self.mobject.copy()
139: (8)         target.scale(self.scale_factor)
140: (8)         target.set_color(self.color)
141: (0)         return target
142: (4)     class Flash(AnimationGroup):
143: (4)         """Send out lines in all directions.
144: (4)         Parameters
145: (4)         -----
146: (8)         point
147: (8)             The center of the flash lines. If it is a :class:`.~Mobject` its
center will be used.
148: (8)         line_length
149: (8)             The length of the flash lines.
150: (8)         num_lines
151: (8)             The number of flash lines.
152: (8)         flash_radius
153: (8)             The distance from `point` at which the flash lines start.
154: (8)         line_stroke_width
155: (8)             The stroke width of the flash lines.
156: (8)         color
157: (8)             The color of the flash lines.
158: (8)         time_width
159: (4)             The time width used for the flash lines. See
:class:`.~ShowPassingFlash` for more details.
160: (8)         run_time
161: (8)             The duration of the animation.
162: (8)         kwargs
163: (4)             Additional arguments to be passed to the :class:`~.Succession`  

constructor
164: (4) Examples
165: (4) -----
166: (8) .. manim:: UsingFlash
167: (12)     class UsingFlash(Scene):
168: (16)         def construct(self):
169: (16)             dot = Dot(color=YELLOW).shift(DOWN)
170: (16)             self.add(Tex("Flash the dot below:"), dot)
171: (16)             self.play(Flash(dot))
172: (16)             self.wait()
173: (8) .. manim:: FlashOnCircle
174: (12)     class FlashOnCircle(Scene):
175: (16)         def construct(self):
176: (16)             radius = 2
177: (16)             circle = Circle(radius)
178: (16)             self.add(circle)
179: (20)             self.play(Flash(
180: (20)                 circle, line_length=1,
181: (20)                 num_lines=30, color=RED,
182: (20)                 flash_radius=radius+SMALL_BUFF,
183: (20)                 time_width=0.3, run_time=2,
184: (16)                 rate_func = rush_from
185: (4)             ))

```

```

186: (4)             def __init__(self,
187: (8)                 point: Union[np.ndarray, Mobject],
188: (8)                 line_length: float = 0.2,
189: (8)                 num_lines: int = 12,
190: (8)                 flash_radius: float = 0.1,
191: (8)                 line_stroke_width: int = 3,
192: (8)                 color: str = YELLOW,
193: (8)                 time_width: float = 1,
194: (8)                 run_time: float = 1.0,
195: (8)                 **kwargs
196: (8)
197: (4)             ) -> None:
198: (8)             if isinstance(point, Mobject):
199: (12)                 self.point = point.get_center()
200: (8)
201: (12)             else:
202: (8)                 self.point = point
203: (8)             self.color = color
204: (8)             self.line_length = line_length
205: (8)             self.num_lines = num_lines
206: (8)             self.flash_radius = flash_radius
207: (8)             self.line_stroke_width = line_stroke_width
208: (8)             self.run_time = run_time
209: (8)             self.time_width = time_width
210: (8)             self.animation_config = kwargs
211: (8)             self.lines = self.create_lines()
212: (8)             animations = self.create_line_anims()
213: (4)             super().__init__(*animations, group=self.lines)
214: (8)
215: (8)         def create_lines(self) -> VGroup:
216: (12)             lines = VGroup()
217: (12)             for angle in np.arange(0, TAU, TAU / self.num_lines):
218: (12)                 line = Line(self.point, self.point + self.line_length * RIGHT)
219: (12)                 line.shift((self.flash_radius) * RIGHT)
220: (12)                 line.rotate(angle, about_point=self.point)
221: (12)                 lines.add(line)
222: (8)             lines.set_color(self.color)
223: (8)             lines.set_stroke(width=self.line_stroke_width)
224: (8)             return lines
225: (4)         def create_line_anims(self) -> Iterable["ShowPassingFlash"]:
226: (8)
227: (12)             return [
228: (16)                 ShowPassingFlash(
229: (16)                     line,
230: (16)                     time_width=self.time_width,
231: (16)                     run_time=self.run_time,
232: (16)                     **self.animation_config,
233: (12)
234: (0)             ]
235: (4)             class ShowPassingFlash(ShowPartial):
236: (4)                 """Show only a sliver of the VMobject each frame.
237: (4)                 Parameters
238: (4)                 -----
239: (4)                 mobject
240: (8)                     The mobject whose stroke is animated.
241: (4)                 time_width
242: (8)                     The length of the sliver relative to the length of the stroke.
243: (4)                 Examples
244: (4)                 -----
245: (4)                 .. manim:: TimeWidthValues
246: (8)                     class TimeWidthValues(Scene):
247: (12)                         def construct(self):
248: (16)                             p = RegularPolygon(5, color=DARK_GRAY,
249: (16)                             stroke_width=6).scale(3)
250: (16)                             lbl = VMobject()
251: (16)                             self.add(p, lbl)
252: (16)                             p = p.copy().set_color(BLUE)
253: (16)                             for time_width in [0.2, 0.5, 1, 2]:
254: (20)                                 lbl.become(Tex(r"\texttt{time}\_width=
{{%.1f}}"%time_width))
255: (20)

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
253: (24)                                p.copy().set_color(BLUE),
254: (24)                                run_time=2,
255: (24)                                time_width=time_width
256: (20)                                ))
257: (4) See Also
258: (4) -----
259: (4) :class:`~.Create`
260: (4)
261: (4) def __init__(self, mobject: "VMobject", time_width: float = 0.1, **kwargs)
-> None:
262: (8)     self.time_width = time_width
263: (8)     super().__init__(mobject, remover=True, introducer=True, **kwargs)
264: (4) def _get_bounds(self, alpha: float) -> Tuple[float]:
265: (8)     tw = self.time_width
266: (8)     upper = interpolate(0, 1 + tw, alpha)
267: (8)     lower = upper - tw
268: (8)     upper = min(upper, 1)
269: (8)     lower = max(lower, 0)
270: (8)     return (lower, upper)
271: (4) def clean_up_from_scene(self, scene: Scene) -> None:
272: (8)     super().clean_up_from_scene(scene)
273: (8)     for submob, start in self.get_all_families_zipped():
274: (12)         submob.pointwise_become_partial(start, 0, 1)
275: (0) class ShowPassingFlashWithThinningStrokeWidth(AnimationGroup):
276: (4)     def __init__(self, vmobject, n_segments=10, time_width=0.1, remover=True,
**kwargs):
277: (8)         self.n_segments = n_segments
278: (8)         self.time_width = time_width
279: (8)         self.remover = remover
280: (8)         max_stroke_width = vmobject.get_stroke_width()
281: (8)         max_time_width = kwargs.pop("time_width", self.time_width)
282: (8)         super().__init__(
283: (12)             *
284: (16)                 ShowPassingFlash(
285: (20)                     vmobject.copy().set_stroke(width=stroke_width),
286: (20)                     time_width=time_width,
287: (20)                     **kwargs,
288: (16)             )
289: (16)             for stroke_width, time_width in zip(
290: (20)                 np.linspace(0, max_stroke_width, self.n_segments),
291: (20)                 np.linspace(max_time_width, 0, self.n_segments),
292: (16)             )
293: (12)         ),
294: (8)     )
295: (0) class ApplyWave(Homotopy):
296: (4)     """Send a wave through the Mobject distorting it temporarily.
297: (4)     Parameters
298: (4)     -----
299: (4)     mobject
300: (8)         The mobject to be distorted.
301: (4)     direction
302: (8)         The direction in which the wave nudges points of the shape
303: (4)     amplitude
304: (8)         The distance points of the shape get shifted
305: (4)     wave_func
306: (8)         The function defining the shape of one wave flank.
307: (4)     time_width
308: (8)         The length of the wave relative to the width of the mobject.
309: (4)     ripples
310: (8)         The number of ripples of the wave
311: (4)     run_time
312: (8)         The duration of the animation.
313: (4)     Examples
314: (4)     -----
315: (4)     .. manim:: ApplyingWaves
316: (8)         class ApplyingWaves(Scene):
317: (12)             def construct(self):
318: (16)                 tex = Tex("WaveWaveWaveWaveWave").scale(2)
319: (16)                 self.play(ApplyWave(tex))

```

```

320: (16)                         self.play(ApplyWave(
321: (20)                           tex,
322: (20)                           direction=RIGHT,
323: (20)                           time_width=0.5,
324: (20)                           amplitude=0.3
325: (16)                         ))
326: (16)                         self.play(ApplyWave(
327: (20)                           tex,
328: (20)                           rate_func=linear,
329: (20)                           ripples=4
330: (16)                         ))
331: (4) """
332: (4)     def __init__(
333: (8)         self,
334: (8)         mobject: Mobject,
335: (8)         direction: np.ndarray = UP,
336: (8)         amplitude: float = 0.2,
337: (8)         wave_func: Callable[[float], float] = smooth,
338: (8)         time_width: float = 1,
339: (8)         ripples: int = 1,
340: (8)         run_time: float = 2,
341: (8)         **kwargs
342: (4)     ) -> None:
343: (8)         x_min = mobject.get_left()[0]
344: (8)         x_max = mobject.get_right()[0]
345: (8)         vect = amplitude * normalize(direction)
346: (8)         def wave(t):
347: (12)             t = 1 - t
348: (12)             if t >= 1 or t <= 0:
349: (16)                 return 0
350: (12)             phases = ripples * 2
351: (12)             phase = int(t * phases)
352: (12)             if phase == 0:
353: (16)                 return wave_func(t * phases)
354: (12)             elif phase == phases - 1:
355: (16)                 t -= phase / phases # Time relative to the phase
356: (16)                 return (1 - wave_func(t * phases)) * (2 * (ripples % 2) - 1)
357: (12)             else:
358: (16)                 phase = int((phase - 1) / 2)
359: (16)                 t -= (2 * phase + 1) / phases
360: (16)                 return (1 - 2 * wave_func(t * ripples)) * (1 - 2 * ((phase) %
2))
361: (8)         def homotopy(
362: (12)             x: float,
363: (12)             y: float,
364: (12)             z: float,
365: (12)             t: float,
366: (8)         ) -> Tuple[float, float, float]:
367: (12)             upper = interpolate(0, 1 + time_width, t)
368: (12)             lower = upper - time_width
369: (12)             relative_x = inverse_interpolate(x_min, x_max, x)
370: (12)             wave_phase = inverse_interpolate(lower, upper, relative_x)
371: (12)             nudge = wave(wave_phase) * vect
372: (12)             return np.array([x, y, z]) + nudge
373: (8)             super().__init__(homotopy, mobject, run_time=run_time, **kwargs)
374: (0)     class Wiggle(Animation):
375: (4)         """Wiggle a Mobject.
376: (4)         Parameters
377: (4)         -----
378: (4)         mobject
379: (8)             The mobject to wiggle.
380: (4)         scale_value
381: (8)             The factor by which the mobject will be temporarily scaled.
382: (4)         rotation_angle
383: (8)             The wiggle angle.
384: (4)         n_wiggles
385: (8)             The number of wiggles.
386: (4)         scale_about_point
387: (8)             The point about which the mobject gets scaled.

```

```

388: (4)          rotate_about_point
389: (8)            The point around which the mobject gets rotated.
390: (4)          run_time
391: (8)            The duration of the animation
392: (4)          Examples
393: (4)          -----
394: (4)          .. manim:: ApplyingWaves
395: (8)            class ApplyingWaves(Scene):
396: (12)              def construct(self):
397: (16)                  tex = Tex("Wiggle").scale(3)
398: (16)                  self.play(Wiggle(tex))
399: (16)                  self.wait()
400: (4)          """
401: (4)          def __init__(
402: (8)              self,
403: (8)              mobject: Mobject,
404: (8)              scale_value: float = 1.1,
405: (8)              rotation_angle: float = 0.01 * TAU,
406: (8)              n_wiggles: int = 6,
407: (8)              scale_about_point: Optional[np.ndarray] = None,
408: (8)              rotate_about_point: Optional[np.ndarray] = None,
409: (8)              run_time: float = 2,
410: (8)              **kwargs
411: (4)          ) -> None:
412: (8)              self.scale_value = scale_value
413: (8)              self.rotation_angle = rotation_angle
414: (8)              self.n_wiggles = n_wiggles
415: (8)              self.scale_about_point = scale_about_point
416: (8)              self.rotate_about_point = rotate_about_point
417: (8)              super().__init__(mobject, run_time=run_time, **kwargs)
418: (4)          def get_scale_about_point(self) -> np.ndarray:
419: (8)              if self.scale_about_point is None:
420: (12)                  return self.mobject.get_center()
421: (8)              return self.scale_about_point
422: (4)          def get_rotate_about_point(self) -> np.ndarray:
423: (8)              if self.rotate_about_point is None:
424: (12)                  return self.mobject.get_center()
425: (8)              return self.rotate_about_point
426: (4)          def interpolate_submobject(
427: (8)              self,
428: (8)              submobject: Mobject,
429: (8)              starting_submobject: Mobject,
430: (8)              alpha: float,
431: (4)          ) -> None:
432: (8)              submobject.points[:, :] = starting_submobject.points
433: (8)              submobject.scale(
434: (12)                  interpolate(1, self.scale_value, there_and_back(alpha)),
435: (12)                  about_point=self.get_scale_about_point(),
436: (8)
437: (8)
438: (12)                  submobject.rotate(
439: (12)                      wiggle(alpha, self.n_wiggles) * self.rotation_angle,
440: (8)                      about_point=self.get_rotate_about_point(),
441: (0)          )
442: (4)          class Circumscribe(Succession):
443: (4)              """Draw a temporary line surrounding the mobject.
444: (4)              Parameters
445: (4)              -----
446: (4)              mobject
447: (8)                  The mobject to be circumscribed.
448: (8)              shape
449: (8)                  The shape with which to surround the given mobject. Should be either
450: (8)                      :class:`~.Rectangle` or :class:`~.Circle`
451: (8)              fade_in
452: (4)                  Whether to make the surrounding shape to fade in. It will be drawn
453: (8)              otherwise.
454: (4)              fade_out
455: (8)                  Whether to make the surrounding shape to fade out. It will be undrawn
456: (8)              otherwise.
457: (4)              time_width

```

```

455: (8)           The time_width of the drawing and undrawing. Gets ignored if either
`fade_in` or `fade_out` is `True`.
456: (4)           buff
457: (8)           The distance between the surrounding shape and the given mobject.
458: (4)           color
459: (8)           The color of the surrounding shape.
460: (4)           run_time
461: (8)           The duration of the entire animation.
462: (4)           kwargs
463: (8)           Additional arguments to be passed to the :class:`~.Succession`  

constructor
464: (4)           Examples
465: (4)           -----
466: (4)           .. manim:: UsingCircumscribe
467: (8)           class UsingCircumscribe(Scene):
468: (12)           def construct(self):
469: (16)           lbl = Tex(r"Circum-\\"\\scribe").scale(2)
470: (16)           self.add(lbl)
471: (16)           self.play(Circumscribe(lbl))
472: (16)           self.play(Circumscribe(lbl, Circle))
473: (16)           self.play(Circumscribe(lbl, fade_out=True))
474: (16)           self.play(Circumscribe(lbl, time_width=2))
475: (16)           self.play(Circumscribe(lbl, Circle, True))
476: (4)           """
477: (4)           def __init__(
478: (8)               self,
479: (8)               mobject: Mobject,
480: (8)               shape: Type = Rectangle,
481: (8)               fade_in=False,
482: (8)               fade_out=False,
483: (8)               time_width=0.3,
484: (8)               buff: float = SMALL_BUFF,
485: (8)               color: ParsableManimColor = YELLOW,
486: (8)               run_time=1,
487: (8)               stroke_width=DEFAULT_STROKE_WIDTH,
488: (8)               **kwargs
489: (4)           ):
490: (8)           if shape is Rectangle:
491: (12)           frame = SurroundingRectangle(
492: (16)               mobject,
493: (16)               color,
494: (16)               buff,
495: (16)               stroke_width=stroke_width,
496: (12)           )
497: (8)           elif shape is Circle:
498: (12)           frame = Circle(color=color, stroke_width=stroke_width).surround(
499: (16)               mobject,
500: (16)               buffer_factor=1,
501: (12)           )
502: (12)           radius = frame.width / 2
503: (12)           frame.scale((radius + buff) / radius)
504: (8)           else:
505: (12)           raise ValueError("shape should be either Rectangle or Circle.")
506: (8)           if fade_in and fade_out:
507: (12)           super().__init__(
508: (16)               FadeIn(frame, run_time=run_time / 2),
509: (16)               FadeOut(frame, run_time=run_time / 2),
510: (16)               **kwargs,
511: (12)           )
512: (8)           elif fade_in:
513: (12)           frame.reverse_direction()
514: (12)           super().__init__(
515: (16)               FadeIn(frame, run_time=run_time / 2),
516: (16)               Uncreate(frame, run_time=run_time / 2),
517: (16)               **kwargs,
518: (12)           )
519: (8)           elif fade_out:
520: (12)           super().__init__(
521: (16)               Create(frame, run_time=run_time / 2),

```

```

522: (16)                     FadeOut(frame, run_time=run_time / 2),
523: (16)                         **kwargs,
524: (12)                     )
525: (8)                     else:
526: (12)                         super().__init__(
527: (16)                             ShowPassingFlash(frame, time_width, run_time=run_time),
**kwargs
528: (12)                     )

```

-----

## File 12 - cli\_colors.py:

```

1: (0)                     from __future__ import annotations
2: (0)                     import configparser
3: (0)                     from cloup import Context, HelpFormatter, HelpTheme, Style
4: (0)                     __all__ = ["parse_cli_ctx"]
5: (0)                     def parse_cli_ctx(parser: configparser.SectionProxy) -> Context:
6: (4)                         formatter_settings: dict[str, str | int] = {
7: (8)                             "indent_increment": int(parser["indent_increment"]),
8: (8)                             "width": int(parser["width"]),
9: (8)                             "col1_max_width": int(parser["col1_max_width"]),
10: (8)                            "col2_min_width": int(parser["col2_min_width"]),
11: (8)                            "col_spacing": int(parser["col_spacing"]),
12: (8)                            "row_sep": parser["row_sep"] if parser["row_sep"] else None,
13: (4)                         }
14: (4)                         theme_settings = {}
15: (4)                         theme_keys = {
16: (8)                             "command_help",
17: (8)                             "invoked_command",
18: (8)                             "heading",
19: (8)                             "constraint",
20: (8)                             "section_help",
21: (8)                             "col1",
22: (8)                             "col2",
23: (8)                             "epilog",
24: (4)                         }
25: (4)                         for k, v in parser.items():
26: (8)                             if k in theme_keys and v:
27: (12)                                 theme_settings.update({k: Style(v)})
28: (4)                         formatter = {}
29: (4)                         theme = parser["theme"] if parser["theme"] else None
30: (4)                         if theme is None:
31: (8)                             formatter = HelpFormatter.settings(
32: (12)                                 theme=HelpTheme(**theme_settings), **formatter_settings # type:
ignore[arg-type]
33: (8)                             )
34: (4)                             elif theme.lower() == "dark":
35: (8)                                 formatter = HelpFormatter.settings(
36: (12)                                     theme=HelpTheme.dark().with_(**theme_settings),
**formatter_settings # type: ignore[arg-type]
37: (8)                             )
38: (4)                             elif theme.lower() == "light":
39: (8)                                 formatter = HelpFormatter.settings(
40: (12)                                     theme=HelpTheme.light().with_(**theme_settings),
**formatter_settings # type: ignore[arg-type]
41: (8)                             )
42: (4)                             return Context.settings(
43: (8)                                 align_option_groups=parser["align_option_groups"].lower() == "true",
44: (8)                                 align_sections=parser["align_sections"].lower() == "true",
45: (8)                                 show_constraints=True,
46: (8)                                 formatter_settings=formatter,
47: (4)                             )

```

-----

## File 13 - composition.py:

```
1: (0)                     """Tools for displaying multiple animations at once."""

```

```

2: (0)         from __future__ import annotations
3: (0)         import types
4: (0)         from typing import TYPE_CHECKING, Callable, Iterable, Sequence
5: (0)         import numpy as np
6: (0)         from manim._config import config
7: (0)         from manim.animation.animation import Animation, prepare_animation
8: (0)         from manim.constants import RendererType
9: (0)         from manim.mobject.mobject import Group, Mobject
10: (0)        from manim.mobject.opengl.opengl_mobject import OpenGLGroup
11: (0)        from manim.scene.scene import Scene
12: (0)        from manim.utils.iterables import remove_list_redundancies
13: (0)        from manim.utils.parameter_parsing import flatten_iterable_parameters
14: (0)        from manim.utils.rate_functions import linear
15: (0)        if TYPE_CHECKING:
16: (4)            from manim.mobject.opengl_vectorized_mobject import OpenGLGroup
17: (4)            from manim.mobject.types.vectorized_mobject import VGroup
18: (0)        __all__ = ["AnimationGroup", "Succession", "LaggedStart", "LaggedStartMap"]
19: (0)        DEFAULT_LAGGED_START_LAG_RATIO: float = 0.05
20: (0)        class AnimationGroup(Animation):
21: (4)            """Plays a group or series of :class:`~.Animation` .
22: (4)            Parameters
23: (4)            -----
24: (4)            animations
25: (8)                Sequence of :class:`~.Animation` objects to be played.
26: (4)            group
27: (8)                A group of multiple :class:`~.Mobject` .
28: (4)            run_time
29: (8)                The duration of the animation in seconds.
30: (4)            rate_func
31: (8)                The function defining the animation progress based on the relative
32: (8)                runtime (see :mod:`~.rate_functions` ) .
33: (4)            lag_ratio
34: (8)                Defines the delay after which the animation is applied to subobjects.
A lag_ratio of
35: (8)                    ``n.nn`` means the next animation will play when ``nnn%`` of the
current animation has played.
36: (8)                    Defaults to 0.0, meaning that all animations will be played together.
37: (8)                    This does not influence the total runtime of the animation. Instead
the runtime
38: (8)                    of individual animations is adjusted so that the complete animation
has the defined
39: (8)                    run time.
40: (4)                    """
41: (4)        def __init__(
42: (8)            self,
43: (8)            *animations: Animation | Iterable[Animation] |
types.GeneratorType[Animation],
44: (8)            group: Group | VGroup | OpenGLGroup | OpenGLGroup = None,
45: (8)            run_time: float | None = None,
46: (8)            rate_func: Callable[[float], float] = linear,
47: (8)            lag_ratio: float = 0,
48: (8)            **kwargs,
49: (4)        ) -> None:
50: (8)            arg_anim = flatten_iterable_parameters(animations)
51: (8)            self.animations = [prepare_animation(anim) for anim in arg_anim]
52: (8)            self.rate_func = rate_func
53: (8)            self.group = group
54: (8)            if self.group is None:
55: (12)                mobjects = remove_list_redundancies(
56: (16)                    [anim.mobject for anim in self.animations if not
anim.is_introducer()],
57: (12)
58: (12)
59: (16)
60: (12)
61: (16)
62: (8)
63: (12)
**kwargs
)
```

```

64: (8) )
65: (8) self.run_time: float = self.init_run_time(run_time)
66: (4) def get_all_mobjects(self) -> Sequence[Mobject]:
67: (8) return list(self.group)
68: (4) def begin(self) -> None:
69: (8) if self.run_time <= 0:
70: (12)     tmp = (
71: (16)         "Please set the run_time to be positive"
72: (16)         if len(self.animations) != 0
73: (16)         else "Please add at least one Animation with positive
run_time"
74: (12)     )
75: (12)     raise ValueError(
76: (16)         f"{self} has a run_time of 0 seconds, this cannot be "
77: (16)         f"rendered correctly. {tmp}."
78: (12)     )
79: (8)     self.anim_group_time = 0.0
80: (8)     if self.suspend_mobject_updating:
81: (12)         self.group.suspend_updating()
82: (8)     for anim in self.animations:
83: (12)         anim.begin()
84: (4)     def _setup_scene(self, scene) -> None:
85: (8)         for anim in self.animations:
86: (12)             anim._setup_scene(scene)
87: (4)     def finish(self) -> None:
88: (8)         self.interpolate(1)
89: (8)         self.anims_begun[:] = True
90: (8)         self.anims_finished[:] = True
91: (8)         if self.suspend_mobject_updating:
92: (12)             self.group.resume_updating()
93: (4)     def clean_up_from_scene(self, scene: Scene) -> None:
94: (8)         self._on_finish(scene)
95: (8)         for anim in self.animations:
96: (12)             if self.remover:
97: (16)                 anim.remover = self.remover
98: (12)                 anim.clean_up_from_scene(scene)
99: (4)     def update_mobjects(self, dt: float) -> None:
100: (8)         for anim in self.anims_with_timings["anim"][
101: (12)             self.anims_begun & ~self.anims_finished
102: (8)         ]:
103: (12)             anim.update_mobjects(dt)
104: (4)     def init_run_time(self, run_time) -> float:
105: (8)         """Calculates the run time of the animation, if different from
``run_time``.
106: (8)         Parameters
107: (8)         -----
108: (8)         run_time
109: (12)             The duration of the animation in seconds.
110: (8)         Returns
111: (8)         -----
112: (8)         run_time
113: (12)             The duration of the animation in seconds.
114: (8)         """
115: (8)         self.build_animations_with_timings()
116: (8)         self.max_end_time = max(self.anims_with_timings["end"], default=0)
117: (8)         return self.max_end_time if run_time is None else run_time
118: (4)     def build_animations_with_timings(self) -> None:
119: (8)         """Creates a list of triplets of the form (anim, start_time,
end_time)."""
120: (8)         run_times = np.array([anim.run_time for anim in self.animations])
121: (8)         num_animations = run_times.shape[0]
122: (8)         dtype = [("anim", "O"), ("start", "f8"), ("end", "f8")]
123: (8)         self.anims_with_timings = np.zeros(num_animations, dtype=dtype)
124: (8)         self.anims_begun = np.zeros(num_animations, dtype=bool)
125: (8)         self.anims_finished = np.zeros(num_animations, dtype=bool)
126: (8)         if num_animations == 0:
127: (12)             return
128: (8)         lags = run_times[:-1] * self.lag_ratio
129: (8)         self.anims_with_timings["anim"] = self.animations

```

```

130: (8)                     self.anims_with_timings["start"][1:] = np.add.accumulate(lags)
131: (8)                     self.anims_with_timings["end"] = self.anims_with_timings["start"] +
run_times
132: (4)                     def interpolate(self, alpha: float) -> None:
133: (8)                         anim_group_time = self.rate_func(alpha) * self.max_end_time
134: (8)                         time_goes_back = anim_group_time < self.anim_group_time
135: (8)                         awt = self.anims_with_timings
136: (8)                         new_begun = anim_group_time >= awt["start"]
137: (8)                         new_finished = anim_group_time > awt["end"]
138: (8)                         to_update = awt[
139: (12)                             (self.anims_begun | new_begun) & (~self.anims_finished | ~new_finished)
140: (8)                         ]
141: (8)                         run_times = to_update["end"] - to_update["start"]
142: (8)                         sub_alphas = (anim_group_time - to_update["start"]) / run_times
143: (8)                         if time_goes_back:
144: (12)                             sub_alphas[sub_alphas < 0] = 0
145: (8)                         else:
146: (12)                             sub_alphas[sub_alphas > 1] = 1
147: (8)                         for anim_to_update, sub_alpha in zip(to_update["anim"], sub_alphas):
148: (12)                             anim_to_update.interpolate(sub_alpha)
149: (8)                             self.anim_group_time = anim_group_time
150: (8)                             self.anims_begun = new_begun
151: (8)                             self.anims_finished = new_finished
152: (0)                     class Succession(AnimationGroup):
153: (4)                         """Plays a series of animations in succession.
154: (4)                         Parameters
155: (4)                         -----
156: (4)                         animations
157: (8)                             Sequence of :class:`~.Animation` objects to be played.
158: (4)                         lag_ratio
159: (8)                             Defines the delay after which the animation is applied to subobjects.
A lag_ratio of
160: (8)                             ``n.nn`` means the next animation will play when ``nnn%`` of the
current animation has played.
161: (8)                             Defaults to 1.0, meaning that the next animation will begin when 100%
of the current
162: (8)                             animation has played.
163: (8)                             This does not influence the total runtime of the animation. Instead
the runtime
164: (8)                             of individual animations is adjusted so that the complete animation
has the defined
165: (8)                             run time.
166: (4)                     Examples
167: (4)                     -----
168: (4)                     .. manim:: SuccessionExample
169: (8)                     class SuccessionExample(Scene):
170: (12)                         def construct(self):
171: (16)                             dot1 = Dot(point=LEFT * 2 + UP * 2, radius=0.16, color=BLUE)
172: (16)                             dot2 = Dot(point=LEFT * 2 + DOWN * 2, radius=0.16,
color=MAROON)
173: (16)                             dot3 = Dot(point=RIGHT * 2 + DOWN * 2, radius=0.16,
color=GREEN)
174: (16)                             dot4 = Dot(point=RIGHT * 2 + UP * 2, radius=0.16,
color=YELLOW)
175: (16)
176: (16)
177: (20)                             self.add(dot1, dot2, dot3, dot4)
178: (20)                             self.play(Succession(
179: (20)                                 dot1.animate.move_to(dot2),
180: (20)                                 dot2.animate.move_to(dot3),
181: (16)                                 dot3.animate.move_to(dot4),
182: (4)                                 dot4.animate.move_to(dot1)
183: (4)                         ))
184: (8)                         """
185: (4)                     def __init__(self, *animations: Animation, lag_ratio: float = 1, **kwargs)
-> None:
186: (8)                         super().__init__(*animations, lag_ratio=lag_ratio, **kwargs)
187: (8)                         def begin(self) -> None:
188: (8)                             assert len(self.animations) > 0
189: (8)                             self.update_active_animation(0)

```

```

188: (4)             def finish(self) -> None:
189: (8)                 while self.active_animation is not None:
190: (12)                     self.next_animation()
191: (4)             def update_mobjects(self, dt: float) -> None:
192: (8)                 if self.active_animation:
193: (12)                     self.active_animation.update_mobjects(dt)
194: (4)             def _setup_scene(self, scene) -> None:
195: (8)                 if scene is None:
196: (12)                     return
197: (8)                 if self.is_introducer():
198: (12)                     for anim in self.animations:
199: (16)                         if not anim.is_introducer() and anim.mobject is not None:
200: (20)                             scene.add(anim.mobject)
201: (8)             self.scene = scene
202: (4)             def update_active_animation(self, index: int) -> None:
203: (8)                 self.active_index = index
204: (8)                 if index >= len(self.animations):
205: (12)                     self.active_animation: Animation | None = None
206: (12)                     self.active_start_time: float | None = None
207: (12)                     self.active_end_time: float | None = None
208: (8)                 else:
209: (12)                     self.active_animation = self.animations[index]
210: (12)                     self.active_animation._setup_scene(self.scene)
211: (12)                     self.active_animation.begin()
212: (12)                     self.active_start_time = self.anims_with_timings[index]["start"]
213: (12)                     self.active_end_time = self.anims_with_timings[index]["end"]
214: (4)             def next_animation(self) -> None:
215: (8)                 """Proceeds to the next animation.
216: (8)                 This method is called right when the active animation finishes.
217: (8)
218: (8)                 if self.active_animation is not None:
219: (12)                     self.active_animation.finish()
220: (8)                     self.update_active_animation(self.active_index + 1)
221: (4)             def interpolate(self, alpha: float) -> None:
222: (8)                 current_time = self.rate_func(alpha) * self.max_end_time
223: (8)                 while self.active_end_time is not None and current_time >=
self.active_end_time:
224: (12)
225: (8)             if self.active_animation is not None and self.active_start_time is not
None:
226: (12)                 elapsed = current_time - self.active_start_time
227: (12)                 active_run_time = self.active_animation.run_time
228: (12)                 subalpha = elapsed / active_run_time if active_run_time != 0.0
else 1.0
229: (12)                     self.active_animation.interpolate(subalpha)
230: (0)             class LaggedStart(AnimationGroup):
231: (4)                 """Adjusts the timing of a series of :class:`~.Animation` according to
``lag_ratio``.
232: (4)                 Parameters
233: (4)                 -----
234: (4)                 animations
235: (8)                     Sequence of :class:`~.Animation` objects to be played.
236: (4)                 lag_ratio
237: (8)                     Defines the delay after which the animation is applied to submobjects.
A lag_ratio of
238: (8)                         ``n.nn`` means the next animation will play when ``nnn%`` of the
current animation has played.
239: (8)                         Defaults to 0.05, meaning that the next animation will begin when 5%
of the current
240: (8)                         animation has played.
241: (8)                         This does not influence the total runtime of the animation. Instead
the runtime
242: (8)                         of individual animations is adjusted so that the complete animation
has the defined
243: (8)                         run time.
244: (4)                 Examples
245: (4)                 -----
246: (4)                     .. manim:: LaggedStartExample
247: (8)                         class LaggedStartExample(Scene):

```

```

248: (12)             def construct(self):
249: (16)                 title = Text("lag_ratio = 0.25").to_edge(UP)
250: (16)                 dot1 = Dot(point=LEFT * 2 + UP, radius=0.16)
251: (16)                 dot2 = Dot(point=LEFT * 2, radius=0.16)
252: (16)                 dot3 = Dot(point=LEFT * 2 + DOWN, radius=0.16)
253: (16)                 line_25 = DashedLine(
254: (20)                     start=LEFT + UP * 2,
255: (20)                     end=LEFT + DOWN * 2,
256: (20)                     color=RED
257: (16)                 )
258: (16)                 label = Text("25%", font_size=24).next_to(line_25, UP)
259: (16)                 self.add(title, dot1, dot2, dot3, line_25, label)
260: (16)                 self.play(LaggedStart(
261: (20)                     dot1.animate.shift(RIGHT * 4),
262: (20)                     dot2.animate.shift(RIGHT * 4),
263: (20)                     dot3.animate.shift(RIGHT * 4),
264: (20)                     lag_ratio=0.25,
265: (20)                     run_time=4
266: (16)                 ))
267: (4)             """
268: (4)             def __init__(
269: (8)                 self,
270: (8)                 *animations: Animation,
271: (8)                 lag_ratio: float = DEFAULT_LAGGED_START_LAG_RATIO,
272: (8)                 **kwargs,
273: (4)             ):
274: (8)                 super().__init__(*animations, lag_ratio=lag_ratio, **kwargs)
275: (0)             class LaggedStartMap(LaggedStart):
276: (4)                 """Plays a series of :class:`~.Animation` while mapping a function to
277: (4)                 subobjects.
278: (4)             Parameters
279: (4)             -----
280: (8)                 AnimationClass
281: (8)                     :class:`~.Animation` to apply to mobject.
282: (8)                 mobject
283: (8)                     :class:`~.Mobject` whose subobjects the animation, and optionally the
284: (4)                     function,
285: (8)                     are to be applied.
286: (4)                 arg_creator
287: (8)                     Function which will be applied to :class:`~.Mobject`.
288: (4)                 run_time
289: (4)                     The duration of the animation in seconds.
290: (4)             Examples
291: (8)             .. manim:: LaggedStartMapExample
292: (12)                 class LaggedStartMapExample(Scene):
293: (16)                     def construct(self):
294: (16)                         title = Tex("LaggedStartMap").to_edge(UP, buff=LARGE_BUFF)
295: (20)                         dots = VGroup(
296: (20)                             *[Dot(radius=0.16) for _ in range(35)]
297: (20)                             ).arrange_in_grid(rows=5, cols=7, buff=MED_LARGE_BUFF)
298: (16)                         self.add(dots, title)
299: (16)                         for mob in dots, title:
300: (20)                             self.play(LaggedStartMap(
301: (24)                                 ApplyMethod, mob,
302: (24)                                 lambda m : (m.set_color, YELLOW),
303: (24)                                 lag_ratio = 0.1,
304: (24)                                 rate_func = there_and_back,
305: (20)                                 run_time = 2
306: (4)                         ))
307: (4)             """
308: (4)             def __init__(
309: (8)                 self,
310: (8)                 AnimationClass: Callable[..., Animation],
311: (8)                 mobject: Mobject,
312: (8)                 arg_creator: Callable[[Mobject], str] = None,
313: (8)                 run_time: float = 2,
314: (8)                 **kwargs,
314: (4)             ) -> None:

```

```

315: (8)             args_list = []
316: (8)             for submob in mobject:
317: (12)                 if arg_creator:
318: (16)                     args_list.append(arg_creator(submob))
319: (12)                 else:
320: (16)                         args_list.append((submob,))
321: (8)             anim_kwarg = dict(kwarg)
322: (8)             if "lag_ratio" in anim_kwarg:
323: (12)                 anim_kwarg.pop("lag_ratio")
324: (8)             animations = [AnimationClass(*args, **anim_kwarg) for args in
325: (8)             args_list]
326: (8)             super().__init__(*animations, run_time=run_time, **kwarg)

-----

```

## File 14 - logger\_utils.py:

```

1: (0)         """Utilities to create and set the logger.
2: (0)         Manim's logger can be accessed as ``manim.logger`` , or as
3: (0)         ``logging.getLogger("manim")`` , once the library has been imported. Manim
also
4: (0)         exports a second object, ``console`` , which should be used to print on screen
5: (0)         messages that need not be logged.
6: (0)         Both ``logger`` and ``console`` use the ``rich`` library to produce rich text
7: (0)         format.
8: (0) """
9: (0)         from __future__ import annotations
10: (0)         import configparser
11: (0)         import copy
12: (0)         import json
13: (0)         import logging
14: (0)         from typing import TYPE_CHECKING
15: (0)         from rich import color, errors
16: (0)         from rich import print as printf
17: (0)         from rich.console import Console
18: (0)         from rich.logging import RichHandler
19: (0)         from rich.theme import Theme
20: (0)         if TYPE_CHECKING:
21: (4)             from pathlib import Path
22: (0)         __all__ = ["make_logger", "parse_theme", "set_file_logger", "JSONFormatter"]
23: (0)         HIGHLIGHTED_KEYWORDS = [ # these keywords are highlighted specially
24: (4)             "Played",
25: (4)             "animations",
26: (4)             "scene",
27: (4)             "Reading",
28: (4)             "Writing",
29: (4)             "script",
30: (4)             "arguments",
31: (4)             "Invalid",
32: (4)             "Aborting",
33: (4)             "module",
34: (4)             "File",
35: (4)             "Rendering",
36: (4)             "Rendered",
37: (0)         ]
38: (0)         WRONG_COLOR_CONFIG_MSG = """
39: (0)             [logging.level.error]Your colour configuration couldn't be parsed.
40: (0)             Loading the default color configuration.[/logging.level.error]
41: (0) """
42: (0)         def make_logger(
43: (4)             parser: configparser.SectionProxy,
44: (4)             verbosity: str,
45: (0)         ) -> tuple[logging.Logger, Console, Console]:
46: (4)             """Make the manim logger and console.
47: (4)             Parameters
48: (4)             -----
49: (4)             parser
50: (8)                 A parser containing any .cfg files in use.
51: (4)             verbosity

```

```

52: (8)                      The verbosity level of the logger.
53: (4)                      Returns
54: (4)
55: (4)                      :class:`logging.Logger`, :class:`rich.Console`, :class:`rich.Console`
56: (8)                      The manim logger and consoles. The first console outputs
57: (8)                      to stdout, the second to stderr. All use the theme returned by
58: (8)                      :func:`parse_theme`.
59: (4)                      See Also
60: (4)
61: (4)                      :func:`~._config.utils.make_config_parser`, :func:`parse_theme`
62: (4)                      Notes
63: (4)
64: (4)                      The ``parser`` is assumed to contain only the options related to
65: (4)                      configuring the logger at the top level.
66: (4)
67: (4)                      """
68: (4)                      theme = parse_theme(parser)
69: (4)                      console = Console(theme=theme)
70: (4)                      error_console = Console(theme=theme, stderr=True)
71: (8)                      rich_handler = RichHandler(
72: (8)                          console=console,
73: (8)                          show_time=parser.getboolean("log_timestamps"),
74: (8)                          keywords=HIGHLIGHTED_KEYWORDS,
75: (4)                      )
76: (4)                      logger = logging.getLogger("manim")
77: (4)                      logger.addHandler(rich_handler)
78: (4)                      logger.setLevel(verbosity)
79: (4)                      return logger, console, error_console
80: (0)                  def parse_theme(parser: configparser.SectionProxy) -> Theme:
81: (4)                      """Configure the rich style of logger and console output.
82: (4)                      Parameters
83: (4)
84: (4)                      parser
85: (4)                          A parser containing any .cfg files in use.
86: (4)                      Returns
87: (4)
88: (8)                      :class:`rich.Theme`
89: (4)                          The rich theme to be used by the manim logger.
90: (4)                      See Also
91: (4)
92: (4)
93: (4)                      :func:`make_logger`.
94: (4)
95: (4)                      theme = {key.replace("_", "."): parser[key] for key in parser}
96: (4)                      theme["log.width"] = None if theme["log.width"] == "-1" else
int(theme["log.width"])
97: (4)                      theme["log.height"] = (
98: (8)                          None if theme["log.height"] == "-1" else int(theme["log.height"]))
99: (4)                      theme["log.timestamps"] = False
try:
100: (8)                          custom_theme = Theme(
101: (12)                            {
102: (16)                              k: v
103: (16)                              for k, v in theme.items()
104: (16)                              if k not in ["log.width", "log.height", "log.timestamps"]
105: (12)                            },
106: (8)                          )
except (color.ColorParseError, errors.StyleSyntaxError):
107: (4)                            printf(WRONG_COLOR_CONFIG_MSG)
108: (8)                            custom_theme = None
109: (8)
110: (4)                            return custom_theme
111: (0)                  def set_file_logger(scene_name: str, module_name: str, log_dir: Path) -> None:
112: (4)                      """Add a file handler to manim logger.
113: (4)                      The path to the file is built using ``config.log_dir``.
114: (4)                      Parameters
115: (4)
116: (4)                      scene_name
117: (8)                          The name of the scene, used in the name of the log file.
118: (4)                      module_name
119: (8)                          The name of the module, used in the name of the log file.

```

```

120: (4)             log_dir
121: (8)             Path to the folder where log files are stored.
122: (4)
123: (4)             """
124: (4)             log_file_name = f"{module_name}_{scene_name}.log"
125: (4)             log_file_path = log_dir / log_file_name
126: (4)             file_handler = logging.FileHandler(log_file_path, mode="w")
127: (4)             file_handler.setFormatter(JSONFormatter())
128: (4)             logger = logging.getLogger("manim")
129: (4)             logger.addHandler(file_handler)
130: (0)             logger.info("Log file will be saved in %(logpath)s", {"logpath":
131: (4)             log_file_path})
132: (4)
133: (4)
134: (4)             class JSONFormatter(logging.Formatter):
135: (8)             """A formatter that outputs logs in a custom JSON format.
136: (8)             This class is used internally for testing purposes.
137: (8)
138: (12)             def format(self, record: logging.LogRecord) -> str:
139: (16)             """Format the record in a custom JSON format."""
140: (8)             record_c = copy.deepcopy(record)
141: (12)             if record_c.args:
142: (16)                 for arg in record_c.args:
143: (16)                     record_c.args[arg] = "<>"
144: (16)             return json.dumps(
145: (12)                 {
146: (8)                     "levelname": record_c.levelname,
147: (16)                     "module": record_c.module,
148: (16)                     "message": super().format(record_c),
149: (12)                 },
150: (8)             )
-----
```

## File 15 - group.py:

```

1: (0)             """Manim's cfg subcommand.
2: (0)             Manim's cfg subcommand is accessed in the command-line interface via ``manim
3: (0)             cfg``. Here you can specify options, subcommands, and subgroups for the cfg
4: (0)             group.
5: (0)
6: (0)             from __future__ import annotations
7: (0)             from ast import literal_eval
8: (0)             from pathlib import Path
9: (0)             import cloup
10: (0)            from rich.errors import StyleSyntaxError
11: (0)            from rich.style import Style
12: (0)            from ... import cli_ctx_settings, console
13: (0)            from ..._config.utils import config_file_paths, make_config_parser
14: (0)            from ...constants import EPILOG
15: (0)            from ...utils.file_ops import guarantee_existence, open_file
16: (0)            RICH_COLOUR_INSTRUCTIONS: str = """
17: (0)                [red]The default colour is used by the input statement.
18: (0)                If left empty, the default colour will be used.[/red]
19: (0)                [magenta] For a full list of styles, visit[/magenta]
20: (0)                [green]https://rich.readthedocs.io/en/latest/style.html[/green]
21: (0)
22: (0)            RICH_NON_STYLE_ENTRIES: str = ["log.width", "log.height", "log.timestamps"]
23: (0)            __all__ = [
24: (4)                "value_from_string",
25: (4)                "is_valid_style",
26: (4)                "replace_keys",
27: (4)                "cfg",
28: (4)                "write",
29: (4)                "show",
30: (4)                "export",
31: (0)            ]
32: (0)            def value_from_string(value: str) -> str | int | bool:
33: (4)                """Extracts the literal of proper datatype from a string.
34: (4)                Parameters
35: (4)                -----
```

```

36: (4)           value
37: (8)             The value to check get the literal from.
38: (4)           Returns
39: (4)
40: (4)           -----
41: (8)             Union[:class:`str`, :class:`int`, :class:`bool`]
42: (4)             Returns the literal of appropriate datatype.
43: (4)
44: (4)           """
45: (4)             try:
46: (8)               value = literal_eval(value)
47: (4)             except (SyntaxError, ValueError):
48: (8)               pass
49: (4)             return value
50: (0)           def _is_expected_datatype(value: str, expected: str, style: bool = False) ->
51: (4)             """
52: (4)               Checks whether `value` is the same datatype as `expected`,
53: (4)               and checks if it is a valid `style` if `style` is true.
54: (4)             Parameters
55: (4)             -----
56: (4)             value
57: (8)               The string of the value to check (obtained from reading the user
58: (4)               input).
59: (4)             expected
60: (8)               The string of the literal datatype must be matched by `value`.
61: (4)               reading the cfg file.
62: (4)             style
63: (8)               Whether or not to confirm if `value` is a style, by default False
64: (4)             Returns
65: (4)             -----
66: (4)             :class:`bool`
67: (8)               Whether or not `value` matches the datatype of `expected`.
68: (4)             """
69: (4)             value = value_from_string(value)
70: (4)             expected = type(value_from_string(expected))
71: (4)             return isinstance(value, expected) and (is_valid_style(value) if style
72: (4)             else True)
73: (0)           def is_valid_style(style: str) -> bool:
74: (4)             """
75: (4)               Checks whether the entered color is a valid color according to rich
76: (4)             Parameters
77: (4)             -----
78: (4)             style
79: (8)               The style to check whether it is valid.
80: (4)             Returns
81: (4)             -----
82: (4)             Boolean
83: (8)               Returns whether it is valid style or not according to rich.
84: (4)             """
85: (4)             try:
86: (8)               Style.parse(style)
87: (4)               return True
88: (4)             except StyleSyntaxError:
89: (8)               return False
90: (0)           def replace_keys(default: dict) -> dict:
91: (4)             """
92: (4)               Replaces _ to . and vice versa in a dictionary for rich
93: (4)             Parameters
94: (4)             -----
95: (4)             default
96: (8)               The dictionary to check and replace
97: (4)             Returns
98: (4)             -----
99: (4)             :class:`dict`
100: (8)            The dictionary which is modified by replacing _ with . and vice versa
    """
    for key in default:
        if "_" in key:
            temp = default[key]
            del default[key]
            key = key.replace("_", ".")
            default[key] = temp

```

```

101: (8)             else:
102: (12)             temp = default[key]
103: (12)             del default[key]
104: (12)             key = key.replace(".", "_")
105: (12)             default[key] = temp
106: (4)             return default
107: (0)         @cloop.group(
108: (4)             context_settings=cli_ctx_settings,
109: (4)             invoke_without_command=True,
110: (4)             no_args_is_help=True,
111: (4)             epilog=EPILOG,
112: (4)             help="Manim configuration files.",
113: (0)
114: (0)         @cloop.pass_context
115: (0)     def cfg(ctx):
116: (4)         """Responsible for the cfg subcommand."""
117: (4)         pass
118: (0)     @cfg.command(context_settings=cli_ctx_settings, no_args_is_help=True)
119: (0)     @cloop.option(
120: (4)         "-l",
121: (4)         "--level",
122: (4)         type=cloop.Choice(["user", "cwd"], case_sensitive=False),
123: (4)         default="cwd",
124: (4)         help="Specify if this config is for user or the working directory.",
125: (0)
126: (0)     @cloop.option("-o", "--open", "openfile", is_flag=True)
127: (0)     def write(level: str = None, openfile: bool = False) -> None:
128: (4)         config_paths = config_file_paths()
129: (4)         console.print(
130: (8)             "[yellow bold]Manim Configuration File Writer[/yellow bold]",
131: (8)             justify="center",
132: (4)         )
133: (4)         USER_CONFIG_MSG = f"""A configuration file at [yellow]{config_paths[1]}
134: (0) has been created with your required changes.
This will be used when running the manim command. If you want to override this
config,
135: (0)     you will have to create a manim.cfg in the local directory, where you want
those changes to be overridden."""
136: (4)     CWD_CONFIG_MSG = f"""A configuration file at [yellow]{config_paths[2]}
137: (0) has been created.
To save your config please save that file and place it in your current working
directory, from where you run the manim command."""
138: (4)     parser = make_config_parser()
139: (4)     if not openfile:
140: (8)         action = "save this as"
141: (8)         for category in parser:
142: (12)             console.print(f"{category}", style="bold green underline")
143: (12)             default = parser[category]
144: (12)             if category == "logger":
145: (16)                 console.print(RICH_COLOUR_INSTRUCTIONS)
146: (16)                 default = replace_keys(default)
147: (12)                 for key in default:
148: (16)                     if category == "logger" and key not in RICH_NON_STYLE_ENTRIES:
149: (20)                         desc = "style"
150: (20)                         style = default[key]
151: (16)                     else:
152: (20)                         desc = "value"
153: (20)                         style = None
154: (16)                     console.print(f"Enter the {desc} for {key} ", style=style,
155: (16) end="")
156: (20)
157: (24)             if category != "logger" or key in RICH_NON_STYLE_ENTRIES:
158: (24)                 defaultval = (
159: (24)                     repr(default[key])
160: (20)                     if isinstance(value_from_string(default[key]), str)
161: (20)                     else default[key]
162: (16)                 )
163: (20)                 console.print(f"(defaults to {defaultval}) :", end="")
try:
    temp = input()

```

```

164: (16)             except EOFError:
165: (20)                 raise Exception(
166: (24)                     """Not enough values in input.
167: (0)             You may have added a new entry to default.cfg, in which case you will have to
168: (0)             modify write_cfg_subcmd_input to account for it."""",
169: (20)                     )
170: (16)             if temp:
171: (20)                 while temp and not _is_expected_datatype(
172: (24)                     temp,
173: (24)                     default[key],
174: (24)                     bool(style),
175: (20)                 ):
176: (24)                     console.print(
177: (28)                         f"[red bold]Invalid {desc}. Try again.[/red"
bold]",
178: (24)
179: (24)
180: (28)                     )
181: (28)                     console.print(
182: (28)                         f"Enter the {desc} for {key}:",
style=style,
end="",
)
183: (24)                     temp = input()
184: (24)                     default[key] = temp.replace("%", "%")
185: (20)             default = replace_keys(default) if category == "logger" else
186: (12)             default
187: (12)             parser[category] = {
188: (16)                 i: v.replace("%", "%") for i, v in dict(default).items()
189: (12)             }
190: (4)         else:
191: (8)             action = "open"
192: (4)         if level is None:
193: (8)             console.print(
194: (12)                 f"Do you want to {action} the default config for this User?(y/n"
[[n]],
195: (12)
196: (12)                 style="dim purple",
end="",
)
197: (8)
198: (8)                 action_to_userpath = input()
199: (4)             else:
200: (8)                 action_to_userpath = ""
201: (4)             if action_to_userpath.lower() == "y" or level == "user":
202: (8)                 cfg_file_path = config_paths[1]
203: (8)                 guarantee_existence(config_paths[1].parents[0])
204: (8)                 console.print(USER_CONFIG_MSG)
205: (4)             else:
206: (8)                 cfg_file_path = config_paths[2]
207: (8)                 guarantee_existence(config_paths[2].parents[0])
208: (8)                 console.print(CWD_CONFIG_MSG)
209: (4)             with cfg_file_path.open("w") as fp:
210: (8)                 parser.write(fp)
211: (4)             if openfile:
212: (8)                 open_file(cfg_file_path)
213: (0)             @cfg.command(context_settings=cli_ctx_settings)
214: (0)             def show():
215: (4)                 parser = make_config_parser()
216: (4)                 rich_non_style_entries = [a.replace(".", "_") for a in
RICH_NON_STYLE_ENTRIES]
217: (4)                 for category in parser:
218: (8)                     console.print(f"{category}", style="bold green underline")
219: (8)                     for entry in parser[category]:
220: (12)                         if category == "logger" and entry not in rich_non_style_entries:
221: (16)                             console.print(f"{entry} :", end="")
222: (16)                             console.print(
223: (20)                                 f" {parser[category][entry]}",
style=parser[category][entry],
)
224: (20)                         else:
225: (16)                             console.print(f"{entry} : {parser[category][entry]}")
226: (12)                         console.print("\n")
227: (16)
228: (8)

```

```

229: (0)          @cfg.command(context_settings=cli_ctx_settings)
230: (0)          @cloop.option("-d", "--directory", default=Path.cwd())
231: (0)          @cloop.pass_context
232: (0)          def export(ctx, directory):
233: (4)            directory_path = Path(directory)
234: (4)            if directory_path.absolute == Path.cwd().absolute:
235: (8)              console.print(
236: (12)                """You are reading the config from the same directory you are
237: (0)      exporting to.
238: (0)      This means that the exported config will overwrite the config for this
239: (12)      directory.
240: (12)      Are you sure you want to continue? (y/n)""",
241: (8)                  style="red bold",
242: (8)                  end="",
243: (4)                  )
244: (8)                  proceed = input().lower() == "y"
245: (4)            else:
246: (8)              proceed = True
247: (4)            if proceed:
248: (8)              if not directory_path.is_dir():
249: (8)                console.print(f"Creating folder: {directory}.", style="red bold")
250: (8)                directory_path.mkdir(parents=True)
251: (8)                ctx.invoke(write)
252: (8)                from_path = Path.cwd() / "manim.cfg"
253: (8)                to_path = directory_path / "manim.cfg"
254: (8)                console.print(f"Exported final Config at {from_path} to {to_path}.")
255: (4)            else:
256: (8)              console.print("Aborted...", style="red bold")

```

-----

## File 16 - update.py:

```

1: (0)          """Animations that update mobjects."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = ["UpdateFromFunc", "UpdateFromAlphaFunc",
4: (0)          "MaintainPositionRelativeTo"]
5: (0)          import operator as op
6: (0)          import typing
7: (0)          from manim.animation.animation import Animation
8: (0)          if typing.TYPE_CHECKING:
9: (4)            from manim.mobject.mobject import Mobject
10: (0)          class UpdateFromFunc(Animation):
11: (4)            """
12: (4)              update_function of the form func(mobject), presumably
13: (4)              to be used when the state of one mobject is dependent
14: (4)              on another simultaneously animated mobject
15: (4)            """
16: (8)            def __init__(
17: (8)              self,
18: (8)              mobject: Mobject,
19: (8)              update_function: typing.Callable[[Mobject], typing.Any],
20: (8)              suspend_mobject_updating: bool = False,
21: (8)              **kwargs,
22: (4)            ) -> None:
23: (8)              self.update_function = update_function
24: (8)              super().__init__(
25: (8)                mobject, suspend_mobject_updating=suspend_mobject_updating,
26: (8)                )
27: (8)              def interpolate_mobject(self, alpha: float) -> None:
28: (8)                self.update_function(self.mobject)
29: (0)          class UpdateFromAlphaFunc(UpdateFromFunc):
30: (4)            def interpolate_mobject(self, alpha: float) -> None:
31: (8)              self.update_function(self.mobject, self.rate_func(alpha))
32: (4)          class MaintainPositionRelativeTo(Animation):
33: (0)            def __init__(self, mobject: Mobject, tracked_mobject: Mobject, **kwargs) -
34: (8)              self.tracked_mobject = tracked_mobject

```

```

34: (8)             self.diff = op.sub(
35: (12)           mobject.get_center(),
36: (12)           tracked_mobject.get_center(),
37: (8)         )
38: (8)         super().__init__(mobject, **kwargs)
39: (4)     def interpolate_mobject(self, alpha: float) -> None:
40: (8)         target = self.tracked_mobject.get_center()
41: (8)         location = self.mobject.get_center()
42: (8)         self.mobject.shift(target - location + self.diff)

```

---

## File 17 - camera.py:

```

1: (0)             """A camera converts the mobjects contained in a Scene into an array of
pixels."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["Camera", "BackgroundColoredVMobjectDisplayer"]
4: (0)             import copy
5: (0)             import itertools as it
6: (0)             import operator as op
7: (0)             import pathlib
8: (0)             from functools import reduce
9: (0)             from typing import Any, Callable, Iterable
10: (0)            import cairo
11: (0)            import numpy as np
12: (0)            from PIL import Image
13: (0)            from scipy.spatial.distance import pdist
14: (0)            from .. import config, logger
15: (0)            from ..constants import *
16: (0)            from ..mobject.mobject import Mobject
17: (0)            from ..mobject.types.image_mobject import AbstractImageMobject
18: (0)            from ..mobject.types.point_cloud_mobject import PMobject
19: (0)            from ..mobject.types.vectorized_mobject import VMobject
20: (0)            from ..utils.color import ManimColor, ParsableManimColor, color_to_int_rgba
21: (0)            from ..utils.family import extract_mobject_family_members
22: (0)            from ..utils.images import get_full_raster_image_path
23: (0)            from ..utils.iterables import list_difference_update
24: (0)            from ..utils.space_ops import angle_of_vector
25: (0)            LINE_JOIN_MAP = {
26: (4)                LineJointType.AUTO: None, # TODO: this could be improved
27: (4)                LineJointType.ROUND: cairo.LineJoin.ROUND,
28: (4)                LineJointType.BEVEL: cairo.LineJoin.BEVEL,
29: (4)                LineJointType.MITER: cairo.LineJoin.MITER,
30: (0)            }
31: (0)            CAP_STYLE_MAP = {
32: (4)                CapStyleType.AUTO: None, # TODO: this could be improved
33: (4)                CapStyleType.ROUND: cairo.LineCap.ROUND,
34: (4)                CapStyleType.BUTT: cairo.LineCap.BUTT,
35: (4)                CapStyleType.SQUARE: cairo.LineCap.SQUARE,
36: (0)            }
37: (0)            class Camera:
38: (4)                """Base camera class.
39: (4)                This is the object which takes care of what exactly is displayed
40: (4)                on screen at any given moment.
41: (4)                Parameters
42: (4)                -----
43: (4)                background_image
44: (8)                    The path to an image that should be the background image.
45: (8)                    If not set, the background is filled with
:attr:`self.background_color`
46: (4)                background
47: (8)                    What :attr:`background` is set to. By default, ``None``.
48: (4)                pixel_height
49: (8)                    The height of the scene in pixels.
50: (4)                pixel_width
51: (8)                    The width of the scene in pixels.
52: (4)                kwargs
53: (8)                    Additional arguments (``background_color``, ``background_opacity``)

```

```

54: (8)          to be set.
55: (4)
56: (4)
57: (8)
58: (8)
59: (8)      def __init__(
60: (8)          self,
61: (8)          background_image: str | None = None,
62: (8)          frame_center: np.ndarray = ORIGIN,
63: (8)          image_mode: str = "RGBA",
64: (8)          n_channels: int = 4,
65: (8)          pixel_array_dtype: str = "uint8",
66: (8)          cairo_line_width_multiple: float = 0.01,
67: (8)          use_z_index: bool = True,
68: (8)          background: np.ndarray | None = None,
69: (8)          pixel_height: int | None = None,
70: (8)          pixel_width: int | None = None,
71: (8)          frame_height: float | None = None,
72: (8)          frame_width: float | None = None,
73: (8)          frame_rate: float | None = None,
74: (4)          background_color: ParsableManimColor | None = None,
75: (8)          background_opacity: float | None = None,
76: (8)          **kwargs,
77: (8)      ):
78: (8)          self.background_image = background_image
79: (8)          self.frame_center = frame_center
80: (8)          self.image_mode = image_mode
81: (8)          self.n_channels = n_channels
82: (8)          self.pixel_array_dtype = pixel_array_dtype
83: (8)          self.cairo_line_width_multiple = cairo_line_width_multiple
84: (12)         self.use_z_index = use_z_index
85: (8)         self.background = background
86: (8)         if pixel_height is None:
87: (12)             pixel_height = config["pixel_height"]
88: (8)         self.pixel_height = pixel_height
89: (8)         if pixel_width is None:
90: (12)             pixel_width = config["pixel_width"]
91: (8)         self.pixel_width = pixel_width
92: (8)         if frame_height is None:
93: (12)             frame_height = config["frame_height"]
94: (8)         self.frame_height = frame_height
95: (8)         if frame_width is None:
96: (12)             frame_width = config["frame_width"]
97: (8)         self.frame_width = frame_width
98: (8)         if frame_rate is None:
99: (12)             frame_rate = config["frame_rate"]
100: (8)         self.frame_rate = frame_rate
101: (12)        if background_color is None:
102: (8)             self._background_color =
103: (12)             ManimColor.parse(config["background_color"]))
104: (8)             else:
105: (12)                 self._background_color = ManimColor.parse(background_color)
106: (8)             if background_opacity is None:
107: (8)                 self._background_opacity = config["background_opacity"]
108: (8)             else:
109: (8)                 self._background_opacity = background_opacity
110: (8)             self.max_allowable_norm = config["frame_width"]
111: (8)             self.rgb_max_val = np.iinfo(self.pixel_array_dtype).max
112: (4)             self.pixel_array_to_cairo_context = {}
113: (8)             self.init_background()
114: (8)             self.resize_frame_shape()
115: (4)             self.reset()
116: (4)             def __deepcopy__(self, memo):
117: (8)                 self.canvas = None
118: (8)                 return copy.copy(self)
119: (4)             @property
120: (8)             def background_color(self):
121: (8)                 return self._background_color
122: (4)             @background_color.setter
123: (8)             def background_color(self, color):
124: (8)                 self._background_color = color
125: (8)                 self.init_background()

```

```

122: (4) @property
123: (4)     def background_opacity(self):
124: (8)         return self._background_opacity
125: (4)     @background_opacity.setter
126: (4)     def background_opacity(self, alpha):
127: (8)         self._background_opacity = alpha
128: (8)         self.init_background()
129: (4)     def type_or_raise(self, mobject: Mobject):
130: (8)         """Return the type of mobject, if it is a type that can be rendered.
131: (8)         If `mobject` is an instance of a class that inherits from a class that
132: (8)         can be rendered, return the super class. For example, an instance of
133: (8)         a
134: (8)         Square is also an instance of VMobject, and these can be rendered.
135: (8)         Therefore, `type_or_raise(Square())` returns True.
136: (8)         Parameters
137: (8)             -----
138: (12)             mobject
139: (8)                 The object to take the type of.
140: (8)         Notes
141: (8)             -----
142: (8)             For a list of classes that can currently be rendered, see
143: (8)             :meth:`display_funcs`.
144: (8)         Returns
145: (8)             -----
146: (12)             Type[:class:`~.Mobject`]
147: (8)                 The type of mobjects, if it can be rendered.
148: (8)             Raises
149: (8)             -----
150: (12)             :exc:`TypeError`
151: (8)                 When mobject is not an instance of a class that can be rendered.
152: (8)
153: (12)             self.display_funcs = {
154: (12)                 VMobject: self.display_multiple_vectorized_mobjects,
155: (12)                 PMobject: self.display_multiple_point_cloud_mobjects,
156: (12)                 AbstractImageMobject: self.display_multiple_image_mobjects,
157: (12)                 Mobject: lambda batch, pa: batch, # Do nothing
158: (12)
159: (16)
160: (8)             raise TypeError(f"Displaying an object of class {_type} is not
161: (8)             supported")
162: (8)
163: (8)             def reset_pixel_shape(self, new_height: float, new_width: float):
164: (8)                 """This method resets the height and width
165: (8)                     of a single pixel to the passed new_height and new_width.
166: (8)                     Parameters
167: (8)                         -----
168: (8)                         new_height
169: (12)                             The new height of the entire scene in pixels
170: (8)                         new_width
171: (12)                             The new width of the entire scene in pixels
172: (8)
173: (8)
174: (8)
175: (8)
176: (4)             def resize_frame_shape(self, fixed_dimension: int = 0):
177: (8)
178: (8)
179: (8)
180: (8)
181: (8)
182: (8)
183: (8)
184: (8)
185: (12)
186: (12)
187: (8)

```

```

188: (8)           pixel_height = self.pixel_height
189: (8)           pixel_width = self.pixel_width
190: (8)           frame_height = self.frame_height
191: (8)           frame_width = self.frame_width
192: (8)           aspect_ratio = pixel_width / pixel_height
193: (8)           if fixed_dimension == 0:
194: (12)             frame_height = frame_width / aspect_ratio
195: (8)           else:
196: (12)             frame_width = aspect_ratio * frame_height
197: (8)           self.frame_height = frame_height
198: (8)           self.frame_width = frame_width
199: (4)           def init_background(self):
200: (8)             """Initialize the background.
201: (8)             If self.background_image is the path of an image
202: (8)             the image is set as background; else, the default
203: (8)             background color fills the background.
204: (8)
205: (8)             height = self.pixel_height
206: (8)             width = self.pixel_width
207: (8)             if self.background_image is not None:
208: (12)               path = get_full_raster_image_path(self.background_image)
209: (12)               image = Image.open(path).convert(self.image_mode)
210: (12)               self.background = np.array(image)[:height, :width]
211: (12)               self.background = self.background.astype(self.pixel_array_dtype)
212: (8)             else:
213: (12)               background_rgba = color_to_int_rgba(
214: (16)                 self.background_color,
215: (16)                 self.background_opacity,
216: (12)               )
217: (12)               self.background = np.zeros(
218: (16)                 (height, width, self.n_channels),
219: (16)                 dtype=self.pixel_array_dtype,
220: (12)               )
221: (12)               self.background[:, :] = background_rgba
222: (4)           def get_image(self, pixel_array: np.ndarray | list | tuple | None = None):
223: (8)             """Returns an image from the passed
224: (8)             pixel array, or from the current frame
225: (8)             if the passed pixel array is none.
226: (8)             Parameters
227: (8)             -----
228: (8)             pixel_array
229: (12)               The pixel array from which to get an image, by default None
230: (8)             Returns
231: (8)             -----
232: (8)             PIL.Image
233: (12)               The PIL image of the array.
234: (8)
235: (8)             if pixel_array is None:
236: (12)               pixel_array = self.pixel_array
237: (8)             return Image.fromarray(pixel_array, mode=self.image_mode)
238: (4)           def convert_pixel_array(
239: (8)               self, pixel_array: np.ndarray | list | tuple, convert_from_floats:
240: (4)               ):
241: (8)                 """Converts a pixel array from values that have floats in them
242: (8)                 to proper RGB values.
243: (8)                 Parameters
244: (8)                 -----
245: (8)                 pixel_array
246: (12)                   Pixel array to convert.
247: (8)                 convert_from_floats
248: (12)                   Whether or not to convert float values to ints, by default False
249: (8)                 Returns
250: (8)                 -----
251: (8)                 np.array
252: (12)                   The new, converted pixel array.
253: (8)
254: (8)                 retval = np.array(pixel_array)
255: (8)                 if convert_from_floats:

```

```

256: (12)             retval = np.apply_along_axis(
257: (16)                 lambda f: (f *
self.rgb_max_val).astype(self.pixel_array_dtype),
258: (16)                     2,
259: (16)                     retval,
260: (12)             )
261: (8)             return retval
262: (4)             def set_pixel_array(
263: (8)                 self, pixel_array: np.ndarray | list | tuple, convert_from_floats:
bool = False
264: (4)             ):
265: (8)                 """Sets the pixel array of the camera to the passed pixel array.
Parameters
-----
pixel_array
    The pixel array to convert and then set as the camera's pixel
array.
270: (8)
271: (12)             convert_from_floats
default False
272: (8)
273: (8)             convert_from_floats)
274: (8)
275: (12)             converted_array = self.convert_pixel_array(pixel_array,
276: (12)                 if not (
277: (8)                     hasattr(self, "pixel_array")
278: (12)                     and self.pixel_array.shape == converted_array.shape
279: (8)                 ):
280: (12)                     self.pixel_array = converted_array
else:
281: (4)                     self.pixel_array[:, :, :] = converted_array[:, :, :]
def set_background(
282: (8)                 self, pixel_array: np.ndarray | list | tuple, convert_from_floats:
bool = False
283: (4)             ):
284: (8)                 """Sets the background to the passed pixel_array after converting
285: (8)                 to valid RGB values.
Parameters
-----
pixel_array
    The pixel array to set the background to.
convert_from_floats
    Whether or not to convert floats values to proper RGB valid ones,
by default False
292: (8)
293: (8)             convert_from_floats)
294: (4)             self.background = self.convert_pixel_array(pixel_array,
295: (8)             def make_background_from_func(
296: (4)                 self, coords_to_colors_func: Callable[[np.ndarray], np.ndarray]
297: (8)             ):
298: (8)                 """Makes a pixel array for the background by using coords_to_colors_func
to determine each pixel's color. Each input
299: (8)                 pixel's color. Each input to coords_to_colors_func is an (x, y) pair
in space (in ordinary space coordinates; not
300: (8)                 pixel coordinates), and each output is expected to be an RGBA array of
4 floats.
301: (8)             Parameters
302: (8)             -----
303: (8)             coords_to_colors_func
304: (12)                 The function whose input is an (x,y) pair of coordinates and
305: (12)                 whose return values must be the colors for that point
306: (8)             Returns
307: (8)             -----
308: (8)             np.array
309: (12)                 The pixel array which can then be passed to set_background.
310: (8)             """
311: (8)             logger.info("Starting set_background")
312: (8)             coords = self.get_coords_of_all_pixels()
313: (8)             new_background = np.apply_along_axis(coords_to_colors_func, 2, coords)

```

```

314: (8)                 logger.info("Ending set_background")
315: (8)                 return self.convert_pixel_array(new_background,
convert_from_floats=True)
316: (4)                 def set_background_from_func(
317: (8)                     self, coords_to_colors_func: Callable[[np.ndarray], np.ndarray]
318: (4)                 ):
319: (8)                     """
320: (8)                         Sets the background to a pixel array using coords_to_colors_func to
determine each pixel's color. Each input
321: (8)                             pixel's color. Each input to coords_to_colors_func is an (x, y) pair
in space (in ordinary space coordinates; not
322: (8)                             pixel coordinates), and each output is expected to be an RGBA array of
4 floats.
323: (8)             Parameters
324: (8)             -----
325: (8)             coords_to_colors_func
326: (12)                 The function whose input is an (x,y) pair of coordinates and
327: (12)                 whose return values must be the colors for that point
328: (8)             """
329: (8)
self.set_background(self.make_background_from_func(coords_to_colors_func))
330: (4)             def reset(self):
331: (8)                 """Resets the camera's pixel array
332: (8)                     to that of the background
333: (8)             Returns
334: (8)             -----
335: (8)             Camera
336: (12)                 The camera object after setting the pixel array.
337: (8)             """
338: (8)             self.set_pixel_array(self.background)
339: (8)             return self
340: (4)             def set_frame_to_background(self, background):
341: (8)                 self.set_pixel_array(background)
342: (4)             def get_mobjects_to_display(
343: (8)                 self,
344: (8)                 mobjects: Iterable[Mobject],
345: (8)                 include_submobjects: bool = True,
346: (8)                 excluded_mobjects: list | None = None,
347: (4)             ):
348: (8)                 """Used to get the list of mobjects to display
349: (8)                     with the camera.
350: (8)             Parameters
351: (8)             -----
352: (8)             mobjects
353: (12)                 The Mobjects
354: (8)             include_submobjects
355: (12)                 Whether or not to include the submobjects of mobjects, by default
True
356: (8)             excluded_mobjects
357: (12)                 Any mobjects to exclude, by default None
358: (8)             Returns
359: (8)             -----
360: (8)             list
361: (12)                 list of mobjects
362: (8)             """
363: (8)             if include_submobjects:
364: (12)                 mobjects = extract_mobject_family_members(
365: (16)                     mobjects,
366: (16)                     use_z_index=self.use_z_index,
367: (16)                     only_those_with_points=True,
368: (12)                 )
369: (12)                 if excluded_mobjects:
370: (16)                     all_excluded = extract_mobject_family_members(
371: (20)                         excluded_mobjects,
372: (20)                         use_z_index=self.use_z_index,
373: (16)                     )
374: (16)                     mobjects = list_difference_update(mobjects, all_excluded)
375: (8)             return list(mobjects)
376: (4)             def is_in_frame(self, mobject: Mobject):

```

```

377: (8)             """Checks whether the passed mobject is in
378: (8)             frame or not.
379: (8)             Parameters
380: (8)             -----
381: (8)             mobject
382: (12)            The mobject for which the checking needs to be done.
383: (8)             Returns
384: (8)             -----
385: (8)             bool
386: (12)            True if in frame, False otherwise.
387: (8)             """
388: (8)             fc = self.frame_center
389: (8)             fh = self.frame_height
390: (8)             fw = self.frame_width
391: (8)             return not reduce(
392: (12)             op.or_,
393: (12)             [
394: (16)               mobject.get_right()[0] < fc[0] - fw / 2,
395: (16)               mobject.get_bottom()[1] > fc[1] + fh / 2,
396: (16)               mobject.get_left()[0] > fc[0] + fw / 2,
397: (16)               mobject.get_top()[1] < fc[1] - fh / 2,
398: (12)             ],
399: (8)
400: (4)             )
401: (8)             def capture_mobject(self, mobject: Mobject, **kwargs: Any):
402: (8)                 """Capture mobjects by storing it in :attr:`pixel_array`.
403: (8)                 This is a single-mobject version of :meth:`capture_mobjects`.
404: (8)                 Parameters
405: (8)                 -----
406: (12)                 mobject
407: (8)                     Mobject to capture.
408: (12)                 kwargs
409: (8)                     Keyword arguments to be passed to :meth:`get_mobjects_to_display`.
410: (8)                     """
411: (4)             return self.capture_mobjects([mobject], **kwargs)
412: (8)             def capture_mobjects(self, mobjects: Iterable[Mobject], **kwargs):
413: (8)                 """Capture mobjects by printing them on :attr:`pixel_array`.
414: (8)                 This is the essential function that converts the contents of a Scene
415: (8)                 into an array, which is then converted to an image or video.
416: (8)                 Parameters
417: (8)                 -----
418: (12)                 mobjects
419: (8)                     Mobjects to capture.
420: (12)                 kwargs
421: (8)                     Keyword arguments to be passed to :meth:`get_mobjects_to_display`.
422: (8)                     Notes
423: (8)                     -----
424: (8)                     For a list of classes that can currently be rendered, see
425: (8)                     """
426: (8)                     mobjects = self.get_mobjects_to_display(mobjects, **kwargs)
427: (12)                     for group_type, group in it.groupby(mobjects, self.type_or_raise):
428: (4)                         self.display_funcs[group_type](list(group), self.pixel_array)
429: (8)             def get_cached_cairo_context(self, pixel_array: np.ndarray):
430: (8)                 """Returns the cached cairo context of the passed
431: (8)                 pixel array if it exists, and None if it doesn't.
432: (8)                 Parameters
433: (8)                 -----
434: (12)                 pixel_array
435: (8)                     The pixel array to check.
436: (8)                 Returns
437: (8)                 -----
438: (8)                 cairo.Context
439: (8)                     The cached cairo context.
440: (8)                     """
441: (4)             return self.pixel_array_to_cairo_context.get(id(pixel_array), None)
442: (8)             def cache_cairo_context(self, pixel_array: np.ndarray, ctx:
443: (8)                 """Caches the passed Pixel array into a Cairo Context

```

```

444: (8)           -----
445: (8)           pixel_array
446: (12)          The pixel array to cache
447: (8)           ctx
448: (12)          The context to cache it into.
449: (8)           """
450: (8)           self.pixel_array_to_cairo_context[id(pixel_array)] = ctx
451: (4)           def get_cairo_context(self, pixel_array: np.ndarray):
452: (8)             """Returns the cairo context for a pixel array after
453: (8)             caching it to self.pixel_array_to_cairo_context
454: (8)             If that array has already been cached, it returns the
455: (8)             cached version instead.
456: (8)           Parameters
457: (8)           -----
458: (8)           pixel_array
459: (12)          The Pixel array to get the cairo context of.
460: (8)           Returns
461: (8)           -----
462: (8)           cairo.Context
463: (12)          The cairo context of the pixel array.
464: (8)           """
465: (8)           cached_ctx = self.get_cached_cairo_context(pixel_array)
466: (8)           if cached_ctx:
467: (12)             return cached_ctx
468: (8)           pw = self.pixel_width
469: (8)           ph = self.pixel_height
470: (8)           fw = self.frame_width
471: (8)           fh = self.frame_height
472: (8)           fc = self.frame_center
473: (8)           surface = cairo.ImageSurface.create_for_data(
474: (12)             pixel_array,
475: (12)             cairo.FORMAT_ARGB32,
476: (12)             pw,
477: (12)             ph,
478: (8)         )
479: (8)         ctx = cairo.Context(surface)
480: (8)         ctx.scale(pw, ph)
481: (8)         ctx.set_matrix(
482: (12)           cairo.Matrix(
483: (16)             (pw / fw),
484: (16)             0,
485: (16)             0,
486: (16)             -(ph / fh),
487: (16)             (pw / 2) - fc[0] * (pw / fw),
488: (16)             (ph / 2) + fc[1] * (ph / fh),
489: (12)           ),
490: (8)         )
491: (8)         self.cache_cairo_context(pixel_array, ctx)
492: (8)         return ctx
493: (4)           def display_multiple_vectorized_mobjects(
494: (8)             self, vmobjects: list, pixel_array: np.ndarray
495: (4)           ):
496: (8)             """Displays multiple VMobjects in the pixel_array
497: (8)           Parameters
498: (8)           -----
499: (8)           vmobjects
500: (12)             list of VMobjects to display
501: (8)           pixel_array
502: (12)             The pixel array
503: (8)           """
504: (8)           if len(vmobjects) == 0:
505: (12)             return
506: (8)           batch_image_pairs = it.groupby(vmobjects, lambda vm:
507: (8)             vm.get_background_image())
508: (12)
509: (16)           pixel_array)
510: (12)           for image, batch in batch_image_pairs:
511: (8)             if image:
512: (12)               self.display_multiple_background_colored_vmobjects(batch,
513: (8)             else:

```

```

511: (16)                         self.display_multiple_non_background_colored_vmobjects(
512: (20)                           batch,
513: (20)                           pixel_array,
514: (16)                           )
515: (4)                            def display_multiple_non_background_colored_vmobjects(
516: (8)                              self, vmobjects: list, pixel_array: np.ndarray
517: (4):
518: (8)                                """Displays multiple VMobjects in the cairo context, as long as they
don't have
519: (8)                                background colors.
520: (8)                                Parameters
521: (8)                                -----
522: (8)                                vmobjects
523: (12)                                  list of the VMobjects
524: (8)                                pixel_array
525: (12)                                  The Pixel array to add the VMobjects to.
526: (8)                                """
527: (8)                                ctx = self.get_cairo_context(pixel_array)
528: (8)                                for vmobject in vmobjects:
529: (12)                                  self.display_vectorized(vmobject, ctx)
530: (4)                            def display_vectorized(self, vmobject: VMobject, ctx: cairo.Context):
531: (8)                                """Displays a VMobject in the cairo context
532: (8)                                Parameters
533: (8)                                -----
534: (8)                                vmobject
535: (12)                                  The Vectorized Mobject to display
536: (8)                                ctx
537: (12)                                  The cairo context to use.
538: (8)                                Returns
539: (8)                                -----
540: (8)                                Camera
541: (12)                                  The camera object
542: (8)                                """
543: (8)                                self.set_cairo_context_path(ctx, vmobject)
544: (8)                                self.apply_stroke(ctx, vmobject, background=True)
545: (8)                                self.apply_fill(ctx, vmobject)
546: (8)                                self.apply_stroke(ctx, vmobject)
547: (8)                                return self
548: (4)                            def set_cairo_context_path(self, ctx: cairo.Context, vmobject: VMobject):
549: (8)                                """Sets a path for the cairo context with the vmobject passed
550: (8)                                Parameters
551: (8)                                -----
552: (8)                                ctx
553: (12)                                  The cairo context
554: (8)                                vmobject
555: (12)                                  The VMobject
556: (8)                                Returns
557: (8)                                -----
558: (8)                                Camera
559: (12)                                  Camera object after setting cairo_context_path
560: (8)                                """
561: (8)                                points = self.transform_points_pre_display(vmobject, vmobject.points)
562: (8)                                if len(points) == 0:
563: (12)                                  return
564: (8)                                ctx.new_path()
565: (8)                                subpaths = vmobject.gen_subpaths_from_points_2d(points)
566: (8)                                for subpath in subpaths:
567: (12)                                  quads = vmobject.gen_cubic_bezier_tuples_from_points(subpath)
568: (12)                                  ctx.new_sub_path()
569: (12)                                  start = subpath[0]
570: (12)                                  ctx.move_to(*start[:2])
571: (12)                                  for _p0, p1, p2, p3 in quads:
572: (16)                                      ctx.curve_to(*p1[:2], *p2[:2], *p3[:2])
573: (12)                                      if vmobject.consider_points_equals_2d(subpath[0], subpath[-1]):
574: (16)                                          ctx.close_path()
575: (8)                                      return self
576: (4)                            def set_cairo_context_color(
577: (8)                              self, ctx: cairo.Context, rgbs: np.ndarray, vmobject: VMobject
578: (4) ):
```

```

579: (8)             """Sets the color of the cairo context
580: (8)             Parameters
581: (8)             -----
582: (8)             ctx
583: (12)            The cairo context
584: (8)             rgbas
585: (12)            The RGBA array with which to color the context.
586: (8)             vmobject
587: (12)            The VMobject with which to set the color.
588: (8)             Returns
589: (8)             -----
590: (8)             Camera
591: (12)            The camera object
592: (8)             """
593: (8)             if len(rgbas) == 1:
594: (12)                ctx.set_source_rgba(*rgbas[0][2::-1], rgbas[0][3])
595: (8)             else:
596: (12)                points = vmobject.get_gradient_start_and_end_points()
597: (12)                points = self.transform_points_pre_display(vmobject, points)
598: (12)                pat = cairo.LinearGradient(*it.chain(*(point[:2] for point in
599: (12)                    points)))
600: (12)                step = 1.0 / (len(rgbas) - 1)
601: (12)                offsets = np.arange(0, 1 + step, step)
602: (16)                for rgba, offset in zip(rgbas, offsets):
603: (12)                    pat.add_color_stop_rgba(offset, *rgba[2::-1], rgba[3])
604: (8)                ctx.set_source(pat)
605: (4)             return self
606: (8)             def apply_fill(self, ctx: cairo.Context, vmobject: VMobject):
607: (8)                 """Fills the cairo context
608: (8)                 Parameters
609: (8)                 -----
610: (12)                 ctx
611: (8)                 The cairo context
612: (12)                 vmobject
613: (8)                 The VMobject
614: (8)                 Returns
615: (8)                 -----
616: (12)                 Camera
617: (8)                 The camera object.
618: (8)                 """
619: (8)                 self.set_cairo_context_color(ctx, self.get_fill_rgbs(vmobject),
620: (8)                     ctx.fill_preserve())
621: (8)                 return self
622: (4)             def apply_stroke(
623: (8)                 self, ctx: cairo.Context, vmobject: VMobject, background: bool = False
624: (8)             ):
625: (8)                 """Applies a stroke to the VMobject in the cairo context.
626: (8)                 Parameters
627: (8)                 -----
628: (12)                 ctx
629: (8)                 The cairo context
630: (12)                 vmobject
631: (8)                 The VMobject
632: (12)                 background
633: (12)                 Whether or not to consider the background when applying this
634: (8)                 stroke width, by default False
635: (8)                 Returns
636: (8)                 -----
637: (12)                 Camera
638: (8)                 The camera object with the stroke applied.
639: (8)                 """
640: (8)                 width = vmobject.get_stroke_width(background)
641: (12)                 if width == 0:
642: (8)                     return self
643: (12)                 self.set_cairo_context_color(
644: (12)                     ctx,
645: (12)                     self.get_stroke_rgbs(vmobject, background=background),
645: (12)                     vmobject,

```

```

646: (8) )
647: (8)     ctx.set_line_width(
648: (12)         width
649: (12)             * self.cairo_line_width_multiple
650: (12)             * (self.frame_width / self.frame_width),
651: (8)     )
652: (8)     if vmobject.joint_type != LineJointType.AUTO:
653: (12)         ctx.set_line_join(LINE_JOIN_MAP[vmobject.joint_type])
654: (8)     if vmobject.cap_style != CapStyleType.AUTO:
655: (12)         ctx.set_line_cap(CAP_STYLE_MAP[vmobject.cap_style])
656: (8)     ctx.stroke_preserve()
657: (8)     return self
658: (4) def get_stroke_rgbs(self, vmobject: VMobject, background: bool = False):
659: (8)     """Gets the RGBA array for the stroke of the passed
660: (8)     VMobject.
661: (8)     Parameters
662: (8)     -----
663: (8)     vmobject
664: (12)         The VMobject
665: (8)     background
666: (12)         Whether or not to consider the background when getting the stroke
667: (12)         RGBAs, by default False
668: (8)     Returns
669: (8)     -----
670: (8)     np.ndarray
671: (12)         The RGBA array of the stroke.
672: (8)     """
673: (8)     return vmobject.get_stroke_rgbs(background)
674: (4) def get_fill_rgbs(self, vmobject: VMobject):
675: (8)     """Returns the RGBA array of the fill of the passed VMobject
676: (8)     Parameters
677: (8)     -----
678: (8)     vmobject
679: (12)         The VMobject
680: (8)     Returns
681: (8)     -----
682: (8)     np.array
683: (12)         The RGBA Array of the fill of the VMobject
684: (8)     """
685: (8)     return vmobject.get_fill_rgbs()
686: (4) def get_background_colored_vmobject_displayer(self):
687: (8)     """Returns the background_colored_vmobject_displayer
688: (8)     if it exists or makes one and returns it if not.
689: (8)     Returns
690: (8)     -----
691: (8)     BackGroundColoredVMobjectDisplayer
692: (12)         Object that displays VMobjects that have the same color
693: (12)         as the background.
694: (8)     """
695: (8)     bcvd = "background_colored_vmobject_displayer"
696: (8)     if not hasattr(self, bcvd):
697: (12)         setattr(self, bcvd, BackgroundColoredVMobjectDisplayer(self))
698: (8)     return getattr(self, bcvd)
699: (4) def display_multiple_background_colored_vmobjects(
700: (8)     self, cvmobjects: list, pixel_array: np.ndarray
701: (4) ):
702: (8)     """Displays multiple vmobjects that have the same color as the
background.
703: (8)     Parameters
704: (8)     -----
705: (8)     cvmobjects
706: (12)         List of Colored VMobjects
707: (8)     pixel_array
708: (12)         The pixel array.
709: (8)     Returns
710: (8)     -----
711: (8)     Camera
712: (12)         The camera object.
713: (8)     """

```

```

714: (8)             displayer = self.get_background_colored_vmobobject_displayer()
715: (8)             cvmobject_pixel_array = displayer.display(*cvmobjects)
716: (8)             self.overlay_rgba_array(pixel_array, cvmobject_pixel_array)
717: (8)             return self
718: (4)         def display_multiple_point_cloud_mobjects(
719: (8)             self, pmobjects: list, pixel_array: np.ndarray
720: (4)         ):
721: (8)             """Displays multiple PMobjects by modifying the passed pixel array.
722: (8)             Parameters
723: (8)             -----
724: (8)             pmobjects
725: (12)                 List of PMobjects
726: (8)             pixel_array
727: (12)                 The pixel array to modify.
728: (8)
729: (8)
730: (12)         for pmobject in pmobjects:
731: (16)             self.display_point_cloud(
732: (16)                 pmobject,
733: (16)                 pmobject.points,
734: (16)                 pmobject.rgbas,
735: (16)                 self.adjusted_thickness(pmobject.stroke_width),
736: (12)                 pixel_array,
737: (12)
738: (4)     def display_point_cloud(
739: (8)         self,
740: (8)         pmobject: PMobject,
741: (8)         points: list,
742: (8)         rgbas: np.ndarray,
743: (8)         thickness: float,
744: (8)         pixel_array: np.ndarray,
745: (4)     ):
746: (8)         """Displays a PMobject by modifying the pixel array suitably.
747: (8)         TODO: Write a description for the rgbas argument.
748: (8)         Parameters
749: (8)         -----
750: (12)             pmobject
751: (8)                 Point Cloud Mobject
752: (8)             points
753: (8)                 The points to display in the point cloud mobject
754: (8)             rgbas
755: (12)                 The thickness of each point of the PMobject
756: (8)             pixel_array
757: (12)                 The pixel array to modify.
758: (8)
759: (8)
760: (12)         if len(points) == 0:
761: (8)             return
762: (8)         pixel_coords = self.points_to_pixel_coords(pmobject, points)
763: (8)         pixel_coords = self.thickened_coordinates(pixel_coords, thickness)
764: (8)         rgba_len = pixel_array.shape[2]
765: (8)         rgbas = (self.rgb_max_val * rgbas).astype(self.pixel_array_dtype)
766: (8)         target_len = len(pixel_coords)
767: (8)         factor = target_len // len(rgbas)
768: (8)         rgbas = np.array([rgbas] * factor).reshape((target_len, rgba_len))
769: (8)         on_screen_indices = self.on_screen_pixels(pixel_coords)
770: (8)         pixel_coords = pixel_coords[on_screen_indices]
771: (8)         rgbas = rgbas[on_screen_indices]
772: (8)         ph = self.pixel_height
773: (8)         pw = self.pixel_width
774: (8)         flattener = np.array([1, pw], dtype="int")
775: (8)         flattener = flattener.reshape((2, 1))
776: (8)         indices = np.dot(pixel_coords, flattener)[:, 0]
777: (8)         indices = indices.astype("int")
778: (8)         new_pa = pixel_array.reshape((ph * pw, rgba_len))
779: (8)         new_pa[indices] = rgbas
780: (4)     def display_multiple_image_mobjects(
781: (8)         self, image_mobjects: list, pixel_array: np.ndarray
782: (4)     ):

```

```

783: (8)                                     """Displays multiple image mobjects by modifying the passed
pixel_array.
784: (8)                                     Parameters
785: (8)                                     -----
786: (8)                                     image_mobjects
787: (12)                                    list of ImageMobjects
788: (8)                                     pixel_array
789: (12)                                    The pixel array to modify.
790: (8)
791: (8)
792: (12)                                    for image_mobject in image_mobjects:
793: (4)                                     self.display_image_mobject(image_mobject, pixel_array)
794: (8)                                     def display_image_mobject(
795: (4)                                     self, image_mobject: AbstractImageMobject, pixel_array: np.ndarray
):                                     """
796: (8)                                     Displays an ImageMobject by changing the pixel_array suitably.
797: (8)                                     Parameters
798: (8)
799: (8)                                     image_mobject
800: (12)                                    The imageMobject to display
801: (8)                                     pixel_array
802: (12)                                    The Pixel array to put the imagemobject in.
803: (8)
804: (8)                                     corner_coords = self.points_to_pixel_coords(image_mobject,
image_mobject.points)
805: (8)                                     mode="RGBA")
806: (8)
807: (8)
808: (8)
809: (8)                                     ul_coords, ur_coords, dl_coords, _ = corner_coords
mode="RGBA"
810: (8)                                     right_vect = ur_coords - ul_coords
811: (8)                                     down_vect = dl_coords - ul_coords
812: (8)                                     center_coords = ul_coords + (right_vect + down_vect) / 2
813: (12)                                     sub_image = Image.fromarray(image_mobject.get_pixel_array(),
814: (12)                                     pixel_width = max(int(pdist([ul_coords, ur_coords]).item()), 1)
815: (8)                                     pixel_height = max(int(pdist([ul_coords, dl_coords]).item()), 1)
816: (8)                                     sub_image = sub_image.resize(
817: (8)                                     (pixel_width, pixel_height),
818: (8)                                     resample=image_mobject.resampling_algorithm,
819: (12)                                     )
820: (16)                                     angle = angle_of_vector(right_vect)
821: (16)                                     adjusted_angle = -int(360 * angle / TAU)
822: (16)                                     if adjusted_angle != 0:
823: (12)                                     sub_image = sub_image.rotate(
824: (8)                                     adjusted_angle,
825: (12)                                     resample=image_mobject.resampling_algorithm,
826: (12)                                     expand=1,
827: (8)
828: (8)
829: (8)
830: (8)
831: (12)
832: (12)
833: (16)
834: (16)
835: (16)
836: (16)
837: (12)
838: (8)
839: (8)
840: (4)                                     new_ul_coords = center_coords - np.array(sub_image.size) / 2
np.ndarray):
841: (8)                                     new_ul_coords = new_ul_coords.astype(int)
842: (8)                                     full_image.paste(
843: (8)                                     sub_image,
844: (8)                                     box=(
845: (12)                                     new_ul_coords[0],
846: (8)                                     new_ul_coords[1],
847: (12)                                     new_ul_coords[0] + sub_image.size[0],
new_ul_coords[1] + sub_image.size[1],
),
)
848: (8)                                     self.overlay_PIL_image(pixel_array, full_image)
def overlay_rgba_array(self, pixel_array: np.ndarray, new_array:
849: (8)                                     """Overlays an RGBA array on top of the given Pixel array.
850: (8)                                     Parameters
851: (8)                                     -----
852: (8)                                     pixel_array
853: (12)                                    The original pixel array to modify.
854: (8)                                     new_array
855: (12)                                    The new pixel array to overlay.

```

```

848: (8)             """
849: (8)         self.overlay_PIL_image(pixel_array, self.get_image(new_array))
850: (4)     def overlay_PIL_image(self, pixel_array: np.ndarray, image: Image):
851: (8)         """Overlays a PIL image on the passed pixel array.
852: (8)         Parameters
853: (8)             -----
854: (8)             pixel_array
855: (12)                 The Pixel array
856: (8)             image
857: (12)                 The Image to overlay.
858: (8)         """
859: (8)         pixel_array[:, :] = np.array(
860: (12)             Image.alpha_composite(self.get_image(pixel_array), image),
861: (12)             dtype="uint8",
862: (8)         )
863: (4)     def adjust_out_of_range_points(self, points: np.ndarray):
864: (8)         """If any of the points in the passed array are out of
865: (8)             the viable range, they are adjusted suitably.
866: (8)         Parameters
867: (8)             -----
868: (8)             points
869: (12)                 The points to adjust
870: (8)         Returns
871: (8)             -----
872: (8)             np.array
873: (12)                 The adjusted points.
874: (8)         """
875: (8)         if not np.any(points > self.max_allowable_norm):
876: (12)             return points
877: (8)         norms = np.apply_along_axis(np.linalg.norm, 1, points)
878: (8)         violator_indices = norms > self.max_allowable_norm
879: (8)         violators = points[violator_indices, :]
880: (8)         violator_norms = norms[violator_indices]
881: (8)         reshaped_norms = np.repeat(
882: (12)             violator_norms.reshape((len(violator_norms), 1)),
883: (12)             points.shape[1],
884: (12)             1,
885: (8)         )
886: (8)         rescaled = self.max_allowable_norm * violators / reshaped_norms
887: (8)         points[violator_indices] = rescaled
888: (8)         return points
889: (4)     def transform_points_pre_display(
890: (8)         self,
891: (8)         mobobject,
892: (8)         points,
893: (4)     ): # TODO: Write more detailed docstrings for this method.
894: (8)         if not np.all(np.isfinite(points)):
895: (12)             points = np.zeros((1, 3))
896: (8)         return points
897: (4)     def points_to_pixel_coords(
898: (8)         self,
899: (8)         mobobject,
900: (8)         points,
901: (4)     ): # TODO: Write more detailed docstrings for this method.
902: (8)         points = self.transform_points_pre_display(mobobject, points)
903: (8)         shifted_points = points - self.frame_center
904: (8)         result = np.zeros((len(points), 2))
905: (8)         pixel_height = self.pixel_height
906: (8)         pixel_width = self.pixel_width
907: (8)         frame_height = self.frame_height
908: (8)         frame_width = self.frame_width
909: (8)         width_mult = pixel_width / frame_width
910: (8)         width_add = pixel_width / 2
911: (8)         height_mult = pixel_height / frame_height
912: (8)         height_add = pixel_height / 2
913: (8)         height_mult *= -1
914: (8)         result[:, 0] = shifted_points[:, 0] * width_mult + width_add
915: (8)         result[:, 1] = shifted_points[:, 1] * height_mult + height_add
916: (8)         return result.astype("int")

```

```

917: (4)             def on_screen_pixels(self, pixel_coords: np.ndarray):
918: (8)                 """Returns array of pixels that are on the screen from a given
919: (8)                 array of pixel_coordinates
920: (8)                 Parameters
921: (8)                 -----
922: (8)                 pixel_coords
923: (12)                   The pixel coords to check.
924: (8)                 Returns
925: (8)                 -----
926: (8)                 np.array
927: (12)                   The pixel coords on screen.
928: (8)                 """
929: (8)             return reduce(
930: (12)                 op.and_,
931: (12)                 [
932: (16)                     pixel_coords[:, 0] >= 0,
933: (16)                     pixel_coords[:, 0] < self.pixel_width,
934: (16)                     pixel_coords[:, 1] >= 0,
935: (16)                     pixel_coords[:, 1] < self.pixel_height,
936: (12)                 ],
937: (8)             )
938: (4)             def adjusted_thickness(self, thickness: float) -> float:
939: (8)                 """Computes the adjusted stroke width for a zoomed camera.
940: (8)                 Parameters
941: (8)                 -----
942: (8)                 thickness
943: (12)                   The stroke width of a mobject.
944: (8)                 Returns
945: (8)                 -----
946: (8)                 float
947: (12)                   The adjusted stroke width that reflects zooming in with
948: (12)                     the camera.
949: (8)                 """
950: (8)             big_sum = op.add(config["pixel_height"], config["pixel_width"])
951: (8)             this_sum = op.add(self.pixel_height, self.pixel_width)
952: (8)             factor = big_sum / this_sum
953: (8)             return 1 + (thickness - 1) * factor
954: (4)             def get_thickening_nudges(self, thickness: float):
955: (8)                 """Determine a list of vectors used to nudge
956: (8)                 two-dimensional pixel coordinates.
957: (8)                 Parameters
958: (8)                 -----
959: (8)                 thickness
960: (8)                 Returns
961: (8)                 -----
962: (8)                 np.array
963: (8)                 """
964: (8)             thickness = int(thickness)
965: (8)             _range = list(range(-thickness // 2 + 1, thickness // 2 + 1))
966: (8)             return np.array(list(it.product(_range, _range)))
967: (4)             def thickened_coordinates(self, pixel_coords: np.ndarray, thickness:
float):
968: (8)                 """Returns thickened coordinates for a passed array of pixel coords
and
969: (8)                 a thickness to thicken by.
970: (8)                 Parameters
971: (8)                 -----
972: (8)                 pixel_coords
973: (12)                   Pixel coordinates
974: (8)                 thickness
975: (12)                   Thickness
976: (8)                 Returns
977: (8)                 -----
978: (8)                 np.array
979: (12)                   Array of thickened pixel coords.
980: (8)                 """
981: (8)             nudges = self.get_thickening_nudges(thickness)
982: (8)             pixel_coords = np.array([pixel_coords + nudge for nudge in nudges])
983: (8)             size = pixel_coords.size

```

```

984: (8)             return pixel_coords.reshape((size // 2, 2))
985: (4)         def get_coords_of_all_pixels(self):
986: (8)             """Returns the cartesian coordinates of each pixel.
987: (8)             Returns
988: (8)             -----
989: (8)             np.ndarray
990: (12)             The array of cartesian coordinates.
991: (8)
992: (8)             """
993: (8)             full_space_dims = np.array([self.frame_width, self.frame_height])
994: (8)             full_pixel_dims = np.array([self.pixel_width, self.pixel_height])
995: (12)             uncentered_pixel_coords = np.indices([self.pixel_height,
996: (8)                         self.pixel_width])[2]
997: (8)
998: (12)             ::-1
999: (8)             ].transpose(1, 2, 0)
1000: (8)             uncentered_space_coords = (
1001: (8)                 uncentered_pixel_coords * full_space_dims
1002: (8)             ) / full_pixel_dims
1003: (0)             centered_space_coords = uncentered_space_coords - (full_space_dims /
1004: (4)             2)
1005: (4)             centered_space_coords = centered_space_coords * (1, -1)
1006: (4)             return centered_space_coords
1007: (4)
1008: (4)
1009: (8)         class BackgroundColoredVMobjectDisplayer:
1010: (4)             """Auxiliary class that handles displaying vectorized mobjects with
1011: (4)             a set background image.
1012: (8)             Parameters
1013: (8)             -----
1014: (8)             camera
1015: (8)                 Camera object to use.
1016: (4)
1017: (8)
1018: (4)
1019: (8)
1020: (8)
1021: (8)
1022: (8)
1023: (8)
1024: (4)
1025: (8)
1026: (8)
1027: (8)
1028: (8)
1029: (12)
1030: (8)
1031: (12)
1032: (8)
1033: (12)
1034: (8)
1035: (12)
1036: (8)
1037: (8)
1038: (8)
1039: (12)
1040: (8)
1041: (8)
1042: (8)
1043: (8)
1044: (8)
1045: (4)
1046: (8)
1047: (4)
1048: (8)
1049: (8)
1050: (8)

    def __init__(self, camera: Camera):
        self.camera = camera
        self.file_name_to_pixel_array_map = {}
        self.pixel_array = np.array(camera.pixel_array)
        self.reset_pixel_array()

    def reset_pixel_array(self):
        self.pixel_array[:, :] = 0

    def resize_background_array(
            self,
            background_array: np.ndarray,
            new_width: float,
            new_height: float,
            mode: str = "RGBA",
    ):
        """Resizes the pixel array representing the background.
        Parameters
        -----
        background_array
            The pixel
        new_width
            The new width of the background
        new_height
            The new height of the background
        mode
            The PIL image mode, by default "RGBA"
        Returns
        -----
        np.array
            The numpy pixel array of the resized background.
        """
        image = Image.fromarray(background_array)
        image = image.convert(mode)
        resized_image = image.resize((new_width, new_height))
        return np.array(resized_image)

    def resize_background_array_to_match(
            self, background_array: np.ndarray, pixel_array: np.ndarray
    ):
        """Resizes the background array to match the passed pixel array.
        Parameters
        -----

```

```

1051: (8)           background_array
1052: (12)          The prospective pixel array.
1053: (8)
1054: (12)          pixel_array
1055: (8)          The pixel array whose width and height should be matched.
1056: (8)
1057: (8)
1058: (12)          Returns
1059: (8)          -----
1060: (8)          np.array
1061: (8)          The resized background array.
1062: (8)          """
1063: (4)          height, width = pixel_array.shape[:2]
1064: (8)          mode = "RGBA" if pixel_array.shape[2] == 4 else "RGB"
1065: (8)          return self.resize_background_array(background_array, width, height,
1066: (8)          mode)
1067: (8)
1068: (12)          def get_background_array(self, image: Image.Image | pathlib.Path | str):
1069: (8)          """Gets the background array that has the passed file_name.
1070: (8)          Parameters
1071: (8)          -----
1072: (12)          image
1073: (8)          The background image or its file name.
1074: (8)          Returns
1075: (8)          -----
1076: (12)          np.ndarray
1077: (8)          The pixel array of the image.
1078: (8)          """
1079: (12)          image_key = str(image)
1080: (8)          if image_key in self.file_name_to_pixel_array_map:
1081: (12)              return self.file_name_to_pixel_array_map[image_key]
1082: (8)          if isinstance(image, str):
1083: (12)              full_path = get_full_raster_image_path(image)
1084: (8)              image = Image.open(full_path)
1085: (8)              back_array = np.array(image)
1086: (4)              pixel_array = self.pixel_array
1087: (8)              if not np.all(pixel_array.shape == back_array.shape):
1088: (12)                  back_array = self.resize_background_array_to_match(back_array,
1089: (8)                      self.file_name_to_pixel_array_map[image_key] = back_array
1090: (8)                      return back_array
1091: (8)
1092: (12)          def display(self, *cvmobjects: VMobject):
1093: (8)          """Displays the colored VMobjects.
1094: (8)          Parameters
1095: (12)          *cvmobjects
1096: (8)          The VMobjects
1097: (8)
1098: (8)          Returns
1099: (8)          -----
1100: (12)          np.array
1101: (12)          The pixel array with the `cvmobjects` displayed.
1102: (12)
1103: (16)
1104: (16)
1105: (12)
1106: (12)
1107: (16)
1108: (16)
1109: (12)
1110: (12)
1111: (16)
1112: (12)
1113: (16)
1114: (12)
1115: (8)
cv.get_background_image())
curr_array = None
for image, batch in batch_image_pairs:
    background_array = self.get_background_array(image)
    pixel_array = self.pixel_array
    self.camera.display_multiple_non_background_colored_vmo...
        batch,
        pixel_array,
    )
    new_array = np.array(
        (background_array * pixel_array.astype("float") / 255),
        dtype=self.camera.pixel_array_dtype,
    )
    if curr_array is None:
        curr_array = new_array
    else:
        curr_array = np.maximum(curr_array, new_array)
    self.reset_pixel_array()
return curr_array

```

## File 18 - numbers.py:

```

1: (0)         """Animations for changing numbers."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = ["ChangingDecimal", "ChangeDecimalToValue"]
4: (0)         import typing
5: (0)         from manim.mobject.text.numbers import DecimalNumber
6: (0)         from ..animation.animation import Animation
7: (0)         from ..utils.bezier import interpolate
8: (0)         class ChangingDecimal(Animation):
9: (4)             def __init__(
10: (8)                 self,
11: (8)                 decimal_mob: DecimalNumber,
12: (8)                 number_update_func: typing.Callable[[float], float],
13: (8)                 suspend_mobject_updating: bool | None = False,
14: (8)                 **kwargs,
15: (4)             ) -> None:
16: (8)                 self.check_validity_of_input(decimal_mob)
17: (8)                 self.number_update_func = number_update_func
18: (8)                 super().__init__(
19: (12)                     decimal_mob, suspend_mobject_updating=suspend_mobject_updating,
**kwargs
20: (8)             )
21: (4)             def check_validity_of_input(self, decimal_mob: DecimalNumber) -> None:
22: (8)                 if not isinstance(decimal_mob, DecimalNumber):
23: (12)                     raise TypeError("ChangingDecimal can only take in a
DecimalNumber")
24: (4)             def interpolate_mobject(self, alpha: float) -> None:
25: (8)                 self.mobject.set_value(self.number_update_func(self.rate_func(alpha)))
26: (0)         class ChangeDecimalToValue(ChangingDecimal):
27: (4)             def __init__(
28: (8)                 self, decimal_mob: DecimalNumber, target_number: int, **kwargs
29: (4)             ) -> None:
30: (8)                 start_number = decimal_mob.number
31: (8)                 super().__init__(
32: (12)                     decimal_mob, lambda a: interpolate(start_number, target_number,
a), **kwargs
33: (8)             )

```

## File 19 - movement.py:

```

1: (0)         """Animations related to movement."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = [
4: (4)             "Homotopy",
5: (4)             "SmoothedVectorizedHomotopy",
6: (4)             "ComplexHomotopy",
7: (4)             "PhaseFlow",
8: (4)             "MoveAlongPath",
9: (0)
10: (0)            ]
11: (0)            from typing import TYPE_CHECKING, Any, Callable
12: (0)            import numpy as np
13: (0)            from ..animation.animation import Animation
14: (0)            from ..utils.rate_functions import linear
15: (0)            if TYPE_CHECKING:
16: (4)                from ..mobject.mobject import Mobject, VMobject
17: (4)            class Homotopy(Animation):
18: (4)                """A Homotopy.
19: (4)                This is an animation transforming the points of a mobject according
20: (4)                to the specified transformation function. With the parameter :math:`t`
21: (4)                moving from 0 to 1 throughout the animation and :math:`(x, y, z)`
22: (4)                describing the coordinates of the point of a mobject,
23: (4)                the function passed to the ``homotopy`` keyword argument should
24: (4)                transform the tuple :math:`(x, y, z, t)` to :math:`(x', y', z')`,
the coordinates the original point is transformed to at time :math:`t``.

```

```

25: (4)             Parameters
26: (4)             -----
27: (4)             homotopy
28: (8)                 A function mapping :math:`(x, y, z, t)` to :math:`(x', y', z')`.
29: (4)             mobject
30: (8)                 The mobject transformed under the given homotopy.
31: (4)             run_time
32: (8)                 The run time of the animation.
33: (4)             apply_function_kwds
34: (8)                 Keyword arguments propagated to :meth:`.Mobject.apply_function`.
35: (4)             kwargs
36: (8)                 Further keyword arguments passed to the parent class.
37: (4)
38: (4)             def __init__(
39: (8)                 self,
40: (8)                 homotopy: Callable[[float, float, float, float], tuple[float, float,
41: (8)                     float]],
42: (8)                 mobject: Mobject,
43: (8)                 run_time: float = 3,
44: (8)                 apply_function_kwds: dict[str, Any] | None = None,
45: (8)                 **kwargs,
46: (4)             ) -> None:
47: (8)                 self.homotopy = homotopy
48: (8)                 self.apply_function_kwds = (
49: (8)                     apply_function_kwds if apply_function_kwds is not None else {}
50: (8)                 )
51: (4)             super().__init__(mobject, run_time=run_time, **kwargs)
52: (8)             def function_at_time_t(self, t: float) -> tuple[float, float, float]:
53: (8)                 return lambda p: self.homotopy(*p, t)
54: (4)             def interpolate_submobject(
55: (8)                 self,
56: (8)                 submobject: Mobject,
57: (8)                 starting_submobject: Mobject,
58: (8)                 alpha: float,
59: (4)             ) -> None:
60: (8)                 submobject.points = starting_submobject.points
61: (8)                 submobject.apply_function(
62: (8)                     self.function_at_time_t(alpha), **self.apply_function_kwds
63: (0)
64: (4)             class SmoothedVectorizedHomotopy(Homotopy):
65: (8)                 def interpolate_submobject(
66: (8)                     self,
67: (8)                     submobject: Mobject,
68: (8)                     starting_submobject: Mobject,
69: (8)                     alpha: float,
70: (4)                 ) -> None:
71: (8)                     super().interpolate_submobject(submobject, starting_submobject, alpha)
72: (0)
73: (4)             class ComplexHomotopy(Homotopy):
74: (8)                 def __init__(
75: (8)                     self, complex_homotopy: Callable[[complex], float], mobject: Mobject,
76: (8)                     **kwargs
77: (4)                 ) -> None:
78: (8)                     """
79: (8)                         Complex Homotopy a function Cx[0, 1] to C
80: (8)                     """
81: (12)                     def homotopy(
82: (12)                         x: float,
83: (12)                         y: float,
84: (12)                         z: float,
85: (12)                         t: float,
86: (12)                     ) -> tuple[float, float, float]:
87: (12)                         c = complex_homotopy(complex(x, y), t)
88: (12)                         return (c.real, c.imag, z)
89: (0)
90: (4)             class PhaseFlow(Animation):
91: (8)                 def __init__(

```

```

92: (8)             mobject: Mobject,
93: (8)             virtual_time: float = 1,
94: (8)             suspend_mobject_updating: bool = False,
95: (8)             rate_func: Callable[[float], float] = linear,
96: (8)             **kwargs,
97: (4)         ) -> None:
98: (8)             self.virtual_time = virtual_time
99: (8)             self.function = function
100: (8)            super().__init__(
101: (12)                mobject,
102: (12)                suspend_mobject_updating=suspend_mobject_updating,
103: (12)                rate_func=rate_func,
104: (12)                **kwargs,
105: (8)
106: (4)        def interpolate_mobject(self, alpha: float) -> None:
107: (8)            if hasattr(self, "last_alpha"):
108: (12)                dt = self.virtual_time * (
109: (16)                    self.rate_func(alpha) - self.rate_func(self.last_alpha)
110: (12)                )
111: (12)                self.mobject.apply_function(lambda p: p + dt * self.function(p))
112: (8)            self.last_alpha = alpha
113: (0)        class MoveAlongPath(Animation):
114: (4)            """Make one mobject move along the path of another mobject.
115: (4)            .. manim:: MoveAlongPathExample
116: (8)            class MoveAlongPathExample(Scene):
117: (12)                def construct(self):
118: (16)                    d1 = Dot().set_color(ORANGE)
119: (16)                    l1 = Line(LEFT, RIGHT)
120: (16)                    l2 = VMobject()
121: (16)                    self.add(d1, l1, l2)
122: (16)                    l2.add_updater(lambda x: x.become(Line(LEFT,
d1.get_center()).set_color(ORANGE)))
123: (16)                    self.play(MoveAlongPath(d1, l1), rate_func=linear)
124: (4)            """
125: (4)            def __init__(self,
126: (8)                mobject: Mobject,
127: (8)                path: VMobject,
128: (8)                suspend_mobject_updating: bool | None = False,
129: (8)                **kwargs,
130: (8)
131: (4)        ) -> None:
132: (8)            self.path = path
133: (8)            super().__init__(mobject, suspend_mobject_updating=suspend_mobject_updating,
134: (12)                **kwargs)
135: (8)
136: (4)            def interpolate_mobject(self, alpha: float) -> None:
137: (8)                point = self.path.point_from_proportion(self.rate_func(alpha))
138: (8)                self.mobject.move_to(point)

```

---

#### File 20 - rotation.py:

```

1: (0)        """Animations related to rotation."""
2: (0)        from __future__ import annotations
3: (0)        __all__ = ["Rotating", "Rotate"]
4: (0)        from typing import TYPE_CHECKING, Callable, Sequence
5: (0)        import numpy as np
6: (0)        from ..animation.animation import Animation
7: (0)        from ..animation.transform import Transform
8: (0)        from ..constants import OUT, PI, TAU
9: (0)        from ..utils.rate_functions import linear
10: (0)       if TYPE_CHECKING:
11: (4)           from ..mobject.mobject import Mobject
12: (0)       class Rotating(Animation):
13: (4)           def __init__(self,
14: (8)               mobject: Mobject,
15: (8)

```

```

16: (8)                 axis: np.ndarray = OUT,
17: (8)                 radians: np.ndarray = TAU,
18: (8)                 about_point: np.ndarray | None = None,
19: (8)                 about_edge: np.ndarray | None = None,
20: (8)                 run_time: float = 5,
21: (8)                 rate_func: Callable[[float], float] = linear,
22: (8)                 **kwargs,
23: (4)             ) -> None:
24: (8)                 self.axis = axis
25: (8)                 self.radians = radians
26: (8)                 self.about_point = about_point
27: (8)                 self.about_edge = about_edge
28: (8)                 super().__init__(mobject, run_time=run_time, rate_func=rate_func,
**kwargs)
29: (4)             def interpolate_mobject(self, alpha: float) -> None:
30: (8)                 self.mobject.become(self.starting_mobject)
31: (8)                 self.mobject.rotate(
32: (12)                     self.rate_func(alpha) * self.radians,
33: (12)                     axis=self.axis,
34: (12)                     about_point=self.about_point,
35: (12)                     about_edge=self.about_edge,
36: (8)                 )
37: (0)             class Rotate(Transform):
38: (4)                 """Animation that rotates a Mobject.
39: (4)                 Parameters
40: (4)                 -----
41: (4)                 mobject
42: (8)                     The mobject to be rotated.
43: (4)                 angle
44: (8)                     The rotation angle.
45: (4)                 axis
46: (8)                     The rotation axis as a numpy vector.
47: (4)                 about_point
48: (8)                     The rotation center.
49: (4)                 about_edge
50: (8)                     If ``about_point`` is ``None``, this argument specifies
51: (8)                     the direction of the bounding box point to be taken as
52: (8)                     the rotation center.
53: (4)             Examples
54: (4)             -----
55: (4)             .. manim:: UsingRotate
56: (8)                 class UsingRotate(Scene):
57: (12)                     def construct(self):
58: (16)                         self.play(
59: (20)                             Rotate(
60: (24)                                 Square(side_length=0.5).shift(UP * 2),
61: (24)                                 angle=2*PI,
62: (24)                                 about_point=ORIGIN,
63: (24)                                 rate_func=linear,
64: (20)                         ),
65: (20)                         Rotate(Square(side_length=0.5), angle=2*PI,
rate_func=linear),
66: (20)                     )
67: (4)             """
68: (4)             def __init__(
69: (8)                 self,
70: (8)                 mobject: Mobject,
71: (8)                 angle: float = PI,
72: (8)                 axis: np.ndarray = OUT,
73: (8)                 about_point: Sequence[float] | None = None,
74: (8)                 about_edge: Sequence[float] | None = None,
75: (8)                 **kwargs,
76: (4)             ) -> None:
77: (8)                 if "path_arc" not in kwargs:
78: (12)                     kwargs["path_arc"] = angle
79: (8)                 if "path_arc_axis" not in kwargs:
80: (12)                     kwargs["path_arc_axis"] = axis
81: (8)                 self.angle = angle
82: (8)                 self.axis = axis

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
83: (8)             self.about_edge = about_edge
84: (8)             self.about_point = about_point
85: (8)             if self.about_point is None:
86: (12)                 self.about_point = mobject.get_center()
87: (8)             super().__init__(mobject, path_arc_centers=self.about_point, **kwargs)
88: (4)         def create_target(self) -> Mobject:
89: (8)             target = self.mobject.copy()
90: (8)             target.rotate(
91: (12)                 self.angle,
92: (12)                 axis=self.axis,
93: (12)                 about_point=self.about_point,
94: (12)                 about_edge=self.about_edge,
95: (8)
96: (8)             )
return target
```

-----

## File 21 - \_\_init\_\_.py:

```
1: (0)             """Animations and utility mobjects related to update functions.
2: (0)             Modules
3: (0)             =====
4: (0)             .. autosummary::
5: (4)                 :toctree: ../reference
6: (4)                 ~mobject_update_utils
7: (4)                 ~update
8: (0)             """
```

-----

## File 22 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 23 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 24 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 25 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 26 - transform.py:

```
1: (0)             """Animations transforming one mobject into another."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "Transform",
5: (4)                 "ReplacementTransform",
6: (4)                 "TransformFromCopy",
7: (4)                 "ClockwiseTransform",
8: (4)                 "CounterclockwiseTransform",
9: (4)                 "MoveToTarget",
10: (4)                "ApplyMethod",
11: (4)                "ApplyPointwiseFunction",
12: (4)                "ApplyPointwiseFunctionToCenter",
13: (4)                "FadeToColor",
```

```

14: (4)           "FadeTransform",
15: (4)           "FadeTransformPieces",
16: (4)           "ScaleInPlace",
17: (4)           "ShrinkToCenter",
18: (4)           "Restore",
19: (4)           "ApplyFunction",
20: (4)           "ApplyMatrix",
21: (4)           "ApplyComplexFunction",
22: (4)           "CyclicReplace",
23: (4)           "Swap",
24: (4)           "TransformAnimations",
25: (0)
26: (0)       ]
27: (0)       import inspect
28: (0)       import types
29: (0)       from typing import TYPE_CHECKING, Any, Callable, Iterable, Sequence
30: (0)       import numpy as np
31: (0)       from manim.mobject.opengl.opengl_mobject import OpenGLGroup, OpenGLObject
32: (0)       from .. import config
33: (0)       from ..animation.animation import Animation
34: (0)       from ..constants import (
35: (4)           DEFAULT_POINTWISE_FUNCTION_RUN_TIME,
36: (4)           DEGREES,
37: (4)           ORIGIN,
38: (4)           OUT,
39: (4)           RendererType,
40: (0)
41: (0)       )
42: (0)       from ..mobject.mobject import Group, Mobject
43: (0)       from ..utils.paths import path_along_arc, path_along_circles
44: (0)       from ..utils.rate_functions import smooth, squish_rate_func
45: (0)       if TYPE_CHECKING:
46: (4)           from ..scene.scene import Scene
47: (0)       class Transform(Animation):
48: (4)           """A Transform transforms a Mobject into a target Mobject.
49: (4)           Parameters
50: (4)           -----
51: (4)           mobject
52: (8)               The :class:`.Mobject` to be transformed. It will be mutated to become
the ``target_mobject``.
53: (4)           target_mobject
54: (8)               The target of the transformation.
55: (4)           path_func
56: (8)               A function defining the path that the points of the ``mobject`` are
being moved
57: (8)               along until they match the points of the ``target_mobject``, see
58: (4)           path_arc
59: (8)               The arc angle (in radians) that the points of ``mobject`` will follow
60: (4)           path_arc_axis
61: (8)               The axis to rotate along if using a circular path arc, see
62: (4)           path_arc_centers
63: (8)               The center of the circular arcs along which the points of ``mobject``
are
64: (8)               moved by the transformation.
65: (8)               If this is set and ``path_func`` is not set, then a
66: (4)           ``path_along_circles`` path will be generated
67: (8)               using the ``path_arc`` parameters and stored in ``path_func``. If
68: (8)               ``path_func`` is set, this and the
69: (8)                   other ``path_arc`` fields are set as attributes, but a ``path_func``
is not generated from it.
70: (4)           replace_mobject_with_target_in_scene
71: (8)               Controls which mobject is replaced when the transformation is
complete.
72: (8)               If set to True, ``mobject`` will be removed from the scene and
``target_mobject`` will

```

```

71: (8)           replace it. Otherwise, ``target_mobject`` is never added and
`mobject`` just takes its shape.
72: (4)           Examples
73: (4)           -----
74: (4)           .. manim :: TransformPathArc
75: (8)           class TransformPathArc(Scene):
76: (12)           def construct(self):
77: (16)               def make_arc_path(start, end, arc_angle):
78: (20)                   points = []
79: (20)                   p_fn = path_along_arc(arc_angle)
80: (20)                   for alpha in range(0, 11):
81: (24)                       points.append(p_fn(start, end, alpha / 10.0))
82: (20)                   path = VMobject(stroke_color=YELLOW)
83: (20)                   path.set_points_smoothly(points)
84: (20)                   return path
85: (16)           left = Circle(stroke_color=BLUE_E, fill_opacity=1.0,
radius=0.5).move_to(LEFT * 2)
86: (16)           colors = [TEAL_A, TEAL_B, TEAL_C, TEAL_D, TEAL_E, GREEN_A]
87: (16)           examples = [-90, 0, 30, 90, 180, 270]
88: (16)           anims = []
89: (16)           for idx, angle in enumerate(examples):
90: (20)               left_c = left.copy().shift((3 - idx) * UP)
91: (20)               left_c.fill_color = colors[idx]
92: (20)               right_c = left_c.copy().shift(4 * RIGHT)
93: (20)               path_arc = make_arc_path(left_c.get_center(),
right_c.get_center(),
94: (45)                   arc_angle=angle * DEGREES)
95: (20)               desc = Text('%d°' % examples[idx]).next_to(left_c, LEFT)
96: (20)               self.add(
97: (24)                   path_arc.set_z_index(1),
98: (24)                   desc.set_z_index(2),
99: (24)                   left_c.set_z_index(3),
100: (20)               )
101: (20)               anims.append(Transform(left_c, right_c, path_arc=angle *
DEGREES))
102: (16)           self.play(*anim, run_time=2)
103: (16)           self.wait()
104: (4)           """
105: (4)           def __init__(
106: (8)               self,
107: (8)               mobject: Mobject | None,
108: (8)               target_mobject: Mobject | None = None,
109: (8)               path_func: Callable | None = None,
110: (8)               path_arc: float = 0,
111: (8)               path_arc_axis: np.ndarray = OUT,
112: (8)               path_arc_centers: np.ndarray = None,
113: (8)               replace_mobject_with_target_in_scene: bool = False,
114: (8)               **kwargs,
115: (4)           ) -> None:
116: (8)               self.path_arc_axis: np.ndarray = path_arc_axis
117: (8)               self.path_arc_centers: np.ndarray = path_arc_centers
118: (8)               self.path_arc: float = path_arc
119: (8)               if path_func is not None:
120: (12)                   self.path_func: Callable = path_func
121: (8)               elif self.path_arc_centers is not None:
122: (12)                   self.path_func = path_along_circles(
123: (16)                       path_arc,
124: (16)                       self.path_arc_centers,
125: (16)                       self.path_arc_axis,
126: (12)                   )
127: (8)               self.replace_mobject_with_target_in_scene: bool = (
128: (12)                   replace_mobject_with_target_in_scene
129: (8)               )
130: (8)               self.target_mobject: Mobject = (
131: (12)                   target_mobject if target_mobject is not None else Mobject()
132: (8)               )
133: (8)               super().__init__(mobject, **kwargs)
134: (4)           @property
135: (4)           def path_arc(self) -> float:

```

```

136: (8)             return self._path_arc
137: (4)             @path_arc.setter
138: (4)             def path_arc(self, path_arc: float) -> None:
139: (8)                 self._path_arc = path_arc
140: (8)                 self._path_func = path_along_arc(
141: (12)                     arc_angle=self._path_arc,
142: (12)                     axis=self.path_arc_axis,
143: (8)                 )
144: (4)             @property
145: (4)             def path_func(
146: (8)                 self,
147: (4)             ) -> Callable[
148: (8)                 [Iterable[np.ndarray], Iterable[np.ndarray], float],
149: (8)                 Iterable[np.ndarray],
150: (4)             ]:
151: (8)                 return self._path_func
152: (4)             @path_func.setter
153: (4)             def path_func(
154: (8)                 self,
155: (8)                 path_func: Callable[
156: (12)                     [Iterable[np.ndarray], Iterable[np.ndarray], float],
157: (12)                     Iterable[np.ndarray],
158: (8)                 ],
159: (4)             ) -> None:
160: (8)                 if path_func is not None:
161: (12)                     self._path_func = path_func
162: (4)             def begin(self) -> None:
163: (8)                 self.target_mobject = self.create_target()
164: (8)                 self.target_copy = self.target_mobject.copy()
165: (8)                 if config.renderer == RendererType.OPENGL:
166: (12)                     self.mobject.align_data_and_family(self.target_copy)
167: (8)                 else:
168: (12)                     self.mobject.align_data(self.target_copy)
169: (8)                 super().begin()
170: (4)             def create_target(self) -> Mobject:
171: (8)                 return self.target_mobject
172: (4)             def clean_up_from_scene(self, scene: Scene) -> None:
173: (8)                 super().clean_up_from_scene(scene)
174: (8)                 if self.replace_mobject_with_target_in_scene:
175: (12)                     scene.replace(self.mobject, self.target_mobject)
176: (4)             def get_all_mobjects(self) -> Sequence[Mobject]:
177: (8)                 return [
178: (12)                     self.mobject,
179: (12)                     self.starting_mobject,
180: (12)                     self.target_mobject,
181: (12)                     self.target_copy,
182: (8)                 ]
183: (4)             def get_all_families_zipped(self) -> Iterable[tuple]: # more precise
typing?
184: (8)                 mobs = [
185: (12)                     self.mobject,
186: (12)                     self.starting_mobject,
187: (12)                     self.target_copy,
188: (8)                 ]
189: (8)                 if config.renderer == RendererType.OPENGL:
190: (12)                     return zip(*(mob.get_family() for mob in mobs))
191: (8)                     return zip(*(mob.family_members_with_points() for mob in mobs))
192: (4)             def interpolate_submobject(
193: (8)                 self,
194: (8)                 submobject: Mobject,
195: (8)                 starting_submobject: Mobject,
196: (8)                 target_copy: Mobject,
197: (8)                 alpha: float,
198: (4)             ) -> Transform:
199: (8)                 submobject.interpolate(starting_submobject, target_copy, alpha,
self.path_func)
200: (8)                     return self
201: (0)             class ReplacementTransform(Transform):
202: (4)                 """Replaces and morphs a mobject into a target mobject.

```

```

203: (4)             Parameters
204: (4)             -----
205: (4)             mobject
206: (8)             The starting :class:`~.Mobject`.
207: (4)             target_mobject
208: (8)             The target :class:`~.Mobject`.
209: (4)             kwargs
210: (8)             Further keyword arguments that are passed to :class:`Transform`.
211: (4)             Examples
212: (4)             -----
213: (4)             .. manim:: ReplacementTransformOrTransform
214: (8)             :quality: low
215: (8)             class ReplacementTransformOrTransform(Scene):
216: (12)             def construct(self):
217: (16)                 r_transform = VGroup(*[Integer(i) for i in range(1,4)])
218: (16)                 text_1 = Text("ReplacementTransform", color=RED)
219: (16)                 r_transform.add(text_1)
220: (16)                 transform = VGroup(*[Integer(i) for i in range(4,7)])
221: (16)                 text_2 = Text("Transform", color=BLUE)
222: (16)                 transform.add(text_2)
223: (16)                 ints = VGroup(r_transform, transform)
224: (16)                 texts = VGroup(text_1, text_2).scale(0.75)
225: (16)                 r_transform.arrange(direction=UP, buff=1)
226: (16)                 transform.arrange(direction=UP, buff=1)
227: (16)                 ints.arrange(buff=2)
228: (16)                 self.add(ints, texts)
229: (16)                 self.play(ReplacementTransform(r_transform[0],
r_transform[1]))
230: (16)                 self.play(ReplacementTransform(r_transform[1],
r_transform[2]))
231: (16)                 self.play(Transform(transform[0], transform[1]))
232: (16)                 self.play(Transform(transform[1], transform[2]))
233: (16)                 self.wait()
234: (4)             """
235: (4)             def __init__(self, mobject: Mobject, target_mobject: Mobject, **kwargs) ->
None:
236: (8)                 super().__init__(
237: (12)                     mobject, target_mobject,
replace_mobject_with_target_in_scene=True, **kwargs
238: (8)                 )
239: (0)             class TransformFromCopy(Transform):
240: (4)             """
241: (4)             Performs a reversed Transform
242: (4)             """
243: (4)             def __init__(self, mobject: Mobject, target_mobject: Mobject, **kwargs) ->
None:
244: (8)                 super().__init__(target_mobject, mobject, **kwargs)
245: (4)                 def interpolate(self, alpha: float) -> None:
246: (8)                     super().interpolate(1 - alpha)
247: (0)             class ClockwiseTransform(Transform):
248: (4)             """Transforms the points of a mobject along a clockwise oriented arc.
See also
-----
251: (4)             :class:`.Transform`, :class:`.CounterclockwiseTransform`
252: (4)             Examples
253: (4)             -----
254: (4)             .. manim:: ClockwiseExample
255: (8)                 class ClockwiseExample(Scene):
256: (12)                 def construct(self):
257: (16)                     dl, dr = Dot(), Dot()
258: (16)                     sl, sr = Square(), Square()
259: (16)                     VGroup(dl, sl).arrange(DOWN).shift(2*LEFT)
260: (16)                     VGroup(dr, sr).arrange(DOWN).shift(2*RIGHT)
261: (16)                     self.add(dl, dr)
262: (16)                     self.wait()
263: (16)                     self.play(
264: (20)                         ClockwiseTransform(dl, sl),
265: (20)                         Transform(dr, sr)
266: (16)                     )

```

```

267: (16)                         self.wait()
268: (4)
269: (4)
270: (8)
271: (8)
272: (8)
273: (8)
274: (8)
275: (4)
276: (8)
277: (0)
278: (4)
279: (4)
280: (4)
281: (4)
282: (4)
283: (4)
284: (4)
285: (8)
286: (12)
287: (16)
num_decimal_places=3), DecimalNumber(number=1.618, num_decimal_places=3))
288: (16)
289: (16)
290: (16)
num_decimal_places=3), DecimalNumber(number=3.141, num_decimal_places=3))
291: (16)
292: (16)
293: (16)
294: (16)
295: (16)
296: (16)
297: (16)
298: (16)
299: (16)
c_transform[1]))
300: (16)
301: (4)
302: (4)
303: (8)
304: (8)
305: (8)
306: (8)
307: (8)
308: (4)
309: (8)
310: (0)
311: (4)
312: (4)
:attr:`target`
313: (4)
the attribute,
314: (4)
mobject
315: (4)
316: (4)
317: (4)
318: (4)
319: (8)
320: (12)
321: (16)
322: (16)
323: (16)
324: (16)
325: (16)
326: (16)
327: (4)
328: (4)
def __init__(self,
             mobject,
             target_mobject,
             path_arc = -np.pi,
             **kwargs,
            ) -> None:
    super().__init__(mobject, target_mobject, path_arc=path_arc, **kwargs)
class Counter-clockwiseTransform(Transform):
    """Transforms the points of a mobject along a counter-clockwise oriented
arc.
See also
-----
:class:`.Transform` , :class:`.ClockwiseTransform`
Examples
-----
.. manim:: Counter-clockwiseTransform_vs_Transform
    class Counter-clockwiseTransform_vs_Transform(Scene):
        def construct(self):
            c_transform = VGroup(DecimalNumber(number=3.141,
num_decimal_places=3), DecimalNumber(number=1.618, num_decimal_places=3))
            text_1 = Text("Counter-clockwiseTransform", color=RED)
            c_transform.add(text_1)
            transform = VGroup(DecimalNumber(number=1.618,
num_decimal_places=3), DecimalNumber(number=3.141, num_decimal_places=3))
            text_2 = Text("Transform", color=BLUE)
            transform.add(text_2)
            ints = VGroup(c_transform, transform)
            texts = VGroup(text_1, text_2).scale(0.75)
            c_transform.arrange(direction=UP, buff=1)
            transform.arrange(direction=UP, buff=1)
            ints.arrange(buff=2)
            self.add(ints, texts)
            self.play(Counter-clockwiseTransform(c_transform[0],
c_transform[1]))
        self.play(Transform(transform[0], transform[1]))
    """
def __init__(self,
             mobject,
             target_mobject,
             path_arc = np.pi,
             **kwargs,
            ) -> None:
    super().__init__(mobject, target_mobject, path_arc=path_arc, **kwargs)
class MoveToTarget(Transform):
    """Transforms a mobject to the mobject stored in its ``target`` attribute.
After calling the :meth:`~.Mobject.generate_target` method, the
attribute of the mobject is populated with a copy of it. After modifying
playing the :class:`.MoveToTarget` animation transforms the original
into the modified one stored in the :attr:`target` attribute.
Examples
-----
.. manim:: MoveToTargetExample
    class MoveToTargetExample(Scene):
        def construct(self):
            c = Circle()
            c.generate_target()
            c.target.set_fill(color=GREEN, opacity=0.5)
            c.target.shift(2*RIGHT + UP).scale(0.5)
            self.add(c)
            self.play(MoveToTarget(c))
    """
def __init__(self, mobject: Mobject, **kwargs) -> None:

```

```

329: (8)             self.check_validity_of_input(mobject)
330: (8)             super().__init__(mobject, mobject.target, **kwargs)
331: (4)         def check_validity_of_input(self, mobject: Mobject) -> None:
332: (8)             if not hasattr(mobject, "target"):
333: (12)                 raise ValueError(
334: (16)                     "MoveToTarget called on mobject" "without attribute 'target'", )
335: (12)
336: (0)     class _MethodAnimation(MoveToTarget):
337: (4)         def __init__(self, mobject, methods):
338: (8)             self.methods = methods
339: (8)             super().__init__(mobject)
340: (4)         def finish(self) -> None:
341: (8)             for method, method_args, method_kwargs in self.methods:
342: (12)                 method.__func__(self.mobject, *method_args, **method_kwargs)
343: (8)             super().finish()
344: (0)     class ApplyMethod(Transform):
345: (4)         """Animates a mobject by applying a method.
346: (4)         Note that only the method needs to be passed to this animation,
347: (4)         it is not required to pass the corresponding mobject. Furthermore,
348: (4)         this animation class only works if the method returns the modified
349: (4)         mobject.
350: (4)         Parameters
351: (4)         -----
352: (4)         method
353: (8)             The method that will be applied in the animation.
354: (4)         args
355: (8)             Any positional arguments to be passed when applying the method.
356: (4)         kwargs
357: (8)             Any keyword arguments passed to :class:`~.Transform` .
358: (4)         """
359: (4)         def __init__(
360: (8)             self, method: Callable, *args, **kwargs
361: (4)         ) -> None: # method typing (we want to specify Mobject method)? for args?
362: (8)             self.check_validity_of_input(method)
363: (8)             self.method = method
364: (8)             self.method_args = args
365: (8)             super().__init__(method.__self__, **kwargs)
366: (4)         def check_validity_of_input(self, method: Callable) -> None:
367: (8)             if not inspect.ismethod(method):
368: (12)                 raise ValueError(
369: (16)                     "Whoops, looks like you accidentally invoked "
370: (16)                     "the method you want to animate",
371: (12)
372: (8)                     )
373: (4)             assert isinstance(method.__self__, (Mobject, OpenGLMobject))
374: (8)         def create_target(self) -> Mobject:
375: (8)             method = self.method
376: (8)             args = list(self.method_args)
377: (12)             if len(args) > 0 and isinstance(args[-1], dict):
378: (8)                 method_kwargs = args.pop()
379: (12)             else:
380: (8)                 method_kwargs = {}
381: (8)             target = method.__self__.copy()
382: (8)             method.__func__(target, *args, **method_kwargs)
383: (8)             return target
383: (0)     class ApplyPointwiseFunction(ApplyMethod):
384: (4)         """Animation that applies a pointwise function to a mobject.
385: (4)         Examples
386: (4)         -----
387: (4)         .. manim:: WarpSquare
388: (8)             :quality: low
389: (8)             class WarpSquare(Scene):
390: (12)                 def construct(self):
391: (16)                     square = Square()
392: (16)                     self.play(
393: (20)                         ApplyPointwiseFunction(
394: (24)                             lambda point:
complex_to_R3(np.exp(R3_to_complex(point))), square
395: (20)
396: (16)
)

```

```

12/20/24, 4:24 AM manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
397: (16)                         self.wait()
398: (4)
399: (4)             """
400: (8)             def __init__(self,
401: (8)                 function: types.MethodType,
402: (8)                 mobject: Mobject,
403: (8)                 run_time: float = DEFAULT_POINTWISE_FUNCTION_RUN_TIME,
404: (8)                 **kwargs,
405: (4)             ) -> None:
406: (8)                 super().__init__(mobject.apply_function, function, run_time=run_time,
**kwargs)
407: (0)             class ApplyPointwiseFunctionToCenter(ApplyPointwiseFunction):
408: (4)                 def __init__(self, function: types.MethodType, mobject: Mobject, **kwargs)
-> None:
409: (8)                     self.function = function
410: (8)                     super().__init__(mobject.move_to, **kwargs)
411: (4)             def begin(self) -> None:
412: (8)                 self.method_args = [self.function(self.mobject.get_center())]
413: (8)                 super().begin()
414: (0)             class FadeToColor(ApplyMethod):
415: (4)                 """Animation that changes color of a mobject.
416: (4)             Examples
417: (4)             -----
418: (4)             .. manim:: FadeToColorExample
419: (8)                 class FadeToColorExample(Scene):
420: (12)                     def construct(self):
421: (16)                         self.play(FadeToColor(Text("Hello World!"), color=RED))
422: (4)                     """
423: (4)             def __init__(self, mobject: Mobject, color: str, **kwargs) -> None:
424: (8)                 super().__init__(mobject.set_color, color, **kwargs)
425: (0)             class ScaleInPlace(ApplyMethod):
426: (4)                 """Animation that scales a mobject by a certain factor.
427: (4)             Examples
428: (4)             -----
429: (4)             .. manim:: ScaleInPlaceExample
430: (8)                 class ScaleInPlaceExample(Scene):
431: (12)                     def construct(self):
432: (16)                         self.play(ScaleInPlace(Text("Hello World!"), 2))
433: (4)                     """
434: (4)             def __init__(self, mobject: Mobject, scale_factor: float, **kwargs) ->
None:
435: (8)                 super().__init__(mobject.scale, scale_factor, **kwargs)
436: (0)             class ShrinkToCenter(ScaleInPlace):
437: (4)                 """Animation that makes a mobject shrink to center.
438: (4)             Examples
439: (4)             -----
440: (4)             .. manim:: ShrinkToCenterExample
441: (8)                 class ShrinkToCenterExample(Scene):
442: (12)                     def construct(self):
443: (16)                         self.play(ShrinkToCenter(Text("Hello World!")))
444: (4)                     """
445: (4)             def __init__(self, mobject: Mobject, **kwargs) -> None:
446: (8)                 super().__init__(mobject, 0, **kwargs)
447: (0)             class Restore(ApplyMethod):
448: (4)                 """Transforms a mobject to its last saved state.
449: (4)                 To save the state of a mobject, use the :meth:`~.Mobject.save_state` method.
450: (4)
451: (4)
452: (4)             Examples
453: (4)             -----
454: (8)             .. manim:: RestoreExample
455: (12)                 class RestoreExample(Scene):
456: (16)                     def construct(self):
457: (16)                         s = Square()
458: (16)                         s.save_state()
459: (16)                         self.play(FadeIn(s))
460: (16)                         self.play(s.animate.set_color(PURPLE).set_opacity(0.5).shift(2*LEFT).scale(3))
461: (16)                         self.play(s.animate.shift(5*DOWN).rotate(PI/4))
462: (16)                         self.wait()

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
461: (16)                      self.play(Restore(s), run_time=2)
462: (4)          """
463: (4)          def __init__(self, mobject: Mobject, **kwargs) -> None:
464: (8)              super().__init__(mobject.restore, **kwargs)
465: (0)      class ApplyFunction(Transform):
466: (4)          def __init__(self, function: types.MethodType, mobject: Mobject, **kwargs):
-> None:
467: (8)              self.function = function
468: (8)              super().__init__(mobject, **kwargs)
469: (4)          def create_target(self) -> Any:
470: (8)              target = self.function(self.mobject.copy())
471: (8)              if not isinstance(target, (Mobject, OpenGLObject)):
472: (12)                  raise TypeError(
473: (16)                      "Functions passed to ApplyFunction must return object of type
Mobject",
474: (12)                  )
475: (8)              return target
476: (0)      class ApplyMatrix(ApplyPointwiseFunction):
477: (4)          """Applies a matrix transform to an mobject.
478: (4)          Parameters
479: (4)          -----
480: (4)          matrix
481: (8)              The transformation matrix.
482: (4)          mobject
483: (8)              The :class:`~.Mobject`.
484: (4)          about_point
485: (8)              The origin point for the transform. Defaults to ``ORIGIN``.
486: (4)          kwargs
487: (8)              Further keyword arguments that are passed to
:class:`ApplyPointwiseFunction`.
488: (4)          Examples
489: (4)          -----
490: (4)          .. manim:: ApplyMatrixExample
491: (8)              class ApplyMatrixExample(Scene):
492: (12)                  def construct(self):
493: (16)                      matrix = [[1, 1], [0, 2/3]]
494: (16)                      self.play(ApplyMatrix(matrix, Text("Hello World!")),
ApplyMatrix(matrix, NumberPlane())))
495: (4)          """
496: (4)          def __init__(
497: (8)              self,
498: (8)              matrix: np.ndarray,
499: (8)              mobject: Mobject,
500: (8)              about_point: np.ndarray = ORIGIN,
501: (8)              **kwargs,
502: (4)          ) -> None:
503: (8)              matrix = self.initialize_matrix(matrix)
504: (8)              def func(p):
505: (12)                  return np.dot(p - about_point, matrix.T) + about_point
506: (8)              super().__init__(func, mobject, **kwargs)
507: (4)          def initialize_matrix(self, matrix: np.ndarray) -> np.ndarray:
508: (8)              matrix = np.array(matrix)
509: (8)              if matrix.shape == (2, 2):
510: (12)                  new_matrix = np.identity(3)
511: (12)                  new_matrix[:2, :2] = matrix
512: (12)                  matrix = new_matrix
513: (8)              elif matrix.shape != (3, 3):
514: (12)                  raise ValueError("Matrix has bad dimensions")
515: (8)              return matrix
516: (0)      class ApplyComplexFunction(ApplyMethod):
517: (4)          def __init__(self, function: types.MethodType, mobject: Mobject, **kwargs):
-> None:
518: (8)              self.function = function
519: (8)              method = mobject.apply_complex_function
520: (8)              super().__init__(method, function, **kwargs)
521: (4)          def __init_path_func(self) -> None:
522: (8)              func1 = self.function(complex(1))
523: (8)              self.path_arc = np.log(func1).imag
524: (8)              super().__init_path_func()
```

```

525: (0)         class CyclicReplace(Transform):
526: (4)             """An animation moving mobjects cyclically.
527: (4)             In particular, this means: the first mobject takes the place
528: (4)             of the second mobject, the second one takes the place of
529: (4)             the third mobject, and so on. The last mobject takes the
530: (4)             place of the first one.
531: (4)             Parameters
532: (4)                 -----
533: (4)             mobjects
534: (8)                 List of mobjects to be transformed.
535: (4)             path_arc
536: (8)                 The angle of the arc (in radians) that the mobjects will follow to
reach
537: (8)                     their target.
538: (4)             kwargs
539: (8)                 Further keyword arguments that are passed to :class:`.Transform`.
540: (4)             Examples
541: (4)                 -----
542: (4)             .. manim :: CyclicReplaceExample
543: (8)             class CyclicReplaceExample(Scene):
544: (12)                 def construct(self):
545: (16)                     group = VGroup(Square(), Circle(), Triangle(), Star())
546: (16)                     group.arrange(RIGHT)
547: (16)                     self.add(group)
548: (16)                     for _ in range(4):
549: (20)                         self.play(CyclicReplace(*group))
550: (4)                     """
551: (4)                 def __init__(
552: (8)                     self, *mobjects: Mobject, path_arc: float = 90 * DEGREES, **kwargs
553: (4)                 ) -> None:
554: (8)                     self.group = Group(*mobjects)
555: (8)                     super().__init__(self.group, path_arc=path_arc, **kwargs)
556: (4)                 def create_target(self) -> Group:
557: (8)                     target = self.group.copy()
558: (8)                     cycled_targets = [target[-1], *target[:-1]]
559: (8)                     for m1, m2 in zip(cycled_targets, self.group):
560: (12)                         m1.move_to(m2)
561: (8)                     return target
562: (0)             class Swap(CyclicReplace):
563: (4)                 pass # Renaming, more understandable for two entries
564: (0)             class TransformAnimations(Transform):
565: (4)                 def __init__(
566: (8)                     self,
567: (8)                     start_anim: Animation,
568: (8)                     end_anim: Animation,
569: (8)                     rate_func: Callable = squish_rate_func(smooth),
570: (8)                     **kwargs,
571: (4)                 ) -> None:
572: (8)                     self.start_anim = start_anim
573: (8)                     self.end_anim = end_anim
574: (8)                     if "run_time" in kwargs:
575: (12)                         self.run_time = kwargs.pop("run_time")
576: (8)                     else:
577: (12)                         self.run_time = max(start_anim.run_time, end_anim.run_time)
578: (8)                     for anim in start_anim, end_anim:
579: (12)                         anim.set_run_time(self.run_time)
580: (8)                     if (
581: (12)                         start_anim.starting_mobject is not None
582: (12)                         and end_anim.starting_mobject is not None
583: (12)                         and start_anim.starting_mobject.get_num_points()
584: (12)                         != end_anim.starting_mobject.get_num_points()
585: (8)                     ):
586: (12)                         start_anim.starting_mobject.align_data(end_anim.starting_mobject)
587: (12)                         for anim in start_anim, end_anim:
588: (16)                             if isinstance(anim, Transform) and anim.starting_mobject is
not None:
589: (20)                                 anim.starting_mobject.align_data(anim.target_mobject)
590: (8)                     super().__init__(
591: (12)                         start_anim.mobject, end_anim.mobject, rate_func=rate_func,

```

```

**kwargs
592: (8) )
593: (8)     start_anim.mobject = self.starting_mobject
594: (8)     end_anim.mobject = self.target_mobject
595: (4) def interpolate(self, alpha: float) -> None:
596: (8)     self.start_anim.interpolate(alpha)
597: (8)     self.end_anim.interpolate(alpha)
598: (8)     super().interpolate(alpha)
599: (0) class FadeTransform(Transform):
600: (4)     """Fades one mobject into another.
601: (4)     Parameters
602: (4)     -----
603: (4)     mobject
604: (8)         The starting :class:`~.Mobject`.
605: (4)     target_mobject
606: (8)         The target :class:`~.Mobject`.
607: (4)     stretch
608: (8)         Controls whether the target :class:`~.Mobject` is stretched during
609: (8)         the animation. Default: ``True``.
610: (4)     dim_to_match
611: (8)         If the target mobject is not stretched automatically, this allows
612: (8)         to adjust the initial scale of the target :class:`~.Mobject` while
613: (8)         it is shifted in. Setting this to 0, 1, and 2, respectively,
614: (8)         matches the length of the target with the length of the starting
615: (8)         :class:`~.Mobject` in x, y, and z direction, respectively.
616: (4)     kwargs
617: (8)         Further keyword arguments are passed to the parent class.
618: (4) Examples
619: (4) -----
620: (4) .. manim:: DifferentFadeTransforms
621: (8)     class DifferentFadeTransforms(Scene):
622: (12)         def construct(self):
623: (16)             starts = [Rectangle(width=4, height=1) for _ in range(3)]
624: (16)             VGroup(*starts).arrange(DOWN, buff=1).shift(3*LEFT)
625: (16)             targets = [Circle(fill_opacity=1).scale(0.25) for _ in
range(3)]
626: (16)             VGroup(*targets).arrange(DOWN, buff=1).shift(3*RIGHT)
627: (16)             self.play(*[FadeIn(s) for s in starts])
628: (16)             self.play(
629: (20)                 FadeTransform(starts[0], targets[0], stretch=True),
630: (20)                 FadeTransform(starts[1], targets[1], stretch=False,
dim_to_match=0),
631: (20)                 FadeTransform(starts[2], targets[2], stretch=False,
dim_to_match=1)
632: (16)             )
633: (16)             self.play(*[FadeOut(mobj) for mobj in self.mobjects])
634: (4) """
635: (4)     def __init__(self, mobject, target_mobject, stretch=True, dim_to_match=1,
**kwargs):
636: (8)         self.to_add_on_completion = target_mobject
637: (8)         self.stretch = stretch
638: (8)         self.dim_to_match = dim_to_match
639: (8)         mobject.save_state()
640: (8)         if config.renderer == RendererType.OPENGL:
641: (12)             group = OpenGLGroup(mobject, target_mobject.copy())
642: (8)         else:
643: (12)             group = Group(mobject, target_mobject.copy())
644: (8)             super().__init__(group, **kwargs)
645: (4)     def begin(self):
646: (8)         """Initial setup for the animation.
647: (8)         The mobject to which this animation is bound is a group consisting of
648: (8)         both the starting and the ending mobject. At the start, the ending
649: (8)         mobject replaces the starting mobject (and is completely faded). In
the
650: (8)         end, it is set to be the other way around.
651: (8) """
652: (8)         self.ending_mobject = self.mobject.copy()
653: (8)         Animation.begin(self)
654: (8)         start, end = self.starting_mobject, self.ending_mobject

```

```

655: (8)                 for m0, m1 in ((start[1], start[0]), (end[0], end[1])):
656: (12)                   self.ghost_to(m0, m1)
657: (4)             def ghost_to(self, source, target):
658: (8)                 """Replaces the source by the target and sets the opacity to 0.
659: (8)                 If the provided target has no points, and thus a location of [0, 0, 0]
660: (8)                 the source will simply fade out where it currently is.
661: (8)
662: (8)                 if target.get_num_points() or target.submobjects:
663: (12)                     source.replace(target, stretch=self.stretch,
dim_to_match=self.dim_to_match)
664: (8)                         source.set_opacity(0)
665: (4)             def get_all_mobjects(self) -> Sequence[Mobject]:
666: (8)                 return [
667: (12)                     self.mobject,
668: (12)                     self.starting_mobject,
669: (12)                     self.ending_mobject,
670: (8)                 ]
671: (4)             def get_all_families_zipped(self):
672: (8)                 return Animation.get_all_families_zipped(self)
673: (4)             def clean_up_from_scene(self, scene):
674: (8)                 Animation.clean_up_from_scene(self, scene)
675: (8)                 scene.remove(self.mobject)
676: (8)                 self.mobject[0].restore()
677: (8)                 scene.add(self.to_add_on_completion)
678: (0)         class FadeTransformPieces(FadeTransform):
679: (4)             """Fades submobjects of one mobject into submobjects of another one.
680: (4)             See also
681: (4)             -----
682: (4)             :class:`~.FadeTransform`
683: (4)             Examples
684: (4)             -----
685: (4)             .. manim:: FadeTransformSubmobjects
686: (8)                 class FadeTransformSubmobjects(Scene):
687: (12)                     def construct(self):
688: (16)                         src = VGroup(Square(), Circle()).shift(LEFT + UP)
689: (16)                         src.shift(3*LEFT + 2*UP)
690: (16)                         src_copy = src.copy().shift(4*DOWN)
691: (16)                         target = VGroup(Circle(), Triangle()).shift(RIGHT + DOWN)
692: (16)                         target.shift(3*RIGHT + 2*UP)
693: (16)                         target_copy = target.copy().shift(4*DOWN)
694: (16)                         self.play(FadeIn(src), FadeIn(src_copy))
695: (16)                         self.play(
696: (20)                             FadeTransform(src, target),
697: (20)                             FadeTransformPieces(src_copy, target_copy)
698: (16)                         )
699: (16)                         self.play(*[FadeOut(mobj) for mobj in self.mobjects])
700: (4)             """
701: (4)             def begin(self):
702: (8)                 self.mobject[0].align_submobjects(self.mobject[1])
703: (8)                 super().begin()
704: (4)             def ghost_to(self, source, target):
705: (8)                 """Replaces the source submobjects by the target submobjects and sets
706: (8)                 the opacity to 0.
707: (8)
708: (8)                 for sm0, sm1 in zip(source.get_family(), target.get_family()):
709: (12)                     super().ghost_to(sm0, sm1)

-----

```

## File 27 - specialized.py:

```

1: (0)             from __future__ import annotations
2: (0)             __all__ = ["Broadcast"]
3: (0)             from typing import Any, Sequence
4: (0)             from manim.animation.transform import Restore
5: (0)             from ..constants import *
6: (0)             from .composition import LaggedStart
7: (0)             class Broadcast(LaggedStart):
8: (4)                 """Broadcast a mobject starting from an ``initial_width``, up to the

```

actual size of the mobject.

9: (4) Parameters  
10: (4) -----  
11: (4) mobject  
12: (8) The mobject to be broadcast.  
13: (4) focal\_point  
14: (8) The center of the broadcast, by default ORIGIN.  
15: (4) n\_mobs  
16: (8) The number of mobjects that emerge from the focal point, by default 5.  
17: (4) initial\_opacity  
18: (8) The starting stroke opacity of the mobjects emitted from the broadcast, by default 1.  
19: (4) final\_opacity  
20: (8) The final stroke opacity of the mobjects emitted from the broadcast, by default 0.  
21: (4) initial\_width  
22: (8) The initial width of the mobjects, by default 0.0.  
23: (4) remover  
24: (8) Whether the mobjects should be removed from the scene after the animation, by default True.  
25: (4) lag\_ratio  
26: (8) The time between each iteration of the mobject, by default 0.2.  
27: (4) run\_time  
28: (8) The total duration of the animation, by default 3.  
29: (4) kwargs  
30: (8) Additional arguments to be passed to :class:`~.LaggedStart`.  
31: (4) Examples  
32: (4) -----  
33: (4) .. manim:: BroadcastExample  
34: (8) class BroadcastExample(Scene):  
35: (12) def construct(self):  
36: (16) mob = Circle(radius=4, color=TEAL\_A)  
37: (16) self.play(Broadcast(mob))  
38: (4) """  
39: (4) def \_\_init\_\_(  
40: (8) self,  
41: (8) mobject,  
42: (8) focal\_point: Sequence[float] = ORIGIN,  
43: (8) n\_mobs: int = 5,  
44: (8) initial\_opacity: float = 1,  
45: (8) final\_opacity: float = 0,  
46: (8) initial\_width: float = 0.0,  
47: (8) remover: bool = True,  
48: (8) lag\_ratio: float = 0.2,  
49: (8) run\_time: float = 3,  
50: (8) \*\*kwargs: Any,  
51: (4) ):  
52: (8) self.focal\_point = focal\_point  
53: (8) self.n\_mobs = n\_mobs  
54: (8) self.initial\_opacity = initial\_opacity  
55: (8) self.final\_opacity = final\_opacity  
56: (8) self.initial\_width = initial\_width  
57: (8) anims = []  
58: (8) if mobject.fill\_opacity:  
59: (12) fill\_o = True  
60: (8) else:  
61: (12) fill\_o = False  
62: (8) for \_ in range(self.n\_mobs):  
63: (12) mob = mobject.copy()  
64: (12) if fill\_o:  
65: (16) mob.set\_opacity(self.final\_opacity)  
66: (12) else:  
67: (16) mob.set\_stroke(opacity=self.final\_opacity)  
68: (12) mob.move\_to(self.focal\_point)  
69: (12) mob.save\_state()  
70: (12) mob.set(width=self.initial\_width)  
71: (12) if fill\_o:  
72: (16) mob.set\_opacity(self.initial\_opacity)  
73: (12) else:

```

74: (16)                         mob.set_stroke(opacity=self.initial_opacity)
75: (12)                         anims.append(Restore(mob, remover=remover))
76: (8)                          super().__init__(*anim, run_time=run_time, lag_ratio=lag_ratio,
**kwargs)
-----
```

File 28 - multi\_camera.py:

```

1: (0)             """A camera supporting multiple perspectives."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["MultiCamera"]
4: (0)             from manim.mobject.types.image_mobject import ImageMobject
5: (0)             from ..camera.moving_camera import MovingCamera
6: (0)             from ..utils.iterables import list_difference_update
7: (0)             class MultiCamera(MovingCamera):
8: (4)                 """Camera Object that allows for multiple perspectives."""
9: (4)                 def __init__(
10: (8)                     self,
11: (8)                     image_mobjects_from_cameras: ImageMobject | None = None,
12: (8)                     allow_cameras_to_capture_their_own_display=False,
13: (8)                     **kwargs,
14: (4)                 ):
15: (8)                     """Initialises the MultiCamera
16: (8)                     Parameters
17: (8)                     -----
18: (8)                     image_mobjects_from_cameras
19: (8)                     kwargs
20: (12)                     Any valid keyword arguments of MovingCamera.
21: (8)                     """
22: (8)                     self.image_mobjects_from_cameras = []
23: (8)                     if image_mobjects_from_cameras is not None:
24: (12)                         for imfc in image_mobjects_from_cameras:
25: (16)                             self.add_image_mobject_from_camera(imfc)
26: (8)                     self.allow_cameras_to_capture_their_own_display = (
27: (12)                         allow_cameras_to_capture_their_own_display
28: (8)                     )
29: (8)                     super().__init__(**kwargs)
30: (4)             def add_image_mobject_from_camera(self, image_mobject_from_camera:
ImageMobject):
31: (8)                 """Adds an ImageMobject that's been obtained from the camera
32: (8)                 into the list ``self.image_mobject_from_cameras``
33: (8)                 Parameters
34: (8)                 -----
35: (8)                 image_mobject_from_camera
36: (12)                 The ImageMobject to add to self.image_mobject_from_cameras
37: (8)                 """
38: (8)                 imfc = image_mobject_from_camera
39: (8)                 assert isinstance(imfc.camera, MovingCamera)
40: (8)                 self.image_mobjects_from_cameras.append(imfc)
41: (4)             def update_sub_cameras(self):
42: (8)                 """Reshape sub_camera pixel_arrays"""
43: (8)                 for imfc in self.image_mobjects_from_cameras:
44: (12)                     pixel_height, pixel_width = self.pixel_array.shape[:2]
45: (12)                     imfc.camera.frame_shape = (
46: (16)                         imfc.camera.frame.height,
47: (16)                         imfc.camera.frame.width,
48: (12)                     )
49: (12)                     imfc.camera.reset_pixel_shape(
50: (16)                         int(pixel_height * imfc.height / self.frame_height),
51: (16)                         int(pixel_width * imfc.width / self.frame_width),
52: (12)                     )
53: (4)             def reset(self):
54: (8)                 """Resets the MultiCamera.
55: (8)                 Returns
56: (8)                 -----
57: (8)                 MultiCamera
58: (12)                 The reset MultiCamera
59: (8)                 """
-----
```

```

60: (8)                 for imfc in self.image_mobjects_from_cameras:
61: (12)                   imfc.camera.reset()
62: (8)                   super().reset()
63: (8)                   return self
64: (4)             def capture_mobjects(self, mobjects, **kwargs):
65: (8)                 self.update_sub_cameras()
66: (8)                 for imfc in self.image_mobjects_from_cameras:
67: (12)                     to_add = list(mobjects)
68: (12)                     if not self.allow_cameras_to_capture_their_own_display:
69: (16)                         to_add = list_difference_update(to_add, imfc.get_family())
70: (12)                     imfc.camera.capture_mobjects(to_add, **kwargs)
71: (8)                 super().capture_mobjects(mobjects, **kwargs)
72: (4)             def get_mobjects_indicating_movement(self):
73: (8)                 """Returns all mobjects whose movement implies that the camera
74: (8)                 should think of all other mobjects on the screen as moving
75: (8)                 Returns
76: (8)                 -----
77: (8)                 list
78: (8)                 """
79: (8)                 return [self.frame] + [
80: (12)                     imfc.camera.frame for imfc in self.image_mobjects_from_cameras
81: (8)                 ]

```

---

## File 29 - speedmodifier.py:

```

1: (0)             """Utilities for modifying the speed at which animations are played."""
2: (0)             from __future__ import annotations
3: (0)             import inspect
4: (0)             import types
5: (0)             from typing import TYPE_CHECKING, Callable
6: (0)             from numpy import piecewise
7: (0)             from ..animation.animation import Animation, Wait, prepare_animation
8: (0)             from ..animation.composition import AnimationGroup
9: (0)             from ..mobject.mobject import Mobject, _AnimationBuilder
10: (0)            from ..scene.scene import Scene
11: (0)            if TYPE_CHECKING:
12: (4)                from ..mobject.mobject import Updater
13: (0)            __all__ = ["ChangeSpeed"]
14: (0)            class ChangeSpeed(Animation):
15: (4)                """Modifies the speed of passed animation.
16: (4)                :class:`AnimationGroup` with different ``lag_ratio`` can also be used
17: (4)                which combines multiple animations into one.
18: (4)                The ``run_time`` of the passed animation is changed to modify the speed.
19: (4)                Parameters
20: (4)                -----
21: (4)                anim
22: (8)                    Animation of which the speed is to be modified.
23: (4)                speedinfo
24: (8)                    Contains nodes (percentage of ``run_time``) and its corresponding
speed factor.
25: (4)
26: (8)            rate_func
27: (4)                Overrides ``rate_func`` of passed animation, applied before changing
speed.
28: (4)            Examples
29: (4)            -----
30: (8)            .. manim:: SpeedModifierExample
31: (12)                class SpeedModifierExample(Scene):
32: (16)                    def construct(self):
33: (16)                        a = Dot().shift(LEFT * 4)
34: (16)                        b = Dot().shift(RIGHT * 4)
35: (16)                        self.add(a, b)
36: (20)                        self.play(
37: (24)                            ChangeSpeed(
38: (28)                                AnimationGroup(
39: (28)                                    a.animate(run_time=1).shift(RIGHT * 8),
40: (28)                                    b.animate(run_time=1).shift(LEFT * 8),
41: (24)                                ),

```

```

41: (24)                                     speedinfo={0.3: 1, 0.4: 0.1, 0.6: 0.1, 1: 1},
42: (24)                                     rate_func=linear,
43: (20)                                     )
44: (16)                                     )
45: (4) .. manim:: SpeedModifierUpdaterExample
46: (8)     class SpeedModifierUpdaterExample(Scene):
47: (12)         def construct(self):
48: (16)             a = Dot().shift(LEFT * 4)
49: (16)             self.add(a)
50: (16)             ChangeSpeed.add_updater(a, lambda x, dt: x.shift(RIGHT * 4 *
dt))
51: (16)             self.play(
52: (20)                 ChangeSpeed(
53: (24)                     Wait(2),
54: (24)                     speedinfo={0.4: 1, 0.5: 0.2, 0.8: 0.2, 1: 1},
55: (24)                     affects_speed_updaters=True,
56: (20)                 )
57: (16)             )
58: (4) .. manim:: SpeedModifierUpdaterExample2
59: (8)     class SpeedModifierUpdaterExample2(Scene):
60: (12)         def construct(self):
61: (16)             a = Dot().shift(LEFT * 4)
62: (16)             self.add(a)
63: (16)             ChangeSpeed.add_updater(a, lambda x, dt: x.shift(RIGHT * 4 *
dt))
64: (16)             self.wait()
65: (16)             self.play(
66: (20)                 ChangeSpeed(
67: (24)                     Wait(),
68: (24)                     speedinfo={1: 0},
69: (24)                     affects_speed_updaters=True,
70: (20)                 )
71: (16)             )
72: (4) """
73: (4)     dt = 0
74: (4)     is_changing_dt = False
75: (4)     def __init__(
76: (8)         self,
77: (8)         anim: Animation | _AnimationBuilder,
78: (8)         speedinfo: dict[float, float],
79: (8)         rate_func: Callable[[float], float] | None = None,
80: (8)         affects_speed_updaters: bool = True,
81: (8)         **kwargs,
82: (4)     ) -> None:
83: (8)         if issubclass(type(anim), AnimationGroup):
84: (12)             self.anim = type(anim)(
85: (16)                 *map(self.setup, anim.animations),
86: (16)                 group=anim.group,
87: (16)                 run_time=anim.run_time,
88: (16)                 rate_func=anim.rate_func,
89: (16)                 lag_ratio=anim.lag_ratio,
90: (12)             )
91: (8)         else:
92: (12)             self.anim = self.setup(anim)
93: (8)         if affects_speed_updaters:
94: (12)             assert (
95: (16)                 ChangeSpeed.is_changing_dt is False
96: (12)             ), "Only one animation at a time can play that changes speed (dt)
for ChangeSpeed updaters"
97: (12)             ChangeSpeed.is_changing_dt = True
98: (12)             self.t = 0
99: (8)             self.affects_speed_updaters = affects_speed_updaters
100: (8)             self.rate_func = self.anim.rate_func if rate_func is None else
101: (8)             self.speed_modifier = lambda x, init_speed, final_speed: (
102: (12)                 (final_speed**2 - init_speed**2) * x**2 / 4 + init_speed * x
103: (8)             )
104: (8)             self.f_inv_1 = lambda init_speed, final_speed: 2 / (init_speed +
final_speed)

```

```

105: (8)             if 0 not in speedinfo:
106: (12)             speedinfo[0] = 1
107: (8)             if 1 not in speedinfo:
108: (12)                 speedinfo[1] = sorted(speedinfo.items())[-1][1]
109: (8)             self.speedinfo = dict(sorted(speedinfo.items()))
110: (8)             self.functions = []
111: (8)             self.conditions = []
112: (8)             scaled_total_time = self.get_scaled_total_time()
113: (8)             prevnode = 0
114: (8)             init_speed = self.speedinfo[0]
115: (8)             curr_time = 0
116: (8)             for node, final_speed in list(self.speedinfo.items())[1:]:
117: (12)                 dur = node - prevnode
118: (12)                 def condition(
119: (16)                     t,
120: (16)                     curr_time=curr_time,
121: (16)                     init_speed=init_speed,
122: (16)                     final_speed=final_speed,
123: (16)                     dur=dur,
124: (12)                 ):
125: (16)                     lower_bound = curr_time / scaled_total_time
126: (16)                     upper_bound = (
127: (20)                         curr_time + self.f_inv_1(init_speed, final_speed) * dur
128: (16)                     ) / scaled_total_time
129: (16)                     return lower_bound <= t <= upper_bound
130: (12)                 self.conditions.append(condition)
131: (12)                 def function(
132: (16)                     t,
133: (16)                     curr_time=curr_time,
134: (16)                     init_speed=init_speed,
135: (16)                     final_speed=final_speed,
136: (16)                     dur=dur,
137: (16)                     prevnode=prevnode,
138: (12)                 ):
139: (16)                     return (
140: (20)                         self.speed_modifier(
141: (24)                             (scaled_total_time * t - curr_time) / dur,
142: (24)                             init_speed,
143: (24)                             final_speed,
144: (20)                         )
145: (20)                         * dur
146: (20)                         + prevnode
147: (16)                     )
148: (12)                     self.functions.append(function)
149: (12)                     curr_time += self.f_inv_1(init_speed, final_speed) * dur
150: (12)                     prevnode = node
151: (12)                     init_speed = final_speed
152: (8)                 def func(t):
153: (12)                     if t == 1:
154: (16)                         ChangeSpeed.is_changing_dt = False
155: (12)                     new_t = piecewise(
156: (16)                         self.rate_func(t),
157: (16)                         [condition(self.rate_func(t)) for condition in
self.conditions],
158: (16)                         self.functions,
159: (12)                     )
160: (12)                     if self.affects_speed_updaters:
161: (16)                         ChangeSpeed.dt = (new_t - self.t) * self.anim.run_time
162: (16)                         self.t = new_t
163: (12)                     return new_t
164: (8)                     self.anim.set_rate_func(func)
165: (8)                     super().__init__(
166: (12)                         self.anim.mobject,
167: (12)                         rate_func=self.rate_func,
168: (12)                         run_time=scaled_total_time * self.anim.run_time,
169: (12)                         **kwargs,
170: (8)                     )
171: (4)                     def setup(self, anim):
172: (8)                         if type(anim) is Wait:

```

```

173: (12)                     anim.interpolate = types.MethodType(
174: (16)                         lambda self, alpha: self.rate_func(alpha), anim
175: (12)                     )
176: (8)                     return prepare_animation(anim)
177: (4)             def get_scaled_total_time(self) -> float:
178: (8)                 """The time taken by the animation under the assumption that the
``run_time`` is 1."""
179: (8)                     prevnode = 0
180: (8)                     init_speed = self.speedinfo[0]
181: (8)                     total_time = 0
182: (8)                     for node, final_speed in list(self.speedinfo.items())[1:]:
183: (12)                         dur = node - prevnode
184: (12)                         total_time += dur * self.f_inv_1(init_speed, final_speed)
185: (12)                         prevnode = node
186: (12)                         init_speed = final_speed
187: (8)                     return total_time
188: (4)             @classmethod
189: (4)             def add_updater(
190: (8)                 cls,
191: (8)                 mobject: Mobject,
192: (8)                 update_function: Updater,
193: (8)                 index: int | None = None,
194: (8)                 call_updater: bool = False,
195: (4)             ):
196: (8)                 """This static method can be used to apply speed change to updaters.
197: (8)                 This updater will follow speed and rate function of any
198: (8)                 :class:`.ChangeSpeed` animation that is playing with ``affects_speed_updaters=True``. By
199: (8)                 default, updater functions added via the usual :meth:`.Mobject.add_updater` do not respect the change of animation speed.
200: (8)             Parameters
201: (8)             -----
202: (8)             mobject
203: (8)                 The mobject to which the updater should be attached.
204: (12)             update_function
205: (8)                 The function that is called whenever a new frame is rendered.
206: (12)             index
207: (8)                 The position in the list of the mobject's updaters at which the
208: (12)                 function should be inserted.
209: (12)             call_updater
210: (8)                 If ``True``, calls the update function when attaching it to the
211: (12)                 mobject.
212: (12)             See also
213: (8)             -----
214: (8)             :class:`.ChangeSpeed`
215: (8)             :meth:`.Mobject.add_updater`
216: (8)             """
217: (8)
218: (8)             if "dt" in inspect.signature(update_function).parameters:
219: (12)                 mobject.add_updater(
220: (16)                     lambda mob, dt: update_function(
221: (20)                         mob, ChangeSpeed.dt if ChangeSpeed.is_changing_dt else dt
222: (16)                     ),
223: (16)                     index=index,
224: (16)                     call_updater=call_updater,
225: (12)                 )
226: (8)             else:
227: (12)                 mobject.add_updater(update_function, index=index,
call_updater=call_updater)
228: (4)             def interpolate(self, alpha: float) -> None:
229: (8)                 self.anim.interpolate(alpha)
230: (4)             def update_mobjects(self, dt: float) -> None:
231: (8)                 self.anim.update_mobjects(dt)
232: (4)             def finish(self) -> None:
233: (8)                 ChangeSpeed.is_changing_dt = False
234: (8)                 self.anim.finish()
235: (4)             def begin(self) -> None:
236: (8)                 self.anim.begin()

```

```

237: (4)         def clean_up_from_scene(self, scene: Scene) -> None:
238: (8)             self.anim.clean_up_from_scene(scene)
239: (4)         def _setup_scene(self, scene) -> None:
240: (8)             self.anim._setup_scene(scene)
-----
```

## File 30 - moving\_camera.py:

```

1: (0)         """A camera able to move through a scene.
2: (0)         .. SEEALSO::
3: (4)             :mod:`.moving_camera_scene`
4: (0)         """
5: (0)         from __future__ import annotations
6: (0)         __all__ = ["MovingCamera"]
7: (0)         import numpy as np
8: (0)         from .. import config
9: (0)         from ..camera.camera import Camera
10: (0)        from ..constants import DOWN, LEFT, RIGHT, UP
11: (0)        from ..mobject.frame import ScreenRectangle
12: (0)        from ..mobject.mobject import Mobject
13: (0)        from ..utils.color import WHITE
14: (0)        class MovingCamera(Camera):
15: (4)            """
16: (4)                Stays in line with the height, width and position of it's 'frame', which
17: (4)                is a Rectangle
18: (8)                .. SEEALSO::
19: (4)                    :class:`.MovingCameraScene`
20: (4)            """
21: (8)            def __init__(
22: (8)                self,
23: (8)                frame=None,
24: (8)                fixed_dimension=0, # width
25: (8)                default_frame_stroke_color=WHITE,
26: (8)                default_frame_stroke_width=0,
27: (8)                **kwargs,
28: (8)            ):
29: (4)                """
30: (8)                    Frame is a Mobject, (should almost certainly be a rectangle)
31: (8)                    determining which region of space the camera displays
32: (8)                """
33: (8)                self.fixed_dimension = fixed_dimension
34: (8)                self.default_frame_stroke_color = default_frame_stroke_color
35: (8)                self.default_frame_stroke_width = default_frame_stroke_width
36: (12)               if frame is None:
37: (12)                   frame = ScreenRectangle(height=config["frame_height"])
38: (16)                   frame.set_stroke(
39: (16)                       self.default_frame_stroke_color,
40: (12)                       self.default_frame_stroke_width,
41: (8)                   )
42: (8)                   self.frame = frame
43: (4)                   super().__init__(**kwargs)
44: (4)               @property
45: (8)               def frame_height(self):
46: (8)                   """
47: (8)                       Returns the height of the frame.
48: (8)                   Returns
49: (12)                   -----
50: (8)                   float
51: (8)                   The height of the frame.
52: (4)               """
53: (8)                   return self.frame.height
54: (4)               @property
55: (8)               def frame_width(self):
56: (8)                   """
57: (8)                       Returns the width of the frame
58: (8)                   Returns
59: (8)                   -----
60: (12)                   float
61: (8)                   The width of the frame.
62: (4)               """

```

```

60: (8)                         return self.frame.width
61: (4) @property
62: (4)     def frame_center(self):
63: (8)         """Returns the centerpoint of the frame in cartesian coordinates.
64: (8)         Returns
65: (8)         -----
66: (8)         np.array
67: (12)             The cartesian coordinates of the center of the frame.
68: (8)         """
69: (8)         return self.frame.get_center()
70: (4) @frame_height.setter
71: (4)     def frame_height(self, frame_height: float):
72: (8)         """Sets the height of the frame in MUnits.
73: (8)         Parameters
74: (8)         -----
75: (8)         frame_height
76: (12)             The new frame_height.
77: (8)         """
78: (8)         self.frame.stretch_to_fit_height(frame_height)
79: (4) @frame_width.setter
80: (4)     def frame_width(self, frame_width: float):
81: (8)         """Sets the width of the frame in MUnits.
82: (8)         Parameters
83: (8)         -----
84: (8)         frame_width
85: (12)             The new frame_width.
86: (8)         """
87: (8)         self.frame.stretch_to_fit_width(frame_width)
88: (4) @frame_center.setter
89: (4)     def frame_center(self, frame_center: np.ndarray | list | tuple | Mobject):
90: (8)         """Sets the centerpoint of the frame.
91: (8)         Parameters
92: (8)         -----
93: (8)         frame_center
94: (12)             The point to which the frame must be moved.
95: (12)             If is of type mobject, the frame will be moved to
96: (12)                 the center of that mobject.
97: (8)         """
98: (8)         self.frame.move_to(frame_center)
99: (4)     def capture_mobjects(self, mobjects, **kwargs):
100: (8)         super().capture_mobjects(mobjects, **kwargs)
101: (4)     def get_cached_cairo_context(self, pixel_array):
102: (8)         """
103: (8)             Since the frame can be moving around, the cairo
104: (8)             context used for updating should be regenerated
105: (8)             at each frame. So no caching.
106: (8)         """
107: (8)         return None
108: (4)     def cache_cairo_context(self, pixel_array, ctx):
109: (8)         """
110: (8)             Since the frame can be moving around, the cairo
111: (8)             context used for updating should be regenerated
112: (8)             at each frame. So no caching.
113: (8)         """
114: (8)         pass
115: (4)     def get_mobjects_indicating_movement(self):
116: (8)         """
117: (8)             Returns all mobjects whose movement implies that the camera
118: (8)             should think of all other mobjects on the screen as moving
119: (8)             Returns
120: (8)             -----
121: (8)             list
122: (8)         """
123: (8)             return [self.frame]
124: (4)     def auto_zoom(
125: (8)         self,
126: (8)         mobjects: list[Mobject],
127: (8)         margin: float = 0,
128: (8)         only_mobjects_in_frame: bool = False,

```

```

129: (8)             animate: bool = True,
130: (4)         ):
131: (8)             """Zooms on to a given array of mobjects (or a singular mobject)
132: (8)             and automatically resizes to frame all the mobjects.
133: (8)             .. NOTE::
134: (12)                 This method only works when 2D-objects in the XY-plane are
considered, it
135: (12)                 will not work correctly when the camera has been rotated.
136: (8)
137: (8)
138: (8)
139: (12)             Parameters
140: (8)             -----
141: (12)                 mobjects
142: (8)                     The mobject or array of mobjects that the camera will focus on.
143: (12)                 margin
144: (8)                     The width of the margin that is added to the frame (optional, 0 by
default).
145: (12)                 only_mobjects_in_frame
146: (8)                     If set to ``True``, only allows focusing on mobjects that are
already in frame.
147: (12)                 corresponding animation
148: (8)
149: (12)             Returns
150: (12)             Union[_AnimationBuilder, ScreenRectangle]
151: (8)                 _AnimationBuilder that zooms the camera view to a given list of
position.
152: (8)                 or ScreenRectangle with position and size updated to zoomed
153: (8)
154: (8)
155: (8)
156: (8)
157: (12)             """
158: (16)                 scene_critical_x_left = None
159: (12)                 scene_critical_x_right = None
160: (16)                 scene_critical_y_up = None
161: (12)                 scene_critical_y_down = None
162: (16)                 for m in mobjects:
163: (16)                     if (m == self.frame) or (
164: (16)                         only_mobjects_in_frame and not self.is_in_frame(m)
165: (16)                     ):
166: (12)                         continue
167: (16)                         if scene_critical_x_left is None:
168: (20)                             scene_critical_x_left = m.get_critical_point(LEFT)[0]
169: (16)                             scene_critical_x_right = m.get_critical_point(RIGHT)[0]
170: (20)                             scene_critical_y_up = m.get_critical_point(UP)[1]
171: (16)                             scene_critical_y_down = m.get_critical_point(DOWN)[1]
172: (20)                         else:
173: (16)                             if m.get_critical_point(LEFT)[0] < scene_critical_x_left:
174: (20)                                 scene_critical_x_left = m.get_critical_point(LEFT)[0]
175: (8)                             if m.get_critical_point(RIGHT)[0] > scene_critical_x_right:
176: (20)                                 scene_critical_x_right = m.get_critical_point(RIGHT)[0]
177: (8)                             if m.get_critical_point(UP)[1] > scene_critical_y_up:
178: (20)                                 scene_critical_y_up = m.get_critical_point(UP)[1]
179: (8)                             if m.get_critical_point(DOWN)[1] < scene_critical_y_down:
180: (20)                                 scene_critical_y_down = m.get_critical_point(DOWN)[1]
181: (8)             x = (scene_critical_x_left + scene_critical_x_right) / 2
182: (8)             y = (scene_critical_y_up + scene_critical_y_down) / 2
183: (12)             new_width = abs(scene_critical_x_left - scene_critical_x_right)
184: (8)             new_height = abs(scene_critical_y_up - scene_critical_y_down)
185: (8)             m_target = self.frame.animate if animate else self.frame
186: (8)             if new_width / self.frame.width > new_height / self.frame.height:
187: (12)                 return m_target.set_x(x).set_y(y).set(width=new_width + margin)
188: (8)             else:
189: (12)                 return m_target.set_x(x).set_y(y).set(height=new_height + margin)

```

#### File 31 - mapping\_camera.py:

```

1: (0)             """A camera that allows mapping between objects."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["MappingCamera", "OldMultiCamera", "SplitScreenCamera"]

```

```

4: (0)          import math
5: (0)          import numpy as np
6: (0)          from ..camera.camera import Camera
7: (0)          from ..mobobject.types.vectorized_mobobject import VMobobject
8: (0)          from ..utils.config_ops import DictAsObject
9: (0)          class MappingCamera(Camera):
10: (4)          """Camera object that allows mapping
11: (4)          between objects.
12: (4)          """
13: (4)          def __init__(
14: (8)              self,
15: (8)              mapping_func=lambda p: p,
16: (8)              min_num_curves=50,
17: (8)              allow_object_intrusion=False,
18: (8)              **kwargs,
19: (4)          ):
20: (8)              self.mapping_func = mapping_func
21: (8)              self.min_num_curves = min_num_curves
22: (8)              self.allow_object_intrusion = allow_object_intrusion
23: (8)              super().__init__(**kwargs)
24: (4)          def points_to_pixel_coords(self, mobobject, points):
25: (8)              return super().points_to_pixel_coords(
26: (12)                  mobobject,
27: (12)                  np.apply_along_axis(self.mapping_func, 1, points),
28: (8)              )
29: (4)          def capture_mobjects(self, mobjects, **kwargs):
30: (8)              mobjects = self.get_mobjects_to_display(mobjects, **kwargs)
31: (8)              if self.allow_object_intrusion:
32: (12)                  mobobject_copies = mobjects
33: (8)              else:
34: (12)                  mobobject_copies = [mobobject.copy() for mobobject in mobjects]
35: (8)              for mobobject in mobobject_copies:
36: (12)                  if (
37: (16)                      isinstance(mobobject, VMobobject)
38: (16)                      and 0 < mobobject.get_num_curves() < self.min_num_curves
39: (12)                  ):
40: (16)                      mobobject.insert_n_curves(self.min_num_curves)
41: (8)              super().capture_mobjects(
42: (12)                  mobobject_copies,
43: (12)                  include_submobjects=False,
44: (12)                  excluded_mobjects=None,
45: (8)              )
46: (0)          class OldMultiCamera(Camera):
47: (4)          def __init__(self, *cameras_with_start_positions, **kwargs):
48: (8)              self.shifted_cameras = [
49: (12)                  DictAsObject(
50: (16)                      {
51: (20)                          "camera": camera_with_start_positions[0],
52: (20)                          "start_x": camera_with_start_positions[1][1],
53: (20)                          "start_y": camera_with_start_positions[1][0],
54: (20)                          "end_x": camera_with_start_positions[1][1]
55: (20)                          + camera_with_start_positions[0].pixel_width,
56: (20)                          "end_y": camera_with_start_positions[1][0]
57: (20)                          + camera_with_start_positions[0].pixel_height,
58: (16)                      },
59: (12)                  )
60: (12)                  for camera_with_start_positions in cameras_with_start_positions
61: (8)              ]
62: (8)              super().__init__(**kwargs)
63: (4)          def capture_mobjects(self, mobjects, **kwargs):
64: (8)              for shifted_camera in self.shifted_cameras:
65: (12)                  shifted_camera.camera.capture_mobjects(mobjects, **kwargs)
66: (12)                  self.pixel_array[
67: (16)                      shifted_camera.start_y : shifted_camera.end_y,
68: (16)                      shifted_camera.start_x : shifted_camera.end_x,
69: (12)                  ] = shifted_camera.camera.pixel_array
70: (4)          def set_background(self, pixel_array, **kwargs):
71: (8)              for shifted_camera in self.shifted_cameras:
72: (12)                  shifted_camera.camera.set_background(

```

```

73: (16)           pixel_array[
74: (20)             shifted_camera.start_y : shifted_camera.end_y,
75: (20)             shifted_camera.start_x : shifted_camera.end_x,
76: (16)           ],
77: (16)           **kwargs,
78: (12)       )
79: (4)     def set_pixel_array(self, pixel_array, **kwargs):
80: (8)       super().set_pixel_array(pixel_array, **kwargs)
81: (8)       for shifted_camera in self.shifted_cameras:
82: (12)         shifted_camera.camera.set_pixel_array(
83: (16)           pixel_array[
84: (20)             shifted_camera.start_y : shifted_camera.end_y,
85: (20)             shifted_camera.start_x : shifted_camera.end_x,
86: (16)           ],
87: (16)           **kwargs,
88: (12)       )
89: (4)     def init_background(self):
90: (8)       super().init_background()
91: (8)       for shifted_camera in self.shifted_cameras:
92: (12)         shifted_camera.camera.init_background()
93: (0)   class SplitScreenCamera(OldMultiCamera):
94: (4)     def __init__(self, left_camera, right_camera, **kwargs):
95: (8)       Camera.__init__(self, **kwargs) # to set attributes such as
pixel_width
96: (8)       self.left_camera = left_camera
97: (8)       self.right_camera = right_camera
98: (8)       half_width = math.ceil(self.pixel_width / 2)
99: (8)       for camera in [self.left_camera, self.right_camera]:
100: (12)         camera.reset_pixel_shape(camera.pixel_height, half_width)
101: (8)       super().__init__(
102: (12)           (left_camera, (0, 0)),
103: (12)           (right_camera, (0, half_width)),
104: (8)       )
-----
```

### File 32 - three\_d\_camera.py:

```

1: (0) """A camera that can be positioned and oriented in three-dimensional space."""
2: (0) from __future__ import annotations
3: (0) __all__ = ["ThreeDCamera"]
4: (0) from typing import Callable
5: (0) import numpy as np
6: (0) from manim.mobject.mobject import Mobject
7: (0) from manim.mobject.three_d.three_d_utils import (
8: (4)     get_3d_vmob_end_corner,
9: (4)     get_3d_vmob_end_corner_unit_normal,
10: (4)    get_3d_vmob_start_corner,
11: (4)    get_3d_vmob_start_corner_unit_normal,
12: (0) )
13: (0) from manim.mobject.value_tracker import ValueTracker
14: (0) from .. import config
15: (0) from ..camera.camera import Camera
16: (0) from ..constants import *
17: (0) from ..mobject.types.point_cloud_mobject import Point
18: (0) from ..utils.color import get_shaded_rgb
19: (0) from ..utils.family import extract_mobject_family_members
20: (0) from ..utils.space_ops import rotation_about_z, rotation_matrix
21: (0) class ThreeDCamera(Camera):
22: (4)     def __init__(
23: (8)         self,
24: (8)         focal_distance=20.0,
25: (8)         shading_factor=0.2,
26: (8)         default_distance=5.0,
27: (8)         light_source_start_point=9 * DOWN + 7 * LEFT + 10 * OUT,
28: (8)         should_apply_shading=True,
29: (8)         exponential_projection=False,
30: (8)         phi=0,
31: (8)         theta=-90 * DEGREES,
```

```

32: (8)                     gamma=0,
33: (8)                     zoom=1,
34: (8)                     **kwargs,
35: (4)                   ):
36: (8)                     """Initializes the ThreeDCamera
37: (8)                     Parameters
38: (8)                     -----
39: (8)                     *kwargs
40: (12)                     Any keyword argument of Camera.
41: (8)
42: (8)
stroke_width=0)
43: (8)                     super().__init__(**kwargs)
44: (8)                     self.focal_distance = focal_distance
45: (8)                     self.phi = phi
46: (8)                     self.theta = theta
47: (8)                     self.gamma = gamma
48: (8)                     self.zoom = zoom
49: (8)                     self.shading_factor = shading_factor
50: (8)                     self.default_distance = default_distance
51: (8)                     self.light_source_start_point = light_source_start_point
52: (8)                     self.light_source = Point(self.light_source_start_point)
53: (8)                     self.should_apply_shading = should_apply_shading
54: (8)                     self.exponential_projection = exponential_projection
55: (8)                     self.max_allowable_norm = 3 * config["frame_width"]
56: (8)                     self.phi_tracker = ValueTracker(self.phi)
57: (8)                     self.theta_tracker = ValueTracker(self.theta)
58: (8)                     self.focal_distance_tracker = ValueTracker(self.focal_distance)
59: (8)                     self.gamma_tracker = ValueTracker(self.gamma)
60: (8)                     self.zoom_tracker = ValueTracker(self.zoom)
61: (8)                     self.fixed_orientation_mobjects = {}
62: (8)                     self.fixed_in_frame_mobjects = set()
63: (8)                     self.reset_rotation_matrix()
64: (4)
65: (4)
def frame_center(self):
66: (8)                     return self._frame_center.points[0]
@frame_center.setter
67: (4)
68: (4)
def frame_center(self, point):
69: (8)                     self._frame_center.move_to(point)
70: (4)
71: (8)
72: (8)
73: (4)
74: (8)
focal_distance,
75: (8)
76: (8)
77: (8)
78: (8)
79: (12)
80: (8)
81: (8)
82: (12)
83: (12)
84: (12)
85: (12)
86: (12)
87: (8)
88: (4)
89: (8)
90: (12)
91: (8)
92: (12)
93: (12)
94: (16)
95: (12)
96: (16)
97: (12)
98: (16)
                    """
                    Returns
                    -----
                    list
                    list of ValueTracker objects
                    """
return [
    self.phi_tracker,
    self.theta_tracker,
    self.focal_distance_tracker,
    self.gamma_tracker,
    self.zoom_tracker,
]
def modified_rgbs(self, vmobject, rgbs):
    if not self.should_apply_shading:
        return rgbs
    if vmobject.shade_in_3d and (vmobject.get_num_points() > 0):
        light_source_point = self.light_source.points[0]
        if len(rgbs) < 2:
            shaded_rgbs = rgbs.repeat(2, axis=0)
        else:
            shaded_rgbs = np.array(rgbs[:2])
        shaded_rgbs[0, :3] = get_shaded_rgb(
            shaded_rgbs[0, :3],

```

```

99: (16)                               get_3d_vmob_start_corner(vmobject),
100: (16)                               get_3d_vmob_start_corner_unit_normal(vmobject),
101: (16)                               light_source_point,
102: (12)
103: (12)
104: (16)                               shaded_rgbs[1, :3] = get_shaded_rgb(
105: (16)                               shaded_rgbs[1, :3],
106: (16)                               get_3d_vmob_end_corner(vmobject),
107: (16)                               get_3d_vmob_end_corner_unit_normal(vmobject),
108: (12)                               light_source_point,
109: (12)
110: (8)                               return rgbs
111: (4) def get_stroke_rgbs(
112: (8)     self,
113: (8)     vmobject,
114: (8)     background=False,
115: (4)         ): # NOTE : DocStrings From parent
116: (8)         return self.modified_rgbs(vmobject,
117: (4)         )
118: (8)         vmobject.get_stroke_rgbs(background))
119: (4) def get_fill_rgbs(self, vmobject): # NOTE : DocStrings From parent
120: (8)     return self.modified_rgbs(vmobject, vmobject.get_fill_rgbs())
121: (8) def get_mobjects_to_display(self, *args, **kwargs): # NOTE : DocStrings
122: (8)     From parent
123: (12)     mobjects = super().get_mobjects_to_display(*args, **kwargs)
124: (16)     rot_matrix = self.get_rotation_matrix()
125: (12)     def z_key(mob):
126: (8)         if not (hasattr(mob, "shade_in_3d") and mob.shade_in_3d):
127: (12)             return np.inf
128: (16)             return np.dot(mob.get_z_index_reference_point(), rot_matrix.T)[2]
129: (8)         return sorted(mobjects, key=z_key)
130: (4) def get_phi(self):
131: (8)     """Returns the Polar angle (the angle off Z_AXIS) phi.
132: (8)     Returns
133: (8)     -----
134: (8)     float
135: (8)     The Polar angle in radians.
136: (4)     """
137: (8)     def get_theta(self):
138: (8)         """Returns the Azimuthal i.e the angle that spins the camera around
139: (8)         the Z_AXIS.
140: (12)         Returns
141: (8)         -----
142: (8)         float
143: (8)         The Azimuthal angle in radians.
144: (4)         """
145: (8)         def get_focal_distance(self):
146: (8)             """Returns focal_distance of the Camera.
147: (8)             Returns
148: (8)             -----
149: (8)             float
150: (8)             The focal_distance of the Camera in MUnits.
151: (4)             """
152: (8)             def get_gamma(self):
153: (8)                 """Returns the rotation of the camera about the vector from the ORIGIN
154: (8)                 to the Camera.
155: (8)                 Returns
156: (8)                 -----
157: (8)                 float
158: (8)                 The angle of rotation of the camera about the vector
159: (8)                 from the ORIGIN to the Camera in radians
160: (4)                 """
161: (8)                 def get_zoom(self):
162: (8)                     """Returns the zoom amount of the camera.
163: (8)                     Returns

```

```

164: (8)           float
165: (12)          The zoom amount of the camera.
166: (8)
167: (8)           """
168: (4)            return self.zoom_tracker.get_value()
169: (8)            def set_phi(self, value: float):
170: (8)              """Sets the polar angle i.e the angle between Z_AXIS and Camera
through ORIGIN in radians.
171: (8)            Parameters
172: (8)              -----
173: (12)             value
174: (8)                The new value of the polar angle in radians.
175: (8)                """
176: (4)            def set_theta(self, value: float):
177: (8)              """Sets the azimuthal angle i.e the angle that spins the camera around
Z_AXIS in radians.
178: (8)            Parameters
179: (8)              -----
180: (8)             value
181: (12)            The new value of the azimuthal angle in radians.
182: (8)             """
183: (8)            self.theta_tracker.set_value(value)
184: (4)            def set_focal_distance(self, value: float):
185: (8)              """Sets the focal_distance of the Camera.
186: (8)            Parameters
187: (8)              -----
188: (8)             value
189: (12)            The focal_distance of the Camera.
190: (8)             """
191: (8)            self.focal_distance_tracker.set_value(value)
192: (4)            def set_gamma(self, value: float):
193: (8)              """Sets the angle of rotation of the camera about the vector from the
ORIGIN to the Camera.
194: (8)            Parameters
195: (8)              -----
196: (8)             value
197: (12)            The new angle of rotation of the camera.
198: (8)             """
199: (8)            self.gamma_tracker.set_value(value)
200: (4)            def set_zoom(self, value: float):
201: (8)              """Sets the zoom amount of the camera.
202: (8)            Parameters
203: (8)              -----
204: (8)             value
205: (12)            The zoom amount of the camera.
206: (8)             """
207: (8)            self.zoom_tracker.set_value(value)
208: (4)            def reset_rotation_matrix(self):
209: (8)              """Sets the value of self.rotation_matrix to
210: (8)                the matrix corresponding to the current position of the camera
211: (8)                """
212: (8)            self.rotation_matrix = self.generate_rotation_matrix()
213: (4)            def get_rotation_matrix(self):
214: (8)              """Returns the matrix corresponding to the current position of the
camera.
215: (8)            Returns
216: (8)              -----
217: (8)            np.array
218: (12)            The matrix corresponding to the current position of the camera.
219: (8)            """
220: (8)            return self.rotation_matrix
221: (4)            def generate_rotation_matrix(self):
222: (8)              """Generates a rotation matrix based off the current position of the
camera.
223: (8)            Returns
224: (8)              -----
225: (8)            np.array
226: (12)            The matrix corresponding to the current position of the camera.
227: (8)            """

```

```

228: (8)             phi = self.get_phi()
229: (8)             theta = self.get_theta()
230: (8)             gamma = self.get_gamma()
231: (8)             matrices = [
232: (12)             rotation_about_z(-theta - 90 * DEGREES),
233: (12)             rotation_matrix(-phi, RIGHT),
234: (12)             rotation_about_z(gamma),
235: (8)         ]
236: (8)         result = np.identity(3)
237: (8)         for matrix in matrices:
238: (12)             result = np.dot(matrix, result)
239: (8)         return result
240: (4)     def project_points(self, points: np.ndarray | list):
241: (8)         """Applies the current rotation_matrix as a projection
242: (8)         matrix to the passed array of points.
243: (8)         Parameters
244: (8)             -----
245: (8)             points
246: (12)                 The list of points to project.
247: (8)         Returns
248: (8)             -----
249: (8)             np.array
250: (12)                 The points after projecting.
251: (8)         """
252: (8)         frame_center = self.frame_center
253: (8)         focal_distance = self.get_focal_distance()
254: (8)         zoom = self.get_zoom()
255: (8)         rot_matrix = self.get_rotation_matrix()
256: (8)         points = points - frame_center
257: (8)         points = np.dot(points, rot_matrix.T)
258: (8)         zs = points[:, 2]
259: (8)         for i in 0, 1:
260: (12)             if self.exponential_projection:
261: (16)                 factor = np.exp(zs / focal_distance)
262: (16)                 lt0 = zs < 0
263: (16)                 factor[lt0] = focal_distance / (focal_distance - zs[lt0])
264: (12)             else:
265: (16)                 factor = focal_distance / (focal_distance - zs)
266: (16)                 factor[(focal_distance - zs) < 0] = 10**6
267: (12)                 points[:, i] *= factor * zoom
268: (8)         return points
269: (4)     def project_point(self, point: list | np.ndarray):
270: (8)         """Applies the current rotation_matrix as a projection
271: (8)         matrix to the passed point.
272: (8)         Parameters
273: (8)             -----
274: (8)             point
275: (12)                 The point to project.
276: (8)         Returns
277: (8)             -----
278: (8)             np.array
279: (12)                 The point after projection.
280: (8)         """
281: (8)         return self.project_points(point.reshape((1, 3)))[0, :]
282: (4)     def transform_points_pre_display(
283: (8)             self,
284: (8)             mobject,
285: (8)             points,
286: (4)             ): # TODO: Write Docstrings for this Method.
287: (8)             points = super().transform_points_pre_display(mobject, points)
288: (8)             fixed_orientation = mobject in self.fixed_orientation_mobjects
289: (8)             fixed_in_frame = mobject in self.fixed_in_frame_mobjects
290: (8)             if fixed_in_frame:
291: (12)                 return points
292: (8)             if fixed_orientation:
293: (12)                 center_func = self.fixed_orientation_mobjects[mobject]
294: (12)                 center = center_func()
295: (12)                 new_center = self.project_point(center)
296: (12)                 return points + (new_center - center)

```

```

297: (8)           else:
298: (12)         return self.project_points(points)
299: (4)          def add_fixed_orientation_mobjects(
300: (8)            self,
301: (8)            *mobjects: Mobject,
302: (8)            use_static_center_func: bool = False,
303: (8)            center_func: Callable[[], np.ndarray] | None = None,
304: (4)          ):
305: (8)            """This method allows the mobject to have a fixed orientation,
306: (8)            even when the camera moves around.
307: (8)            E.G If it was passed through this method, facing the camera, it
308: (8)            will continue to face the camera even as the camera moves.
309: (8)            Highly useful when adding labels to graphs and the like.
310: (8)          Parameters
311: (8)          -----
312: (8)          *mobjects
313: (12)        The mobject whose orientation must be fixed.
314: (8)          use_static_center_func
315: (12)        Whether or not to use the function that takes the mobject's
316: (12)        center as centerpoint, by default False
317: (8)          center_func
318: (12)        The function which returns the centerpoint
319: (12)        with respect to which the mobject will be oriented, by default
None
320: (8)        """
321: (8)        def get_static_center_func(mobject):
322: (12)          point = mobject.get_center()
323: (12)          return lambda: point
324: (8)        for mobject in mobjects:
325: (12)          if center_func:
326: (16)            func = center_func
327: (12)          elif use_static_center_func:
328: (16)            func = get_static_center_func(mobject)
329: (12)          else:
330: (16)            func = mobject.get_center
331: (12)          for submob in mobject.get_family():
332: (16)            self.fixed_orientation_mobjects[submob] = func
333: (4)          def add_fixed_in_frame_mobjects(self, *mobjects: Mobject):
334: (8)            """This method allows the mobject to have a fixed position,
335: (8)            even when the camera moves around.
336: (8)            E.G If it was passed through this method, at the top of the frame, it
337: (8)            will continue to be displayed at the top of the frame.
338: (8)            Highly useful when displaying Titles or formulae or the like.
339: (8)          Parameters
340: (8)          -----
341: (8)          **mobjects
342: (12)        The mobject to fix in frame.
343: (8)        """
344: (8)        for mobject in extract_mobject_family_members(mobjects):
345: (12)          self.fixed_in_frame_mobjects.add(mobject)
346: (4)          def remove_fixed_orientation_mobjects(self, *mobjects: Mobject):
347: (8)            """If a mobject was fixed in its orientation by passing it through
348: (8)            :meth:`.add_fixed_orientation_mobjects`, then this undoes that fixing.
349: (8)            The Mobject will no longer have a fixed orientation.
350: (8)          Parameters
351: (8)          -----
352: (8)          mobjects
353: (12)        The mobjects whose orientation need not be fixed any longer.
354: (8)        """
355: (8)        for mobject in extract_mobject_family_members(mobjects):
356: (12)          if mobject in self.fixed_orientation_mobjects:
357: (16)            del self.fixed_orientation_mobjects[mobject]
358: (4)          def remove_fixed_in_frame_mobjects(self, *mobjects: Mobject):
359: (8)            """If a mobject was fixed in frame by passing it through
360: (8)            :meth:`.add_fixed_in_frame_mobjects`, then this undoes that fixing.
361: (8)            The Mobject will no longer be fixed in frame.
362: (8)          Parameters
363: (8)          -----
364: (8)          mobjects

```

```

365: (12)           The mobjects which need not be fixed in frame any longer.
366: (8)
367: (8)
368: (12)
369: (16)           """
370:             for mobject in extract_mobject_family_members(mobjects):
371:                 if mobject in self.fixed_in_frame_mobjects:
372:                     self.fixed_in_frame_mobjects.remove(mobject)
373:
374-----
```

## File 33 - mobject\_update\_utils.py:

```

1: (0)           """Utility functions for continuous animation of mobjects."""
2: (0)           from __future__ import annotations
3: (0)           __all__ = [
4: (4)             "assert_is_mobject_method",
5: (4)             "always",
6: (4)             "f_always",
7: (4)             "always_redraw",
8: (4)             "always_shift",
9: (4)             "always_rotate",
10: (4)            "turn_animation_into_updater",
11: (4)            "cycle_animation",
12: (0)
13: (0)           import inspect
14: (0)           from typing import TYPE_CHECKING, Callable
15: (0)           import numpy as np
16: (0)           from manim.constants import DEGREES, RIGHT
17: (0)           from manim.mobject.mobject import Mobject
18: (0)           from manim.opengl import OpenGLMobject
19: (0)           from manim.utils.space_ops import normalize
20: (0)           if TYPE_CHECKING:
21: (4)             from manim.animation.animation import Animation
22: (0)           def assert_is_mobject_method(method: Callable) -> None:
23: (4)             assert inspect.ismethod(method)
24: (4)             mobject = method.__self__
25: (4)             assert isinstance(mobject, (Mobject, OpenGLMobject))
26: (0)           def always(method: Callable, *args, **kwargs) -> Mobject:
27: (4)             assert_is_mobject_method(method)
28: (4)             mobject = method.__self__
29: (4)             func = method.__func__
30: (4)             mobject.add_updater(lambda m: func(m, *args, **kwargs))
31: (4)             return mobject
32: (0)           def f_always(method: Callable[[Mobject], None], *arg_generators, **kwargs) ->
Mobject:
33: (4)             """
34: (4)               More functional version of always, where instead
35: (4)               of taking in args, it takes in functions which output
36: (4)               the relevant arguments.
37: (4)             """
38: (4)             assert_is_mobject_method(method)
39: (4)             mobject = method.__self__
40: (4)             func = method.__func__
41: (4)             def updater(mob):
42: (8)               args = [arg_generator() for arg_generator in arg_generators]
43: (8)               func(mob, *args, **kwargs)
44: (4)               mobject.add_updater(updater)
45: (4)               return mobject
46: (0)           def always_redraw(func: Callable[[], Mobject]) -> Mobject:
47: (4)             """Redraw the mobject constructed by a function every frame.
48: (4)             This function returns a mobject with an attached updater that
49: (4)             continuously regenerates the mobject according to the
50: (4)             specified function.
51: (4)             Parameters
52: (4)             -----
53: (4)             func
54: (8)               A function without (required) input arguments that returns
55: (8)               a mobject.
56: (4)             Examples
57: (4)             -----
58: (4)               .. manim:: TangentAnimation
```

```

59: (8)                     class TangentAnimation(Scene):
60: (12)                   def construct(self):
61: (16)                     ax = Axes()
62: (16)                     sine = ax.plot(np.sin, color=RED)
63: (16)                     alpha = ValueTracker(0)
64: (16)                     point = always_redraw(
65: (20)                       lambda: Dot(
66: (24)                         sine.point_from_proportion(alpha.get_value()),
67: (24)                         color=BLUE
68: (20)                     )
69: (16)                 )
70: (16)                 tangent = always_redraw(
71: (20)                   lambda: TangentLine(
72: (24)                     sine,
73: (24)                     alpha=alpha.get_value(),
74: (24)                     color=YELLOW,
75: (24)                     length=4
76: (20)                 )
77: (16)             )
78: (16)             self.add(ax, sine, point, tangent)
79: (16)             self.play(alpha.animate.set_value(1), rate_func=linear,
run_time=2)
80: (4)             """
81: (4)             mob = func()
82: (4)             mob.add_updater(lambda _: mob.become(func()))
83: (4)             return mob
84: (0)             def always_shift(
85: (4)               mobject: Mobject, direction: np.ndarray[np.float64] = RIGHT, rate: float =
0.1
86: (0)             ) -> Mobject:
87: (4)               """A mobject which is continuously shifted along some direction
88: (4)               at a certain rate.
89: (4)               Parameters
90: (4)               -----
91: (4)               mobject
92: (8)                 The mobject to shift.
93: (4)               direction
94: (8)                 The direction to shift. The vector is normalized, the specified
magnitude
95: (8)                 is not relevant.
96: (4)               rate
97: (8)                 Length in Manim units which the mobject travels in one
98: (8)                 second along the specified direction.
99: (4)               Examples
100: (4)               -----
101: (4)               .. manim:: ShiftingSquare
102: (8)                 class ShiftingSquare(Scene):
103: (12)                   def construct(self):
104: (16)                     sq = Square().set_fill(opacity=1)
105: (16)                     tri = Triangle()
106: (16)                     VGroup(sq, tri).arrange(LEFT)
107: (16)                     always_shift(sq, RIGHT, rate=5)
108: (16)                     self.add(sq)
109: (16)                     self.play(tri.animate.set_fill(opacity=1))
110: (4)             """
111: (4)             mobject.add_updater(lambda m, dt: m.shift(dt * rate *
normalize(direction)))
112: (4)             return mobject
113: (0)             def always_rotate(mobject: Mobject, rate: float = 20 * DEGREES, **kwargs) ->
Mobject:
114: (4)               """A mobject which is continuously rotated at a certain rate.
115: (4)               Parameters
116: (4)               -----
117: (4)               mobject
118: (8)                 The mobject to be rotated.
119: (4)               rate
120: (8)                 The angle which the mobject is rotated by
121: (8)                 over one second.
122: (4)               kwargs

```

```

123: (8)             Further arguments to be passed to :meth:`.Mobject.rotate`.
124: (4)             Examples
125: (4)
126: (4)             .. manim:: SpinningTriangle
127: (8)             class SpinningTriangle(Scene):
128: (12)             def construct(self):
129: (16)                 tri = Triangle().set_fill(opacity=1).set_z_index(2)
130: (16)                 sq = Square().to_edge(LEFT)
131: (16)                 always_rotate(tri, rate=2*PI, about_point=ORIGIN)
132: (16)                 self.add(tri, sq)
133: (16)                 self.play(sq.animate.to_edge(RIGHT), rate_func=linear,
run_time=1)
134: (4)
135: (4)             """
136: (4)             mobject.add_updater(lambda m, dt: m.rotate(dt * rate, **kwargs))
137: (0)             return mobject
138: (4)             def turn_animation_into_updater(
139: (0)                 animation: Animation, cycle: bool = False, **kwargs
140: (0)             ) -> Mobject:
141: (4)             """
142: (4)             Add an updater to the animation's mobject which applies
143: (4)             the interpolation and update functions of the animation
144: (4)             If cycle is True, this repeats over and over. Otherwise,
145: (4)             the updater will be popped upon completion
146: (4)             Examples
147: (4)
148: (8)             .. manim:: WelcomeToManim
149: (12)             class WelcomeToManim(Scene):
150: (16)                 def construct(self):
151: (16)                     words = Text("Welcome to")
152: (16)                     banner = ManimBanner().scale(0.5)
153: (16)                     VGroup(words, banner).arrange(DOWN)
154: (16)                     turn_animation_into_updater(Write(words, run_time=0.9))
155: (16)                     self.add(words)
156: (16)                     self.wait(0.5)
157: (4)                     self.play(banner.expand(), run_time=0.5)
158: (4)
159: (4)             mobject = animation.mobject
160: (4)             animation.suspend_mobject_updating = False
161: (4)             animation.begin()
162: (4)             animation.total_time = 0
163: (8)             def update(m: Mobject, dt: float):
164: (8)                 run_time = animation.get_run_time()
165: (8)                 time_ratio = animation.total_time / run_time
166: (12)                 if cycle:
167: (8)                     alpha = time_ratio % 1
168: (12)                 else:
169: (12)                     alpha = np.clip(time_ratio, 0, 1)
170: (16)                     if alpha >= 1:
171: (16)                         animation.finish()
172: (16)                         m.remove_updater(update)
173: (8)                         return
174: (8)                         animation.interpolate(alpha)
175: (8)                         animation.update_mobjects(dt)
176: (4)                         animation.total_time += dt
177: (4)                         mobject.add_updater(update)
178: (0)             return turn_animation_into_updater(animation, cycle=True, **kwargs)
179: (4)

-----

```

## File 34 - transform\_matching\_parts.py:

```

1: (0)             """Animations that try to transform Mobjects while keeping track of identical
parts."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["TransformMatchingShapes", "TransformMatchingTex"]
4: (0)             from typing import TYPE_CHECKING
5: (0)             import numpy as np

```

```

6: (0)           from manim.mobject.opengl.opengl_mobject import OpenGLGroup, OpenGLMobject
7: (0)           from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLGroup,
OpenGLMobject
8: (0)           from ..config import config
9: (0)           from ..constants import RendererType
10: (0)          from ..mobject.mobject import Group, Mobject
11: (0)          from ..mobject.types.vectorized_mobject import VGroup, VMobject
12: (0)          from .composition import AnimationGroup
13: (0)          from .fading import FadeIn, FadeOut
14: (0)          from .transform import FadeTransformPieces, Transform
15: (0)          if TYPE_CHECKING:
16: (4)              from ..scene.scene import Scene
17: (0)          class TransformMatchingAbstractBase(AnimationGroup):
18: (4)              """Abstract base class for transformations that keep track of matching
parts.
19: (4)              Subclasses have to implement the two static methods
20: (4)              :meth:`~.TransformMatchingAbstractBase.get_mobject_parts` and
21: (4)              :meth:`~.TransformMatchingAbstractBase.get_mobject_key`.
22: (4)              Basically, this transformation first maps all submobjects returned
23: (4)              by the ``get_mobject_parts`` method to certain keys by applying the
24: (4)              ``get_mobject_key`` method. Then, submobjects with matching keys
25: (4)              are transformed into each other.
26: (4)              Parameters
27: (4)              -----
28: (4)              mobject
29: (8)                  The starting :class:`~.Mobject`.
30: (4)              target_mobject
31: (8)                  The target :class:`~.Mobject`.
32: (4)              transform_mismatches
33: (8)                  Controls whether submobjects without a matching key are transformed
34: (8)                  into each other by using :class:`~.Transform`. Default: ``False``.
35: (4)              fade_transform_mismatches
36: (8)                  Controls whether submobjects without a matching key are transformed
37: (8)                  into each other by using :class:`~.FadeTransform`. Default: ``False``.
38: (4)              key_map
39: (8)                  Optional. A dictionary mapping keys belonging to some of the starting
submobjects (i.e., the return values of the ``get_mobject_key``
40: (8)                  to some keys belonging to the target mobject's submobjects that should
method)
41: (8)                  be transformed although the keys don't match.
42: (8)
43: (4)              kwargs
44: (8)                  All further keyword arguments are passed to the submobject
transformations.
45: (4)              Note
46: (4)              -----
47: (4)              If neither ``transform_mismatches`` nor ``fade_transform_mismatches``
48: (4)              are set to ``True``, submobjects without matching keys in the starting
49: (4)              mobject are faded out in the direction of the unmatched submobjects in
50: (4)              the target mobject, and unmatched submobjects in the target mobject
51: (4)              are faded in from the direction of the unmatched submobjects in the
52: (4)              start mobject.
53: (4)              """
54: (4)              def __init__(
55: (8)                  self,
56: (8)                  mobject,
57: (8)                  target_mobject,
58: (8)                  transform_mismatches: bool = False,
59: (8)                  fade_transform_mismatches: bool = False,
60: (8)                  key_map: dict | None = None,
61: (8)                  **kwargs,
62: (4)              ):
63: (8)                  if isinstance(mobject, OpenGLMobject):
64: (12)                      group_type = OpenGLGroup
65: (8)                  elif isinstance(mobject, OpenGLMobject):
66: (12)                      group_type = OpenGLGroup
67: (8)                  elif isinstance(mobject, VMobject):
68: (12)                      group_type = VGroup
69: (8)                  else:

```

```

70: (12)                     group_type = Group
71: (8)                      source_map = self.get_shape_map(mobject)
72: (8)                      target_map = self.get_shape_map(target_mobject)
73: (8)                      if key_map is None:
74: (12)                        key_map = {}
75: (8)                      transform_source = group_type()
76: (8)                      transform_target = group_type()
77: (8)                      kwargs["final_alpha_value"] = 0
78: (8)                      for key in set(source_map).intersection(target_map):
79: (12)                        transform_source.add(source_map[key])
80: (12)                        transform_target.add(target_map[key])
81: (8)                      anims = [Transform(transform_source, transform_target, **kwargs)]
82: (8)                      key_mapped_source = group_type()
83: (8)                      key_mapped_target = group_type()
84: (8)                      for key1, key2 in key_map.items():
85: (12)                        if key1 in source_map and key2 in target_map:
86: (16)                          key_mapped_source.add(source_map[key1])
87: (16)                          key_mapped_target.add(target_map[key2])
88: (16)                          source_map.pop(key1, None)
89: (16)                          target_map.pop(key2, None)
90: (8)                      if len(key_mapped_source) > 0:
91: (12)                        anims.append(
92: (16)                            FadeTransformPieces(key_mapped_source, key_mapped_target,
93: (12)                                )
94: (8)                          fade_source = group_type()
95: (8)                          fade_target = group_type()
96: (8)                          for key in set(source_map).difference(target_map):
97: (12)                            fade_source.add(source_map[key])
98: (8)                          for key in set(target_map).difference(source_map):
99: (12)                            fade_target.add(target_map[key])
100: (8)                         fade_target_copy = fade_target.copy()
101: (8)                         if transform_mismatches:
102: (12)                           if "replace_mobject_with_target_in_scene" not in kwargs:
103: (16)                             kwargs["replace_mobject_with_target_in_scene"] = True
104: (12)                             anims.append(Transform(fade_source, fade_target, **kwargs))
105: (8)                         elif fade_transform_mismatches:
106: (12)                           anims.append(FadeTransformPieces(fade_source, fade_target,
107: (8)                               )
108: (12)                               anims.append(FadeOut(fade_source, target_position=fade_target,
109: (12)                                   )
110: (16)                                   anims.append(
111: (12)                                       FadeIn(fade_target_copy, target_position=fade_target,
112: (8)                                           )
113: (8)                                         super().__init__(*anim)
114: (8)                                         self.to_remove = [mobject, fade_target_copy]
115: (8)                                         self.to_add = target_mobject
116: (4)                                         def get_shape_map(self, mobject: Mobject) -> dict:
117: (8)                                           shape_map = {}
118: (12)                                           for sm in self.get_mobject_parts(mobject):
119: (12)                                             key = self.get_mobject_key(sm)
120: (16)                                             if key not in shape_map:
121: (20)                                               if config["renderer"] == RendererType.OPENGL:
122: (16)                                                 shape_map[key] = OpenGLGroup()
123: (20)                                               else:
124: (12)                                                 shape_map[key] = VGroup()
125: (8)                                                 shape_map[key].add(sm)
126: (4)                                         return shape_map
127: (8)                                         def clean_up_from_scene(self, scene: Scene) -> None:
128: (12)                                           for anim in self.animations:
129: (8)                                             anim.interpolate(0)
130: (8)                                             scene.remove(self.mobject)
131: (8)                                             scene.remove(*self.to_remove)
132: (4)                                             scene.add(self.to_add)
133: (4) @staticmethod
134: (8)                                         def get_mobject_parts(mobject: Mobject):
135: (8)                                           raise NotImplementedError("To be implemented in subclass.")

```

```

135: (4) @staticmethod
136: (4)     def get_mobject_key(mobject: Mobject):
137: (8)         raise NotImplementedError("To be implemented in subclass.")
138: (0) class TransformMatchingShapes(TransformMatchingAbstractBase):
139: (4)     """An animation trying to transform groups by matching the shape
140: (4)     of their submobjects.
141: (4)     Two submobjects match if the hash of their point coordinates after
142: (4)     normalization (i.e., after translation to the origin, fixing the
143: (4)     submobject height at 1 unit, and rounding the coordinates to three decimal places)
144: (4)     matches.
145: (4)     See also
146: (4)     -----
147: (4)     :class:`~.TransformMatchingAbstractBase`
148: (4)     Examples
149: (4)     -----
150: (4)     .. manim:: Anagram
151: (8)         class Anagram(Scene):
152: (12)             def construct(self):
153: (16)                 src = Text("the morse code")
154: (16)                 tar = Text("here come dots")
155: (16)                 self.play(Write(src))
156: (16)                 self.wait(0.5)
157: (16)                 self.play(TransformMatchingShapes(src, tar, path_arc=PI/2))
158: (16)                 self.wait(0.5)
159: (4)             """
160: (4)             def __init__(
161: (8)                 self,
162: (8)                 mobject: Mobject,
163: (8)                 target_mobject: Mobject,
164: (8)                 transform_mismatches: bool = False,
165: (8)                 fade_transform_mismatches: bool = False,
166: (8)                 key_map: dict | None = None,
167: (8)                 **kwargs,
168: (4)             ):
169: (8)                 super().__init__(
170: (12)                     mobject,
171: (12)                     target_mobject,
172: (12)                     transform_mismatches=transform_mismatches,
173: (12)                     fade_transform_mismatches=fade_transform_mismatches,
174: (12)                     key_map=key_map,
175: (12)                     **kwargs,
176: (8)             )
177: (4)             @staticmethod
178: (4)             def get_mobject_parts(mobject: Mobject) -> list[Mobject]:
179: (8)                 return mobject.family_members_with_points()
180: (4)             @staticmethod
181: (4)             def get_mobject_key(mobject: Mobject) -> int:
182: (8)                 mobject.save_state()
183: (8)                 mobject.center()
184: (8)                 mobject.set(height=1)
185: (8)                 result = hash(np.round(mobject.points, 3).tobytes())
186: (8)                 mobject.restore()
187: (8)                 return result
188: (0) class TransformMatchingTex(TransformMatchingAbstractBase):
189: (4)     """A transformation trying to transform rendered LaTeX strings.
190: (4)     Two submobjects match if their ``tex_string`` matches.
191: (4)     See also
192: (4)     -----
193: (4)     :class:`~.TransformMatchingAbstractBase`
194: (4)     Examples
195: (4)     -----
196: (4)     .. manim:: MatchingEquationParts
197: (8)         class MatchingEquationParts(Scene):
198: (12)             def construct(self):
199: (16)                 variables = VGroup(MathTex("a"), MathTex("b"),
MathTex("c")).arrange_submobjects().shift(UP)
200: (16)                 eq1 = MathTex("{x}^2", "+", "{y}^2", "=", "{z}^2")
201: (16)                 eq2 = MathTex("{a}^2", "+", "{b}^2", "=", "{c}^2")

```

```

202: (16)           eq3 = MathTex("{{a}}^2", "=", "{{c}}^2", "-", "{{b}}^2")
203: (16)           self.add(eq1)
204: (16)           self.wait(0.5)
205: (16)           self.play(TransformMatchingTex(Group(eq1, variables), eq2))
206: (16)           self.wait(0.5)
207: (16)           self.play(TransformMatchingTex(eq2, eq3))
208: (16)           self.wait(0.5)
209: (4)           """
210: (4)           def __init__(
211: (8)               self,
212: (8)               mobject: Mobject,
213: (8)               target_mobject: Mobject,
214: (8)               transform_mismatches: bool = False,
215: (8)               fade_transform_mismatches: bool = False,
216: (8)               key_map: dict | None = None,
217: (8)               **kwargs,
218: (4):
219: (8)           super().__init__(
220: (12)               mobject,
221: (12)               target_mobject,
222: (12)               transform_mismatches=transform_mismatches,
223: (12)               fade_transform_mismatches=fade_transform_mismatches,
224: (12)               key_map=key_map,
225: (12)               **kwargs,
226: (8)
227: (4)           )
228: (4)           @staticmethod
229: (8)           def get_mobject_parts(mobject: Mobject) -> list[Mobject]:
230: (12)               if isinstance(mobject, (Group, VGroup, OpenGLGroup, OpenGLVGGroup)):
231: (16)                   return [
232: (16)                       p
233: (16)                       for s in mobject.submobjects
234: (12)                           for p in TransformMatchingTex.get_mobject_parts(s)
235: (8)
236: (12)               else:
237: (12)                   assert hasattr(mobject, "tex_string")
238: (4)                   return mobject.submobjects
239: (4)           @staticmethod
240: (8)           def get_mobject_key(mobject: Mobject) -> str:
241: (8)               return mobject.tex_string

```

---

## File 35 - checks.py:

```

1: (0)           """Auxiliary module for the checkhealth subcommand, contains
2: (0)           the actual check implementations."""
3: (0)           from __future__ import annotations
4: (0)           import os
5: (0)           import shutil
6: (0)           import subprocess
7: (0)           from typing import Callable
8: (0)           from ...config import config
9: (0)           __all__ = ["HEALTH_CHECKS"]
10: (0)          HEALTH_CHECKS = []
11: (0)          def healthcheck(
12: (4)              description: str,
13: (4)              recommendation: str,
14: (4)              skip_on_failed: list[Callable | str] | None = None,
15: (4)              post_fail_fix_hook: Callable | None = None,
16: (0):
17: (4)              """Decorator used for declaring health checks.
18: (4)              This decorator attaches some data to a function,
19: (4)              which is then added to a list containing all checks.
20: (4)              Parameters
21: (4)              -----
22: (4)              description
23: (8)                  A brief description of this check, displayed when
24: (8)                      the checkhealth subcommand is run.
25: (4)              recommendation

```

```

26: (8)             Help text which is displayed in case the check fails.
27: (4)             skip_on_failed
28: (8)                 A list of check functions which, if they fail, cause
29: (8)                 the current check to be skipped.
30: (4)             post_fail_fix_hook
31: (8)                 A function that is supposed to (interactively) help
32: (8)                 to fix the detected problem, if possible. This is
33: (8)                 only called upon explicit confirmation of the user.
34: (4)             Returns
35: (4)             -----
36: (4)             A check function, as required by the checkhealth subcommand.
37: (4)
38: (4)         """
39: (8)             if skip_on_failed is None:
40: (8)                 skip_on_failed = []
41: (8)             skip_on_failed = [
42: (8)                 skip.__name__ if callable(skip) else skip for skip in skip_on_failed
43: (4)             ]
44: (4)         def decorator(func):
45: (8)             func.description = description
46: (8)             func.recommendation = recommendation
47: (8)             func.skip_on_failed = skip_on_failed
48: (8)             func.post_fail_fix_hook = post_fail_fix_hook
49: (8)             HEALTH_CHECKS.append(func)
50: (4)             return func
51: (0)         return decorator
52: (4) @healthcheck(
53: (4)     description="Checking whether manim is on your PATH",
54: (4)     recommendation=(
55: (8)         "The command <manim> is currently not on your system's PATH.\n\n"
56: (8)         "You can work around this by calling the manim module directly "
57: (8)         "via <python -m manim> instead of just <manim>.\n\n"
58: (8)         "To fix the PATH issue properly: "
59: (8)         "Usually, the Python package installer pip issues a warning "
60: (8)         "during the installation which contains more information. "
61: (8)         "Consider reinstalling manim via <pip uninstall manim> "
62: (8)         "followed by <pip install manim> to see the warning again, "
63: (8)         "then consult the internet on how to modify your system's "
64: (8)         "PATH variable."
65: (4)     ),
66: (0) )
67: (4)     def is_manim_on_path():
68: (4)         path_to_manim = shutil.which("manim")
69: (4)         return path_to_manim is not None
70: (4) @healthcheck(
71: (4)     description="Checking whether the executable belongs to manim",
72: (4)     recommendation=(
73: (8)         "The command <manim> does not belong to your installed version "
74: (8)         "of this library, it likely belongs to manimgl / manimlib.\n\n"
75: (8)         "Run manim via <python -m manim> or via <manimce>, or uninstall "
76: (8)         "and reinstall manim via <pip install --upgrade "
77: (8)         "--force-reinstall manim> to fix this."
78: (4)     ),
79: (0)     skip_on_failed=[is_manim_on_path],
80: (0) )
81: (4)     def is_manim_executable_associated_to_this_library():
82: (4)         path_to_manim = shutil.which("manim")
83: (4)         with open(path_to_manim, "rb") as f:
84: (4)             manim_exec = f.read()
85: (0)         return b"manim.__main__" in manim_exec or b"%~dp0\\manim" in manim_exec
86: (4) @healthcheck(
87: (4)     description="Checking whether ffmpeg is available",
88: (4)     recommendation=(
89: (8)         "Manim does not work without ffmpeg. Please follow our "
90: (8)         "installation instructions "
91: (8)         "at https://docs.manim.community/en/stable/installation.html "
92: (8)         "to download ffmpeg. Then, either ...\\n\\n"
93: (8)         "(a) ... make the ffmpeg executable available to your system's
PATH,\n"
93: (8)         "(b) or, alternatively, use <manim cfg write --open> to create a "

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
94: (8)                     "custom configuration and set the ffmpeg_executable variable to the "
95: (8)                     "full absolute path to the ffmpeg executable."
96: (4)
97: (0)
98: (0)
99: (4)
100: (4)
101: (0)
102: (4)
103: (4)
104: (8)                 description="Checking whether ffmpeg is working",
105: (8)                 recommendation=(
106: (8)                   "Your installed version of ffmpeg does not support x264 encoding, "
107: (8)                   "which manim requires. Please follow our installation instructions "
108: (8)                   "at https://docs.manim.community/en/stable/installation.html "
109: (8)                   "to download and install a newer version of ffmpeg."
110: (0)
111: (0)
112: (4)
113: (8)                 skip_on_failed=[is_ffmpeg_available],
114: (8)
115: (4)
116: (4)
117: (8)
118: (8)
119: (4)
120: (0)
121: (4)
122: (4)
123: (8)                 @healthcheck(
124: (8)                   description="Checking whether latex is available",
125: (8)                   recommendation=(
126: (8)                     "Manim cannot find <latex> on your system's PATH. "
127: (8)                     "You will not be able to use Tex and MathTex mobjects "
128: (8)                     "in your scenes.\n\n"
129: (8)                     "Consult our installation instructions "
130: (8)                     "at https://docs.manim.community/en/stable/installation.html "
131: (0)
132: (0)
133: (4)
134: (4)
135: (0)
136: (4)
137: (4)
138: (8)                 @healthcheck(
139: (8)                   description="Checking whether dvipsvgm is available",
140: (8)                   recommendation=(
141: (8)                     "Manim could find <latex>, but not <dvipsvgm> on your system's "
142: (8)                     "PATH. Make sure your installed LaTeX distribution comes with "
143: (8)                     "dvipsvgm and consider installing a larger distribution if it "
144: (8)                     "does not."
145: (0)
146: (4)
147: (4)
-----
```

## File 36 - commands.py:

```
1: (0)             """A CLI utility helping to diagnose problems with
2: (0)             your Manim installation.
3: (0)
4: (0)             from __future__ import annotations
5: (0)             import sys
6: (0)             import click
7: (0)             import cloup
8: (0)             from .checks import HEALTH_CHECKS
9: (0)             __all__ = ["checkhealth"]
10: (0)            @cloup.command(
```

```

11: (4)             context_settings=None,
12: (0)
13: (0)         def checkhealth():
14: (4)             """This subcommand checks whether Manim is installed correctly
15: (4)             and has access to its required (and optional) system dependencies.
16: (4)             """
17: (4)             click.echo(f"Python executable: {sys.executable}\n")
18: (4)             click.echo("Checking whether your installation of Manim Community is
19: (4)             healthy...")
20: (4)
21: (8)         failed_checks = []
22: (8)         for check in HEALTH_CHECKS:
23: (12)             click.echo(f"- {check.description} ... ", nl=False)
24: (12)             if any(
25: (8)                 failed_check.__name__ in check.skip_on_failed
26: (12)                 for failed_check in failed_checks
27: (12)             ):
28: (8)                 click.secho("SKIPPED", fg="blue")
29: (8)                 continue
30: (12)             check_result = check()
31: (8)             if check_result:
32: (12)                 click.secho("PASSED", fg="green")
33: (12)             else:
34: (8)                 click.secho("FAILED", fg="red")
35: (4)                 failed_checks.append(check)
36: (8)             click.echo()
37: (12)             if failed_checks:
38: (12)                 click.echo(
39: (8)                     "There are problems with your installation, "
40: (8)                     "here are some recommendations to fix them:"
41: (12)                 )
42: (12)                 for ind, failed_check in enumerate(failed_checks):
43: (16)                     click.echo(failed_check.recommendation)
44: (4)                     if ind + 1 < len(failed_checks):
45: (16)                         click.confirm("Continue with next recommendation?")
46: (8)                 else: # no problems detected!
47: (12)                     click.echo("No problems detected, your installation seems healthy!")
48: (8)                     render_test_scene = click.confirm(
49: (8)                         "Would you like to render and preview a test scene?"
50: (12)                     )
51: (12)                     if render_test_scene:
52: (16)                         import manim as mn
53: (20)                         class CheckHealthDemo(mn.Scene):
54: (20)                             def construct(self):
55: (20)                                 banner = mn.ManimBanner().shift(mn.UP * 0.5)
56: (20)                                 self.play(banner.create())
57: (20)                                 self.wait(0.5)
58: (20)                                 self.play(banner.expand())
59: (20)                                 self.wait(0.5)
60: (20)                                 text_left = mn.Text("All systems operational!")
61: (20)                                 formula_right = mn.MathTex(r"\oint_{\gamma} f(z)dz = 0")
62: (20)                                 text_tex_group = mn.VGroup(text_left, formula_right)
63: (20)                                 text_tex_group.arrange(mn.RIGHT, buff=1).next_to(banner,
64: (20)                                     mn.DOWN)
65: (24)                                 self.play(mn.Write(text_tex_group))
66: (24)                                 self.wait(0.5)
67: (20)                                 self.play(
68: (24)                                     mn.FadeOut(banner, shift=mn.UP),
69: (24)                                     mn.FadeOut(text_tex_group, shift=mn.DOWN),
68: (12)                                 )
69: (16)             with mn.tempconfig({"preview": True, "disable_caching": True}):
69: (16)                 CheckHealthDemo().render()

```

---

File 37 - commands.py:

```

1: (0)             """Manim's init subcommand.
2: (0)             Manim's init subcommand is accessed in the command-line interface via ``manim
3: (0)             init``. Here you can specify options, subcommands, and subgroups for the init

```

```

4: (0)         group.
5: (0)         """
6: (0)             from __future__ import annotations
7: (0)             import configparser
8: (0)             from pathlib import Path
9: (0)             import click
10: (0)            import cloup
11: (0)            from ... import console
12: (0)            from ...constants import CONTEXT_SETTINGS, EPILOG, QUALITIES
13: (0)            from ...utils.file_ops import (
14: (4)                add_import_statement,
15: (4)                copy_template_files,
16: (4)                get_template_names,
17: (4)                get_template_path,
18: (0)
19: (0)        CFG_DEFAULTS = {
20: (4)            "frame_rate": 30,
21: (4)            "background_color": "BLACK",
22: (4)            "background_opacity": 1,
23: (4)            "scene_names": "Default",
24: (4)            "resolution": (854, 480),
25: (0)
26: (0)        __all__ = ["select_resolution", "update_cfg", "project", "scene"]
27: (0)    def select_resolution():
28: (4)        """Prompts input of type click.Choice from user. Presents options from
QUALITIES constant.
29: (4)        Returns
30: (4)        -----
31: (8)            :class:`tuple`
32: (12)            Tuple containing height and width.
33: (4)
34: (4)        resolution_options = []
35: (4)        for quality in QUALITIES.items():
36: (8)            resolution_options.append(
37: (12)                (quality[1]["pixel_height"], quality[1]["pixel_width"]),
38: (8)
39: (4)        resolution_options.pop()
40: (4)        choice = click.prompt(
41: (8)            "\nSelect resolution:\n",
42: (8)            type=cloup.Choice([f"{i[0]}p" for i in resolution_options]),
43: (8)            show_default=False,
44: (8)            default="480p",
45: (4)
46: (4)        return [res for res in resolution_options if f"{res[0]}p" == choice][0]
47: (0)    def update_cfg(cfg_dict: dict, project_cfg_path: Path):
48: (4)        """Updates the manim.cfg file after reading it from the project_cfg_path.
49: (4)        Parameters
50: (4)        -----
51: (4)        cfg_dict
52: (8)            values used to update manim.cfg found project_cfg_path.
53: (4)        project_cfg_path
54: (8)            Path of manim.cfg file.
55: (4)
56: (4)        config = configparser.ConfigParser()
57: (4)        config.read(project_cfg_path)
58: (4)        cli_config = config["CLI"]
59: (4)        for key, value in cfg_dict.items():
60: (8)            if key == "resolution":
61: (12)                cli_config["pixel_height"] = str(value[0])
62: (12)                cli_config["pixel_width"] = str(value[1])
63: (8)
64: (12)            else:
65: (8)                cli_config[key] = str(value)
66: (8)            with project_cfg_path.open("w") as conf:
67: (0)                config.write(conf)
68: (4)            @cloup.command(
69: (4)                context_settings=CONTEXT_SETTINGS,
70: (0)                epilog=EPILOG,
71: (0)            )
71: (0)            @cloup.argument("project_name", type=Path, required=False)

```

```

72: (0)          @cloup.option(
73: (4)            "-d",
74: (4)            "--default",
75: (4)            "default_settings",
76: (4)            is_flag=True,
77: (4)            help="Default settings for project creation.",
78: (4)            nargs=1,
79: (0)
80: (0)        )
81: (4)        def project(default_settings, **args):
82: (4)            """Creates a new project.
83: (4)            PROJECT_NAME is the name of the folder in which the new project will be
84: (4)            initialized.
85: (8)            """
86: (4)            if args["project_name"]:
87: (8)                project_name = args["project_name"]
88: (4)            else:
89: (8)                project_name = click.prompt("Project Name", type=Path)
90: (8)            template_name = click.prompt(
91: (8)                "Template",
92: (8)                type=click.Choice(get_template_names(), False),
93: (8)                default="Default",
94: (4)            )
95: (12)            if project_name.is_dir():
96: (8)                console.print(
97: (8)                    f"\nFolder [red]{project_name}[/red] exists. Please type another
98: (8)                    name\n",
99: (8)                )
100: (8)            else:
101: (8)                project_name.mkdir()
102: (12)            new_cfg = {}
103: (16)            new_cfg_path = Path.resolve(project_name / "manim.cfg")
104: (20)            if not default_settings:
105: (16)                for key, value in CFG_DEFAULTS.items():
106: (20)                    if key == "scene_names":
107: (16)                        new_cfg[key] = template_name + "Template"
108: (20)                    elif key == "resolution":
109: (12)                        new_cfg[key] = select_resolution()
110: (12)                    else:
111: (16)                        new_cfg[key] = click.prompt(f"\n{key}", default=value)
112: (16)            console.print("\n", new_cfg)
113: (8)            if click.confirm("Do you want to continue?", default=True,
114: (12)                abort=True):
115: (12)                copy_template_files(project_name, template_name)
116: (16)                update_cfg(new_cfg, new_cfg_path)
117: (12)            else:
118: (16)                copy_template_files(project_name, template_name)
119: (12)                update_cfg(CFG_DEFAULTS, new_cfg_path)
120: (0)        @cloup.command(
121: (4)            context_settings=CONTEXT_SETTINGS,
122: (4)            no_args_is_help=True,
123: (4)            epilog=EPILOG,
124: (0)
125: (0)            @cloup.argument("scene_name", type=str, required=True)
126: (0)            @cloup.argument("file_name", type=str, required=False)
127: (0)        def scene(**args):
128: (4)            """Inserts a SCENE to an existing FILE or creates a new FILE.
129: (4)            SCENE is the name of the scene that will be inserted.
130: (4)            FILE is the name of file in which the SCENE will be inserted.
131: (4)            """
132: (4)            template_name = click.prompt(
133: (8)                "template",
134: (8)                type=click.Choice(get_template_names(), False),
135: (8)                default="Default",
136: (4)            )
137: (4)            scene = (get_template_path() / f"
138: (4)                {template_name}.mtp").resolve().read_text()
139: (4)            scene = scene.replace(template_name + "Template", args["scene_name"], 1)
140: (4)            if args["file_name"]:
141: (8)                file_name = Path(args["file_name"])

```

```

137: (8)             if file_name.suffix != ".py":
138: (12)             file_name = file_name.with_suffix(file_name.suffix + ".py")
139: (8)             if file_name.is_file():
140: (12)                 with file_name.open("a") as f:
141: (16)                     f.write("\n\n\n" + scene)
142: (8)             else:
143: (12)                 file_name.write_text("\n\n\n" + scene)
144: (12)                 add_import_statement(file_name)
145: (4)             else:
146: (8)                 with Path("main.py").open("a") as f:
147: (12)                     f.write("\n\n\n" + scene)
148: (0)             @cloup.group(
149: (4)                 context_settings=CONTEXT_SETTINGS,
150: (4)                 invoke_without_command=True,
151: (4)                 no_args_is_help=True,
152: (4)                 epilog=EPILOG,
153: (4)                 help="Create a new project or insert a new scene.",
154: (0)             )
155: (0)             @cloup.pass_context
156: (0)             def init(ctx):
157: (4)                 pass
158: (0)                 init.add_command(project)
159: (0)                 init.add_command(scene)

```

-----

File 38 - \_\_init\_\_.py:

```
1: (0)
```

-----

File 39 - commands.py:

```

1: (0)             """Manim's plugin subcommand.
2: (0)             Manim's plugin subcommand is accessed in the command-line interface via
```manim
3: (0)             plugin``. Here you can specify options, subcommands, and subgroups for the
plugin
4: (0)             group.
5: (0)             """
6: (0)             from __future__ import annotations
7: (0)             import cloup
8: (0)             from ...constants import CONTEXT_SETTINGS, EPILOG
9: (0)             from ...plugins.plugins_flags import list_plugins
10: (0)             __all__ = ["plugins"]
11: (0)             @cloup.command(
12: (4)                 context_settings=CONTEXT_SETTINGS,
13: (4)                 no_args_is_help=True,
14: (4)                 epilog=EPILOG,
15: (4)                 help="Manages Manim plugins.",
16: (0)             )
17: (0)             @cloup.option(
18: (4)                 "-l",
19: (4)                 "--list",
20: (4)                 "list_available",
21: (4)                 is_flag=True,
22: (4)                 help="List available plugins.",
23: (0)             )
24: (0)             def plugins(list_available):
25: (4)                 if list_available:
26: (8)                     list_plugins()

```

-----

File 40 - \_\_init\_\_.py:

```
1: (0)
```

## File 41 - commands.py:

```

1: (0)         """Manim's default subcommand, render.
2: (0)         Manim's render subcommand is accessed in the command-line interface via
3: (0)         ```manim```, but can be more explicitly accessed with ```manim render````. Here you
4: (0)         can specify options, and arguments for the render command.
5: (0)         """
6: (0)         from __future__ import annotations
7: (0)         import http.client
8: (0)         import json
9: (0)         import sys
10: (0)        import urllib.error
11: (0)        import urllib.request
12: (0)        from pathlib import Path
13: (0)        from typing import cast
14: (0)        import cloup
15: (0)        from ... import __version__, config, console, error_console, logger
16: (0)        from ..._config import tempconfig
17: (0)        from ...constants import EPILOG, RendererType
18: (0)        from ...utils.module_ops import scene_classes_from_file
19: (0)        from .ease_of_access_options import ease_of_access_options
20: (0)        from .global_options import global_options
21: (0)        from .output_options import output_options
22: (0)        from .render_options import render_options
23: (0)        __all__ = ["render"]
24: (0)        @cloup.command(
25: (4)            context_settings=None,
26: (4)            no_args_is_help=True,
27: (4)            epilog=EPILOG,
28: (0)        )
29: (0)        @cloup.argument("file", type=Path, required=True)
30: (0)        @cloup.argument("scene_names", required=False, nargs=-1)
31: (0)        @global_options
32: (0)        @output_options
33: (0)        @render_options # type: ignore
34: (0)        @ease_of_access_options
35: (0)        def render(
36: (4)            **args,
37: (0)        ):
38: (4)            """Render SCENE(S) from the input FILE.
39: (4)            FILE is the file path of the script or a config file.
40: (4)            SCENES is an optional list of scenes in the file.
41: (4)            """
42: (4)            if args["save_as_gif"]:
43: (8)                logger.warning("--save_as_gif is deprecated, please use --format=gif
instead!")
44: (8)
45: (4)            if args["save_pngs"]:
46: (8)                logger.warning("--save_pngs is deprecated, please use --format=png
instead!")
47: (8)
48: (4)            if args["show_in_file_browser"]:
49: (8)                logger.warning(
50: (12)                    "The short form of show_in_file_browser is deprecated and will be
moved to support --format.",
51: (8)                )
52: (4)                class ClickArgs:
53: (8)                    def __init__(self, args):
54: (12)                        for name in args:
55: (16)                            setattr(self, name, args[name])
56: (8)
57: (12)                    def __get_kwargs(self):
58: (8)                        return list(self.__dict__.items())
59: (12)                    def __eq__(self, other):
60: (16)                        if not isinstance(other, ClickArgs):
61: (12)                            return NotImplemented
62: (8)                        return vars(self) == vars(other)
63: (8)                    def __contains__(self, key):

```

```

63: (12)             return key in self.__dict__
64: (8)              def __repr__(self):
65: (12)                  return str(self.__dict__)
66: (4)                  click_args = ClickArgs(args)
67: (4)                  if args["jupyter"]:
68: (8)                      return click_args
69: (4)                  config.digest_args(click_args)
70: (4)                  file = Path(config.input_file)
71: (4)                  if config.renderer == RendererType.OPENGL:
72: (8)                      from manim.renderer.opengl_renderer import OpenGLRenderer
73: (8)                      try:
74: (12)                          renderer = OpenGLRenderer()
75: (12)                          keep_running = True
76: (12)                          while keep_running:
77: (16)                              for SceneClass in scene_classes_from_file(file):
78: (20)                                  with tempconfig({}):
79: (24)                                      scene = SceneClass(renderer)
80: (24)                                      rerun = scene.render()
81: (20)                                      if rerun or config["write_all"]:
82: (24)  renderer.num_plays = 0
83: (24)  continue
84: (20)                                      else:
85: (24)  keep_running = False
86: (24)  break
87: (16)  if config["write_all"]:
88: (20)  keep_running = False
89: (8)              except Exception:
90: (12)                  error_console.print_exception()
91: (12)                  sys.exit(1)
92: (4)              else:
93: (8)                  for SceneClass in scene_classes_from_file(file):
94: (12)                      try:
95: (16)                          with tempconfig({}):
96: (20)                              scene = SceneClass()
97: (20)                              scene.render()
98: (12)                          except Exception:
99: (16)                              error_console.print_exception()
100: (16)                             sys.exit(1)
101: (4)                  if config.notify_outdated_version:
102: (8)                      manim_info_url = "https://pypi.org/pypi/manim/json"
103: (8)                      warn_prompt = "Cannot check if latest release of manim is installed"
104: (8)                      try:
105: (12)                          with urllib.request.urlopen(
106: (16)                              urllib.request.Request(manim_info_url),
107: (16)                              timeout=10,
108: (12)                          ) as response:
109: (16)                              response = cast(http.client.HTTPResponse, response)
110: (16)                              json_data = json.loads(response.read())
111: (8)                      except urllib.error.HTTPError:
112: (12)                          logger.debug("HTTP Error: %s", warn_prompt)
113: (8)                      except urllib.error.URLError:
114: (12)                          logger.debug("URL Error: %s", warn_prompt)
115: (8)                      except json.JSONDecodeError:
116: (12)                          logger.debug(
117: (16)                              "Error while decoding JSON from %r: %s", manim_info_url,
warn_prompt
118: (12) )
119: (8)              except Exception:
120: (12)                  logger.debug("Something went wrong: %s", warn_prompt)
121: (8)              else:
122: (12)                  stable = json_data["info"]["version"]
123: (12)                  if stable != __version__:
124: (16)                      console.print(
125: (20)                          f"You are using manim version [red]{stable}[/red],"
but version [green]{stable}[/green] is available.",
126: (16) )
127: (16)                      console.print(
128: (20)                          "You should consider upgrading via [yellow]pip install -U"
manim[/yellow]",
```

```
129: (16) )
130: (4)     return args
```

-----  
File 42 - \_\_init\_\_.py:

```
1: (0)
```

-----  
File 43 - default\_group.py:

```
1: (0)     """``DefaultGroup`` allows a subcommand to act as the main command.
2: (0)     In particular, this class is what allows ``manim`` to act as ``manim render``.
3: (0)     .. note::
4: (4)         This is a vendored version of https://github.com/click-contrib/click-
default-group/
5: (4)         under the BSD 3-Clause "New" or "Revised" License.
6: (4)         This library isn't used as a dependency as we need to inherit from
``cloop.Group`` instead
7: (4)             of ``click.Group``.
8: (0) """
9: (0)     import warnings
10: (0)    import cloup
11: (0)    __all__ = ["DefaultGroup"]
12: (0)    class DefaultGroup(cloop.Group):
13: (4)        """Invokes a subcommand marked with ``default=True`` if any subcommand not
14: (4) chosen.
15: (4) """
16: (4)        def __init__(self, *args, **kwargs):
17: (8)            if not kwargs.get("ignore_unknown_options", True):
18: (12)                raise ValueError("Default group accepts unknown options")
19: (8)            self.ignore_unknown_options = True
20: (8)            self.default_cmd_name = kwargs.pop("default", None)
21: (8)            self.default_if_no_args = kwargs.pop("default_if_no_args", False)
22: (8)            super().__init__(*args, **kwargs)
23: (4)        def set_default_command(self, command):
24: (8)            """Sets a command function as the default command."""
25: (8)            cmd_name = command.name
26: (8)            self.add_command(command)
27: (8)            self.default_cmd_name = cmd_name
28: (4)        def parse_args(self, ctx, args):
29: (8)            if not args and self.default_if_no_args:
30: (12)                args.insert(0, self.default_cmd_name)
31: (8)            return super().parse_args(ctx, args)
32: (4)        def get_command(self, ctx, cmd_name):
33: (8)            if cmd_name not in self.commands:
34: (12)                ctx.arg0 = cmd_name
35: (12)                cmd_name = self.default_cmd_name
36: (8)            return super().get_command(ctx, cmd_name)
37: (4)        def resolve_command(self, ctx, args):
38: (8)            base = super()
39: (8)            cmd_name, cmd, args = base.resolve_command(ctx, args)
40: (8)            if hasattr(ctx, "arg0"):
41: (12)                args.insert(0, ctx.arg0)
42: (12)                cmd_name = cmd.name
43: (8)            return cmd_name, cmd, args
44: (4)        def command(self, *args, **kwargs):
45: (8)            default = kwargs.pop("default", False)
46: (8)            decorator = super().command(*args, **kwargs)
47: (8)            if not default:
48: (12)                return decorator
49: (8)            warnings.warn(
50: (12)                "Use default param of DefaultGroup or set_default_command()
instead",
51: (12)                DeprecationWarning,
52: (8)            )
53: (8)            def _decorator(f):
```

```

54: (12)             cmd = decorator(f)
55: (12)             self.set_default_command(cmd)
56: (12)             return cmd
57: (8)              return _decorator
-----
```

File 44 - global\_options.py:

```

1: (0)          from __future__ import annotations
2: (0)          import re
3: (0)          from cloup import Choice, option, option_group
4: (0)          from ... import logger
5: (0)          __all__ = ["global_options"]
6: (0)          def validate_gui_location(ctx, param, value):
7: (4)            if value:
8: (8)              try:
9: (12)                x_offset, y_offset = map(int, re.split(r"[;,\-]", value))
10: (12)               return (x_offset, y_offset)
11: (8)            except Exception:
12: (12)              logger.error("GUI location option is invalid.")
13: (12)              exit()
14: (0)          global_options = option_group(
15: (4)            "Global options",
16: (4)            option(
17: (8)              "-c",
18: (8)              "--config_file",
19: (8)              help="Specify the configuration file to use for render settings.",
20: (8)              default=None,
21: (4)            ),
22: (4)            option(
23: (8)              "--custom_folders",
24: (8)              is_flag=True,
25: (8)              default=None,
26: (8)              help="Use the folders defined in the [custom_folders] section of the "
27: (8)              "config file to define the output folder structure.",
28: (4)            ),
29: (4)            option(
30: (8)              "--disable_caching",
31: (8)              is_flag=True,
32: (8)              default=None,
33: (8)              help="Disable the use of the cache (still generates cache files.).",
34: (4)            ),
35: (4)            option(
36: (8)              "--flush_cache",
37: (8)              is_flag=True,
38: (8)              help="Remove cached partial movie files.",
39: (8)              default=None,
40: (4)            ),
41: (4)            option("--tex_template", help="Specify a custom TeX template file.",
default=None),
42: (4)            option(
43: (8)              "-v",
44: (8)              "--verbosity",
45: (8)              type=Choice(
46: (12)                ["DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"],
47: (12)                case_sensitive=False,
48: (8)              ),
49: (8)              help="Verbosity of CLI output. Changes ffmpeg log level unless 5+.",
50: (8)              default=None,
51: (4)            ),
52: (4)            option(
53: (8)              "--notify_outdated_version/--silent",
54: (8)              is_flag=True,
55: (8)              default=None,
56: (8)              help="Display warnings for outdated installation.",
57: (4)            ),
58: (4)            option(
59: (8)              "--enable_gui",
```

```

60: (8)                  is_flag=True,
61: (8)                  help="Enable GUI interaction.",
62: (8)                  default=None,
63: (4) ),
64: (4)     option(
65: (8)         "--gui_location",
66: (8)         default=None,
67: (8)         callback=validate_gui_location,
68: (8)         help="Starting location for the GUI.",
69: (4) ),
70: (4)     option(
71: (8)         "--fullscreen",
72: (8)         is_flag=True,
73: (8)         help="Expand the window to its maximum possible size.",
74: (8)         default=None,
75: (4) ),
76: (4)     option(
77: (8)         "--enable_wireframe",
78: (8)         is_flag=True,
79: (8)         help="Enable wireframe debugging mode in opengl.",
80: (8)         default=None,
81: (4) ),
82: (4)     option(
83: (8)         "--force_window",
84: (8)         is_flag=True,
85: (8)         help="Force window to open when using the opengl renderer, intended
for debugging as it may impact performance",
86: (8)         default=False,
87: (4) ),
88: (4)     option(
89: (8)         "--dry_run",
90: (8)         is_flag=True,
91: (8)         help="Renders animations without outputting image or video files and
disables the window",
92: (8)         default=False,
93: (4) ),
94: (4)     option(
95: (8)         "--no_latex_cleanup",
96: (8)         is_flag=True,
97: (8)         help="Prevents deletion of .aux, .dvi, and .log files produced by Tex
and MathTex.",
98: (8)         default=False,
99: (4) ),
100: (4)    option(
101: (8)        "--preview_command",
102: (8)        help="The command used to preview the output file (for example vlc for
video files)",
103: (8)        default="",
104: (4) ),
105: (0) )

```

---

File 45 - ease\_of\_access\_options.py:

```

1: (0)                  from __future__ import annotations
2: (0)                  from cloup import Choice, option, option_group
3: (0)                  __all__ = ["ease_of_access_options"]
4: (0)                  ease_of_access_options = option_group(
5: (4)                      "Ease of access options",
6: (4)                      option(
7: (8)                          "--progress_bar",
8: (8)                          default=None,
9: (8)                          show_default=False,
10: (8)                          type=Choice(
11: (12)                              ["display", "leave", "none"],
12: (12)                              case_sensitive=False,
13: (8)                          ),
14: (8)                          help="Display progress bars and/or keep them displayed."),

```

```

15: (4)                 ),
16: (4)                 option(
17: (8)                   "-p",
18: (8)                   "--preview",
19: (8)                   is_flag=True,
20: (8)                   help="Preview the Scene's animation. OpenGL does a live preview in a "
21: (8)                   "popup window. Cairo opens the rendered video file in the system "
22: (8)                   "default media player.",
23: (8)                   default=None,
24: (4)             ),
25: (4)             option(
26: (8)               "-f",
27: (8)               "--show_in_file_browser",
28: (8)               is_flag=True,
29: (8)               help="Show the output file in the file browser.",
30: (8)               default=None,
31: (4)         ),
32: (4)         option(
33: (8)           "--jupyter",
34: (8)           is_flag=True,
35: (8)           help="Using jupyter notebook magic.",
36: (8)           default=None,
37: (4)     ),
38: (0)   )
-----
```

#### File 46 - output\_options.py:

```

1: (0)                 from __future__ import annotations
2: (0)                 from cloup import IntRange, Path, option, option_group
3: (0)                 __all__ = ["output_options"]
4: (0)                 output_options = option_group(
5: (4)                   "Output options",
6: (4)                   option(
7: (8)                     "-o",
8: (8)                     "--output_file",
9: (8)                     type=str,
10: (8)                    default=None,
11: (8)                     help="Specify the filename(s) of the rendered scene(s).",
12: (4)       ),
13: (4)       option(
14: (8)         "-0",
15: (8)         "--zero_pad",
16: (8)         type=IntRange(0, 9),
17: (8)         default=None,
18: (8)         help="Zero padding for PNG file names.",
19: (4)   ),
20: (4)   option(
21: (8)     "--write_to_movie",
22: (8)     is_flag=True,
23: (8)     default=None,
24: (8)     help="Write the video rendered with opengl to a file.",
25: (4)   ),
26: (4)   option(
27: (8)     "--media_dir",
28: (8)     type=Path(),
29: (8)     default=None,
30: (8)     help="Path to store rendered videos and latex.",
31: (4)   ),
32: (4)   option(
33: (8)     "--log_dir",
34: (8)     type=Path(),
35: (8)     help="Path to store render logs.",
36: (8)     default=None,
37: (4)   ),
38: (4)   option(
39: (8)     "--log_to_file",
40: (8)     is_flag=True,
```

```

41: (8)             default=None,
42: (8)             help="Log terminal output to file.",
43: (4)         ),
44: (0)     )
-----
```

File 47 - gui.py:

```

1: (0)         from __future__ import annotations
2: (0)         from pathlib import Path
3: (0)         try:
4: (4)             import dearpygui.dearpygui as dpg
5: (4)             dearpygui_imported = True
6: (0)         except ImportError:
7: (4)             dearpygui_imported = False
8: (0)         from .. import __version__, config
9: (0)         from ..utils.module_ops import scene_classes_from_file
10: (0)        __all__ = ["configure_pygui"]
11: (0)        if dearpygui_imported:
12: (4)            dpg.create_context()
13: (4)            window = dpg.generate_uuid()
14: (0)        def configure_pygui(renderer, widgets, update=True):
15: (4)            if not dearpygui_imported:
16: (8)                raise RuntimeError("Attempted to use DearPyGUI when it isn't
imported.")
17: (4)            if update:
18: (8)                dpg.delete_item(window)
19: (4)            else:
20: (8)                dpg.create_viewport()
21: (8)                dpg.setup_dearpygui()
22: (8)                dpg.show_viewport()
23: (4)                dpg.set_viewport_title(title=f"Manim Community v{__version__}")
24: (4)                dpg.set_viewport_width(1015)
25: (4)                dpg.set_viewport_height(540)
26: (4)                def rerun_callback(sender, data):
27: (8)                    renderer.scene.queue.put(("rerun_gui", [], {}))
28: (4)                def continue_callback(sender, data):
29: (8)                    renderer.scene.queue.put(("exit_gui", [], {}))
30: (4)                def scene_selection_callback(sender, data):
31: (8)                    config["scene_names"] = (dpg.get_value(sender),)
32: (8)                    renderer.scene.queue.put(("rerun_gui", [], {}))
33: (4)                    scene_classes = scene_classes_from_file(Path(config["input_file"]),
full_list=True)
34: (4)                    scene_names = [scene_class.__name__ for scene_class in scene_classes]
35: (4)                    with dpg.window(
36: (8)                        id=window,
37: (8)                        label="Manim GUI",
38: (8)                        pos=[config["gui_location"][0], config["gui_location"][1]],
39: (8)                        width=1000,
40: (8)                        height=500,
41: (4)                    ):
42: (8)                        dpg.set_global_font_scale(2)
43: (8)                        dpg.add_button(label="Rerun", callback=rerun_callback)
44: (8)                        dpg.add_button(label="Continue", callback=continue_callback)
45: (8)                        dpg.add_combo(
46: (12)                            label="Selected scene",
47: (12)                            items=scene_names,
48: (12)                            callback=scene_selection_callback,
49: (12)                            default_value=config["scene_names"][0],
50: (8)                        )
51: (8)                        dpg.add_separator()
52: (8)                        if len(widgets) != 0:
53: (12)                            with dpg.collapsing_header(
54: (16)                                label=f"{config['scene_names'][0]} widgets",
55: (16)                                default_open=True,
56: (12)                            ):
57: (16)                                for widget_config in widgets:
58: (20)                                    widget_config_copy = widget_config.copy()
```

```

59: (20)                         name = widget_config_copy["name"]
60: (20)                         widget = widget_config_copy["widget"]
61: (20)                         if widget != "separator":
62: (24)                             del widget_config_copy["name"]
63: (24)                             del widget_config_copy["widget"]
64: (24)                             setattr(dpg, f"add_{widget}")(label=name,
**widget_config_copy)
65: (20)                         else:
66: (24)                             dpg.add_separator()
67: (4)                           if not update:
68: (8)                             dpg.start_dearpygui()

-----

```

## File 48 - arc.py:

```

1: (0)             r"""Mobjects that are curved.
2: (0)             Examples
3: (0)             -----
4: (0)             .. manim:: UsefulAnnotations
5: (4)               :save_last_frame:
6: (4)               class UsefulAnnotations(Scene):
7: (8)                 def construct(self):
8: (12)                   m0 = Dot()
9: (12)                   m1 = AnnotationDot()
10: (12)                  m2 = LabeledDot("ii")
11: (12)                  m3 = LabeledDot(MathTex(r"\alpha").set_color(ORANGE))
12: (12)                  m4 = CurvedArrow(2*LEFT, 2*RIGHT, radius= -5)
13: (12)                  m5 = CurvedArrow(2*LEFT, 2*RIGHT, radius= 8)
14: (12)                  m6 = CurvedDoubleArrow(ORIGIN, 2*RIGHT)
15: (12)                  self.add(m0, m1, m2, m3, m4, m5, m6)
16: (12)                  for i, mobj in enumerate(self.mobjects):
17: (16)                      mobj.shift(DOWN * (i-3))
18: (0)             """
19: (0)             from __future__ import annotations
20: (0)             __all__ = [
21: (4)                 "TipableVMobject",
22: (4)                 "Arc",
23: (4)                 "ArcBetweenPoints",
24: (4)                 "CurvedArrow",
25: (4)                 "CurvedDoubleArrow",
26: (4)                 "Circle",
27: (4)                 "Dot",
28: (4)                 "AnnotationDot",
29: (4)                 "LabeledDot",
30: (4)                 "Ellipse",
31: (4)                 "AnnularSector",
32: (4)                 "Sector",
33: (4)                 "Annulus",
34: (4)                 "CubicBezier",
35: (4)                 "ArcPolygon",
36: (4)                 "ArcPolygonFromArcs",
37: (0)             ]
38: (0)             import itertools
39: (0)             import warnings
40: (0)             from typing import TYPE_CHECKING
41: (0)             import numpy as np
42: (0)             from typing_extensions import Self
43: (0)             from manim.constants import *
44: (0)             from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
45: (0)             from manim.mobject.types.vectorized_mobject import VGroup, VMobject
46: (0)             from manim.utils.color import BLACK, BLUE, RED, WHITE, ParsableManimColor
47: (0)             from manim.utils.iterables import adjacent_pairs
48: (0)             from manim.utils.space_ops import (
49: (4)                 angle_of_vector,
50: (4)                 cartesian_to_spherical,
51: (4)                 line_intersection,
52: (4)                 perpendicular_bisector,
53: (4)                 rotate_vector,

```

```

54: (0) )
55: (0)     if TYPE_CHECKING:
56: (4)         import manim.mobject.geometry.tips as tips
57: (4)         from manim.mobject.mobject import Mobject
58: (4)         from manim.mobject.text.tex_mobject import SingleStringMathTex, Tex
59: (4)         from manim.mobject.text.text_mobject import Text
60: (4)         from manim.typing import CubicBezierPoints, Point3D,
QuadraticBezierPoints, Vector3D
61: (0)     class TipableVMobject(VMobject, metaclass=ConvertToOpenGL):
62: (4)         """Meant for shared functionality between Arc and Line.
63: (4)         Functionality can be classified broadly into these groups:
64: (8)             * Adding, Creating, Modifying tips
65: (12)                 - add_tip calls create_tip, before pushing the new tip
66: (16)                     into the TipableVMobject's list of submobjects
67: (12)                 - stylistic and positional configuration
68: (8)             * Checking for tips
69: (12)                 - Boolean checks for whether the TipableVMobject has a tip
70: (16)                     and a starting tip
71: (8)             * Getters
72: (12)                 - Straightforward accessors, returning information pertaining
73: (16)                     to the TipableVMobject instance's tip(s), its length etc
74: (4)         """
75: (4)     def __init__(
76: (8)         self,
77: (8)         tip_length: float = DEFAULT_ARROW_TIP_LENGTH,
78: (8)         normal_vector: Vector3D = OUT,
79: (8)         tip_style: dict = {},
80: (8)         **kwargs,
81: (4)     ) -> None:
82: (8)         self.tip_length: float = tip_length
83: (8)         self.normal_vector: Vector3D = normal_vector
84: (8)         self.tip_style: dict = tip_style
85: (8)         super().__init__(**kwargs)
86: (4)     def add_tip(
87: (8)         self,
88: (8)         tip: tips.ArrowTip | None = None,
89: (8)         tip_shape: type[tips.ArrowTip] | None = None,
90: (8)         tip_length: float | None = None,
91: (8)         tip_width: float | None = None,
92: (8)         at_start: bool = False,
93: (4)     ) -> Self:
94: (8)         """Adds a tip to the TipableVMobject instance, recognising
95: (8)             that the endpoints might need to be switched if it's
96: (8)             a 'starting tip' or not.
97: (8)         """
98: (8)         if tip is None:
99: (12)             tip = self.create_tip(tip_shape, tip_length, tip_width, at_start)
100: (8)         else:
101: (12)             self.position_tip(tip, at_start)
102: (8)             self.reset_endpoints_based_on_tip(tip, at_start)
103: (8)             self.assign_tip_attr(tip, at_start)
104: (8)             self.add(tip)
105: (8)             return self
106: (4)     def create_tip(
107: (8)         self,
108: (8)         tip_shape: type[tips.ArrowTip] | None = None,
109: (8)         tip_length: float = None,
110: (8)         tip_width: float = None,
111: (8)         at_start: bool = False,
112: (4)     ):
113: (8)         """Stylises the tip, positions it spatially, and returns
114: (8)             the newly instantiated tip to the caller.
115: (8)         """
116: (8)         tip = self.get_unpositioned_tip(tip_shape, tip_length, tip_width)
117: (8)         self.position_tip(tip, at_start)
118: (8)         return tip
119: (4)     def get_unpositioned_tip(
120: (8)         self,
121: (8)         tip_shape: type[tips.ArrowTip] | None = None,

```

```

122: (8)             tip_length: float | None = None,
123: (8)             tip_width: float | None = None,
124: (4)
125: (8)         ): """Returns a tip that has been stylistically configured,
126: (8)         but has not yet been given a position in space.
127: (8)         """
128: (8)             """
129: (8)             from manim.mobject.geometry.tips import ArrowTriangleFilledTip
130: (8)             style = {}
131: (12)             if tip_shape is None:
132: (8)                 tip_shape = ArrowTriangleFilledTip
133: (12)             if tip_shape is ArrowTriangleFilledTip:
134: (16)                 if tip_width is None:
135: (12)                     tip_width = self.get_default_tip_length()
136: (8)                     style.update({"width": tip_width})
137: (12)             if tip_length is None:
138: (8)                 tip_length = self.get_default_tip_length()
139: (8)             color = self.get_color()
140: (8)             style.update({"fill_color": color, "stroke_color": color})
141: (8)             style.update(self.tip_style)
142: (8)             tip = tip_shape(length=tip_length, **style)
143: (4)             return tip
144: (8)
145: (12)         def position_tip(self, tip: tips.ArrowTip, at_start: bool = False):
146: (12)             if at_start:
147: (8)                 anchor = self.get_start()
148: (12)                 handle = self.get_first_handle()
149: (12)
150: (8)
151: (8)
152: (12)
153: (8)
154: (8)
155: (12)
156: (16)
157: (16)
158: (16)
159: (12)
160: (12)
161: (16)
162: (16)
163: (12)
164: (12)
165: (8)
166: (8)
167: (4)
168: (8)
169: (12)
170: (8)
171: (12)
172: (8)
173: (12)
174: (8)
175: (4)
176: (8)
177: (12)
178: (8)
179: (12)
180: (8)
181: (4)
182: (8)
183: (4)
184: (8)
185: (4)
186: (8)
187: (8)
188: (8)
189: (12)

             tip.rotate(
                 angles[1] - PI - tip.tip_angle,
             ) # Rotates the tip along the azimuthal
             if not hasattr(self, "_init_positioning_axis"):
                 axis = [
                     np.sin(angles[1]),
                     -np.cos(angles[1]),
                     0,
                 ] # Obtains the perpendicular of the tip
             tip.rotate(
                 -angles[2] + PI / 2,
                 axis=axis,
             ) # Rotates the tip along the vertical wrt the axis
             self._init_positioning_axis = axis
             tip.shift(anchor - tip.tip_point)
             return tip
167: (4)         def reset_endpoints_based_on_tip(self, tip: tips.ArrowTip, at_start: bool)
-> Self:
168: (8)             if self.get_length() == 0:
169: (12)                 return self
170: (8)
171: (12)             if at_start:
172: (8)                 self.put_start_and_end_on(tip.base, self.get_end())
173: (8)
174: (12)
175: (8)
176: (12)
177: (8)
178: (12)
179: (8)
180: (12)
181: (8)
182: (12)
183: (8)
184: (12)
185: (8)
186: (12)
187: (8)
188: (12)
189: (12)

             if at_start:
                 self.start_tip = tip
             else:
                 self.tip = tip
             return self
181: (4)         def has_tip(self) -> bool:
182: (8)             return hasattr(self, "tip") and self.tip in self
183: (4)         def has_start_tip(self) -> bool:
184: (8)             return hasattr(self, "start_tip") and self.start_tip in self
185: (4)         def pop_tips(self) -> VGroup:
186: (8)             start, end = self.get_start_and_end()
187: (8)             result = self.get_group_class()()
188: (8)
189: (12)             if self.has_tip():
                 result.add(self.tip)

```

```

190: (12)                     self.remove(self.tip)
191: (8)                      if self.has_start_tip():
192: (12)                        result.add(self.start_tip)
193: (12)                        self.remove(self.start_tip)
194: (8)                      self.put_start_and_end_on(start, end)
195: (8)                      return result
196: (4)  def get_tips(self) -> VGroup:
197: (8)    """Returns a VGroup (collection of VMobjects) containing
198: (8)    the TipableVMObject instance's tips.
199: (8)
200: (8)    result = self.get_group_class()()
201: (8)    if hasattr(self, "tip"):
202: (12)      result.add(self.tip)
203: (8)    if hasattr(self, "start_tip"):
204: (12)      result.add(self.start_tip)
205: (8)    return result
206: (4)  def get_tip(self):
207: (8)    """Returns the TipableVMobject instance's (first) tip,
208: (8)    otherwise throws an exception."""
209: (8)    tips = self.get_tips()
210: (8)    if len(tips) == 0:
211: (12)      raise Exception("tip not found")
212: (8)    else:
213: (12)      return tips[0]
214: (4)  def get_default_tip_length(self) -> float:
215: (8)    return self.tip_length
216: (4)  def get_first_handle(self) -> Point3D:
217: (8)    return self.points[1]
218: (4)  def get_last_handle(self) -> Point3D:
219: (8)    return self.points[-2]
220: (4)  def get_end(self) -> Point3D:
221: (8)    if self.has_tip():
222: (12)      return self.tip.get_start()
223: (8)    else:
224: (12)      return super().get_end()
225: (4)  def get_start(self) -> Point3D:
226: (8)    if self.has_start_tip():
227: (12)      return self.start_tip.get_start()
228: (8)    else:
229: (12)      return super().get_start()
230: (4)  def get_length(self) -> np.floating:
231: (8)    start, end = self.get_start_and_end()
232: (8)    return np.linalg.norm(start - end)
233: (0)  class Arc(TipableVMobject):
234: (4)    """A circular arc.
235: (4)    Examples
236: (4)    -----
237: (4)    A simple arc of angle Pi.
238: (4)    .. manim:: ArcExample
239: (8)      :save_last_frame:
240: (8)      class ArcExample(Scene):
241: (12)        def construct(self):
242: (16)          self.add(Arc(angle=PI))
243: (4)
244: (4)        def __init__(
245: (8)          self,
246: (8)          radius: float = 1.0,
247: (8)          start_angle: float = 0,
248: (8)          angle: float = TAU / 4,
249: (8)          num_components: int = 9,
250: (8)          arc_center: Point3D = ORIGIN,
251: (8)          **kwargs,
252: (4)        ):
253: (8)          if radius is None: # apparently None is passed by ArcBetweenPoints
254: (12)            radius = 1.0
255: (8)          self.radius = radius
256: (8)          self.num_components: int = num_components
257: (8)          self.arc_center: Point3D = arc_center
258: (8)          self.start_angle: float = start_angle

```

```

259: (8)             self.angle: float = angle
260: (8)             self._failed_to_get_center: bool = False
261: (8)             super().__init__(**kwargs)
262: (4)         def generate_points(self) -> None:
263: (8)             self._set_pre_positioned_points()
264: (8)             self.scale(self.radius, about_point=ORIGIN)
265: (8)             self.shift(self.arc_center)
266: (4)         def init_points(self) -> None:
267: (8)             self.set_points(
268: (12)                 Arc._create_quadratic_bezier_points(
269: (16)                     angle=self.angle,
270: (16)                     start_angle=self.start_angle,
271: (16)                     n_components=self.num_components,
272: (12)                     ),
273: (8)                 )
274: (8)             self.scale(self.radius, about_point=ORIGIN)
275: (8)             self.shift(self.arc_center)
276: (4)         @staticmethod
277: (4)         def _create_quadratic_bezier_points(
278: (8)             angle: float, start_angle: float = 0, n_components: int = 8
279: (4)         ) -> QuadraticBezierPoints:
280: (8)             samples = np.array(
281: (12)                 [
282: (16)                     [np.cos(a), np.sin(a), 0]
283: (16)                     for a in np.linspace(
284: (20)                         start_angle,
285: (20)                         start_angle + angle,
286: (20)                         2 * n_components + 1,
287: (16)                         )
288: (12)                     ],
289: (8)                 )
290: (8)             theta = angle / n_components
291: (8)             samples[1::2] /= np.cos(theta / 2)
292: (8)             points = np.zeros((3 * n_components, 3))
293: (8)             points[0::3] = samples[0:-1:2]
294: (8)             points[1::3] = samples[1::2]
295: (8)             points[2::3] = samples[2::2]
296: (8)             return points
297: (4)         def _set_pre_positioned_points(self) -> None:
298: (8)             anchors = np.array(
299: (12)                 [
300: (16)                     np.cos(a) * RIGHT + np.sin(a) * UP
301: (16)                     for a in np.linspace(
302: (20)                         self.start_angle,
303: (20)                         self.start_angle + self.angle,
304: (20)                         self.num_components,
305: (16)                         )
306: (12)                     ],
307: (8)                 )
308: (8)             d_theta = self.angle / (self.num_components - 1.0)
309: (8)             tangent_vectors = np.zeros(anchors.shape)
310: (8)             tangent_vectors[:, 1] = anchors[:, 0]
311: (8)             tangent_vectors[:, 0] = -anchors[:, 1]
312: (8)             handles1 = anchors[:-1] + (d_theta / 3) * tangent_vectors[:-1]
313: (8)             handles2 = anchors[1:] - (d_theta / 3) * tangent_vectors[1:]
314: (8)             self.set_anchors_and_handles(anchors[:-1], handles1, handles2,
anchors[1:])
315: (4)         def get_arc_center(self, warning: bool = True) -> Point3D:
316: (8)             """Looks at the normals to the first two
317: (8)             anchors, and finds their intersection points
318: (8)             """
319: (8)             a1, h1, h2, a2 = self.points[:4]
320: (8)             if np.all(a1 == a2):
321: (12)                 return np.copy(a1)
322: (8)             t1 = h1 - a1
323: (8)             t2 = h2 - a2
324: (8)             n1 = rotate_vector(t1, TAU / 4)
325: (8)             n2 = rotate_vector(t2, TAU / 4)
326: (8)             try:

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
327: (12)             return line_intersection(line1=(a1, a1 + n1), line2=(a2, a2 + n2))
328: (8)          except Exception:
329: (12)              if warning:
330: (16)                  warnings.warn("Can't find Arc center, using ORIGIN instead")
331: (12)              self._failed_to_get_center = True
332: (12)              return np.array(ORIGIN)
333: (4)          def move_arc_center_to(self, point: Point3D) -> Self:
334: (8)              self.shift(point - self.get_arc_center())
335: (8)              return self
336: (4)          def stop_angle(self) -> float:
337: (8)              return angle_of_vector(self.points[-1] - self.get_arc_center()) % TAU
338: (0)      class ArcBetweenPoints(Arc):
339: (4)          """Inherits from Arc and additionally takes 2 points between which the arc
is spanned.
340: (4)          Example
341: (4)          -----
342: (4)          .. manim:: ArcBetweenPointsExample
343: (6)          class ArcBetweenPointsExample(Scene):
344: (10)              def construct(self):
345: (14)                  circle = Circle(radius=2, stroke_color=GREY)
346: (14)                  dot_1 = Dot(color=GREEN).move_to([2, 0, 0]).scale(0.5)
347: (14)                  dot_1_text = Tex("(2,0)").scale(0.5).next_to(dot_1,
RIGHT).set_color(BLUE)
348: (14)
349: (14)                  UP).set_color(BLUE)
350: (14)                  stroke_color=YELLOW)
351: (14)                  self.add(circle, dot_1, dot_2, dot_1_text, dot_2_text)
352: (14)                  self.play(Create(arc))
353: (4)          """
354: (4)          def __init__(
355: (8)              self,
356: (8)              start: Point3D,
357: (8)              end: Point3D,
358: (8)              angle: float = TAU / 4,
359: (8)              radius: float = None,
360: (8)              **kwargs,
361: (4)          ) -> None:
362: (8)              if radius is not None:
363: (12)                  self.radius = radius
364: (12)                  if radius < 0:
365: (16)                      sign = -2
366: (16)                      radius *= -1
367: (12)                  else:
368: (16)                      sign = 2
369: (12)                  halfdist = np.linalg.norm(np.array(start) - np.array(end)) / 2
370: (12)                  if radius < halfdist:
371: (16)                      raise ValueError(
372: (20)                          """ArcBetweenPoints called with a radius that is
373: (28)                          smaller than half the distance between the
points."""",
374: (16)
375: (12)                  )
376: (12)                  arc_height = radius - np.sqrt(radius**2 - halfdist**2)
377: (8)                  angle = np.arccos((radius - arc_height) / radius) * sign
378: (8)                  super().__init__(radius=radius, angle=angle, **kwargs)
379: (12)                  if angle == 0:
380: (8)                      self.set_points_as_corners([LEFT, RIGHT])
381: (8)                  self.put_start_and_end_on(start, end)
382: (12)                  if radius is None:
383: (12)                      center = self.get_arc_center(warning=False)
384: (16)                      if not self._failed_to_get_center:
385: (12)                          self.radius = np.linalg.norm(np.array(start) -
np.array(center))
386: (12)                      else:
387: (16)                          self.radius = np.inf
388: (0)      class CurvedArrow(ArcBetweenPoints):
389: (4)          def __init__(self, start_point: Point3D, end_point: Point3D, **kwargs) ->
None:

```

```

389: (8)                     from manim.mobject.geometry.tips import ArrowTriangleFilledTip
390: (8)                     tip_shape = kwargs.pop("tip_shape", ArrowTriangleFilledTip)
391: (8)                     super().__init__(start_point, end_point, **kwargs)
392: (8)                     self.add_tip(tip_shape=tip_shape)
393: (0)             class CurvedDoubleArrow(CurvedArrow):
394: (4)                 def __init__(self, start_point: Point3D, end_point: Point3D, **kwargs) ->
None:
395: (8)                     if "tip_shape_end" in kwargs:
396: (12)                         kwargs["tip_shape"] = kwargs.pop("tip_shape_end")
397: (8)                     from manim.mobject.geometry.tips import ArrowTriangleFilledTip
398: (8)                     tip_shape_start = kwargs.pop("tip_shape_start",
ArrowTriangleFilledTip)
399: (8)                     super().__init__(start_point, end_point, **kwargs)
400: (8)                     self.add_tip(at_start=True, tip_shape=tip_shape_start)
401: (0)             class Circle(Arc):
402: (4)                 """A circle.
403: (4)                 Parameters
404: (4)                 -----
405: (4)                 color
406: (8)                     The color of the shape.
407: (4)                 kwargs
408: (8)                     Additional arguments to be passed to :class:`Arc`
409: (4)                 Examples
410: (4)                 -----
411: (4)                 .. manim:: CircleExample
412: (8)                     :save_last_frame:
413: (8)                     class CircleExample(Scene):
414: (12)                         def construct(self):
415: (16)                             circle_1 = Circle(radius=1.0)
416: (16)                             circle_2 = Circle(radius=1.5, color=GREEN)
417: (16)                             circle_3 = Circle(radius=1.0, color=BLUE_B, fill_opacity=1)
418: (16)                             circle_group = Group(circle_1, circle_2,
circle_3).arrange(buff=1)
419: (16)                             self.add(circle_group)
420: (4)                         """
421: (4)                 def __init__(
422: (8)                     self,
423: (8)                     radius: float | None = None,
424: (8)                     color: ParsableManimColor = RED,
425: (8)                     **kwargs,
426: (4)                 ) -> None:
427: (8)                     super().__init__(
428: (12)                         radius=radius,
429: (12)                         start_angle=0,
430: (12)                         angle=TAU,
431: (12)                         color=color,
432: (12)                         **kwargs,
433: (8)                     )
434: (4)                 def surround(
435: (8)                     self,
436: (8)                     mobject: Mobject,
437: (8)                     dim_to_match: int = 0,
438: (8)                     stretch: bool = False,
439: (8)                     buffer_factor: float = 1.2,
440: (4)                 ) -> Self:
441: (8)                     """Modifies a circle so that it surrounds a given mobject.
442: (8)                     Parameters
443: (8)                     -----
444: (8)                     mobject
445: (12)                         The mobject that the circle will be surrounding.
446: (8)                     dim_to_match
447: (8)                     buffer_factor
448: (12)                         Scales the circle with respect to the mobject. A `buffer_factor` <
1 makes the circle smaller than the mobject.
449: (8)                     stretch
450: (12)                         Stretches the circle to fit more tightly around the mobject. Note:
Does not work with :class:`Line`  

451: (8)                     Examples
452: (8)                     -----

```

```

453: (8)          .. manim:: CircleSurround
454: (12)         :save_last_frame:
455: (12)         class CircleSurround(Scene):
456: (16)           def construct(self):
457: (20)             triangle1 = Triangle()
458: (20)             circle1 = Circle().surround(triangle1)
459: (20)             group1 = Group(triangle1,circle1) # treat the two mobjects
as one
460: (20)
461: (20)
462: (20)
463: (20)
464: (20)
465: (20)
466: (20)
467: (20)
468: (8)           group = Group(group1, group2, group3).arrange(buff=1)
469: (8)           self.add(group)
470: (8)
471: (8)
472: (4)           """
473: (8)           self.replace(mobject, dim_to_match, stretch)
474: (8)           self.width = np.sqrt(mobject.width**2 + mobject.height**2)
475: (8)           return self.scale(buffer_factor)
476: (8)
477: (12)         def point_at_angle(self, angle: float) -> Point3D:
478: (8)             """Returns the position of a point on the circle.
479: (8)             Parameters
480: (8)             -----
481: (8)             angle
482: (8)                 The angle of the point along the circle in radians.
483: (8)             Returns
484: (8)             -----
485: (8)             :class:`numpy.ndarray`
486: (8)                 The location of the point along the circle's circumference.
487: (16)             Examples
488: (20)
489: (20)
490: (20)
491: (20)
492: (20)
493: (20)
494: (8)
495: (8)             .. manim:: PointAtAngleExample
496: (8)               :save_last_frame:
497: (8)               class PointAtAngleExample(Scene):
498: (8)                 def construct(self):
499: (16)                   circle = Circle(radius=2.0)
500: (20)                   p1 = circle.point_at_angle(PI/2)
501: (20)                   p2 = circle.point_at_angle(270*DEGREES)
502: (20)                   s1 = Square(side_length=0.25).move_to(p1)
503: (20)                   s2 = Square(side_length=0.25).move_to(p2)
504: (20)                   self.add(circle, s1, s2)
505: (8)
506: (8)
507: (8)
508: (16)
509: (20)
color=RED)
510: (20)
511: (24)
512: (24)
513: (24)
514: (20)
515: (20)
516: (8)
517: (8)
518: (12)
      """
      start_angle = angle_of_vector(self.points[0] - self.get_center())
      proportion = (angle - start_angle) / TAU
      proportion -= np.floor(proportion)
      return self.point_from_proportion(proportion)
    @staticmethod
    def from_three_points(p1: Point3D, p2: Point3D, p3: Point3D, **kwargs) ->
Self:
      """
      Returns a circle passing through the specified
      three points.
    Example
    -----
    .. manim:: CircleFromPointsExample
      :save_last_frame:
      class CircleFromPointsExample(Scene):
        def construct(self):
          circle = Circle.from_three_points(LEFT, LEFT + UP, UP * 2,
dots = VGroup(
          Dot(LEFT),
          Dot(LEFT + UP),
          Dot(UP * 2),
        )
        self.add(NumberPlane(), circle, dots)
      """
      center = line_intersection(
        perpendicular_bisector([p1, p2]),

```

```

519: (12)           perpendicular_bisector([p2, p3]),
520: (8)             )
521: (8)             radius = np.linalg.norm(p1 - center)
522: (8)             return Circle(radius=radius, **kwargs).shift(center)
523: (0) class Dot(Circle):
524: (4)     """A circle with a very small radius.
525: (4)     Parameters
526: (4)     -----
527: (4)     point
528: (8)         The location of the dot.
529: (4)     radius
530: (8)         The radius of the dot.
531: (4)     stroke_width
532: (8)         The thickness of the outline of the dot.
533: (4)     fill_opacity
534: (8)         The opacity of the dot's fill_colour
535: (4)     color
536: (8)         The color of the dot.
537: (4)     kwargs
538: (8)         Additional arguments to be passed to :class:`Circle`
539: (4) Examples
540: (4) -----
541: (4) .. manim:: DotExample
542: (8)   :save_last_frame:
543: (8)   class DotExample(Scene):
544: (12)     def construct(self):
545: (16)       dot1 = Dot(point=LEFT, radius=0.08)
546: (16)       dot2 = Dot(point=ORIGIN)
547: (16)       dot3 = Dot(point=RIGHT)
548: (16)       self.add(dot1,dot2,dot3)
549: (4)   """
550: (4)   def __init__(
551: (8)       self,
552: (8)       point: Point3D = ORIGIN,
553: (8)       radius: float = DEFAULT_DOT_RADIUS,
554: (8)       stroke_width: float = 0,
555: (8)       fill_opacity: float = 1.0,
556: (8)       color: ParsableManimColor = WHITE,
557: (8)       **kwargs,
558: (4)   ) -> None:
559: (8)       super().__init__(
560: (12)           arc_center=point,
561: (12)           radius=radius,
562: (12)           stroke_width=stroke_width,
563: (12)           fill_opacity=fill_opacity,
564: (12)           color=color,
565: (12)           **kwargs,
566: (8)   )
567: (0) class AnnotationDot(Dot):
568: (4)     """A dot with bigger radius and bold stroke to annotate scenes."""
569: (4)     def __init__(
570: (8)       self,
571: (8)       radius: float = DEFAULT_DOT_RADIUS * 1.3,
572: (8)       stroke_width: float = 5,
573: (8)       stroke_color: ParsableManimColor = WHITE,
574: (8)       fill_color: ParsableManimColor = BLUE,
575: (8)       **kwargs,
576: (4)   ) -> None:
577: (8)       super().__init__(
578: (12)           radius=radius,
579: (12)           stroke_width=stroke_width,
580: (12)           stroke_color=stroke_color,
581: (12)           fill_color=fill_color,
582: (12)           **kwargs,
583: (8)   )
584: (0) class LabeledDot(Dot):
585: (4)     """A :class:`Dot` containing a label in its center.
586: (4)     Parameters
587: (4)     -----

```

```

588: (4)             label
589: (8)             The label of the :class:`Dot`. This is rendered as :class:`~.MathTex`
590: (8)             by default (i.e., when passing a :class:`str`), but other classes
591: (8)             representing rendered strings like :class:`~.Text` or :class:`~.Tex`
592: (8)             can be passed as well.
593: (4)             radius
594: (8)             The radius of the :class:`Dot`. If ``None`` (the default), the radius
595: (8)             is calculated based on the size of the ``label``.
596: (4)             Examples
597: (4)             -----
598: (4)             .. manim:: SeveralLabeledDots
599: (8)             :save_last_frame:
600: (8)             class SeveralLabeledDots(Scene):
601: (12)             def construct(self):
602: (16)                 sq = Square(fill_color=RED, fill_opacity=1)
603: (16)                 self.add(sq)
604: (16)                 dot1 = LabeledDot(Tex("42", color=RED))
605: (16)                 dot2 = LabeledDot(MathTex("a", color=GREEN))
606: (16)                 dot3 = LabeledDot(Text("ii", color=BLUE))
607: (16)                 dot4 = LabeledDot("3")
608: (16)                 dot1.next_to(sq, UL)
609: (16)                 dot2.next_to(sq, UR)
610: (16)                 dot3.next_to(sq, DL)
611: (16)                 dot4.next_to(sq, DR)
612: (16)                 self.add(dot1, dot2, dot3, dot4)
613: (4)             """
614: (4)             def __init__(
615: (8)                 self,
616: (8)                 label: str | SingleStringMathTex | Text | Tex,
617: (8)                 radius: float | None = None,
618: (8)                 **kwargs,
619: (4)             ) -> None:
620: (8)                 if isinstance(label, str):
621: (12)                     from manim import MathTex
622: (12)                     rendered_label = MathTex(label, color=BLACK)
623: (8)                 else:
624: (12)                     rendered_label = label
625: (8)                 if radius is None:
626: (12)                     radius = 0.1 + max(rendered_label.width, rendered_label.height) /
2
627: (8)                 super().__init__(radius=radius, **kwargs)
628: (8)                 rendered_label.move_to(self.get_center())
629: (8)                 self.add(rendered_label)
630: (0)             class Ellipse(Circle):
631: (4)                 """A circular shape; oval, circle.
632: (4)                 Parameters
633: (4)                 -----
634: (4)                 width
635: (7)                 The horizontal width of the ellipse.
636: (4)                 height
637: (7)                 The vertical height of the ellipse.
638: (4)                 kwargs
639: (7)                 Additional arguments to be passed to :class:`Circle`.
640: (4)             Examples
641: (4)             -----
642: (4)             .. manim:: EllipseExample
643: (8)             :save_last_frame:
644: (8)             class EllipseExample(Scene):
645: (12)             def construct(self):
646: (16)                 ellipse_1 = Ellipse(width=2.0, height=4.0, color=BLUE_B)
647: (16)                 ellipse_2 = Ellipse(width=4.0, height=1.0, color=BLUE_D)
648: (16)                 ellipse_group = Group(ellipse_1, ellipse_2).arrange(buff=1)
649: (16)                 self.add(ellipse_group)
650: (4)             """
651: (4)             def __init__(self, width: float = 2, height: float = 1, **kwargs) -> None:
652: (8)                 super().__init__(**kwargs)
653: (8)                 self.stretch_to_fit_width(width)
654: (8)                 self.stretch_to_fit_height(height)
655: (0)             class AnnularSector(Arc):

```

```

656: (4)             """A sector of an annulus.
657: (4)             Parameters
658: (4)             -----
659: (4)             inner_radius
660: (7)                 The inside radius of the Annular Sector.
661: (4)             outer_radius
662: (7)                 The outside radius of the Annular Sector.
663: (4)             angle
664: (7)                 The clockwise angle of the Annular Sector.
665: (4)             start_angle
666: (7)                 The starting clockwise angle of the Annular Sector.
667: (4)             fill_opacity
668: (7)                 The opacity of the color filled in the Annular Sector.
669: (4)             stroke_width
670: (7)                 The stroke width of the Annular Sector.
671: (4)             color
672: (7)                 The color filled into the Annular Sector.
673: (4)             Examples
674: (4)             -----
675: (4)             .. manim:: AnnularSectorExample
676: (8)             :save_last_frame:
677: (8)             class AnnularSectorExample(Scene):
678: (12)             def construct(self):
679: (16)                     self.camera.background_color = WHITE
680: (16)                     s1 = AnnularSector(color=YELLOW).move_to(2 * UL)
681: (16)                     s2 = AnnularSector(inner_radius=1.5, outer_radius=2, angle=45
* DEGREES, color=RED).move_to(2 * UR)
682: (16)                     s3 = AnnularSector(inner_radius=1, outer_radius=1.5, angle=PI,
fill_opacity=0.25, color=BLUE).move_to(2 * DL)
683: (16)                     s4 = AnnularSector(inner_radius=1, outer_radius=1.5, angle=-3
* PI / 2, color=GREEN).move_to(2 * DR)
684: (16)                     self.add(s1, s2, s3, s4)
685: (4)             """
686: (4)             def __init__(
687: (8)                     self,
688: (8)                     inner_radius: float = 1,
689: (8)                     outer_radius: float = 2,
690: (8)                     angle: float = TAU / 4,
691: (8)                     start_angle: float = 0,
692: (8)                     fill_opacity: float = 1,
693: (8)                     stroke_width: float = 0,
694: (8)                     color: ParsableManimColor = WHITE,
695: (8)                     **kwargs,
696: (4)             ) -> None:
697: (8)                     self.inner_radius = inner_radius
698: (8)                     self.outer_radius = outer_radius
699: (8)                     super().__init__(
700: (12)                         start_angle=start_angle,
701: (12)                         angle=angle,
702: (12)                         fill_opacity=fill_opacity,
703: (12)                         stroke_width=stroke_width,
704: (12)                         color=color,
705: (12)                         **kwargs,
706: (8)             )
707: (4)             def generate_points(self) -> None:
708: (8)                     inner_arc, outer_arc = (
709: (12)                         Arc(
710: (16)                             start_angle=self.start_angle,
711: (16)                             angle=self.angle,
712: (16)                             radius=radius,
713: (16)                             arc_center=self.arc_center,
714: (12)                         )
715: (12)                         for radius in (self.inner_radius, self.outer_radius)
716: (8)                     )
717: (8)                     outer_arc.reverse_points()
718: (8)                     self.append_points(inner_arc.points)
719: (8)                     self.add_line_to(outer_arc.points[0])
720: (8)                     self.append_points(outer_arc.points)
721: (8)                     self.add_line_to(inner_arc.points[0])

```

```

722: (4)             init_points = generate_points
723: (0)             class Sector(AnnularSector):
724: (4)                 """A sector of a circle.
725: (4)                 Examples
726: (4)                 -----
727: (4)                 .. manim:: ExampleSector
728: (8)                     :save_last_frame:
729: (8)                     class ExampleSector(Scene):
730: (12)                         def construct(self):
731: (16)                             sector = Sector(outer_radius=2, inner_radius=1)
732: (16)                             sector2 = Sector(outer_radius=2.5,
inner_radius=0.8).move_to([-3, 0, 0])
733: (16)                             sector.set_color(RED)
734: (16)                             sector2.set_color(PINK)
735: (16)                             self.add(sector, sector2)
736: (4)                         """
737: (4)                 def __init__(
738: (8)                     self, outer_radius: float = 1, inner_radius: float = 0, **kwargs
739: (4)                 ) -> None:
740: (8)                     super().__init__(inner_radius=inner_radius, outer_radius=outer_radius,
**kwargs)
741: (0)             class Annulus(Circle):
742: (4)                 """Region between two concentric :class:`Circles <.Circle>`.
743: (4)                 Parameters
744: (4)                 -----
745: (4)                 inner_radius
746: (8)                     The radius of the inner :class:`Circle`.
747: (4)                 outer_radius
748: (8)                     The radius of the outer :class:`Circle`.
749: (4)                 kwargs
750: (8)                     Additional arguments to be passed to :class:`Annulus`
751: (4)                 Examples
752: (4)                 -----
753: (4)                 .. manim:: AnnulusExample
754: (8)                     :save_last_frame:
755: (8)                     class AnnulusExample(Scene):
756: (12)                         def construct(self):
757: (16)                             annulus_1 = Annulus(inner_radius=0.5,
outer_radius=1).shift(UP)
758: (16)                             annulus_2 = Annulus(inner_radius=0.3, outer_radius=0.6,
color=RED).next_to(annulus_1, DOWN)
759: (16)                             self.add(annulus_1, annulus_2)
760: (4)                         """
761: (4)                 def __init__(
762: (8)                     self,
763: (8)                     inner_radius: float | None = 1,
764: (8)                     outer_radius: float | None = 2,
765: (8)                     fill_opacity: float = 1,
766: (8)                     stroke_width: float = 0,
767: (8)                     color: ParsableManimColor = WHITE,
768: (8)                     mark_paths_closed: bool = False,
769: (8)                     **kwargs,
770: (4)                 ) -> None:
771: (8)                     self.mark_paths_closed = mark_paths_closed # is this even used?
772: (8)                     self.inner_radius = inner_radius
773: (8)                     self.outer_radius = outer_radius
774: (8)                     super().__init__(
775: (12)                         fill_opacity=fill_opacity, stroke_width=stroke_width, color=color,
**kwargs
776: (8)                     )
777: (4)                 def generate_points(self) -> None:
778: (8)                     self.radius = self.outer_radius
779: (8)                     outer_circle = Circle(radius=self.outer_radius)
780: (8)                     inner_circle = Circle(radius=self.inner_radius)
781: (8)                     inner_circle.reverse_points()
782: (8)                     self.append_points(outer_circle.points)
783: (8)                     self.append_points(inner_circle.points)
784: (8)                     self.shift(self.arc_center)
785: (4)                     init_points = generate_points

```

```

786: (0)         class CubicBezier(VMobject, metaclass=ConvertToOpenGL):
787: (4)             """A cubic Bézier curve.
788: (4)             Example
789: (4)             -----
790: (4)             .. manim:: BezierSplineExample
791: (8)                 :save_last_frame:
792: (8)                 class BezierSplineExample(Scene):
793: (12)                     def construct(self):
794: (16)                         p1 = np.array([-3, 1, 0])
795: (16)                         p1b = p1 + [1, 0, 0]
796: (16)                         d1 = Dot(point=p1).set_color(BLUE)
797: (16)                         l1 = Line(p1, p1b)
798: (16)                         p2 = np.array([3, -1, 0])
799: (16)                         p2b = p2 - [1, 0, 0]
800: (16)                         d2 = Dot(point=p2).set_color(RED)
801: (16)                         l2 = Line(p2, p2b)
802: (16)                         bezier = CubicBezier(p1b, p1b + 3 * RIGHT, p2b - 3 * RIGHT,
p2b)
803: (16)                         self.add(l1, d1, l2, d2, bezier)
804: (4)             """
805: (4)             def __init__(
806: (8)                 self,
807: (8)                 start_anchor: CubicBezierPoints,
808: (8)                 start_handle: CubicBezierPoints,
809: (8)                 end_handle: CubicBezierPoints,
810: (8)                 end_anchor: CubicBezierPoints,
811: (8)                 **kwargs,
812: (4)             ) -> None:
813: (8)                 super().__init__(**kwargs)
814: (8)                 self.add_cubic_bezier_curve(start_anchor, start_handle, end_handle,
end_anchor)
815: (0)             class ArcPolygon(VMobject, metaclass=ConvertToOpenGL):
816: (4)                 """A generalized polygon allowing for points to be connected with arcs.
817: (4)                 This version tries to stick close to the way :class:`Polygon` is used.
Points
818: (4)                 can be passed to it directly which are used to generate the according arcs
819: (4)                 (using :class:`ArcBetweenPoints`). An angle or radius can be passed to it
to
820: (4)                 use across all arcs, but to configure arcs individually an ``arc_config``
list
821: (4)                 has to be passed with the syntax explained below.
Parameters
822: (4)
823: (4)
824: (4)
vertices
825: (8)                 A list of vertices, start and end points for the arc segments.
angle
826: (4)
827: (8)                 The angle used for constructing the arcs. If no other parameters
828: (8)                 are set, this angle is used to construct all arcs.
radius
829: (4)
830: (8)                 The circle radius used to construct the arcs. If specified,
831: (8)                 overrides the specified ``angle``.
arc_config
832: (4)
833: (8)                 When passing a ``dict``, its content will be passed as keyword
834: (8)                 arguments to :class:`~.ArcBetweenPoints`. Otherwise, a list
835: (8)                 of dictionaries containing values that are passed as keyword
836: (8)                 arguments for every individual arc can be passed.
kwargs
837: (4)
838: (8)                 Further keyword arguments that are passed to the constructor of
839: (8)                 :class:`~.VMobject`.
Attributes
840: (4)
841: (4)
arcs : :class:`list`
842: (4)
843: (8)                 The arcs created from the input parameters::
844: (12)                     >>> from manim import ArcPolygon
845: (12)                     >>> ap = ArcPolygon([0, 0, 0], [2, 0, 0], [0, 2, 0])
846: (12)                     >>> ap.arcs
847: (12)                     [ArcBetweenPoints, ArcBetweenPoints, ArcBetweenPoints]
.. tip::
848: (4)                     Two instances of :class:`ArcPolygon` can be transformed properly into
849: (8)

```

```

one
850: (8)           another as well. Be advised that any arc initialized with ``angle=0`` will actually be a straight line, so if a straight section should
851: (8)           transform into an arced section or vice versa, initialize the straight
seamlessly
852: (8)           with a negligible angle instead (such as ``angle=0.0001``).
853: (8)
854: (4)           .. note::
855: (8)           There is an alternative version (:class:`ArcPolygonFromArcs`) that is
instantiated
856: (8)           with pre-defined arcs.
857: (4)           See Also
858: (4)           -----
859: (4)           :class:`ArcPolygonFromArcs`  

860: (4)           Examples
861: (4)           -----
862: (4)           .. manim:: SeveralArcPolygons
863: (8)           class SeveralArcPolygons(Scene):
864: (12)           def construct(self):
865: (16)               a = [0, 0, 0]
866: (16)               b = [2, 0, 0]
867: (16)               c = [0, 2, 0]
868: (16)               ap1 = ArcPolygon(a, b, c, radius=2)
869: (16)               ap2 = ArcPolygon(a, b, c, angle=45*DEGREES)
870: (16)               ap3 = ArcPolygon(a, b, c, arc_config={'radius': 1.7, 'color': RED})
871: (16)               ap4 = ArcPolygon(a, b, c, color=RED, fill_opacity=1,
872: (44)                           arc_config=[{'radius': 1.7,
873: (44)                               {'angle': 20*DEGREES, 'color':
874: (44)                                   {'radius': 1}])
875: (16)                           ap_group = VGroup(ap1, ap2, ap3, ap4).arrange()
876: (16)                           self.play(*[Create(ap) for ap in [ap1, ap2, ap3, ap4]])
877: (16)                           self.wait()
878: (4)           For further examples see :class:`ArcPolygonFromArcs`.
879: (4)           """
880: (4)           def __init__(
881: (8)               self,
882: (8)               *vertices: Point3D,
883: (8)               angle: float = PI / 4,
884: (8)               radius: float | None = None,
885: (8)               arc_config: list[dict] | None = None,
886: (8)               **kwargs,
887: (4)           ) -> None:
888: (8)               n = len(vertices)
889: (8)               point_pairs = [(vertices[k], vertices[(k + 1) % n]) for k in range(n)]
890: (8)               if not arc_config:
891: (12)                   if radius:
892: (16)                       all_arc_configs = itertools.repeat({"radius": radius},
len(point_pairs))
893: (12)
894: (16)               else:
895: (12)                   all_arc_configs = itertools.repeat({"angle": angle},
len(point_pairs))
896: (12)
897: (8)
898: (12)
899: (12)
900: (8)
901: (12)
902: (12)
903: (8)
904: (8)
905: (8)
906: (8)
907: (12)
908: (8)
909: (0)           class ArcPolygonFromArcs(VMobject, metaclass=ConvertToOpenGL):

```

```

910: (4)         """A generalized polygon allowing for points to be connected with arcs.
911: (4)         This version takes in pre-defined arcs to generate the ArcPolygon and
912: (4)         introduces little new syntax. However unlike :class:`Polygon` it can't be created
913: (4)         with points directly.
914: (4)         For proper appearance the passed arcs should connect seamlessly:
915: (4)         ``[a,b][b,c][c,a]``
916: (4)         If there are any gaps between the arcs, those will be filled in
917: (4)         with straight lines, which can be used deliberately for any straight
918: (4)         sections. Arcs can also be passed as straight lines such as an arc
919: (4)         initialized with ``angle=0``.
920: (4)         Parameters
921: (4)         -----
922: (4)         arcs
923: (8)           These are the arcs from which the ArcPolygon is assembled.
924: (4)         kwargs
925: (8)           Keyword arguments that are passed to the constructor of
926: (8)           :class:`~.VMobject`. Affects how the ArcPolygon itself is drawn,
927: (8)           but doesn't affect passed arcs.
928: (4)         Attributes
929: (4)         -----
930: (4)         arcs
931: (8)           The arcs used to initialize the ArcPolygonFromArcs:::
932: (12)           >>> from manim import ArcPolygonFromArcs, Arc, ArcBetweenPoints
933: (12)           >>> ap = ArcPolygonFromArcs(Arc(), ArcBetweenPoints([1,0,0],
934: (12)             [0,1,0]), Arc())
935: (12)
936: (4)           .. tip::
937: (8)             Two instances of :class:`ArcPolygon` can be transformed properly into
938: (8)             one another as well. Be advised that any arc initialized with
939: (8)             ``angle=0`` will actually be a straight line, so if a straight section should
940: (8)             transform into an arced section or vice versa, initialize the straight
941: (8)             section with a negligible angle instead (such as ``angle=0.0001``).
942: (4)           .. note::
943: (8)             There is an alternative version (:class:`ArcPolygon`) that can be
944: (8)             instantiated
945: (4)             with points.
946: (8)             .. seealso::
947: (4)               :class:`ArcPolygon`
948: (4)             Examples
949: (4)             -----
950: (4)               One example of an arcpolygon is the Reuleaux triangle.
951: (4)               Instead of 3 straight lines connecting the outer points,
952: (4)               a Reuleaux triangle has 3 arcs connecting those points,
953: (4)               making a shape with constant width.
954: (4)               Passed arcs are stored as subobjects in the arcpolygon.
955: (4)               This means that the arcs are changed along with the arcpolygon,
956: (4)               for example when it's shifted, and these arcs can be manipulated
957: (4)               after the arcpolygon has been initialized.
958: (4)               Also both the arcs contained in an :class:`~.ArcPolygonFromArcs`, as well
959: (4)               as the arcpolygon itself are drawn, which affects draw time in :class:`~.Create`
960: (4)               for example. In most cases the arcs themselves don't
961: (4)               need to be drawn, in which case they can be passed as invisible.
962: (8)               .. manim:: ArcPolygonExample
963: (12)                 class ArcPolygonExample(Scene):
964: (16)                   def construct(self):
965: (16)                     arc_conf = {"stroke_width": 0}
966: (22)                     poly_conf = {"stroke_width": 10, "stroke_color": BLUE,
967: (16)                         "fill_opacity": 1, "color": PURPLE}
968: (16)                     a = [-1, 0, 0]
969: (16)                     b = [1, 0, 0]
970: (16)                     c = [0, np.sqrt(3), 0]
971: (16)                     arc0 = ArcBetweenPoints(a, b, radius=2, **arc_conf)
971: (16)                     arc1 = ArcBetweenPoints(b, c, radius=2, **arc_conf)

```

12/20/24, 4:24 AM

manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

972: (16)                               arc2 = ArcBetweenPoints(c, a, radius=2, **arc_conf)
973: (16)                               reuleaux_tri = ArcPolygonFromArcs(arc0, arc1, arc2,
**poly_conf)
974: (16)                               self.play(FadeIn(reuleaux_tri))
975: (16)                               self.wait(2)
976: (4)        The arcpolygon itself can also be hidden so that instead only the
contained
977: (4)        arcs are drawn. This can be used to easily debug arcs or to highlight
them.
978: (4)        .. manim:: ArcPolygonExample2
979: (8)            class ArcPolygonExample2(Scene):
980: (12)                def construct(self):
981: (16)                    arc_conf = {"stroke_width": 3, "stroke_color": BLUE,
982: (20)                        "fill_opacity": 0.5, "color": GREEN}
983: (16)                    poly_conf = {"color": None}
984: (16)                    a = [-1, 0, 0]
985: (16)                    b = [1, 0, 0]
986: (16)                    c = [0, np.sqrt(3), 0]
987: (16)                    arc0 = ArcBetweenPoints(a, b, radius=2, **arc_conf)
988: (16)                    arc1 = ArcBetweenPoints(b, c, radius=2, **arc_conf)
989: (16)                    arc2 = ArcBetweenPoints(c, a, radius=2, stroke_color=RED)
990: (16)                    reuleaux_tri = ArcPolygonFromArcs(arc0, arc1, arc2,
**poly_conf)
991: (16)                               self.play(FadeIn(reuleaux_tri))
992: (16)                               self.wait(2)
993: (4)        """
994: (4)        def __init__(self, *arcs: Arc | ArcBetweenPoints, **kwargs) -> None:
995: (8)            if not all(isinstance(m, (Arc, ArcBetweenPoints)) for m in arcs):
996: (12)                raise ValueError(
997: (16)                    "All ArcPolygon submobjects must be of type
Arc/ArcBetweenPoints",
998: (12)                )
999: (8)            super().__init__(**kwargs)
1000: (8)            self.add(*arcs)
1001: (8)            self.arcs = [*arcs]
1002: (8)            from .line import Line
1003: (8)            for arc1, arc2 in adjacent_pairs(arcs):
1004: (12)                self.append_points(arc1.points)
1005: (12)                line = Line(arc1.get_end(), arc2.get_start())
1006: (12)                len_ratio = line.get_length() / arc1.get_arc_length()
1007: (12)                if np.isnan(len_ratio) or np.isinf(len_ratio):
1008: (16)                    continue
1009: (12)                line.insert_n_curves(int(arc1.get_num_curves() * len_ratio))
1010: (12)                self.append_points(line.points)

```

---

File 49 - logo.py:

```

1: (0)        """Utilities for Manim's logo and banner."""
2: (0)        from __future__ import annotations
3: (0)        __all__ = ["ManimBanner"]
4: (0)        import svgelements as se
5: (0)        from manim.animation.updaters.update import UpdateFromAlphaFunc
6: (0)        from manim.mobject.geometry.arc import Circle
7: (0)        from manim.mobject.geometry.polygram import Square, Triangle
8: (0)        from .. import constants as cst
9: (0)        from ..animation.animation import override_animation
10: (0)       from ..animation.composition import AnimationGroup, Succession
11: (0)       from ..animation.creation import Create, SpiralIn
12: (0)       from ..animation.fading import FadeIn
13: (0)       from ..mobject.svg.svg_mobject import VMobjectFromSVGPath
14: (0)       from ..mobject.types.vectorized_mobject import VGroup
15: (0)       from ..utils.rate_functions import ease_in_out_cubic, smooth
16: (0)       MANIM_SVG_PATHS: list[se.Path] = [
17: (4)           se.Path( # double stroke letter M
18: (8)               "M4.64259-2.092154L2.739726-6.625156C2.660025-6.824408 2.650062-
6.824408 "
19: (8)               "2.381071-6.824408H.52802C.348692-6.824408 .199253-6.824408 .199253-

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

6.645"  
 20: (8) "081C.199253-6.475716 .37858-6.475716 .428394-6.475716C.547945-  
 6.475716 ."  
 21: (8) "816936-6.455791 1.036115-6.37609V-1.05604C1.036115-.846824  
 1.036115-.408"  
 22: (8) "468 .358655-.348692C.169365-.328767 .169365-.18929  
 .169365-.179328C.1693"  
 23: (8) "65 0 .328767 0 .508095 0H2.052304C2.231631 0 2.381071 0  
 2.381071-.179328"  
 24: (8) "C2.381071-.268991 2.30137-.33873 2.221669-.348692C1.454545-.408468  
 1.454"  
 25: (8) "545-.826899 1.454545-1.05604V-6.017435L1.464508-  
 6.027397L3.895392-.20921"  
 26: (8) "5C3.975093-.029888 4.044832 0 4.104608 0C4.224159 0 4.254047-.079701  
 4.3"  
 27: (8) "03861-.199253L6.744707-6.027397L6.75467-6.017435V-  
 1.05604C6.75467-.84682"  
 28: (8) "4 6.75467-.408468 6.07721-.348692C5.88792-.328767 5.88792-.18929  
 5.88792"  
 29: (8) "-.179328C5.88792 0 6.047323 0 6.22665 0H8.886675C9.066002 0 9.215442  
 0 9"  
 30: (8) ".215442-.179328C9.215442-.268991 9.135741-.33873  
 9.05604-.348692C8.28891"  
 31: (8) "7-.408468 8.288917-.826899 8.288917-1.05604V-5.768369C8.288917-  
 5.977584 "  
 32: (8) "8.288917-6.41594 8.966376-6.475716C9.066002-6.485679 9.155666-  
 6.535492 9"  
 33: (8) ".155666-6.645081C9.155666-6.824408 9.006227-6.824408 8.826899-  
 6.824408H6"  
 34: (8) ".90411C6.645081-6.824408 6.625156-6.824408 6.535492-6.615193L4.64259-  
 2.0"  
 35: (8) "92154ZM4.343711-1.912827C4.423412-1.743462 4.433375-1.733499  
 4.552927-1."  
 36: (8) "693649L4.11457-.637609H4.094645L1.823163-6.057285C1.77335-6.1868  
 1.69364"  
 37: (8) "9-6.356164 1.554172-6.475716H2.420922L4.343711-  
 1.912827ZM1.334994-.34869"  
 38: (8) "2H1.165629C1.185554-.37858 1.205479-.408468  
 1.225405-.428394C1.235367-.4"  
 39: (8) "38356 1.235367-.448319 1.24533-.458281L1.334994-.348692ZM7.103362-  
 6.4757"  
 40: (8) "16H8.159402C7.940224-6.22665 7.940224-5.967621 7.940224-5.788294V-  
 1.0361"  
 41: (8) "15C7.940224-.856787 7.940224-.597758  
 8.169365-.348692H6.884184C7.103362-"  
 42: (8) ".597758 7.103362-.856787 7.103362-1.036115V-6.475716Z"  
 43: (4) ),  
 44: (4) se.Path( # letter a  
 45: (8) "M1.464508-4.024907C1.464508-4.234122 1.743462-4.393524 2.092154-  
 4.393524"  
 46: (8) "C2.669988-4.393524 2.929016-4.124533 2.929016-3.516812V-  
 2.789539C1.77335"  
 47: (8) "-2.440847 .249066-2.042341 .249066-.916563C.249066-.308842 .71731  
 .13947"  
 48: (8) "7 1.354919 .139477C1.92279 .139477 2.381071-.059776  
 2.929016-.557908C3.0"  
 49: (8) "38605-.049813 3.257783 .139477 3.745953 .139477C4.174346 .139477  
 4.48318"  
 50: (8) "8-.019925  
 4.861768-.428394L4.712329-.637609L4.612702-.537983C4.582814-.5"  
 51: (8) "08095 4.552927-.498132 4.503113-.498132C4.363636-.498132  
 4.293898-.58779"  
 52: (8) "6 4.293898-.747198V-3.347447C4.293898-4.184309 3.536737-4.712329  
 2.32129"  
 53: (8) "5-4.712329C1.195517-4.712329 .438356-4.204234 .438356-  
 3.457036C.438356-3"  
 54: (8) ".048568 .67746-2.799502 1.085928-2.799502C1.484433-2.799502 1.763387-  
 3.0"  
 55: (8) "38605 1.763387-3.377335C1.763387-3.676214 1.464508-3.88543 1.464508-

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

4.02"
56: (8) "4907ZM2.919054-.996264C2.650062-.687422 2.450809-.56787
2.211706-.56787C"
57: (8) "1.912827-.56787 1.703611-.836862 1.703611-1.235367C1.703611-1.8132
2.122"
58: (8) "042-2.231631 2.919054-2.440847V-.996264Z"
59: (4),
60: (4) se.Path( # letter n
61: (8) "M2.948941-4.044832C3.297634-4.044832 3.466999-3.775841 3.466999-
3.217933"
62: (8) "V-.806974C3.466999-.438356 3.337484-.278954
2.998755-.239103V0H5.339975V"
63: (8) "-.239103C4.951432-.268991 4.851806-.388543 4.851806-.806974V-
3.307597C4."
64: (8) "851806-4.164384 4.323786-4.712329 3.506849-4.712329C2.909091-4.712329
2."
65: (8) "450809-4.433375 2.082192-3.845579V-4.592777H.179328V-
4.353674C.617684-4."
66: (8) "283935 .707347-4.184309 .707347-3.765878V-.836862C.707347-.418431
.62764"
67: (8) "6-.328767 .179328-.239103V0H2.580324V-.239103C2.211706-.288917
2.092154-"
68: (8) ".438356 2.092154-.806974V-3.466999C2.092154-3.576588 2.530511-
4.044832 2"
69: (8) ".948941-4.044832Z"
70: (4),
71: (4) se.Path( # letter i
72: (8) "M2.15193-4.592777H.239103V-4.353674C.67746-4.26401 .767123-4.174346
.767"
73: (8) "123-3.765878V-.836862C.767123-.428394 .697385-.348692
.239103-.239103V0H"
74: (8) "2.6401V-.239103C2.291407-.288917 2.15193-.428394 2.15193-.806974V-
4.5927"
75: (8) "77ZM1.454545-6.884184C1.026152-6.884184 .67746-6.535492 .67746-
6.117061C"
76: (8) ".67746-5.668742 1.006227-5.339975 1.444583-5.339975S2.221669-5.668742
2."
77: (8) "221669-6.107098C2.221669-6.535492 1.882939-6.884184 1.454545-
6.884184Z"
78: (4),
79: (4) se.Path( # letter m
80: (8) "M2.929016-4.044832C3.317559-4.044832 3.466999-3.815691 3.466999-
3.217933"
81: (8) "V-.806974C3.466999-.398506 3.35741-.268991
2.988792-.239103V0H5.32005V-."
82: (8) "239103C4.971357-.278954 4.851806-.428394 4.851806-.806974V-
3.466999C4.85"
83: (8) "1806-3.576588 5.310087-4.044832 5.69863-4.044832C6.07721-4.044832
6.2266"
84: (8) "5-3.805729 6.22665-3.217933V-.806974C6.22665-.388543 6.117061-.268991
5."
85: (8) "738481-.239103V0H8.109589V-.239103C7.721046-.259029 7.611457-.37858
7.61"
86: (8) "1457-.806974V-3.307597C7.611457-4.164384 7.083437-4.712329 6.266501-
4.71"
87: (8) "2329C5.69863-4.712329 5.32005-4.483188 4.801993-3.845579C4.503113-
4.4732"
88: (8) "25 4.154421-4.712329 3.526775-4.712329S2.440847-4.443337 2.062267-
3.8455"
89: (8) "79V-4.592777H.179328V-4.353674C.617684-4.293898 .707347-4.174346
.707347"
90: (8) "-3.765878V-.836862C.707347-.428394 .617684-.318804
.179328-.239103V0H2.5"
91: (8) "50436V-.239103C2.201743-.288917 2.092154-.428394 2.092154-.806974V-
3.466"
92: (8) "999C2.092154-3.58655 2.530511-4.044832 2.929016-4.044832Z"
93: (4),
94: (0) ],
95: (0) class ManimBanner(VGroup):

```

```

96: (4)             """Convenience class representing Manim's banner.
97: (4)             Can be animated using custom methods.
98: (4)             Parameters
99: (4)             -----
100: (4)             dark_theme
101: (8)                 If ``True`` (the default), the dark theme version of the logo
102: (8)                 (with light text font) will be rendered. Otherwise, if ``False``,
103: (8)                 the light theme version (with dark text font) is used.
104: (4)             Examples
105: (4)             -----
106: (4)             .. manim:: DarkThemeBanner
107: (8)                 class DarkThemeBanner(Scene):
108: (12)                     def construct(self):
109: (16)                         banner = ManimBanner()
110: (16)                         self.play(banner.create())
111: (16)                         self.play(banner.expand())
112: (16)                         self.wait()
113: (16)                         self.play(Unwrite(banner))
114: (4)             .. manim:: LightThemeBanner
115: (8)                 class LightThemeBanner(Scene):
116: (12)                     def construct(self):
117: (16)                         self.camera.background_color = "#ece6e2"
118: (16)                         banner = ManimBanner(dark_theme=False)
119: (16)                         self.play(banner.create())
120: (16)                         self.play(banner.expand())
121: (16)                         self.wait()
122: (16)                         self.play(Unwrite(banner))
123: (4)             """
124: (4)             def __init__(self, dark_theme: bool = True):
125: (8)                 super().__init__()
126: (8)                 logo_green = "#81b29a"
127: (8)                 logo_blue = "#454866"
128: (8)                 logo_red = "#e07a5f"
129: (8)                 m_height_over_anim_height = 0.75748
130: (8)                 self.font_color = "#ece6e2" if dark_theme else "#343434"
131: (8)                 self.scale_factor = 1
132: (8)                 self.M =
VMObjectFromSVGPath(MANIM_SVG_PATHS[0]).flip(cst.RIGHT).center()
133: (8)                 self.M.set(stroke_width=0).scale(
134: (12)                     7 * cst.DEFAULT_FONT_SIZE * cst.SCALE_FACTOR_PER_FONT_POINT
135: (8)
136: (8)                     )
137: (12)                     self.M.set_fill(color=self.font_color, opacity=1).shift(
138: (8)                         2.25 * cst.LEFT + 1.5 * cst.UP
139: (8)
140: (8)                     )
141: (8)
fill_opacity=1).shift(cst.RIGHT)
142: (8)                     self.shapes = VGroup(self.triangle, self.square, self.circle)
143: (8)                     self.add(self.shapes, self.M)
144: (8)                     self.move_to(cst.ORIGIN)
145: (8)                     anim = VGroup()
146: (8)                     for ind, path in enumerate(MANIM_SVG_PATHS[1:]):
147: (12)                         tex = VMObjectFromSVGPath(path).flip(cst.RIGHT).center()
148: (12)                         tex.set(stroke_width=0).scale(
149: (16)                             cst.DEFAULT_FONT_SIZE * cst.SCALE_FACTOR_PER_FONT_POINT
150: (12)
151: (12)                             )
152: (16)                             if ind > 0:
153: (12)                                 tex.next_to(anim, buff=0.01)
154: (12)                                 tex.align_to(self.M, cst.DOWN)
155: (8)                                 anim.add(tex)
156: (8)                                 anim.set_fill(color=self.font_color, opacity=1)
157: (8)                                 anim.height = m_height_over_anim_height * self.M.height
158: (4)                                 self.anim = anim
def scale(self, scale_factor: float, **kwargs) -> ManimBanner:
159: (8)             """Scale the banner by the specified scale factor.
160: (8)             Parameters
161: (8)             -----
162: (8)             scale_factor

```

```

163: (12)                      The factor used for scaling the banner.
164: (8)                       Returns
165: (8)
166: (8)
167: (12)                      :class:`~.ManimBanner`
168: (8)                       The scaled banner.
169: (8)                       """
170: (8)                       self.scale_factor *= scale_factor
171: (12)                      if self.anim not in self.submobjects:
172: (8)                          self.anim.scale(scale_factor, **kwargs)
173: (4)                           return super().scale(scale_factor, **kwargs)
174: (4) @override_animation(Create)
175: (8) def create(self, run_time: float = 2) -> AnimationGroup:
176: (8)     """The creation animation for Manim's logo.
177: (8)     Parameters
178: (8)     -----
179: (12)     run_time
180: (8)         The run time of the animation.
181: (8)     Returns
182: (8)
183: (12)     :class:`~.AnimationGroup`
184: (8)         An animation to be used in a :meth:`.Scene.play` call.
185: (8)
186: (12)     return AnimationGroup(
187: (12)         SpiralIn(self.shapes, run_time=run_time),
188: (12)         FadeIn(self.M, run_time=run_time / 2),
189: (8)             lag_ratio=0.1,
190: (4)     )
191: (8) def expand(self, run_time: float = 1.5, direction="center") -> Succession:
192: (8)     """An animation that expands Manim's logo into its banner.
193: (8)     The returned animation transforms the banner from its initial
194: (8)     state (representing Manim's logo with just the icons) to its
195: (8)     expanded state (showing the full name together with the icons).
196: (8)     See the class documentation for how to use this.
197: (12)     .. note::
198: (12)         Before calling this method, the text "anim" is not a
199: (12)         submobject of the banner object. After the expansion,
200: (12)         it is added as a submobject so subsequent animations
201: (8)             to the banner object apply to the text "anim" as well.
202: (8)     Parameters
203: (8)
204: (12)     run_time
205: (8)         The run time of the animation.
206: (12)     direction
207: (8)         The direction in which the logo is expanded.
208: (8)     Returns
209: (8)
210: (12)     :class:`~.Succession`
211: (8)         An animation to be used in a :meth:`.Scene.play` call.
212: (8) Examples
213: (8)
214: (12)     .. manim:: ExpandDirections
215: (16)         class ExpandDirections(Scene):
216: (20)             def construct(self):
217: (20)                 banners = [ManimBanner().scale(0.5).shift(UP*x) for x in
218: (-2, 0, 2)]
219: (20)
220: (24)                 self.play(
221: (24)                     banners[0].expand(direction="right"),
222: (24)                     banners[1].expand(direction="center"),
223: (24)                     banners[2].expand(direction="left"),
224: (20)                 )
225: (8)                 """
226: (8)                 if direction not in ["left", "right", "center"]:
227: (8)                     raise ValueError("direction must be 'left', 'right' or 'center'.")
228: (8)                 m_shape_offset = 6.25 * self.scale_factor
229: (8)                 shape_sliding_overshoot = self.scale_factor * 0.8
230: (8)                 m_anim_buff = 0.06
231: (8)                 self.anim.next_to(self.M, buff=m_anim_buff).align_to(self.M, cst.DOWN)
232: (8)                 self.anim.set_opacity(0)
233: (8)                 self.shapes.save_state()

```

```

231: (8)             m_clone = self.anim[-1].copy()
232: (8)             self.add(m_clone)
233: (8)             m_clone.move_to(self.shapes)
234: (8)             self.M.save_state()
235: (8)             left_group = VGroup(self.M, self.anim, m_clone)
236: (8)             def shift(vector):
237: (12)                 self.shapes.restore()
238: (12)                 left_group.align_to(self.M.saved_state, cst.LEFT)
239: (12)                 if direction == "right":
240: (16)                     self.shapes.shift(vector)
241: (12)                 elif direction == "center":
242: (16)                     self.shapes.shift(vector / 2)
243: (16)                     left_group.shift(-vector / 2)
244: (12)                 elif direction == "left":
245: (16)                     left_group.shift(-vector)
246: (8)             def slide_and_uncover(mob, alpha):
247: (12)                 shift(alpha * (m_shape_offset + shape_sliding_overshoot) *
248: (12)                     for letter in mob.anim:
249: (16)                         if mob.square.get_center()[0] > letter.get_center()[0]:
250: (20)                             letter.set_opacity(1)
251: (20)                             self.add_to_back(letter)
252: (12)                 if alpha == 1:
253: (16)                     self.remove(*[self.anim])
254: (16)                     self.add_to_back(self.anim)
255: (16)                     mob.shapes.set_z_index(0)
256: (16)                     mob.shapes.save_state()
257: (16)                     mob.M.save_state()
258: (8)             def slide_back(mob, alpha):
259: (12)                 if alpha == 0:
260: (16)                     m_clone.set_opacity(1)
261: (16)                     m_clone.move_to(mob.anim[-1])
262: (16)                     mob.anim.set_opacity(1)
263: (12)                     shift(alpha * shape_sliding_overshoot * cst.LEFT)
264: (12)                 if alpha == 1:
265: (16)                     mob.remove(m_clone)
266: (16)                     mob.add_to_back(mob.shapes)
267: (8)             return Succession(
268: (12)                 UpdateFromAlphaFunc(
269: (16)                     self,
270: (16)                     slide_and_uncover,
271: (16)                     run_time=run_time * 2 / 3,
272: (16)                     rate_func=ease_in_out_cubic,
273: (12)                 ),
274: (12)                 UpdateFromAlphaFunc(
275: (16)                     self,
276: (16)                     slide_back,
277: (16)                     run_time=run_time * 1 / 3,
278: (16)                     rate_func=smooth,
279: (12)                 ),
280: (8)             )

```

-----  
File 50 - line.py:

```

1: (0)             r"""Mobjects that are lines or variations of them."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "Line",
5: (4)                 "DashedLine",
6: (4)                 "TangentLine",
7: (4)                 "Elbow",
8: (4)                 "Arrow",
9: (4)                 "Vector",
10: (4)                "DoubleArrow",
11: (4)                "Angle",
12: (4)                "RightAngle",
13: (0)             ]

```

```

14: (0)             from typing import TYPE_CHECKING
15: (0)             import numpy as np
16: (0)             from manim import config
17: (0)             from manim.constants import *
18: (0)             from manim.mobject.geometry.arc import Arc, ArcBetweenPoints, Dot,
TipableVMobject
19: (0)             from manim.mobject.geometry.tips import ArrowTriangleFilledTip
20: (0)             from manim.mobject.mobject import Mobject
21: (0)             from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
22: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject
23: (0)             from manim.mobject.types.vectorized_mobject import DashedVMobject, VGroup,
VMobject
24: (0)             from manim.utils.color import WHITE
25: (0)             from manim.utils.space_ops import angle_of_vector, line_intersection,
normalize
26: (0)             if TYPE_CHECKING:
27: (4)                 from typing_extensions import Self
28: (4)                 from manim.typing import Point2D, Point3D, Vector3D
29: (4)                 from manim.utils.color import ParsableManimColor
30: (4)                 from ..matrix import Matrix # Avoid circular import
31: (0)             class Line(TipableVMobject):
32: (4)                 def __init__(
33: (8)                     self,
34: (8)                     start: Point3D = LEFT,
35: (8)                     end: Point3D = RIGHT,
36: (8)                     buff: float = 0,
37: (8)                     path_arc: float | None = None,
38: (8)                     **kwargs,
39: (4)                 ) -> None:
40: (8)                     self.dim = 3
41: (8)                     self.buff = buff
42: (8)                     self.path_arc = path_arc
43: (8)                     self._set_start_and_end_attrs(start, end)
44: (8)                     super().__init__(**kwargs)
45: (4)                 def generate_points(self) -> None:
46: (8)                     self.set_points_by_ends(
47: (12)                         start=self.start,
48: (12)                         end=self.end,
49: (12)                         buff=self.buff,
50: (12)                         path_arc=self.path_arc,
51: (8)                     )
52: (4)                 def set_points_by_ends(
53: (8)                     self,
54: (8)                     start: Point3D,
55: (8)                     end: Point3D,
56: (8)                     buff: float = 0,
57: (8)                     path_arc: float = 0,
58: (4)                 ) -> None:
59: (8)                     if path_arc:
60: (12)                         arc = ArcBetweenPoints(self.start, self.end, angle=self.path_arc)
61: (12)                         self.set_points(arc.points)
62: (8)                     else:
63: (12)                         self.set_points_as_corners([start, end])
64: (8)                         self._account_for_buff(buff)
65: (4)                     init_points = generate_points
66: (4)                     def _account_for_buff(self, buff: float) -> Self:
67: (8)                         if buff == 0:
68: (12)                             return
69: (8)                         if self.path_arc == 0:
70: (12)                             length = self.get_length()
71: (8)                         else:
72: (12)                             length = self.get_arc_length()
73: (8)                         if length < 2 * buff:
74: (12)                             return
75: (8)                         buff_proportion = buff / length
76: (8)                         self.pointwise_become_partial(self, buff_proportion, 1 -
buff_proportion)
77: (8)                         return self
78: (4)                     def _set_start_and_end_attrs(self, start: Point3D, end: Point3D) -> None:

```

```

79: (8)                     rough_start = self._pointify(start)
80: (8)                     rough_end = self._pointify(end)
81: (8)                     vect = normalize(rough_end - rough_start)
82: (8)                     self.start = self._pointify(start, vect)
83: (8)                     self.end = self._pointify(end, -vect)
84: (4)             def _pointify(
85: (8)                 self,
86: (8)                 mob_or_point: Mobject | Point3D,
87: (8)                 direction: Vector3D | None = None,
88: (4)             ) -> Point3D:
89: (8)                 """Transforms a mobject into its corresponding point. Does nothing if
a point is passed.
90: (8)                 ``direction`` determines the location of the point along its bounding
box in that direction.
91: (8)             Parameters
92: (8)             -----
93: (8)             mob_or_point
94: (12)                 The mobject or point.
95: (8)
96: (12)                 The direction.
97: (8)
98: (8)             if isinstance(mob_or_point, (Mobject, OpenGLMobject)):
99: (12)                 mob = mob_or_point
100: (12)                 if direction is None:
101: (16)                     return mob.get_center()
102: (12)                 else:
103: (16)                     return mob.get_boundary_point(direction)
104: (8)             return np.array(mob_or_point)
105: (4)             def set_path_arc(self, new_value: float) -> None:
106: (8)                 self.path_arc = new_value
107: (8)                 self.init_points()
108: (4)             def put_start_and_end_on(self, start: Point3D, end: Point3D) -> Self:
109: (8)                 """Sets starts and end coordinates of a line.
110: (8)             Examples
111: (8)
112: (8)                 .. manim:: LineExample
113: (12)                     class LineExample(Scene):
114: (16)                         def construct(self):
115: (20)                             d = VGroup()
116: (20)                             for i in range(0,10):
117: (24)                                 d.add(Dot())
118: (20)                             d.arrange_in_grid(buff=1)
119: (20)                             self.add(d)
120: (20)                             l= Line(d[0], d[1])
121: (20)                             self.add(l)
122: (20)                             self.wait()
123: (20)                             l.put_start_and_end_on(d[1].get_center(),
d[2].get_center())
124: (20)                             self.wait()
125: (20)                             l.put_start_and_end_on(d[4].get_center(),
d[7].get_center())
126: (20)                             self.wait()
127: (8)
128: (8)                         curr_start, curr_end = self.get_start_and_end()
129: (8)                         if np.all(curr_start == curr_end):
130: (12)                             self.start = start
131: (12)                             self.end = end
132: (12)                             self.generate_points()
133: (8)                         return super().put_start_and_end_on(start, end)
134: (4)             def get_vector(self) -> Vector3D:
135: (8)                 return self.get_end() - self.get_start()
136: (4)             def get_unit_vector(self) -> Vector3D:
137: (8)                 return normalize(self.get_vector())
138: (4)             def get_angle(self) -> float:
139: (8)                 return angle_of_vector(self.get_vector())
140: (4)             def get_projection(self, point: Point3D) -> Vector3D:
141: (8)                 """Returns the projection of a point onto a line.
142: (8)             Parameters
143: (8)

```

```

144: (8)           point
145: (12)         The point to which the line is projected.
146: (8)
147: (8)
148: (8)
149: (8)
150: (8)
151: (4)         def get_slope(self) -> float:
152: (8)           return np.tan(self.get_angle())
153: (4)         def set_angle(self, angle: float, about_point: Point3D | None = None) ->
Self:
154: (8)           if about_point is None:
155: (12)             about_point = self.get_start()
156: (8)           self.rotate(
157: (12)             angle - self.get_angle(),
158: (12)             about_point=about_point,
159: (8)
160: (8)
161: (4)         def set_length(self, length: float) -> Self:
162: (8)           return self.scale(length / self.get_length())
163: (0)         class DashedLine(Line):
164: (4)           """A dashed :class:`Line` .
165: (4)           Parameters
166: (4)           -----
167: (4)           args
168: (8)             Arguments to be passed to :class:`Line`
169: (4)           dash_length
170: (8)             The length of each individual dash of the line.
171: (4)           dashed_ratio
172: (8)             The ratio of dash space to empty space. Range of 0-1.
173: (4)           kwargs
174: (8)             Additional arguments to be passed to :class:`Line`
175: (4)           .. seealso::
176: (8)             :class:`~.DashedVMobject`
177: (4)           Examples
178: (4)           -----
179: (4)           .. manim:: DashedLineExample
180: (8)             :save_last_frame:
181: (8)             class DashedLineExample(Scene):
182: (12)               def construct(self):
183: (16)                 dashed_1 = DashedLine(config.left_side, config.right_side,
dash_length=2.0).shift(UP*2)
184: (16)                 dashed_2 = DashedLine(config.left_side, config.right_side)
185: (16)                 dashed_3 = DashedLine(config.left_side, config.right_side,
dashed_ratio=0.1).shift(DOWN*2)
186: (16)                 self.add(dashed_1, dashed_2, dashed_3)
187: (4)               """
188: (4)               def __init__(
189: (8)                 self,
190: (8)                 *args,
191: (8)                 dash_length: float = DEFAULT_DASH_LENGTH,
192: (8)                 dashed_ratio: float = 0.5,
193: (8)                 **kwargs,
194: (4)             ) -> None:
195: (8)               self.dash_length = dash_length
196: (8)               self.dashed_ratio = dashed_ratio
197: (8)               super().__init__(*args, **kwargs)
198: (8)               dashes = DashedVMobject(
199: (12)                 self,
200: (12)                 num_dashes=self._calculate_num_dashes(),
201: (12)                 dashed_ratio=dashed_ratio,
202: (8)
203: (8)
204: (8)
205: (4)             def _calculate_num_dashes(self) -> int:
206: (8)               """Returns the number of dashes in the dashed line.
207: (8)               Examples
208: (8)               -----
209: (8)               ::
```

```

210: (12)                               >>> DashedLine().__calculate_num_dashes()
211: (12)                               20
212: (8)                                """
213: (8)                                return max(
214: (12)                                2,
215: (12)                                int(np.ceil((self.get_length() / self.dash_length) *
self.dashed_ratio)),
216: (8)                                )
217: (4)      def get_start(self) -> Point3D:
218: (8)        """Returns the start point of the line.
219: (8)        Examples
220: (8)        -----
221: (8)        :::
222: (12)          >>> DashedLine().get_start()
223: (12)          array([-1.,  0.,  0.])
224: (8)        """
225: (8)        if len(self.submobjects) > 0:
226: (12)          return self.submobjects[0].get_start()
227: (8)        else:
228: (12)          return super().get_start()
229: (4)      def get_end(self) -> Point3D:
230: (8)        """Returns the end point of the line.
231: (8)        Examples
232: (8)        -----
233: (8)        :::
234: (12)          >>> DashedLine().get_end()
235: (12)          array([1.,  0.,  0.])
236: (8)        """
237: (8)        if len(self.submobjects) > 0:
238: (12)          return self.submobjects[-1].get_end()
239: (8)        else:
240: (12)          return super().get_end()
241: (4)      def get_first_handle(self) -> Point3D:
242: (8)        """Returns the point of the first handle.
243: (8)        Examples
244: (8)        -----
245: (8)        :::
246: (12)          >>> DashedLine().get_first_handle()
247: (12)          array([-0.98333333,  0.           ,  0.           ])
248: (8)        """
249: (8)        return self.submobjects[0].points[1]
250: (4)      def get_last_handle(self) -> Point3D:
251: (8)        """Returns the point of the last handle.
252: (8)        Examples
253: (8)        -----
254: (8)        :::
255: (12)          >>> DashedLine().get_last_handle()
256: (12)          array([0.98333333, 0.           , 0.           ])
257: (8)        """
258: (8)        return self.submobjects[-1].points[-2]
259: (0)    class TangentLine(Line):
260: (4)      """Constructs a line tangent to a :class:`~.VMobject` at a specific point.
261: (4)      Parameters
262: (4)      -----
263: (4)      vmob
264: (8)        The VMobject on which the tangent line is drawn.
265: (4)      alpha
266: (8)        How far along the shape that the line will be constructed. range: 0-1.
267: (4)      length
268: (8)        Length of the tangent line.
269: (4)      d_alpha
270: (8)        The ``dx`` value
271: (4)      kwargs
272: (8)        Additional arguments to be passed to :class:`Line`
273: (4)      .. seealso::
274: (8)        :meth:`~.VMobject.point_from_proportion`
275: (4)      Examples
276: (4)      -----
277: (4)      .. manim:: TangentLineExample

```

```

278: (8)                      :save_last_frame:
279: (8)                      class TangentLineExample(Scene):
280: (12)                     def construct(self):
281: (16)                     circle = Circle(radius=2)
282: (16)                     line_1 = TangentLine(circle, alpha=0.0, length=4,
color=BLUE_D) # right           line_2 = TangentLine(circle, alpha=0.4, length=4, color=GREEN)
283: (16)                         self.add(circle, line_1, line_2)
284: (16)                     """
285: (4)                     def __init__(
286: (4)                         self,
287: (8)                         vmob: VMobject,
288: (8)                         alpha: float,
289: (8)                         length: float = 1,
290: (8)                         d_alpha: float = 1e-6,
291: (8)                         **kwargs,
292: (8)                     ) -> None:
293: (4)                         self.length = length
294: (8)                         self.d_alpha = d_alpha
295: (8)                         da = self.d_alpha
296: (8)                         a1 = np.clip(alpha - da, 0, 1)
297: (8)                         a2 = np.clip(alpha + da, 0, 1)
298: (8)                         super().__init__(
299: (8)                             vmob.point_from_proportion(a1), vmob.point_from_proportion(a2),
300: (12)                         **kwargs
301: (8)                     )
302: (8)                     self.scale(self.length / self.get_length())
303: (0)                     class Elbow(VMobject, metaclass=ConvertToOpenGL):
304: (4)                         """Two lines that create a right angle about each other: L-shape.
305: (4)                         Parameters
306: (4)                         -----
307: (4)                         width
308: (8)                           The length of the elbow's sides.
309: (4)                         angle
310: (8)                           The rotation of the elbow.
311: (4)                         kwargs
312: (8)                           Additional arguments to be passed to :class:`~.VMobject`  

313: (4)                         .. seealso::
314: (8)                           :class:`RightAngle`
315: (4)                         Examples
316: (4)                         -----
317: (4)                         .. manim:: ElbowExample
318: (8)                           :save_last_frame:
319: (8)                           class ElbowExample(Scene):
320: (12)                             def construct(self):
321: (16)                               elbow_1 = Elbow()
322: (16)                               elbow_2 = Elbow(width=2.0)
323: (16)                               elbow_3 = Elbow(width=2.0, angle=5*PI/4)
324: (16)                               elbow_group = Group(elbow_1, elbow_2, elbow_3).arrange(buff=1)
325: (16)                               self.add(elbow_group)
326: (4)                             """
327: (4)                         def __init__(self, width: float = 0.2, angle: float = 0, **kwargs) ->
None:
328: (8)                           self.angle = angle
329: (8)                           super().__init__(**kwargs)
330: (8)                           self.set_points_as_corners([UP, UP + RIGHT, RIGHT])
331: (8)                           self.scale_to_fit_width(width, about_point=ORIGIN)
332: (8)                           self.rotate(self.angle, about_point=ORIGIN)
333: (0)                     class Arrow(Line):
334: (4)                         """An arrow.
335: (4)                         Parameters
336: (4)                         -----
337: (4)                         args
338: (8)                           Arguments to be passed to :class:`Line`.
339: (4)                           stroke_width
340: (8)                           The thickness of the arrow. Influenced by
:attr:`max_stroke_width_to_length_ratio`.
341: (4)                           buff

```

```

342: (8)                 The distance of the arrow from its start and end points.
343: (4)                 max_tip_length_to_length_ratio
344: (8)                 :attr:`tip_length` scales with the length of the arrow. Increasing
this ratio raises the max value of :attr:`tip_length`.
345: (4)                 max_stroke_width_to_length_ratio
346: (8)                 :attr:`stroke_width` scales with the length of the arrow. Increasing
this ratio ratios the max value of :attr:`stroke_width`.
347: (4)                 kwargs
348: (8)                 Additional arguments to be passed to :class:`Line`.
349: (4)                 .. seealso::
350: (8)                     :class:`ArrowTip`
351: (8)                     :class:`CurvedArrow`
352: (4)                 Examples
353: (4)                 -----
354: (4)                 .. manim:: ArrowExample
355: (8)                 :save_last_frame:
356: (8)                     from manim.mobject.geometry.tips import ArrowSquareTip
357: (8)                     class ArrowExample(Scene):
358: (12)                     def construct(self):
359: (16)                         arrow_1 = Arrow(start=RIGHT, end=LEFT, color=GOLD)
360: (16)                         arrow_2 = Arrow(start=RIGHT, end=LEFT, color=GOLD,
tip_shape=ArrowSquareTip).shift(DOWN)
361: (16)                         g1 = Group(arrow_1, arrow_2)
362: (16)                         square = Square(color=MAROON_A)
363: (16)                         arrow_3 = Arrow(start=LEFT, end=RIGHT)
364: (16)                         arrow_4 = Arrow(start=LEFT, end=RIGHT,
buff=0).next_to(arrow_1, UP)
365: (16)                         g2 = Group(arrow_3, arrow_4, square)
366: (16)                         arrow_5 = Arrow(start=ORIGIN, end=config.top).shift(LEFT * 4)
367: (16)                         arrow_6 = Arrow(start=config.top + DOWN,
end=config.top).shift(LEFT * 3)
368: (16)                         g3 = Group(arrow_5, arrow_6)
369: (16)                         self.add(Group(g1, g2, g3).arrange(buff=2))
370: (4)                 .. manim:: ArrowExample
371: (8)                 :save_last_frame:
372: (8)                 class ArrowExample(Scene):
373: (12)                     def construct(self):
374: (16)                         left_group = VGroup()
375: (16)                         for buff in np.arange(0, 2.2, 0.45):
376: (20)                             left_group += Arrow(buff=buff, start=2 * LEFT, end=2 *
RIGHT)
377: (16)                         left_group.arrange(DOWN)
378: (16)                         left_group.move_to(4 * LEFT)
379: (16)                         middle_group = VGroup()
380: (16)                         for i in np.arange(0, 5, 0.5):
381: (20)                             middle_group += Arrow(max_stroke_width_to_length_ratio=i)
382: (16)                         middle_group.arrange(DOWN)
383: (16)                         UR_group = VGroup()
384: (16)                         for i in np.arange(0, 0.3, 0.1):
385: (20)                             UR_group += Arrow(max_tip_length_to_length_ratio=i)
386: (16)                         UR_group.arrange(DOWN)
387: (16)                         UR_group.move_to(4 * RIGHT + 2 * UP)
388: (16)                         DR_group = VGroup()
389: (16)                         DR_group += Arrow(start=LEFT, end=RIGHT, color=BLUE,
tip_shape=ArrowSquareTip)
390: (16)                         DR_group += Arrow(start=LEFT, end=RIGHT, color=BLUE,
tip_shape=ArrowSquareFilledTip)
391: (16)                         DR_group += Arrow(start=LEFT, end=RIGHT, color=YELLOW,
tip_shape=ArrowCircleTip)
392: (16)                         DR_group += Arrow(start=LEFT, end=RIGHT, color=YELLOW,
tip_shape=ArrowCircleFilledTip)
393: (16)                         DR_group.arrange(DOWN)
394: (16)                         DR_group.move_to(4 * RIGHT + 2 * DOWN)
395: (16)                         self.add(left_group, middle_group, UR_group, DR_group)
396: (4)                         """
397: (4)                     def __init__(
398: (8)                         self,
399: (8)                         *args,
400: (8)                         stroke_width: float = 6,
```

```

401: (8)                                buff: float = MED_SMALL_BUFF,
402: (8)                                max_tip_length_to_length_ratio: float = 0.25,
403: (8)                                max_stroke_width_to_length_ratio: float = 5,
404: (8)                                **kwargs,
405: (4)        ) -> None:
406: (8)                                self.max_tip_length_to_length_ratio = max_tip_length_to_length_ratio
407: (8)                                self.max_stroke_width_to_length_ratio =
max_stroke_width_to_length_ratio
408: (8)                                tip_shape = kwargs.pop("tip_shape", ArrowTriangleFilledTip)
409: (8)                                super().__init__(*args, buff=	buff, stroke_width=stroke_width,
**kwargs)
410: (8)                                self.initial_stroke_width = self.stroke_width
411: (8)                                self.add_tip(tip_shape=tip_shape)
412: (8)                                self._set_stroke_width_from_length()
413: (4)        def scale(self, factor: float, scale_tips: bool = False, **kwargs) ->
Self:
414: (8)            r"""Scale an arrow, but keep stroke width and arrow tip size fixed.
415: (8)            .. seealso::
416: (12)                :meth:`~.Mobject.scale`  

417: (8)            Examples
418: (8)            -----
419: (8)            ::  

420: (12)                >>> arrow = Arrow(np.array([-1, -1, 0]), np.array([1, 1, 0]),
buff=0)
421: (12)                >>> scaled_arrow = arrow.scale(2)
422: (12)                >>> np.round(scaled_arrow.get_start_and_end(), 8) + 0
array([[-2., -2.,  0.],
       [ 2.,  2.,  0.]])
423: (12)                >>> arrow.tip.length == scaled_arrow.tip.length
424: (19)                True
425: (12)            Manually scaling the object using the default method
426: (12)            :meth:`~.Mobject.scale` does not have the same properties:
427: (8)            ::  

428: (8)                >>> new_arrow = Arrow(np.array([-1, -1, 0]), np.array([1, 1, 0]),
buff=0)
429: (12)                >>> another_scaled_arrow = VMobject.scale(new_arrow, 2)
430: (12)                >>> another_scaled_arrow.tip.length == arrow.tip.length
431: (12)                False
432: (12)            """
433: (8)            if self.get_length() == 0:
434: (8)                return self
435: (12)            if scale_tips:
436: (8)                super().scale(factor, **kwargs)
437: (12)                self._set_stroke_width_from_length()
438: (12)                return self
439: (12)            has_tip = self.has_tip()
440: (8)            has_start_tip = self.has_start_tip()
441: (8)            if has_tip or has_start_tip:
442: (8)                old_tips = self.pop_tips()
443: (12)            super().scale(factor, **kwargs)
444: (8)            self._set_stroke_width_from_length()
445: (8)            if has_tip:
446: (8)                self.add_tip(tip=old_tips[0])
447: (12)            if has_start_tip:
448: (8)                self.add_tip(tip=old_tips[1], at_start=True)
449: (12)            return self
450: (8)        def get_normal_vector(self) -> Vector3D:
451: (4)            """Returns the normal of a vector.
452: (8)            Examples
453: (8)            -----
454: (8)            ::  

455: (8)                >>> np.round(Arrow().get_normal_vector()) + 0. # add 0. to avoid
negative 0 in output
456: (12)                array([ 0.,  0., -1.])
457: (12)            """
458: (8)            p0, p1, p2 = self.tip.get_start_anchors()[:3]
459: (8)            return normalize(np.cross(p2 - p1, p1 - p0))
460: (8)        def reset_normal_vector(self) -> Self:
461: (4)            """Resets the normal of a vector"""
462: (8)            self.normal_vector = self.get_normal_vector()
463: (8)

```

```

464: (8)             return self
465: (4)         def get_default_tip_length(self) -> float:
466: (8)             """Returns the default tip_length of the arrow.
467: (8)             Examples
468: (8)             -----
469: (8)             :::
470: (12)                 >>> Arrow().get_default_tip_length()
471: (12)                 0.35
472: (8)
473: (8)             """
474: (8)             max_ratio = self.max_tip_length_to_length_ratio
475: (4)         return min(self.tip_length, max_ratio * self.get_length())
476: (8)
477: (8)     def _set_stroke_width_from_length(self) -> Self:
478: (8)         """Sets stroke width based on length."""
479: (8)         max_ratio = self.max_stroke_width_to_length_ratio
480: (12)         if config.renderer == RendererType.OPENGL:
481: (16)             self.set_stroke(
482: (16)                 width=min(self.initial_stroke_width, max_ratio *
483: (16)                     recurse=False,
484: (12)             )
485: (16)         else:
486: (16)             self.set_stroke(
487: (16)                 width=min(self.initial_stroke_width, max_ratio *
488: (16)                     family=False,
489: (12)             )
490: (8)             return self
491: (0)         class Vector(Arrow):
492: (4)             """A vector specialized for use in graphs.
493: (4)             .. caution::
494: (8)                 Do not confuse with the :class:`~.Vector2D`,
495: (8)                 :class:`~.Vector3D` or :class:`~.VectorND` type aliases,
496: (8)                 which are not Mobjects!
497: (4)             Parameters
498: (4)             -----
499: (4)             direction
500: (8)                 The direction of the arrow.
501: (4)             buff
502: (8)                 The distance of the vector from its endpoints.
503: (4)             kwargs
504: (4)                 Additional arguments to be passed to :class:`Arrow`
505: (4)             Examples
506: (4)             -----
507: (4)             .. manim:: VectorExample
508: (8)                 :save_last_frame:
509: (8)                 class VectorExample(Scene):
510: (12)                     def construct(self):
511: (16)                         plane = NumberPlane()
512: (16)                         vector_1 = Vector([1,2])
513: (16)                         vector_2 = Vector([-5,-2])
514: (16)                         self.add(plane, vector_1, vector_2)
515: (8)
516: (4)             def __init__(
517: (8)                 self, direction: Point2D | Point3D = RIGHT, buff: float = 0, **kwargs
518: (4)             ) -> None:
519: (8)                 self.buff = buff
520: (8)                 if len(direction) == 2:
521: (8)                     direction = np.hstack([direction, 0])
522: (4)             super().__init__(ORIGIN, direction, buff=buff, **kwargs)
523: (8)             def coordinate_label(
524: (8)                 self,
525: (8)                 integer_labels: bool = True,
526: (8)                 n_dim: int = 2,
527: (8)                 color: ParsableManimColor | None = None,
528: (8)                 **kwargs,
529: (4)             ) -> Matrix:
530: (8)                 """Creates a label based on the coordinates of the vector.
531: (8)                 Parameters
532: (8)                 -----

```

```

531: (8)             integer_labels
532: (12)            Whether or not to round the coordinates to integers.
533: (8)
534: (12)            n_dim
535: (8)            The number of dimensions of the vector.
536: (12)            color
537: (8)            Sets the color of label, optional.
538: (12)            kwargs
539: (8)            Additional arguments to be passed to :class:`~.Matrix`.
540: (8)
541: (8)            Returns
542: (12)            -----
543: (8)            :class:`~.Matrix`
544: (8)            The label.
545: (8)            Examples
546: (12)            -----
547: (12)            .. manim:: VectorCoordinateLabel
548: (16)            :save_last_frame:
549: (20)            class VectorCoordinateLabel(Scene):
550: (20)            def construct(self):
551: (20)            plane = NumberPlane()
552: (20)            vec_1 = Vector([1, 2])
553: (20)            vec_2 = Vector([-3, -2])
554: (20)            label_1 = vec_1.coordinate_label()
555: (8)            label_2 = vec_2.coordinate_label(color=YELLOW)
556: (8)            self.add(plane, vec_1, vec_2, label_1, label_2)
557: (8)
558: (8)
559: (12)            """
560: (8)            from ..matrix import Matrix
561: (8)            vect = np.array(self.get_end())
562: (8)            if integer_labels:
563: (8)            vect = np.round(vect).astype(int)
564: (8)            vect = vect[:n_dim]
565: (8)            vect = vect.reshape((n_dim, 1))
566: (12)            label = Matrix(vect, **kwargs)
567: (8)            label.scale(LARGE_BUFF - 0.2)
568: (12)            shift_dir = np.array(self.get_end())
569: (8)            if shift_dir[0] >= 0: # Pointing right
570: (8)            shift_dir -= label.get_left() + DEFAULT_MOBJECT_TO_MOBJECT_BUFFER
571: (8)            * LEFT
572: (8)            else: # Pointing left
573: (0)            shift_dir -= label.get_right() + DEFAULT_MOBJECT_TO_MOBJECT_BUFFER
574: (4)            * RIGHT
575: (4)            label.shift(shift_dir)
576: (4)            if color is not None:
577: (4)            label.set_color(color)
578: (8)            return label
579: (4)
580: (8)            class DoubleArrow(Arrow):
581: (4)            """An arrow with tips on both ends.
582: (4)            Parameters
583: (4)            -----
584: (4)            args
585: (4)            Arguments to be passed to :class:`Arrow`
586: (4)            kwargs
587: (8)            Additional arguments to be passed to :class:`Arrow`
588: (4)            .. seealso::
589: (8)            :class:`~ArrowTip`
590: (12)            :class:`~CurvedDoubleArrow`
591: (16)
592: (16)
593: (16)
594: (16)
end=circle.get_right())
tip_shape_start=ArrowCircleFilledTip)
group = Group(Group(circle, d_arrow), d_arrow_2).arrange(UP,
buff=1)

```

```

595: (16)                     self.add(group)
596: (4)          .. manim:: DoubleArrowExample2
597: (8)          :save_last_frame:
598: (8)          class DoubleArrowExample2(Scene):
599: (12)          def construct(self):
600: (16)              box = Square()
601: (16)              p1 = box.get_left()
602: (16)              p2 = box.get_right()
603: (16)              d1 = DoubleArrow(p1, p2, buff=0)
604: (16)              d2 = DoubleArrow(p1, p2, buff=0, tip_length=0.2, color=YELLOW)
605: (16)              d3 = DoubleArrow(p1, p2, buff=0, tip_length=0.4, color=BLUE)
606: (16)              Group(d1, d2, d3).arrange(DOWN)
607: (16)              self.add(box, d1, d2, d3)
608: (4)
609: (4)          """
610: (8)          def __init__(self, *args, **kwargs) -> None:
611: (12)              if "tip_shape_end" in kwargs:
612: (8)                  kwargs["tip_shape"] = kwargs.pop("tip_shape_end")
613: (8)                  tip_shape_start = kwargs.pop("tip_shape_start",
614: (8)                      ArrowTriangleFilledTip)
615: (0)          super().__init__(*args, **kwargs)
616: (4)          self.add_tip(at_start=True, tip_shape=tip_shape_start)
617: (4)      class Angle(VMobject, metaclass=ConvertToOpenGL):
618: (4)          """
619: (4)          A circular arc or elbow-type mobject representing an angle of two
620: (4)          lines.
621: (4)          Parameters
622: (4)          -----
623: (4)          line1 :
624: (8)              The first line.
625: (4)          line2 :
626: (8)              The second line.
627: (4)          radius :
628: (8)              The radius of the :class:`Arc`.
629: (4)          quadrant
630: (8)              A sequence of two :class:`int` numbers determining which of the 4
631: (4)          quadrants should be used.
632: (8)          closer to the end point (1)
633: (4)              The first value indicates whether to anchor the arc on the first line
634: (8)              or start point (-1), and the second value functions similarly for the
635: (8)              end (1) or start (-1) of the second line.
636: (4)              Possibilities: (1,1), (-1,1), (1,-1), (-1,-1).
637: (4)          other_angle :
638: (8)              Toggles between the two possible angles defined by two points and an
639: (4)          arc center. If set to
640: (8)              False (default), the arc will always go counterclockwise from the
641: (4)          point on line1 until
642: (8)              the point on line2 is reached. If set to True, the angle will go
643: (4)          clockwise from line1 to line2.
644: (4)          dot
645: (8)              Allows for a :class:`Dot` in the arc. Mainly used as an convention to
646: (4)          indicate a right angle.
647: (8)              The dot can be customized in the next three parameters.
648: (4)          dot_radius
649: (8)              The radius of the :class:`Dot`. If not specified otherwise, this
650: (4)          radius will be 1/10 of the arc radius.
651: (4)          dot_distance
652: (8)              Relative distance from the center to the arc: 0 puts the dot in the
653: (4)          center and 1 on the arc itself.
654: (4)          dot_color
655: (8)              The color of the :class:`Dot`.
656: (4)          elbow
657: (8)              Produces an elbow-type mobject indicating a right angle, see
658: (4)          :class:`RightAngle` for more information
659: (4)          and a shorthand.
660: (4)          **kwargs
661: (8)              Further keyword arguments that are passed to the constructor of
662: (4)          :class:`Arc` or :class:`Elbow`.
663: (4)          Examples
664: (4)          -----
665: (4)          The first example shows some right angles with a dot in the middle while

```

the second example shows

```

652: (4)           all 8 possible angles defined by two lines.
653: (4)           .. manim:: RightArcAngleExample
654: (8)           :save_last_frame:
655: (8)           class RightArcAngleExample(Scene):
656: (12)          def construct(self):
657: (16)          line1 = Line( LEFT, RIGHT )
658: (16)          line2 = Line( DOWN, UP )
659: (16)          rightarcangles = [
660: (20)          Angle(line1, line2, dot=True),
661: (20)          Angle(line1, line2, radius=0.4, quadrant=(1,-1), dot=True,
other_angle=True),
662: (20)          Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8, dot=True, dot_color=YELLOW, dot_radius=0.04, other_angle=True),
663: (20)          Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED, dot=True, dot_color=GREEN, dot_radius=0.08),
664: (16)
665: (16)
666: (16)
667: (20)          plots = VGroup()
668: (20)          for angle in rightarcangles:
669: (16)            plot=VGroup(line1.copy(),line2.copy(), angle)
670: (16)            plots.add(plot)
671: (4)            plots.arrange(buff=1.5)
672: (8)            self.add(plots)
673: (8)           .. manim:: AngleExample
674: (12)          :save_last_frame:
675: (16)          class AngleExample(Scene):
676: (16)          def construct(self):
677: (16)            line1 = Line( LEFT + (1/3) * UP, RIGHT + (1/3) * DOWN )
678: (20)            line2 = Line( DOWN + (1/3) * RIGHT, UP + (1/3) * LEFT )
679: (20)            angles = [
680: (20)              Angle(line1, line2),
681: (20)              Angle(line1, line2, radius=0.4, quadrant=(1,-1),
other_angle=True),
682: (20)              Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8, other_angle=True),
683: (20)              Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED),
684: (20)              Angle(line1, line2, other_angle=True),
685: (20)              Angle(line1, line2, radius=0.4, quadrant=(1,-1)),
686: (16)
687: (16)
688: (16)
689: (20)              Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8),
690: (20)              Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED, other_angle=True),
691: (16)
692: (16)
693: (4)           plots = VGroup()
694: (8)           for angle in angles:
695: (8)             plot=VGroup(line1.copy(),line2.copy(), angle)
696: (12)             plots.add(VGroup(plot, SurroundingRectangle(plot,
buff=0.3)))
697: (16)
698: (16)
699: (20)
700: (20)
701: (20)
702: (16)
703: (16)
704: (16)
705: (16)
706: (16)
707: (16)

```

the second example shows

```

    .. manim:: RightArcAngleExample
    :save_last_frame:
    class RightArcAngleExample(Scene):
        def construct(self):
            line1 = Line( LEFT, RIGHT )
            line2 = Line( DOWN, UP )
            rightarcangles = [
                Angle(line1, line2, dot=True),
                Angle(line1, line2, radius=0.4, quadrant=(1,-1), dot=True,
other_angle=True),
                Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8, dot=True, dot_color=YELLOW, dot_radius=0.04, other_angle=True),
                Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED, dot=True, dot_color=GREEN, dot_radius=0.08),
            ]
            plots = VGroup()
            for angle in rightarcangles:
                plot=VGroup(line1.copy(),line2.copy(), angle)
                plots.add(plot)
            plots.arrange(buff=1.5)
            self.add(plots)

    .. manim:: AngleExample
    :save_last_frame:
    class AngleExample(Scene):
        def construct(self):
            line1 = Line( LEFT + (1/3) * UP, RIGHT + (1/3) * DOWN )
            line2 = Line( DOWN + (1/3) * RIGHT, UP + (1/3) * LEFT )
            angles = [
                Angle(line1, line2),
                Angle(line1, line2, radius=0.4, quadrant=(1,-1),
other_angle=True),
                Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8, other_angle=True),
                Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED),
                Angle(line1, line2, other_angle=True),
                Angle(line1, line2, radius=0.4, quadrant=(1,-1)),
                Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8),
                Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED, other_angle=True),
            ]
            plots = VGroup()
            for angle in angles:
                plot=VGroup(line1.copy(),line2.copy(), angle)
                plots.add(VGroup(plot, SurroundingRectangle(plot,
buff=0.3)))
            plots.arrange_in_grid(rows=2,buff=1)
            self.add(plots)

    .. manim:: FilledAngle
    :save_last_frame:
    class FilledAngle(Scene):
        def construct(self):
            l1 = Line(ORIGIN, 2 * UP + RIGHT).set_color(GREEN)
            l2 = (
                Line(ORIGIN, 2 * UP + RIGHT)
                .set_color(GREEN)
                .rotate(-20 * DEGREES, about_point=ORIGIN)
            )
            norm = l1.get_length()
            a1 = Angle(l1, l2, other_angle=True, radius=norm -
0.5).set_color(GREEN)
            a2 = Angle(l1, l2, other_angle=True,
radius=norm).set_color(GREEN)
            q1 = a1.points # save all coordinates of points of angle a1
            q2 = a2.reverse_direction().points # save all coordinates of
points of angle a1 (in reversed direction)

```

```

708: (16)           pnts = np.concatenate([q1, q2, q1[0].reshape(1, 3)]) # adds
points and ensures that path starts and ends at same point
709: (16)           mfill = VMobject().set_color(ORANGE)
710: (16)           mfill.set_points_as_corners(pnts).set_fill(GREEN, opacity=1)
711: (16)           self.add(l1, l2)
712: (16)           self.add(mfill)
713: (4)
714: (4)       def __init__(self,
715: (8)           line1: Line,
716: (8)           line2: Line,
717: (8)           radius: float | None = None,
718: (8)           quadrant: Point2D = (1, 1),
719: (8)           other_angle: bool = False,
720: (8)           dot: bool = False,
721: (8)           dot_radius: float | None = None,
722: (8)           dot_distance: float = 0.55,
723: (8)           dot_color: ParsableManimColor = WHITE,
724: (8)           elbow: bool = False,
725: (8)           **kwargs,
726: (8)
727: (4)       ) -> None:
728: (8)           super().__init__(**kwargs)
729: (8)           self.lines = (line1, line2)
730: (8)           self.quadrant = quadrant
731: (8)           self.dot_distance = dot_distance
732: (8)           self.elbow = elbow
733: (8)           inter = line_intersection(
734: (12)               [line1.get_start(), line1.get_end()],
735: (12)               [line2.get_start(), line2.get_end()],
736: (8)
737: (8)
738: (12)           if radius is None:
739: (16)               if quadrant[0] == 1:
740: (12)                   dist_1 = np.linalg.norm(line1.get_end() - inter)
741: (16)               else:
742: (12)                   dist_1 = np.linalg.norm(line1.get_start() - inter)
743: (16)               if quadrant[1] == 1:
744: (12)                   dist_2 = np.linalg.norm(line2.get_end() - inter)
745: (16)               else:
746: (12)                   dist_2 = np.linalg.norm(line2.get_start() - inter)
747: (16)               if np.minimum(dist_1, dist_2) < 0.6:
748: (12)                   radius = (2 / 3) * np.minimum(dist_1, dist_2)
749: (16)               else:
750: (12)                   radius = 0.4
751: (12)           else:
752: (8)               self.radius = radius
753: (8)               anchor_angle_1 = inter + quadrant[0] * radius *
754: (8)               anchor_angle_2 = inter + quadrant[1] * radius *
755: (12)
756: (16)
757: (16)
758: (16)
759: (12)
760: (12)
761: (12)
762: (16)
763: (12)
764: (8)
765: (12)
766: (12)
767: (12)
768: (16)
769: (16)
770: (20)
771: (16)
772: (20)
773: (12)

```

```

774: (16)                         start_angle = angle_1
775: (16)                         if angle_2 < angle_1:
776: (20)                             angle_fin = -angle_1 + angle_2
777: (16)                         else:
778: (20)                             angle_fin = -2 * np.pi + (angle_2 - angle_1)
779: (12)                         self.angle_value = angle_fin
780: (12)                         angle_mobject = Arc(
781: (16)                             radius=radius,
782: (16)                             angle=self.angle_value,
783: (16)                             start_angle=start_angle,
784: (16)                             arc_center=inter,
785: (16)                             **kwargs,
786: (12)                         )
787: (12)                         if dot:
788: (16)                             if dot_radius is None:
789: (20)                                 dot_radius = radius / 10
790: (16)                             else:
791: (20)                                 self.dot_radius = dot_radius
792: (16)                             right_dot = Dot(ORIGIN, radius=dot_radius, color=dot_color)
793: (16)                             dot_anchor = (
794: (20)                                 inter
795: (20)                                 + (angle_mobject.get_center() - inter)
796: (20)                                 / np.linalg.norm(angle_mobject.get_center() - inter)
797: (20)                                 * radius
798: (20)                                 * dot_distance
799: (16)                             )
800: (16)                             right_dot.move_to(dot_anchor)
801: (16)                             self.add(right_dot)
802: (8)                         self.set_points(angle_mobject.points)
803: (4)                         def get_lines(self) -> VGroup:
804: (8)                             """Get the lines forming an angle of the :class:`Angle` class.
805: (8)                             Returns
806: (8)                             -----
807: (8)                             :class:`~.VGroup`
808: (12)                             A :class:`~.VGroup` containing the lines that form the angle of
the :class:`Angle` class.
809: (8)                         Examples
810: (8)                         -----
811: (8)                         :::
812: (12)                             >>> line_1, line_2 = Line(ORIGIN, RIGHT), Line(ORIGIN, UR)
813: (12)                             >>> angle = Angle(line_1, line_2)
814: (12)                             >>> angle.get_lines()
815: (12)                             VGroup(Line, Line)
816: (8)                         """
817: (8)                         return VGroup(*self.lines)
818: (4)                         def get_value(self, degrees: bool = False) -> float:
819: (8)                             """Get the value of an angle of the :class:`Angle` class.
820: (8)                             Parameters
821: (8)                             -----
822: (8)                             degrees
823: (12)                             A boolean to decide the unit (deg/rad) in which the value of the
angle is returned.
824: (8)                         Returns
825: (8)                         -----
826: (8)                         :class:`float`
827: (12)                         The value in degrees/radians of an angle of the :class:`Angle`  

class.
828: (8)                         Examples
829: (8)                         -----
830: (8)                         .. manim:: GetValueExample
831: (12)                             :save_last_frame:
832: (12)                             class GetValueExample(Scene):
833: (16)                             def construct(self):
834: (20)                                 line1 = Line(LEFT+(1/3)*UP, RIGHT+(1/3)*DOWN)
835: (20)                                 line2 = Line(DOWN+(1/3)*RIGHT, UP+(1/3)*LEFT)
836: (20)                                 angle = Angle(line1, line2, radius=0.4)
837: (20)                                 value = DecimalNumber(angle.get_value(degrees=True),
unit="^{\circ}")
838: (20)                                 value.next_to(angle, UR)

```

```

839: (20)                                     self.add(line1, line2, angle, value)
840: (8)                                     """
841: (8)                                     return self.angle_value / DEGREES if degrees else self.angle_value
842: (4)                                     @staticmethod
843: (4)                                     def from_three_points(A: Point3D, B: Point3D, C: Point3D, **kwargs) ->
Angle:
844: (8)                                     """The angle between the lines AB and BC.
845: (8)                                     This constructs the angle :math:`\angle ABC`.
846: (8)                                     Parameters
847: (8)                                     -----
848: (8)                                     A
849: (12)                                     The endpoint of the first angle leg
850: (8)                                     B
851: (12)                                     The vertex of the angle
852: (8)                                     C
853: (12)                                     The endpoint of the second angle leg
854: (8)                                     **kwargs
855: (12)                                     Further keyword arguments are passed to :class:`.Angle`
856: (8)                                     Returns
857: (8)                                     -----
858: (8)                                     The Angle calculated from the three points
859: (20)                                     Angle(line1, line2, radius=0.5, quadrant=(-1,1),
stroke_width=8),
860: (20)                                     Angle(line1, line2, radius=0.7, quadrant=(-1,-1),
color=RED, other_angle=True),
861: (8)                                     Examples
862: (8)                                     -----
863: (8)                                     .. manim:: AngleFromThreePointsExample
864: (12)                                     :save_last_frame:
865: (12)                                     class AngleFromThreePointsExample(Scene):
866: (16)                                     def construct(self):
867: (20)                                     sample_angle = Angle.from_three_points(UP, ORIGIN, LEFT)
868: (20)                                     red_angle = Angle.from_three_points(LEFT + UP, ORIGIN,
RIGHT, radius=.8, quadrant=(-1,-1), color=RED, stroke_width=8, other_angle=True)
869: (20)                                     self.add(red_angle, sample_angle)
870: (8)                                     """
871: (8)                                     return Angle(Line(B, A), Line(B, C), **kwargs)
872: (0)                                     class RightAngle(Angle):
873: (4)                                     """An elbow-type mobject representing a right angle between two lines.
874: (4)                                     Parameters
875: (4)                                     -----
876: (4)                                     line1
877: (8)                                     The first line.
878: (4)                                     line2
879: (8)                                     The second line.
880: (4)                                     length
881: (8)                                     The length of the arms.
882: (4)                                     **kwargs
883: (8)                                     Further keyword arguments that are passed to the constructor of
:class:`Angle`.
884: (4)                                     Examples
885: (4)                                     -----
886: (4)                                     .. manim:: RightAngleExample
887: (8)                                     :save_last_frame:
888: (8)                                     class RightAngleExample(Scene):
889: (12)                                     def construct(self):
890: (16)                                     line1 = Line(LEFT, RIGHT)
891: (16)                                     line2 = Line(DOWN, UP)
892: (16)                                     rightangles = [
893: (20)                                     RightAngle(line1, line2),
894: (20)                                     RightAngle(line1, line2, length=0.4, quadrant=(1,-1)),
895: (20)                                     RightAngle(line1, line2, length=0.5, quadrant=(-1,1),
stroke_width=8),
896: (20)                                     RightAngle(line1, line2, length=0.7, quadrant=(-1,-1),
color=RED),
897: (16)                                     ]
898: (16)                                     plots = VGroup()
899: (16)                                     for rightangle in rightangles:
900: (20)                                     plot=VGroup(line1.copy(),line2.copy(), rightangle)

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
901: (20)                      plots.add(plot)
902: (16)                      plots.arrange(buff=1.5)
903: (16)                      self.add(plots)
904: (4) """
905: (4)     def __init__(self, line1: Line, line2: Line, length: float | None = None, **kwargs)
906: (8)         ) -> None:
907: (4)             super().__init__(line1, line2, radius=length, elbow=True, **kwargs)
-----
```

File 51 - tips.py:

```
1: (0) """A collection of tip mobjects for use with :class:`~.TipableVMobject`."""
2: (0) from __future__ import annotations
3: (0) __all__ = [
4: (4)     "ArrowTip",
5: (4)     "ArrowCircleFilledTip",
6: (4)     "ArrowCircleTip",
7: (4)     "ArrowSquareTip",
8: (4)     "ArrowSquareFilledTip",
9: (4)     "ArrowTriangleTip",
10: (4)     "ArrowTriangleFilledTip",
11: (4)     "StealthTip",
12: (0) ]
13: (0) from typing import TYPE_CHECKING
14: (0) import numpy as np
15: (0) from manim.constants import *
16: (0) from manim.mobject.geometry.arc import Circle
17: (0) from manim.mobject.geometry.polygram import Square, Triangle
18: (0) from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
19: (0) from manim.mobject.types.vectorized_mobject import VMobject
20: (0) from manim.utils.space_ops import angle_of_vector
21: (0) if TYPE_CHECKING:
22: (4)     from manim.typing import Point3D, Vector3D
23: (0) class ArrowTip(VMobject, metaclass=ConvertToOpenGL):
24: (4)     """Base class for arrow tips.
25: (4)     .. seealso::
26: (8)         :class:`~ArrowTriangleTip`
27: (8)         :class:`~ArrowTriangleFilledTip`
28: (8)         :class:`~ArrowCircleTip`
29: (8)         :class:`~ArrowCircleFilledTip`
30: (8)         :class:`~ArrowSquareTip`
31: (8)         :class:`~ArrowSquareFilledTip`
32: (8)         :class:`~StealthTip`
33: (4) Examples
34: (4) -----
35: (4) Cannot be used directly, only intended for inheritance::
36: (8)     >>> tip = ArrowTip()
37: (8)     Traceback (most recent call last):
38: (8)     ...
39: (8)     NotImplementedError: Has to be implemented in inheriting subclasses.
40: (4) Instead, use one of the pre-defined ones, or make
41: (4) a custom one like this:
42: (4) .. manim:: CustomTipExample
43: (8)     >>> from manim import RegularPolygon, Arrow
44: (8)     >>> class MyCustomArrowTip(ArrowTip, RegularPolygon):
45: (8)         ...     def __init__(self, length=0.35, **kwargs):
46: (8)             ...         RegularPolygon.__init__(self, n=5, **kwargs)
47: (8)             ...         self.width = length
48: (8)             ...         self.stretch_to_fit_height(length)
49: (8)     >>> arr = Arrow(np.array([-2, -2, 0]), np.array([2, 2, 0]),
50: (8)                 ...             tip_shape=MyCustomArrowTip)
51: (8)     >>> isinstance(arr.tip, RegularPolygon)
52: (8)     True
53: (8)     >>> from manim import Scene, Create
54: (8)     >>> class CustomTipExample(Scene):
55: (8)         ...     def construct(self):
56: (8)             ...         self.play(Create(arr))
```

```

57: (4)             Using a class inherited from :class:`ArrowTip` to get a non-filled
58: (4)             tip is a shorthand to manually specifying the arrow tip style as follows::
59: (8)             >>> arrow = Arrow(np.array([0, 0, 0]), np.array([1, 1, 0]),
60: (8)             ...                     tip_style={'fill_opacity': 0, 'stroke_width': 3})
61: (4)             The following example illustrates the usage of all of the predefined
62: (4)             arrow tips.
63: (4)             .. manim:: ArrowTipsShowcase
64: (8)                 :save_last_frame:
65: (8)                 class ArrowTipsShowcase(Scene):
66: (12)                   def construct(self):
67: (16)                     tip_names = [
68: (20)                       'Default (YELLOW)', 'ArrowTriangleTip', 'Default',
69: (20)                       'ArrowSquareTip',
70: (16)                   ]
71: (16)                     big_arrows = [
72: (20)                       Arrow(start=[-4, 3.5, 0], end=[2, 3.5, 0], color=YELLOW),
73: (20)                       Arrow(start=[-4, 2.5, 0], end=[2, 2.5, 0],
tip_shape=ArrowTriangleTip),
74: (20)                       Arrow(start=[-4, 1.5, 0], end=[2, 1.5, 0]),
75: (20)                       Arrow(start=[-4, 0.5, 0], end=[2, 0.5, 0],
tip_shape=ArrowSquareTip),
76: (20)                       Arrow([-4, -0.5, 0], [2, -0.5, 0],
tip_shape=ArrowSquareFilledTip),
77: (20)                       Arrow([-4, -1.5, 0], [2, -1.5, 0],
tip_shape=ArrowCircleTip),
78: (20)                       Arrow([-4, -2.5, 0], [2, -2.5, 0],
tip_shape=ArrowCircleFilledTip),
79: (20)                       Arrow([-4, -3.5, 0], [2, -3.5, 0], tip_shape=StealthTip)
80: (16)                   ]
81: (16)                     small_arrows = (
82: (20)                       arrow.copy().scale(0.5, scale_tips=True).next_to(arrows,
RIGHT) for arrow in big_arrows
83: (16)                   )
84: (16)                     labels = (
85: (20)                       Text(tip_names[i], font='monospace', font_size=20,
color=BLUE).next_to(big_arrows[i], LEFT) for i in range(len(big_arrows))
86: (16)                   )
87: (16)                     self.add(*big_arrows, *small_arrows, *labels)
88: (4)             """
89: (4)             def __init__(self, *args, **kwargs) -> None:
90: (8)                 raise NotImplementedError("Has to be implemented in inheriting
subclasses.")
91: (4)             @property
92: (4)                 def base(self) -> Point3D:
93: (8)                     """The base point of the arrow tip.
94: (8)                     This is the point connecting to the arrow line.
95: (8)                     Examples
96: (8)                     -----
97: (8)                     :::
98: (12)                         >>> from manim import Arrow
99: (12)                         >>> arrow = Arrow(np.array([0, 0, 0]), np.array([2, 0, 0]),
buff=0)
100: (12)                         >>> arrow.tip.base.round(2) + 0. # add 0. to avoid negative 0 in
output
101: (12)                         array([1.65, 0. , 0. ])
102: (8)             """
103: (8)                 return self.point_from_proportion(0.5)
104: (4)             @property
105: (4)                 def tip_point(self) -> Point3D:
106: (8)                     """The tip point of the arrow tip.
107: (8)                     Examples
108: (8)                     -----
109: (8)                     :::
110: (12)                         >>> from manim import Arrow
111: (12)                         >>> arrow = Arrow(np.array([0, 0, 0]), np.array([2, 0, 0]),
buff=0)
112: (12)                         >>> arrow.tip.tip_point.round(2) + 0.
```

```

113: (12)           array([2., 0., 0.])
114: (8)           """
115: (8)           return self.points[0]
116: (4)           @property
117: (4)           def vector(self) -> Vector3D:
118: (8)               """The vector pointing from the base point to the tip point.
119: (8)               Examples
120: (8)               -----
121: (8)               :::
122: (12)                   >>> from manim import Arrow
123: (12)                   >>> arrow = Arrow(np.array([0, 0, 0]), np.array([2, 2, 0]),
124: (12)                   buff=0)
125: (12)                   >>> arrow.tip.vector.round(2) + 0.
126: (8)                   array([0.25, 0.25, 0.  ])
127: (8)           """
128: (4)           return self.tip_point - self.base
129: (4)           @property
130: (8)           def tip_angle(self) -> float:
131: (8)               """The angle of the arrow tip.
132: (8)               Examples
133: (8)               -----
134: (12)               :::
135: (12)                   >>> from manim import Arrow
136: (12)                   >>> arrow = Arrow(np.array([0, 0, 0]), np.array([1, 1, 0]),
137: (12)                   buff=0)
138: (8)                   >>> round(arrow.tip.tip_angle, 5) == round(PI/4, 5)
139: (8)                   True
140: (4)           """
141: (4)           return angle_of_vector(self.vector)
142: (8)           @property
143: (8)           def length(self) -> np.floating:
144: (8)               """The length of the arrow tip.
145: (8)               Examples
146: (12)               :::
147: (12)                   >>> from manim import Arrow
148: (12)                   >>> arrow = Arrow(np.array([0, 0, 0]), np.array([1, 2, 0]))
149: (12)                   >>> round(arrow.tip.length, 3)
150: (8)                   0.35
151: (8)           """
152: (0)           return np.linalg.norm(self.vector)
153: (4)           class StealthTip(ArrowTip):
154: (4)               """'Stealth' fighter / kite arrow shape.
155: (4)               Naming is inspired by the corresponding
156: (4)               `TikZ arrow shape <https://tikz.dev/tikz-arrows#sec-16.3>`__.
157: (4)               """
158: (8)           def __init__(
159: (8)               self,
160: (8)               fill_opacity=1,
161: (8)               stroke_width=3,
162: (8)               length=DEFAULT_ARROW_TIP_LENGTH / 2,
163: (8)               start_angle=PI,
164: (4)               **kwargs,
165: (8)           ):
166: (8)               self.start_angle = start_angle
167: (12)               VMobject.__init__(
168: (8)                   self, fill_opacity=fill_opacity, stroke_width=stroke_width,
169: (8)                   **kwargs
170: (12)               )
171: (16)               self.set_points_as_corners(
172: (16)                   [
173: (16)                       [2, 0, 0], # tip
174: (16)                       [-1.2, 1.6, 0],
175: (16)                       [0, 0, 0], # base
176: (16)                       [-1.2, -1.6, 0],
177: (12)                       [2, 0, 0], # close path, back to tip
178: (8)                   ]
178: (8)               )
178: (8)           self.scale(length / self.length)

```

```

179: (4) @property
180: (4)     def length(self):
181: (8)         """The length of the arrow tip.
182: (8)         In this case, the length is computed as the height of
183: (8)         the triangle encompassing the stealth tip (otherwise,
184: (8)         the tip is scaled too large).
185: (8)         """
186: (8)         return np.linalg.norm(self.vector) * 1.6
187: (0) class ArrowTriangleTip(ArrowTip, Triangle):
188: (4)     r"""Triangular arrow tip."""
189: (4)     def __init__(
190: (8)         self,
191: (8)         fill_opacity: float = 0,
192: (8)         stroke_width: float = 3,
193: (8)         length: float = DEFAULT_ARROW_TIP_LENGTH,
194: (8)         width: float = DEFAULT_ARROW_TIP_LENGTH,
195: (8)         start_angle: float = PI,
196: (8)         **kwargs,
197: (4)     ) -> None:
198: (8)         Triangle.__init__(
199: (12)             self,
200: (12)             fill_opacity=fill_opacity,
201: (12)             stroke_width=stroke_width,
202: (12)             start_angle=start_angle,
203: (12)             **kwargs,
204: (8)
205: (8)
206: (8)         self.width = width
207: (8)         self.stretch_to_fit_width(length)
208: (8)         self.stretch_to_fit_height(width)
209: (0) class ArrowTriangleFilledTip(ArrowTriangleTip):
210: (4)     r"""Triangular arrow tip with filled tip.
211: (4)     This is the default arrow tip shape.
212: (4)
213: (8)     def __init__(
214: (4)         self, fill_opacity: float = 1, stroke_width: float = 0, **kwargs
215: (4)     ) -> None:
216: (8)         super().__init__(fill_opacity=fill_opacity, stroke_width=stroke_width,
217: (0)         **kwargs)
218: (4)         class ArrowCircleTip(ArrowTip, Circle):
219: (4)             r"""Circular arrow tip."""
220: (8)             def __init__(
221: (8)                 self,
222: (8)                 fill_opacity: float = 0,
223: (8)                 stroke_width: float = 3,
224: (8)                 length: float = DEFAULT_ARROW_TIP_LENGTH,
225: (8)                 start_angle: float = PI,
226: (8)                 **kwargs,
227: (8)             ) -> None:
228: (12)                 self.start_angle = start_angle
229: (8)                 Circle.__init__(
230: (8)                     self, fill_opacity=fill_opacity, stroke_width=stroke_width,
231: (8)                     )
232: (0)                 self.width = length
233: (8)                 self.stretch_to_fit_height(length)
234: (0)         class ArrowCircleFilledTip(ArrowCircleTip):
235: (4)             r"""Circular arrow tip with filled tip."""
236: (4)             def __init__(
237: (8)                 self, fill_opacity: float = 1, stroke_width: float = 0, **kwargs
238: (0)             ) -> None:
239: (4)                 super().__init__(fill_opacity=fill_opacity, stroke_width=stroke_width,
240: (4)                 **kwargs)
241: (8)         class ArrowSquareTip(ArrowTip, Square):
242: (4)             r"""Square arrow tip."""
243: (4)             def __init__(
244: (8)                 self,

```

```

245: (8)                 start_angle: float = PI,
246: (8)                 **kwargs,
247: (4)             ) -> None:
248: (8)                 self.start_angle = start_angle
249: (8)                 Square.__init__(
250: (12)                     self,
251: (12)                     fill_opacity=fill_opacity,
252: (12)                     stroke_width=stroke_width,
253: (12)                     side_length=length,
254: (12)                     **kwargs,
255: (8)                 )
256: (8)                 self.width = length
257: (8)                 self.stretch_to_fit_height(length)
258: (0)             class ArrowSquareFilledTip(ArrowSquareTip):
259: (4)                 """Square arrow tip with filled tip."""
260: (4)             def __init__(
261: (8)                 self, fill_opacity: float = 1, stroke_width: float = 0, **kwargs
262: (4)             ) -> None:
263: (8)                 super().__init__(fill_opacity=fill_opacity, stroke_width=stroke_width,
**kwargs)
-----
```

## File 52 - frame.py:

```

1: (0)                 """Special rectangles."""
2: (0)                 from __future__ import annotations
3: (0)             __all__ = [
4: (4)                     "ScreenRectangle",
5: (4)                     "FullScreenRectangle",
6: (0)
7: (0)                 from manim.mobject.geometry.polygram import Rectangle
8: (0)                 from .. import config
9: (0)             class ScreenRectangle(Rectangle):
10: (4)                 def __init__(self, aspect_ratio=16.0 / 9.0, height=4, **kwargs):
11: (8)                     super().__init__(width=aspect_ratio * height, height=height, **kwargs)
12: (4)             @property
13: (4)                 def aspect_ratio(self):
14: (8)                     """The aspect ratio.
15: (8)                     When set, the width is stretched to accommodate
16: (8)                     the new aspect ratio.
17: (8)                     """
18: (8)                     return self.width / self.height
19: (4)             @aspect_ratio.setter
20: (4)                 def aspect_ratio(self, value):
21: (8)                     self.stretch_to_fit_width(value * self.height)
22: (0)             class FullScreenRectangle(ScreenRectangle):
23: (4)                 def __init__(self, **kwargs):
24: (8)                     super().__init__(**kwargs)
25: (8)                     self.height = config["frame_height"]
-----
```

## File 53 - graph.py:

```

1: (0)                 """Mobjects used to represent mathematical graphs (think graph theory, not
plotting)."""
2: (0)                 from __future__ import annotations
3: (0)             __all__ = [
4: (4)                     "Graph",
5: (4)                     "DiGraph",
6: (0)
7: (0)                     ]
8: (0)                 import itertools as it
9: (0)                 from copy import copy
9: (0)                 from typing import TYPE_CHECKING, Any, Hashable, Iterable, Literal, Protocol,
cast
10: (0)                 import networkx as nx
11: (0)                 import numpy as np
12: (0)                 if TYPE_CHECKING:
-----
```

```

13: (4)             from typing_extensions import TypeAlias
14: (4)             from manim.typing import Point3D
15: (4)             NxGraph: TypeAlias = nx.classes.graph.Graph | nx.classes.digraph.DiGraph
16: (0)             from manim.animation.composition import AnimationGroup
17: (0)             from manim.animation.creation import Create, Uncreate
18: (0)             from manim.mobject.geometry.arc import Dot, LabeledDot
19: (0)             from manim.mobject.geometry.line import Line
20: (0)             from manim.mobject.mobject import Mobject, override_animate
21: (0)             from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
22: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject
23: (0)             from manim.mobject.text.tex_mobject import MathTex
24: (0)             from manim.mobject.types.vectorized_mobject import VMobject
25: (0)             from manim.utils.color import BLACK
26: (0)         class LayoutFunction(Protocol):
27: (4)             """A protocol for automatic layout functions that compute a layout for a
graph to be used in :meth:`~.Graph.change_layout` .
28: (4)             .. note:: The layout function must be a pure function, i.e., it must not
modify the graph passed to it.
29: (4)             Examples
30: (4)             -----
31: (4)             Here is an example that arranges nodes in an  $n \times m$  grid in sorted order.
32: (4)             .. manim:: CustomLayoutExample
33: (8)             :save_last_frame:
34: (8)             class CustomLayoutExample(Scene):
35: (12)             def construct(self):
36: (16)                 import numpy as np
37: (16)                 import networkx as nx
38: (16)                 def custom_layout(
39: (20)                     graph: nx.Graph,
40: (20)                     scale: float | tuple[float, float, float] = 2,
41: (20)                     n: int | None = None,
42: (20)                     *args: Any,
43: (20)                     **kwargs: Any,
44: (16)                 ):
45: (20)                     nodes = sorted(list(graph))
46: (20)                     height = len(nodes) // n
47: (20)                     return {
48: (24)                         node: (scale * np.array([
49: (28)                             (i % n) - (n-1)/2,
50: (28)                             -(i // n) + height/2,
51: (28)                             0
52: (24)                         ])) for i, node in enumerate(graph)
53: (20)                     }
54: (16)                     n = 4
55: (16)                     graph = Graph(
56: (20)                         [i for i in range(4 * 2 - 1)],
57: (20)                         [(0, 1), (0, 4), (1, 2), (1, 5), (2, 3), (2, 6), (4, 5),
58: (20)                         (5, 6)],
59: (20)                         labels=True,
60: (20)                         layout=custom_layout,
61: (16)                         layout_config={'n': n}
62: (16)                     )
63: (4)                     self.add(graph)
64: (4)             Several automatic layouts are provided by manim, and can be used by
passing their name as the ``layout`` parameter to :meth:`~.Graph.change_layout` .
65: (4)             Alternatively, a custom layout function can be passed to
:meth:`~.Graph.change_layout` as the ``layout`` parameter. Such a function must adhere to the
:class:`~.LayoutFunction` protocol.
66: (4)             The :class:`~.LayoutFunction` s provided by manim are illustrated below:
67: (4)             - Circular Layout: places the vertices on a circle
68: (8)             .. manim:: CircularLayout
69: (8)             :save_last_frame:
70: (12)             class CircularLayout(Scene):
71: (16)             def construct(self):
72: (20)                 graph = Graph(
73: (20)                     [1, 2, 3, 4, 5, 6],
74: (20)                     [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
74: (20)                     layout="circular",

```

```

75: (20)                                labels=True
76: (16)                                )
77: (16)                                self.add(graph)
78: (4) - Kamada Kawai Layout: tries to place the vertices such that the given
distances between them are respected
79: (4) .. manim:: KamadaKawaiLayout
80: (8) :save_last_frame:
81: (8) class KamadaKawaiLayout(Scene):
82: (12) def construct(self):
83: (16)     from collections import defaultdict
84: (16)     distances: dict[int, dict[int, float]] = defaultdict(dict)
85: (16)     distances[1][2] = 1 # distance between vertices 1 and 2 is 1
86: (16)     distances[2][3] = 1 # distance between vertices 2 and 3 is 1
87: (16)     distances[3][4] = 2 # etc
88: (16)     distances[4][5] = 3
89: (16)     distances[5][6] = 5
90: (16)     distances[6][1] = 8
91: (16)     graph = Graph(
92: (20)         [1, 2, 3, 4, 5, 6],
93: (20)         [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1)],
94: (20)         layout="kamada_kawai",
95: (20)         layout_config={"dist": distances},
96: (20)         layout_scale=4,
97: (20)         labels=True
98: (16)     )
99: (16)     self.add(graph)
100: (4) - Partite Layout: places vertices into distinct partitions
101: (4) .. manim:: PartiteLayout
102: (8) :save_last_frame:
103: (8) class PartiteLayout(Scene):
104: (12) def construct(self):
105: (16)     graph = Graph(
106: (20)         [1, 2, 3, 4, 5, 6],
107: (20)         [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
108: (20)         layout="partite",
109: (20)         layout_config={"partitions": [[1,2],[3,4],[5,6]]},
110: (20)         labels=True
111: (16)     )
112: (16)     self.add(graph)
113: (4) - Planar Layout: places vertices such that edges do not cross
114: (4) .. manim:: PlanarLayout
115: (8) :save_last_frame:
116: (8) class PlanarLayout(Scene):
117: (12) def construct(self):
118: (16)     graph = Graph(
119: (20)         [1, 2, 3, 4, 5, 6],
120: (20)         [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
121: (20)         layout="planar",
122: (20)         layout_scale=4,
123: (20)         labels=True
124: (16)     )
125: (16)     self.add(graph)
126: (4) - Random Layout: randomly places vertices
127: (4) .. manim:: RandomLayout
128: (8) :save_last_frame:
129: (8) class RandomLayout(Scene):
130: (12) def construct(self):
131: (16)     graph = Graph(
132: (20)         [1, 2, 3, 4, 5, 6],
133: (20)         [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
134: (20)         layout="random",
135: (20)         labels=True
136: (16)     )
137: (16)     self.add(graph)
138: (4) - Shell Layout: places vertices in concentric circles
139: (4) .. manim:: ShellLayout

```

```

140: (8)             :save_last_frame:
141: (8)             class ShellLayout(Scene):
142: (12)            def construct(self):
143: (16)            nlist = [[1, 2, 3], [4, 5, 6, 7, 8, 9]]
144: (16)            graph = Graph(
145: (20)              [1, 2, 3, 4, 5, 6, 7, 8, 9],
146: (20)              [(1, 2), (2, 3), (3, 1), (4, 1), (4, 2), (5, 2), (6, 2),
(6, 3), (7, 3), (8, 3), (8, 1), (9, 1)],
147: (20)              layout="shell",
148: (20)              layout_config={"nlist": nlist},
149: (20)              labels=True
150: (16)
151: (16)            self.add(graph)
152: (4)             - Spectral Layout: places vertices using the eigenvectors of the graph
Laplacian (clusters nodes which are an approximation of the ratio cut)
153: (4)             .. manim:: SpectralLayout
154: (8)             :save_last_frame:
155: (8)             class SpectralLayout(Scene):
156: (12)            def construct(self):
157: (16)            graph = Graph(
158: (20)              [1, 2, 3, 4, 5, 6],
159: (20)              [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
160: (20)              layout="spectral",
161: (20)              labels=True
162: (16)
163: (16)            self.add(graph)
164: (4)             - Sprial Layout: places vertices in a spiraling pattern
165: (4)             .. manim:: SpiralLayout
166: (8)             :save_last_frame:
167: (8)             class SpiralLayout(Scene):
168: (12)            def construct(self):
169: (16)            graph = Graph(
170: (20)              [1, 2, 3, 4, 5, 6],
171: (20)              [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
172: (20)              layout="spiral",
173: (20)              labels=True
174: (16)
175: (16)            self.add(graph)
176: (4)             - Spring Layout: places nodes according to the Fruchterman-Reingold force-
directed algorithm (attempts to minimize edge length while maximizing node separation)
177: (4)             .. manim:: SpringLayout
178: (8)             :save_last_frame:
179: (8)             class SpringLayout(Scene):
180: (12)            def construct(self):
181: (16)            graph = Graph(
182: (20)              [1, 2, 3, 4, 5, 6],
183: (20)              [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 1),
(1, 3), (3, 5)],
184: (20)              layout="spring",
185: (20)              labels=True
186: (16)
187: (16)            self.add(graph)
188: (4)             - Tree Layout: places vertices into a tree with a root node and branches
(can only be used with legal trees)
189: (4)             .. manim:: TreeLayout
190: (8)             :save_last_frame:
191: (8)             class TreeLayout(Scene):
192: (12)            def construct(self):
193: (16)            graph = Graph(
194: (20)              [1, 2, 3, 4, 5, 6, 7],
195: (20)              [(1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (3, 7)],
196: (20)              layout="tree",
197: (20)              layout_config={"root_vertex": 1},
198: (20)              labels=True
199: (16)
200: (16)            self.add(graph)
201: (4)             """

```

```

202: (4)             def __call__(self,
203: (8)                 graph: NxGraph,
204: (8)                 scale: float | tuple[float, float, float] = 2,
205: (8)                 *args: Any,
206: (8)                 **kwargs: Any,
207: (8)             ) -> dict[Hashable, Point3D]:
208: (4)                 """Given a graph and a scale, return a dictionary of coordinates.
209: (8)             Parameters
210: (8)             -----
211: (8)                 graph : NxGraph
212: (8)                     The underlying NetworkX graph to be laid out. DO NOT MODIFY.
213: (12)                 scale : float | tuple[float, float, float], optional
214: (8)                         Either a single float value, or a tuple of three float values
215: (12)                         specifying the scale along each axis.
216: (8)             Returns
217: (8)             -----
218: (8)                 dict[Hashable, Point3D]
219: (12)                 A dictionary mapping vertices to their positions.
220: (8)             """
221: (8)             ...
222: (0)             def _partite_layout(
223: (4)                 nx_graph: NxGraph,
224: (4)                 scale: float = 2,
225: (4)                 partitions: list[list[Hashable]] | None = None,
226: (4)                 **kwargs: Any,
227: (0)             ) -> dict[Hashable, Point3D]:
228: (4)                 if partitions is None or len(partitions) == 0:
229: (8)                     raise ValueError(
230: (12)                         "The partite layout requires partitions parameter to contain the
partition of the vertices",
231: (8)                     )
232: (4)                 partition_count = len(partitions)
233: (4)                 for i in range(partition_count):
234: (8)                     for v in partitions[i]:
235: (12)                     if nx_graph.nodes[v] is None:
236: (16)                         raise ValueError(
237: (20)                             "The partition must contain arrays of vertices in the
graph",
238: (16)                         )
239: (12)                     nx_graph.nodes[v]["subset"] = i
240: (4)                     for v in nx_graph.nodes:
241: (8)                         if "subset" not in nx_graph.nodes[v]:
242: (12)                             nx_graph.nodes[v]["subset"] = partition_count
243: (4)                     return nx.layout.multipartite_layout(nx_graph, scale=scale, **kwargs)
244: (0)             def _random_layout(nx_graph: NxGraph, scale: float = 2, **kwargs: Any):
245: (4)                 auto_layout = nx.layout.random_layout(nx_graph, **kwargs)
246: (4)                 for k, v in auto_layout.items():
247: (8)                     auto_layout[k] = 2 * scale * (v - np.array([0.5, 0.5]))
248: (4)                 return {k: np.append(v, [0]) for k, v in auto_layout.items()}
249: (0)             def _tree_layout(
250: (4)                 T: NxGraph,
251: (4)                 root_vertex: Hashable | None = None,
252: (4)                 scale: float | tuple | None = 2,
253: (4)                 vertex_spacing: tuple | None = None,
254: (4)                 orientation: str = "down",
255: (0)             ):
256: (4)                 if root_vertex is None:
257: (8)                     raise ValueError("The tree layout requires the root_vertex parameter")
258: (4)                 if not nx.is_tree(T):
259: (8)                     raise ValueError("The tree layout must be used with trees")
260: (4)                 children = {root_vertex: list(T.neighbors(root_vertex))}
261: (4)                 stack = [list(children[root_vertex]).copy()]
262: (4)                 stick = [root_vertex]
263: (4)                 parent = {u: root_vertex for u in children[root_vertex]}
264: (4)                 pos = {}
265: (4)                 obstruction = [0.0] * len(T)
266: (4)                 if orientation == "down":
267: (8)                     o = -1

```

```

268: (4)
269: (8)         else:
270: (4)             o = 1
271: (8)             def slide(v, dx):
272: (8)                 """
273: (8)                     Shift the vertex v and its descendants to the right by dx.
274: (8)                     Precondition: v and its descendants have already had their
275: (8)                     positions computed.
276: (8)                 """
277: (8)                 level = [v]
278: (12)                 while level:
279: (12)                     nextlevel = []
280: (16)                     for u in level:
281: (16)                         x, y = pos[u]
282: (16)                         x += dx
283: (16)                         obstruction[y] = max(x + 1, obstruction[y])
284: (16)                         pos[u] = x, y
285: (16)                         nextlevel += children[u]
286: (12)                     level = nextlevel
287: (4)                 while stack:
288: (8)                     C = stack[-1]
289: (8)                     if not C:
290: (12)                         p = stick.pop()
291: (12)                         stack.pop()
292: (12)                         cp = children[p]
293: (12)                         y = o * len(stack)
294: (16)                         if not cp:
295: (16)                             x = obstruction[y]
296: (12)                             pos[p] = x, y
297: (16)                         else:
298: (16)                             x = sum(pos[c][0] for c in cp) / float(len(cp))
299: (16)                             pos[p] = x, y
300: (16)                             ox = obstruction[y]
301: (20)                             if x < ox:
302: (20)                                 slide(p, ox - x)
303: (12)                                 x = ox
304: (12)                                 obstruction[y] = x + 1
305: (8)                                 continue
306: (8)                             t = C.pop()
307: (8)                             pt = parent[t]
308: (8)                             ct = [u for u in list(T.neighbors(t)) if u != pt]
309: (12)                             for c in ct:
310: (8)                                 parent[c] = t
311: (8)                                 children[t] = copy(ct)
312: (8)                                 stack.append(ct)
313: (8)                                 stick.append(t)
314: (4)                                 x_min = min(pos.values(), key=lambda t: t[0])[0]
315: (4)                                 x_max = max(pos.values(), key=lambda t: t[0])[0]
316: (4)                                 y_min = min(pos.values(), key=lambda t: t[1])[1]
317: (4)                                 y_max = max(pos.values(), key=lambda t: t[1])[1]
318: (4)                                 center = np.array([x_min + x_max, y_min + y_max, 0]) / 2
319: (4)                                 height = y_max - y_min
320: (4)                                 width = x_max - x_min
321: (8)                                 if vertex_spacing is None:
322: (12)                                     if isinstance(scale, (float, int)) and (width > 0 or height > 0):
323: (8)   sf = 2 * scale / max(width, height)
324: (12)                                     elif isinstance(scale, tuple):
325: (16)   if scale[0] is not None and width > 0:
326: (12)   sw = 2 * scale[0] / width
327: (16)   else:
328: (12)   sw = 1
329: (16)   if scale[1] is not None and height > 0:
330: (12)   sh = 2 * scale[1] / height
331: (16)   else:
332: (12)   sh = 1
333: (8)   sf = np.array([sw, sh, 0])
334: (12)   else:
335: (4)   sf = 1
336: (8)                                 else:
337: (8)                                     sx, sy = vertex_spacing

```

```

337: (8)                 sf = np.array([sx, sy, 0])
338: (4)                 return {v: (np.array([x, y, 0]) - center) * sf for v, (x, y) in
pos.items()}
339: (0)                 LayoutName = Literal[
340: (4)                   "circular",
341: (4)                   "kamada_kawai",
342: (4)                   "partite",
343: (4)                   "planar",
344: (4)                   "random",
345: (4)                   "shell",
346: (4)                   "spectral",
347: (4)                   "spiral",
348: (4)                   "spring",
349: (4)                   "tree",
350: (0)
351: (0)                 _layouts: dict[LayoutName, LayoutFunction] = {
352: (4)                   "circular": cast(LayoutFunction, nx.layout.circular_layout),
353: (4)                   "kamada_kawai": cast(LayoutFunction, nx.layout.kamada_kawai_layout),
354: (4)                   "partite": cast(LayoutFunction, _partite_layout),
355: (4)                   "planar": cast(LayoutFunction, nx.layout.planar_layout),
356: (4)                   "random": cast(LayoutFunction, _random_layout),
357: (4)                   "shell": cast(LayoutFunction, nx.layout.shell_layout),
358: (4)                   "spectral": cast(LayoutFunction, nx.layout.spectral_layout),
359: (4)                   "spiral": cast(LayoutFunction, nx.layout.spiral_layout),
360: (4)                   "spring": cast(LayoutFunction, nx.layout.spring_layout),
361: (4)                   "tree": cast(LayoutFunction, _tree_layout),
362: (0)
363: (0)                 def _determine_graph_layout(
364: (4)                   nx_graph: nx.classes.graph.Graph | nx.classes.digraph.DiGraph,
365: (4)                   layout: LayoutName | dict[Hashable, Point3D] | LayoutFunction = "spring",
366: (4)                   layout_scale: float | tuple[float, float, float] = 2,
367: (4)                   layout_config: dict[str, Any] | None = None,
368: (0)                 ) -> dict[Hashable, Point3D]:
369: (4)                     if layout_config is None:
370: (8)                         layout_config = {}
371: (4)                     if isinstance(layout, dict):
372: (8)                         return layout
373: (4)                     elif layout in _layouts:
374: (8)                         auto_layout = _layouts[layout](nx_graph, scale=layout_scale,
**layout_config)
375: (8)
376: (12)                    if (
377: (12)                      layout_config.get("dim") == 3
378: (8)                      or auto_layout[next(auto_layout.__iter__())].shape[0] == 3
379: (12)
380: (8)
381: (12)                      ):
382: (4)                          return auto_layout
383: (8)
384: (12)                    else:
385: (16)                        return {k: np.append(v, [0]) for k, v in auto_layout.items()}
386: (12)
387: (8)
388: (12)                    except TypeError as e:
389: (16)                        raise ValueError(
f"The layout '{layout}' is neither a recognized layout, a
layout function,"
390: (16)                            "nor a vertex placement dictionary.",
)
391: (12)
392: (0)                 class GenericGraph(VMobject, metaclass=ConvertToOpenGL):
393: (4)                     """Abstract base class for graphs (that is, a collection of vertices
394: (4)                     connected with edges).
395: (4)                     Graphs can be instantiated by passing both a list of (distinct, hashable)
396: (4)                     vertex names, together with list of edges (as tuples of vertex names). See
397: (4)                     the examples for concrete implementations of this class for details.
398: (4)                     .. note::
399: (8)                         This implementation uses updaters to make the edges move with
400: (8)                         the vertices.
401: (4)                     See also
402: (4)                         -----

```

```

403: (4)           :class:`.Graph`
404: (4)           :class:`.DiGraph`
405: (4)           Parameters
406: (4)           -----
407: (4)           vertices
408: (8)           A list of vertices. Must be hashable elements.
409: (4)           edges
410: (8)           A list of edges, specified as tuples `` $(u, v)$ `` where both `` $u$ `` and `` $v$ `` are vertices.
411: (8)
412: (4)           labels
413: (8)           Controls whether or not vertices are labeled. If ``False`` (the default),
414: (8)           the vertices are not labeled; if ``True`` they are labeled using their names (as specified in ``vertices``) via :class:`~.MathTex`.
415: (8)
416: (4)           Alternatively,
417: (8)
418: (8)           custom labels can be specified by passing a dictionary whose keys are the vertices, and whose values are the corresponding vertex labels (rendered via, e.g., :class:`~.Text` or :class:`~.Tex`).
419: (4)           label_fill_color
420: (8)           Sets the fill color of the default labels generated when ``labels`` is set to ``True``. Has no effect for other values of ``labels``.
421: (8)
422: (4)           layout
423: (8)           Either one of ``"spring"`` (the default), ``"circular"``,
424: (8)           ``"planar"`` , ``"random"`` , ``"shell"`` , ``"spectral"`` , ``"spiral"`` ,
425: (8)           ``"tree"`` , and ``"partite"`` for automatic vertex positioning primarily using ``networkx`` (see `their documentation <https://networkx.org/documentation/stable/reference/drawing.html#module-networkx.drawing.layout>`_ for more details), a dictionary specifying a coordinate (value) for each vertex (key) for manual positioning, or a :class:`~.LayoutFunction` with a user-defined automatic layout.
426: (8)
427: (8)
428: (8)
429: (4)           layout_config
430: (8)           Only for automatic layouts. A dictionary whose entries are passed as keyword arguments to the named layout or automatic
431: (8)
432: (8)           layout function
433: (8)           specified via ``layout``.
434: (8)           The ``tree`` layout also accepts a special parameter
435: (8)           ``vertex_spacing`` passed as a keyword argument inside the ``layout_config`` dictionary.
436: (8)           Passing a tuple `` $(space_x, space_y)$ `` as this argument overrides the value of ``layout_scale`` and ensures that vertices are arranged in a way such that the centers of siblings in the same layer are at least `` $space_x$ `` units apart horizontally, and neighboring layers are spaced `` $space_y$ `` units vertically.
437: (8)
438: (8)
439: (8)
440: (4)           layout_scale
441: (8)           The scale of automatically generated layouts: the vertices will be arranged such that the coordinates are located within the interval `` $[-scale, scale]$ ``. Some layouts accept a tuple `` $(scale_x, scale_y)$ `` causing the first coordinate to be in the interval `` $[-scale_x, scale_x]$ `` and the second in `` $[-scale_y, scale_y]$ ``. Default: 2.
442: (8)
443: (8)
444: (8)
445: (8)
446: (4)           vertex_type
447: (8)           The mobject class used for displaying vertices in the scene.
448: (4)           vertex_config
449: (8)           Either a dictionary containing keyword arguments to be passed to the class specified via ``vertex_type``, or a dictionary whose keys are the vertices, and whose values are dictionaries containing keyword arguments for the mobject related to the corresponding vertex.
450: (8)
451: (8)
452: (8)
453: (4)           vertex_mobjects
454: (8)           A dictionary whose keys are the vertices, and whose values are mobjects to be used as vertices. Passing vertices here overrides all other configuration options for a vertex.
455: (8)
456: (8)
457: (4)           edge_type
458: (8)           The mobject class used for displaying edges in the scene.
459: (4)           edge_config
460: (8)           Either a dictionary containing keyword arguments to be passed

```

```

461: (8)          to the class specified via ``edge_type``, or a dictionary whose
462: (8)          keys are the edges, and whose values are dictionaries containing
463: (8)          keyword arguments for the mobject related to the corresponding edge.
464: (4)
465: (4)
466: (8)
467: (8)
468: (8)
469: (8)
470: (8)
471: (8)
472: "spring",
473: (8)
474: (8)
475: (8)
476: (8)
477: (8)
478: (8)
479: (8)
480: (8)
481: (4)
482: (8)
483: (8)
484: (8)
485: (8)
486: (8)
487: (8)
488: (12)
489: (8)
490: (12)
491: (16)
492: (20)
vertices
493: (16)
494: (12)
495: (16)
496: (8)
497: (12)
498: (8)
499: (12)
500: (8)
501: (12)
502: (8)
503: (8)
504: (12)
505: (16)
506: (12)
507: (8)
508: (12)
vertices
509: (8)
510: (8)
511: (8)
512: (12)
513: (8)
vertices}
514: (8)
515: (8)
516: (12)
517: (12)
518: (12)
519: (12)
520: (12)
521: (8)
522: (8)
523: (12)
524: (8)
525: (8)
    """
    def __init__(
        self,
        vertices: list[Hashable],
        edges: list[tuple[Hashable, Hashable]],
        labels: bool | dict = False,
        label_fill_color: str = BLACK,
        layout: LayoutName | dict[Hashable, Point3D] | LayoutFunction =
"spring",
        layout_scale: float | tuple[float, float, float] = 2,
        layout_config: dict | None = None,
        vertex_type: type[Mobject] = Dot,
        vertex_config: dict | None = None,
        vertex_mobjects: dict | None = None,
        edge_type: type[Mobject] = Line,
        partitions: list[list[Hashable]] | None = None,
        root_vertex: Hashable | None = None,
        edge_config: dict | None = None,
    ) -> None:
        super().__init__()
        nx_graph = self._empty_networkx_graph()
        nx_graph.add_nodes_from(vertices)
        nx_graph.add_edges_from(edges)
        self._graph = nx_graph
        if isinstance(labels, dict):
            self._labels = labels
        elif isinstance(labels, bool):
            if labels:
                self._labels = {
                    v: MathTex(v, fill_color=label_fill_color) for v in
                    vertices
                }
            else:
                self._labels = {}
        if self._labels and vertex_type is Dot:
            vertex_type = LabeledDot
        if vertex_mobjects is None:
            vertex_mobjects = {}
        if vertex_config is None:
            vertex_config = {}
        default_vertex_config = {}
        if vertex_config:
            default_vertex_config = {
                k: v for k, v in vertex_config.items() if k not in vertices
            }
        self._vertex_config = {
            v: vertex_config.get(v, copy(default_vertex_config)) for v in
            vertices
        }
        self.default_vertex_config = default_vertex_config
        for v, label in self._labels.items():
            self._vertex_config[v]["label"] = label
        self.vertices = {v: vertex_type(**self._vertex_config[v]) for v in
            vertices}
        self.vertices.update(vertex_mobjects)
        self.change_layout(
            layout=layout,
            layout_scale=layout_scale,
            layout_config=layout_config,
            partitions=partitions,
            root_vertex=root_vertex,
        )
        if edge_config is None:
            edge_config = {}
        default_tip_config = {}
        default_edge_config = {}

```

```

526: (8)             if edge_config:
527: (12)             default_tip_config = edge_config.pop("tip_config", {})
528: (12)             default_edge_config = {
529: (16)                 k: v
530: (16)                 for k, v in edge_config.items()
531: (16)                 if not isinstance(
532: (20)                     k, tuple
533: (16)                 ) # everything that is not an edge is an option
534: (12)             }
535: (8)             self._edge_config = {}
536: (8)             self._tip_config = {}
537: (8)             for e in edges:
538: (12)                 if e in edge_config:
539: (16)                     self._tip_config[e] = edge_config[e].pop(
540: (20)                         "tip_config", copy(default_tip_config)
541: (16)                     )
542: (16)                     self._edge_config[e] = edge_config[e]
543: (12)                 else:
544: (16)                     self._tip_config[e] = copy(default_tip_config)
545: (16)                     self._edge_config[e] = copy(default_edge_config)
546: (8)             self.default_edge_config = default_edge_config
547: (8)             self._populate_edge_dict(edges, edge_type)
548: (8)             self.add(*self.vertices.values())
549: (8)             self.add(*self.edges.values())
550: (8)             self.add_updater(self.update_edges)
551: (4)             @staticmethod
552: (4)             def _empty_networkx_graph() -> nx.classes.graph.Graph:
553: (8)                 """Return an empty networkx graph for the given graph type."""
554: (8)                 raise NotImplementedError("To be implemented in concrete subclasses")
555: (4)             def __populate_edge_dict(
556: (8)                 self, edges: list[tuple[Hashable, Hashable]], edge_type: type[Mobject]
557: (4)             ):
558: (8)                 """Helper method for populating the edges of the graph."""
559: (8)                 raise NotImplementedError("To be implemented in concrete subclasses")
560: (4)             def __getitem__(self: Graph, v: Hashable) -> Mobject:
561: (8)                 return self.vertices[v]
562: (4)             def __create_vertex(
563: (8)                 self,
564: (8)                 vertex: Hashable,
565: (8)                 position: Point3D | None = None,
566: (8)                 label: bool = False,
567: (8)                 label_fill_color: str = BLACK,
568: (8)                 vertex_type: type[Mobject] = Dot,
569: (8)                 vertex_config: dict | None = None,
570: (8)                 vertex_mobject: dict | None = None,
571: (4)             ) -> tuple[Hashable, Point3D, dict, Mobject]:
572: (8)                 if position is None:
573: (12)                     position = self.get_center()
574: (8)                 if vertex_config is None:
575: (12)                     vertex_config = {}
576: (8)                 if vertex in self.vertices:
577: (12)                     raise ValueError(
578: (16)                         f"Vertex identifier '{vertex}' is already used for a vertex in
this graph.",
579: (12)                     )
580: (8)                 if label is True:
581: (12)                     label = MathTex(vertex, fill_color=label_fill_color)
582: (8)                 elif vertex in self._labels:
583: (12)                     label = self._labels[vertex]
584: (8)                 elif not isinstance(label, (Mobject, OpenGLMobject)):
585: (12)                     label = None
586: (8)                 base_vertex_config = copy(self.default_vertex_config)
587: (8)                 base_vertex_config.update(vertex_config)
588: (8)                 vertex_config = base_vertex_config
589: (8)                 if label is not None:
590: (12)                     vertex_config["label"] = label
591: (12)                     if vertex_type is Dot:
592: (16)                         vertex_type = LabeledDot
593: (8)                     if vertex_mobject is None:

```

```

594: (12)           vertex_mobject = vertex_type(**vertex_config)
595: (8)             vertex_mobject.move_to(position)
596: (8)             return (vertex, position, vertex_config, vertex_mobject)
597: (4)         def _add_created_vertex(
598: (8)             self,
599: (8)             vertex: Hashable,
600: (8)             position: Point3D,
601: (8)             vertex_config: dict,
602: (8)             vertex_mobject: Mobject,
603: (4)         ) -> Mobject:
604: (8)             if vertex in self.vertices:
605: (12)                 raise ValueError(
606: (16)                     f"Vertex identifier '{vertex}' is already used for a vertex in
this graph.",
607: (12)                     )
608: (8)             self._graph.add_node(vertex)
609: (8)             self._layout[vertex] = position
610: (8)             if "label" in vertex_config:
611: (12)                 self._labels[vertex] = vertex_config["label"]
612: (8)                 self._vertex_config[vertex] = vertex_config
613: (8)                 self.vertices[vertex] = vertex_mobject
614: (8)                 self.vertices[vertex].move_to(position)
615: (8)                 self.add(self.vertices[vertex])
616: (8)             return self.vertices[vertex]
617: (4)         def _add_vertex(
618: (8)             self,
619: (8)             vertex: Hashable,
620: (8)             position: Point3D | None = None,
621: (8)             label: bool = False,
622: (8)             label_fill_color: str = BLACK,
623: (8)             vertex_type: type[Mobject] = Dot,
624: (8)             vertex_config: dict | None = None,
625: (8)             vertex_mobject: dict | None = None,
626: (4)         ) -> Mobject:
627: (8)             """Add a vertex to the graph.
Parameters
-----
vertex
A hashable vertex identifier.
position
The coordinates where the new vertex should be added. If ``None``,
of the graph is used.
label
Controls whether or not the vertex is labeled. If ``False`` (the
the vertex is not labeled; if ``True`` it is labeled using its
names (as specified in ``vertex``) via :class:`~.MathTex`.
any :class:`~.Mobject` can be passed to be used as the label.
label_fill_color
Sets the fill color of the default labels generated when
is set to ``True``. Has no effect for other values of ``label``.
vertex_type
The mobject class used for displaying vertices in the scene.
vertex_config
A dictionary containing keyword arguments to be passed to
the class specified via ``vertex_type``.
vertex_mobject
The mobject to be used as the vertex. Overrides all other
vertex customization options.
"""
628: (8)
629: (8)
630: (8)
631: (12)
632: (8)
633: (12)
the center
634: (12)
635: (8)
636: (12)
default),
637: (12)
638: (12)
Alternatively,
639: (12)
640: (8)
641: (12)
``labels``
642: (12)
643: (8)
644: (12)
645: (8)
646: (12)
647: (12)
648: (8)
649: (12)
650: (12)
651: (8)
652: (8)
653: (12)
654: (16)
655: (16)
656: (16)
657: (16)
return self._add_created_vertex(
    *self._create_vertex(
        vertex=vertex,
        position=position,
        label=label,
        label_fill_color=label_fill_color,

```

```

658: (16)                     vertex_type=vertex_type,
659: (16)                     vertex_config=vertex_config,
660: (16)                     vertex_mobject=vertex_mobject,
661: (12)                 )
662: (8)             )
663: (4)         def _create_vertices(
664: (8)             self: Graph,
665: (8)             *vertices: Hashable,
666: (8)             positions: dict | None = None,
667: (8)             labels: bool = False,
668: (8)             label_fill_color: str = BLACK,
669: (8)             vertex_type: type[Mobject] = Dot,
670: (8)             vertex_config: dict | None = None,
671: (8)             vertex_mobjects: dict | None = None,
672: (4)         ) -> Iterable[tuple[Hashable, Point3D, dict, Mobject]]:
673: (8)             if positions is None:
674: (12)                 positions = {}
675: (8)             if vertex_mobjects is None:
676: (12)                 vertex_mobjects = {}
677: (8)             graph_center = self.get_center()
678: (8)             base_positions = {v: graph_center for v in vertices}
679: (8)             base_positions.update(positions)
680: (8)             positions = base_positions
681: (8)             if isinstance(labels, bool):
682: (12)                 labels = {v: labels for v in vertices}
683: (8)             else:
684: (12)                 assert isinstance(labels, dict)
685: (12)                 base_labels = {v: False for v in vertices}
686: (12)                 base_labels.update(labels)
687: (12)                 labels = base_labels
688: (8)             if vertex_config is None:
689: (12)                 vertex_config = copy(self.default_vertex_config)
690: (8)             assert isinstance(vertex_config, dict)
691: (8)             base_vertex_config = copy(self.default_vertex_config)
692: (8)             base_vertex_config.update(
693: (12)                 {key: val for key, val in vertex_config.items() if key not in
694: (8)                     vertices},
695: (8)             )
696: (12)             vertex_config = {
697: (12)                 v: (vertex_config[v] if v in vertex_config else
698: (8)                     for v in vertices
699: (8)                 )
700: (12)             return [
701: (16)                 self._create_vertex(
702: (16)                     v,
703: (16)                     position=positions[v],
704: (16)                     label=labels[v],
705: (16)                     label_fill_color=label_fill_color,
706: (16)                     vertex_type=vertex_type,
707: (16)                     vertex_config=vertex_config[v],
708: (12)                     vertex_mobject=vertex_mobjects[v] if v in vertex_mobjects else
709: (12)                         )
710: (8)                     for v in vertices
711: (4)                 ]
712: (8)             def add_vertices(
713: (8)                 self: Graph,
714: (8)                 *vertices: Hashable,
715: (8)                 positions: dict | None = None,
716: (8)                 labels: bool = False,
717: (8)                 label_fill_color: str = BLACK,
718: (8)                 vertex_type: type[Mobject] = Dot,
719: (8)                 vertex_config: dict | None = None,
720: (4)                 vertex_mobjects: dict | None = None,
721: (8)             ):
722: (8)                 """Add a list of vertices to the graph.
723: (8)                 Parameters
724: (8)                 -----

```

```

724: (8)           vertices
725: (12)          Hashable vertex identifiers.
726: (8)           positions
727: (12)          A dictionary specifying the coordinates where the new vertices
should be added.
728: (12)          If ``None``, all vertices are created at the center of the graph.
729: (8)           labels
730: (12)          Controls whether or not the vertex is labeled. If ``False`` (the
default),
731: (12)          the vertex is not labeled; if ``True`` it is labeled using its
732: (12)          names (as specified in ``vertex``) via :class:`~.MathTex`.
Alternatively,
733: (12)          any :class:`~.Mobject` can be passed to be used as the label.
734: (8)           label_fill_color
735: (12)          Sets the fill color of the default labels generated when
``labels``
736: (12)          is set to ``True``. Has no effect for other values of ``labels``.
737: (8)           vertex_type
738: (12)          The mobject class used for displaying vertices in the scene.
739: (8)           vertex_config
740: (12)          A dictionary containing keyword arguments to be passed to
741: (12)          the class specified via ``vertex_type``.
742: (8)           vertex_mobjects
743: (12)          A dictionary whose keys are the vertex identifiers, and whose
744: (12)          values are mobjects that should be used as vertices. Overrides
745: (12)          all other vertex customization options.
746: (8)           """
747: (8)           return [
748: (12)             self._add_created_vertex(*v)
749: (12)             for v in self._create_vertices(
750: (16)               *vertices,
751: (16)               positions=positions,
752: (16)               labels=labels,
753: (16)               label_fill_color=label_fill_color,
754: (16)               vertex_type=vertex_type,
755: (16)               vertex_config=vertex_config,
756: (16)               vertex_mobjects=vertex_mobjects,
757: (12)             )
758: (8)           ]
759: (4)           @override_animate(add_vertices)
760: (4)           def _add_vertices_animation(self, *args, anim_args=None, **kwargs):
761: (8)             if anim_args is None:
762: (12)               anim_args = {}
763: (8)             animation = anim_args.pop("animation", Create)
764: (8)             vertex_mobjects = self._create_vertices(*args, **kwargs)
765: (8)             def on_finish(scene: Scene):
766: (12)               for v in vertex_mobjects:
767: (16)                 scene.remove(v[-1])
768: (16)                 self._add_created_vertex(*v)
769: (8)             return AnimationGroup(
770: (12)               *(animation(v[-1], **anim_args) for v in vertex_mobjects),
771: (12)               group=self,
772: (12)               _on_finish=on_finish,
773: (8)             )
774: (4)           def _remove_vertex(self, vertex):
775: (8)             """Remove a vertex (as well as all incident edges) from the graph.
776: (8)             Parameters
777: (8)             -----
778: (8)             vertex
779: (12)               The identifier of a vertex to be removed.
780: (8)             Returns
781: (8)             -----
782: (8)             Group
783: (12)               A mobject containing all removed objects.
784: (8)             """
785: (8)             if vertex not in self.vertices:
786: (12)               raise ValueError(
787: (16)                 f"The graph does not contain a vertex with identifier
'{vertex}'",

```

```

788: (12) )
789: (8)     self._graph.remove_node(vertex)
790: (8)     self._layout.pop(vertex)
791: (8)     if vertex in self._labels:
792: (12)         self._labels.pop(vertex)
793: (8)     self._vertex_config.pop(vertex)
794: (8)     edge_tuples = [e for e in self.edges if vertex in e]
795: (8)     for e in edge_tuples:
796: (12)         self._edge_config.pop(e)
797: (8)     to_remove = [self.edges.pop(e) for e in edge_tuples]
798: (8)     to_remove.append(self.vertices.pop(vertex))
799: (8)     self.remove(*to_remove)
800: (8)     return self.get_group_class()(*to_remove)
801: (4) def remove_vertices(self, *vertices):
802: (8)     """Remove several vertices from the graph.
803: (8)     Parameters
804: (8)     -----
805: (8)     vertices
806: (12)         Vertices to be removed from the graph.
807: (8)     Examples
808: (8)     -----
809: (8)     :::
810: (12)         >>> G = Graph([1, 2, 3], [(1, 2), (2, 3)])
811: (12)         >>> removed = G.remove_vertices(2, 3); removed
812: (12)             VGroup(Line, Line, Dot, Dot)
813: (12)         >>> G
814: (12)             Undirected graph on 1 vertices and 0 edges
815: (8)     """
816: (8)     mobjects = []
817: (8)     for v in vertices:
818: (12)         mobjects.extend(self._remove_vertex(v).submobjects)
819: (8)     return self.get_group_class()(*mobjects)
820: (4) @override_animate(remove_vertices)
821: (4) def _remove_vertices_animation(self, *vertices, anim_args=None):
822: (8)     if anim_args is None:
823: (12)         anim_args = {}
824: (8)     animation = anim_args.pop("animation", Uncreate)
825: (8)     mobjects = self.remove_vertices(*vertices)
826: (8)     return AnimationGroup(
827: (12)         *(animation(mobj, **anim_args) for mobj in mobjects), group=self
828: (8)     )
829: (4) def _add_edge(
830: (8)     self,
831: (8)     edge: tuple[Hashable, Hashable],
832: (8)     edge_type: type[Mobject] = Line,
833: (8)     edge_config: dict | None = None,
834: (4) ):
835: (8)     """Add a new edge to the graph.
836: (8)     Parameters
837: (8)     -----
838: (8)     edge
839: (12)         The edge (as a tuple of vertex identifiers) to be added. If a non-
840: (12)         existing
841: (12)         vertex is passed, a new vertex with default settings will be
842: (8)         new vertices yourself beforehand to customize them.
843: (12)         edge_type
844: (8)             The mobject class used for displaying edges in the scene.
845: (12)         edge_config
846: (12)             A dictionary containing keyword arguments to be passed
847: (8)                 to the class specified via ``edge_type``.
848: (8)         Returns
849: (8)         -----
850: (12)         Group
851: (8)             A group containing all newly added vertices and edges.
852: (8)         """
853: (12)         if edge_config is None:
854: (8)             edge_config = self.default_edge_config.copy()

```

```

855: (8)             for v in edge:
856: (12)             if v not in self.vertices:
857: (16)                 added_mobjects.append(self._add_vertex(v))
858: (8)
859: (8)             u, v = edge
860: (8)             self._graph.add_edge(u, v)
861: (8)             base_edge_config = self.default_edge_config.copy()
862: (8)             base_edge_config.update(edge_config)
863: (8)             edge_config = base_edge_config
864: (8)             self._edge_config[(u, v)] = edge_config
865: (12)             edge_mobject = edge_type(
866: (8)                         self[u].get_center(), self[v].get_center(), z_index=-1,
867: (8)                         )
868: (8)             self.edges[(u, v)] = edge_mobject
869: (8)             self.add(edge_mobject)
870: (8)             added_mobjects.append(edge_mobject)
871: (4)             return self.get_group_class()(*added_mobjects)
872: (8)
873: (8)
874: (8)         def add_edges(
875: (8)             self,
876: (8)             *edges: tuple[Hashable, Hashable],
877: (4)             edge_type: type[Mobject] = Line,
878: (8)             edge_config: dict | None = None,
879: (8)             **kwargs,
880: (8)             ):
881: (8)             """Add new edges to the graph.
882: (12)             Parameters
883: (12)             -----
884: (12)             edges
885: (8)                 Edges (as tuples of vertex identifiers) to be added. If a non-
886: (12)             existing
887: (12)             vertex is passed, a new vertex with default settings will be
888: (12)             new vertices yourself beforehand to customize them.
889: (12)             edge_type
890: (12)                 The mobject class used for displaying edges in the scene.
891: (12)             edge_config
892: (12)                 A dictionary either containing keyword arguments to be passed
893: (8)                 to the class specified via ``edge_type``, or a dictionary
894: (12)                 whose keys are the edge tuples, and whose values are dictionaries
895: (12)                 containing keyword arguments to be passed for the construction
896: (8)                 of the corresponding edge.
897: (8)             kwargs
898: (8)                 Any further keyword arguments are passed to :meth:`.add_vertices`
899: (12)                 which is used to create new vertices in the passed edges.
900: (8)
901: (8)
902: (12)             Returns
903: (8)
904: (8)             -----
905: (8)             Group
906: (8)                 A group containing all newly added vertices and edges.
907: (8)
908: (12)             """
909: (8)             if edge_config is None:
910: (8)                 edge_config = {}
911: (8)             non_edge_settings = {k: v for (k, v) in edge_config.items() if k not
912: (8)             in edges}
913: (8)
914: (12)
915: (16)
916: (20)
917: (20)
918: (20)
919: (16)

```

```

920: (16)                         for edge in edges
921: (12)                     ),
922: (12)                     added_vertices,
923: (8)
924: (8)
925: (4)                     )
926: (4) @override_animate(add_edges)
927: (8)                     def _add_edges_animation(self, *args, anim_args=None, **kwargs):
928: (12)                         if anim_args is None:
929: (8)                             anim_args = {}
930: (8)                         animation = anim_args.pop("animation", Create)
931: (8)                         mobjects = self.add_edges(*args, **kwargs)
932: (12)                         return AnimationGroup(
933: (8)                             *(animation(mobj, **anim_args) for mobj in mobjects), group=self
934: (4)                         )
935: (8)                     def _remove_edge(self, edge: tuple[Hashable]):
936: (8)                         """Remove an edge from the graph.
937: (8)                         Parameters
938: (8)                         -----
939: (12)                         edge
940: (8)                             The edge (i.e., a tuple of vertex identifiers) to be removed from
941: (8)                             the graph.
942: (8)
943: (12)                         Returns
944: (8)                         -----
945: (8)                         Mobject
946: (12)                         The removed edge.
947: (8)
948: (8)
949: (8)
950: (8)
951: (8)
952: (4)                     def remove_edges(self, *edges: tuple[Hashable]):
953: (8)                         """Remove several edges from the graph.
954: (8)
955: (8)
956: (8)
957: (12)                         Parameters
958: (8)                         -----
959: (8)                         edges
960: (8)                             Edges to be removed from the graph.
961: (12)                         Returns
962: (8)                         -----
963: (8)                         Group
964: (8)                             A group containing all removed edges.
965: (4)
966: (4) @override_animate(remove_edges)
967: (8)                     def _remove_edges_animation(self, *edges, anim_args=None):
968: (12)                         if anim_args is None:
969: (8)                             anim_args = {}
970: (8)                         animation = anim_args.pop("animation", Uncreate)
971: (8)                         mobjects = self.remove_edges(*edges)
972: (4)                         return AnimationGroup(*(animation(mobj, **anim_args) for mobj in
mobjects))
973: (4)
974: (8) @classmethod
975: (4)                     def from_networkx(
976: (8)                         cls, nxgraph: nx.classes.graph.Graph | nx.classes.digraph.DiGraph,
977: (8)                         **kwargs
978: (8)                     ):
979: (8)                         """Build a :class:`~.Graph` or :class:`~.DiGraph` from a
980: (8)                         given ``networkx`` graph.
981: (12)                         Parameters
982: (8)                         -----
983: (12)                         nxgraph
984: (8)                             A ``networkx`` graph or digraph.
985: (8)                         **kwargs
986: (8)                             Keywords to be passed to the constructor of :class:`~.Graph`.
987: (8)                         Examples
988: (8)                         -----

```

```

986: (8)          .. manim:: ImportNetworkxGraph
987: (12)         import networkx as nx
988: (12)         nxgraph = nx.erdos_renyi_graph(14, 0.5)
989: (12)         class ImportNetworkxGraph(Scene):
990: (16)           def construct(self):
991: (20)             G = Graph.from_networkx(nxgraph, layout="spring",
layout_scale=3.5)
992: (20)             self.play(Create(G))
993: (20)             self.play(*[G[v].animate.move_to(5*RIGHT*np.cos(ind/7 *
PI) +
994: (53)               3*UP*np.sin(ind/7 * PI))
995: (32)                 for ind, v in enumerate(G.vertices)])
996: (20)             self.play(Uncreate(G))
997: (8)           """
998: (8)             return cls(list(nxgraph.nodes), list(nxgraph.edges), **kwargs)
999: (4)         def change_layout(
1000: (8)           self,
1001: (8)             layout: LayoutName | dict[Hashable, Point3D] | LayoutFunction =
1002: (8)               layout_scale: float | tuple[float, float, float] = 2,
1003: (8)               layout_config: dict[str, Any] | None = None,
1004: (8)               partitions: list[list[Hashable]] | None = None,
1005: (8)               root_vertex: Hashable | None = None,
1006: (4)         ) -> Graph:
1007: (8)           """Change the layout of this graph.
1008: (8)           See the documentation of :class:`~.Graph` for details about the
1009: (8)           keyword arguments.
1010: (8)           Examples
1011: (8)           -----
1012: (8)             .. manim:: ChangeGraphLayout
1013: (12)             class ChangeGraphLayout(Scene):
1014: (16)               def construct(self):
1015: (20)                 G = Graph([1, 2, 3, 4, 5], [(1, 2), (2, 3), (3, 4), (4,
5)],
1016: (30)                   layout={1: [-2, 0, 0], 2: [-1, 0, 0], 3: [0, 0,
0],
1017: (38)                     4: [1, 0, 0], 5: [2, 0, 0]}
1018: (30)               )
1019: (20)               self.play(Create(G))
1020: (20)               self.play(G.animate.change_layout("circular"))
1021: (20)               self.wait()
1022: (8)             """
1023: (8)             layout_config = {} if layout_config is None else layout_config
1024: (8)             if partitions is not None and "partitions" not in layout_config:
1025: (12)               layout_config["partitions"] = partitions
1026: (8)             if root_vertex is not None and "root_vertex" not in layout_config:
1027: (12)               layout_config["root_vertex"] = root_vertex
1028: (8)             self._layout = _determine_graph_layout(
1029: (12)               self._graph,
1030: (12)               layout=layout,
1031: (12)               layout_scale=layout_scale,
1032: (12)               layout_config=layout_config,
1033: (8)             )
1034: (8)             for v in self.vertices:
1035: (12)               self[v].move_to(self._layout[v])
1036: (8)             return self
1037: (0)         class Graph(GenericGraph):
1038: (4)           """An undirected graph (vertices connected with edges).
1039: (4)           The graph comes with an updater which makes the edges stick to
1040: (4)           the vertices when moved around. See :class:`.DiGraph` for
1041: (4)           a version with directed edges.
1042: (4)           See also
1043: (4)           -----
1044: (4)             :class:`.GenericGraph`
1045: (4)           Parameters
1046: (4)           -----
1047: (4)             vertices
1048: (8)               A list of vertices. Must be hashable elements.
1049: (4)             edges

```

```

1050: (8)           A list of edges, specified as tuples ``(u, v)`` where both ``u``
1051: (8)           and ``v`` are vertices. The vertex order is irrelevant.
1052: (4)
1053: (8)           labels
1054: (8)           Controls whether or not vertices are labeled. If ``False`` (the
1055: (8)           default),
1056: (8)           the vertices are not labeled; if ``True`` they are labeled using their
1057: (8)           names (as specified in ``vertices``) via :class:`~.MathTex`.
1058: (8)           Alternatively,
1059: (4)           custom labels can be specified by passing a dictionary whose keys are
1060: (8)           the vertices, and whose values are the corresponding vertex labels
1061: (8)           (rendered via, e.g., :class:`~.Text` or :class:`~.Tex`).
1062: (4)           label_fill_color
1063: (8)           Sets the fill color of the default labels generated when ``labels``
1064: (8)           is set to ``True``. Has no effect for other values of ``labels``.
1065: (8)           layout
1066: (8)           Either one of ``"spring"`` (the default), ``"circular"``,
1067: (8)           ``"kamada_kawai"``,
1068: (8)           ``"planar"`` , ``"random"`` , ``"shell"`` , ``"spectral"`` , ``"spiral"`` ,
1069: (4)           ``"tree"`` , and ``"partite"`` .
1070: (8)           for automatic vertex positioning using ``networkx``
1071: (8)           (see `their documentation
1072: (8)           <https://networkx.org/documentation/stable/reference/drawing.html#module-networkx.drawing.layout>_
1073: (8)           for more details), or a dictionary specifying a coordinate (value)
1074: (8)           for each vertex (key) for manual positioning.
1075: (8)           layout_config
1076: (8)           Only for automatically generated layouts. A dictionary whose entries
1077: (8)           are passed as keyword arguments to the automatic layout algorithm
1078: (8)           specified via ``layout`` of ``networkx``.
1079: (8)           The ``tree`` layout also accepts a special parameter
1080: (4)           ``vertex_spacing``
1081: (8)           passed as a keyword argument inside the ``layout_config`` dictionary.
1082: (8)           Passing a tuple ``(space_x, space_y)`` as this argument overrides
1083: (8)           the value of ``layout_scale`` and ensures that vertices are arranged
1084: (8)           in a way such that the centers of siblings in the same layer are
1085: (8)           at least ``space_x`` units apart horizontally, and neighboring layers
1086: (4)           are spaced ``space_y`` units vertically.
1087: (8)           layout_scale
1088: (8)           The scale of automatically generated layouts: the vertices will
1089: (8)           be arranged such that the coordinates are located within the
1090: (8)           interval ``[-scale, scale]``. Some layouts accept a tuple ``(scale_x,
1091: (8)           scale_y)``,
1092: (8)           causing the first coordinate to be in the interval ``[-scale_x,
1093: (4)           and the second in ``[-scale_y, scale_y]``. Default: 2.
1094: (8)           vertex_type
1095: (8)           The mobject class used for displaying vertices in the scene.
1096: (8)           vertex_config
1097: (4)           Either a dictionary containing keyword arguments to be passed to
1098: (8)           the class specified via ``vertex_type`` , or a dictionary whose keys
1099: (4)           are the vertices, and whose values are dictionaries containing keyword
1100: (8)           arguments for the mobject related to the corresponding vertex.
1101: (8)           vertex_mobjects
1102: (8)           A dictionary whose keys are the vertices, and whose values are
1103: (8)           mobjects to be used as vertices. Passing vertices here overrides
1104: (4)           all other configuration options for a vertex.
1105: (4)           edge_type
1106: (4)           The mobject class used for displaying edges in the scene.
1107: (4)           edge_config
1108: (4)           Either a dictionary containing keyword arguments to be passed
1109: (8)           to the class specified via ``edge_type`` , or a dictionary whose
1109: (8)           keys are the edges, and whose values are dictionaries containing
1109: (8)           keyword arguments for the mobject related to the corresponding edge.
1109: (8)           Examples
1109: (8)           -----
1109: (8)           First, we create a small graph and demonstrate that the edges move
1109: (8)           together with the vertices.
1109: (8)           .. manim:: MovingVertices
1109: (8)               class MovingVertices(Scene):

```

```

1110: (12)
1111: (16)
1112: (16)
1113: (16)
1114: (16)
1115: (16)
1116: (16)
1117: (26)
1118: (26)
1119: (26)
1120: (16)
1121: (4)
1122: (4)
1123: (8)
1124: (8)
1125: (12)
1126: (16)
1127: (16)
1128: (25)
1129: (25)
1130: (16)
1131: (31)
1132: (31)
1133: (16)
1134: (26)
1135: (16)
1136: (16)
1137: (16)
1138: (16)
1139: (4)
1140: (4)
1141: (8)
1142: (8)
1143: (12)
1144: (16)
1145: (16)
1146: (16)
0]}
1147: (16)
1148: (16)
1149: (4)
1150: (4)
1151: (4)
1152: (4)
1153: (8)
1154: (8)
1155: (8)
1156: (4)
1157: (8)
1158: (8)
1159: (12)
1160: (16)
1161: (16)
1162: (25)
1163: (25)
1164: (16)
1165: (26)
1166: (26)
1167: (39)
1168: (39)
1169: (16)
1170: (4)
1171: (4)
1172: (4)
1173: (8)
partitions
1174: (8)
column.
1175: (4)

def construct(self):
    vertices = [1, 2, 3, 4]
    edges = [(1, 2), (2, 3), (3, 4), (1, 3), (1, 4)]
    g = Graph(vertices, edges)
    self.play(Create(g))
    self.wait()
    self.play(g[1].animate.move_to([1, 1, 0]),
              g[2].animate.move_to([-1, 1, 0]),
              g[3].animate.move_to([1, -1, 0]),
              g[4].animate.move_to([-1, -1, 0]))
    self.wait()

There are several automatic positioning algorithms to choose from:
.. manim:: GraphAutoPosition
    :save_last_frame:
    class GraphAutoPosition(Scene):
        def construct(self):
            vertices = [1, 2, 3, 4, 5, 6, 7, 8]
            edges = [(1, 7), (1, 8), (2, 3), (2, 4), (2, 5),
                      (2, 8), (3, 4), (6, 1), (6, 2),
                      (6, 3), (7, 2), (7, 4)]
            autolayouts = ["spring", "circular", "kamada_kawai",
                           "planar", "random", "shell",
                           "spectral", "spiral"]
            graphs = [Graph(vertices, edges, layout=lt).scale(0.5)
                      for lt in autolayouts]
            r1 = VGroup(*graphs[:3]).arrange()
            r2 = VGroup(*graphs[3:6]).arrange()
            r3 = VGroup(*graphs[6:]).arrange()
            self.add(VGroup(r1, r2, r3).arrange(direction=DOWN))

Vertices can also be positioned manually:
.. manim:: GraphManualPosition
    :save_last_frame:
    class GraphManualPosition(Scene):
        def construct(self):
            vertices = [1, 2, 3, 4]
            edges = [(1, 2), (2, 3), (3, 4), (4, 1)]
            lt = {1: [0, 0, 0], 2: [1, 1, 0], 3: [1, -1, 0], 4: [-1, 0,
0]}
            G = Graph(vertices, edges, layout=lt)
            self.add(G)

The vertices in graphs can be labeled, and configurations for vertices
and edges can be modified both by default and for specific vertices and
edges.
.. note::
    In ``edge_config``, edges can be passed in both directions: if
    ````(u, v)```` is an edge in the graph, both ````(u, v)```` as well
    as ````(v, u)```` can be used as keys in the dictionary.
.. manim:: LabeledModifiedGraph
    :save_last_frame:
    class LabeledModifiedGraph(Scene):
        def construct(self):
            vertices = [1, 2, 3, 4, 5, 6, 7, 8]
            edges = [(1, 7), (1, 8), (2, 3), (2, 4), (2, 5),
                      (2, 8), (3, 4), (6, 1), (6, 2),
                      (6, 3), (7, 2), (7, 4)]
            g = Graph(vertices, edges, layout="circular", layout_scale=3,
                      labels=True, vertex_config={7: {"fill_color": RED}},
                      edge_config={(1, 7): {"stroke_color": RED},
                                   (2, 7): {"stroke_color": RED},
                                   (4, 7): {"stroke_color": RED}})

            self.add(g)

You can also lay out a partite graph on columns by specifying
a list of the vertices on each side and choosing the partite layout.
.. note::
    All vertices in your graph which are not listed in any of the
    partitions
        are collected in their own partition and rendered in the rightmost
.. manim:: PartiteGraph

```

```

1176: (8)           :save_last_frame:
1177: (8)           import networkx as nx
1178: (8)           class PartiteGraph(Scene):
1179: (12)             def construct(self):
1180: (16)               G = nx.Graph()
1181: (16)               G.add_nodes_from([0, 1, 2, 3])
1182: (16)               G.add_edges_from([(0, 2), (0,3), (1, 2)])
1183: (16)               graph = Graph(list(G.nodes), list(G.edges), layout="partite",
partitions=[[0, 1]])
1184: (16)               self.play(Create(graph))
1185: (4)           The representation of a linear artificial neural network is facilitated
1186: (4)           by the use of the partite layout and defining partitions for each layer.
1187: (4)           .. manim:: LinearNN
1188: (8)           :save_last_frame:
1189: (8)           class LinearNN(Scene):
1190: (12)             def construct(self):
1191: (16)               edges = []
1192: (16)               partitions = []
1193: (16)               c = 0
1194: (16)               layers = [2, 3, 3, 2] # the number of neurons in each layer
1195: (16)               for i in layers:
1196: (20)                 partitions.append(list(range(c + 1, c + i + 1)))
1197: (20)                 c += i
1198: (16)               for i, v in enumerate(layers[1:]):
1199: (24)                 last = sum(layers[:i+1])
1200: (24)                 for j in range(v):
1201: (28)                   for k in range(last - layers[i], last):
1202: (32)                     edges.append((k + 1, j + last + 1))
vertices = np.arange(1, sum(layers) + 1)
graph = Graph(
    vertices,
    edges,
    layout='partite',
    partitions=partitions,
    layout_scale=3,
    vertex_config={'radius': 0.20},
)
self.add(graph)
1213: (4)           The custom tree layout can be used to show the graph
1214: (4)           by distance from the root vertex. You must pass the root vertex
1215: (4)           of the tree.
.. manim:: Tree
1217: (8)           import networkx as nx
1218: (8)           class Tree(Scene):
1219: (12)             def construct(self):
1220: (16)               G = nx.Graph()
1221: (16)               G.add_node("ROOT")
1222: (16)               for i in range(5):
1223: (20)                 G.add_node("Child_%i" % i)
1224: (20)                 G.add_node("Grandchild_%i" % i)
1225: (20)                 G.add_node("Greatgrandchild_%i" % i)
1226: (20)                 G.add_edge("ROOT", "Child_%i" % i)
1227: (20)                 G.add_edge("Child_%i" % i, "Grandchild_%i" % i)
1228: (20)                 G.add_edge("Grandchild_%i" % i, "Greatgrandchild_%i" % i)
1229: (16)               self.play(Create(
1230: (20)                 Graph(list(G.nodes), list(G.edges), layout="tree",
root_vertex="ROOT")))
1231: (4)           The following code sample illustrates the use of the ``vertex_spacing```
1232: (4)           layout parameter specific to the ``"tree"`` layout. As mentioned
1233: (4)           above, setting ``vertex_spacing`` overrides the specified value
1234: (4)           for ``layout_scale``, and as such it is harder to control the size
1235: (4)           of the mobject. However, we can adjust the captured frame and
1236: (4)           zoom out by using a :class:`.MovingCameraScene`::
1237: (8)           class LargeTreeGeneration(MovingCameraScene):
1238: (12)             DEPTH = 4
1239: (12)             CHILDREN_PER_VERTEX = 3
1240: (12)             LAYOUT_CONFIG = {"vertex_spacing": (0.5, 1)}
1241: (12)             VERTEX_CONF = {"radius": 0.25, "color": BLUE_B, "fill_opacity": 1}
1242: (12)             def expand_vertex(self, g, vertex_id: str, depth: int):

```

```

1243: (16)             new_vertices = [f"{{vertex_id}}/{i}" for i in
range(self.CHILDREN_PER_VERTEX)]
1244: (16)             new_edges = [(vertex_id, child_id) for child_id in
new_vertices]
1245: (16)             g.add_edges(
1246: (20)                 *new_edges,
1247: (20)                 vertex_config=self.VERTEX_CONF,
1248: (20)                 positions={
1249: (24)                     k: g.vertices[vertex_id].get_center() + 0.1 * DOWN for
k in new_vertices
1250: (20)                 },
1251: (16)
1252: (16)             )
1253: (20)             if depth < self.DEPTH:
1254: (24)                 for child_id in new_vertices:
1255: (20)                     self.expand_vertex(g, child_id, depth + 1)
1256: (16)             return g
1257: (12)         def construct(self):
1258: (16)             g = Graph(["ROOT"], [], vertex_config=self.VERTEX_CONF)
1259: (16)             g = self.expand_vertex(g, "ROOT", 1)
1260: (16)             self.add(g)
1261: (16)             self.play(
1262: (20)                 g.animate.change_layout(
1263: (24)                     "tree",
1264: (24)                     root_vertex="ROOT",
1265: (20)                     layout_config=self.LAYOUT_CONFIG,
1266: (16)                 )
1267: (16)             self.play(self.camera.auto_zoom(g, margin=1), run_time=0.5)
1268: (4)
1269: (4)             """
1270: (4)             @staticmethod
1271: (8)             def _empty_networkx_graph() -> nx.Graph:
1272: (4)                 return nx.Graph()
1273: (8)             def _populate_edge_dict(
1274: (4)                 self, edges: list[tuple[Hashable, Hashable]], edge_type: type[Mobject]
1275: (8)             ):
1276: (12)                 self.edges = {
1277: (16)                     (u, v): edge_type(
1278: (16)                         self[u].get_center(),
1279: (16)                         self[v].get_center(),
1280: (16)                         z_index=-1,
1281: (12)                         **self._edge_config[(u, v)],
1282: (12)                     )
1283: (8)                     for (u, v) in edges
1284: (8)                 }
1285: (4)             def update_edges(self, graph):
1286: (8)                 for (u, v), edge in graph.edges.items():
1287: (12)                     edge.put_start_and_end_on(graph[u].get_center(),
graph[v].get_center())
1288: (4)             def __repr__(self: Graph) -> str:
1289: (0)                 return f"Undirected graph on {len(self.vertices)} vertices and
{len(self.edges)} edges"
1290: (4)             class DiGraph(GenericGraph):
1291: (4)                 """A directed graph.
1292: (8)                 .. note::
1293: (8)                     In contrast to undirected graphs, the order in which vertices in a
given
1294: (8)                     edge are specified is relevant here.
1295: (4)             See also
1296: (4)             -----
1297: (4)                 :class:`.GenericGraph`
1298: (4)             Parameters
1299: (4)             -----
1300: (8)                 vertices
1301: (4)                     A list of vertices. Must be hashable elements.
edges
1302: (8)                     A list of edges, specified as tuples ``(u, v)`` where both ``u``
1303: (8)                     and ``v`` are vertices. The edge is directed from ``u`` to ``v``.
1304: (4)             labels
1305: (8)                     Controls whether or not vertices are labeled. If ``False`` (the

```

```

default),
1306: (8)          the vertices are not labeled; if ``True`` they are labeled using their
1307: (8)          names (as specified in ``vertices``) via :class:`~.MathTex`.
Alternatively,
1308: (8)          custom labels can be specified by passing a dictionary whose keys are
1309: (8)          the vertices, and whose values are the corresponding vertex labels
1310: (8)          (rendered via, e.g., :class:`~.Text` or :class:`~.Tex`).
1311: (4)          label_fill_color
1312: (8)          Sets the fill color of the default labels generated when ``labels``
1313: (8)          is set to ``True``. Has no effect for other values of ``labels``.
1314: (4)          layout
1315: (8)          Either one of ``"spring"`` (the default), ``"circular"``,
``"kamada_kawai"``,
1316: (8)          ``"planar"`` , ``"random"`` , ``"shell"`` , ``"spectral"`` , ``"spiral"`` ,
``"tree"`` , and ``"partite"``.
1317: (8)          for automatic vertex positioning using ``networkx``
1318: (8)          (see `their documentation
<https://networkx.org/documentation/stable/reference/drawing.html#module-networkx.drawing.layout>`_
1319: (8)          for more details), or a dictionary specifying a coordinate (value)
1320: (8)          for each vertex (key) for manual positioning.
layout_config
1321: (4)          Only for automatically generated layouts. A dictionary whose entries
1322: (8)          are passed as keyword arguments to the automatic layout algorithm
1323: (8)          specified via ``layout`` of ``networkx``.
1324: (8)          The ``tree`` layout also accepts a special parameter
1325: (8)          ``vertex_spacing``.
1326: (8)          passed as a keyword argument inside the ``layout_config`` dictionary.
1327: (8)          Passing a tuple ``space_x, space_y`` as this argument overrides
1328: (8)          the value of ``layout_scale`` and ensures that vertices are arranged
1329: (8)          in a way such that the centers of siblings in the same layer are
1330: (8)          at least ``space_x`` units apart horizontally, and neighboring layers
1331: (8)          are spaced ``space_y`` units vertically.
layout_scale
1332: (4)          The scale of automatically generated layouts: the vertices will
1333: (8)          be arranged such that the coordinates are located within the
1334: (8)          interval ``[-scale, scale]``. Some layouts accept a tuple ``(scale_x,
scale_y)``,
1335: (8)          causing the first coordinate to be in the interval ``[-scale_x,
1336: (8)          scale_x]``,
1337: (8)          and the second in ``[-scale_y, scale_y]``. Default: 2.
vertex_type
1338: (4)          The mobject class used for displaying vertices in the scene.
vertex_config
1339: (8)          Either a dictionary containing keyword arguments to be passed to
1340: (4)          the class specified via ``vertex_type`` , or a dictionary whose keys
1341: (8)          are the vertices, and whose values are dictionaries containing keyword
1342: (8)          arguments for the mobject related to the corresponding vertex.
1343: (8)
1344: (8)
1345: (4)          vertex_mobjects
1346: (8)          A dictionary whose keys are the vertices, and whose values are
1347: (8)          mobjects to be used as vertices. Passing vertices here overrides
1348: (8)          all other configuration options for a vertex.
edge_type
1349: (4)          The mobject class used for displaying edges in the scene.
edge_config
1350: (8)          Either a dictionary containing keyword arguments to be passed
1351: (4)          to the class specified via ``edge_type`` , or a dictionary whose
1352: (8)          keys are the edges, and whose values are dictionaries containing
1353: (8)          keyword arguments for the mobject related to the corresponding edge.
1354: (8)          You can further customize the tip by adding a ``tip_config`` .
1355: (8)
1356: (8)
dictionary
1357: (8)          for global styling, or by adding the dict to a specific
``edge_config``.
1358: (4)          Examples
1359: (4)          -----
1360: (4)          .. manim:: MovingDiGraph
1361: (8)          class MovingDiGraph(Scene):
1362: (12)          def construct(self):
1363: (16)          vertices = [1, 2, 3, 4]

```

```

1364: (16)             edges = [(1, 2), (2, 3), (3, 4), (1, 3), (1, 4)]
1365: (16)             g = DiGraph(vertices, edges)
1366: (16)             self.add(g)
1367: (16)             self.play(
1368: (20)                 g[1].animate.move_to([1, 1, 1]),
1369: (20)                 g[2].animate.move_to([-1, 1, 2]),
1370: (20)                 g[3].animate.move_to([1, -1, -1]),
1371: (20)                 g[4].animate.move_to([-1, -1, 0]),
1372: (16)
1373: (16)             )
1374: (4)             self.wait()
1375: (4)             You can customize the edges and arrow tips globally or locally.
1376: (8)             .. manim:: CustomDiGraph
1377: (12)             class CustomDiGraph(Scene):
1378: (16)                 def construct(self):
1379: (16)                     vertices = [i for i in range(5)]
1380: (20)                     edges = [
1381: (20)                         (0, 1),
1382: (20)                         (1, 2),
1383: (20)                         (3, 2),
1384: (16)                         (3, 4),
1385: (16)
1386: (20)                     edge_config = {
1387: (20)                         "stroke_width": 2,
1388: (24)                         "tip_config": {
1389: (24)                             "tip_shape": ArrowSquareTip,
1390: (20)                             "tip_length": 0.15,
1391: (20)
1392: (24)                         },
1393: (24)                         (3, 4): {
1394: (20)                             "color": RED,
1395: (16)                             "tip_config": {"tip_length": 0.25, "tip_width": 0.25}
1396: (16)
1397: (20)                     g = DiGraph(
1398: (20)                         vertices,
1399: (20)                         edges,
1400: (20)                         labels=True,
1401: (20)                         layout="circular",
1402: (16)                         edge_config=edge_config,
1403: (16)                         ).scale(1.4)
1404: (16)                         self.play(Create(g))
1405: (4)                         self.wait()
1406: (4)             Since this implementation respects the labels boundary you can also use
1407: (4)             it for an undirected moving graph with labels.
1408: (8)             .. manim:: UndirectedMovingDiGraph
1409: (12)             class UndirectedMovingDiGraph(Scene):
1410: (16)                 def construct(self):
1411: (16)                     vertices = [i for i in range(5)]
1412: (20)                     edges = [
1413: (20)                         (0, 1),
1414: (20)                         (1, 2),
1415: (20)                         (3, 2),
1416: (16)                         (3, 4),
1417: (16)
1418: (20)                     edge_config = {
1419: (20)                         "stroke_width": 2,
1420: (20)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1421: (16)                         (3, 4): {"color": RED},
1422: (16)
1423: (20)                     g = DiGraph(
1424: (20)                         vertices,
1425: (20)                         edges,
1426: (20)                         labels=True,
1427: (20)                         layout="circular",
1428: (16)                         edge_config=edge_config,
1429: (16)                         ).scale(1.4)
1430: (16)                         self.play(Create(g))
1431: (16)                         self.wait()
1432: (20)                         self.play(
1433: (20)                             g[1].animate.move_to([1, 1, 1]),
1434: (20)                             g[2].animate.move_to([-1, 1, 2]),
1435: (20)                             g[3].animate.move_to([1, -1, -1]),
1436: (20)                             g[4].animate.move_to([-1, -1, 0]),
1437: (16)
1438: (16)                         )
1439: (16)                         self.wait()
1440: (4)             You can customize the edges and arrow tips globally or locally.
1441: (4)             .. manim:: CustomUndirectedMovingDiGraph
1442: (8)             class CustomUndirectedMovingDiGraph(Scene):
1443: (12)                 def construct(self):
1444: (16)                     vertices = [i for i in range(5)]
1445: (16)                     edges = [
1446: (20)                         (0, 1),
1447: (20)                         (1, 2),
1448: (20)                         (3, 2),
1449: (20)                         (3, 4),
1450: (16)
1451: (16)                     edge_config = {
1452: (16)                         "stroke_width": 2,
1453: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1454: (16)                         (3, 4): {"color": RED},
1455: (16)
1456: (20)                     g = DiGraph(
1457: (20)                         vertices,
1458: (20)                         edges,
1459: (20)                         labels=True,
1460: (20)                         layout="circular",
1461: (16)                         edge_config=edge_config,
1462: (16)                         ).scale(1.4)
1463: (16)                         self.play(Create(g))
1464: (16)                         self.wait()
1465: (16)                         self.play(
1466: (16)                             g[1].animate.move_to([1, 1, 1]),
1467: (16)                             g[2].animate.move_to([-1, 1, 2]),
1468: (16)                             g[3].animate.move_to([1, -1, -1]),
1469: (16)                             g[4].animate.move_to([-1, -1, 0]),
1470: (16)
1471: (16)                         )
1472: (16)                         self.wait()
1473: (4)             You can customize the edges and arrow tips globally or locally.
1474: (4)             .. manim:: CustomCustomUndirectedMovingDiGraph
1475: (8)             class CustomCustomUndirectedMovingDiGraph(Scene):
1476: (12)                 def construct(self):
1477: (16)                     vertices = [i for i in range(5)]
1478: (16)                     edges = [
1479: (20)                         (0, 1),
1480: (20)                         (1, 2),
1481: (20)                         (3, 2),
1482: (20)                         (3, 4),
1483: (16)
1484: (16)                     edge_config = {
1485: (16)                         "stroke_width": 2,
1486: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1487: (16)                         (3, 4): {"color": RED},
1488: (16)
1489: (20)                     g = DiGraph(
1490: (20)                         vertices,
1491: (20)                         edges,
1492: (20)                         labels=True,
1493: (20)                         layout="circular",
1494: (16)                         edge_config=edge_config,
1495: (16)                         ).scale(1.4)
1496: (16)                         self.play(Create(g))
1497: (16)                         self.wait()
1498: (16)                         self.play(
1499: (16)                             g[1].animate.move_to([1, 1, 1]),
1500: (16)                             g[2].animate.move_to([-1, 1, 2]),
1501: (16)                             g[3].animate.move_to([1, -1, -1]),
1502: (16)                             g[4].animate.move_to([-1, -1, 0]),
1503: (16)
1504: (16)                         )
1505: (16)                         self.wait()
1506: (4)             You can customize the edges and arrow tips globally or locally.
1507: (4)             .. manim:: CustomCustomCustomUndirectedMovingDiGraph
1508: (8)             class CustomCustomCustomUndirectedMovingDiGraph(Scene):
1509: (12)                 def construct(self):
1510: (16)                     vertices = [i for i in range(5)]
1511: (16)                     edges = [
1512: (20)                         (0, 1),
1513: (20)                         (1, 2),
1514: (20)                         (3, 2),
1515: (20)                         (3, 4),
1516: (16)
1517: (16)                     edge_config = {
1518: (16)                         "stroke_width": 2,
1519: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1520: (16)                         (3, 4): {"color": RED},
1521: (16)
1522: (20)                     g = DiGraph(
1523: (20)                         vertices,
1524: (20)                         edges,
1525: (20)                         labels=True,
1526: (20)                         layout="circular",
1527: (16)                         edge_config=edge_config,
1528: (16)                         ).scale(1.4)
1529: (16)                         self.play(Create(g))
1530: (16)                         self.wait()
1531: (16)                         self.play(
1532: (16)                             g[1].animate.move_to([1, 1, 1]),
1533: (16)                             g[2].animate.move_to([-1, 1, 2]),
1534: (16)                             g[3].animate.move_to([1, -1, -1]),
1535: (16)                             g[4].animate.move_to([-1, -1, 0]),
1536: (16)
1537: (16)                         )
1538: (16)                         self.wait()
1539: (4)             You can customize the edges and arrow tips globally or locally.
1540: (4)             .. manim:: CustomCustomCustomCustomUndirectedMovingDiGraph
1541: (8)             class CustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1542: (12)                 def construct(self):
1543: (16)                     vertices = [i for i in range(5)]
1544: (16)                     edges = [
1545: (20)                         (0, 1),
1546: (20)                         (1, 2),
1547: (20)                         (3, 2),
1548: (20)                         (3, 4),
1549: (16)
1550: (16)                     edge_config = {
1551: (16)                         "stroke_width": 2,
1552: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1553: (16)                         (3, 4): {"color": RED},
1554: (16)
1555: (20)                     g = DiGraph(
1556: (20)                         vertices,
1557: (20)                         edges,
1558: (20)                         labels=True,
1559: (20)                         layout="circular",
1560: (16)                         edge_config=edge_config,
1561: (16)                         ).scale(1.4)
1562: (16)                         self.play(Create(g))
1563: (16)                         self.wait()
1564: (16)                         self.play(
1565: (16)                             g[1].animate.move_to([1, 1, 1]),
1566: (16)                             g[2].animate.move_to([-1, 1, 2]),
1567: (16)                             g[3].animate.move_to([1, -1, -1]),
1568: (16)                             g[4].animate.move_to([-1, -1, 0]),
1569: (16)
1570: (16)                         )
1571: (16)                         self.wait()
1572: (4)             You can customize the edges and arrow tips globally or locally.
1573: (4)             .. manim:: CustomCustomCustomCustomCustomUndirectedMovingDiGraph
1574: (8)             class CustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1575: (12)                 def construct(self):
1576: (16)                     vertices = [i for i in range(5)]
1577: (16)                     edges = [
1578: (20)                         (0, 1),
1579: (20)                         (1, 2),
1580: (20)                         (3, 2),
1581: (20)                         (3, 4),
1582: (16)
1583: (16)                     edge_config = {
1584: (16)                         "stroke_width": 2,
1585: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1586: (16)                         (3, 4): {"color": RED},
1587: (16)
1588: (20)                     g = DiGraph(
1589: (20)                         vertices,
1590: (20)                         edges,
1591: (20)                         labels=True,
1592: (20)                         layout="circular",
1593: (16)                         edge_config=edge_config,
1594: (16)                         ).scale(1.4)
1595: (16)                         self.play(Create(g))
1596: (16)                         self.wait()
1597: (16)                         self.play(
1598: (16)                             g[1].animate.move_to([1, 1, 1]),
1599: (16)                             g[2].animate.move_to([-1, 1, 2]),
1600: (16)                             g[3].animate.move_to([1, -1, -1]),
1601: (16)                             g[4].animate.move_to([-1, -1, 0]),
1602: (16)
1603: (16)                         )
1604: (16)                         self.wait()
1605: (4)             You can customize the edges and arrow tips globally or locally.
1606: (4)             .. manim:: CustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1607: (8)             class CustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1608: (12)                 def construct(self):
1609: (16)                     vertices = [i for i in range(5)]
1610: (16)                     edges = [
1611: (20)                         (0, 1),
1612: (20)                         (1, 2),
1613: (20)                         (3, 2),
1614: (20)                         (3, 4),
1615: (16)
1616: (16)                     edge_config = {
1617: (16)                         "stroke_width": 2,
1618: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1619: (16)                         (3, 4): {"color": RED},
1620: (16)
1621: (20)                     g = DiGraph(
1622: (20)                         vertices,
1623: (20)                         edges,
1624: (20)                         labels=True,
1625: (20)                         layout="circular",
1626: (16)                         edge_config=edge_config,
1627: (16)                         ).scale(1.4)
1628: (16)                         self.play(Create(g))
1629: (16)                         self.wait()
1630: (16)                         self.play(
1631: (16)                             g[1].animate.move_to([1, 1, 1]),
1632: (16)                             g[2].animate.move_to([-1, 1, 2]),
1633: (16)                             g[3].animate.move_to([1, -1, -1]),
1634: (16)                             g[4].animate.move_to([-1, -1, 0]),
1635: (16)
1636: (16)                         )
1637: (16)                         self.wait()
1638: (4)             You can customize the edges and arrow tips globally or locally.
1639: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1640: (8)             class CustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1641: (12)                 def construct(self):
1642: (16)                     vertices = [i for i in range(5)]
1643: (16)                     edges = [
1644: (20)                         (0, 1),
1645: (20)                         (1, 2),
1646: (20)                         (3, 2),
1647: (20)                         (3, 4),
1648: (16)
1649: (16)                     edge_config = {
1650: (16)                         "stroke_width": 2,
1651: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1652: (16)                         (3, 4): {"color": RED},
1653: (16)
1654: (20)                     g = DiGraph(
1655: (20)                         vertices,
1656: (20)                         edges,
1657: (20)                         labels=True,
1658: (20)                         layout="circular",
1659: (16)                         edge_config=edge_config,
1660: (16)                         ).scale(1.4)
1661: (16)                         self.play(Create(g))
1662: (16)                         self.wait()
1663: (16)                         self.play(
1664: (16)                             g[1].animate.move_to([1, 1, 1]),
1665: (16)                             g[2].animate.move_to([-1, 1, 2]),
1666: (16)                             g[3].animate.move_to([1, -1, -1]),
1667: (16)                             g[4].animate.move_to([-1, -1, 0]),
1668: (16)
1669: (16)                         )
1670: (16)                         self.wait()
1671: (4)             You can customize the edges and arrow tips globally or locally.
1672: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1673: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1674: (12)                 def construct(self):
1675: (16)                     vertices = [i for i in range(5)]
1676: (16)                     edges = [
1677: (20)                         (0, 1),
1678: (20)                         (1, 2),
1679: (20)                         (3, 2),
1680: (20)                         (3, 4),
1681: (16)
1682: (16)                     edge_config = {
1683: (16)                         "stroke_width": 2,
1684: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1685: (16)                         (3, 4): {"color": RED},
1686: (16)
1687: (20)                     g = DiGraph(
1688: (20)                         vertices,
1689: (20)                         edges,
1690: (20)                         labels=True,
1691: (20)                         layout="circular",
1692: (16)                         edge_config=edge_config,
1693: (16)                         ).scale(1.4)
1694: (16)                         self.play(Create(g))
1695: (16)                         self.wait()
1696: (16)                         self.play(
1697: (16)                             g[1].animate.move_to([1, 1, 1]),
1698: (16)                             g[2].animate.move_to([-1, 1, 2]),
1699: (16)                             g[3].animate.move_to([1, -1, -1]),
1700: (16)                             g[4].animate.move_to([-1, -1, 0]),
1701: (16)
1702: (16)                         )
1703: (16)                         self.wait()
1704: (4)             You can customize the edges and arrow tips globally or locally.
1705: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1706: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1707: (12)                 def construct(self):
1708: (16)                     vertices = [i for i in range(5)]
1709: (16)                     edges = [
1710: (20)                         (0, 1),
1711: (20)                         (1, 2),
1712: (20)                         (3, 2),
1713: (20)                         (3, 4),
1714: (16)
1715: (16)                     edge_config = {
1716: (16)                         "stroke_width": 2,
1717: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1718: (16)                         (3, 4): {"color": RED},
1719: (16)
1720: (20)                     g = DiGraph(
1721: (20)                         vertices,
1722: (20)                         edges,
1723: (20)                         labels=True,
1724: (20)                         layout="circular",
1725: (16)                         edge_config=edge_config,
1726: (16)                         ).scale(1.4)
1727: (16)                         self.play(Create(g))
1728: (16)                         self.wait()
1729: (16)                         self.play(
1730: (16)                             g[1].animate.move_to([1, 1, 1]),
1731: (16)                             g[2].animate.move_to([-1, 1, 2]),
1732: (16)                             g[3].animate.move_to([1, -1, -1]),
1733: (16)                             g[4].animate.move_to([-1, -1, 0]),
1734: (16)
1735: (16)                         )
1736: (16)                         self.wait()
1737: (4)             You can customize the edges and arrow tips globally or locally.
1738: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1739: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1740: (12)                 def construct(self):
1741: (16)                     vertices = [i for i in range(5)]
1742: (16)                     edges = [
1743: (20)                         (0, 1),
1744: (20)                         (1, 2),
1745: (20)                         (3, 2),
1746: (20)                         (3, 4),
1747: (16)
1748: (16)                     edge_config = {
1749: (16)                         "stroke_width": 2,
1750: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1751: (16)                         (3, 4): {"color": RED},
1752: (16)
1753: (20)                     g = DiGraph(
1754: (20)                         vertices,
1755: (20)                         edges,
1756: (20)                         labels=True,
1757: (20)                         layout="circular",
1758: (16)                         edge_config=edge_config,
1759: (16)                         ).scale(1.4)
1760: (16)                         self.play(Create(g))
1761: (16)                         self.wait()
1762: (16)                         self.play(
1763: (16)                             g[1].animate.move_to([1, 1, 1]),
1764: (16)                             g[2].animate.move_to([-1, 1, 2]),
1765: (16)                             g[3].animate.move_to([1, -1, -1]),
1766: (16)                             g[4].animate.move_to([-1, -1, 0]),
1767: (16)
1768: (16)                         )
1769: (16)                         self.wait()
1770: (4)             You can customize the edges and arrow tips globally or locally.
1771: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1772: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1773: (12)                 def construct(self):
1774: (16)                     vertices = [i for i in range(5)]
1775: (16)                     edges = [
1776: (20)                         (0, 1),
1777: (20)                         (1, 2),
1778: (20)                         (3, 2),
1779: (20)                         (3, 4),
1780: (16)
1781: (16)                     edge_config = {
1782: (16)                         "stroke_width": 2,
1783: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1784: (16)                         (3, 4): {"color": RED},
1785: (16)
1786: (20)                     g = DiGraph(
1787: (20)                         vertices,
1788: (20)                         edges,
1789: (20)                         labels=True,
1790: (20)                         layout="circular",
1791: (16)                         edge_config=edge_config,
1792: (16)                         ).scale(1.4)
1793: (16)                         self.play(Create(g))
1794: (16)                         self.wait()
1795: (16)                         self.play(
1796: (16)                             g[1].animate.move_to([1, 1, 1]),
1797: (16)                             g[2].animate.move_to([-1, 1, 2]),
1798: (16)                             g[3].animate.move_to([1, -1, -1]),
1799: (16)                             g[4].animate.move_to([-1, -1, 0]),
1800: (16)
1801: (16)                         )
1802: (16)                         self.wait()
1803: (4)             You can customize the edges and arrow tips globally or locally.
1804: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1805: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1806: (12)                 def construct(self):
1807: (16)                     vertices = [i for i in range(5)]
1808: (16)                     edges = [
1809: (20)                         (0, 1),
1810: (20)                         (1, 2),
1811: (20)                         (3, 2),
1812: (20)                         (3, 4),
1813: (16)
1814: (16)                     edge_config = {
1815: (16)                         "stroke_width": 2,
1816: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1817: (16)                         (3, 4): {"color": RED},
1818: (16)
1819: (20)                     g = DiGraph(
1820: (20)                         vertices,
1821: (20)                         edges,
1822: (20)                         labels=True,
1823: (20)                         layout="circular",
1824: (16)                         edge_config=edge_config,
1825: (16)                         ).scale(1.4)
1826: (16)                         self.play(Create(g))
1827: (16)                         self.wait()
1828: (16)                         self.play(
1829: (16)                             g[1].animate.move_to([1, 1, 1]),
1830: (16)                             g[2].animate.move_to([-1, 1, 2]),
1831: (16)                             g[3].animate.move_to([1, -1, -1]),
1832: (16)                             g[4].animate.move_to([-1, -1, 0]),
1833: (16)
1834: (16)                         )
1835: (16)                         self.wait()
1836: (4)             You can customize the edges and arrow tips globally or locally.
1837: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1838: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1839: (12)                 def construct(self):
1840: (16)                     vertices = [i for i in range(5)]
1841: (16)                     edges = [
1842: (20)                         (0, 1),
1843: (20)                         (1, 2),
1844: (20)                         (3, 2),
1845: (20)                         (3, 4),
1846: (16)
1847: (16)                     edge_config = {
1848: (16)                         "stroke_width": 2,
1849: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1850: (16)                         (3, 4): {"color": RED},
1851: (16)
1852: (20)                     g = DiGraph(
1853: (20)                         vertices,
1854: (20)                         edges,
1855: (20)                         labels=True,
1856: (20)                         layout="circular",
1857: (16)                         edge_config=edge_config,
1858: (16)                         ).scale(1.4)
1859: (16)                         self.play(Create(g))
1860: (16)                         self.wait()
1861: (16)                         self.play(
1862: (16)                             g[1].animate.move_to([1, 1, 1]),
1863: (16)                             g[2].animate.move_to([-1, 1, 2]),
1864: (16)                             g[3].animate.move_to([1, -1, -1]),
1865: (16)                             g[4].animate.move_to([-1, -1, 0]),
1866: (16)
1867: (16)                         )
1868: (16)                         self.wait()
1869: (4)             You can customize the edges and arrow tips globally or locally.
1870: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1871: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1872: (12)                 def construct(self):
1873: (16)                     vertices = [i for i in range(5)]
1874: (16)                     edges = [
1875: (20)                         (0, 1),
1876: (20)                         (1, 2),
1877: (20)                         (3, 2),
1878: (20)                         (3, 4),
1879: (16)
1880: (16)                     edge_config = {
1881: (16)                         "stroke_width": 2,
1882: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1883: (16)                         (3, 4): {"color": RED},
1884: (16)
1885: (20)                     g = DiGraph(
1886: (20)                         vertices,
1887: (20)                         edges,
1888: (20)                         labels=True,
1889: (20)                         layout="circular",
1890: (16)                         edge_config=edge_config,
1891: (16)                         ).scale(1.4)
1892: (16)                         self.play(Create(g))
1893: (16)                         self.wait()
1894: (16)                         self.play(
1895: (16)                             g[1].animate.move_to([1, 1, 1]),
1896: (16)                             g[2].animate.move_to([-1, 1, 2]),
1897: (16)                             g[3].animate.move_to([1, -1, -1]),
1898: (16)                             g[4].animate.move_to([-1, -1, 0]),
1899: (16)
1900: (16)                         )
1901: (16)                         self.wait()
1902: (4)             You can customize the edges and arrow tips globally or locally.
1903: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1904: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1905: (12)                 def construct(self):
1906: (16)                     vertices = [i for i in range(5)]
1907: (16)                     edges = [
1908: (20)                         (0, 1),
1909: (20)                         (1, 2),
1910: (20)                         (3, 2),
1911: (20)                         (3, 4),
1912: (16)
1913: (16)                     edge_config = {
1914: (16)                         "stroke_width": 2,
1915: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1916: (16)                         (3, 4): {"color": RED},
1917: (16)
1918: (20)                     g = DiGraph(
1919: (20)                         vertices,
1920: (20)                         edges,
1921: (20)                         labels=True,
1922: (20)                         layout="circular",
1923: (16)                         edge_config=edge_config,
1924: (16)                         ).scale(1.4)
1925: (16)                         self.play(Create(g))
1926: (16)                         self.wait()
1927: (16)                         self.play(
1928: (16)                             g[1].animate.move_to([1, 1, 1]),
1929: (16)                             g[2].animate.move_to([-1, 1, 2]),
1930: (16)                             g[3].animate.move_to([1, -1, -1]),
1931: (16)                             g[4].animate.move_to([-1, -1, 0]),
1932: (16)
1933: (16)                         )
1934: (16)                         self.wait()
1935: (4)             You can customize the edges and arrow tips globally or locally.
1936: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1937: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1938: (12)                 def construct(self):
1939: (16)                     vertices = [i for i in range(5)]
1940: (16)                     edges = [
1941: (20)                         (0, 1),
1942: (20)                         (1, 2),
1943: (20)                         (3, 2),
1944: (20)                         (3, 4),
1945: (16)
1946: (16)                     edge_config = {
1947: (16)                         "stroke_width": 2,
1948: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1949: (16)                         (3, 4): {"color": RED},
1950: (16)
1951: (20)                     g = DiGraph(
1952: (20)                         vertices,
1953: (20)                         edges,
1954: (20)                         labels=True,
1955: (20)                         layout="circular",
1956: (16)                         edge_config=edge_config,
1957: (16)                         ).scale(1.4)
1958: (16)                         self.play(Create(g))
1959: (16)                         self.wait()
1960: (16)                         self.play(
1961: (16)                             g[1].animate.move_to([1, 1, 1]),
1962: (16)                             g[2].animate.move_to([-1, 1, 2]),
1963: (16)                             g[3].animate.move_to([1, -1, -1]),
1964: (16)                             g[4].animate.move_to([-1, -1, 0]),
1965: (16)
1966: (16)                         )
1967: (16)                         self.wait()
1968: (4)             You can customize the edges and arrow tips globally or locally.
1969: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
1970: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
1971: (12)                 def construct(self):
1972: (16)                     vertices = [i for i in range(5)]
1973: (16)                     edges = [
1974: (20)                         (0, 1),
1975: (20)                         (1, 2),
1976: (20)                         (3, 2),
1977: (20)                         (3, 4),
1978: (16)
1979: (16)                     edge_config = {
1980: (16)                         "stroke_width": 2,
1981: (16)                         "tip_config": {"tip_length": 0, "tip_width": 0},
1982: (16)                         (3, 4): {"color": RED},
1983: (16)
1984: (20)                     g = DiGraph(
1985: (20)                         vertices,
1986: (20)                         edges,
1987: (20)                         labels=True,
1988: (20)                         layout="circular",
1989: (16)                         edge_config=edge_config,
1990: (16)                         ).scale(1.4)
1991: (16)                         self.play(Create(g))
1992: (16)                         self.wait()
1993: (16)                         self.play(
1994: (16)                             g[1].animate.move_to([1, 1, 1]),
1995: (16)                             g[2].animate.move_to([-1, 1, 2]),
1996: (16)                             g[3].animate.move_to([1, -1, -1]),
1997: (16)                             g[4].animate.move_to([-1, -1, 0]),
1998: (16)
1999: (16)                         )
2000: (16)                         self.wait()
2001: (4)             You can customize the edges and arrow tips globally or locally.
2002: (4)             .. manim:: CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph
2003: (8)             class CustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomCustomUndirectedMovingDiGraph(Scene):
2004: (12)                 def construct(self):
2005: (16)                     vertices = [i for i in range(5)]
2006: (16)                     edges = [
2007: (20)                         (0, 1),
2008: (20)                         (1, 2),
2009: (20)                         (3, 2),
2010: (20)                         (3, 4),
2011: (16)
2012: (16)                     edge_config =
```

```

1433: (20)                     g[2].animate.move_to([-1, 1, 2]),
1434: (20)                     g[3].animate.move_to([-1.5, -1.5, -1]),
1435: (20)                     g[4].animate.move_to([1, -2, -1]),
1436: (16)                     )
1437: (16)                     self.wait()
1438: (4)                     """
1439: (4)             @staticmethod
1440: (4)             def _empty_networkx_graph() -> nx.DiGraph:
1441: (8)                 return nx.DiGraph()
1442: (4)             def _populate_edge_dict(
1443: (8)                 self, edges: list[tuple[Hashable, Hashable]], edge_type: type[Mobject]
1444: (4)             ):
1445: (8)                 self.edges = {
1446: (12)                     (u, v): edge_type(
1447: (16)                         self[u],
1448: (16)                         self[v],
1449: (16)                         z_index=-1,
1450: (16)                         **self._edge_config[(u, v)],
1451: (12)                     )
1452: (12)                     for (u, v) in edges
1453: (8)                 }
1454: (8)                     for (u, v), edge in self.edges.items():
1455: (12)                         edge.add_tip(**self._tip_config[(u, v)])
1456: (4)             def update_edges(self, graph):
1457: (8)                 """Updates the edges to stick at their corresponding vertices.
1458: (8)                 Arrow tips need to be repositioned since otherwise they can be
1459: (8)                 deformed.
1460: (8)                 """
1461: (8)                     for (u, v), edge in graph.edges.items():
1462: (12)                         edge_type = type(edge)
1463: (12)                         tip = edge.pop_tips()[0]
1464: (12)                         new_edge = edge_type(self[u], self[v], **self._edge_config[(u,
v)])
1465: (12)                         edge.become(new_edge)
1466: (12)                         edge.add_tip(tip)
1467: (4)             def __repr__(self: DiGraph) -> str:
1468: (8)                 return f"Directed graph on {len(self.vertices)} vertices and
{len(self.edges)} edges"

```

---

File 54 - scale.py:

```

1: (0)                     from __future__ import annotations
2: (0)                     import math
3: (0)                     from typing import TYPE_CHECKING, Any, Iterable
4: (0)                     import numpy as np
5: (0)                     __all__ = ["LogBase", "LinearBase"]
6: (0)                     from manim.mobject.text.numbers import Integer
7: (0)                     if TYPE_CHECKING:
8: (4)                         from manim.mobject.mobject import Mobject
9: (0)                     class _ScaleBase:
10: (4)                         """Scale baseclass for graphing/functions.
11: (4)                         Parameters
12: (4)                         -----
13: (4)                         custom_labels
14: (8)                             Whether to create custom labels when plotted on a
:class:`~.NumberLine`.
15: (4)                         """
16: (4)                     def __init__(self, custom_labels: bool = False):
17: (8)                         self.custom_labels = custom_labels
18: (4)                     def function(self, value: float) -> float:
19: (8)                         """The function that will be used to scale the values.
20: (8)                         Parameters
21: (8)                         -----
22: (8)                         value
23: (12)                             The number/``np.ndarray`` to be scaled.
24: (8)                         Returns
25: (8)                         -----

```

```

26: (8)           float
27: (12)         The value after it has undergone the scaling.
28: (8)
29: (8)
30: (8)
31: (12)       Raises
32: (8)         -----
33: (8)         NotImplementedError
34: (4)           Must be subclassed.
35: (8)           """
36: (8)           raise NotImplementedError
37: (4)       def inverse_function(self, value: float) -> float:
38: (8)           """The inverse of ``function``. Used for plotting on a particular
39: (12)         axis.
40: (8)
41: (8)           Raises
42: (4)           -----
43: (8)           NotImplementedError
44: (12)           Must be subclassed.
45: (8)           """
46: (8)           raise NotImplementedError
47: (4)       def get_custom_labels(
48: (8)           self,
49: (8)           val_range: Iterable[float],
50: (12)       ) -> Iterable[Mobject]:
51: (8)           """Custom instructions for generating labels along an axis.
52: (8)           Parameters
53: (8)           -----
54: (8)           val_range
55: (12)           The position of labels. Also used for defining the content of the
56: (8)           labels.
57: (8)
58: (8)
59: (12)       Returns
60: (8)           -----
61: (8)           Dict
62: (0)             A list consisting of the labels.
63: (4)             Can be passed to :meth:`~.NumberLine.add_labels()` along with
64: (12)             ``val_range``.
65: (8)
66: (8)
67: (8)
68: (12)       Raises
69: (8)           -----
70: (8)           NotImplementedError
71: (8)           Can be subclassed, optional.
72: (4)           """
73: (8)           raise NotImplementedError
74: (4)       class LinearBase(_ScaleBase):
75: (8)           def __init__(self, scale_factor: float = 1.0):
76: (8)               """The default scaling class.
77: (8)               Parameters
78: (8)               -----
79: (8)               scale_factor
80: (4)                 The slope of the linear function, by default 1.0
81: (8)                 """
82: (4)                 super().__init__()
83: (8)                 self.scale_factor = scale_factor
84: (4)       def function(self, value: float) -> float:
85: (8)           """Multiplies the value by the scale factor.
86: (8)           Parameters
87: (8)           -----
88: (8)           value
89: (12)             Value to be multiplied by the scale factor.
90: (8)             """
91: (8)             return self.scale_factor * value
92: (4)       def inverse_function(self, value: float) -> float:
93: (8)           """Inverse of function. Divides the value by the scale factor.
94: (8)           Parameters
95: (8)           -----
96: (8)           value
97: (12)             value to be divided by the scale factor.
98: (8)             """
99: (8)             return value / self.scale_factor
100: (0)      class LogBase(_ScaleBase):
101: (4)        def __init__(self, base: float = 10, custom_labels: bool = True):
102: (8)            """Scale for logarithmic graphs/functions.
103: (8)            Parameters

```

```

92: (8)          -----
93: (8)          base
94: (12)         The base of the log, by default 10.
95: (8)          custom_labels
96: (12)         For use with :class:`~.Axes`:
97: (12)         Whether or not to include ``LaTeX`` axis labels, by default True.
98: (8)          Examples
99: (8)          -----
100: (8)         .. code-block:: python
101: (12)         func = ParametricFunction(lambda x: x, scaling=LogBase(base=2))
102: (8)
103: (8)         super().__init__()
104: (8)         self.base = base
105: (8)         self.custom_labels = custom_labels
106: (4)         def function(self, value: float) -> float:
107: (8)           """Scales the value to fit it to a logarithmic
scale.``self.function(5)==10**5``"""
108: (8)           return self.base**value
109: (4)         def inverse_function(self, value: float) -> float:
110: (8)           """Inverse of ``function``. The value must be greater than 0"""
111: (8)           if isinstance(value, np.ndarray):
112: (12)             condition = value.any() <= 0
113: (12)             func = lambda value, base: np.log(value) / np.log(base)
114: (8)
115: (12)
116: (12)
117: (8)
118: (12)
119: (16)
the function"
120: (12)
121: (8)         )
122: (8)         value = func(value, self.base)
123: (4)         return value
124: (8)
125: (8)         def get_custom_labels(
126: (8)           self,
127: (8)           val_range: Iterable[float],
128: (4)           unit_decimal_places: int = 0,
129: (8)           **base_config: dict[str, Any],
130: (8)           ) -> list[Mobject]:
131: (8)             """Produces custom :class:`~.Integer` labels in the form of ``10^2``.
Parameters
-----
132: (8)           val_range
133: (12)
exponent.
134: (8)
135: (12)
136: (8)
137: (12)
138: (8)
139: (8)
140: (12)
141: (16)
142: (16)
{unit_decimal_places}f"},",
143: (16)
144: (12)
145: (12)
146: (8)
147: (8)
-----
```

File 55 - matrix.py:

```

1: (0)          r"""Mobjects representing matrices.
2: (0)          Examples
3: (0)          -----
4: (0)          .. manim:: MatrixExamples
```

```

5: (4)             :save_last_frame:
6: (4)         class MatrixExamples(Scene):
7: (8)             def construct(self):
8: (12)                 m0 = Matrix([["\pi", 0], [-1, 1]])
9: (12)                 m1 = IntegerMatrix([[1.5, 0.], [12, -1.3]],
10: (16)                     left_bracket="(",
11: (16)                     right_bracket=")")
12: (12)                 m2 = DecimalMatrix(
13: (16)                     [[3.456, 2.122], [33.2244, 12.33]],
14: (16)                     element_to_mobject_config={"num_decimal_places": 2},
15: (16)                     left_bracket="\{",
16: (16)                     right_bracket="\}")
17: (12)                 m3 = MobjectMatrix(
18: (16)                     [[Circle().scale(0.3), Square().scale(0.3)],
19: (16)                     [MathTex("\pi").scale(2), Star().scale(0.3)]],
20: (16)                     left_bracket="\langle",
21: (16)                     right_bracket="\rangle")
22: (12)                 g = Group(m0, m1, m2, m3).arrange_in_grid(buff=2)
23: (12)                 self.add(g)
24: (0)             """
25: (0)             from __future__ import annotations
26: (0)             __all__ = [
27: (4)                 "Matrix",
28: (4)                 "DecimalMatrix",
29: (4)                 "IntegerMatrix",
30: (4)                 "MobjectMatrix",
31: (4)                 "matrix_to_tex_string",
32: (4)                 "matrix_to_mobject",
33: (4)                 "get_det_text",
34: (0)             ]
35: (0)             import itertools as it
36: (0)             from typing import Iterable, Sequence
37: (0)             import numpy as np
38: (0)             from manim.mobject.mobject import Mobject
39: (0)             from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
40: (0)             from manim.mobject.text.numbers import DecimalNumber, Integer
41: (0)             from manim.mobject.text.tex_mobject import MathTex, Tex
42: (0)             from ..constants import *
43: (0)             from ..mobject.types.vectorized_mobject import VGroup, VMobject
44: (0)             def matrix_to_tex_string(matrix):
45: (4)                 matrix = np.array(matrix).astype("str")
46: (4)                 if matrix.ndim == 1:
47: (8)                     matrix = matrix.reshape((matrix.size, 1))
48: (4)                 n_rows, n_cols = matrix.shape
49: (4)                 prefix = "\left[ \begin{array}{%s}" % ("c" * n_cols)
50: (4)                 suffix = "\end{array} \right]"
51: (4)                 rows = [" ".join(row) for row in matrix]
52: (4)                 return prefix + " \\ ".join(rows) + suffix
53: (0)             def matrix_to_mobject(matrix):
54: (4)                 return MathTex(matrix_to_tex_string(matrix))
55: (0)             class Matrix(VMobject, metaclass=ConvertToOpenGL):
56: (4)                 """A mobject that displays a matrix on the screen.
57: (4)                 Parameters
58: (4)                 -----
59: (4)                 matrix
60: (8)                     A numpy 2d array or list of lists.
61: (4)                 v_buff
62: (8)                     Vertical distance between elements, by default 0.8.
63: (4)                 h_buff
64: (8)                     Horizontal distance between elements, by default 1.3.
65: (4)                 bracket_h_buff
66: (8)                     Distance of the brackets from the matrix, by default
``MED_SMALL_BUFF``.
67: (4)                 bracket_v_buff
68: (8)                     Height of the brackets, by default ``MED_SMALL_BUFF``.
69: (4)                 add_background_rectangles_to_entries
70: (8)                     ``True`` if should add backgraound rectangles to entries, by default
``False``.
71: (4)                 include_background_rectangle

```

```

72: (8)                      ``True`` if should include background rectangle, by default ``False``.
73: (4)                      element_to_mob
74: (8)                      The mob object class used to construct the elements, by default
:class:`~.MathTex`.
75: (4)                      element_to_mob_config
76: (8)                      Additional arguments to be passed to the constructor in
``element_to_mob``,
77: (8)                      by default ``{}``.
78: (4)                      element_alignment_corner
79: (8)                      The corner to which elements are aligned, by default ``DR``.
80: (4)                      left_bracket
81: (8)                      The left bracket type, by default `` "[" ``.
82: (4)                      right_bracket
83: (8)                      The right bracket type, by default `` "]" ``.
84: (4)                      stretch_brackets
85: (8)                      ``True`` if should stretch the brackets to fit the height of matrix
contents, by default ``True``.
86: (4)                      bracket_config
87: (8)                      Additional arguments to be passed to :class:`~.MathTex` when
constructing
88: (8)                      the brackets.
89: (4)                      Examples
90: (4)                      -----
91: (4)                      The first example shows a variety of uses of this module while the second
example
92: (4)                      explains the use of the options `add_background_rectangles_to_entries` and
`include_background_rectangle`.
93: (4)
94: (4)                      .. manim:: MatrixExamples
95: (8)                      :save_last_frame:
96: (8)                      class MatrixExamples(Scene):
97: (12)                      def construct(self):
98: (16)                      m0 = Matrix([[2, "\\\pi"], [-1, 1]])
99: (16)                      m1 = Matrix([[2, 0, 4], [-1, 1, 5]],
100: (20)                         v_buff=1.3,
101: (20)                         h_buff=0.8,
102: (20)                         bracket_h_buff=SMALL_BUFF,
103: (20)                         bracket_v_buff=SMALL_BUFF,
104: (20)                         left_bracket="\\"{",
105: (20)                         right_bracket="\\"}")
106: (16)                      m1.add(SurroundingRectangle(m1.get_columns()[1]))
107: (16)                      m2 = Matrix([[2, 1], [-1, 3]],
108: (20)                         element_alignment_corner=UL,
109: (20)                         left_bracket="(",
110: (20)                         right_bracket=")")
111: (16)                      m3 = Matrix([[2, 1], [-1, 3]],
112: (20)                         left_bracket="\\\langle",
113: (20)                         right_bracket="\\\rangle")
114: (16)
115: (16)                      m4 = Matrix([[2, 1], [-1, 3]],
116: (16)                         ).set_column_colors(RED, GREEN)
117: (16)                      m5 = Matrix([[2, 1], [-1, 3]],
118: (16)                         ).set_row_colors(RED, GREEN)
119: (20)                      g = Group(
120: (16)                         m0,m1,m2,m3,m4,m5
121: (16)                         ).arrange_in_grid(buff=2)
122: (4)                         self.add(g)
123: (8)
124: (8)                      .. manim:: BackgroundRectanglesExample
125: (12)                      :save_last_frame:
126: (16)                      class BackgroundRectanglesExample(Scene):
127: (16)                      def construct(self):
128: (16)                         background= Rectangle().scale(3.2)
129: (16)                         background.set_fill(opacity=.5)
130: (16)                         background.set_color([TEAL, RED, YELLOW])
131: (20)                         self.add(background)
132: (16)                         m0 = Matrix([[12, -30], [-1, 15]],
133: (20)                           add_background_rectangles_to_entries=True)
134: (16)                         m1 = Matrix([[2, 0], [-1, 1]],
135: (16)                           include_background_rectangle=True)
136: (16)                         m2 = Matrix([[12, -30], [-1, 15]])
137: (16)                         g = Group(m0, m1, m2).arrange(buff=2)

```

```

136: (16)                         self.add(g)
137: (4)                         """
138: (4)             def __init__(
139: (8)                 self,
140: (8)                 matrix: Iterable,
141: (8)                 v_buff: float = 0.8,
142: (8)                 h_buff: float = 1.3,
143: (8)                 bracket_h_buff: float = MED_SMALL_BUFF,
144: (8)                 bracket_v_buff: float = MED_SMALL_BUFF,
145: (8)                 add_background_rectangles_to_entries: bool = False,
146: (8)                 include_background_rectangle: bool = False,
147: (8)                 element_to_mobject: type[MathTex] = MathTex,
148: (8)                 element_to_mobject_config: dict = {},
149: (8)                 element_alignment_corner: Sequence[float] = DR,
150: (8)                 left_bracket: str = "[",
151: (8)                 right_bracket: str = "]",
152: (8)                 stretch_brackets: bool = True,
153: (8)                 bracket_config: dict = {},
154: (8)                 **kwargs,
155: (4)             ):
156: (8)                 self.v_buff = v_buff
157: (8)                 self.h_buff = h_buff
158: (8)                 self.bracket_h_buff = bracket_h_buff
159: (8)                 self.bracket_v_buff = bracket_v_buff
160: (8)                 self.add_background_rectangles_to_entries =
add_background_rectangles_to_entries
161: (8)                     self.include_background_rectangle = include_background_rectangle
162: (8)                     self.element_to_mobject = element_to_mobject
163: (8)                     self.element_to_mobject_config = element_to_mobject_config
164: (8)                     self.element_alignment_corner = element_alignment_corner
165: (8)                     self.left_bracket = left_bracket
166: (8)                     self.right_bracket = right_bracket
167: (8)                     self.stretch_brackets = stretch_brackets
168: (8)                     super().__init__(**kwargs)
169: (8)                     mob_matrix = self._matrix_to_mob_matrix(matrix)
170: (8)                     self._organize_mob_matrix(mob_matrix)
171: (8)                     self.elements = VGroup(*it.chain(*mob_matrix))
172: (8)                     self.add(self.elements)
173: (8)                     self._add_brackets(self.left_bracket, self.right_bracket,
**bracket_config)
174: (8)                         self.center()
175: (8)                         self.mob_matrix = mob_matrix
176: (8)                         if self.add_background_rectangles_to_entries:
177: (12)                             for mob in self.elements:
178: (16)                                 mob.add_background_rectangle()
179: (8)                             if self.include_background_rectangle:
180: (12)                                 self.add_background_rectangle()
181: (4)             def _matrix_to_mob_matrix(self, matrix):
182: (8)                 return [
183: (12)                     [
184: (16)                         self.element_to_mobject(item,
**self.element_to_mobject_config)
185: (16)                         for item in row
186: (12)                     ]
187: (12)                     for row in matrix
188: (8)                 ]
189: (4)             def _organize_mob_matrix(self, matrix):
190: (8)                 for i, row in enumerate(matrix):
191: (12)                     for j, _ in enumerate(row):
192: (16)                         mob = matrix[i][j]
193: (16)                         mob.move_to(
194: (20)                             i * self.v_buff * DOWN + j * self.h_buff * RIGHT,
195: (20)                             self.element_alignment_corner,
196: (16)                         )
197: (8)                     return self
198: (4)             def _add_brackets(self, left: str = "[", right: str = "]", **kwargs):
199: (8)                         """Adds the brackets to the Matrix mobject.
200: (8)                         See Latex document for various bracket types.
201: (8)                         Parameters

```

```

202: (8)           -----
203: (8)           left
204: (12)          the left bracket, by default "["
205: (8)           right
206: (12)          the right bracket, by default "]"
207: (8)           Returns
208: (8)           -----
209: (8)           :class:`Matrix`:
210: (12)          The current matrix object (self).
211: (8)           """
212: (8)           BRACKET_HEIGHT = 0.5977
213: (8)           n = int((self.height) / BRACKET_HEIGHT) + 1
214: (8)           empty_tex_array = "".join(
215: (12)             [
216: (16)               r"\begin{array}{c}",
217: (16)               *n * [r"\quad \\"],
218: (16)               r"\end{array}",
219: (12)             ]
220: (8)         )
221: (8)         tex_left = "".join(
222: (12)           [
223: (16)             r"\left" + left,
224: (16)             empty_tex_array,
225: (16)             r"\right.",
226: (12)           ]
227: (8)         )
228: (8)         tex_right = "".join(
229: (12)           [
230: (16)             r"\left.",
231: (16)             empty_tex_array,
232: (16)             r"\right" + right,
233: (12)           ]
234: (8)         )
235: (8)         l_bracket = MathTex(tex_left, **kwargs)
236: (8)         r_bracket = MathTex(tex_right, **kwargs)
237: (8)         bracket_pair = VGroup(l_bracket, r_bracket)
238: (8)         if self.stretch_brackets:
239: (12)             bracket_pair.stretch_to_fit_height(self.height + 2 *
self.bracket_v_buff)
240: (8)             l_bracket.next_to(self, LEFT, self.bracket_h_buff)
241: (8)             r_bracket.next_to(self, RIGHT, self.bracket_h_buff)
242: (8)             self.brackets = bracket_pair
243: (8)             self.add(l_bracket, r_bracket)
244: (8)             return self
245: (4)           def get_columns(self):
246: (8)             """Return columns of the matrix as VGroups.
247: (8)             Returns
248: (8)             -----
249: (8)             List[:class:`~.VGroup`]
250: (12)             Each VGroup contains a column of the matrix.
251: (8)             Examples
252: (8)             -----
253: (8)             .. manim:: GetColumnsExample
254: (12)               :save_last_frame:
255: (12)               class GetColumnsExample(Scene):
256: (16)                 def construct(self):
257: (20)                   m0 = Matrix([["\pi", 3], [1, 5]])
258: (20)                   m0.add(SurroundingRectangle(m0.get_columns()[1]))
259: (20)                   self.add(m0)
260: (8)                 """
261: (8)                 return VGroup(
262: (12)                   *
263: (16)                     VGroup(*(row[i] for row in self.mob_matrix))
264: (16)                     for i in range(len(self.mob_matrix[0])))
265: (12)                   )
266: (8)                 )
267: (4)           def set_column_colors(self, *colors: str):
268: (8)             """Set individual colors for each columns of the matrix.
269: (8)             Parameters

```

```

270: (8)           -----
271: (8)           colors
272: (12)          The list of colors; each color specified corresponds to a column.
273: (8)
274: (8)
275: (8)
276: (12)          Returns
277: (8)           -----
278: (8)           :class:`Matrix`
279: (8)             The current matrix object (self).
280: (12)           Examples
281: (12)           -----
282: (16)           .. manim:: SetColumnColorsExample
283: (20)             :save_last_frame:
284: (20)             class SetColumnColorsExample(Scene):
285: (20)               def construct(self):
286: (20)                 m0 = Matrix([["\pi", 1], [-1, 3]],
287: (20)                   ).set_column_colors([RED,BLUE], GREEN)
288: (20)                 self.add(m0)
289: (20)               """
290: (8)               columns = self.get_columns()
291: (8)               for color, column in zip(colors, columns):
292: (8)                 column.set_color(color)
293: (8)               return self
294: (4)             def get_rows(self):
295: (8)               """Return rows of the matrix as VGroups.
296: (8)               Returns
297: (8)               -----
298: (8)               List[:class:`~.VGroup`]
299: (8)                 Each VGroup contains a row of the matrix.
300: (12)             Examples
301: (12)             -----
302: (16)             .. manim:: GetRowsExample
303: (20)               :save_last_frame:
304: (20)               class GetRowsExample(Scene):
305: (20)                 def construct(self):
306: (20)                   m0 = Matrix([["\pi", 3], [1, 5]])
307: (20)                   m0.add(SurroundingRectangle(m0.get_rows()[1]))
308: (20)                   self.add(m0)
309: (20)                 """
310: (8)                 return VGroup(*({VGroup(*row) for row in self.mob_matrix}))
311: (8)             def set_row_colors(self, *colors: str):
312: (8)               """Set individual colors for each row of the matrix.
313: (8)               Parameters
314: (8)               -----
315: (8)               colors
316: (8)                 The list of colors; each color specified corresponds to a row.
317: (8)               Returns
318: (8)               -----
319: (8)               :class:`Matrix`
320: (8)                 The current matrix object (self).
321: (12)             Examples
322: (12)             -----
323: (16)             .. manim:: SetRowColorsExample
324: (20)               :save_last_frame:
325: (20)               class SetRowColorsExample(Scene):
326: (20)                 def construct(self):
327: (20)                   m0 = Matrix([["\pi", 1], [-1, 3]],
328: (20)                     ).set_row_colors([RED,BLUE], GREEN)
329: (20)                     self.add(m0)
330: (20)                 """
331: (8)                 rows = self.get_rows()
332: (8)                 for color, row in zip(colors, rows):
333: (8)                   row.set_color(color)
334: (8)                 return self
335: (8)             def add_background_to_entries(self):
336: (8)               """Add a black background rectangle to the matrix,
337: (8)               see above for an example.
338: (12)             Returns
339: (12)             -----
340: (12)             :class:`Matrix`
341: (12)               The current matrix object (self).

```

```

339: (8)             """
340: (8)             for mob in self.get_entries():
341: (12)             mob.add_background_rectangle()
342: (8)             return self
343: (4)             def get_mob_matrix(self):
344: (8)                 """Return the underlying mob matrix mobjects.
345: (8)                 Returns
346: (8)                 -----
347: (8)                 List[:class:`~.VGroup`]
348: (12)                 Each VGroup contains a row of the matrix.
349: (8)
350: (8)             return self.mob_matrix
351: (4)             def get_entries(self):
352: (8)                 """Return the individual entries of the matrix.
353: (8)                 Returns
354: (8)                 -----
355: (8)                 :class:`~.VGroup`
356: (12)                 VGroup containing entries of the matrix.
357: (8)             Examples
358: (8)             -----
359: (8)             .. manim:: GetEntriesExample
360: (12)             :save_last_frame:
361: (12)             class GetEntriesExample(Scene):
362: (16)             def construct(self):
363: (20)                 m0 = Matrix([[2, 3], [1, 5]])
364: (20)                 ent = m0.get_entries()
365: (20)                 colors = [BLUE, GREEN, YELLOW, RED]
366: (20)                 for k in range(len(colors)):
367: (24)                     ent[k].set_color(colors[k])
368: (20)                 self.add(m0)
369: (8)             """
370: (8)             return self.elements
371: (4)             def get_brackets(self):
372: (8)                 """Return the bracket mobjects.
373: (8)                 Returns
374: (8)                 -----
375: (8)                 List[:class:`~.VGroup`]
376: (12)                 Each VGroup contains a bracket
377: (8)             Examples
378: (8)             -----
379: (8)             .. manim:: GetBracketsExample
380: (12)             :save_last_frame:
381: (12)             class GetBracketsExample(Scene):
382: (16)             def construct(self):
383: (20)                 m0 = Matrix([["\pi", 3], [1, 5]])
384: (20)                 bra = m0.get_brackets()
385: (20)                 colors = [BLUE, GREEN]
386: (20)                 for k in range(len(colors)):
387: (24)                     bra[k].set_color(colors[k])
388: (20)                 self.add(m0)
389: (8)             """
390: (8)             return self.brackets
391: (0)             class DecimalMatrix(Matrix):
392: (4)                 """A mobject that displays a matrix with decimal entries on the screen.
393: (4)             Examples
394: (4)             -----
395: (4)             .. manim:: DecimalMatrixExample
396: (8)             :save_last_frame:
397: (8)             class DecimalMatrixExample(Scene):
398: (12)             def construct(self):
399: (16)                 m0 = DecimalMatrix(
400: (20)                     [[3.456, 2.122], [33.2244, 12]],
401: (20)                     element_to_mobject_config={"num_decimal_places": 2},
402: (20)                     left_bracket="\{",
403: (20)                     right_bracket="\}")
404: (16)                 self.add(m0)
405: (4)             """
406: (4)             def __init__(
407: (8)                 self,

```

```

408: (8)             matrix: Iterable,
409: (8)             element_to_mobject: Mobject = DecimalNumber,
410: (8)             element_to_mobject_config: dict[str, Mobject] = {"num_decimal_places": 1},
411: (8)             **kwargs,
412: (4)             ):
413: (8)             """
414: (8)             Will round/truncate the decimal places as per the provided config.
415: (8)             Parameters
416: (8)             -----
417: (8)             matrix
418: (12)             A numpy 2d array or list of lists
419: (8)             element_to_mobject
420: (12)             Mobject to use, by default DecimalNumber
421: (8)             element_to_mobject_config
422: (12)             Config for the desired mobject, by default {"num_decimal_places": 1}
423: (8)             """
424: (8)             super().__init__(
425: (12)             matrix,
426: (12)             element_to_mobject=element_to_mobject,
427: (12)             element_to_mobject_config=element_to_mobject_config,
428: (12)             **kwargs,
429: (8)             )
430: (0)             class IntegerMatrix(Matrix):
431: (4)             """A mobject that displays a matrix with integer entries on the screen.
432: (4)             Examples
433: (4)             -----
434: (4)             .. manim:: IntegerMatrixExample
435: (8)             :save_last_frame:
436: (8)             class IntegerMatrixExample(Scene):
437: (12)             def construct(self):
438: (16)             m0 = IntegerMatrix(
439: (20)             [[3.7, 2], [42.2, 12]],
440: (20)             left_bracket="(",
441: (20)             right_bracket=")")
442: (16)             self.add(m0)
443: (4)             """
444: (4)             def __init__(
445: (8)             self, matrix: Iterable, element_to_mobject: Mobject = Integer,
446: (8)             **kwargs
447: (4)             ):
448: (8)             """
449: (8)             Will round if there are decimal entries in the matrix.
450: (8)             Parameters
451: (8)             -----
452: (12)             matrix
453: (8)             A numpy 2d array or list of lists
454: (12)             element_to_mobject
455: (8)             Mobject to use, by default Integer
456: (8)             """
457: (0)             super().__init__(matrix, element_to_mobject=element_to_mobject,
458: (4)             **kwargs)
459: (4)             class MobjectMatrix(Matrix):
460: (4)             """A mobject that displays a matrix of mobject entries on the screen.
461: (4)             Examples
462: (4)             -----
463: (4)             .. manim:: MobjectMatrixExample
464: (8)             :save_last_frame:
465: (8)             class MobjectMatrixExample(Scene):
466: (12)             def construct(self):
467: (16)             a = Circle().scale(0.3)
468: (16)             b = Square().scale(0.3)
469: (16)             c = MathTex("\pi").scale(2)
470: (16)             d = Star().scale(0.3)
471: (16)             m0 = MobjectMatrix([[a, b], [c, d]])
472: (16)             self.add(m0)
473: (4)             """
474: (4)             def __init__(self, matrix, element_to_mobject=lambda m: m, **kwargs):

```

```

473: (8)                     super().__init__(matrix, element_to_mobject=element_to_mobject,
474: (0)                         **kwargs)
475: (4)             def get_det_text(
476: (4)                 matrix: Matrix,
477: (4)                 determinant: int | str | None = None,
478: (4)                 background_rect: bool = False,
479: (4)                 initial_scale_factor: float = 2,
480: (0)             ):
481: (4)                 r"""Helper function to create determinant.
482: (4)                 Parameters
483: (4)                 -----
484: (4)                 matrix
485: (4)                     The matrix whose determinant is to be created
486: (4)                     determinant
487: (4)                     The value of the determinant of the matrix
488: (4)                 background_rect
489: (4)                     The background rectangle
490: (4)                 initial_scale_factor
491: (4)                     The scale of the text `det` w.r.t the matrix
492: (4)             Returns
493: (4)             -----
494: (4)             :class:`~.VGroup`
495: (4)                 A VGroup containing the determinant
496: (4)             Examples
497: (4)             -----
498: (8)             .. manim:: DeterminantOfAMatrix
499: (8)                 :save_last_frame:
500: (12)             class DeterminantOfAMatrix(Scene):
501: (16)                 def construct(self):
502: (20)                     matrix = Matrix([
503: (20)                         [2, 0],
504: (16)                         [-1, 1]
505: (16)                     ])
506: (28)                     det = get_det_text(matrix,
507: (28)                         determinant=3,
508: (16)                         initial_scale_factor=1)
509: (16)                     self.add(matrix)
510: (4)                     self.add(det)
511: (4)             """
512: (4)             parens = MathTex("(", ")")
513: (4)             parens.scale(initial_scale_factor)
514: (4)             parens.stretch_to_fit_height(matrix.height)
515: (4)             l_paren, r_paren = parens.split()
516: (4)             l_paren.next_to(matrix, LEFT, buff=0.1)
517: (4)             r_paren.next_to(matrix, RIGHT, buff=0.1)
518: (4)             det = Tex("det")
519: (4)             det.scale(initial_scale_factor)
520: (4)             det.next_to(l_paren, LEFT, buff=0.1)
521: (4)             if background_rect:
522: (8)                 det.add_background_rectangle()
523: (4)             det_text = VGroup(det, l_paren, r_paren)
524: (4)             if determinant is not None:
525: (8)                 eq = MathTex("=")
526: (8)                 eq.next_to(r_paren, RIGHT, buff=0.1)
527: (8)                 result = MathTex(str(determinant))
528: (8)                 result.next_to(eq, RIGHT, buff=0.2)
529: (4)                 det_text.add(eq, result)
530: (4)             return det_text

```

-----  
File 56 - mobject.py:

```

1: (0)                     """Base classes for objects that can be displayed."""
2: (0)                     from __future__ import annotations
3: (0)                     __all__ = ["Mobject", "Group", "override_animate"]
4: (0)                     import copy
5: (0)                     import inspect
6: (0)                     import itertools as it

```

```

7: (0)          import math
8: (0)          import operator as op
9: (0)          import random
10: (0)         import sys
11: (0)         import types
12: (0)         import warnings
13: (0)         from functools import partialmethod, reduce
14: (0)         from pathlib import Path
15: (0)         from typing import TYPE_CHECKING, Callable, Iterable, Literal
16: (0)         import numpy as np
17: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
18: (0)         from .. import config, logger
19: (0)         from ..constants import *
20: (0)         from ..utils.color import (
21: (4)             BLACK,
22: (4)             WHITE,
23: (4)             YELLOW_C,
24: (4)             ManimColor,
25: (4)             ParsableManimColor,
26: (4)             color_gradient,
27: (4)             interpolate_color,
28: (0)
29: (0)     )
30: (0)     from ..utils.exceptions import MultiAnimationOverrideException
31: (0)     from ..utils.iterables import list_update, remove_list_redundancies
32: (0)     from ..utils.paths import straight_path
33: (0)     from ..utils.space_ops import angle_between_vectors, normalize,
rotation_matrix
34: (0) if TYPE_CHECKING:
35: (4)     from typing_extensions import Self, TypeAlias
36: (4)     from manim.typing import (
37: (8)         FunctionOverride,
38: (8)         Image,
39: (8)         ManimFloat,
40: (8)         ManimInt,
41: (8)         MappingFunction,
42: (8)         PathFuncType,
43: (8)         Point3D,
44: (8)         Point3D_Array,
45: (8)         Vector3D,
46: (4)
47: (4)     )
48: (4)     from ..animation.animation import Animation
49: (4)     TimeBasedUpdater: TypeAlias = Callable[["Mobject", float], object]
50: (4)     NonTimeBasedUpdater: TypeAlias = Callable[["Mobject"], object]
51: (4)     Updater: TypeAlias = NonTimeBasedUpdater | TimeBasedUpdater
52: (0) class Mobject:
53: (4)     """Mathematical Object: base class for objects that can be displayed on
screen.
54: (4)     There is a compatibility layer that allows for
55: (4)     getting and setting generic attributes with ``get_*``
56: (4)     and ``set_*`` methods. See :meth:`set` for more details.
57: (4)     Attributes
58: (4)     -----
59: (4)     submobjects : List[:class:`~Mobject`]
60: (8)         The contained objects.
61: (4)     points : :class:`numpy.ndarray`
62: (8)         The points of the objects.
63: (4)     .. seealso::
64: (12)         :class:`~.VMobject`
65: (4)
66: (4)     animation_overrides = {}
67: (4)     @classmethod
68: (4)     def __init_subclass__(cls, **kwargs) -> None:
69: (8)         super().__init_subclass__(**kwargs)
70: (8)         cls.animation_overrides: dict[
71: (12)             type[Animation],
72: (8)             FunctionOverride,
73: (8)         ] = {}
74: (8)         cls._add_intrinsic_animation_overrides()
75: (8)         cls._original_init_ = cls.__init__

```

```

74: (4)             def __init__(self,
75: (8)                 color: ParsableManimColor | list[ParsableManimColor] = WHITE,
76: (8)                 name: str | None = None,
77: (8)                 dim: int = 3,
78: (8)                 target=None,
79: (8)                 z_index: float = 0,
80: (8)
81: (4)             ) -> None:
82: (8)                 self.name = self.__class__.__name__ if name is None else name
83: (8)                 self.dim = dim
84: (8)                 self.target = target
85: (8)                 self.z_index = z_index
86: (8)                 self.point_hash = None
87: (8)                 self.submobjects = []
88: (8)                 self.updaters: list[Updater] = []
89: (8)                 self.updating_suspended = False
90: (8)                 self.color = ManimColor.parse(color)
91: (8)                 self.reset_points()
92: (8)                 self.generate_points()
93: (8)                 self.init_colors()
94: (4)             @classmethod
95: (4)             def animation_override_for(
96: (8)                 cls,
97: (8)                 animation_class: type[Animation],
98: (4)             ) -> FunctionOverride | None:
99: (8)                 """Returns the function defining a specific animation override for
this class.

100: (8)             Parameters
101: (8)             -----
102: (8)             animation_class
103: (12)                 The animation class for which the override function should be
returned.

104: (8)             Returns
105: (8)             -----
106: (8)             Optional[Callable[[Mobject, ...], Animation]]
107: (12)                 The function returning the override animation or ``None`` if no
such animation
108: (12)                 override is defined.

109: (8)             """
110: (8)                 if animation_class in cls.animation_overrides:
111: (12)                     return cls.animation_overrides[animation_class]
112: (8)                 return None
113: (4)             @classmethod
114: (4)             def _add_intrinsic_animation_overrides(cls) -> None:
115: (8)                 """Initializes animation overrides marked with the
:func:`~.override_animation` decorator.

116: (8)
117: (8)
118: (8)
119: (12)
120: (16)
121: (12)
122: (12)
123: (16)
124: (16)
125: (4)
126: (4)
127: (8)
128: (8)
129: (8)
130: (4)
131: (8)
132: (8)
133: (8)
134: (8)
135: (8)
136: (12)
137: (8)
138: (12)
:func:`~.override_animation`:
decorator.

118: (8)
119: (12)
120: (16)
121: (12)
122: (12)
123: (16)
124: (16)
125: (4)
126: (4)
127: (8)
128: (8)
129: (8)
130: (4)
131: (8)
132: (8)
133: (8)
134: (8)
135: (8)
136: (12)
137: (8)
138: (12)
for method_name in dir(cls):
    if method_name.startswith("__"):
        continue
    method = getattr(cls, method_name)
    if hasattr(method, "_override_animation"):
        animation_class = method._override_animation
        cls.add_animation_override(animation_class, method)

@classmethod
def add_animation_override(
    cls,
    animation_class: type[Animation],
    override_func: FunctionOverride,
) -> None:
    """Add an animation override.
This does not apply to subclasses.

Parameters
-----
animation_class
    The animation type to be overridden
override_func
    The function returning an animation replacing the default

```

animation. It gets

```

139: (12)           passed the parameters given to the animation constructor.
140: (8)           Raises
141: (8)
142: (8)
143: (12)           MultiAnimationOverrideException
144: (8)           If the overridden animation was already overridden.
145: (8)
146: (12)           if animation_class not in cls.animation_overrides:
147: (8)               cls.animation_overrides[animation_class] = override_func
148: (12)           else:
149: (16)               raise MultiAnimationOverrideException(
150: (16)                   f"The animation {animation_class.__name__} for "
151: (16)                   f"{cls.__name__} is overridden by more than one method: "
152: (16)                   f"{cls.animation_overrides[animation_class].__qualname__} and "
153: (12)                   f"{override_func.__qualname__}.",
154: (4)           )
155: (4)           @classmethod
156: (8)           def set_default(cls, **kwargs) -> None:
157: (8)               """Sets the default values of keyword arguments.
158: (8)               If this method is called without any additional keyword
159: (8)               arguments, the original default values of the initialization
160: (8)               method of this class are restored.
161: (8)           Parameters
162: (8)
163: (12)           kwargs
164: (12)               Passing any keyword argument will update the default
165: (12)               values of the keyword arguments of the initialization
166: (8)               function of this class.
167: (8)
168: (8)
169: (12)
170: (12)
171: (12)
172: (12)
173: (12)
174: (12)
175: (12)
176: (8)
177: (12)
178: (12)
179: (12)
180: (16)
181: (20)
182: (20)
183: (20)
184: (8)
185: (8)
186: (12)
187: (8)
188: (12)
189: (4)
190: (4)
191: (8)
192: (8)
applying
193: (8)
194: (8)
square,
195: (8)
196: (8)
197: (8)
198: (12)
199: (8)
200: (12)
201: (12)
202: (12)
203: (12)

::: >>> from manim import Square, GREEN
::: >>> Square.set_default(color=GREEN, fill_opacity=0.25)
::: >>> s = Square(); s.color, s.fill_opacity
::: (ManimColor('#83C167'), 0.25)
::: >>> Square.set_default()
::: >>> s = Square(); s.color, s.fill_opacity
::: (ManimColor('#FFFFFF'), 0.0)
.. manim:: ChangedDefaultTextColor
::: :save_last_frame:
config.background_color = WHITE
class ChangedDefaultTextColor(Scene):
    def construct(self):
        Text.set_default(color=BLACK)
        self.add(Text("Changing default values is easy!"))
        Text.set_default(color=WHITE)
    """
if kwargs:
    cls.__init__ = partialmethod(cls.__init__, **kwargs)
else:
    cls.__init__ = cls._original_init__
@property
def animate(self) -> _AnimationBuilder | Self:
    """Used to animate the application of any method of :code:`self`.
    Any method called on :code:`animate` is converted to an animation of
    that method on the mobject itself.
    For example, :code:`square.animate.set_fill(WHITE)` sets the fill color of a
    while :code:`square.animate.set_fill(WHITE)` animates this action.
    Multiple methods can be put in a single animation once via chaining:
    :::
        self.play(my_mobject.animate.shift(RIGHT).rotate(PI))
.. warning::
    Passing multiple animations for the same :class:`Mobject` in one
    call to :meth:`~Scene.play` is discouraged and will most likely
    not work properly. Instead of writing an animation like
    :::
```

```

204: (16)                         self.play(my_mobject.animate.shift(RIGHT),
my_mobject.animate.rotate(PI))
205: (12)                         make use of method chaining.
206: (8)                          Keyword arguments that can be passed to :meth:`.Scene.play` can be
passed
207: (8)                         directly after accessing ```.animate```, like so::
208: (12)                         self.play(my_mobject.animate(rate_func=linear).shift(RIGHT))
209: (8)                         This is especially useful when animating simultaneous ```.animate```
calls that
210: (8)                         you want to behave differently::
211: (12)                         self.play(
212: (16)                           mobject1.animate(run_time=2).rotate(PI),
213: (16)                           mobject2.animate(rate_func=there_and_back).shift(RIGHT),
214: (12)                           )
215: (8) ..seealso::
216: (12)   :func:`override_animate`
217: (8) Examples
218: (8)
219: (8) .. manim:: AnimateExample
220: (12)   class AnimateExample(Scene):
221: (16)     def construct(self):
222: (20)       s = Square()
223: (20)       self.play(Create(s))
224: (20)       self.play(s.animate.shift(RIGHT))
225: (20)       self.play(s.animate.scale(2))
226: (20)       self.play(s.animate.rotate(PI / 2))
227: (20)       self.play(Uncreate(s))
228: (8) .. manim:: AnimateChainExample
229: (12)   class AnimateChainExample(Scene):
230: (16)     def construct(self):
231: (20)       s = Square()
232: (20)       self.play(Create(s))
233: (20)       self.play(s.animate.shift(RIGHT).scale(2).rotate(PI / 2))
234: (20)       self.play(Uncreate(s))
235: (8) .. manim:: AnimateWithArgsExample
236: (12)   class AnimateWithArgsExample(Scene):
237: (16)     def construct(self):
238: (20)       s = Square()
239: (20)       c = Circle()
240: (20)       VGroup(s, c).arrange(RIGHT, buff=2)
241: (20)       self.add(s, c)
242: (20)       self.play(
243: (24)         s.animate(run_time=2).rotate(PI / 2),
244: (24)         c.animate(rate_func=there_and_back).shift(RIGHT),
245: (20)       )
246: (8) .. warning::
247: (12)   ```.animate```
248: (13)   will interpolate the :class:`~.Mobject` between its points prior
to
249: (13)   This may
250: (13)   result in unexpected behavior when attempting to interpolate
along paths,
251: (13)   or rotations.
252: (13)   If you want animations to consider the points between, consider
using
253: (13)     :class:`~.ValueTracker` with updaters instead.
254: (8)
255: (8)     """
256: (4)   return _AnimationBuilder(self)
257: (8)   def __deepcopy__(self, clone_from_id) -> Self:
258: (8)     cls = self.__class__
259: (8)     result = cls.__new__(cls)
260: (8)     clone_from_id[id(self)] = result
261: (12)     for k, v in self.__dict__.items():
262: (8)       setattr(result, k, copy.deepcopy(v, clone_from_id))
263: (8)     result.original_id = str(id(self))
264: (4)     return result
265: (8)   def __repr__(self) -> str:
266: (8)     return str(self.name)

```

```
266: (4)
267: (8)
268: (8)
269: (4)
270: (8)
271: (8)
272: implemented by
273: (8)
274: (4)
275: (8)
276: (8)
277: implemented by
278: (8)
279: (4)
280: (8)
281: (8)
282: (8)
283: (8)
284: (8)
285: (8)
286: (12)
287: (8)
288: (8)
289: (8)
290: (12)
291: (8)
292: (8)
293: (8)
294: (12)
295: (8)
296: (12)
297: :class:`Mobject` .
298: (8)
299: (8)
300: (8)
301: (8)
302: (8)
303: (8)
304: (8)
305: (8)
306: (8)
307: (8)
308: (8)
309: (8)
310: (12)
311: (12)
312: (12)
313: (8)
314: (12)
315: (12)
316: (12)
317: (8)
318: (12)
319: (12)
320: (12)
321: (12)
322: (8)
323: (8)
324: (12)
325: (12)
326: (12)
327: (12)
328: (12)
329: (12)
330: (12)
331: (8)

def reset_points(self) -> None:
    """Sets :attr:`points` to be an empty array."""
    self.points = np.zeros((0, self.dim))

def init_colors(self) -> None:
    """Initializes the colors.
    Gets called upon creation. This is an empty method that can be
    implemented by subclasses.
    """

def generate_points(self) -> None:
    """Initializes :attr:`points` and therefore the shape.
    Gets called upon creation. This is an empty method that can be
    implemented by subclasses.
    """

def add(self, *mobjects: Mobject) -> Self:
    """Add mobjects as submobjects.
    The mobjects are added to :attr:`submobjects`.
    Subclasses of mobject may implement ``+`` and ``+=`` dunder methods.
    Parameters
    -----
    mobjects
        The mobjects to add.
    Returns
    -----
    :class:`Mobject`
        ``self``
    Raises
    -----
    :class:`ValueError`
        When a mobject tries to add itself.
    :class:`TypeError`
        When trying to add an object that is not an instance of
    Notes
    -----
    A mobject cannot contain itself, and it cannot contain a submobject
    more than once. If the parent mobject is displayed, the newly-added
    submobjects will also be displayed (i.e. they are automatically added
    to the parent Scene).
    See Also
    -----
    :meth:`remove`
    :meth:`add_to_back`
    Examples
    -----
    ::

        >>> outer = Mobject()
        >>> inner = Mobject()
        >>> outer = outer.add(inner)
    Duplicates are not added again::
        >>> outer = outer.add(inner)
        >>> len(outer.submobjects)
        1
    Adding an object to itself raises an error::
        >>> outer.add(outer)
        Traceback (most recent call last):
        ...
        ValueError: Mobject cannot contain self
    A given mobject cannot be added as a submobject
    twice to some parent::
        >>> parent = Mobject(name="parent")
        >>> child = Mobject(name="child")
        >>> parent.add(child, child)
        [...] WARNING ...
        parent
        >>> parent.submobjects
        [child]
    """
```

```

332: (8)             for m in mobjects:
333: (12)             if not isinstance(m, Mobject):
334: (16)                 raise TypeError("All submobjects must be of type Mobject")
335: (12)             if m is self:
336: (16)                 raise ValueError("Mobject cannot contain self")
337: (8)             unique_mobjects = remove_list_redundancies(mobjects)
338: (8)             if len(mobjects) != len(unique_mobjects):
339: (12)                 logger.warning(
340: (16)                     "Attempted adding some Mobject as a child more than once, "
341: (16)                     "this is not possible. Repetitions are ignored.",
342: (12)                 )
343: (8)             self.submobjects = list_update(self.submobjects, unique_mobjects)
344: (8)         return self
345: (4)     def insert(self, index: int, mobject: Mobject) -> None:
346: (8)         """Inserts a mobject at a specific position into self.submobjects
347: (8)         Effectively just calls ``self.submobjects.insert(index, mobject)``,
348: (8)         where ``self.submobjects`` is a list.
349: (8)         Highly adapted from ``Mobject.add``.
350: (8)     Parameters
351: (8)     -----
352: (8)     index
353: (12)         The index at which
354: (8)     mobject
355: (12)         The mobject to be inserted.
356: (8)     """
357: (8)     if not isinstance(mobject, Mobject):
358: (12)         raise TypeError("All submobjects must be of type Mobject")
359: (8)     if mobject is self:
360: (12)         raise ValueError("Mobject cannot contain self")
361: (8)     self.submobjects.insert(index, mobject)
362: (4)     def __add__(self, mobject: Mobject):
363: (8)         raise NotImplementedError
364: (4)     def __iadd__(self, mobject: Mobject):
365: (8)         raise NotImplementedError
366: (4)     def add_to_back(self, *mobjects: Mobject) -> Self:
367: (8)         """Add all passed mobjects to the back of the submobjects.
368: (8)         If :attr:`submobjects` already contains the given mobjects, they just
369: (8)         to the back instead.
370: (8)     Parameters
371: (8)     -----
372: (8)     mobjects
373: (12)         The mobjects to add.
374: (8)     Returns
375: (8)     -----
376: (8)     :class:`Mobject`
377: (12)         ``self``
378: (8)     .. note::
379: (12)         Technically, this is done by adding (or moving) the mobjects to
380: (12)         the head of :attr:`submobjects`. The head of this list is rendered
381: (12)         first, which places the corresponding mobjects behind the
382: (12)         subsequent list members.
383: (8)     Raises
384: (8)     -----
385: (8)     :class:`ValueError`
386: (12)         When a mobject tries to add itself.
387: (8)     :class:`TypeError`
388: (12)         When trying to add an object that is not an instance of
389: (8)         :class:`Mobject`.
390: (8)     Notes
391: (8)     -----
392: (8)         A mobject cannot contain itself, and it cannot contain a submobject
393: (8)         more than once. If the parent mobject is displayed, the newly-added
394: (8)         submobjects will also be displayed (i.e. they are automatically added
395: (8)         to the parent Scene).
396: (8)     See Also
397: (8)     -----
398: (8)     :meth:`remove`
398: (8)     :meth:`add`
```

```

399: (8)
400: (8)
401: (12)
402: (8)
403: (12)
404: (16)
405: (8)
406: (8)
407: (8)
408: (4)
409: (8)
410: (8)
411: (8)
412: (8)
413: (8)
414: (8)
415: (12)
416: (8)
417: (8)
418: (8)
419: (12)
420: (8)
421: (8)
422: (8)
423: (8)
424: (8)
425: (12)
426: (16)
427: (8)
428: (4)
429: (8)
430: (4)
431: (8)
432: (4)
433: (8)
434: (8)
435: (8)
436: (8)
437: (8)
438: (8)
439: (8)
440: (12)
441: (12)
442: (12)
443: (12)
444: (12)
445: (12)
446: (12)
447: (8)
448: (8)
449: (8)
450: (8)
451: (12)
452: (12)
453: (12)
454: (12)
455: (8)
456: (8)
457: (8)
458: (12)
459: (8)
460: (8)
461: (8)
462: (12)
463: (8)
464: (8)
465: (8)
466: (12)
467: (12)

        """
        if self in mobjects:
            raise ValueError("A mobject shouldn't contain itself")
        for mobject in mobjects:
            if not isinstance(mobject, Mobject):
                raise TypeError("All submobjects must be of type Mobject")
        self.remove(*mobjects)
        self.submobjects = list(dict.fromkeys(mobjects)) + self.submobjects
        return self

    def remove(self, *mobjects: Mobject) -> Self:
        """Remove :attr:`submobjects`.

        The mobjects are removed from :attr:`submobjects`, if they exist.
        Subclasses of mobject may implement ``-`` and ``-=`` dunder methods.

        Parameters
        -----
        mobjects
            The mobjects to remove.

        Returns
        -----
        :class:`Mobject`
            ``self``

        See Also
        -----
        :meth:`add`

        """
        for mobject in mobjects:
            if mobject in self.submobjects:
                self.submobjects.remove(mobject)
        return self

    def __sub__(self, other):
        raise NotImplementedError

    def __isub__(self, other):
        raise NotImplementedError

    def set(self, **kwargs) -> Self:
        """Sets attributes.

        I.e. ``my_mobject.set(foo=1)`` applies ``my_mobject.foo = 1``.
        This is a convenience to be used along with :attr:`animate` to
        animate setting attributes.

        In addition to this method, there is a compatibility
        layer that allows ``get_*`` and ``set_*`` methods to
        get and set generic attributes. For instance::
            >>> mob = Mobject()
            >>> mob.set_foo(0)
            Mobject
            >>> mob.get_foo()
            0
            >>> mob.foo
            0

        This compatibility layer does not interfere with any
        ``get_*`` or ``set_*`` methods that are explicitly
        defined.

        .. warning::
            This compatibility layer is for backwards compatibility
            and is not guaranteed to stay around. Where applicable,
            please prefer getting/setting attributes normally or with
            the :meth:`set` method.

        Parameters
        -----
        **kwargs
            The attributes and corresponding values to set.

        Returns
        -----
        :class:`Mobject`
            ``self``

        Examples
        -----
        ::

            >>> mob = Mobject()
            >>> mob.set(foo=0)

```

```

468: (12)             Mobject
469: (12)             >>> mob.foo
470: (12)             0
471: (8)             """
472: (8)             for attr, value in kwargs.items():
473: (12)                 setattr(self, attr, value)
474: (8)             return self
475: (4)             def __getattr__(self, attr: str) -> types.MethodType:
476: (8)                 if attr.startswith("get_"):
477: (12)                     to_get = attr[4:]
478: (12)                     def getter(self):
479: (16)                         warnings.warn(
480: (20)                             "This method is not guaranteed to stay around. Please
prefer "
481: (20)                             "getting the attribute normally.", 
482: (20)                             DeprecationWarning,
483: (20)                             stacklevel=2,
484: (16)                     )
485: (16)                     return getattr(self, to_get)
486: (12)             return types.MethodType(getter, self)
487: (8)             if attr.startswith("set_"):
488: (12)                 to_set = attr[4:]
489: (12)                 def setter(self, value):
490: (16)                     warnings.warn(
491: (20)                         "This method is not guaranteed to stay around. Please
prefer "
492: (20)                         "setting the attribute normally or with Mobject.set()", 
493: (20)                         DeprecationWarning,
494: (20)                         stacklevel=2,
495: (16)                     )
496: (16)                     setattr(self, to_set, value)
497: (16)                     return self
498: (12)             return types.MethodType(setter, self)
499: (8)             raise AttributeError(f"{type(self).__name__} object has no attribute
'{attr}'")
500: (4)             @property
501: (4)             def width(self) -> float:
502: (8)                 """The width of the mobject.
503: (8)                 Returns
504: (8)                 -----
505: (8)                 :class:`float`
506: (8)                 Examples
507: (8)                 -----
508: (8)                 .. manim:: WidthExample
509: (12)                     class WidthExample(Scene):
510: (16)                         def construct(self):
511: (20)                             decimal = DecimalNumber().to_edge(UP)
512: (20)                             rect = Rectangle(color=BLUE)
513: (20)                             rect_copy = rect.copy().set_stroke(GRAY, opacity=0.5)
514: (20)                             decimal.add_updater(lambda d: d.set_value(rect.width))
515: (20)                             self.add(rect_copy, rect, decimal)
516: (20)                             self.play(rect.animate.set(width=7))
517: (20)                             self.wait()
518: (8)             See also
519: (8)             -----
520: (8)             :meth:`length_over_dim`
521: (8)             """
522: (8)             return self.length_over_dim(0)
523: (4)             @width.setter
524: (4)             def width(self, value: float):
525: (8)                 self.scale_to_fit_width(value)
526: (4)             @property
527: (4)             def height(self) -> float:
528: (8)                 """The height of the mobject.
529: (8)                 Returns
530: (8)                 -----
531: (8)                 :class:`float`
532: (8)                 Examples
533: (8)                 -----

```

```

534: (8)          .. manim:: HeightExample
535: (12)         class HeightExample(Scene):
536: (16)         def construct(self):
537: (20)         decimal = DecimalNumber().to_edge(UP)
538: (20)         rect = Rectangle(color=BLUE)
539: (20)         rect_copy = rect.copy().set_stroke(GRAY, opacity=0.5)
540: (20)         decimal.add_updater(lambda d: d.set_value(rect.height))
541: (20)         self.add(rect_copy, rect, decimal)
542: (20)         self.play(rect.animate.set(height=5))
543: (20)         self.wait()
544: (8)         See also
545: (8)         -----
546: (8)         :meth:`length_over_dim`
547: (8)         """
548: (8)         return self.length_over_dim(1)
549: (4) @height.setter
550: (4)     def height(self, value: float):
551: (8)         self.scale_to_fit_height(value)
552: (4) @property
553: (4)     def depth(self) -> float:
554: (8)         """The depth of the mobject.
555: (8)     Returns
556: (8)     -----
557: (8)     :class:`float`
558: (8)     See also
559: (8)     -----
560: (8)     :meth:`length_over_dim`
561: (8)     """
562: (8)     return self.length_over_dim(2)
563: (4) @depth.setter
564: (4)     def depth(self, value: float):
565: (8)         self.scale_to_fit_depth(value)
566: (4)     def get_array_attrs(self) -> list[Literal["points"]]:
567: (8)         return ["points"]
568: (4)     def apply_over_attr_arrays(self, func: MappingFunction) -> Self:
569: (8)         for attr in self.get_array_attrs():
570: (12)             setattr(self, attr, func(getattr(self, attr)))
571: (8)         return self
572: (4)     def get_image(self, camera=None) -> Image:
573: (8)         if camera is None:
574: (12)             from ..camera.camera import Camera
575: (12)             camera = Camera()
576: (8)             camera.capture_mobject(self)
577: (8)             return camera.get_image()
578: (4)     def show(self, camera=None) -> None:
579: (8)         self.get_image(camera=camera).show()
580: (4)     def save_image(self, name: str | None = None) -> None:
581: (8)         """Saves an image of only this :class:`Mobject` at its position to a
582: (8)         file."""
583: (8)         self.get_image().save(
584: (12)             Path(config.get_dir("video_dir")).joinpath((name or str(self)) +
585: (8)         )
586: (4)     def copy(self) -> Self:
587: (8)         """Create and return an identical copy of the :class:`Mobject`-
588: (8)         :attr:`submobjects` .
589: (8)         Returns
590: (8)         -----
591: (8)         :class:`Mobject`
592: (12)             The copy.
593: (8)         Note
594: (8)         -----
595: (8)         The clone is initially not visible in the Scene, even if the original
596: (8)         was.
597: (8)         """
598: (4)         return copy.deepcopy(self)
      def generate_target(self, use_deepcopy: bool = False) -> Self:

```

```

599: (8)                     self.target = None # Prevent unbounded linear recursion
600: (8)
601: (12)                    if use_deepcopy:
602: (8)                      self.target = copy.deepcopy(self)
603: (12)                    else:
604: (8)                      self.target = self.copy()
605: (4)                      return self.target
606: (8)
607: (8)                    def update(self, dt: float = 0, recursive: bool = True) -> Self:
608: (8)                      """Apply all updaters.
609: (8)                      Does nothing if updating is suspended.
610: (8)                      Parameters
611: (8)                      -----
612: (12)                      dt
613: (8)                          The parameter ``dt`` to pass to the update functions. Usually this
614: (8)                          is the
615: (8)                          time in seconds since the last call of ``update``.
616: (8)                          recursive
617: (8)                              Whether to recursively update all submobjects.
618: (12)                          Returns
619: (8)
620: (8)
621: (8)                          -----
622: (8)                          :meth:`add_updater`
623: (8)                          :meth:`get_updaters`
624: (8)                          """
625: (12)                    if self.updating_suspended:
626: (8)                      return self
627: (12)                    for updater in self.updaters:
628: (16)                      if "dt" in inspect.signature(updater).parameters:
629: (12)                        updater(self, dt)
630: (16)                      else:
631: (8)                        updater(self)
632: (12)                    if recursive:
633: (16)                      for submob in self.submobjects:
634: (8)                        submob.update(dt, recursive)
635: (4)                    return self
636: (8)
637: (8)                    def get_time_based_updaters(self) -> list[TimeBasedUpdater]:
638: (8)                      """Return all updaters using the ``dt`` parameter.
639: (8)                      The updaters use this parameter as the input for difference in time.
640: (8)                      Returns
641: (12)                      -----
642: (8)                      List[:class:`Callable`]
643: (8)                          The list of time based updaters.
644: (8)
645: (8)                      See Also
646: (8)                      -----
647: (8)                      :meth:`get_updaters`
648: (12)                      :meth:`has_time_based_updater`
649: (12)
650: (12)                    """
651: (8)                    return [
652: (12)                      updater
653: (12)                        for updater in self.updaters
654: (12)                        if "dt" in inspect.signature(updater).parameters
655: (8)                    ]
656: (8)
657: (12)                    def has_time_based_updater(self) -> bool:
658: (12)                      """Test if ``self`` has a time based updater.
659: (8)                      Returns
660: (8)                      -----
661: (8)                      :class:`bool`
662: (8)                          ``True`` if at least one updater uses the ``dt`` parameter,
663: (8)                          ``False`` otherwise.
664: (12)                      See Also
665: (8)                      -----
666: (8)                      :meth:`get_time_based_updaters`
667: (8)
668: (8)                    return any(
669: (12)                      "dt" in inspect.signature(updater).parameters for updater in
670: (12)                      self.updaters

```

```

665: (8) )
666: (4)     def get_updaters(self) -> list[Updater]:
667: (8)         """Return all updaters.
668: (8)         Returns
669: (8)         -----
670: (8)         List[:class:`Callable`]
671: (12)             The list of updaters.
672: (8)         See Also
673: (8)         -----
674: (8)             :meth:`add_updater`
675: (8)             :meth:`get_time_based_updaters`
676: (8)             """
677: (8)         return self.updaters
678: (4)     def get_family_updaters(self) -> list[Updater]:
679: (8)         return list(it.chain(*(sm.get_updaters() for sm in
self.get_family())))
680: (4)     def add_updater(
681: (8)         self,
682: (8)         update_function: Updater,
683: (8)         index: int | None = None,
684: (8)         call_updater: bool = False,
685: (4)     ) -> Self:
686: (8)         """Add an update function to this mobject.
687: (8)         Update functions, or updaters in short, are functions that are applied
to the
688: (8)         Mobject in every frame.
689: (8)         Parameters
690: (8)         -----
691: (8)         update_function
692: (12)             The update function to be added.
693: (12)             Whenever :meth:`update` is called, this update function gets
called using
694: (12)
695: (12)             ``self`` as the first parameter.
parameter,
696: (12)             The updater can have a second parameter ``dt``. If it uses this
the time
697: (12)
698: (8)
699: (12)
``self.updaters``.
700: (12)
end.
701: (8)
702: (12)
updater will
703: (12)
704: (8)
705: (8)
706: (8)
707: (12)
708: (8)
709: (8)
710: (8)
711: (12)
712: (16)
713: (20)
714: (24)
715: (24)
716: (20)
717: (20)
718: (20)
719: (20)
720: (20)
run_time=TAU, rate_func=linear))
721: (8)
722: (12)
723: (16)
724: (20)
.. manim:: NextToUpdater
    class NextToUpdater(Scene):
        def construct(self):
            def dot_position(mobject):
                mobject.set_value(dot.get_center()[0])
                mobject.next_to(dot)
            dot = Dot(RIGHT*3)
            label = DecimalNumber()
            label.add_updater(dot_position)
            self.add(dot, label)
            self.play(Rotating(dot, about_point=ORIGIN, angle=TAU,

```

```

725: (20)                                     square.add_updater(lambda mobject, dt:
mobject.rotate(dt*90*DEGREES))                  self.add(square)
726: (20)                                     self.wait(2)
727: (20)                                     See also
728: (8)                                     -----
729: (8)                                     :meth:`get_updaters`
730: (8)                                     :meth:`remove_updater`
731: (8)                                     :class:`~.UpdateFromFunc`
732: (8)                                     """
733: (8)                                     if index is None:
734: (8)                                         self.updaters.append(update_function)
735: (12)                                     else:
736: (8)                                         self.updaters.insert(index, update_function)
737: (12)                                     if call_updater:
738: (8)                                         parameters = inspect.signature(update_function).parameters
739: (12)                                         if "dt" in parameters:
740: (12)                                             update_function(self, 0)
741: (16)                                         else:
742: (12)                                             update_function(self)
743: (16)                                     return self
744: (8)                                     def remove_updater(self, update_function: Updater) -> Self:
745: (4)                                         """Remove an updater.
746: (8)                                         If the same updater is applied multiple times, every instance gets
747: (8)                                         removed.
748: (8)                                         Parameters
749: (8)                                         -----
750: (8)                                         update_function
751: (12)                                         The update function to be removed.
752: (8)                                         Returns
753: (8)                                         -----
754: (8)                                         :class:`Mobject`  

755: (12)                                         ``self``
756: (8)                                         See also
757: (8)                                         -----
758: (8)                                         :meth:`clear_updaters`
759: (8)                                         :meth:`add_updater`
760: (8)                                         :meth:`get_updaters`
761: (8)                                         """
762: (8)                                         while update_function in self.updaters:
763: (12)                                             self.updaters.remove(update_function)
764: (8)                                         return self
765: (4)                                     def clear_updaters(self, recursive: bool = True) -> Self:
766: (8)                                         """Remove every updater.
767: (8)                                         Parameters
768: (8)                                         -----
769: (8)                                         recursive
770: (12)                                         Whether to recursively call ``clear_updaters`` on all submobjects.
771: (8)                                         Returns
772: (8)                                         -----
773: (8)                                         :class:`Mobject`  

774: (12)                                         ``self``
775: (8)                                         See also
776: (8)                                         -----
777: (8)                                         :meth:`remove_updater`
778: (8)                                         :meth:`add_updater`
779: (8)                                         :meth:`get_updaters`
780: (8)                                         """
781: (8)                                         self.updaters = []
782: (8)                                         if recursive:
783: (12)                                             for submob in self.submobjects:
784: (16)                                                 submob.clear_updaters()
785: (8)                                         return self
786: (4)                                     def match_updaters(self, mobject: Mobject) -> Self:
787: (8)                                         """Match the updaters of the given mobject.
788: (8)                                         Parameters
789: (8)                                         -----
790: (8)                                         mobject
791: (12)                                         The mobject whose updaters get matched.

```

```

792: (8)             Returns
793: (8)             -----
794: (8)             :class:`Mobject`  

795: (12)             ``self``
796: (8)             Note
797: (8)             -----
798: (8)             All updaters from submobjects are removed, but only updaters of the
given
799: (8)             mobject are matched, not those of it's submobjects.
800: (8)             See also
801: (8)             -----
802: (8)             :meth:`add_updater`
803: (8)             :meth:`clear_updaters`
804: (8)             """
805: (8)             self.clear_updaters()
806: (8)             for updater in mobject.get_updaters():
807: (12)             self.add_updater(updater)
808: (8)             return self
809: (4)             def suspend_updating(self, recursive: bool = True) -> Self:
810: (8)             """Disable updating from updaters and animations.
811: (8)             Parameters
812: (8)             -----
813: (8)             recursive
814: (12)             Whether to recursively suspend updating on all submobjects.
815: (8)             Returns
816: (8)             -----
817: (8)             :class:`Mobject`  

818: (12)             ``self``
819: (8)             See also
820: (8)             -----
821: (8)             :meth:`resume_updating`
822: (8)             :meth:`add_updater`
823: (8)             """
824: (8)             self.updating_suspended = True
825: (8)             if recursive:
826: (12)             for submob in self.submobjects:
827: (16)             submob.suspend_updating(recursive)
828: (8)             return self
829: (4)             def resume_updating(self, recursive: bool = True) -> Self:
830: (8)             """Enable updating from updaters and animations.
831: (8)             Parameters
832: (8)             -----
833: (8)             recursive
834: (12)             Whether to recursively enable updating on all submobjects.
835: (8)             Returns
836: (8)             -----
837: (8)             :class:`Mobject`  

838: (12)             ``self``
839: (8)             See also
840: (8)             -----
841: (8)             :meth:`suspend_updating`
842: (8)             :meth:`add_updater`
843: (8)             """
844: (8)             self.updating_suspended = False
845: (8)             if recursive:
846: (12)             for submob in self.submobjects:
847: (16)             submob.resume_updating(recursive)
848: (8)             self.update(dt=0, recursive=recursive)
849: (8)             return self
850: (4)             def apply_to_family(self, func: Callable[[Mobject], None]) -> None:
851: (8)             """Apply a function to ``self`` and every submobject with points
recursively.
852: (8)             Parameters
853: (8)             -----
854: (8)             func
855: (12)             The function to apply to each mobject. ``func`` gets passed the
respective
856: (12)             (sub)mobject as parameter.
857: (8)             Returns

```

```

858: (8)           -----
859: (8)           :class:`Mobject`  
     ``self``  
860: (12)          See also  
861: (8)           -----  
862: (8)           :meth:`family_members_with_points`  
863: (8)           """  
864: (8)           for mob in self.family_members_with_points():  
865: (8)             func(mob)  
866: (12)          def shift(self, *vectors: Vector3D) -> Self:  
867: (4)            """Shift by the given vectors.  
868: (8)            Parameters  
869: (8)            -----  
870: (8)            vectors  
871: (8)              Vectors to shift by. If multiple vectors are given, they are added  
872: (12)             together.  
873: (12)            Returns  
874: (8)            -----  
875: (8)            :class:`Mobject`  
876: (8)             ``self``  
877: (12)            See also  
878: (8)            -----  
879: (8)            :meth:`move_to`  
880: (8)            """  
881: (8)            total_vector = reduce(op.add, vectors)  
882: (8)            for mob in self.family_members_with_points():  
883: (8)              mob.points = mob.points.astype("float")  
884: (12)             mob.points += total_vector  
885: (12)            return self  
886: (8)          def scale(self, scale_factor: float, **kwargs) -> Self:  
887: (4)            r"""Scale the size by a factor.  
888: (8)            Default behavior is to scale about the center of the mobject.  
889: (8)            Parameters  
890: (8)            -----  
891: (8)            scale_factor  
892: (8)              The scaling factor :math:`\alpha`. If :math:`0 < |\alpha| < 1`,  
893: (12)             will shrink, and for :math:`|\alpha| > 1` it will grow.  
894: (12)             if :math:`\alpha < 0`, the mobject is also flipped.  
895: (12)            kwargs  
896: (8)              Additional keyword arguments passed to  
897: (12)               :meth:`apply_points_function_about_point`.  
898: (12)            Returns  
899: (8)            -----  
900: (8)            :class:`Mobject`  
901: (8)             ``self``  
902: (12)            Examples  
903: (8)            -----  
904: (8)            .. manim:: MobjectScaleExample  
905: (8)              :save_last_frame:  
906: (12)              class MobjectScaleExample(Scene):  
907: (12)                def construct(self):  
908: (16)                  f1 = Text("F")  
909: (20)                  f2 = Text("F").scale(2)  
910: (20)                  f3 = Text("F").scale(0.5)  
911: (20)                  f4 = Text("F").scale(-1)  
912: (20)                  vgroup = VGroup(f1, f2, f3, f4).arrange(6 * RIGHT)  
913: (20)                  self.add(vgroup)  
914: (20)            See also  
915: (8)            -----  
916: (8)            :meth:`move_to`  
917: (8)            """  
918: (8)            self.apply_points_function_about_point(  
919: (8)              lambda points: scale_factor * points, **kwargs  
920: (12)            )  
921: (8)            return self  
922: (8)          def rotate_about_origin(self, angle: float, axis: Vector3D = OUT, axes=[])  
-> Self:  


```

```

924: (8)                                     """Rotates the :class:`~.Mobject` about the ORIGIN, which is at
[0,0,0]."""
925: (8)                                     return self.rotate(angle, axis, about_point=ORIGIN)
926: (4)                                     def rotate(
927: (8)                                         self,
928: (8)                                         angle: float,
929: (8)                                         axis: Vector3D = OUT,
930: (8)                                         about_point: Point3D | None = None,
931: (8)                                         **kwargs,
932: (4)                                     ) -> Self:
933: (8)                                     """Rotates the :class:`~.Mobject` about a certain point."""
934: (8)                                     rot_matrix = rotation_matrix(angle, axis)
935: (8)                                     self.apply_points_function_about_point(
936: (12)                                         lambda points: np.dot(points, rot_matrix.T), about_point, **kwargs
937: (8)
938: (8)                                     )
939: (4)                                     return self
940: (8)                                     def flip(self, axis: Vector3D = UP, **kwargs) -> Self:
941: (8)                                     """Flips/Mirrors an mobject about its center.
942: (8)                                     Examples
943: (8)                                     -----
944: (12)                                         .. manim:: FlipExample
945: (12)                                         :save_last_frame:
946: (16)                                         class FlipExample(Scene):
947: (20)                                         def construct(self):
948: (20)                                             s= Line(LEFT, RIGHT+UP).shift(4*LEFT)
949: (20)                                             self.add(s)
950: (20)                                             s2= s.copy().flip()
951: (8)                                             self.add(s2)
952: (8)
953: (4)                                     return self.rotate(TAU / 2, axis, **kwargs)
954: (8)                                     def stretch(self, factor: float, dim: int, **kwargs) -> Self:
955: (8)                                     def func(points):
956: (12)                                         points[:, dim] *= factor
957: (12)                                         return points
958: (8)                                         self.apply_points_function_about_point(func, **kwargs)
959: (8)
960: (4)                                     return self
961: (12)                                     if len(kwargs) == 0:
962: (8)                                         kwargs["about_point"] = ORIGIN
963: (12)                                         self.apply_points_function_about_point(
964: (8)                                             lambda points: np.apply_along_axis(function, 1, points), **kwargs
965: (8)
966: (8)                                         )
967: (8)
968: (8)                                         return self
969: (4)                                         def apply_function_to_submobject_positions(self, function:
MappingFunction) -> Self:
970: (8)                                         for submob in self.submobjects:
971: (12)                                             submob.apply_function_to_position(function)
972: (8)
973: (4)                                         return self
974: (8)                                         def apply_matrix(self, matrix, **kwargs) -> Self:
975: (12)                                         if ("about_point" not in kwargs) and ("about_edge" not in kwargs):
976: (8)                                             kwargs["about_point"] = ORIGIN
977: (8)                                         full_matrix = np.identity(self.dim)
978: (8)                                         matrix = np.array(matrix)
979: (8)                                         full_matrix[: matrix.shape[0], : matrix.shape[1]] = matrix
980: (12)                                         self.apply_points_function_about_point(
981: (8)                                             lambda points: np.dot(points, full_matrix.T), **kwargs
982: (8)
983: (4)                                         )
984: (8)
985: (4)                                         return self
986: (8)                                         def apply_complex_function(
987: (8)                                             self, function: Callable[[complex], complex], **kwargs
988: (8)                                         ) -> Self:
989: (8)                                         """Applies a complex function to a :class:`Mobject`.
The x and y Point3Ds correspond to the real and imaginary parts
respectively.
988: (8)                                         Example
989: (8)                                         -----

```

```

990: (8)          .. manim:: ApplyFuncExample
991: (12)         class ApplyFuncExample(Scene):
992: (16)         def construct(self):
993: (20)         circ = Circle().scale(1.5)
994: (20)         circ_ref = circ.copy()
995: (20)         circ.apply_complex_function(
996: (24)             lambda x: np.exp(x*1j)
997: (20)         )
998: (20)         t = ValueTracker(0)
999: (20)         circ.add_updater(
1000: (24)             lambda x:
x.become(circ_ref.copy().apply_complex_function(
1001: (28)                 lambda x: np.exp(x+t.get_value()*1j)
1002: (24)             ).set_color(BLUE)
1003: (20)         )
1004: (20)         self.add(circ_ref)
1005: (20)         self.play(TransformFromCopy(circ_ref, circ))
1006: (20)         self.play(t.animate.set_value(TAU), run_time=3)
1007: (8)         """
1008: (8)         def R3_func(point):
1009: (12)             x, y, z = point
1010: (12)             xy_complex = function(complex(x, y))
1011: (12)             return [xy_complex.real, xy_complex.imag, z]
1012: (8)             return self.apply_function(R3_func)
1013: (4)         def reverse_points(self) -> Self:
1014: (8)             for mob in self.family_members_with_points():
1015: (12)                 mob.apply_over_attr_arrays(lambda arr:
np.array(list(reversed(arr))))
1016: (8)             return self
1017: (4)         def repeat(self, count: int) -> Self:
1018: (8)             """This can make transition animations nicer"""
1019: (8)             def repeat_array(array):
1020: (12)                 return reduce(lambda a1, a2: np.append(a1, a2, axis=0), [array] *
count)
1021: (8)             for mob in self.family_members_with_points():
1022: (12)                 mob.apply_over_attr_arrays(repeat_array)
1023: (8)             return self
1024: (4)         def apply_points_function_about_point(
1025: (8)             self,
1026: (8)             func: MappingFunction,
1027: (8)             about_point: Point3D = None,
1028: (8)             about_edge=None,
1029: (4)         ) -> Self:
1030: (8)             if about_point is None:
1031: (12)                 if about_edge is None:
1032: (16)                     about_edge = ORIGIN
1033: (12)                     about_point = self.get_critical_point(about_edge)
1034: (8)             for mob in self.family_members_with_points():
1035: (12)                 mob.points -= about_point
1036: (12)                 mob.points = func(mob.points)
1037: (12)                 mob.points += about_point
1038: (8)             return self
1039: (4)         def pose_at_angle(self, **kwargs):
1040: (8)             self.rotate(TAU / 14, RIGHT + UP, **kwargs)
1041: (8)             return self
1042: (4)         def center(self) -> Self:
1043: (8)             """Moves the center of the mobject to the center of the scene.
1044: (8)             Returns
1045: (8)             -----
1046: (8)             :class:`.Mobject`
1047: (12)                 The centered mobject.
1048: (8)             """
1049: (8)             self.shift(-self.get_center())
1050: (8)             return self
1051: (4)         def align_on_border(
1052: (8)             self, direction: Vector3D, buff: float =
DEFAULT_MOBJECT_TO_EDGE_BUFFER
1053: (4)         ) -> Self:
1054: (8)             """Direction just needs to be a vector pointing towards side or

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1055: (8)                 corner in the 2d plane.
1056: (8)
1057: (8)
1058: (12)
1059: (12)
1060: (12)
1061: (8)
1062: (8)
1063: (8)
1064: (8)
1065: (8)
1066: (8)
1067: (4)             def to_corner(
1068: (8)                     self, corner: Vector3D = DL, buff: float =
DEFAULT_MOBJECT_TO_EDGE_BUFFER
1069: (4)             ) -> Self:
1070: (8)                     """Moves this :class:`~.Mobject` to the given corner of the screen.
1071: (8)                     Returns
1072: (8)                     -----
1073: (8)                     :class:`.Mobject`
1074: (12)                     The newly positioned mobject.
1075: (8)
1076: (8)
1077: (8)
1078: (12)
1079: (12)
1080: (16)             class ToCornerExample(Scene):
1081: (20)                     def construct(self):
1082: (20)                         c = Circle()
1083: (20)                         c.to_corner(UR)
1084: (20)                         t = Tex("To the corner!")
1085: (20)                         t2 = MathTex("x^3").shift(DOWN)
1086: (20)                         self.add(c,t,t2)
1087: (20)                         t.to_corner(DL, buff=0)
1088: (20)                         t2.to_corner(UL, buff=1.5)
1089: (8)
1090: (4)             def to_edge(
1091: (8)                     self, edge: Vector3D = LEFT, buff: float =
DEFAULT_MOBJECT_TO_EDGE_BUFFER
1092: (4)             ) -> Self:
1093: (8)                     """Moves this :class:`~.Mobject` to the given edge of the screen,
1094: (8)                     without affecting its position in the other dimension.
1095: (8)
1096: (8)
1097: (8)
1098: (12)
1099: (8)
1100: (8)
1101: (8)
1102: (12)
1103: (12)
1104: (16)             class ToEdgeExample(Scene):
1105: (20)                     def construct(self):
1106: (20)                         tex_top = Tex("I am at the top!")
1107: (20)                         tex_top.to_edge(UP)
1108: (20)                         tex_side = Tex("I am moving to the side!")
1109: (20)                         c = Circle().shift(2*DOWN)
1110: (20)                         self.add(tex_top, tex_side)
1111: (20)                         tex_side.to_edge(LEFT)
1112: (8)                         c.to_edge(RIGHT, buff=0)
1113: (8)
1114: (4)             def next_to(
1115: (8)                     self,
1116: (8)                     mobject_or_point: Mobject | Point3D,
1117: (8)                     direction: Vector3D = RIGHT,
1118: (8)                     buff: float = DEFAULT_MOBJECT_TO_MOBJECT_BUFFER,
1119: (8)                     aligned_edge: Vector3D = ORIGIN,
1120: (8)                     submobject_to_align: Mobject | None = None,
1121: (8)                     index_of_submobject_to_align: int | None = None,

```

```

1122: (8)             coor_mask: Vector3D = np.array([1, 1, 1]),
1123: (4)             ) -> Self:
1124: (8)                 """Move this :class:`~.Mobject` next to another's :class:`~.Mobject`"""
1125: (8)             Examples
1126: (8)             -----
1127: (8)             .. manim:: GeometricShapes
1128: (12)                 :save_last_frame:
1129: (12)                 class GeometricShapes(Scene):
1130: (16)                     def construct(self):
1131: (20)                         d = Dot()
1132: (20)                         c = Circle()
1133: (20)                         s = Square()
1134: (20)                         t = Triangle()
1135: (20)                         d.next_to(c, RIGHT)
1136: (20)                         s.next_to(c, LEFT)
1137: (20)                         t.next_to(c, DOWN)
1138: (20)                         self.add(d, c, s, t)
1139: (8)             """
1140: (8)             if isinstance(mobject_or_point, Mobject):
1141: (12)                 mob = mobject_or_point
1142: (12)                 if index_of_submobject_to_align is not None:
1143: (16)                     target_aligner = mob[index_of_submobject_to_align]
1144: (12)                 else:
1145: (16)                     target_aligner = mob
1146: (12)                 target_point = target_aligner.get_critical_point(aligned_edge +
direction)
1147: (8)
1148: (12)             else:
1149: (8)                 target_point = mobject_or_point
1150: (12)             if submobject_to_align is not None:
1151: (8)                 aligner = submobject_to_align
1152: (12)             elif index_of_submobject_to_align is not None:
1153: (8)                 aligner = self[index_of_submobject_to_align]
1154: (12)             else:
1155: (8)                 aligner = self
1156: (8)             point_to_align = aligner.get_critical_point(aligned_edge - direction)
1157: (8)             self.shift((target_point - point_to_align + buff * direction) *
coor_mask)
1158: (4)             return self
1159: (8)         def shift_onto_screen(self, **kwargs) -> Self:
1160: (8)             space_lengths = [config["frame_x_radius"], config["frame_y_radius"]]
1161: (12)             for vect in UP, DOWN, LEFT, RIGHT:
1162: (12)                 dim = np.argmax(np.abs(vect))
1163: (12)                 buff = kwargs.get("buff", DEFAULT_MOBJECT_TO_EDGE_BUFFER)
1164: (12)                 max_val = space_lengths[dim] - buff
1165: (12)                 edge_center = self.get_edge_center(vect)
1166: (16)                 if np.dot(edge_center, vect) > max_val:
1167: (8)                     self.to_edge(vect, **kwargs)
1168: (4)
1169: (8)             return self
1170: (12)             if self.get_left()[0] > config["frame_x_radius"]:
1171: (8)                 return True
1172: (12)             if self.get_right()[0] < -config["frame_x_radius"]:
1173: (8)                 return True
1174: (12)             if self.get_bottom()[1] > config["frame_y_radius"]:
1175: (8)                 return True
1176: (12)             if self.get_top()[1] < -config["frame_y_radius"]:
1177: (8)                 return True
1178: (4)             return False
1179: (8)         def stretch_about_point(self, factor: float, dim: int, point: Point3D) ->
Self:
1180: (4)             return self.stretch(factor, dim, about_point=point)
1181: (8)         def rescale_to_fit(
1182: (4)             self, length: float, dim: int, stretch: bool = False, **kwargs
1183: (4)         ) -> Self:
1184: (8)             old_length = self.length_over_dim(dim)
1185: (12)             if old_length == 0:
1186: (8)                 return self

```

```

1187: (12)             self.stretch(length / old_length, dim, **kwargs)
1188: (8)         else:
1189: (12)             self.scale(length / old_length, **kwargs)
1190: (8)         return self
1191: (4)     def scale_to_fit_width(self, width: float, **kwargs) -> Self:
1192: (8)         """Scales the :class:`~.Mobject` to fit a width while keeping
height/depth proportional.
1193: (8)             Returns
1194: (8)             -----
1195: (8)             :class:`Mobject`
1196: (12)                 ``self``
1197: (8)             Examples
1198: (8)             -----
1199: (8)         :::
1200: (12)             >>> from manim import *
1201: (12)             >>> sq = Square()
1202: (12)             >>> sq.height
1203: (12)             2.0
1204: (12)             >>> sq.scale_to_fit_width(5)
1205: (12)             Square
1206: (12)             >>> sq.width
1207: (12)             5.0
1208: (12)             >>> sq.height
1209: (12)             5.0
1210: (8)         """
1211: (8)         return self.rescale_to_fit(width, 0, stretch=False, **kwargs)
1212: (4)     def stretch_to_fit_width(self, width: float, **kwargs) -> Self:
1213: (8)         """Stretches the :class:`~.Mobject` to fit a width, not keeping
height/depth proportional.
1214: (8)             Returns
1215: (8)             -----
1216: (8)             :class:`Mobject`
1217: (12)                 ``self``
1218: (8)             Examples
1219: (8)             -----
1220: (8)         :::
1221: (12)             >>> from manim import *
1222: (12)             >>> sq = Square()
1223: (12)             >>> sq.height
1224: (12)             2.0
1225: (12)             >>> sq.stretch_to_fit_width(5)
1226: (12)             Square
1227: (12)             >>> sq.width
1228: (12)             5.0
1229: (12)             >>> sq.height
1230: (12)             2.0
1231: (8)         """
1232: (8)         return self.rescale_to_fit(width, 0, stretch=True, **kwargs)
1233: (4)     def scale_to_fit_height(self, height: float, **kwargs) -> Self:
1234: (8)         """Scales the :class:`~.Mobject` to fit a height while keeping
width/depth proportional.
1235: (8)             Returns
1236: (8)             -----
1237: (8)             :class:`Mobject`
1238: (12)                 ``self``
1239: (8)             Examples
1240: (8)             -----
1241: (8)         :::
1242: (12)             >>> from manim import *
1243: (12)             >>> sq = Square()
1244: (12)             >>> sq.width
1245: (12)             2.0
1246: (12)             >>> sq.scale_to_fit_height(5)
1247: (12)             Square
1248: (12)             >>> sq.height
1249: (12)             5.0
1250: (12)             >>> sq.width
1251: (12)             5.0
1252: (8)         """

```

```

1253: (8)             return self.rescale_to_fit(height, 1, stretch=False, **kwargs)
1254: (4)         def stretch_to_fit_height(self, height: float, **kwargs) -> Self:
1255: (8)             """Stretches the :class:`~.Mobject` to fit a height, not keeping
width/depth proportional.
1256: (8)             Returns
1257: (8)             -----
1258: (8)             :class:`Mobject`
1259: (12)             ``self``
1260: (8)         Examples
1261: (8)         -----
1262: (8)         :::
1263: (12)             >>> from manim import *
1264: (12)             >>> sq = Square()
1265: (12)             >>> sq.width
1266: (12)             2.0
1267: (12)             >>> sq.stretch_to_fit_height(5)
1268: (12)             Square
1269: (12)             >>> sq.height
1270: (12)             5.0
1271: (12)             >>> sq.width
1272: (12)             2.0
1273: (8)             """
1274: (8)             return self.rescale_to_fit(height, 1, stretch=True, **kwargs)
1275: (4)         def scale_to_fit_depth(self, depth: float, **kwargs) -> Self:
1276: (8)             """Scales the :class:`~.Mobject` to fit a depth while keeping
width/height proportional."""
1277: (8)             return self.rescale_to_fit(depth, 2, stretch=False, **kwargs)
1278: (4)         def stretch_to_fit_depth(self, depth: float, **kwargs) -> Self:
1279: (8)             """Stretches the :class:`~.Mobject` to fit a depth, not keeping
width/height proportional."""
1280: (8)             return self.rescale_to_fit(depth, 2, stretch=True, **kwargs)
1281: (4)         def set_coord(self, value, dim: int, direction: Vector3D = ORIGIN) ->
Self:
1282: (8)             curr = self.get_coord(dim, direction)
1283: (8)             shift_vect = np.zeros(self.dim)
1284: (8)             shift_vect[dim] = value - curr
1285: (8)             self.shift(shift_vect)
1286: (8)             return self
1287: (4)         def set_x(self, x: float, direction: Vector3D = ORIGIN) -> Self:
1288: (8)             """Set x value of the center of the :class:`~.Mobject` (`int` or
``float``)"""
1289: (8)             return self.set_coord(x, 0, direction)
1290: (4)         def set_y(self, y: float, direction: Vector3D = ORIGIN) -> Self:
1291: (8)             """Set y value of the center of the :class:`~.Mobject` (`int` or
``float``)"""
1292: (8)             return self.set_coord(y, 1, direction)
1293: (4)         def set_z(self, z: float, direction: Vector3D = ORIGIN) -> Self:
1294: (8)             """Set z value of the center of the :class:`~.Mobject` (`int` or
``float``)"""
1295: (8)             return self.set_coord(z, 2, direction)
1296: (4)         def space_out_submobjects(self, factor: float = 1.5, **kwargs) -> Self:
1297: (8)             self.scale(factor, **kwargs)
1298: (8)             for submob in self.submobjects:
1299: (12)                 submob.scale(1.0 / factor)
1300: (8)             return self
1301: (4)         def move_to(
1302: (8)             self,
1303: (8)             point_or_mobject: Point3D | Mobject,
1304: (8)             aligned_edge: Vector3D = ORIGIN,
1305: (8)             coor_mask: Vector3D = np.array([1, 1, 1]),
1306: (4)         ) -> Self:
1307: (8)             """Move center of the :class:`~.Mobject` to certain Point3D."""
1308: (8)             if isinstance(point_or_mobject, Mobject):
1309: (12)                 target = point_or_mobject.get_critical_point(aligned_edge)
1310: (8)             else:
1311: (12)                 target = point_or_mobject
1312: (8)             point_to_align = self.get_critical_point(aligned_edge)
1313: (8)             self.shift((target - point_to_align) * coor_mask)
1314: (8)             return self

```

```

1315: (4)             def replace(
1316: (8)               self, mobject: Mobject, dim_to_match: int = 0, stretch: bool = False
1317: (4)           ) -> Self:
1318: (8)               if not mobject.get_num_points() and not mobject.submobjects:
1319: (12)                   raise Warning("Attempting to replace mobject with no points")
1320: (8)               if stretch:
1321: (12)                   self.stretch_to_fit_width(mobject.width)
1322: (12)                   self.stretch_to_fit_height(mobject.height)
1323: (8)               else:
1324: (12)                   self.rescale_to_fit(
1325: (16)                       mobject.length_over_dim(dim_to_match),
1326: (16)                       dim_to_match,
1327: (16)                       stretch=False,
1328: (12)                   )
1329: (8)                   self.shift(mobject.get_center() - self.get_center())
1330: (8)               return self
1331: (4)             def surround(
1332: (8)               self,
1333: (8)               mobject: Mobject,
1334: (8)               dim_to_match: int = 0,
1335: (8)               stretch: bool = False,
1336: (8)               buff: float = MED_SMALL_BUFF,
1337: (4)           ) -> Self:
1338: (8)               self.replace(mobject, dim_to_match, stretch)
1339: (8)               length = mobject.length_over_dim(dim_to_match)
1340: (8)               self.scale((length + buff) / length)
1341: (8)               return self
1342: (4)             def put_start_and_end_on(self, start: Point3D, end: Point3D) -> Self:
1343: (8)               curr_start, curr_end = self.get_start_and_end()
1344: (8)               curr_vect = curr_end - curr_start
1345: (8)               if np.all(curr_vect == 0):
1346: (12)                   self.points = start
1347: (12)               return self
1348: (8)               target_vect = np.array(end) - np.array(start)
1349: (8)               axis = (
1350: (12)                   normalize(np.cross(curr_vect, target_vect))
1351: (12)                   if np.linalg.norm(np.cross(curr_vect, target_vect)) != 0
1352: (12)                   else OUT
1353: (8)
1354: (8)
1355: (12)
1356: (12)
1357: (8)
1358: (8)
1359: (12)
1360: (12)
1361: (12)
1362: (8)
1363: (8)
1364: (8)
1365: (4)             def add_background_rectangle(
1366: (8)               self, color: ParsableManimColor | None = None, opacity: float = 0.75,
**kwargs
1367: (4)           ) -> Self:
1368: (8)               """Add a BackgroundRectangle as submobject.
1369: (8)               The BackgroundRectangle is added behind other submobjects.
1370: (8)               This can be used to increase the mobjects visibility in front of a
noisy background.
1371: (8)               Parameters
1372: (8)               -----
1373: (8)               color
1374: (12)                   The color of the BackgroundRectangle
1375: (8)               opacity
1376: (12)                   The opacity of the BackgroundRectangle
1377: (8)               kwargs
1378: (12)                   Additional keyword arguments passed to the BackgroundRectangle
constructor
1379: (8)
1380: (8)               Returns
-----
```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1381: (8)           :class:`Mobject`  

1382: (12)          ``self``  

1383: (8)           See Also  

1384: (8)           -----  

1385: (8)           :meth:`add_to_back`  

1386: (8)           :class:`~.BackgroundRectangle`  

1387: (8)           """  

1388: (8)           from manim.mobject.geometry.shape_matchers import BackgroundRectangle  

1389: (8)           self.background_rectangle = BackgroundRectangle(  

1390: (12)          self, color=color, fill_opacity=opacity, **kwargs  

1391: (8)          )  

1392: (8)          self.add_to_back(self.background_rectangle)  

1393: (8)          return self  

1394: (4)           def add_background_rectangle_to_submobjects(self, **kwargs) -> Self:  

1395: (8)           for submobject in self.submobjects:  

1396: (12)          submobject.add_background_rectangle(**kwargs)  

1397: (8)           return self  

1398: (4)           def add_background_rectangle_to_family_members_with_points(self, **kwargs)  

-> Self:  

1399: (8)           for mob in self.family_members_with_points():  

1400: (12)          mob.add_background_rectangle(**kwargs)  

1401: (8)           return self  

1402: (4)           def set_color(  

1403: (8)           self, color: ParsableManimColor = YELLOW_C, family: bool = True  

1404: (4)           ) -> Self:  

1405: (8)           """Condition is function which takes in one arguments, (x, y, z).  

1406: (8)           Here it just recurses to submobjects, but in subclasses this  

1407: (8)           should be further implemented based on the the inner workings  

1408: (8)           of color  

1409: (8)           """  

1410: (8)           if family:  

1411: (12)          for submob in self.submobjects:  

1412: (16)          submob.set_color(color, family=family)  

1413: (8)           self.color = ManimColor.parse(color)  

1414: (8)           return self  

1415: (4)           def set_color_by_gradient(self, *colors: ParsableManimColor) -> Self:  

1416: (8)           """  

1417: (8)           Parameters  

1418: (8)           -----  

1419: (8)           colors  

1420: (12)          The colors to use for the gradient. Use like  

`set_color_by_gradient(RED, BLUE, GREEN)`.  

1421: (8)           self.color = ManimColor.parse(color)  

1422: (8)           return self  

1423: (8)           """  

1424: (8)           self.set_submobject_colors_by_gradient(*colors)  

1425: (8)           return self  

1426: (4)           def set_colors_by_radial_gradient(  

1427: (8)           self,  

1428: (8)           center: Point3D | None = None,  

1429: (8)           radius: float = 1,  

1430: (8)           inner_color: ParsableManimColor = WHITE,  

1431: (8)           outer_color: ParsableManimColor = BLACK,  

1432: (4)           ) -> Self:  

1433: (8)           self.set_submobject_colors_by_radial_gradient(  

1434: (12)          center,  

1435: (12)          radius,  

1436: (12)          inner_color,  

1437: (12)          outer_color,  

1438: (8)          )  

1439: (8)          return self  

1440: (4)           def set_submobject_colors_by_gradient(self, *colors:  

Iterable[ParsableManimColor]):  

1441: (8)           if len(colors) == 0:  

1442: (12)          raise ValueError("Need at least one color")  

1443: (8)           elif len(colors) == 1:  

1444: (12)          return self.set_color(*colors)  

1445: (8)           mobs = self.family_members_with_points()  

1446: (8)           new_colors = color_gradient(colors, len(mobs))

```

```

1447: (8)             for mob, color in zip(mobs, new_colors):
1448: (12)                 mob.set_color(color, family=False)
1449: (8)             return self
1450: (4)         def set_submobject_colors_by_radial_gradient(
1451: (8)             self,
1452: (8)             center: Point3D | None = None,
1453: (8)             radius: float = 1,
1454: (8)             inner_color: ParsableManimColor = WHITE,
1455: (8)             outer_color: ParsableManimColor = BLACK,
1456: (4)         ) -> Self:
1457: (8)             if center is None:
1458: (12)                 center = self.get_center()
1459: (8)             for mob in self.family_members_with_points():
1460: (12)                 t = np.linalg.norm(mob.get_center() - center) / radius
1461: (12)                 t = min(t, 1)
1462: (12)                 mob_color = interpolate_color(inner_color, outer_color, t)
1463: (12)                 mob.set_color(mob_color, family=False)
1464: (8)             return self
1465: (4)         def to_original_color(self) -> Self:
1466: (8)             self.set_color(self.color)
1467: (8)             return self
1468: (4)         def fade_to(
1469: (8)             self, color: ParsableManimColor, alpha: float, family: bool = True
1470: (4)         ) -> Self:
1471: (8)             if self.get_num_points() > 0:
1472: (12)                 new_color = interpolate_color(self.get_color(), color, alpha)
1473: (12)                 self.set_color(new_color, family=False)
1474: (8)             if family:
1475: (12)                 for submob in self.submobjects:
1476: (16)                     submob.fade_to(color, alpha)
1477: (8)             return self
1478: (4)         def fade(self, darkness: float = 0.5, family: bool = True) -> Self:
1479: (8)             if family:
1480: (12)                 for submob in self.submobjects:
1481: (16)                     submob.fade(darkness, family)
1482: (8)             return self
1483: (4)         def get_color(self) -> ManimColor:
1484: (8)             """Returns the color of the :class:`~.Mobject`"""
1485: (8)             Examples
1486: (8)             -----
1487: (8)             :::
1488: (12)                 >>> from manim import Square, RED
1489: (12)                 >>> Square(color=RED).get_color() == RED
1490: (12)                 True
1491: (8)             """
1492: (8)             return self.color
1493: (4)         def save_state(self) -> Self:
1494: (8)             """Save the current state (position, color & size). Can be restored
with :meth:`~.Mobject.restore`."""
1495: (8)             if hasattr(self, "saved_state"):
1496: (12)                 self.saved_state = None
1497: (8)             self.saved_state = self.copy()
1498: (8)             return self
1499: (4)         def restore(self) -> Self:
1500: (8)             """Restores the state that was previously saved with
:meth:`~.Mobject.save_state`."""
1501: (8)             if not hasattr(self, "saved_state") or self.saved_state is None:
1502: (12)                 raise Exception("Trying to restore without having saved")
1503: (8)             self.become(self.saved_state)
1504: (8)             return self
1505: (4)         def reduce_across_dimension(self, reduce_func: Callable, dim: int):
1506: (8)             """Find the min or max value from a dimension across all points in
this and submobjects."""
1507: (8)             assert dim >= 0 and dim <= 2
1508: (8)             if len(self.submobjects) == 0 and len(self.points) == 0:
1509: (12)                 return 0
1510: (8)             if len(self.points) == 0:
1511: (12)                 rv = None
1512: (8)             else:

```

```

1513: (12)             rv = reduce_func(self.points[:, dim])
1514: (8)              for mobj in self.submobjects:
1515: (12)                value = mobj.reduce_across_dimension(reduce_func, dim)
1516: (12)                if rv is None:
1517: (16)                  rv = value
1518: (12)                else:
1519: (16)                  rv = reduce_func([value, rv])
1520: (8)              return rv
1521: (4) def nonempty_submobjects(self) -> list[Self]:
1522: (8)    return [
1523: (12)      submob
1524: (12)      for submob in self.submobjects
1525: (12)      if len(submob.submobjects) != 0 or len(submob.points) != 0
1526: (8)    ]
1527: (4) def get_merged_array(self, array_attr: str) -> np.ndarray:
1528: (8)    """Return all of a given attribute from this mobject and all
submobjects.
1529: (8)    May contain duplicates; the order is in a depth-first (pre-order)
1530: (8)    traversal of the submobjects.
1531: (8)    """
1532: (8)    result = getattr(self, array_attr)
1533: (8)    for submob in self.submobjects:
1534: (12)      result = np.append(result, submob.get_merged_array(array_attr),
axis=0)
1535: (8)    return result
1536: (4) def get_all_points(self) -> Point3D_Array:
1537: (8)    """Return all points from this mobject and all submobjects.
1538: (8)    May contain duplicates; the order is in a depth-first (pre-order)
1539: (8)    traversal of the submobjects.
1540: (8)    """
1541: (8)    return self.get_merged_array("points")
1542: (4) def get_points_defining_boundary(self) -> Point3D_Array:
1543: (8)    return self.get_all_points()
1544: (4) def get_num_points(self) -> int:
1545: (8)    return len(self.points)
1546: (4) def get_extremum_along_dim(
1547: (8)    self, points: Point3D_Array | None = None, dim: int = 0, key: int = 0
1548: (4) ) -> np.ndarray | float:
1549: (8)    if points is None:
1550: (12)      points = self.get_points_defining_boundary()
1551: (8)    values = points[:, dim]
1552: (8)    if key < 0:
1553: (12)      return np.min(values)
1554: (8)    elif key == 0:
1555: (12)      return (np.min(values) + np.max(values)) / 2
1556: (8)    else:
1557: (12)      return np.max(values)
1558: (4) def get_critical_point(self, direction: Vector3D) -> Point3D:
1559: (8)    """Picture a box bounding the :class:`~.Mobject`. Such a box has
1560: (8)    9 'critical points': 4 corners, 4 edge center, the
1561: (8)    center. This returns one of them, along the given direction.
1562: (8)    """
1563: (12)    sample = Arc(start_angle=PI/7, angle = PI/5)
1564: (12)    max_y_1 = sample.get_top()[1]
1565: (12)    max_y_2 = sample.get_critical_point(UP)[1]
1566: (12)    max_y_3 = sample.get_extremum_along_dim(dim=1, key=1)
1567: (8)    """
1568: (8)    result = np.zeros(self.dim)
1569: (8)    all_points = self.get_points_defining_boundary()
1570: (8)    if len(all_points) == 0:
1571: (12)      return result
1572: (8)    for dim in range(self.dim):
1573: (12)      result[dim] = self.get_extremum_along_dim(
1574: (16)        all_points,
1575: (16)        dim=dim,
1576: (16)        key=direction[dim],
1577: (12)      )
1578: (8)    return result
1579: (4) def get_edge_center(self, direction: Vector3D) -> Point3D:

```

```

1580: (8)             """Get edge Point3Ds for certain direction."""
1581: (8)             return self.get_critical_point(direction)
1582: (4)             def get_corner(self, direction: Vector3D) -> Point3D:
1583: (8)                 """Get corner Point3Ds for certain direction."""
1584: (8)                 return self.get_critical_point(direction)
1585: (4)             def get_center(self) -> Point3D:
1586: (8)                 """Get center Point3Ds"""
1587: (8)                 return self.get_critical_point(np.zeros(self.dim))
1588: (4)             def get_center_of_mass(self) -> Point3D:
1589: (8)                 return np.apply_along_axis(np.mean, 0, self.get_all_points())
1590: (4)             def get_boundary_point(self, direction: Vector3D) -> Point3D:
1591: (8)                 all_points = self.get_points_defining_boundary()
1592: (8)                 index = np.argmax(np.dot(all_points, np.array(direction).T))
1593: (8)                 return all_points[index]
1594: (4)             def get_midpoint(self) -> Point3D:
1595: (8)                 """Get Point3Ds of the middle of the path that forms the
:class:`~.Mobject`."""

1596: (8)             Examples
1597: (8)             -----
1598: (8)             .. manim:: AngleMidPoint
1599: (12)             :save_last_frame:
1600: (12)             class AngleMidPoint(Scene):
1601: (16)                 def construct(self):
1602: (20)                     line1 = Line(ORIGIN, 2*RIGHT)
1603: (20)                     line2 = Line(ORIGIN,
2*RIGHT).rotate_about_origin(80*DEGREES)
1604: (20)                     a = Angle(line1, line2, radius=1.5, other_angle=False)
1605: (20)                     d = Dot(a.get_midpoint()).set_color(RED)
1606: (20)                     self.add(line1, line2, a, d)
1607: (20)                     self.wait()
1608: (8)             """
1609: (8)             return self.point_from_proportion(0.5)
1610: (4)             def get_top(self) -> Point3D:
1611: (8)                 """Get top Point3Ds of a box bounding the :class:`~.Mobject`"""
1612: (8)                 return self.get_edge_center(UP)
1613: (4)             def get_bottom(self) -> Point3D:
1614: (8)                 """Get bottom Point3Ds of a box bounding the :class:`~.Mobject`"""
1615: (8)                 return self.get_edge_center(DOWN)
1616: (4)             def get_right(self) -> Point3D:
1617: (8)                 """Get right Point3Ds of a box bounding the :class:`~.Mobject`"""
1618: (8)                 return self.get_edge_center(RIGHT)
1619: (4)             def get_left(self) -> Point3D:
1620: (8)                 """Get left Point3Ds of a box bounding the :class:`~.Mobject`"""
1621: (8)                 return self.get_edge_center(LEFT)
1622: (4)             def get_zenith(self) -> Point3D:
1623: (8)                 """Get zenith Point3Ds of a box bounding a 3D :class:`~.Mobject`"""
1624: (8)                 return self.get_edge_center(OUT)
1625: (4)             def get_nadir(self) -> Point3D:
1626: (8)                 """Get nadir (opposite the zenith) Point3Ds of a box bounding a 3D
:class:`~.Mobject`."""
1627: (8)                 return self.get_edge_center(IN)
1628: (4)             def length_over_dim(self, dim: int) -> float:
1629: (8)                 """Measure the length of an :class:`~.Mobject` in a certain
direction."""
1630: (8)                 return self.reduce_across_dimension(
1631: (12)                     max,
1632: (12)                     dim,
1633: (8)                 ) - self.reduce_across_dimension(min, dim)
1634: (4)             def get_coord(self, dim: int, direction: Vector3D = ORIGIN):
1635: (8)                 """Meant to generalize ``get_x``, ``get_y`` and ``get_z``"""
1636: (8)                 return self.get_extremum_along_dim(dim=dim, key=direction[dim])
1637: (4)             def get_x(self, direction: Vector3D = ORIGIN) -> ManimFloat:
1638: (8)                 """Returns x Point3D of the center of the :class:`~.Mobject` as
``float``"""
1639: (8)                 return self.get_coord(0, direction)
1640: (4)             def get_y(self, direction: Vector3D = ORIGIN) -> ManimFloat:
1641: (8)                 """Returns y Point3D of the center of the :class:`~.Mobject` as
``float``"""
1642: (8)                 return self.get_coord(1, direction)

```

```

1643: (4)             def get_z(self, direction: Vector3D = ORIGIN) -> ManimFloat:
1644: (8)                 """Returns z Point3D of the center of the :class:`~.Mobject` as
``float``"""
1645: (8)                     return self.get_coord(2, direction)
1646: (4)             def get_start(self) -> Point3D:
1647: (8)                 """Returns the point, where the stroke that surrounds the
:class:`~.Mobject` starts."""
1648: (8)                     self.throw_error_if_no_points()
1649: (8)                     return np.array(self.points[0])
1650: (4)             def get_end(self) -> Point3D:
1651: (8)                 """Returns the point, where the stroke that surrounds the
:class:`~.Mobject` ends."""
1652: (8)                     self.throw_error_if_no_points()
1653: (8)                     return np.array(self.points[-1])
1654: (4)             def get_start_and_end(self) -> tuple[Point3D, Point3D]:
1655: (8)                 """Returns starting and ending point of a stroke as a ``tuple``."""
1656: (8)                 return self.get_start(), self.get_end()
1657: (4)             def point_from_proportion(self, alpha: float) -> Point3D:
1658: (8)                 raise NotImplementedError("Please override in a child class.")
1659: (4)             def proportion_from_point(self, point: Point3D) -> float:
1660: (8)                 raise NotImplementedError("Please override in a child class.")
1661: (4)             def get_pieces(self, n_pieces: float) -> Group:
1662: (8)                 template = self.copy()
1663: (8)                 template.submobjects = []
1664: (8)                 alphas = np.linspace(0, 1, n_pieces + 1)
1665: (8)                 return Group(
1666: (12)                     *
1667: (16)                         template.copy().pointwise_become_partial(self, a1, a2)
1668: (16)                         for a1, a2 in zip(alphas[:-1], alphas[1:])
1669: (12)                 )
1670: (8)             )
1671: (4)             def get_z_index_reference_point(self) -> Point3D:
1672: (8)                 z_index_group = getattr(self, "z_index_group", self)
1673: (8)                 return z_index_group.get_center()
1674: (4)             def has_points(self) -> bool:
1675: (8)                 """Check if :class:`~.Mobject` contains points."""
1676: (8)                 return len(self.points) > 0
1677: (4)             def has_no_points(self) -> bool:
1678: (8)                 """Check if :class:`~.Mobject` *does not* contains points."""
1679: (8)                 return not self.has_points()
1680: (4)             def match_color(self, mobject: Mobject) -> Self:
1681: (8)                 """Match the color with the color of another :class:`~.Mobject`."""
1682: (8)                 return self.set_color(mobject.get_color())
1683: (4)             def match_dim_size(self, mobject: Mobject, dim: int, **kwargs) -> Self:
1684: (8)                 """Match the specified dimension with the dimension of another
:class:`~.Mobject`."""
1685: (8)                 return self.rescale_to_fit(mobject.length_over_dim(dim), dim,
**kwargs)
1686: (4)             def match_width(self, mobject: Mobject, **kwargs) -> Self:
1687: (8)                 """Match the width with the width of another :class:`~.Mobject`."""
1688: (8)                 return self.match_dim_size(mobject, 0, **kwargs)
1689: (4)             def match_height(self, mobject: Mobject, **kwargs) -> Self:
1690: (8)                 """Match the height with the height of another :class:`~.Mobject`."""
1691: (8)                 return self.match_dim_size(mobject, 1, **kwargs)
1692: (4)             def match_depth(self, mobject: Mobject, **kwargs) -> Self:
1693: (8)                 """Match the depth with the depth of another :class:`~.Mobject`."""
1694: (8)                 return self.match_dim_size(mobject, 2, **kwargs)
1695: (4)             def match_coord(
1696: (8)                 self, mobject: Mobject, dim: int, direction: Vector3D = ORIGIN
1697: (4)             ) -> Self:
1698: (8)                 """Match the Point3Ds with the Point3Ds of another
:class:`~.Mobject`."""
1699: (8)                 return self.set_coord(
1700: (12)                     mobject.get_coord(dim, direction),
1701: (12)                     dim=dim,
1702: (12)                     direction=direction,
1703: (8)                 )
1704: (4)             def match_x(self, mobject: Mobject, direction=ORIGIN) -> Self:
1705: (8)                 """Match x coord. to the x coord. of another :class:`~.Mobject`."""

```

```

1706: (8)             return self.match_coord(mobject, 0, direction)
1707: (4)             def match_y(self, mobject: Mobject, direction=ORIGIN) -> Self:
1708: (8)                 """Match y coord. to the x coord. of another :class:`~.Mobject`."""
1709: (8)                 return self.match_coord(mobject, 1, direction)
1710: (4)             def match_z(self, mobject: Mobject, direction=ORIGIN) -> Self:
1711: (8)                 """Match z coord. to the x coord. of another :class:`~.Mobject`."""
1712: (8)                 return self.match_coord(mobject, 2, direction)
1713: (4)             def align_to(
1714: (8)                 self,
1715: (8)                 mobject_or_point: Mobject | Point3D,
1716: (8)                 direction: Vector3D = ORIGIN,
1717: (4)             ) -> Self:
1718: (8)                 """Aligns mobject to another :class:`~.Mobject` in a certain
direction.
1719: (8)             Examples:
1720: (8)                 mob1.align_to(mob2, UP) moves mob1 vertically so that its
1721: (8)                 top edge lines up with mob2's top edge.
1722: (8)
1723: (8)                 """
1724: (12)                 if isinstance(mobject_or_point, Mobject):
1725: (8)                     point = mobject_or_point.get_critical_point(direction)
1726: (12)                 else:
1727: (8)                     point = mobject_or_point
1728: (12)                 for dim in range(self.dim):
1729: (16)                     if direction[dim] != 0:
1730: (8)                         self.set_coord(point[dim], dim, direction)
1731: (4)             return self
1732: (8)             def __getitem__(self, value):
1733: (8)                 self_list = self.split()
1734: (12)                 if isinstance(value, slice):
1735: (12)                     GroupClass = self.get_group_class()
1736: (8)                     return GroupClass(*self_list.__getitem__(value))
1737: (4)                 return self_list.__getitem__(value)
1738: (8)             def __iter__(self):
1739: (4)                 return iter(self.split())
1740: (8)             def __len__(self):
1741: (4)                 return len(self.split())
1742: (8)             def get_group_class(self) -> type[Group]:
1743: (4)                 return Group
1744: (4)             @staticmethod
1745: (8)             def get_mobject_type_class() -> type[Mobject]:
1746: (8)                 """Return the base class of this mobject type."""
1747: (8)                 return Mobject
1748: (8)             def split(self) -> list[Self]:
1749: (8)                 result = [self] if len(self.points) > 0 else []
1750: (4)                 return result + self.submobjects
1751: (8)             def get_family(self, recurse: bool = True) -> list[Self]:
1752: (8)                 sub_families = [x.get_family() for x in self.submobjects]
1753: (8)                 all_mobjects = [self] + list(it.chain(*sub_families))
1754: (4)                 return remove_list_redundancies(all_mobjects)
1755: (8)             def family_members_with_points(self) -> list[Self]:
1756: (4)                 return [m for m in self.get_family() if m.get_num_points() > 0]
1757: (8)             def arrange(
1758: (8)                 self,
1759: (8)                 direction: Vector3D = RIGHT,
1760: (8)                 buff: float = DEFAULT_MOBJECT_TO_MOBJECT_BUFFER,
1761: (8)                 center: bool = True,
1762: (4)                 **kwargs,
1763: (8)             ) -> Self:
1764: (8)                 """Sorts :class:`~.Mobject` next to each other on screen.
1765: (8)             Examples
1766: (8)                 -----
1767: (12)                 .. manim:: Example
1768: (12)                     :save_last_frame:
1769: (16)                     class Example(Scene):
1770: (20)                         def construct(self):
1771: (20)                             s1 = Square()
1772: (20)                             s2 = Square()
1773: (20)                             s3 = Square()
1774: (20)                             s4 = Square()

```

```

1774: (20)           x = VGroup(s1, s2, s3, s4).set_x(0).arrange(buff=1.0)
1775: (20)           self.add(x)
1776: (8)           """
1777: (8)           for m1, m2 in zip(self.submobjects, self.submobjects[1:]):
1778: (12)             m2.next_to(m1, direction, buff, **kwargs)
1779: (8)           if center:
1780: (12)             self.center()
1781: (8)           return self
1782: (4)           def arrange_in_grid(
1783: (8)             self,
1784: (8)             rows: int | None = None,
1785: (8)             cols: int | None = None,
1786: (8)             buff: float | tuple[float, float] = MED_SMALL_BUFF,
1787: (8)             cell_alignment: Vector3D = ORIGIN,
1788: (8)             row_alignments: str | None = None, # "ucd"
1789: (8)             col_alignments: str | None = None, # "lcr"
1790: (8)             row_heights: Iterable[float | None] | None = None,
1791: (8)             col_widths: Iterable[float | None] | None = None,
1792: (8)             flow_order: str = "rd",
1793: (8)             **kwargs,
1794: (4)         ) -> Self:
1795: (8)           """Arrange submobjects in a grid.
1796: (8)           Parameters
1797: (8)           -----
1798: (8)           rows
1799: (12)             The number of rows in the grid.
1800: (8)           cols
1801: (12)             The number of columns in the grid.
1802: (8)           buff
1803: (12)             The gap between grid cells. To specify a different buffer in the
horizontal and
1804: (12)             vertical directions, a tuple of two values can be given - ````(row,
col)````.
1805: (8)           cell_alignment
1806: (12)             The way each submobject is aligned in its grid cell.
1807: (8)           row_alignments
1808: (12)             The vertical alignment for each row (top to bottom). Accepts the
following characters: ``"u"`` - 
1809: (12)               up, ``"c"`` - center, ``"d"`` - down.
1810: (8)           col_alignments
1811: (12)             The horizontal alignment for each column (left to right). Accepts
the following characters ``"l"`` - left,
1812: (12)               ``"c"`` - center, ``"r"`` - right.
1813: (8)           row_heights
1814: (12)             Defines a list of heights for certain rows (top to bottom). If the
list contains
1815: (12)               ``None``, the corresponding row will fit its height automatically
based
1816: (12)               on the highest element in that row.
1817: (8)           col_widths
1818: (12)             Defines a list of widths for certain columns (left to right). If
the list contains ``None``, the
1819: (12)               corresponding column will fit its width automatically based on the
widest element in that column.
1820: (8)           flow_order
1821: (12)             The order in which submobjects fill the grid. Can be one of the
following values:
1822: (12)               "rd", "dr", "ld", "dl", "ru", "ur", "lu", "ul". ("rd" -> fill
rightwards then downwards)
1823: (8)           Returns
1824: (8)           -----
1825: (8)           :class:`Mobject`  
           ``self``
1826: (12)           Raises
1827: (8)           -----
1828: (8)           ValueError
1829: (8)             If ``rows`` and ``cols`` are too small to fit all submobjects.
1830: (12)           ValueError
1831: (8)             If :code:`cols`, :code:`col_alignments` and :code:`col_widths` or
1832: (12)

```

```

:code:`rows`,
1833: (12) :code:`row_alignments` and :code:`row_heights` have mismatching
sizes.
1834: (8) Notes
1835: (8)
1836: (8) -----
will be chosen big
1837: (8) If only one of ``cols`` and ``rows`` is set implicitly, the other one
to be about the same, enough to fit all submobjects. If neither is set, they will be chosen
1838: (8) tending towards ``cols`` > ``rows`` (simply because videos are wider
than they are high).
1839: (8) If both ``cell_alignment`` and ``row_alignments`` / ``col_alignments``
are defined, the latter has higher priority.
1840: (8) Examples
1841: (8)
1842: (8)
1843: (8)
1844: (12)
1845: (12)
1846: (16)
1847: (20)
1848: (20)
1849: (20)
1850: (8)
1851: (12)
1852: (12)
1853: (16)
1854: (20)
1855: (24)
0.5).add(Text(str(i+1)).scale(0.5))
1856: (24)
1857: (20)
1858: (20)
1859: (20)
1860: (24)
1861: (24)
1862: (24)
1863: (24)
1864: (24)
1865: (24)
1866: (20)
1867: (8)
1868: (8)
1869: (8)
1870: (8)
1871: (8)
1872: (12)
1873: (16)
1874: (12)
1875: (16)
1876: (12)
1877: (16)
1878: (8)
1879: (8)
1880: (8)
1881: (12)
1882: (8)
1883: (12)
1884: (8)
1885: (12)
1886: (8)
1887: (12)
submobjetc."
1888: (8)
1889: (12)
1890: (12)
1891: (8)
1892: (12)
1893: (8) :code:`row_alignments` and :code:`row_heights` have mismatching

```

Notes

-----

If only one of ``cols`` and ``rows`` is set implicitly, the other one enough to fit all submobjects. If neither is set, they will be chosen tending towards ``cols`` > ``rows`` (simply because videos are wider than they are high).

If both ``cell\_alignment`` and ``row\_alignments`` / ``col\_alignments`` are defined, the latter has higher priority.

Examples

-----

```

.. manim:: ExampleBoxes
    :save_last_frame:
    class ExampleBoxes(Scene):
        def construct(self):
            boxes=VGroup(*[Square() for s in range(0,6)])
            boxes.arrange_in_grid(rows=2, buff=0.1)
            self.add(boxes)

.. manim:: ArrangeInGrid
    :save_last_frame:
    class ArrangeInGrid(Scene):
        def construct(self):
            boxes = VGroup([
                Rectangle(WHITE, 0.5,
0.5).add(Text(str(i+1)).scale(0.5))
for i in range(24)
])
            self.add(boxes)
            boxes.arrange_in_grid(
                buff=(0.25,0.5),
                col_alignments="lccccr",
                row_alignments="uccd",
                col_widths=[1, *[None]*4, 1],
                row_heights=[1, None, None, 1],
                flow_order="dr"
            )
"""

from manim.mobject.geometry.line import Line
mobs = self.submobjects.copy()
start_pos = self.get_center()
def init_size(num, alignments, sizes):
    if num is not None:
        return num
    if alignments is not None:
        return len(alignments)
    if sizes is not None:
        return len(sizes)
cols = init_size(cols, col_alignments, col_widths)
rows = init_size(rows, row_alignments, row_heights)
if rows is None and cols is None:
    cols = math.ceil(math.sqrt(len(mobs)))
if rows is None:
    rows = math.ceil(len(mobs) / cols)
if cols is None:
    cols = math.ceil(len(mobs) / rows)
if rows * cols < len(mobs):
    raise ValueError("Too few rows and columns to fit all
submobjetc.")

if isinstance(buff, tuple):
    buff_x = buff[0]
    buff_y = buff[1]
else:
    buff_x = buff_y = buff
def init_alignments(alignments, num, mapping, name, dir):

```

```

1894: (12)             if alignments is None:
1895: (16)                 return [cell_alignment * dir] * num
1896: (12)             if len(alignment) != num:
1897: (16)                 raise ValueError(f"{name}_alignments has a mismatching size.")
1898: (12)             alignments = list(alignment)
1899: (12)             for i in range(num):
1900: (16)                 alignment[i] = mapping[alignments[i]]
1901: (12)             return alignment
1902: (8)             row_alignments = init_alignments(
1903: (12)                 row_alignments,
1904: (12)                 rows,
1905: (12)                 {"u": UP, "c": ORIGIN, "d": DOWN},
1906: (12)                 "row",
1907: (12)                 RIGHT,
1908: (8)
1909: (8)             col_alignments = init_alignments(
1910: (12)                 col_alignments,
1911: (12)                 cols,
1912: (12)                 {"l": LEFT, "c": ORIGIN, "r": RIGHT},
1913: (12)                 "col",
1914: (12)                 UP,
1915: (8)
1916: (8)             mapper = {
1917: (12)                 "dr": lambda r, c: (rows - r - 1) + c * rows,
1918: (12)                 "dl": lambda r, c: (rows - r - 1) + (cols - c - 1) * rows,
1919: (12)                 "ur": lambda r, c: r + c * rows,
1920: (12)                 "ul": lambda r, c: r + (cols - c - 1) * rows,
1921: (12)                 "rd": lambda r, c: (rows - r - 1) * cols + c,
1922: (12)                 "ld": lambda r, c: (rows - r - 1) * cols + (cols - c - 1),
1923: (12)                 "ru": lambda r, c: r * cols + c,
1924: (12)                 "lu": lambda r, c: r * cols + (cols - c - 1),
1925: (8)
1926: (8)             if flow_order not in mapper:
1927: (12)                 raise ValueError(
1928: (16)                     'flow_order must be one of the following values: "dr", "rd",
"ld", "dl", "ru", "ur", "lu", "ul".',
1929: (12)                     )
1930: (8)             flow_order = mapper[flow_order]
1931: (8)             def reverse(maybe_list):
1932: (12)                 if maybe_list is not None:
1933: (16)                     maybe_list = list(maybe_list)
1934: (16)                     maybe_list.reverse()
1935: (16)                     return maybe_list
1936: (8)             row_alignments = reverse(row_alignments)
1937: (8)             row_heights = reverse(row_heights)
1938: (8)             placeholder = Mobject()
1939: (8)             mobs.extend([placeholder] * (rows * cols - len(mobs)))
1940: (8)             grid = [[mobs[flow_order(r, c)] for c in range(cols)] for r in
range(rows)]
1941: (8)             measured_heights =
1942: (12)                 max(grid[r][c].height for c in range(cols)) for r in range(rows)
1943: (8)
1944: (8)
1945: (12)             measured_widths =
1946: (8)                 max(grid[r][c].width for r in range(rows)) for c in range(cols)
1947: (8)
1948: (12)             def init_sizes(sizes, num, measures, name):
1949: (16)                 if sizes is None:
1950: (12)                     sizes = [None] * num
1951: (16)                     if len(sizes) != num:
1952: (12)                         raise ValueError(f"{name} has a mismatching size.")
1953: (16)                     return [
1954: (12)                         sizes[i] if sizes[i] is not None else measures[i] for i in
range(num)
1955: (8)                     ]
1956: (8)             heights = init_sizes(row_heights, rows, measured_heights,
1957: (8)             widths = init_sizes(col_widths, cols, measured_widths, "col_widths")
1958: (8)             x, y = 0, 0
for r in range(rows):

```

```

1959: (12)           x = 0
1960: (12)           for c in range(cols):
1961: (16)             if grid[r][c] is not placeholder:
1962: (20)               alignment = row_alignments[r] + col_alignments[c]
1963: (20)               line = Line(
1964: (24)                 x * RIGHT + y * UP,
1965: (24)                 (x + widths[c]) * RIGHT + (y + heights[r]) * UP,
1966: (20)               )
1967: (20)               grid[r][c].move_to(line, alignment)
1968: (16)             x += widths[c] + buff_x
1969: (12)             y += heights[r] + buff_y
1970: (8)             self.move_to(start_pos)
1971: (8)             return self
1972: (4)         def sort(
1973: (8)           self,
1974: (8)             point_to_num_func: Callable[[Point3D], ManimInt] = lambda p: p[0],
1975: (8)             submob_func: Callable[[Mobject], ManimInt] | None = None,
1976: (4)         ) -> Self:
1977: (8)           """Sorts the list of :attr:`submobjects` by a function defined by
``submob_func``."""
1978: (8)           if submob_func is None:
1979: (12)             def submob_func(m: Mobject):
1980: (16)               return point_to_num_func(m.get_center())
1981: (8)             self.submobjects.sort(key=submob_func)
1982: (8)             return self
1983: (4)         def shuffle(self, recursive: bool = False) -> None:
1984: (8)           """Shuffles the list of :attr:`submobjects`."""
1985: (8)           if recursive:
1986: (12)             for submob in self.submobjects:
1987: (16)               submob.shuffle(recursive=True)
1988: (8)             random.shuffle(self.submobjects)
1989: (4)         def invert(self, recursive: bool = False) -> None:
1990: (8)           """Inverts the list of :attr:`submobjects`.
1991: (8)             Parameters
1992: (8)               -----
1993: (8)               recursive
1994: (12)                 If ``True``, all submobject lists of this mobject's family are
inverted.
1995: (8)
1996: (8)
1997: (8)
1998: (12)             .. manim:: InvertSumobjectsExample
1999: (16)               class InvertSumobjectsExample(Scene):
2000: (20)                 def construct(self):
2001: (20)                   s = VGroup(*[Dot().shift(i*0.1*RIGHT) for i in
2002: (20)                     range(-20,20)])
2003: (20)
2004: (20)
2005: (8)
2006: (8)
2007: (12)
2008: (16)
2009: (8)
2010: (4)         def arrange_submobjects(self, *args, **kwargs) -> Self:
2011: (8)           """Arrange the position of :attr:`submobjects` with a small buffer.
2012: (8)             Examples
2013: (8)
2014: (8)             .. manim:: ArrangeSumobjectsExample
2015: (12)               :save_last_frame:
2016: (12)               class ArrangeSumobjectsExample(Scene):
2017: (16)                 def construct(self):
2018: (20)                   s= VGroup(*
[Dot().shift(i*0.1*RIGHT*np.random.uniform(-1,1)+UP*np.random.uniform(-1,1)) for i in
range(0,15)])
2019: (20)                     s.shift(UP).set_color(BLUE)
2020: (20)                     s2= s.copy().set_color(RED)
2021: (20)                     s2.arrange_submobjects()
2022: (20)                     s2.shift(DOWN)

```

```

2023: (20)                         self.add(s,s2)
2024: (8)                          """
2025: (8)                          return self.arrange(*args, **kwargs)
2026: (4)  def sort_submobjects(self, *args, **kwargs) -> Self:
2027: (8)      """Sort the :attr:`submobjects`"""
2028: (8)      return self.sort(*args, **kwargs)
2029: (4)  def shuffle_submobjects(self, *args, **kwargs) -> None:
2030: (8)      """Shuffles the order of :attr:`submobjects`"""
2031: (8)  Examples
2032: (8)  -----
2033: (8)  .. manim:: ShuffleSubmobjectsExample
2034: (12)    class ShuffleSubmobjectsExample(Scene):
2035: (16)      def construct(self):
2036: (20)          s= VGroup(*[Dot().shift(i*0.1*RIGHT) for i in
range(-20,20)])
2037: (20)          s2= s.copy()
2038: (20)          s2.shuffle_submobjects()
2039: (20)          s2.shift(DOWN)
2040: (20)          self.play(Write(s), Write(s2))
2041: (8)      """
2042: (8)      return self.shuffle(*args, **kwargs)
2043: (4)  def align_data(self, mobject: Mobject, skip_point_alignment: bool = False)
-> None:
2044: (8)      """Aligns the data of this mobject with another mobject.
2045: (8)      Afterwards, the two mobjects will have the same number of submobjects
2046: (8)      (see :meth:`.align_submobjects`), the same parent structure (see
2047: (8)      :meth:`.null_point_align`). If ``skip_point_alignment`` is false,
2048: (8)      they will also have the same number of points (see
:meth:`.align_points`).
2049: (8)  Parameters
2050: (8)  -----
2051: (8)  mobject
2052: (12)    The other mobject this mobject should be aligned to.
2053: (8)  skip_point_alignment
2054: (12)    Controls whether or not the computationally expensive
2055: (12)    point alignment is skipped (default: False).
2056: (8)  """
2057: (8)  self.null_point_align(mobject)
2058: (8)  self.align_submobjects(mobject)
2059: (8)  if not skip_point_alignment:
2060: (12)    self.align_points(mobject)
2061: (8)  for m1, m2 in zip(self.submobjects, mobject.submobjects):
2062: (12)    m1.align_data(m2)
2063: (4)  def get_point_mobject(self, center=None):
2064: (8)      """The simplest :class:`~.Mobject` to be transformed to or from self.
2065: (8)      Should be a point of the appropriate type
2066: (8)      """
2067: (8)      msg = f"get_point_mobject not implemented for
{self.__class__.__name__}"
2068: (8)      raise NotImplementedError(msg)
2069: (4)  def align_points(self, mobject: Mobject) -> Self:
2070: (8)      count1 = self.get_num_points()
2071: (8)      count2 = mobject.get_num_points()
2072: (8)      if count1 < count2:
2073: (12)        self.align_points_with_larger(mobject)
2074: (8)      elif count2 < count1:
2075: (12)        mobject.align_points_with_larger(self)
2076: (8)      return self
2077: (4)  def align_points_with_larger(self, larger_mobject: Mobject):
2078: (8)      raise NotImplementedError("Please override in a child class.")
2079: (4)  def align_submobjects(self, mobject: Mobject) -> Self:
2080: (8)      mob1 = self
2081: (8)      mob2 = mobject
2082: (8)      n1 = len(mob1.submobjects)
2083: (8)      n2 = len(mob2.submobjects)
2084: (8)      mob1.add_n_more_submobjects(max(0, n2 - n1))
2085: (8)      mob2.add_n_more_submobjects(max(0, n1 - n2))
2086: (8)      return self
2087: (4)  def null_point_align(self, mobject: Mobject):

```

```

2088: (8)             """If a :class:`~.Mobject` with points is being aligned to
2089: (8)             one without, treat both as groups, and push
2090: (8)             the one with points into its own submobjects
2091: (8)             list.
2092: (8)             Returns
2093: (8)             -----
2094: (8)             :class:`Mobject`
2095: (12)             ``self``
2096: (8)
2097: (8)             """
2098: (12)             for m1, m2 in (self, mobobject), (mobobject, self):
2099: (16)                 if m1.has_no_points() and m2.has_points():
2100: (8)                     m2.push_self_into_submobjects()
2101: (8)             return self
2102: (4)             def push_self_into_submobjects(self) -> Self:
2103: (8)                 copy = self.copy()
2104: (8)                 copy.submobjects = []
2105: (8)                 self.reset_points()
2106: (8)                 self.add(copy)
2107: (8)                 return self
2108: (4)             def add_n_more_submobjects(self, n: int) -> Self | None:
2109: (8)                 if n == 0:
2110: (8)                     return None
2111: (8)                 curr = len(self.submobjects)
2112: (12)                 if curr == 0:
2113: (12)                     self.submobjects = [self.get_point_mobject() for k in range(n)]
2114: (8)                     return None
2115: (8)                 target = curr + n
2116: (8)                 repeat_indices = (np.arange(target) * curr) // target
2117: (8)                 split_factors = [sum(repeat_indices == i) for i in range(curr)]
2118: (8)                 new_submobs = []
2119: (12)                 for submob, sf in zip(self.submobjects, split_factors):
2120: (12)                     new_submobs.append(submob)
2121: (16)                     for _ in range(1, sf):
2122: (8)                         new_submobs.append(submob.copy().fade(1))
2123: (8)                 self.submobjects = new_submobs
2124: (8)             return self
2125: (4)             def repeat_submobject(self, submob: Mobject) -> Self:
2126: (8)                 return submob.copy()
2127: (4)             def interpolate(
2128: (8)                 self,
2129: (8)                 mobobject1: Mobject,
2130: (8)                 mobobject2: Mobject,
2131: (8)                 alpha: float,
2132: (4)                 path_func: PathFuncType = straight_path(),
2133: (8)             ) -> Self:
2134: (8)                 """Turns this :class:`~.Mobject` into an interpolation between
``mobobject1``
2135: (8)                 and ``mobobject2``.
2136: (8)                 Examples
2137: (8)                 .. manim:: DotInterpolation
2138: (12)                     :save_last_frame:
2139: (12)                     class DotInterpolation(Scene):
2140: (16)                         def construct(self):
2141: (20)                             dotR = Dot(color=DARK_GREY)
2142: (20)                             dotR.shift(2 * RIGHT)
2143: (20)                             dotL = Dot(color=WHITE)
2144: (20)                             dotL.shift(2 * LEFT)
2145: (20)                             dotMiddle = VMobject().interpolate(dotL, dotR, alpha=0.3)
2146: (20)                             self.add(dotL, dotR, dotMiddle)
2147: (8)
2148: (8)                         self.points = path_func(mobobject1.points, mobobject2.points, alpha)
2149: (8)                         self.interpolate_color(mobobject1, mobobject2, alpha)
2150: (8)                         return self
2151: (4)             def interpolate_color(self, mobobject1: Mobject, mobobject2: Mobject, alpha: float):
2152: (8)                 raise NotImplementedError("Please override in a child class.")
2153: (4)             def become(
2154: (8)                 self,

```

```

2155: (8)             mobject: Mobject,
2156: (8)             match_height: bool = False,
2157: (8)             match_width: bool = False,
2158: (8)             match_depth: bool = False,
2159: (8)             match_center: bool = False,
2160: (8)             stretch: bool = False,
2161: (4)         ) -> Self:
2162: (8)             """Edit points, colors and submobjects to be identical
2163: (8)             to another :class:`~.Mobject`"""
2164: (8)             .. note::
2165: (12)                 If both match_height and match_width are ``True`` then the
transformed :class:`~.Mobject`  

2166: (12)                     will match the height first and then the width.
2167: (8)             Parameters
2168: (8)             -----
2169: (8)             match_height
2170: (12)                 Whether or not to preserve the height of the original
2171: (12)                 :class:`~.Mobject`.
2172: (8)             match_width
2173: (12)                 Whether or not to preserve the width of the original
2174: (12)                 :class:`~.Mobject`.
2175: (8)             match_depth
2176: (12)                 Whether or not to preserve the depth of the original
2177: (12)                 :class:`~.Mobject`.
2178: (8)             match_center
2179: (12)                 Whether or not to preserve the center of the original
2180: (12)                 :class:`~.Mobject`.
2181: (8)             stretch
2182: (12)                 Whether or not to stretch the target mobject to match the
2183: (12)                 the proportions of the original :class:`~.Mobject`.
2184: (8)             Examples
2185: (8)             -----
2186: (8)             .. manim:: BecomeScene
2187: (12)                 class BecomeScene(Scene):
2188: (16)                     def construct(self):
2189: (20)                         circ = Circle(fill_color=RED, fill_opacity=0.8)
2190: (20)                         square = Square(fill_color=BLUE, fill_opacity=0.2)
2191: (20)                         self.add(circ)
2192: (20)                         self.wait(0.5)
2193: (20)                         circ.become(square)
2194: (20)                         self.wait(0.5)
2195: (8)             The following examples illustrate how mobject measurements
2196: (8)             change when using the ``match_...`` and ``stretch`` arguments.
2197: (8)             We start with a rectangle that is 2 units high and 4 units wide,
2198: (8)             which we want to turn into a circle of radius 3:::
2199: (12)                 >>> from manim import Rectangle, Circle
2200: (12)                 >>> import numpy as np
2201: (12)                 >>> rect = Rectangle(height=2, width=4)
2202: (12)                 >>> circ = Circle(radius=3)
2203: (8)             With ``stretch=True``, the target circle is deformed to match
2204: (8)             the proportions of the rectangle, which results in the target
2205: (8)             mobject being an ellipse with height 2 and width 4. We can
2206: (8)             check that the resulting points satisfy the ellipse equation
2207: (8)             :math:`x^2/a^2 + y^2/b^2 = 1` with :math:`a = 4/2` and :math:`b = 2/2`
2208: (8)             being the semi-axes:::
2209: (12)                 >>> result = rect.copy().become(circ, stretch=True)
2210: (12)                 >>> result.height, result.width
2211: (12)                 (2.0, 4.0)
2212: (12)                 >>> ellipse_points = np.array(result.get_anchors())
2213: (12)                 >>> ellipse_eq = np.sum(ellipse_points**2 * [1/4, 1, 0], axis=1)
2214: (12)                 >>> np.allclose(ellipse_eq, 1)
2215: (12)                 True
2216: (8)             With ``match_height=True`` and ``match_width=True`` the circle is
2217: (8)             scaled such that the height or the width of the rectangle will
2218: (8)             be preserved, respectively.
2219: (8)             The points of the resulting mobject satisfy the circle equation
2220: (8)             :math:`x^2 + y^2 = r^2` for the corresponding radius :math:`r`:::
2221: (12)                 >>> result = rect.copy().become(circ, match_height=True)
2222: (12)                 >>> result.height, result.width

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2223: (12)                                (2.0, 2.0)
2224: (12)                                >>> circle_points = np.array(result.get_anchors())
2225: (12)                                >>> circle_eq = np.sum(circle_points**2, axis=1)
2226: (12)                                >>> np.allclose(circle_eq, 1)
2227: (12)                                True
2228: (12)                                >>> result = rect.copy().become(circ, match_width=True)
2229: (12)                                >>> result.height, result.width
2230: (12)                                (4.0, 4.0)
2231: (12)                                >>> circle_points = np.array(result.get_anchors())
2232: (12)                                >>> circle_eq = np.sum(circle_points**2, axis=1)
2233: (12)                                >>> np.allclose(circle_eq, 2**2)
2234: (12)                                True
2235: (8)                                 With ``match_center=True``, the resulting mobject is moved such that
2236: (8)                                 its center is the same as the center of the original mobject::
2237: (12)                                >>> rect = rect.shift(np.array([0, 1, 0]))
2238: (12)                                >>> np.allclose(rect.get_center(), circ.get_center())
2239: (12)                                False
2240: (12)                                >>> result = rect.copy().become(circ, match_center=True)
2241: (12)                                >>> np.allclose(rect.get_center(), result.get_center())
2242: (12)                                True
2243: (8)                                 """
2244: (8)                                 mobject = mobject.copy()
2245: (8)                                 if stretch:
2246: (12)                                   mobject.stretch_to_fit_height(self.height)
2247: (12)                                   mobject.stretch_to_fit_width(self.width)
2248: (12)                                   mobject.stretch_to_fit_depth(self.depth)
2249: (8)                                 else:
2250: (12)                                   if match_height:
2251: (16)                                     mobject.match_height(self)
2252: (12)                                   if match_width:
2253: (16)                                     mobject.match_width(self)
2254: (12)                                   if match_depth:
2255: (16)                                     mobject.match_depth(self)
2256: (8)                                 if match_center:
2257: (12)                                   mobject.move_to(self.get_center())
2258: (8)                                 self.align_data(mobject, skip_point_alignment=True)
2259: (8)                                 for sm1, sm2 in zip(self.get_family(), mobject.get_family()):
2260: (12)                                   sm1.points = np.array(sm2.points)
2261: (12)                                   sm1.interpolate_color(sm1, sm2, 1)
2262: (8)                                 return self
2263: (4)                                 def match_points(self, mobject: Mobject, copy_submobjects: bool = True) ->
Self:
2264: (8)                                 """Edit points, positions, and submobjects to be identical
2265: (8)                                 to another :class:`~.Mobject`, while keeping the style unchanged.
2266: (8)                                 Examples
2267: (8)                                 -----
2268: (8)                                 .. manim:: MatchPointsScene
2269: (12)                                 class MatchPointsScene(Scene):
2270: (16)                                   def construct(self):
2271: (20)                                     circ = Circle(fill_color=RED, fill_opacity=0.8)
2272: (20)                                     square = Square(fill_color=BLUE, fill_opacity=0.2)
2273: (20)                                     self.add(circ)
2274: (20)                                     self.wait(0.5)
2275: (20)                                     self.play(circ.animate.match_points(square))
2276: (20)                                     self.wait(0.5)
2277: (8)                                 """
2278: (8)                                 for sm1, sm2 in zip(self.get_family(), mobject.get_family()):
2279: (12)                                   sm1.points = np.array(sm2.points)
2280: (8)                                 return self
2281: (4)                                 def throw_error_if_no_points(self) -> None:
2282: (8)                                 if self.has_no_points():
2283: (12)                                   caller_name = sys._getframe(1).f_code.co_name
2284: (12)                                   raise Exception(
2285: (16)                                     f"Cannot call Mobject.{caller_name} for a Mobject with no
points",
2286: (12)                                     )
2287: (4)                                 def set_z_index(
2288: (8)                                   self,
2289: (8)                                   z_index_value: float,

```

```

2290: (8)             family: bool = True,
2291: (4)         ) -> Self:
2292: (8)             """Sets the :class:`~.Mobject`'s :attr:`z_index` to the value
specified in `z_index_value`.
2293: (8)             Parameters
2294: (8)             -----
2295: (8)             z_index_value
2296: (12)             The new value of :attr:`z_index` set.
2297: (8)
2298: (12)             If ``True``, the :attr:`z_index` value of all submobjects is also
set.
2299: (8)             Returns
2300: (8)             -----
2301: (8)             :class:`Mobject`
2302: (12)             The Mobject itself, after :attr:`z_index` is set. For chaining
purposes. (Returns `self`.)
2303: (8)             Examples
2304: (8)             -----
2305: (8)             .. manim:: SetZIndex
2306: (12)             :save_last_frame:
2307: (12)             class SetZIndex(Scene):
2308: (16)             def construct(self):
2309: (20)                 text = Text('z_index = 3', color =
PURE_RED).shift(UP).set_z_index(3)
2310: (20)                 square = Square(2, fill_opacity=1).set_z_index(2)
2311: (20)                 tex = Tex(r'zIndex = 1', color =
PURE_BLUE).shift(DOWN).set_z_index(1)
2312: (20)                 circle = Circle(radius = 1.7, color = GREEN, fill_opacity
= 1) # z_index = 0
2313: (20)                 self.add(text)
2314: (20)                 self.add(square)
2315: (20)                 self.add(tex)
2316: (20)                 self.add(circle)
2317: (8)             """
2318: (8)             if family:
2319: (12)                 for submob in self.submobjects:
2320: (16)                     submob.set_z_index(z_index_value, family=family)
2321: (8)                 self.z_index = z_index_value
2322: (8)             return self
2323: (4)             def set_z_index_by_z_Point3D(self) -> Self:
2324: (8)                 """Sets the :class:`~.Mobject`'s z Point3D to the value of
:attr:`z_index`.
2325: (8)             Returns
2326: (8)             -----
2327: (8)             :class:`Mobject`
2328: (12)             The Mobject itself, after :attr:`z_index` is set. (Returns
`self`.)
2329: (8)             """
2330: (8)             z_coord = self.get_center()[-1]
2331: (8)             self.set_z_index(z_coord)
2332: (8)             return self
2333: (0)             class Group(Mobject, metaclass=ConvertToOpenGL):
2334: (4)                 """Groups together multiple :class:`Mobjects <.Mobject>`.
2335: (4)             Notes
2336: (4)             -----
2337: (4)                 When adding the same mobject more than once, repetitions are ignored.
2338: (4)                 Use :meth:`.Mobject.copy` to create a separate copy which can then
be added to the group.
2339: (4)             """
2340: (4)             def __init__(self, *mobjects, **kwargs) -> None:
2341: (4)                 super().__init__(**kwargs)
2342: (8)                 self.add(*mobjects)
2343: (8)
2344: (0)             class _AnimationBuilder:
2345: (4)                 def __init__(self, mobject) -> None:
2346: (8)                     self.mobject = mobject
2347: (8)                     self.mobject.generate_target()
2348: (8)                     self.overridden_animation = None
2349: (8)                     self.is_chaining = False
2350: (8)                     self.methods = []

```

```

2351: (8)             self.cannot_pass_args = False
2352: (8)             self.anim_args = {}
2353: (4)         def __call__(self, **kwargs) -> Self:
2354: (8)             if self.cannot_pass_args:
2355: (12)                 raise ValueError(
2356: (16)                     "Animation arguments must be passed before accessing methods
and can only be passed once",
2357: (12)                         )
2358: (8)             self.anim_args = kwargs
2359: (8)             self.cannot_pass_args = True
2360: (8)             return self
2361: (4)         def __getattr__(self, method_name) -> types.MethodType:
2362: (8)             method = getattr(self.mobject.target, method_name)
2363: (8)             has_overridden_animation = hasattr(method, "_override_animate")
2364: (8)             if (self.is_chaining and has_overridden_animation) or
2365: (12)                 self.overridden_animation:
2366: (16)                     raise NotImplementedError(
2367: (16)                         "Method chaining is currently not supported for "
2368: (12)                             "overridden animations",
2369: (8)             )
2370: (12)             def update_target(*method_args, **method_kwargs):
2371: (16)                 if has_overridden_animation:
2372: (20)                     self.overridden_animation = method._override_animate(
2373: (20)                         self.mobject,
2374: (20)                         *method_args,
2375: (20)                         anim_args=self.anim_args,
2376: (16)                         **method_kwargs,
2377: (12)                     )
2378: (16)                 else:
2379: (16)                     self.methods.append([method, method_args, method_kwargs])
2380: (12)                     method(*method_args, **method_kwargs)
2381: (8)                     return self
2382: (8)                     self.is_chaining = True
2383: (8)                     self.cannot_pass_args = True
2384: (8)                     return update_target
2385: (4)         def build(self) -> Animation:
2386: (8)             from ..animation.transform import ( # is this to prevent circular
import?
2387: (12)                 _MethodAnimation,
2388: (8)             )
2389: (12)             if self.overridden_animation:
2390: (8)                 anim = self.overridden_animation
2391: (12)             else:
2392: (8)                 anim = _MethodAnimation(self.mobject, self.methods)
2393: (12)             for attr, value in self.anim_args.items():
2394: (8)                 setattr(anim, attr, value)
2395: (0)             return anim
2396: (4)         def override_animate(method) -> types.FunctionType:
2397: (4)             """Decorator for overriding method animations.
2398: (4)             This allows to specify a method (returning an :class:`~.Animation`)
which is called when the decorated method is used with the ``.animate``

syntax
2399: (4)             for animating the application of a method.
2400: (4)             .. seealso::
2401: (8)                 :attr:`Mobject.animate`
2402: (4)             .. note::
2403: (8)                 Overridden methods cannot be combined with normal or other overridden
methods using method chaining with the ``.animate`` syntax.

Examples
-----
2407: (4)             .. manim:: AnimationOverrideExample
2408: (8)                 class CircleWithContent(VGroup):
2409: (12)                     def __init__(self, content):
2410: (16)                         super().__init__()
2411: (16)                         self.circle = Circle()
2412: (16)                         self.content = content
2413: (16)                         self.add(self.circle, content)
2414: (16)                         content.move_to(self.circle.get_center())
2415: (12)                     def clear_content(self):

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2416: (16)                     self.remove(self.content)
2417: (16)                     self.content = None
2418: (12) @override_animate(clear_content)
2419: (12)     def _clear_content_animation(self, anim_args=None):
2420: (16)         if anim_args is None:
2421: (20)             anim_args = {}
2422: (16)         anim = Uncreate(self.content, **anim_args)
2423: (16)         self.clear_content()
2424: (16)         return anim
2425: (8)     class AnimationOverrideExample(Scene):
2426: (12)         def construct(self):
2427: (16)             t = Text("hello!")
2428: (16)             my_mobject = CircleWithContent(t)
2429: (16)             self.play(Create(my_mobject))
2430: (16)             self.play(my_mobject.animate.clear_content())
2431: (16)             self.wait()
2432: (4) """
2433: (4)     def decorator(animation_method):
2434: (8)         method._override_animate = animation_method
2435: (8)         return animation_method
2436: (4)     return decorator

```

---

## File 57 - labeled.py:

```

1: (0)         r"""Mobjects that inherit from lines and contain a label along the length."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = ["LabeledLine", "LabeledArrow"]
4: (0)         from manim.constants import *
5: (0)         from manim.mobject.geometry.line import Arrow, Line
6: (0)         from manim.mobject.geometry.shape_matchers import (
7: (4)             BackgroundRectangle,
8: (4)             SurroundingRectangle,
9: (0)         )
10: (0)        from manim.mobject.text.tex_mobject import MathTex, Tex
11: (0)        from manim.mobject.text.text_mobject import Text
12: (0)        from manim.utils.color import WHITE, ManimColor, ParsableManimColor
13: (0)        class LabeledLine(Line):
14: (4)            """Constructs a line containing a label box somewhere along its length.
15: (4)            Parameters
16: (4)            -----
17: (4)            label : str | Tex | MathTex | Text
18: (8)                Label that will be displayed on the line.
19: (4)            label_position : float | optional
20: (8)                A ratio in the range [0-1] to indicate the position of the label with
respect to the length of the line. Default value is 0.5.
21: (4)            font_size : float | optional
22: (8)                Control font size for the label. This parameter is only used when
`label` is of type `str`.
23: (4)            label_color: ParsableManimColor | optional
24: (8)                The color of the label's text. This parameter is only used when
`label` is of type `str`.
25: (4)            label_frame : Bool | optional
26: (8)                Add a `SurroundingRectangle` frame to the label box.
27: (4)            frame_fill_color : ParsableManimColor | optional
28: (8)                Background color to fill the label box. If no value is provided, the
background color of the canvas will be used.
29: (4)            frame_fill_opacity : float | optional
30: (8)                Determine the opacity of the label box by passing a value in the range
[0-1], where 0 indicates complete transparency and 1 means full opacity.
31: (4)            .. seealso::
32: (8)                :class:`LabeledArrow`
33: (4)            Examples
34: (4)            -----
35: (4)            .. manim:: LabeledLineExample
36: (8)                :save_last_frame:
37: (8)                class LabeledLineExample(Scene):
38: (12)                    def construct(self):

```

```

39: (16)                     line = LabeledLine(
40: (20)                         label            = '0.5',
41: (20)                         label_position = 0.8,
42: (20)                         font_size       = 20,
43: (20)                         label_color     = WHITE,
44: (20)                         label_frame    = True,
45: (20)                         start=LEFT+DOWN,
46: (20)                         end=RIGHT+UP)
47: (16)                     line.set_length(line.get_length() * 2)
48: (16)                     self.add(line)
49: (4) """
50: (4)     def __init__(
51: (8)         self,
52: (8)         label: str | Tex | MathTex | Text,
53: (8)         label_position: float = 0.5,
54: (8)         font_size: float = DEFAULT_FONT_SIZE,
55: (8)         label_color: ParsableManimColor = WHITE,
56: (8)         label_frame: bool = True,
57: (8)         frame_fill_color: ParsableManimColor = None,
58: (8)         frame_fill_opacity: float = 1,
59: (8)         *args,
60: (8)         **kwargs,
61: (4)     ) -> None:
62: (8)         label_color = ManimColor(label_color)
63: (8)         frame_fill_color = ManimColor(frame_fill_color)
64: (8)         if isinstance(label, str):
65: (12)             from manim import MathTex
66: (12)             rendered_label = MathTex(label, color=label_color,
font_size=font_size)
67: (8)
68: (12)         else:
69: (8)             rendered_label = label
70: (8)         super().__init__(*args, **kwargs)
71: (8)         line_start, line_end = self.get_start_and_end()
72: (8)         new_vec = (line_end - line_start) * label_position
73: (8)         label_coords = line_start + new_vec
74: (8)         rendered_label.move_to(label_coords)
75: (12)         box = BackgroundRectangle(
76: (12)             rendered_label,
77: (12)             buff=0.05,
78: (12)             color=frame_fill_color,
79: (12)             fill_opacity=frame_fill_opacity,
stroke_width=0.5,
80: (8)
81: (8)         self.add(box)
82: (8)         if label_frame:
83: (12)             box_frame = SurroundingRectangle(
84: (16)                 rendered_label, buff=0.05, color=label_color, stroke_width=0.5
85: (12)
86: (12)             self.add(box_frame)
87: (8)             self.add(rendered_label)
88: (0)     class LabeledArrow(LabeledLine, Arrow):
89: (4)         """Constructs an arrow containing a label box somewhere along its length.
90: (4)         This class inherits its label properties from `LabeledLine`, so the main
parameters controlling it are the same.
91: (4)             Parameters
92: (4)             -----
93: (4)             label : str | Tex | MathTex | Text
94: (8)                 Label that will be displayed on the line.
95: (4)             label_position : float | optional
96: (8)                 A ratio in the range [0-1] to indicate the position of the label with
respect to the length of the line. Default value is 0.5.
97: (4)             font_size : float | optional
98: (8)                 Control font size for the label. This parameter is only used when
`label` is of type `str`.
99: (4)             label_color: ParsableManimColor | optional
100: (8)                The color of the label's text. This parameter is only used when
`label` is of type `str`.
101: (4)             label_frame : Bool | optional
102: (8)                Add a `SurroundingRectangle` frame to the label box.

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```
103: (4)             frame_fill_color : ParsableManimColor | optional
104: (8)                 Background color to fill the label box. If no value is provided, the
background color of the canvas will be used.
105: (4)             frame_fill_opacity : float | optional
106: (8)                 Determine the opacity of the label box by passing a value in the range
[0-1], where 0 indicates complete transparency and 1 means full opacity.
107: (4)             .. seealso::
108: (8)                 :class:`~LabeledLine`_
109: (4)             Examples
110: (4)             -----
111: (4)             .. manim:: LabeledArrowExample
112: (8)                 :save_last_frame:
113: (8)                 class LabeledArrowExample(Scene):
114: (12)                     def construct(self):
115: (16)                         l_arrow = LabeledArrow("0.5", start=LEFT*3, end=RIGHT*3 +
UP*2, label_position=0.5)
116: (16)                         self.add(l_arrow)
117: (4)                     """
118: (4)             def __init__(
119: (8)                 self,
120: (8)                 *args,
121: (8)                 **kwargs,
122: (4)             ) -> None:
123: (8)                 super().__init__(*args, **kwargs)
```

-----  
File 58 - \_\_init\_\_.py:

```
1: (0)
```

-----  
File 59 - \_\_init\_\_.py:

```
1: (0)
```

-----  
File 60 - polygram.py:

```
1: (0)             r"""Mobjects that are simple geometric shapes."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "Polygram",
5: (4)                 "Polygon",
6: (4)                 "RegularPolygram",
7: (4)                 "RegularPolygon",
8: (4)                 "Star",
9: (4)                 "Triangle",
10: (4)                "Rectangle",
11: (4)                "Square",
12: (4)                "RoundedRectangle",
13: (4)                "Cutout",
14: (0)            ]
15: (0)            from math import ceil
16: (0)            from typing import TYPE_CHECKING
17: (0)            import numpy as np
18: (0)            from manim.constants import *
19: (0)            from manim.mobject.geometry.arc import ArcBetweenPoints
20: (0)            from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
21: (0)            from manim.mobject.types.vectorized_mobject import VGroup, VMobject
22: (0)            from manim.utils.color import BLUE, WHITE, ParsableManimColor
23: (0)            from manim.utils.iterables import adjacent_n_tuples, adjacent_pairs
24: (0)            from manim.utils.space_ops import angle_between_vectors, normalize,
regular_vertices
25: (0)            if TYPE_CHECKING:
26: (4)                from typing_extensions import Self
27: (4)                from manim.typing import Point3D, Point3D_Array
```

```

28: (4)             from manim.utils.color import ParsableManimColor
29: (0)             class Polygram(VMObject, metaclass=ConvertToOpenGL):
30: (4)                 """A generalized :class:`Polygon`, allowing for disconnected sets of
edges.
31: (4)             Parameters
32: (4)                 -----
33: (4)             vertex_groups
34: (8)                 The groups of vertices making up the :class:`Polygram`.
35: (8)                 The first vertex in each group is repeated to close the shape.
36: (8)                 Each point must be 3-dimensional: ``[x,y,z]``
37: (4)             color
38: (8)                 The color of the :class:`Polygram`.
39: (4)             kwargs
40: (8)                 Forwarded to the parent constructor.
41: (4)             Examples
42: (4)                 -----
43: (4)             .. manim:: PolygramExample
44: (8)                 import numpy as np
45: (8)                 class PolygramExample(Scene):
46: (12)                     def construct(self):
47: (16)                         hexagram = Polygram(
48: (20)                             [[0, 2, 0], [-np.sqrt(3), -1, 0], [np.sqrt(3), -1, 0]],
49: (20)                             [[-np.sqrt(3), 1, 0], [0, -2, 0], [np.sqrt(3), 1, 0]],
50: (16)
51: (16)
52: (16)
53: (16)                         self.add(hexagram)
rate_func=linear)
54: (16)                         dot = Dot()
55: (16)                         self.play(MoveAlongPath(dot, hexagram), run_time=5,
56: (4)                                 self.remove(dot)
57: (4)                                 self.wait())
58: (8)             """
59: (4)             def __init__(
60: (8)                 self, *vertex_groups: Point3D, color: ParsableManimColor = BLUE,
**kwargs
61: (4)             ):
62: (8)                 super().__init__(color=color, **kwargs)
63: (4)                 for vertices in vertex_groups:
64: (12)                     first_vertex, *vertices = vertices
65: (12)                     first_vertex = np.array(first_vertex)
66: (12)                     self.start_new_path(first_vertex)
67: (12)                     self.add_points_as_corners(
68: (16)                         [*np.array(vertex) for vertex in vertices], first_vertex],
69: (12)
70: (4)             def get_vertices(self) -> Point3D_Array:
71: (8)                 """Gets the vertices of the :class:`Polygram`.
72: (8)                 Returns
73: (4)                     -----
74: (8)                     :class:`numpy.ndarray`
75: (8)                     The vertices of the :class:`Polygram`.
76: (8)             Examples
77: (4)                 -----
78: (8)                 :::
79: (12)                     >>> sq = Square()
80: (12)                     >>> sq.get_vertices()
81: (12)                     array([[ 1.,  1.,  0.],
82: (19)                         [-1.,  1.,  0.],
83: (19)                         [-1., -1.,  0.],
84: (19)                         [ 1., -1.,  0.]])
85: (8)
86: (4)             return self.get_start_anchors()
87: (8)             def get_vertex_groups(self) -> np.ndarray[Point3D_Array]:
88: (8)                 """Gets the vertex groups of the :class:`Polygram`.
89: (8)                 Returns
90: (4)                     -----
91: (8)                     :class:`numpy.ndarray`
92: (8)                     The vertex groups of the :class:`Polygram`.
93: (8)             Examples
94: (4)                 -----
95: (8)                 :::

```

```

94: (12)                                     >>> poly = Polygram([ORIGIN, RIGHT, UP], [LEFT, LEFT + UP, 2 *
LEFT])
95: (12)                                     >>> poly.get_vertex_groups()
96: (12)                                     array([[ 0.,  0.,  0.],
97: (20)                                         [ 1.,  0.,  0.],
98: (20)                                         [ 0.,  1.,  0.]],)
99: (12)                                     <BLANKLINE>
100: (19)                                     [[[-1.,  0.,  0.],
101: (20)                                         [-1.,  1.,  0.],
102: (20)                                         [-2.,  0.,  0.]])])
103: (8)                                     """
104: (8)                                     vertex_groups = []
105: (8)                                     group = []
106: (8)                                     for start, end in zip(self.get_start_anchors(),
self.get_end_anchors()):
107: (12)                                     group.append(start)
108: (12)                                     if self.consider_points_equals(end, group[0]):
109: (16)                                         vertex_groups.append(group)
110: (16)                                         group = []
111: (8)                                     return np.array(vertex_groups)
112: (4)                                     def round_corners(
113: (8)                                         self,
114: (8)                                         radius: float | list[float] = 0.5,
115: (8)                                         evenly_distribute_anchors: bool = False,
116: (8)                                         components_per_rounded_corner: int = 2,
117: (4)                                     ) -> Self:
118: (8)                                         """Rounds off the corners of the :class:`Polygram`.
119: (8)                                         Parameters
120: (8)                                         -----
121: (8)                                         radius
122: (12)                                         The curvature of the corners of the :class:`Polygram`.
123: (8)                                         evenly_distribute_anchors
124: (12)                                         Break long line segments into proportionally-sized segments.
125: (8)                                         components_per_rounded_corner
126: (12)                                         The number of points used to represent the rounded corner curve.
127: (8)                                         .. seealso::
128: (12)                                         :class:`.~RoundedRectangle`
129: (8)                                         .. note::
130: (12)                                         If `radius` is supplied as a single value, then the same radius
131: (12)                                         will be applied to all corners. If `radius` is a list, then the
132: (12)                                         individual values will be applied sequentially, with the first
133: (12)                                         corner receiving `radius[0]`, the second corner receiving
134: (12)                                         `radius[1]`, etc. The radius list will be repeated as necessary.
135: (12)                                         The `components_per_rounded_corner` value is provided so that the
136: (12)                                         fidelity of the rounded corner may be fine-tuned as needed. 2 is
137: (12)                                         an appropriate value for most shapes, however a larger value may
be
138: (12)                                         need if the rounded corner is particularly large. 2 is the
minimum
139: (12)                                         number allowed, representing the start and end of the curve. 3
will
140: (12)                                         result in a start, middle, and end point, meaning 2 curves will be
generated.
141: (12)                                         The option to `evenly_distribute_anchors` is provided so that the
142: (12)                                         line segments (the part part of each line remaining after rounding
143: (12)                                         off the corners) can be subdivided to a density similar to that of
144: (12)                                         the average density of the rounded corners. This may be desirable
145: (12)                                         in situations in which an even distribution of curves is desired
146: (12)                                         for use in later transformation animations. Be aware, though,
147: (12)                                         enabling this option can result in an object containing
that
148: (12)                                         significantly more points than the original, especially when the
149: (12)                                         rounded corner curves are small.
150: (12)                                         Examples
151: (8)                                         -----
152: (8)                                         .. manim:: PolygramRoundCorners
153: (8)                                         :save_last_frame:
154: (12)                                         class PolygramRoundCorners(Scene):
155: (12)                                         def construct(self):
156: (16)

```

```

157: (20)                         star = Star(outer_radius=2)
158: (20)                         shapes = VGroup(star)
159: (20)                         shapes.add(star.copy().round_corners(radius=0.1))
160: (20)                         shapes.add(star.copy().round_corners(radius=0.25))
161: (20)                         shapes.arrange(RIGHT)
162: (20)                         self.add(shapes)
163: (8)                          """
164: (8)                          if radius == 0:
165: (12)                          return self
166: (8)                          new_points = []
167: (8)                          for vertices in self.get_vertex_groups():
168: (12)                          arcs = []
169: (12)                          if isinstance(radius, (int, float)):
170: (16)                          radius_list = [radius] * len(vertices)
171: (12)                          else:
172: (16)                          radius_list = radius * ceil(len(vertices) / len(radius))
173: (12)                          for currentRadius, (v1, v2, v3) in zip(
174: (16)                            radius_list, adjacent_n_tuples(vertices, 3)
175: (12)                          ):
176: (16)                            vect1 = v2 - v1
177: (16)                            vect2 = v3 - v2
178: (16)                            unit_vect1 = normalize(vect1)
179: (16)                            unit_vect2 = normalize(vect2)
180: (16)                            angle = angle_between_vectors(vect1, vect2)
181: (16)                            angle *= np.sign(currentRadius)
182: (16)                            cut_off_length = currentRadius * np.tan(angle / 2)
183: (16)                            sign = np.sign(np.cross(vect1, vect2)[2])
184: (16)                            arc = ArcBetweenPoints(
185: (20)                              v2 - unit_vect1 * cut_off_length,
186: (20)                              v2 + unit_vect2 * cut_off_length,
187: (20)                              angle=sign * angle,
188: (20)                              num_components=components_per_rounded_corner,
189: (16)                            )
190: (16)                            arcs.append(arc)
191: (12)                          if evenly_distribute_anchors:
192: (16)                            nonZeroLengthArcs = [arc for arc in arcs if len(arc.points) >
4]
193: (16)                            if len(nonZeroLengthArcs):
194: (20)                              totalArcLength = sum(
195: (24)                                [arc.get_arc_length() for arc in nonZeroLengthArcs]
196: (20)                              )
197: (20)                              totalCurveCount = (
198: (24)                                sum([len(arc.points) for arc in nonZeroLengthArcs]) /
4
199: (20)                                )
200: (20)                                averageLengthPerCurve = totalArcLength / totalCurveCount
201: (16)                                else:
202: (20)                                  averageLengthPerCurve = 1
203: (12)                                arcs = [arcs[-1], *arcs[:-1]]
204: (12)                                from manim.mobject.geometry.line import Line
205: (12)                                for arc1, arc2 in adjacent_pairs(arcs):
206: (16)                                  new_points.extend(arc1.points)
207: (16)                                  line = Line(arc1.get_end(), arc2.get_start())
208: (16)                                  if evenly_distribute_anchors:
209: (20)                                    line.insert_n_curves(
210: (24)                                      ceil(line.get_length() / averageLengthPerCurve)
211: (20)                                    )
212: (16)                                    new_points.extend(line.points)
213: (8)                                    self.set_points(new_points)
214: (8)                                    return self
215: (0)                                class Polygon(Polygram):
216: (4)                                  """A shape consisting of one closed loop of vertices.
217: (4)                                  Parameters
218: (4)                                  -----
219: (4)                                  vertices
220: (8)                                    The vertices of the :class:`Polygon`.
221: (4)                                  kwargs
222: (8)                                    Forwarded to the parent constructor.
223: (4)                                  Examples

```

```

224: (4)          -----
225: (4)          .. manim:: PolygonExample
226: (8)          :save_last_frame:
227: (8)          class PolygonExample(Scene):
228: (12)          def construct(self):
229: (16)          isosceles = Polygon([-5, 1.5, 0], [-2, 1.5, 0], [-3.5, -2, 0])
230: (16)          position_list = [
231: (20)          [4, 1, 0], # middle right
232: (20)          [4, -2.5, 0], # bottom right
233: (20)          [0, -2.5, 0], # bottom left
234: (20)          [0, 3, 0], # top left
235: (20)          [2, 1, 0], # middle
236: (20)          [4, 3, 0], # top right
237: (16)        ]
238: (16)        square_and_triangles = Polygon(*position_list, color=PURPLE_B)
239: (16)        self.add(isosceles, square_and_triangles)
240: (4)      """
241: (4)      def __init__(self, *vertices: Point3D, **kwargs) -> None:
242: (8)          super().__init__(vertices, **kwargs)
243: (0)      class RegularPolygram(Polygram):
244: (4)          """A :class:`Polygram` with regularly spaced vertices.
245: (4)          Parameters
246: (4)          -----
247: (4)          num_vertices
248: (8)          The number of vertices.
249: (4)          density
250: (8)          The density of the :class:`RegularPolygram`.
251: (8)          Can be thought of as how many vertices to hop
252: (8)          to draw a line between them. Every ``density``-th
253: (8)          vertex is connected.
254: (4)          radius
255: (8)          The radius of the circle that the vertices are placed on.
256: (4)          start_angle
257: (8)          The angle the vertices start at; the rotation of
258: (8)          the :class:`RegularPolygram`.
259: (4)          kwargs
260: (8)          Forwarded to the parent constructor.
261: (4)          Examples
262: (4)          -----
263: (4)          .. manim:: RegularPolygramExample
264: (8)          :save_last_frame:
265: (8)          class RegularPolygramExample(Scene):
266: (12)          def construct(self):
267: (16)          pentagram = RegularPolygram(5, radius=2)
268: (16)          self.add(pentagram)
269: (4)      """
270: (4)      def __init__(
271: (8)          self,
272: (8)          num_vertices: int,
273: (8)          *,
274: (8)          density: int = 2,
275: (8)          radius: float = 1,
276: (8)          start_angle: float | None = None,
277: (8)          **kwargs,
278: (4)      ) -> None:
279: (8)          num_gons = np.gcd(num_vertices, density)
280: (8)          num_vertices //= num_gons
281: (8)          density //= num_gons
282: (8)          def gen_polygon_vertices(start_angle):
283: (12)          reg_vertices, start_angle = regular_vertices(
284: (16)              num_vertices,
285: (16)              radius=radius,
286: (16)              start_angle=start_angle,
287: (12)          )
288: (12)          vertices = []
289: (12)          i = 0
290: (12)          while True:
291: (16)              vertices.append(reg_vertices[i])
292: (16)              i += density

```

```

293: (16)           i %= num_vertices
294: (16)           if i == 0:
295: (20)             break
296: (12)           return vertices, start_angle
297: (8)            first_group, self.start_angle = gen_polygon_vertices(start_angle)
298: (8)            vertex_groups = [first_group]
299: (8)            for i in range(1, num_gons):
300: (12)              start_angle = self.start_angle + (i / num_gons) * TAU /
num_vertices
301: (12)                  group, _ = gen_polygon_vertices(start_angle)
302: (12)                  vertex_groups.append(group)
303: (8)                  super().__init__(vertex_groups, **kwargs)
304: (0)  class RegularPolygon(RegularPolygram):
305: (4)    """An n-sided regular :class:`Polygon`.
306: (4)    Parameters
307: (4)    -----
308: (4)    n
309: (8)      The number of sides of the :class:`RegularPolygon`.
310: (4)    kwargs
311: (8)      Forwarded to the parent constructor.
312: (4)    Examples
313: (4)    -----
314: (4)    .. manim:: RegularPolygonExample
315: (8)      :save_last_frame:
316: (8)      class RegularPolygonExample(Scene):
317: (12)        def construct(self):
318: (16)          poly_1 = RegularPolygon(n=6)
319: (16)          poly_2 = RegularPolygon(n=6, start_angle=30*DEGREES,
color=GREEN)
320: (16)          poly_3 = RegularPolygon(n=10, color=RED)
321: (16)          poly_group = Group(poly_1, poly_2,
poly_3).scale(1.5).arrange(buff=1)
322: (16)          self.add(poly_group)
323: (4)        """
324: (4)        def __init__(self, n: int = 6, **kwargs) -> None:
325: (8)          super().__init__(n, density=1, **kwargs)
326: (0)  class Star(Polygon):
327: (4)    """A regular polygram without the intersecting lines.
328: (4)    Parameters
329: (4)    -----
330: (4)    n
331: (8)      How many points on the :class:`Star`.
332: (4)    outer_radius
333: (8)      The radius of the circle that the outer vertices are placed on.
334: (4)    inner_radius
335: (8)      The radius of the circle that the inner vertices are placed on.
336: (8)      If unspecified, the inner radius will be
337: (8)      calculated such that the edges of the :class:`Star`
338: (8)      perfectly follow the edges of its :class:`RegularPolygram`
counterpart.
339: (8)
340: (4)    density
341: (8)      The density of the :class:`Star`. Only used if
342: (8)      ``inner_radius`` is unspecified.
343: (8)      See :class:`RegularPolygram` for more information.
344: (4)    start_angle
345: (8)      The angle the vertices start at; the rotation of
346: (8)      the :class:`Star`.
347: (4)    kwargs
348: (8)      Forwardeds to the parent constructor.
349: (4)    Raises
350: (4)    -----
351: (4)    :exc:`ValueError`
352: (8)      If ``inner_radius`` is unspecified and ``density``
353: (8)      is not in the range ``[1, n/2)``.
354: (4)    Examples
355: (4)    -----
356: (4)    .. manim:: StarExample
357: (8)      :save_as_gif:
358: (8)      class StarExample(Scene):

```

```

359: (12)
360: (16)
361: (16)
362: (16)
363: (16)
364: (16)
365: (4) .. manim:: DifferentDensitiesExample
366: (8) :save_last_frame:
367: (8) class DifferentDensitiesExample(Scene):
368: (12)     def construct(self):
369: (16)         pentagram = RegularPolygram(5, radius=2)
370: (16)         star = Star(outer_radius=2, color=RED)
371: (16)         self.add(pentagram)
372: (4)         self.play(Create(star), run_time=3)
373: (4)         self.play(FadeOut(star), run_time=2)
374: (8)
375: (8)
376: (8)
377: (8)
378: (8)
379: (8)
380: (8)
381: (8)
382: (4)
383: (8)
384: (8)
385: (12)     ) -> None:
386: (16)         inner_angle = TAU / (2 * n)
387: (20)         if inner_radius is None:
388: {n}, (16)
389: (12)
390: (12)
391: (16)
392: (12)
393: (12)
394: (8)
395: (12)
396: (12)
397: (12)
398: (8)
399: (8)
400: (12)
401: (12)
402: (12)
403: (8)
404: (8)
405: (8)
406: (12)
407: (8)
408: (0)     class Triangle(RegularPolygon):
409: (4)         """An equilateral triangle.
410: (4)         Parameters
411: (4)         -----
412: (4)         kwargs
413: (8)             Additional arguments to be passed to :class:`RegularPolygon`  

414: (4)         Examples
415: (4)
416: (4)         .. manim:: TriangleExample
417: (8)         :save_last_frame:
418: (8)         class TriangleExample(Scene):
419: (12)             def construct(self):
420: (16)                 triangle_1 = Triangle()
421: (16)                 triangle_2 = Triangle().scale(2).rotate(60*DEGREES)
422: (16)                 tri_group = Group(triangle_1, triangle_2).arrange(buff=1)
423: (16)                 self.add(tri_group)
424: (4)
425: (4)             def __init__(self, **kwargs) -> None:
426: (8)                 super().__init__(n=3, **kwargs)

```

```

427: (0)         class Rectangle(Polygon):
428: (4)             """A quadrilateral with two sets of parallel sides.
429: (4)             Parameters
430: (4)             -----
431: (4)             color
432: (8)                 The color of the rectangle.
433: (4)             height
434: (8)                 The vertical height of the rectangle.
435: (4)             width
436: (8)                 The horizontal width of the rectangle.
437: (4)             grid_xstep
438: (8)                 Space between vertical grid lines.
439: (4)             grid_ystep
440: (8)                 Space between horizontal grid lines.
441: (4)             mark_paths_closed
442: (8)                 No purpose.
443: (4)             close_new_points
444: (8)                 No purpose.
445: (4)             kwargs
446: (8)                 Additional arguments to be passed to :class:`Polygon`
447: (4)             Examples
448: (4)             -----
449: (4)             .. manim:: RectangleExample
450: (8)                 :save_last_frame:
451: (8)                 class RectangleExample(Scene):
452: (12)                     def construct(self):
453: (16)                         rect1 = Rectangle(width=4.0, height=2.0, grid_xstep=1.0,
grid_ystep=0.5)
454: (16)                         rect2 = Rectangle(width=1.0, height=4.0)
455: (16)                         rect3 = Rectangle(width=2.0, height=2.0, grid_xstep=1.0,
grid_ystep=1.0)
456: (16)                         rect3.grid_lines.set_stroke(width=1)
457: (16)                         rects = Group(rect1, rect2, rect3).arrange(buff=1)
458: (16)                         self.add(rects)
459: (4)             """
460: (4)             def __init__(
461: (8)                 self,
462: (8)                 color: ParsableManimColor = WHITE,
463: (8)                 height: float = 2.0,
464: (8)                 width: float = 4.0,
465: (8)                 grid_xstep: float | None = None,
466: (8)                 grid_ystep: float | None = None,
467: (8)                 mark_paths_closed: bool = True,
468: (8)                 close_new_points: bool = True,
469: (8)                 **kwargs,
470: (4)             ):
471: (8)                 super().__init__(UR, UL, DL, DR, color=color, **kwargs)
472: (8)                 self.stretch_to_fit_width(width)
473: (8)                 self.stretch_to_fit_height(height)
474: (8)                 v = self.get_vertices()
475: (8)                 self.grid_lines = VGroup()
476: (8)                 if grid_xstep or grid_ystep:
477: (12)                     from manim.mobject.geometry.line import Line
478: (12)                     v = self.get_vertices()
479: (8)                 if grid_xstep:
480: (12)                     grid_xstep = abs(grid_xstep)
481: (12)                     count = int(width / grid_xstep)
482: (12)                     grid = VGroup(
483: (16)                         *
484: (20)                             Line(
485: (24)                                 v[1] + i * grid_xstep * RIGHT,
486: (24)                                 v[1] + i * grid_xstep * RIGHT + height * DOWN,
487: (24)                                 color=color,
488: (20)                             )
489: (20)                             for i in range(1, count)
490: (16)                         )
491: (12)                     self.grid_lines.add(grid)
492: (12)                 if grid_ystep:
493: (8)

```

```

494: (12)             grid_ystep = abs(grid_ystep)
495: (12)             count = int(height / grid_ystep)
496: (12)             grid = VGroup(
497: (16)                 *)
498: (20)                     Line(
499: (24)                         v[1] + i * grid_ystep * DOWN,
500: (24)                         v[1] + i * grid_ystep * DOWN + width * RIGHT,
501: (24)                         color=color,
502: (20)                     )
503: (20)                 for i in range(1, count)
504: (16)             )
505: (12)         )
506: (12)             self.grid_lines.add(grid)
507: (8)             if self.grid_lines:
508: (12)                 self.add(self.grid_lines)
509: (0)             class Square(Rectangle):
510: (4)                 """A rectangle with equal side lengths.
511: (4)                 Parameters
512: (4)                 -----
513: (4)                 side_length
514: (8)                     The length of the sides of the square.
515: (4)                 kwargs
516: (8)                     Additional arguments to be passed to :class:`Rectangle`.
517: (4)                 Examples
518: (4)                 -----
519: (4)             .. manim:: SquareExample
520: (8)                 :save_last_frame:
521: (8)                 class SquareExample(Scene):
522: (12)                     def construct(self):
523: (16)                         square_1 = Square(side_length=2.0).shift(DOWN)
524: (16)                         square_2 = Square(side_length=1.0).next_to(square_1,
525: (16)                         direction=UP)
526: (16)                         square_3 = Square(side_length=0.5).next_to(square_2,
527: (16)                         direction=UP)
528: (16)                         self.add(square_1, square_2, square_3)
529: (4)             """
530: (8)             def __init__(self, side_length: float = 2.0, **kwargs) -> None:
531: (8)                 self.side_length = side_length
532: (8)                 super().__init__(height=side_length, width=side_length, **kwargs)
533: (0)             class RoundedRectangle(Rectangle):
534: (4)                 """A rectangle with rounded corners.
535: (4)                 Parameters
536: (4)                 -----
537: (4)                 corner_radius
538: (8)                     The curvature of the corners of the rectangle.
539: (4)                 kwargs
540: (4)                     Additional arguments to be passed to :class:`Rectangle`.
541: (4)                 Examples
542: (4)                 -----
543: (4)             .. manim:: RoundedRectangleExample
544: (8)                 :save_last_frame:
545: (8)                 class RoundedRectangleExample(Scene):
546: (12)                     def construct(self):
547: (16)                         rect_1 = RoundedRectangle(corner_radius=0.5)
548: (16)                         rect_2 = RoundedRectangle(corner_radius=1.5, height=4.0,
549: (16)                         width=4.0)
550: (16)                         rect_group = Group(rect_1, rect_2).arrange(buff=1)
551: (16)                         self.add(rect_group)
552: (4)             """
553: (8)             def __init__(self, corner_radius: float | list[float] = 0.5, **kwargs):
554: (8)                 super().__init__(**kwargs)
555: (8)                 self.corner_radius = corner_radius
556: (8)                 self.round_corners(self.corner_radius)
557: (0)             class Cutout(VMobject, metaclass=ConvertToOpenGL):
558: (4)                 """A shape with smaller cutouts.
559: (4)                 Parameters
559: (4)                 -----
559: (4)                 main_shape
559: (8)                     The primary shape from which cutouts are made.

```

```

560: (4)             mobjects
561: (8)             The smaller shapes which are to be cut out of the ``main_shape``.
562: (4)             kwargs
563: (8)             Further keyword arguments that are passed to the constructor of
564: (8)             :class:`~.VMobject`.
565: (4)             .. warning::
566: (8)             Technically, this class behaves similar to a symmetric difference: if
567: (8)             parts of the ``mobjects`` are not located within the ``main_shape``,
568: (8)             these parts will be added to the resulting :class:`~.VMobject`.
569: (4)             Examples
570: (4)             -----
571: (4)             .. manim:: CutoutExample
572: (8)             class CutoutExample(Scene):
573: (12)             def construct(self):
574: (16)                 s1 = Square().scale(2.5)
575: (16)                 s2 = Triangle().shift(DOWN + RIGHT).scale(0.5)
576: (16)                 s3 = Square().shift(UP + RIGHT).scale(0.5)
577: (16)                 s4 = RegularPolygon(5).shift(DOWN + LEFT).scale(0.5)
578: (16)                 s5 = RegularPolygon(6).shift(UP + LEFT).scale(0.5)
579: (16)                 c = Cutout(s1, s2, s3, s4, s5, fill_opacity=1, color=BLUE,
stroke_color=RED)
580: (16)                     self.play(Write(c), run_time=4)
581: (16)                     self.wait()
582: (4)             """
583: (4)             def __init__(self, main_shape: VMobject, *mobjects: VMobject, **kwargs) ->
None:
584: (8)                 super().__init__(**kwargs)
585: (8)                 self.append_points(main_shape.points)
586: (8)                 if main_shape.get_direction() == "CW":
587: (12)                     sub_direction = "CCW"
588: (8)                 else:
589: (12)                     sub_direction = "CW"
590: (8)                 for mobject in mobjects:
591: (12)                     self.append_points(mobject.force_direction(sub_direction).points)

-----

```

## File 61 - \_\_init\_\_.py:

```

1: (0)             """Various geometric Mobjects.
2: (0)             Modules
3: (0)             =====
4: (0)             .. autosummary::
5: (4)                 :toctree: ../reference
6: (4)                 ~arc
7: (4)                 ~boolean_ops
8: (4)                 ~labeled
9: (4)                 ~line
10: (4)                ~polygram
11: (4)                ~shape_matchers
12: (4)                ~tips
13: (0)             """

```

## File 62 - \_\_init\_\_.py:

```

1: (0)             """Coordinate systems and function graphing related mobjects.
2: (0)             Modules
3: (0)             =====
4: (0)             .. autosummary::
5: (4)                 :toctree: ../reference
6: (4)                 ~coordinate_systems
7: (4)                 ~functions
8: (4)                 ~number_line
9: (4)                 ~probability
10: (4)                ~scale
11: (0)             """

```

-----  
File 63 - \_\_init\_\_.py:

1: (0)

-----

File 64 - constants.py:

```
1: (0)      """
2: (0)      Constant definitions.
3: (0)      """
4: (0)      from __future__ import annotations
5: (0)      from enum import Enum
6: (0)      import numpy as np
7: (0)      from cloup import Context
8: (0)      from PIL.Image import Resampling
9: (0)      from manim.typing import Vector3D
10: (0)     __all__ = [
11: (4)         "SCENE_NOT_FOUND_MESSAGE",
12: (4)         "CHOOSE_NUMBER_MESSAGE",
13: (4)         "INVALID_NUMBER_MESSAGE",
14: (4)         "NO_SCENE_MESSAGE",
15: (4)         "NORMAL",
16: (4)         "ITALIC",
17: (4)         "OBLIQUE",
18: (4)         "BOLD",
19: (4)         "THIN",
20: (4)         "ULTRALIGHT",
21: (4)         "LIGHT",
22: (4)         "SEMILIGHT",
23: (4)         "BOOK",
24: (4)         "MEDIUM",
25: (4)         "SEMIBOLD",
26: (4)         "ULTRABOLD",
27: (4)         "HEAVY",
28: (4)         "ULTRAHEAVY",
29: (4)         "RESAMPLING_ALGORITHMS",
30: (4)         "ORIGIN",
31: (4)         "UP",
32: (4)         "DOWN",
33: (4)         "RIGHT",
34: (4)         "LEFT",
35: (4)         "IN",
36: (4)         "OUT",
37: (4)         "X_AXIS",
38: (4)         "Y_AXIS",
39: (4)         "Z_AXIS",
40: (4)         "UL",
41: (4)         "UR",
42: (4)         "DL",
43: (4)         "DR",
44: (4)         "START_X",
45: (4)         "START_Y",
46: (4)         "DEFAULT_DOT_RADIUS",
47: (4)         "DEFAULT_SMALL_DOT_RADIUS",
48: (4)         "DEFAULT_DASH_LENGTH",
49: (4)         "DEFAULT_ARROW_TIP_LENGTH",
50: (4)         "SMALL_BUFF",
51: (4)         "MED_SMALL_BUFF",
52: (4)         "MED_LARGE_BUFF",
53: (4)         "LARGE_BUFF",
54: (4)         "DEFAULT_MOBJECT_TO_EDGE_BUFFER",
55: (4)         "DEFAULT_MOBJECT_TO_MOBJECT_BUFFER",
56: (4)         "DEFAULT_POINTWISE_FUNCTION_RUN_TIME",
57: (4)         "DEFAULT_WAIT_TIME",
58: (4)         "DEFAULT_POINT_DENSITY_2D",
59: (4)         "DEFAULT_POINT_DENSITY_1D",
```

```

60: (4)             "DEFAULT_STROKE_WIDTH",
61: (4)             "DEFAULT_FONT_SIZE",
62: (4)             "SCALE_FACTOR_PER_FONT_POINT",
63: (4)             "PI",
64: (4)             "TAU",
65: (4)             "DEGREES",
66: (4)             "QUALITIES",
67: (4)             "DEFAULT_QUALITY",
68: (4)             "EPILOG",
69: (4)             "CONTEXT_SETTINGS",
70: (4)             "SHIFT_VALUE",
71: (4)             "CTRL_VALUE",
72: (4)             "RendererType",
73: (4)             "LineJointType",
74: (4)             "CapStyleType",
75: (0)
76: (0)             SCENE_NOT_FOUND_MESSAGE = """
77: (3)             {} is not in the script
78: (0)
79: (0)
80: (0)             CHOOSE_NUMBER_MESSAGE = """
81: (0)             Choose number corresponding to desired scene/arguments.
82: (0)             (Use comma separated list for multiple entries)
83: (0)             Choice(s): """
84: (0)             INVALID_NUMBER_MESSAGE = "Invalid scene numbers have been specified.
Aborting."
85: (3)
86: (0)
87: (0)             NO_SCENE_MESSAGE = """
88: (0)             There are no scenes inside that module
89: (0)
90: (0)             NORMAL = "NORMAL"
91: (0)             ITALIC = "ITALIC"
92: (0)             OBLIQUE = "OBLIQUE"
93: (0)             BOLD = "BOLD"
94: (0)             THIN = "THIN"
95: (0)             ULTRALIGHT = "ULTRALIGHT"
96: (0)             LIGHT = "LIGHT"
97: (0)             SEMILIGHT = "SEMILIGHT"
98: (0)             BOOK = "BOOK"
99: (0)             MEDIUM = "MEDIUM"
100: (0)            SEMIBOLD = "SEMIBOLD"
101: (0)            ULTRABOLD = "ULTRABOLD"
102: (0)            HEAVY = "HEAVY"
103: (0)            ULTRAHEAVY = "ULTRAHEAVY"
104: (0)            RESAMPLING_ALGORITHMS = {
105: (4)              "nearest": Resampling.NEAREST,
106: (4)              "none": Resampling.NEAREST,
107: (4)              "lanczos": Resampling.LANZOS,
108: (4)              "antialias": Resampling.LANZOS,
109: (4)              "bilinear": Resampling.BILINEAR,
110: (4)              "linear": Resampling.BILINEAR,
111: (4)              "bicubic": Resampling.BICUBIC,
112: (0)              "cubic": Resampling.BICUBIC,
113: (0)              "box": Resampling.BOX,
114: (0)              "hamming": Resampling.HAMMING,
115: (0)
116: (0)              ORIGIN: Vector3D = np.array((0.0, 0.0, 0.0))
117: (0)              """The center of the coordinate system."""
118: (0)
119: (0)              UP: Vector3D = np.array((0.0, 1.0, 0.0))
120: (0)              """One unit step in the positive Y direction."""
121: (0)              DOWN: Vector3D = np.array((0.0, -1.0, 0.0))
122: (0)              """One unit step in the negative Y direction."""
123: (0)              RIGHT: Vector3D = np.array((1.0, 0.0, 0.0))
124: (0)              """One unit step in the positive X direction."""
125: (0)              LEFT: Vector3D = np.array((-1.0, 0.0, 0.0))
126: (0)              """One unit step in the negative X direction."""
127: (0)              IN: Vector3D = np.array((0.0, 0.0, -1.0))
128: (0)              """One unit step in the negative Z direction."""
129: (0)              OUT: Vector3D = np.array((0.0, 0.0, 1.0))
130: (0)              """One unit step in the positive Z direction."""
131: (0)              X_AXIS: Vector3D = np.array((1.0, 0.0, 0.0))

```

```

128: (0)             Y_AXIS: Vector3D = np.array((0.0, 1.0, 0.0))
129: (0)             Z_AXIS: Vector3D = np.array((0.0, 0.0, 1.0))
130: (0)             UL: Vector3D = UP + LEFT
131: (0)             """One step up plus one step left."""
132: (0)             UR: Vector3D = UP + RIGHT
133: (0)             """One step up plus one step right."""
134: (0)             DL: Vector3D = DOWN + LEFT
135: (0)             """One step down plus one step left."""
136: (0)             DR: Vector3D = DOWN + RIGHT
137: (0)             """One step down plus one step right."""
138: (0)             START_X = 30
139: (0)             START_Y = 20
140: (0)             DEFAULT_DOT_RADIUS = 0.08
141: (0)             DEFAULT_SMALL_DOT_RADIUS = 0.04
142: (0)             DEFAULT_DASH_LENGTH = 0.05
143: (0)             DEFAULT_ARROW_TIP_LENGTH = 0.35
144: (0)             SMALL_BUFF = 0.1
145: (0)             MED_SMALL_BUFF = 0.25
146: (0)             MED_LARGE_BUFF = 0.5
147: (0)             LARGE_BUFF = 1
148: (0)             DEFAULT_MOBJECT_TO_EDGE_BUFFER = MED_LARGE_BUFF
149: (0)             DEFAULT_MOBJECT_TO_MOBJECT_BUFFER = MED_SMALL_BUFF
150: (0)             DEFAULT_POINTWISE_FUNCTION_RUN_TIME = 3.0
151: (0)             DEFAULT_WAIT_TIME = 1.0
152: (0)             DEFAULT_POINT_DENSITY_2D = 25
153: (0)             DEFAULT_POINT_DENSITY_1D = 10
154: (0)             DEFAULT_STROKE_WIDTH = 4
155: (0)             DEFAULT_FONT_SIZE = 48
156: (0)             SCALE_FACTOR_PER_FONT_POINT = 1 / 960
157: (0)             PI = np.pi
158: (0)             """The ratio of the circumference of a circle to its diameter."""
159: (0)             TAU = 2 * PI
160: (0)             """The ratio of the circumference of a circle to its radius."""
161: (0)             DEGREES = TAU / 360
162: (0)             """The exchange rate between radians and degrees."""
163: (0)             QUALITIES: dict[str, dict[str, str | int | None]] = {
164: (4)               "fourk_quality": {
165: (8)                 "flag": "k",
166: (8)                 "pixel_height": 2160,
167: (8)                 "pixel_width": 3840,
168: (8)                 "frame_rate": 60,
169: (4),
170: (4)               "production_quality": {
171: (8)                 "flag": "p",
172: (8)                 "pixel_height": 1440,
173: (8)                 "pixel_width": 2560,
174: (8)                 "frame_rate": 60,
175: (4),
176: (4)               "high_quality": {
177: (8)                 "flag": "h",
178: (8)                 "pixel_height": 1080,
179: (8)                 "pixel_width": 1920,
180: (8)                 "frame_rate": 60,
181: (4),
182: (4)               "medium_quality": {
183: (8)                 "flag": "m",
184: (8)                 "pixel_height": 720,
185: (8)                 "pixel_width": 1280,
186: (8)                 "frame_rate": 30,
187: (4),
188: (4)               "low_quality": {
189: (8)                 "flag": "l",
190: (8)                 "pixel_height": 480,
191: (8)                 "pixel_width": 854,
192: (8)                 "frame_rate": 15,
193: (4),
194: (4)               "example_quality": {
195: (8)                 "flag": None,
196: (8)                 "pixel_height": 480,

```

```

197: (8)             "pixel_width": 854,
198: (8)             "frame_rate": 30,
199: (4)
200: (0)
201: (0)
202: (0)             DEFAULT_QUALITY = "high_quality"
203: (0)             EPILOG = "Made with <3 by Manim Community developers."
204: (0)             SHIFT_VALUE = 65505
205: (0)             CTRL_VALUE = 65507
206: (4)             CONTEXT_SETTINGS = Context.settings(
207: (4)                 align_option_groups=True,
208: (4)                 align_sections=True,
209: (4)                 show_constraints=True,
210: (0)
211: (0)             class RendererType(Enum):
212: (4)                 """An enumeration of all renderer types that can be assigned to
213: (4)                 the ``config.renderer`` attribute.
214: (4)                 Manim's configuration allows assigning string values to the renderer
215: (4)                 setting, the values are then replaced by the corresponding enum object.
216: (4)                 In other words, you can run::
217: (8)                     config.renderer = "opengl"
218: (4)                 and checking the renderer afterwards reveals that the attribute has
219: (4)                 assumed the value::
220: (8)                     <RendererType.OPENGL: 'opengl'>
221: (4)
222: (4)             CAIRO = "cairo" #: A renderer based on the cairo backend.
223: (0)             OPENGL = "opengl" #: An OpenGL-based renderer.
224: (0)
225: (0)             class LineJointType(Enum):
226: (4)                 """Collection of available line joint types.
227: (4)                 See the example below for a visual illustration of the different
228: (4)                 joint types.
229: (4)                 Examples
230: (4)                 -----
231: (8)                     .. manim:: LineJointVariants
232: (12)                     :save_last_frame:
233: (16)                     class LineJointVariants(Scene):
234: (20)                         def construct(self):
235: (20)                             mob = VMobject(stroke_width=20,
236: (20)                                         color=GREEN).set_points_as_corners([
237: (16)                                             np.array([-2, 0, 0]),
238: (16)                                             np.array([0, 0, 0]),
239: (16)                                             np.array([-2, 1, 0]),
240: (16)                                         ])
241: (16)                                         lines = VGroup(*[mob.copy() for _ in
242: (16)                                         range(len(LineJointType))])
243: (16)                                         for line, joint_type in zip(lines, LineJointType):
244: (20)                                             line.joint_type = joint_type
245: (20)                                             lines.arrange(RIGHT, buff=1)
246: (4)                                             self.add(lines)
247: (4)                                             for line in lines:
248: (20)                                                 label = Text(line.joint_type.name).next_to(line, DOWN)
249: (20)                                                 self.add(label)
250: (4)
251: (0)
252: (4)             AUTO = 0
253: (4)             ROUND = 1
254: (4)             BEVEL = 2
255: (4)             MITER = 3
256: (0)
257: (0)             class CapStyleType(Enum):
258: (4)                 """Collection of available cap styles.
259: (4)                 See the example below for a visual illustration of the different
260: (4)                 cap styles.
261: (4)                 Examples
262: (4)                 -----
263: (4)                     .. manim:: CapStyleVariants
264: (4)                     :save_last_frame:
265: (8)                     class CapStyleVariants(Scene):
266: (12)                     def construct(self):
267: (16)                         arcs = VGroup(*[
268: (20)                             Arc(
269: (24)                               radius=1,
```

```

264: (24)                         start_angle=0,
265: (24)                         angle=TAU / 4,
266: (24)                         stroke_width=20,
267: (24)                         color=GREEN,
268: (24)                         cap_style=cap_style,
269: (20)                     )
270: (20)                 for cap_style in CapStyleType
271: (16)             ])
272: (16)         arcs.arrange(RIGHT, buff=1)
273: (16)         self.add(arcs)
274: (16)         for arc in arcs:
275: (20)             label = Text(arc.cap_style.name,
font_size=24).next_to(arc, DOWN)
276: (20)                     self.add(label)
277: (4)             """
278: (4)             AUTO = 0
279: (4)             ROUND = 1
280: (4)             BUTT = 2
281: (4)             SQUARE = 3

```

---

## File 65 - functions.py:

```

1: (0)             """Mobjects representing function graphs."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["ParametricFunction", "FunctionGraph", "ImplicitFunction"]
4: (0)             from typing import TYPE_CHECKING, Callable, Iterable, Sequence
5: (0)             import numpy as np
6: (0)             from isosurfaces import plot_isoline
7: (0)             from manim import config
8: (0)             from manim.mobject.graphing.scale import LinearBase, _ScaleBase
9: (0)             from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
10: (0)            from manim.mobject.types.vectorized_mobject import VMobject
11: (0)            if TYPE_CHECKING:
12: (4)                from manim.typing import Point2D, Point3D
13: (0)            from manim.utils.color import YELLOW
14: (0)            class ParametricFunction(VMobject, metaclass=ConvertToOpenGL):
15: (4)                """A parametric curve.
16: (4)                Parameters
17: (4)                -----
18: (4)                function
19: (8)                    The function to be plotted in the form of ``lambda t: (x(t), y(t),
z(t))``.
20: (4)                t_range
21: (8)                    Determines the length that the function spans in the form of (t_min,
t_max, step=0.01). By default ``[0, 1]``
22: (4)                scaling
23: (8)                    Scaling class applied to the points of the function. Default of
:class:`~.LinearBase`.
24: (4)                use_smoothing
25: (8)                    Whether to interpolate between the points of the function after they
have been created.
26: (8)                    (Will have odd behaviour with a low number of points)
27: (4)                use_vectorized
28: (8)                    Whether to pass in the generated t value array to the function as
``[t_0, t_1, ...]``.
29: (8)                    Only use this if your function supports it. Output should be a numpy
array
30: (8)                        of shape ``[[x_0, x_1, ...], [y_0, y_1, ...], [z_0, z_1, ...]]`` but
``z`` can
31: (8)                        also be 0 if the Axes is 2D
32: (4)                discontinuities
33: (8)                    Values of t at which the function experiences discontinuity.
34: (4)                dt
35: (8)                    The left and right tolerance for the discontinuities.
36: (4)                Examples
37: (4)                -----
38: (4)                .. manim:: PlotParametricFunction

```

```

39: (8)                      :save_last_frame:
40: (8)                      class PlotParametricFunction(Scene):
41: (12)                     def func(self, t):
42: (16)                     return (np.sin(2 * t), np.sin(3 * t), 0)
43: (12)                     def construct(self):
44: (16)                         func = ParametricFunction(self.func, t_range = (0, TAU),
fill_opacity=0).set_color(RED)
45: (16)                         self.add(func.scale(3))
46: (4) .. manim:: ThreeDParametricSpring
47: (8) :save_last_frame:
48: (8) class ThreeDParametricSpring(ThreeDScene):
49: (12)     def construct(self):
50: (16)         curve1 = ParametricFunction(
51: (20)             lambda u:
52: (24)                 1.2 * np.cos(u),
53: (24)                 1.2 * np.sin(u),
54: (24)                 u * 0.05
55: (20)             ), color=RED, t_range = (-3*TAU, 5*TAU, 0.01)
56: (16)         .set_shade_in_3d(True)
57: (16)         axes = ThreeDAxes()
58: (16)         self.add(axes, curve1)
59: (16)         self.set_camera_orientation(phi=80 * DEGREES, theta=-60 *
DEGREES)
60: (16)         self.wait()
61: (4) .. attention::
62: (8) If your function has discontinuities, you'll have to specify the
location
63: (8) of the discontinuities manually. See the following example for
guidance.
64: (4) .. manim:: DiscontinuousExample
65: (8) :save_last_frame:
66: (8) class DiscontinuousExample(Scene):
67: (12)     def construct(self):
68: (16)         ax1 = NumberPlane((-3, 3), (-4, 4))
69: (16)         ax2 = NumberPlane((-3, 3), (-4, 4))
70: (16)         VGroup(ax1, ax2).arrange()
71: (16)         discontinuous_function = lambda x: (x ** 2 - 2) / (x ** 2 - 4)
72: (16)         incorrect = ax1.plot(discontinuous_function, color=RED)
73: (16)         correct = ax2.plot(
74: (20)             discontinuous_function,
75: (20)             discontinuities=[-2, 2], # discontinuous points
76: (20)             dt=0.1, # left and right tolerance of discontinuity
77: (20)             color=GREEN,
78: (16)         )
79: (16)         self.add(ax1, ax2, incorrect, correct)
80: (4) """
81: (4)     def __init__(
82: (8)         self,
83: (8)         function: Callable[[float], Point3D],
84: (8)         t_range: Point2D | Point3D = (0, 1),
85: (8)         scaling: _ScaleBase = LinearBase(),
86: (8)         dt: float = 1e-8,
87: (8)         discontinuities: Iterable[float] | None = None,
88: (8)         use_smoothing: bool = True,
89: (8)         use_vectorized: bool = False,
90: (8)         **kwargs,
91: (4)     ):
92: (8)         self.function = function
93: (8)         t_range = (0, 1, 0.01) if t_range is None else t_range
94: (8)         if len(t_range) == 2:
95: (12)             t_range = np.array([*t_range, 0.01])
96: (8)         self.scaling = scaling
97: (8)         self.dt = dt
98: (8)         self.discontinuities = discontinuities
99: (8)         self.use_smoothing = use_smoothing
100: (8)        self.use_vectorized = use_vectorized
101: (8)        self.t_min, self.t_max, self.t_step = t_range
102: (8)        super().__init__(**kwargs)
103: (4)     def get_function(self):

```

```

104: (8)             return self.function
105: (4)         def get_point_from_function(self, t):
106: (8)             return self.function(t)
107: (4)         def generate_points(self):
108: (8)             if self.discontinuities is not None:
109: (12)                 discontinuities = filter(
110: (16)                     lambda t: self.t_min <= t <= self.t_max,
111: (16)                     self.discontinuities,
112: (12)                 )
113: (12)                 discontinuities = np.array(list(discontinuities))
114: (12)                 boundary_times = np.array(
115: (16)                     [
116: (20)                         self.t_min,
117: (20)                         self.t_max,
118: (20)                         *(discontinuities - self.dt),
119: (20)                         *(discontinuities + self.dt),
120: (16)                     ],
121: (12)                 )
122: (12)                 boundary_times.sort()
123: (8)             else:
124: (12)                 boundary_times = [self.t_min, self.t_max]
125: (8)             for t1, t2 in zip(boundary_times[0::2], boundary_times[1::2]):
126: (12)                 t_range = np.array(
127: (16)                     [
128: (20)                         *self.scaling.function(np.arange(t1, t2, self.t_step)),
129: (20)                         self.scaling.function(t2),
130: (16)                     ],
131: (12)                 )
132: (12)                 if self.use_vectorized:
133: (16)                     x, y, z = self.function(t_range)
134: (16)                     if not isinstance(z, np.ndarray):
135: (20)                         z = np.zeros_like(x)
136: (16)                     points = np.stack([x, y, z], axis=1)
137: (12)                 else:
138: (16)                     points = np.array([self.function(t) for t in t_range])
139: (12)                 self.start_new_path(points[0])
140: (12)                 self.add_points_as_corners(points[1:])
141: (8)             if self.use_smoothing:
142: (12)                 self.make_smooth()
143: (8)             return self
144: (4)         init_points = generate_points
145: (0)     class FunctionGraph(ParametricFunction):
146: (4)         """A :class:`ParametricFunction` that spans the length of the scene by
default.
147: (4)             Examples
148: (4)             -----
149: (4)             .. manim:: ExampleFunctionGraph
150: (8)                 :save_last_frame:
151: (8)                 class ExampleFunctionGraph(Scene):
152: (12)                     def construct(self):
153: (16)                         cos_func = FunctionGraph(
154: (20)                             lambda t: np.cos(t) + 0.5 * np.cos(7 * t) + (1 / 7) *
np.cos(14 * t),
155: (20)
156: (16)
157: (16)
158: (20)
159: (20)
160: (16)
161: (16)
162: (20)
163: (20)
164: (20)
165: (16)
166: (16)
167: (4)
168: (4)                         color=RED,
169: (12)                     )
170: (16)
171: (16)
172: (20)
173: (16)
174: (16)
175: (20)
176: (16)
177: (16)
178: (20)
179: (16)
180: (16)
181: (20)
182: (16)
183: (16)
184: (20)
185: (16)
186: (16)
187: (20)
188: (16)
189: (16)
190: (20)
191: (16)
192: (16)
193: (20)
194: (16)
195: (16)
196: (20)
197: (16)
198: (16)
199: (20)
200: (16)
201: (16)
202: (20)
203: (16)
204: (16)
205: (20)
206: (16)
207: (16)
208: (20)
209: (16)
210: (16)
211: (20)
212: (16)
213: (16)
214: (20)
215: (16)
216: (16)
217: (20)
218: (16)
219: (16)
220: (20)
221: (16)
222: (16)
223: (20)
224: (16)
225: (16)
226: (20)
227: (16)
228: (16)
229: (20)
230: (16)
231: (16)
232: (20)
233: (16)
234: (16)
235: (20)
236: (16)
237: (16)
238: (20)
239: (16)
240: (16)
241: (20)
242: (16)
243: (16)
244: (20)
245: (16)
246: (16)
247: (20)
248: (16)
249: (16)
250: (20)
251: (16)
252: (16)
253: (20)
254: (16)
255: (16)
256: (20)
257: (16)
258: (16)
259: (20)
260: (16)
261: (16)
262: (20)
263: (16)
264: (16)
265: (20)
266: (16)
267: (16)
268: (20)
269: (16)
270: (16)
271: (20)
272: (16)
273: (16)
274: (20)
275: (16)
276: (16)
277: (20)
278: (16)
279: (16)
280: (20)
281: (16)
282: (16)
283: (20)
284: (16)
285: (16)
286: (20)
287: (16)
288: (16)
289: (20)
290: (16)
291: (16)
292: (20)
293: (16)
294: (16)
295: (20)
296: (16)
297: (16)
298: (20)
299: (16)
300: (16)
301: (20)
302: (16)
303: (16)
304: (20)
305: (16)
306: (16)
307: (20)
308: (16)
309: (16)
310: (20)
311: (16)
312: (16)
313: (20)
314: (16)
315: (16)
316: (20)
317: (16)
318: (16)
319: (20)
320: (16)
321: (16)
322: (20)
323: (16)
324: (16)
325: (20)
326: (16)
327: (16)
328: (20)
329: (16)
330: (16)
331: (20)
332: (16)
333: (16)
334: (20)
335: (16)
336: (16)
337: (20)
338: (16)
339: (16)
340: (20)
341: (16)
342: (16)
343: (20)
344: (16)
345: (16)
346: (20)
347: (16)
348: (16)
349: (20)
350: (16)
351: (16)
352: (20)
353: (16)
354: (16)
355: (20)
356: (16)
357: (16)
358: (20)
359: (16)
360: (16)
361: (20)
362: (16)
363: (16)
364: (20)
365: (16)
366: (16)
367: (20)
368: (16)
369: (16)
370: (20)
371: (16)
372: (16)
373: (20)
374: (16)
375: (16)
376: (20)
377: (16)
378: (16)
379: (20)
380: (16)
381: (16)
382: (20)
383: (16)
384: (16)
385: (20)
386: (16)
387: (16)
388: (20)
389: (16)
390: (16)
391: (20)
392: (16)
393: (16)
394: (20)
395: (16)
396: (16)
397: (20)
398: (16)
399: (16)
400: (20)
401: (16)
402: (16)
403: (20)
404: (16)
405: (16)
406: (20)
407: (16)
408: (16)
409: (20)
410: (16)
411: (16)
412: (20)
413: (16)
414: (16)
415: (20)
416: (16)
417: (16)
418: (20)
419: (16)
420: (16)
421: (20)
422: (16)
423: (16)
424: (20)
425: (16)
426: (16)
427: (20)
428: (16)
429: (16)
430: (20)
431: (16)
432: (16)
433: (20)
434: (16)
435: (16)
436: (20)
437: (16)
438: (16)
439: (20)
440: (16)
441: (16)
442: (20)
443: (16)
444: (16)
445: (20)
446: (16)
447: (16)
448: (20)
449: (16)
450: (16)
451: (20)
452: (16)
453: (16)
454: (20)
455: (16)
456: (16)
457: (20)
458: (16)
459: (16)
460: (20)
461: (16)
462: (16)
463: (20)
464: (16)
465: (16)
466: (20)
467: (16)
468: (16)
469: (20)
470: (16)
471: (16)
472: (20)
473: (16)
474: (16)
475: (20)
476: (16)
477: (16)
478: (20)
479: (16)
480: (16)
481: (20)
482: (16)
483: (16)
484: (20)
485: (16)
486: (16)
487: (20)
488: (16)
489: (16)
490: (20)
491: (16)
492: (16)
493: (20)
494: (16)
495: (16)
496: (20)
497: (16)
498: (16)
499: (20)
500: (16)
501: (16)
502: (20)
503: (16)
504: (16)
505: (20)
506: (16)
507: (16)
508: (20)
509: (16)
510: (16)
511: (20)
512: (16)
513: (16)
514: (20)
515: (16)
516: (16)
517: (20)
518: (16)
519: (16)
520: (20)
521: (16)
522: (16)
523: (20)
524: (16)
525: (16)
526: (20)
527: (16)
528: (16)
529: (20)
530: (16)
531: (16)
532: (20)
533: (16)
534: (16)
535: (20)
536: (16)
537: (16)
538: (20)
539: (16)
540: (16)
541: (20)
542: (16)
543: (16)
544: (20)
545: (16)
546: (16)
547: (20)
548: (16)
549: (16)
550: (20)
551: (16)
552: (16)
553: (20)
554: (16)
555: (16)
556: (20)
557: (16)
558: (16)
559: (20)
560: (16)
561: (16)
562: (20)
563: (16)
564: (16)
565: (20)
566: (16)
567: (16)
568: (20)
569: (16)
570: (16)
571: (20)
572: (16)
573: (16)
574: (20)
575: (16)
576: (16)
577: (20)
578: (16)
579: (16)
580: (20)
581: (16)
582: (16)
583: (20)
584: (16)
585: (16)
586: (20)
587: (16)
588: (16)
589: (20)
590: (16)
591: (16)
592: (20)
593: (16)
594: (16)
595: (20)
596: (16)
597: (16)
598: (20)
599: (16)
600: (16)
601: (20)
602: (16)
603: (16)
604: (20)
605: (16)
606: (16)
607: (20)
608: (16)
609: (16)
610: (20)
611: (16)
612: (16)
613: (20)
614: (16)
615: (16)
616: (20)
617: (16)
618: (16)
619: (20)
620: (16)
621: (16)
622: (20)
623: (16)
624: (16)
625: (20)
626: (16)
627: (16)
628: (20)
629: (16)
630: (16)
631: (20)
632: (16)
633: (16)
634: (20)
635: (16)
636: (16)
637: (20)
638: (16)
639: (16)
640: (20)
641: (16)
642: (16)
643: (20)
644: (16)
645: (16)
646: (20)
647: (16)
648: (16)
649: (20)
650: (16)
651: (16)
652: (20)
653: (16)
654: (16)
655: (20)
656: (16)
657: (16)
658: (20)
659: (16)
660: (16)
661: (20)
662: (16)
663: (16)
664: (20)
665: (16)
666: (16)
667: (20)
668: (16)
669: (16)
670: (20)
671: (16)
672: (16)
673: (20)
674: (16)
675: (16)
676: (20)
677: (16)
678: (16)
679: (20)
680: (16)
681: (16)
682: (20)
683: (16)
684: (16)
685: (20)
686: (16)
687: (16)
688: (20)
689: (16)
690: (16)
691: (20)
692: (16)
693: (16)
694: (20)
695: (16)
696: (16)
697: (20)
698: (16)
699: (16)
700: (20)
701: (16)
702: (16)
703: (20)
704: (16)
705: (16)
706: (20)
707: (16)
708: (16)
709: (20)
710: (16)
711: (16)
712: (20)
713: (16)
714: (16)
715: (20)
716: (16)
717: (16)
718: (20)
719: (16)
720: (16)
721: (20)
722: (16)
723: (16)
724: (20)
725: (16)
726: (16)
727: (20)
728: (16)
729: (16)
730: (20)
731: (16)
732: (16)
733: (20)
734: (16)
735: (16)
736: (20)
737: (16)
738: (16)
739: (20)
740: (16)
741: (16)
742: (20)
743: (16)
744: (16)
745: (20)
746: (16)
747: (16)
748: (20)
749: (16)
750: (16)
751: (20)
752: (16)
753: (16)
754: (20)
755: (16)
756: (16)
757: (20)
758: (16)
759: (16)
760: (20)
761: (16)
762: (16)
763: (20)
764: (16)
765: (16)
766: (20)
767: (16)
768: (16)
769: (20)
770: (16)
771: (16)
772: (20)
773: (16)
774: (16)
775: (20)
776: (16)
777: (16)
778: (20)
779: (16)
780: (16)
781: (20)
782: (16)
783: (16)
784: (20)
785: (16)
786: (16)
787: (20)
788: (16)
789: (16)
790: (20)
791: (16)
792: (16)
793: (20)
794: (16)
795: (16)
796: (20)
797: (16)
798: (16)
799: (20)
800: (16)
801: (16)
802: (20)
803: (16)
804: (16)
805: (20)
806: (16)
807: (16)
808: (20)
809: (16)
8010: (16)
8011: (20)
8012: (16)
8013: (16)
8014: (20)
8015: (16)
8016: (16)
8017: (20)
8018: (16)
8019: (16)
8020: (20)
8021: (16)
8022: (16)
8023: (20)
8024: (16)
8025: (16)
8026: (20)
8027: (16)
8028: (16)
8029: (20)
8030: (16)
8031: (16)
8032: (20)
8033: (16)
8034: (16)
8035: (20)
8036: (16)
8037: (16)
8038: (20)
8039: (16)
8040: (16)
8041: (20)
8042: (16)
8043: (16)
8044: (20)
8045: (16)
8046: (16)
8047: (20)
8048: (16)
8049: (16)
8050: (20)
8051: (16)
8052: (16)
8053: (20)
8054: (16)
8055: (16)
8056: (20)
8057: (16)
8058: (16)
8059: (20)
8060: (16)
8061: (16)
8062: (20)
8063: (16)
8064: (16)
8065: (20)
8066: (16)
8067: (16)
8068: (20)
8069: (16)
8070: (16)
8071: (20)
8072: (16)
8073: (16)
8074: (20)
8075: (16)
8076: (16)
8077: (20)
8078: (16)
8079: (16)
8080: (20)
8081: (16)
8082: (16)
8083: (20)
8084: (16)
8085: (16)
8086: (20)
8087: (16)
8088: (16)
8089: (20)
8090: (16)
8091: (16)
8092: (20)
8093: (16)
8094: (16)
8095: (20)
8096: (16)
8097: (16)
8098: (20)
8099: (16)
80100: (16)
80101: (20)
80102: (16)
80103: (16)
80104: (20)
80105: (16)
80106: (16)
80107: (20)
80108: (16)
80109: (16)
80110: (20)
80111: (16)
80112: (16)
80113: (20)
80114: (16)
80115: (16)
80116: (20)
80117: (16)
80118: (16)
80119: (20)
80120: (16)
80121: (16)
80122: (20)
80123: (16)
80124: (16)
80125: (20)
80126: (16)
80127: (16)
80128: (20)
80129: (16)
80130: (16)
80131: (20)
80132: (16)
80133: (16)
80134: (20)
80135: (16)
80136: (16)
80137: (20)
80138: (16)
80139: (16)
80140: (20)
80141: (16)
80142: (16)
80143: (20)
80144: (16)
80145: (16)
80146: (20)
80147: (16)
80148: (16)
80149: (20)
80150: (16)
80151: (16)
80152: (20)
80153: (16)
80154: (16)
80155: (20)
80156: (16)
80157: (16)
80158: (20)
80159: (16)
80160: (16)
80161: (20)
80162: (16)
80163: (16)
80164: (20)
80165: (16)
80166: (16)
80167: (20)
80168: (16)
80169: (16)
80170: (20)
80171: (16)
80172: (16)
80173: (20)
80174: (16)
80175: (16)
80176: (20)
80177: (16)
80178: (16)
80179: (20)
80180: (16)
80181: (16)
80182: (20)
80183: (16)
80184: (16)
80185: (20)
80186: (16)
80187: (16)
80188: (20)
80189: (16)
80190: (16)
80191: (20)
80192: (16)
80193: (16)
80194: (20)
80195: (16)
80196: (16)
80197: (20)
80198: (16)
80199: (16)
80200: (20)
80201: (16)
80202: (16)
80203: (20)
80204: (16)
80205: (16)
80206: (20)
80207: (16)
80208: (16)
80209: (20)
80210: (16)
80211: (16)
80212: (20)
80213: (16)
80214: (16)
80215: (20)
80216: (16)
80217: (16)
80218: (20)
80219: (16)
80220: (16)
80221: (20)
80222: (16)
80223: (16)
80224: (20)
80225: (16)
80226: (16)
80227: (20)
80228: (16)
80229: (16)
80230: (20)
80231: (16)
80232: (16)
80233: (20)
80234: (16)
80235: (16)
80236: (20)
80237: (16)
80238: (16)
80239: (20)
80240: (16)
80241: (16)
80242: (20)
80243: (16)
80244: (16)
80245: (20)
80246: (16)
80247: (16)
80248: (20)
80249: (16)
80250: (16)
80251: (20)
80252: (16)
80253: (16)
80254: (20)
80255: (16)
80256: (16)
80257: (20)
80258: (16)
80259: (16)
80260: (20)
80261: (16)
80262: (16)
80263: (20)
80264: (16)
80265: (16)
80266: (20)
80267: (16)
80268: (16)
80269: (20)
80270: (16)
80271: (16)
80272: (20)
80273: (16)
80274: (16)
80275: (20)
80276: (16)
80277: (16)
80278: (20)
80279: (16)
80280: (16)
80281: (20)
80282: (16)
80283: (16)
80284: (20)
80285: (16)
80286: (16)
80287: (20)
80288: (16)
80289: (16)
80290: (20)
80291: (16)
80292: (16)
80293: (20)
80294: (16)
80295: (16)
80296: (20)
80297: (16)
80298: (16)
80299: (20)
80300: (16)
80301: (16)
80302: (20)
80303: (16)
80304: (16)
80305: (20)
80306: (16)
80307: (16)
80308: (20)
80309: (16)
80310: (16)
80311: (20)
80312: (16)
80313: (16)
80314: (20)
80315: (16)
80316: (16)
80317: (20)
80318: (16)
80319: (16)
80320: (20)
80321: (16)
80322: (16)
80323: (20)
80324: (16)
80325: (16)
80326: (20)
80327: (16)
80328: (16)
80329: (20)
80330: (16)
80331: (16)
80332: (20)
80333: (16)
80334: (16)
80335: (20)
80336: (16)
80337: (16)
80338: (20)
80339: (16)
80340: (16)
80341: (20)
80342: (16)
80343: (16)
80344: (20)
80345: (16)
80346: (16)
80347: (20)
80348: (16)
80349: (16)
80350: (20)
80351: (16)
80352: (16)
80353: (20)
80354: (16)
80355: (16)
80356: (20)
80357: (16)
80358: (16)
80359: (20)
80360: (16)
80361: (16)
80362: (20)
80363: (16)
80364: (16)
80365: (20)
80366: (16)
80367: (16)
80368: (20)
80369: (16)
80370: (16)
80371: (20)
80372: (16)
80373: (16)
80374: (20)
80375: (16)
80376: (16)
80377: (20)
80378: (16)
80379: (16)
80380: (20)
80381: (16)
80382: (16)
80383: (20)
80384: (16)
80385: (16)
80386: (20)
80387: (16)
80388: (16)
80389: (20)
80390: (16)
80391: (16)
80392: (20)
80393: (16)
80394: (16)
80395: (20)
80396: (16)
80397: (16)
80398: (20)
80399: (16)
80400: (16)
80401: (20)
80402: (16)
80403: (16)
80404: (20)
80405: (16)
80406: (16)
80407: (20)
80408: (16)
80409: (16)
80410: (20)
80411: (16)
80412: (16)
80413: (20)
80414: (16)
80415: (16)
80416: (20)
80417: (16)
80418: (16)
80419: (20)
80420: (16)
80421: (16)
80422: (20)
80423: (16)
80424: (16)
80425: (20)
80426: (16)
80427: (16)
80428: (20)
80429: (16)
80430: (16)
80431: (20)
80432: (16)
80433: (16)
80434: (20)
80435: (16)
80436: (16)
80437: (20)
80438: (16)
80439: (16)
80440: (20)
80441: (16)
80442: (16)
80443: (20)
80444: (16)
80445: (16)
80446: (20)
80447: (16)
80448: (16)
80449: (20)
80450: (16)
80451: (16)
80452: (20)
80453: (16)
80454: (16)
80455: (20)
80456: (16)
80457: (16)
80458: (20)
80459: (16)
80460: (16)
80461: (20)
80462: (16)
80463: (16)
80464: (20)
80465: (16)
80466: (16)
80467: (20)
80468: (16)
80469: (16)
80470: (20)
80471: (16)
80472: (16)
80473: (20)
80474: (16)
80475: (16)
80476: (20)
80477: (16)
80478: (16)
80479: (20)
80480: (16)
80481: (16)
80482: (20)
80483: (16)
80484: (16)
80485: (20)
80486: (16)
80487: (16)
80488: (20)
80489: (16)
80490: (16)
80491: (20)
80492: (16)
80493: (16)
80494: (20)
80495: (16)
80496: (16)
80497: (20)
80498: (16)
80499: (16)
80500: (20)
80501: (16)
80502: (16)
80503: (20)
80504: (16)
80505: (16)
80506: (20)
80507: (16)
80508: (16)
80509: (20)
80510: (16)
80511: (16)
80512: (20)
80513: (16)
80514: (16)
80515: (20)
80516: (16)
80517:
```

```

169: (8)             if x_range is None:
170: (12)             x_range = np.array([-config["frame_x_radius"],
config["frame_x_radius"]])
171: (8)             self.x_range = x_range
172: (8)             self.parametric_function = lambda t: np.array([t, function(t), 0])
173: (8)             self.function = function
174: (8)             super().__init__(self.parametric_function, self.x_range, color=color,
**kwargs)
175: (4)             def get_function(self):
176: (8)                 return self.function
177: (4)             def get_point_from_function(self, x):
178: (8)                 return self.parametric_function(x)
179: (0)             class ImplicitFunction(VMobject, metaclass=ConvertToOpenGL):
180: (4)                 def __init__(
181: (8)                     self,
182: (8)                     func: Callable[[float, float], float],
183: (8)                     x_range: Sequence[float] | None = None,
184: (8)                     y_range: Sequence[float] | None = None,
185: (8)                     min_depth: int = 5,
186: (8)                     max_quads: int = 1500,
187: (8)                     use_smoothing: bool = True,
188: (8)                     **kwargs,
189: (4)                 ):
190: (8)                     """An implicit function.
191: (8)                     Parameters
192: (8)                     -----
193: (8)                     func
194: (12)                         The implicit function in the form ``f(x, y) = 0``.
195: (8)                     x_range
196: (12)                         The x min and max of the function.
197: (8)                     y_range
198: (12)                         The y min and max of the function.
199: (8)                     min_depth
200: (12)                         The minimum depth of the function to calculate.
201: (8)                     max_quads
202: (12)                         The maximum number of quads to use.
203: (8)                     use_smoothing
204: (12)                         Whether or not to smoothen the curves.
205: (8)                     kwargs
206: (12)                         Additional parameters to pass into :class:`VMobject`  

207: (8) .. note::
208: (12)             A small ``min_depth`` :math:`d` means that some small details
might
209: (12)                 be ignored if they don't cross an edge of one of the
210: (12)                 :math:`4^d` uniform quads.
211: (12)                 The value of ``max_quads`` strongly corresponds to the
212: (12)                 quality of the curve, but a higher number of quads
213: (12)                 may take longer to render.
214: (8)             Examples
215: (8)             -----
216: (8)             .. manim:: ImplicitFunctionExample
217: (12)                 :save_last_frame:
218: (12)                 class ImplicitFunctionExample(Scene):
219: (16)                     def construct(self):
220: (20)                         graph = ImplicitFunction(
221: (24)                             lambda x, y: x * y ** 2 - x ** 2 * y - 2,
222: (24)                             color=YELLOW
223: (20)                         )
224: (20)                         self.add(NumberPlane(), graph)
225: (8)             """
226: (8)             self.function = func
227: (8)             self.min_depth = min_depth
228: (8)             self.max_quads = max_quads
229: (8)             self.use_smoothing = use_smoothing
230: (8)             self.x_range = x_range or [
231: (12)                 -config.frame_width / 2,
232: (12)                 config.frame_width / 2,
233: (8)             ]
234: (8)             self.y_range = y_range or [

```

```

235: (12)           -config.frame_height / 2,
236: (12)           config.frame_height / 2,
237: (8)
238: (8)
239: (4)
240: (8)
241: (12)
242: (12)
243: (8)
244: (8)
245: (12)
246: (12)
247: (12)
248: (12)
249: (12)
250: (8)
251: (8)
252: (12)
253: (8)
254: (8)
255: (12)
256: (12)
257: (8)
258: (12)
259: (8)
260: (4)

def generate_points(self):
    p_min, p_max = (
        np.array([self.x_range[0], self.y_range[0]]),
        np.array([self.x_range[1], self.y_range[1]]),
    )
    curves = plot_isoline(
        fn=lambda u: self.function(u[0], u[1]),
        pmin=p_min,
        pmax=p_max,
        min_depth=self.min_depth,
        max_quads=self.max_quads,
    ) # returns a list of lists of 2D points
    curves = [
        np.pad(curve, [(0, 0), (0, 1)]) for curve in curves if curve != []
    ] # add z coord as 0
    for curve in curves:
        self.start_new_path(curve[0])
        self.add_points_as_corners(curve[1:])
    if self.use_smoothing:
        self.make_smooth()
    return self
init_points = generate_points
-----
```

## File 66 - dot\_cloud.py:

```

1: (0)         from __future__ import annotations
2: (0)         __all__ = ["TrueDot", "DotCloud"]
3: (0)         import numpy as np
4: (0)         from manim.constants import ORIGIN, RIGHT, UP
5: (0)         from manim.mobject.opengl.opengl_point_cloud_mobject import OpenGLMobject
6: (0)         from manim.utils.color import YELLOW
7: (0)         class DotCloud(OpenGLMobject):
8: (4)             def __init__(
9: (8)                 self, color=YELLOW, stroke_width=2.0, radius=2.0, density=10, **kwargs
10: (4)             ):
11: (8)                 self.radius = radius
12: (8)                 self.epsilon = 1.0 / density
13: (8)                 super().__init__(
14: (12)                     stroke_width=stroke_width, density=density, color=color, **kwargs
15: (8)                 )
16: (4)             def init_points(self):
17: (8)                 self.points = np.array(
18: (12)                     [
19: (16)                         r * (np.cos(theta) * RIGHT + np.sin(theta) * UP)
20: (16)                         for r in np.arange(self.epsilon, self.radius, self.epsilon)
21: (16)                         for theta in np.linspace(
22: (20)                             0,
23: (20)                             2 * np.pi,
24: (20)                             num=int(2 * np.pi * (r + self.epsilon) / self.epsilon),
25: (16)                         )
26: (12)                     ],
27: (12)                     dtype=np.float32,
28: (8)                 )
29: (4)             def make_3d(self, gloss=0.5, shadow=0.2):
30: (8)                 self.set_gloss(gloss)
31: (8)                 self.set_shadow(shadow)
32: (8)                 self.apply_depth_test()
33: (8)                 return self
34: (0)             class TrueDot(DotCloud):
35: (4)                 def __init__(self, center=ORIGIN, stroke_width=2.0, **kwargs):
36: (8)                     self.radius = stroke_width
37: (8)                     super().__init__(points=[center], stroke_width=stroke_width, **kwargs)
```

## File 67 - boolean\_ops.py:

```

1: (0)         """Boolean operations for two-dimensional mobjects."""
2: (0)         from __future__ import annotations
3: (0)         from typing import TYPE_CHECKING
4: (0)         import numpy as np
5: (0)         from pathops import Path as SkiaPath
6: (0)         from pathops import PathVerb, difference, intersection, union, xor
7: (0)         from manim import config
8: (0)         from manim.mobject.opengl_compatibility import ConvertToOpenGL
9: (0)         from manim.mobject.types.vectorized_mobject import VMobject
10: (0)        if TYPE_CHECKING:
11: (4)            from manim.typing import Point2D_Array, Point3D_Array
12: (0)        from ...constants import RendererType
13: (0)        __all__ = ["Union", "Intersection", "Difference", "Exclusion"]
14: (0)        class _BooleanOps(VMobject, metaclass=ConvertToOpenGL):
15: (4)            """This class contains some helper functions which
16: (4)            helps to convert to and from skia objects and manim
17: (4)            objects (:class:`~.VMobject`).
18: (4)            """
19: (4)            def _convert_2d_to_3d_array(
20: (8)                self,
21: (8)                points: Point2D_Array,
22: (8)                z_dim: float = 0.0,
23: (4)            ) -> Point3D_Array:
24: (8)                """Converts an iterable with coordinates in 2D to 3D by adding
25: (8)                :attr:`z_dim` as the Z coordinate.
26: (8)                Parameters
27: (8)                -----
28: (8)                points:
29: (12)                    An iterable of points.
30: (8)                z_dim:
31: (12)                    Default value for the Z coordinate.
32: (8)                Returns
33: (8)                -----
34: (8)                Point3D_Array
35: (12)                    A list of the points converted to 3D.
36: (8)                Example
37: (8)                -----
38: (8)                >>> a = _BooleanOps()
39: (8)                >>> p = [(1, 2), (3, 4)]
40: (8)                >>> a._convert_2d_to_3d_array(p)
41: (8)                [array([1., 2., 0.]), array([3., 4., 0.])]
42: (8)                """
43: (8)                points = list(points)
44: (8)                for i, point in enumerate(points):
45: (12)                    if len(point) == 2:
46: (16)                        points[i] = np.array(list(point) + [z_dim])
47: (8)                return points
48: (4)            def _convert_vmobject_to_skia_path(self, vmobject: VMobject) -> SkiaPath:
49: (8)                """Converts a :class:`~.VMobject` to SkiaPath. This method only works
for
50: (8)                    cairo renderer because it treats the points as Cubic beizer curves.
51: (8)                    Parameters
52: (8)                    -----
53: (8)                    vmobject:
54: (12)                        The :class:`~.VMobject` to convert from.
55: (8)                    Returns
56: (8)                    -----
57: (8)                    SkiaPath
58: (12)                        The converted path.
59: (8)                    """
60: (8)                    path = SkiaPath()
61: (8)                    if not np.all(np.isfinite(vmobject.points)):
62: (12)                        points = np.zeros((1, 3)) # point invalid?
63: (8)                    else:
64: (12)                        points = vmobject.points

```

```

65: (8)                     if len(points) == 0: # what? No points so return empty path
66: (12)                   return path
67: (8)                     if config.renderer == RendererType.OPENGL:
68: (12)                       subpaths = vmobject.get_subpaths_from_points(points)
69: (12)                       for subpath in subpaths:
70: (16)                           quads = vmobject.get_bezier_tuples_from_points(subpath)
71: (16)                           start = subpath[0]
72: (16)                           path.moveTo(*start[:2])
73: (16)                           for p0, p1, p2 in quads:
74: (20)                               path.quadTo(*p1[:2], *p2[:2])
75: (16)                               if vmobject.consider_points_equals(subpath[0], subpath[-1]):
76: (20)                                   path.close()
77: (8)                     elif config.renderer == RendererType.CAIRO:
78: (12)                       subpaths = vmobject.gen_subpaths_from_points_2d(points)
79: (12)                       for subpath in subpaths:
80: (16)                           quads = vmobject.gen_cubic_bezier_tuples_from_points(subpath)
81: (16)                           start = subpath[0]
82: (16)                           path.moveTo(*start[:2])
83: (16)                           for p0, p1, p2, p3 in quads:
84: (20)                               path.cubicTo(*p1[:2], *p2[:2], *p3[:2])
85: (16)                               if vmobject.consider_points_equals_2d(subpath[0],
subpath[-1]): 
86: (20)                                   path.close()
87: (8)                     return path
88: (4) def _convert_skia_path_to_vmobject(self, path: SkiaPath) -> VMobject:
89: (8)     """Converts SkiaPath back to VMobject.
90: (8)     Parameters
91: (8)     -----
92: (8)     path:
93: (12)         The SkiaPath to convert.
94: (8)     Returns
95: (8)     -----
96: (8)     VMobject:
97: (12)         The converted VMobject.
98: (8)     """
99: (8)     vmobject = self
100: (8)    current_path_start = np.array([0, 0, 0])
101: (8)    for path_verb, points in path:
102: (12)        if path_verb == PathVerb.MOVE:
103: (16)            parts = self._convert_2d_to_3d_array(points)
104: (16)            for part in parts:
105: (20)                current_path_start = part
106: (20)                vmobject.start_new_path(part)
107: (12)            elif path_verb == PathVerb.CUBIC:
108: (16)                n1, n2, n3 = self._convert_2d_to_3d_array(points)
109: (16)                vmobject.add_cubic_bezier_curve_to(n1, n2, n3)
110: (12)            elif path_verb == PathVerb.LINE:
111: (16)                parts = self._convert_2d_to_3d_array(points)
112: (16)                vmobject.add_line_to(parts[0])
113: (12)            elif path_verb == PathVerb.CLOSE:
114: (16)                vmobject.add_line_to(current_path_start)
115: (12)            elif path_verb == PathVerb.QUAD:
116: (16)                n1, n2 = self._convert_2d_to_3d_array(points)
117: (16)                vmobject.add_quadratic_bezier_curve_to(n1, n2)
118: (12)            else:
119: (16)                raise Exception("Unsupported: %s" % path_verb)
120: (8)        return vmobject
121: (0) class Union(_BooleanOps):
122: (4)     """Union of two or more :class:`~.VMobject` s. This returns the common
region of
123: (4)     the :class:`~VMobject` s.
124: (4)     Parameters
125: (4)     -----
126: (4)     vmobjects
127: (8)         The :class:`~.VMobject` s to find the union of.
128: (4)     Raises
129: (4)     -----
130: (4)     ValueError
131: (8)         If less than 2 :class:`~.VMobject` s are passed.

```

```

132: (4)             Example
133: (4)             -----
134: (4)             .. manim:: UnionExample
135: (8)             :save_last_frame:
136: (8)             class UnionExample(Scene):
137: (12)            def construct(self):
138: (16)            sq = Square(color=RED, fill_opacity=1)
139: (16)            sq.move_to([-2, 0, 0])
140: (16)            cr = Circle(color=BLUE, fill_opacity=1)
141: (16)            cr.move_to([-1.3, 0.7, 0])
142: (16)            un = Union(sq, cr, color=GREEN, fill_opacity=1)
143: (16)            un.move_to([1.5, 0.3, 0])
144: (16)            self.add(sq, cr, un)
145: (4)             """
146: (4)             def __init__(self, *vmobjects: VMobject, **kwargs) -> None:
147: (8)             if len(vmobjects) < 2:
148: (12)             raise ValueError("At least 2 mobjects needed for Union.")
149: (8)             super().__init__(**kwargs)
150: (8)             paths = []
151: (8)             for vmobject in vmobjects:
152: (12)             paths.append(self._convert_vmobject_to_skia_path(vmobject))
153: (8)             outpen = SkiaPath()
154: (8)             union(paths, outpen.getPen())
155: (8)             self._convert_skia_path_to_vmobject(outpen)
156: (0)             class Difference(_BooleanOps):
157: (4)             """Subtracts one :class:`~.VMobject` from another one.
158: (4)             Parameters
159: (4)             -----
160: (4)             subject
161: (8)             The 1st :class:`~.VMobject`.
162: (4)             clip
163: (8)             The 2nd :class:`~.VMobject`
164: (4)             Example
165: (4)             -----
166: (4)             .. manim:: DifferenceExample
167: (8)             :save_last_frame:
168: (8)             class DifferenceExample(Scene):
169: (12)            def construct(self):
170: (16)            sq = Square(color=RED, fill_opacity=1)
171: (16)            sq.move_to([-2, 0, 0])
172: (16)            cr = Circle(color=BLUE, fill_opacity=1)
173: (16)            cr.move_to([-1.3, 0.7, 0])
174: (16)            un = Difference(sq, cr, color=GREEN, fill_opacity=1)
175: (16)            un.move_to([1.5, 0, 0])
176: (16)            self.add(sq, cr, un)
177: (4)             """
178: (4)             def __init__(self, subject: VMobject, clip: VMobject, **kwargs) -> None:
179: (8)             super().__init__(**kwargs)
180: (8)             outpen = SkiaPath()
181: (8)             difference(
182: (12)             [self._convert_vmobject_to_skia_path(subject)],
183: (12)             [self._convert_vmobject_to_skia_path(clip)],
184: (12)             outpen.getPen(),
185: (8)             )
186: (8)             self._convert_skia_path_to_vmobject(outpen)
187: (0)             class Intersection(_BooleanOps):
188: (4)             """Find the intersection of two :class:`~.VMobject` s.
189: (4)             This keeps the parts covered by both :class:`~.VMobject` s.
190: (4)             Parameters
191: (4)             -----
192: (4)             vmobjects
193: (8)             The :class:`~.VMobject` to find the intersection.
194: (4)             Raises
195: (4)             -----
196: (4)             ValueError
197: (8)             If less than the 2 :class:`~.VMobject` are passed.
198: (4)             Example
199: (4)             -----
200: (4)             .. manim:: IntersectionExample

```

```

201: (8)           :save_last_frame:
202: (8)           class IntersectionExample(Scene):
203: (12)          def construct(self):
204: (16)          sq = Square(color=RED, fill_opacity=1)
205: (16)          sq.move_to([-2, 0, 0])
206: (16)          cr = Circle(color=BLUE, fill_opacity=1)
207: (16)          cr.move_to([-1.3, 0.7, 0])
208: (16)          un = Intersection(sq, cr, color=GREEN, fill_opacity=1)
209: (16)          un.move_to([1.5, 0, 0])
210: (16)          self.add(sq, cr, un)
211: (4)          """
212: (4)          def __init__(self, *vmobjects: VMobject, **kwargs) -> None:
213: (8)          if len(vmobjects) < 2:
214: (12)          raise ValueError("At least 2 mobjects needed for Intersection.")
215: (8)          super().__init__(**kwargs)
216: (8)          outpen = SkiaPath()
217: (8)          intersection(
218: (12)          [self._convert_vmobject_to_skia_path(vmobjects[0])],
219: (12)          [self._convert_vmobject_to_skia_path(vmobjects[1])],
220: (12)          outpen.getPen(),
221: (8)          )
222: (8)          new_outpen = outpen
223: (8)          for _i in range(2, len(vmobjects)):
224: (12)          new_outpen = SkiaPath()
225: (12)          intersection(
226: (16)          [outpen],
227: (16)          [self._convert_vmobject_to_skia_path(vmobjects[_i])],
228: (16)          new_outpen.getPen(),
229: (12)          )
230: (12)          outpen = new_outpen
231: (8)          self._convert_skia_path_to_vmobject(outpen)
232: (0)          class Exclusion(_BooleanOps):
233: (4)          """Find the XOR between two :class:`~.VMobject` .
234: (4)          This creates a new :class:`~.VMobject` consisting of the region
235: (4)          covered by exactly one of them.
236: (4)          Parameters
237: (4)          -----
238: (4)          subject
239: (8)          The 1st :class:`~.VMobject` .
240: (4)          clip
241: (8)          The 2nd :class:`~.VMobject` 
242: (4)          Example
243: (4)          -----
244: (4)          .. manim:: IntersectionExample
245: (8)          :save_last_frame:
246: (8)          class IntersectionExample(Scene):
247: (12)          def construct(self):
248: (16)          sq = Square(color=RED, fill_opacity=1)
249: (16)          sq.move_to([-2, 0, 0])
250: (16)          cr = Circle(color=BLUE, fill_opacity=1)
251: (16)          cr.move_to([-1.3, 0.7, 0])
252: (16)          un = Exclusion(sq, cr, color=GREEN, fill_opacity=1)
253: (16)          un.move_to([1.5, 0.4, 0])
254: (16)          self.add(sq, cr, un)
255: (4)          """
256: (4)          def __init__(self, subject: VMobject, clip: VMobject, **kwargs) -> None:
257: (8)          super().__init__(**kwargs)
258: (8)          outpen = SkiaPath()
259: (8)          xor(
260: (12)          [self._convert_vmobject_to_skia_path(subject)],
261: (12)          [self._convert_vmobject_to_skia_path(clip)],
262: (12)          outpen.getPen(),
263: (8)          )
264: (8)          self._convert_skia_path_to_vmobject(outpen)
-----
```

File 68 - number\_line.py:

```

1: (0)         """Object representing a number line."""
2: (0)         from __future__ import annotations
3: (0)         from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLMobject
4: (0)         __all__ = ["NumberLine", "UnitInterval"]
5: (0)         from typing import TYPE_CHECKING, Callable, Iterable, Sequence
6: (0)         if TYPE_CHECKING:
7: (4)             from manim.mobject.geometry.tips import ArrowTip
8: (0)         import numpy as np
9: (0)         from manim import config
10: (0)        from manim.constants import *
11: (0)        from manim.mobject.geometry.line import Line
12: (0)        from manim.mobject.graphing.scale import LinearBase, _ScaleBase
13: (0)        from manim.mobject.text.numbers import DecimalNumber
14: (0)        from manim.mobject.text.tex_mobject import MathTex, Tex
15: (0)        from manim.mobject.types.vectorized_mobject import VGroup, VMobject
16: (0)        from manim.utils.bezier import interpolate
17: (0)        from manim.utils.config_ops import merge_dicts_recursively
18: (0)        from manim.utils.space_ops import normalize
19: (0)        class NumberLine(Line):
20: (4)            """Creates a number line with tick marks.
21: (4)            Parameters
22: (4)            -----
23: (4)            x_range
24: (8)                The ``[x_min, x_max, x_step]`` values to create the line.
25: (4)            length
26: (8)                The length of the number line.
27: (4)            unit_size
28: (8)                The distance between each tick of the line. Overwritten by
:attr:`length`, if specified.
29: (4)            include_ticks
30: (8)                Whether to include ticks on the number line.
31: (4)            tick_size
32: (8)                The length of each tick mark.
33: (4)            numbers_with_elongated_ticks
34: (8)                An iterable of specific values with elongated ticks.
35: (4)            longer_tick_multiple
36: (8)                Influences how many times larger elongated ticks are than regular
ticks (2 = 2x).
37: (4)            rotation
38: (8)                The angle (in radians) at which the line is rotated.
39: (4)            stroke_width
40: (8)                The thickness of the line.
41: (4)            include_tip
42: (8)                Whether to add a tip to the end of the line.
43: (4)            tip_width
44: (8)                The width of the tip.
45: (4)            tip_height
46: (8)                The height of the tip.
47: (4)            tip_shape
48: (8)                The mobject class used to construct the tip, or ``None`` (the
49: (8)                default) for the default arrow tip. Passed classes have to inherit
50: (8)                from :class:`.ArrowTip`.
51: (4)            include_numbers
52: (8)                Whether to add numbers to the tick marks. The number of decimal places
is determined
53: (8)                by the step size, this default can be overridden by
``decimal_number_config``.
54: (4)            scaling
55: (8)                The way the ``x_range`` is value is scaled, i.e. :class:`~.LogBase`  

for a logarithmic numberline. Defaults to :class:`~.LinearBase`.
56: (4)            font_size
57: (8)                The size of the label mobjects. Defaults to 36.
58: (4)            label_direction
59: (8)                The specific position to which label mobjects are added on the line.
60: (4)            label_constructor
61: (8)                Determines the mobject class that will be used to construct the labels
of the number line.
62: (4)            line_to_number_buff
63: (8)                The distance between the line and the label mobject.

```

```

64: (4)           decimal_number_config
65: (8)             Arguments that can be passed to :class:`~.numbers.DecimalNumber` to
influence number mobjects.
66: (4)           numbers_to_exclude
67: (8)             An explicit iterable of numbers to not be added to the number line.
68: (4)           numbers_to_include
69: (8)             An explicit iterable of numbers to add to the number line
70: (4)           kwargs
71: (8)             Additional arguments to be passed to :class:`~.Line`.
72: (4)
73: (8)             .. note::
generated
74: (8)               Number ranges that include both negative and positive values will be
75: (8)               from the 0 point, and may not include a tick at the min / max
values as the tick locations are dependent on the step size.
76: (4)             Examples
77: (4)             -----
78: (4)             .. manim:: NumberLineExample
79: (8)               :save_last_frame:
80: (8)               class NumberLineExample(Scene):
81: (12)                 def construct(self):
82: (16)                   10 = NumberLine(
83: (20)                     x_range=[-10, 10, 2],
84: (20)                     length=10,
85: (20)                     color=BLUE,
86: (20)                     include_numbers=True,
87: (20)                     label_direction=UP,
88: (16)                   )
89: (16)                   11 = NumberLine(
90: (20)                     x_range=[-10, 10, 2],
91: (20)                     unit_size=0.5,
92: (20)                     numbers_with_elongated_ticks=[-2, 4],
93: (20)                     include_numbers=True,
94: (20)                     font_size=24,
95: (16)                   )
96: (16)                   num6 = 11.numbers[8]
97: (16)                   num6.set_color(RED)
98: (16)                   12 = NumberLine(
99: (20)                     x_range=[-2.5, 2.5 + 0.5, 0.5],
100: (20)                     length=12,
101: (20)                     decimal_number_config={"num_decimal_places": 2},
102: (20)                     include_numbers=True,
103: (16)                   )
104: (16)                   13 = NumberLine(
105: (20)                     x_range=[-5, 5 + 1, 1],
106: (20)                     length=6,
107: (20)                     include_tip=True,
108: (20)                     include_numbers=True,
109: (20)                     rotation=10 * DEGREES,
110: (16)                   )
111: (16)                   line_group = VGroup(10, 11, 12, 13).arrange(DOWN, buff=1)
112: (16)                   self.add(line_group)
113: (4)             """
114: (4)             def __init__(
115: (8)               self,
116: (8)               x_range: Sequence[float] | None = None, # must be first
117: (8)               length: float | None = None,
118: (8)               unit_size: float = 1,
119: (8)               include_ticks: bool = True,
120: (8)               tick_size: float = 0.1,
121: (8)               numbers_with_elongated_ticks: Iterable[float] | None = None,
122: (8)               longer_tick_multiple: int = 2,
123: (8)               exclude_origin_tick: bool = False,
124: (8)               rotation: float = 0,
125: (8)               stroke_width: float = 2.0,
126: (8)               include_tip: bool = False,
127: (8)               tip_width: float = DEFAULT_ARROW_TIP_LENGTH,
128: (8)               tip_height: float = DEFAULT_ARROW_TIP_LENGTH,
129: (8)               tip_shape: type[ArrowTip] | None = None,
130: (8)               include_numbers: bool = False,

```

```

131: (8)             font_size: float = 36,
132: (8)             label_direction: Sequence[float] = DOWN,
133: (8)             label_constructor: VMobject = MathTex,
134: (8)             scaling: _ScaleBase = LinearBase(),
135: (8)             line_to_number_buff: float = MED_SMALL_BUFF,
136: (8)             decimal_number_config: dict | None = None,
137: (8)             numbers_to_exclude: Iterable[float] | None = None,
138: (8)             numbers_to_include: Iterable[float] | None = None,
139: (8)             **kwargs,
140: (4):
141: (8)             if numbers_to_exclude is None:
142: (12)                 numbers_to_exclude = []
143: (8)             if numbers_with_elongated_ticks is None:
144: (12)                 numbers_with_elongated_ticks = []
145: (8)             if x_range is None:
146: (12)                 x_range = [
147: (16)                     round(-config["frame_x_radius"]),
148: (16)                     round(config["frame_x_radius"]),
149: (16)                     1,
150: (12)                 ]
151: (8)             elif len(x_range) == 2:
152: (12)                 x_range = [*x_range, 1]
153: (8)             if decimal_number_config is None:
154: (12)                 decimal_number_config = {
155: (16)                     "num_decimal_places":
self._decimal_places_from_step(x_range[2]),
156: (12)                 }
157: (8)                 self.x_range = np.array(x_range, dtype=float)
158: (8)                 self.x_min, self.x_max, self.x_step = scaling.function(self.x_range)
159: (8)                 self.length = length
160: (8)                 self.unit_size = unit_size
161: (8)                 self.include_ticks = include_ticks
162: (8)                 self.tick_size = tick_size
163: (8)                 self.numbers_with_elongated_ticks = numbers_with_elongated_ticks
164: (8)                 self.longer_tick_multiple = longer_tick_multiple
165: (8)                 self.exclude_origin_tick = exclude_origin_tick
166: (8)                 self.rotation = rotation
167: (8)                 self.include_tip = include_tip
168: (8)                 self.tip_width = tip_width
169: (8)                 self.tip_height = tip_height
170: (8)                 self.font_size = font_size
171: (8)                 self.include_numbers = include_numbers
172: (8)                 self.label_direction = label_direction
173: (8)                 self.label_constructor = label_constructor
174: (8)                 self.line_to_number_buff = line_to_number_buff
175: (8)                 self.decimal_number_config = decimal_number_config
176: (8)                 self.numbers_to_exclude = numbers_to_exclude
177: (8)                 self.numbers_to_include = numbers_to_include
178: (8)                 self.scaling = scaling
179: (8)                 super().__init__(
180: (12)                     self.x_range[0] * RIGHT,
181: (12)                     self.x_range[1] * RIGHT,
182: (12)                     stroke_width=stroke_width,
183: (12)                     **kwargs,
184: (8)                 )
185: (8)                 if self.length:
186: (12)                     self.set_length(self.length)
187: (12)                     self.unit_size = self.get_unit_size()
188: (8)                 else:
189: (12)                     self.scale(self.unit_size)
190: (8)                 self.center()
191: (8)                 if self.include_tip:
192: (12)                     self.add_tip(
193: (16)                         tip_length=self.tip_height,
194: (16)                         tip_width=self.tip_width,
195: (16)                         tip_shape=tip_shape,
196: (12)                     )
197: (12)                     self.tip.set_stroke(self.stroke_color, self.stroke_width)
198: (8)                 if self.include_ticks:

```

```

199: (12)           self.add_ticks()
200: (8)            self.rotate(self.rotation)
201: (8)            if self.include_numbers or self.numbers_to_include is not None:
202: (12)              if self.scaling.custom_labels:
203: (16)                tick_range = self.get_tick_range()
204: (16)                self.add_labels(
205: (20)                  dict(
206: (24)                    zip(
207: (28)                      tick_range,
208: (28)                        self.scaling.get_custom_labels(
209: (32)                          tick_range,
210: (32)                            unit_decimal_places=decimal_number_config[
211: (36)                              "num_decimal_places"
212: (32)                            ],
213: (28)                          ),
214: (24)                        ),
215: (20)                      ),
216: (16)                    )
217: (12)                  else:
218: (16)                    self.add_numbers(
219: (20)                      x_values=self.numbers_to_include,
220: (20)                      excluding=self.numbers_to_exclude,
221: (20)                      font_size=self.font_size,
222: (16)                    )
223: (4)          def rotate_about_zero(self, angle: float, axis: Sequence[float] = OUT,
224: (8)            **kwargs):
225: (4)              return self.rotate_about_number(0, angle, axis, **kwargs)
226: (8)          def rotate_about_number(
227: (8)            self, number: float, angle: float, axis: Sequence[float] = OUT,
228: (8)            **kwargs
229: (4)          ):
230: (8)              return self.rotate(angle, axis, about_point=self.n2p(number)),
231: (8)          def add_ticks(self):
232: (8)              """Adds ticks to the number line. Ticks can be accessed after creation
233: (8)              via ``self.ticks``."""
234: (8)              ticks = VGroup()
235: (8)              elongated_tick_size = self.tick_size * self.longer_tick_multiple
236: (12)              elongated_tick_offsets = self.numbers_with_elongated_ticks -
237: (12)              self.x_min
238: (16)              for x in self.get_tick_range():
239: (12)                  size = self.tick_size
240: (8)                  if np.any(np.isclose(x - self.x_min, elongated_tick_offsets)):
241: (8)                      size = elongated_tick_size
242: (8)                  ticks.add(self.get_tick(x, size))
243: (8)              self.add(ticks)
244: (8)              self.ticks = ticks
245: (8)          def get_tick(self, x: float, size: float | None = None) -> Line:
246: (8)              """Generates a tick and positions it along the number line.
247: (12)              Parameters
248: (8)                  -----
249: (12)                  x
250: (8)                      The position of the tick.
251: (8)                  size
252: (8)                      The factor by which the tick is scaled.
253: (12)          Returns
254: (8)                  -----
255: (8)                  :class:`~.Line`
256: (12)                  A positioned tick.
257: (8)              """
258: (8)              if size is None:
259: (12)                  size = self.tick_size
260: (8)              result = Line(size * DOWN, size * UP)
261: (8)              result.rotate(self.get_angle())
262: (8)              result.move_to(self.number_to_point(x))
263: (8)              result.match_style(self)
264: (8)              return result
265: (4)          def get_tick_marks(self) -> VGroup:
266: (8)              return self.ticks

```

```

264: (4)             def get_tick_range(self) -> np.ndarray:
265: (8)                 """Generates the range of values on which labels are plotted based on
the
266: (8)                   ``x_range`` attribute of the number line.
267: (8)                   Returns
268: (8)                   -----
269: (8)                   np.ndarray
270: (12)                     A numpy array of floats represnting values along the number line.
271: (8)
272: (8)                     """
273: (8)                     x_min, x_max, x_step = self.x_range
274: (12)                     if not self.include_tip:
275: (8)                         x_max += 1e-6
276: (12)                     if x_min < x_max < 0 or x_max > x_min > 0:
277: (8)                         tick_range = np.arange(x_min, x_max, x_step)
278: (12)                     else:
279: (8)                         start_point = 0
280: (16)                         if self.exclude_origin_tick:
281: (12)                             start_point += x_step
282: (12)                         x_min_segment = np.arange(start_point, np.abs(x_min) + 1e-6,
283: (12)                           x_max_segment = np.arange(start_point, x_max, x_step)
284: (8)                           tick_range = np.unique(np.concatenate((x_min_segment,
285: (4)                               return self.scaling.function(tick_range)
286: (8)             def number_to_point(self, number: float | np.ndarray) -> np.ndarray:
287: (8)                 """Accepts a value along the number line and returns a point with
respect to the scene.
288: (8)                 Parameters
289: (8)                 -----
290: (8)                 number
291: (12)                   The value to be transformed into a coordinate. Or a list of
values.
292: (8)                 Returns
293: (8)                 -----
294: (8)                 np.ndarray
295: (12)                   A point with respect to the scene's coordinate system. Or a list
of points.
296: (8)
297: (8)
298: (12)             Examples
299: (12)             -----
300: (12)                 >>> from manim import NumberLine
301: (12)                 >>> number_line = NumberLine()
302: (12)                 >>> number_line.number_to_point(0)
303: (12)                   array([0., 0., 0.])
304: (12)                 >>> number_line.number_to_point(1)
305: (12)                   array([1., 0., 0.])
306: (19)                 >>> number_line.number_to_point([1,2,3])
307: (19)                   array([[1., 0., 0.],
308: (8)                           [2., 0., 0.],
309: (8)                           [3., 0., 0.]])
310: (8)
311: (8)
312: (8)
313: (8)             number = np.asarray(number)
314: (8)             scalar = number.ndim == 0
315: (8)             number = self.scaling.inverse_function(number)
316: (4)             alphas = (number - self.x_range[0]) / (self.x_range[1] -
317: (8)                           self.x_range[0])
318: (8)
319: (8)
320: (8)
321: (8)
322: (12)             alphas = float(alphas) if scalar else np.vstack(alphas)
323: (8)             val = interpolate(self.get_start(), self.get_end(), alphas)
324: (8)             return val
325: (8)
326: (4)             def point_to_number(self, point: Sequence[float]) -> float:
327: (8)                 """Accepts a point with respect to the scene and returns
328: (8)                   a float along the number line.
329: (8)                 Parameters
330: (8)                 -----
331: (8)                 point
332: (12)                   A sequence of values consisting of ``(x_coord, y_coord,
z_coord)``.
333: (8)
334: (8)
335: (8)
336: (8)
337: (8)
338: (8)
339: (8)
340: (8)
341: (8)
342: (8)
343: (8)
344: (8)
345: (8)
346: (8)
347: (8)
348: (8)
349: (8)
350: (8)
351: (8)
352: (8)
353: (8)
354: (8)
355: (8)
356: (8)
357: (8)
358: (8)
359: (8)
360: (8)
361: (8)
362: (8)
363: (8)
364: (8)
365: (8)
366: (8)
367: (8)
368: (8)
369: (8)
370: (8)
371: (8)
372: (8)
373: (8)
374: (8)
375: (8)
376: (8)
377: (8)
378: (8)
379: (8)
380: (8)
381: (8)
382: (8)
383: (8)
384: (8)
385: (8)
386: (8)
387: (8)
388: (8)
389: (8)
390: (8)
391: (8)
392: (8)
393: (8)
394: (8)
395: (8)
396: (8)
397: (8)
398: (8)
399: (8)
400: (8)
401: (8)
402: (8)
403: (8)
404: (8)
405: (8)
406: (8)
407: (8)
408: (8)
409: (8)
410: (8)
411: (8)
412: (8)
413: (8)
414: (8)
415: (8)
416: (8)
417: (8)
418: (8)
419: (8)
420: (8)
421: (8)
422: (8)
423: (8)
424: (8)
425: (8)
426: (8)
427: (8)
428: (8)
429: (8)
430: (8)
431: (8)
432: (8)
433: (8)
434: (8)
435: (8)
436: (8)
437: (8)
438: (8)
439: (8)
440: (8)
441: (8)
442: (8)
443: (8)
444: (8)
445: (8)
446: (8)
447: (8)
448: (8)
449: (8)
450: (8)
451: (8)
452: (8)
453: (8)
454: (8)
455: (8)
456: (8)
457: (8)
458: (8)
459: (8)
460: (8)
461: (8)
462: (8)
463: (8)
464: (8)
465: (8)
466: (8)
467: (8)
468: (8)
469: (8)
470: (8)
471: (8)
472: (8)
473: (8)
474: (8)
475: (8)
476: (8)
477: (8)
478: (8)
479: (8)
480: (8)
481: (8)
482: (8)
483: (8)
484: (8)
485: (8)
486: (8)
487: (8)
488: (8)
489: (8)
490: (8)
491: (8)
492: (8)
493: (8)
494: (8)
495: (8)
496: (8)
497: (8)
498: (8)
499: (8)
500: (8)
501: (8)
502: (8)
503: (8)
504: (8)
505: (8)
506: (8)
507: (8)
508: (8)
509: (8)
510: (8)
511: (8)
512: (8)
513: (8)
514: (8)
515: (8)
516: (8)
517: (8)
518: (8)
519: (8)
520: (8)
521: (8)
522: (8)
523: (8)
524: (8)
525: (8)
526: (8)
527: (8)
528: (8)
529: (8)
530: (8)
531: (8)
532: (8)
533: (8)
534: (8)
535: (8)
536: (8)
537: (8)
538: (8)
539: (8)
540: (8)
541: (8)
542: (8)
543: (8)
544: (8)
545: (8)
546: (8)
547: (8)
548: (8)
549: (8)
550: (8)
551: (8)
552: (8)
553: (8)
554: (8)
555: (8)
556: (8)
557: (8)
558: (8)
559: (8)
560: (8)
561: (8)
562: (8)
563: (8)
564: (8)
565: (8)
566: (8)
567: (8)
568: (8)
569: (8)
570: (8)
571: (8)
572: (8)
573: (8)
574: (8)
575: (8)
576: (8)
577: (8)
578: (8)
579: (8)
580: (8)
581: (8)
582: (8)
583: (8)
584: (8)
585: (8)
586: (8)
587: (8)
588: (8)
589: (8)
590: (8)
591: (8)
592: (8)
593: (8)
594: (8)
595: (8)
596: (8)
597: (8)
598: (8)
599: (8)
600: (8)
601: (8)
602: (8)
603: (8)
604: (8)
605: (8)
606: (8)
607: (8)
608: (8)
609: (8)
610: (8)
611: (8)
612: (8)
613: (8)
614: (8)
615: (8)
616: (8)
617: (8)
618: (8)
619: (8)
620: (8)
621: (8)
622: (8)
623: (8)
624: (8)
625: (8)
626: (8)
627: (8)
628: (8)
629: (8)
630: (8)
631: (8)
632: (8)
633: (8)
634: (8)
635: (8)
636: (8)
637: (8)
638: (8)
639: (8)
640: (8)
641: (8)
642: (8)
643: (8)
644: (8)
645: (8)
646: (8)
647: (8)
648: (8)
649: (8)
650: (8)
651: (8)
652: (8)
653: (8)
654: (8)
655: (8)
656: (8)
657: (8)
658: (8)
659: (8)
660: (8)
661: (8)
662: (8)
663: (8)
664: (8)
665: (8)
666: (8)
667: (8)
668: (8)
669: (8)
670: (8)
671: (8)
672: (8)
673: (8)
674: (8)
675: (8)
676: (8)
677: (8)
678: (8)
679: (8)
680: (8)
681: (8)
682: (8)
683: (8)
684: (8)
685: (8)
686: (8)
687: (8)
688: (8)
689: (8)
690: (8)
691: (8)
692: (8)
693: (8)
694: (8)
695: (8)
696: (8)
697: (8)
698: (8)
699: (8)
700: (8)
701: (8)
702: (8)
703: (8)
704: (8)
705: (8)
706: (8)
707: (8)
708: (8)
709: (8)
710: (8)
711: (8)
712: (8)
713: (8)
714: (8)
715: (8)
716: (8)
717: (8)
718: (8)
719: (8)
720: (8)
721: (8)
722: (8)
723: (8)
724: (8)
725: (8)
726: (8)
727: (8)
728: (8)
729: (8)
730: (8)
731: (8)
732: (8)
733: (8)
734: (8)
735: (8)
736: (8)
737: (8)
738: (8)
739: (8)
740: (8)
741: (8)
742: (8)
743: (8)
744: (8)
745: (8)
746: (8)
747: (8)
748: (8)
749: (8)
750: (8)
751: (8)
752: (8)
753: (8)
754: (8)
755: (8)
756: (8)
757: (8)
758: (8)
759: (8)
760: (8)
761: (8)
762: (8)
763: (8)
764: (8)
765: (8)
766: (8)
767: (8)
768: (8)
769: (8)
770: (8)
771: (8)
772: (8)
773: (8)
774: (8)
775: (8)
776: (8)
777: (8)
778: (8)
779: (8)
780: (8)
781: (8)
782: (8)
783: (8)
784: (8)
785: (8)
786: (8)
787: (8)
788: (8)
789: (8)
790: (8)
791: (8)
792: (8)
793: (8)
794: (8)
795: (8)
796: (8)
797: (8)
798: (8)
799: (8)
800: (8)
801: (8)
802: (8)
803: (8)
804: (8)
805: (8)
806: (8)
807: (8)
808: (8)
809: (8)
810: (8)
811: (8)
812: (8)
813: (8)
814: (8)
815: (8)
816: (8)
817: (8)
818: (8)
819: (8)
820: (8)
821: (8)
822: (8)
823: (8)
824: (8)
825: (8)
826: (8)
827: (8)
828: (8)
829: (8)
830: (8)
831: (8)
832: (8)
833: (8)
834: (8)
835: (8)
836: (8)
837: (8)
838: (8)
839: (8)
840: (8)
841: (8)
842: (8)
843: (8)
844: (8)
845: (8)
846: (8)
847: (8)
848: (8)
849: (8)
850: (8)
851: (8)
852: (8)
853: (8)
854: (8)
855: (8)
856: (8)
857: (8)
858: (8)
859: (8)
860: (8)
861: (8)
862: (8)
863: (8)
864: (8)
865: (8)
866: (8)
867: (8)
868: (8)
869: (8)
870: (8)
871: (8)
872: (8)
873: (8)
874: (8)
875: (8)
876: (8)
877: (8)
878: (8)
879: (8)
880: (8)
881: (8)
882: (8)
883: (8)
884: (8)
885: (8)
886: (8)
887: (8)
888: (8)
889: (8)
890: (8)
891: (8)
892: (8)
893: (8)
894: (8)
895: (8)
896: (8)
897: (8)
898: (8)
899: (8)
900: (8)
901: (8)
902: (8)
903: (8)
904: (8)
905: (8)
906: (8)
907: (8)
908: (8)
909: (8)
910: (8)
911: (8)
912: (8)
913: (8)
914: (8)
915: (8)
916: (8)
917: (8)
918: (8)
919: (8)
920: (8)
921: (8)
922: (8)
923: (8)
924: (8)
925: (8)
926: (8)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
951: (8)
952: (8)
953: (8)
954: (8)
955: (8)
956: (8)
957: (8)
958: (8)
959: (8)
960: (8)
961: (8)
962: (8)
963: (8)
964: (8)
965: (8)
966: (8)
967: (8)
968: (8)
969: (8)
970: (8)
971: (8)
972: (8)
973: (8)
974: (8)
975: (8)
976: (8)
977: (8)
978: (8)
979: (8)
980: (8)
981: (8)
982: (8)
983: (8)
984: (8)
985: (8)
986: (8)
987: (8)
988: (8)
989: (8)
990: (8)
991: (8)
992: (8)
993: (8)
994: (8)
995: (8)
996: (8)
997: (8)
998: (8)
999: (8)
1000: (8)
1001: (8)
1002: (8)
1003: (8)
1004: (8)
1005: (8)
1006: (8)
1007: (8)
1008: (8)
1009: (8)
1010: (8)
1011: (8)
1012: (8)
1013: (8)
1014: (8)
1015: (8)
1016: (8)
1017: (8)
1018: (8)
1019: (8)
1020: (8)
1021: (8)
1022: (8)
1023: (8)
1024: (8)
1025: (8)
1026: (8)
1027: (8)
1028: (8)
1029: (8)
1030: (8)
1031: (8)
1032: (8)
1033: (8)
1034: (8)
1035: (8)
1036: (8)
1037: (8)
1038: (8)
1039: (8)
1040: (8)
1041: (8)
1042: (8)
1043: (8)
1044: (8)
1045: (8)
1046: (8)
1047: (8)
1048: (8)
1049: (8)
1050: (8)
1051: (8)
1052: (8)
1053: (8)
1054: (8)
1055: (8)
1056: (8)
1057: (8)
1058: (8)
1059: (8)
1060: (8)
1061: (8)
1062: (8)
1063: (8)
1064: (8)
1065: (8)
1066: (8)
1067: (8)
1068: (8)
1069: (8)
1070: (8)
1071: (8)
1072: (8)
1073: (8)
1074: (8)
1075: (8)
1076: (8)
1077: (8)
1078: (8)
1079: (8)
1080: (8)
1081: (8)
1082: (8)
1083: (8)
1084: (8)
1085: (8)
1086: (8)
1087: (8)
1088: (8)
1089: (8)
1090: (8)
1091: (8)
1092: (8)
1093: (8)
1094: (8)
1095: (8)
1096: (8)
1097: (8)
1098: (8)
1099: (8)
1100: (8)
1101: (8)
1102: (8)
1103: (8)
1104: (8)
1105: (8)
1106: (8)
1107: (8)
1108: (8)
1109: (8)
1110: (8)
1111: (8)
1112: (8)
1113: (8)
1114: (8)
1115: (8)
1116: (8)
1117: (8)
1118: (8)
1119: (8)
1120: (8)
1121: (8)
1122: (8)
1123: (8)
1124: (8)
1125: (8)
1126: (8)
1127: (8)
1128: (8)
1129: (8)
1130: (8)
1131: (8)
1132: (8)
1133: (8)
1134: (8)
1135: (8)
1136: (8)
1137: (8)
1138: (8)
1139: (8)
1140: (8)
1141: (8)
1142: (8)
1143: (8)
1144: (8)
1145: (8)
1146: (8)
1147: (8)
1148: (8)
1149: (8)
1150: (8)
1151: (8)
1152: (8)
1153: (8)
1154: (8)
1155: (8)
1156: (8)
1157: (8)
1158: (8)
1159: (8)
1160: (8)
1161: (8)
1162: (8)
1163: (8)
1164: (8)
1165: (8)
1166: (8)
1167: (8)
1168: (8)
1169: (8)
1170: (8)
1171: (8)
1172: (8)
1173: (8)
1174: (8)
1175: (8)
1176: (8)
1177: (8)
1178: (8)
1179: (8)
1180: (8)
1181: (8)
1182: (8)
1183: (8)
1184: (8)
1185: (8)
1186: (8)
1187: (8)
1188: (8)
1189: (8)
1190: (8)
1191: (8)
1192: (8)
1193: (8)
1194: (8)
1195: (8)
1196: (8)
1197: (8)
1198: (8)
1199: (8)
1200: (8)
1201: (8)
1202: (8)
1203: (8)
1204: (8)
1205: (8)
1206: (8)
1207: (8)
1208: (8)
1209: (8)
1210: (8)
1211: (8)
1212: (8)
1213: (8)
1214: (8)
1215: (8)
1216: (8)
1217: (8)
1218: (8)
1219: (8)
1220: (8)
1221: (8)
1222: (8)
1223: (8)
1224: (8)
1225: (8)
1226: (8)
1227: (8)
1228: (8)
1229: (8)
1230: (8)
1231: (8)
1232: (8)
1233: (8)
1234: (8)
1235: (8)
1236: (8)
1237: (8)
1238: (8)
1239: (8)
1240: (8)
1241: (8)
1242: (8)
1243: (8)
1244: (8)
1245: (8)
1246: (8)
1247: (8)
1248: (8)
1249: (8)
1250: (8)
1251: (8)
1252: (8)
1253: (8)
1254: (8)
1255: (8)
1256: (8)
1257: (8)
1258: (8)
1259: (8)
1260: (8)
1261: (8)
1262: (8)
1263: (8)
1264: (8)
1265: (8)
1266: (8)
1267: (8)
1268: (8)
1269: (8)
1270: (8)
1271: (8)
1272: (8)
1273: (8)
1274: (8)
1275: (8)
1276: (8)
1277: (8)
1278: (8)
1279: (8)
1280: (8)
1281: (8)
1282: (8)
1283: (8)
1284: (8)
1285: (8)
1286: (8)
1287: (8)
1288: (8)
1289: (8)
1290: (8)
1291: (8)
1292: (8)
1293: (8)
1294: (8)
1295: (8)
1296: (8)
1297: (8)
1298: (8)
1299: (8)
1300: (8)
1301: (8)
1302: (8)
1303: (8)
1304: (8)
1305: (8)
1306: (8)
1307: (8)
1308: (8)
1309: (8)
1310: (8)
1311: (8)
1312: (8)
1313: (8)
1314: (8)
1315: (8)
1316: (8)
1317: (8)
1318: (8)
1319: (8)
1320: (8)
1321: (8)
1322: (8)
1323: (8)
1324: (8)
1325: (8)
1326: (8)
1327: (8)
1328: (8)
1329: (8)
1330: (8)
1331: (8)
1332: (8)
1333: (8)
1334: (8)
1335: (8)
1336: (8)
1337: (8)
1338: (8)
1339: (8)
1340: (8)
1341: (8)
1342: (8)
1343: (8)
1344: (8)
1345: (8)
1346: (8)
1347: (8)
1348: (8)
1349: (8)
1350: (8)
1351: (8)
1352: (8)
1353: (8)
1354: (8)
1355: (8)
1356: (8)
1357: (8)
1358: (8)
1359: (8)
1360: (8)
1361: (8)
1362: (8)
1363: (8)
1364: (8)
1365: (8)
1366: (8)
1367: (8)
1368: (8)
1369: (8)
1370: (8)
1371: (8)
1372: (8)
1373: (8)
1374: (8)
1375: (8)
1376: (8)
1377: (8)
1378: (8)
1379: (8)
1380: (8)
1381: (8)
1382: (8)
1383: (8)
1384: (8)
1385: (8)
1386: (8)
1387: (8)
1388: (8)
1389: (8)
1390: (8)
1391: (8)
1392: (8)
1393: (8)
1394: (8)
1395: (8)
1396: (8)
1397: (8)
1398: (8)
1399: (8)
1400: (8)
1401: (8)
1402: (8)
1403: (8)
1404: (8)
1405: (8)
1406: (8)
1407: (8)
1408: (8)
1409: (8)
1410: (8)
1411: (8)
1412: (8)
1413: (8)
1414: (8)
1415: (8)
1416: (8)
1417: (8)
1418: (8)
1419: (8)
1420: (8)
1421: (8)
1422: (8)
1423: (8)
1424: (8)
1425: (8)
1426: (8)
1427: (8)
1428: (8)
1429: (8)
1430: (8)
1431: (8)
1432: (8)
1433: (8)
1434: (8)
1435: (8)
1436: (8)
1437: (8)
1438: (8)
1439: (8)
1440: (8)
1441: (8)
1442: (8)
1443: (8)
1444: (8)
1445: (8)
1446: (8)
1447: (8)
1448: (8)
1449: (8)
1450: (8)
1451: (8)
1452: (8)
1453: (8)
1454: (8)
1455: (8)
1456: (8)
1457: (8)
1458: (8)
1459: (8)
1460: (8)
1461: (8)
1462: (8)
1463: (8)
1464: (8)
1465: (8)
1466: (8)
1467: (8)
1468: (8)
1469: (8)
1470: (8)
1471: (8)
1472: (8)
1473: (8)
1474: (8)
1475: (8)
1476: (8)
1477: (8)
1478: (8)
1479: (8)
1480: (8)
1481: (8)
1482: (8)
1483: (8)
1484: (8)
1485: (8)
1486: (8)
1487: (8)
1488: (8)
1489: (8)
1490: (8)
1491: (8)
1492: (8)
1493: (8)
1494: (8)
1495: (8)
1496: (8)
1497: (8)
1498: (8)
1499: (8)
1500: (8)
1501: (8)
1502: (8)
1503: (8)
1504: (8)
1505: (8)
1506: (8)
1507: (8)
1508: (8)
1509: (8)
1510: (8)
1511: (8)
1512: (8)
1513: (8)
1514: (8)
1515: (8)
1516: (8)
1517: (8)
1518: (8)
1519: (8)
1520: (8)
1521: (8)
1522: (8)
1523: (8)
1524: (8)
1525: (8)
1526: (8)
1527: (8)
1528: (8)
1529: (8)
1530: (8)
1531: (8)
1532: (8)
1533: (8)
1534: (8)
1535: (8)
1536: (8)
1537: (8)
1538: (8)
1539: (8)
1540: (8)
1541: (8)
1542: (8)
1543: (8)
1544: (8)
1545: (8)
1546: (8)
1547: (8)
1548: (8)
1549: (8)
1550: (8)
1551: (8)
1552: (8)
1553: (8)
1554: (8)
1555: (8)
1556: (8)
1557: (8)
1558: (8)
1559: (8)
1560: (8)
1561: (8)
1562: (8)
1563: (8)
1564: (8)
1565: (8)
1566: (8)
1567: (8)
1568: (8)
1569: (8)
1570: (8)
1571: (8)
1572: (8)
1573: (8)
1574: (8)
1575: (8)
1576: (8)
1577: (8)
1578: (8)
1579: (8)
1580: (8)
1581: (8)
1582: (8)
1583: (8)
1584: (8)
1585: (8)
1586: (8)
1587: (8)
1588: (8)
1589: (8)
1590: (8)
1591: (8)
1592: (8)
1593: (8)
1594: (8)
1595: (8)
1596: (8)
1597: (8)
1598: (8)
1599: (8)
1600: (8)
1601: (8)
1602: (8)
1603: (8)
1604: (8)
1605: (8)
1606: (8)
1607: (8)
1608: (8)
1609: (8)
1610: (8)
1611: (8)
1612: (8)
1613: (8)
1614: (8)
1615: (8)
1616: (8)
1617: (8)
1618: (8)
1619: (8)
1620: (8)
1621: (8)
1622: (8)
1623: (8)
1624: (8)
1625: (8)
1626: (8)
1627: (8)
1628: (8)
1629: (8)
1630: (8)
1631: (8)
1632: (8)
1633: (8)
1634: (8)
1635: (8)
1636: (8)
1637: (8)
1638: (8)
1639: (8)
1640: (8)
1641: (8)
1642: (8)
1643: (8)
1644: (8)
1645: (8)
1646: (8)
1647: (8)
1648: (8)
1649: (8)
1650: (8)
1651: (8)
1652: (8)
1653: (8)
1654: (8)
1655: (8)
1656: (8
```

```

326: (12)                                A float representing a value along the number line.
327: (8)                                 Examples
328: (8)
329: (12)
330: (12)
331: (12)
332: (12)
333: (12)
334: (12)
335: (12)
336: (12)
337: (8)
338: (8)
339: (8)
340: (8)
341: (8)
unit_vect)
342: (8)                                point = np.asarray(point)
343: (4)                                 start, end = self.get_start_and_end()
344: (8)                                 unit_vect = normalize(end - start)
345: (8)                                 proportion = np.dot(point - start, unit_vect) / np.dot(end - start,
346: (4)                                 unit_vect)
347: (4)                                return interpolate(self.x_min, self.x_max, proportion)
348: (8)                                 def n2p(self, number: float | np.ndarray) -> np.ndarray:
349: (8)                                 """Abbreviation for :meth:`~.NumberLine.number_to_point`."""
350: (8)                                 return self.number_to_point(number)
351: (4)                                 def p2n(self, point: Sequence[float]) -> float:
352: (8)                                 """Abbreviation for :meth:`~.NumberLine.point_to_number`."""
353: (4)                                 return self.point_to_number(point)
354: (8)                                 def get_unit_size(self) -> float:
355: (8)                                 return self.get_length() / (self.x_range[1] - self.x_range[0])
356: (8)                                 def get_unit_vector(self) -> np.ndarray:
357: (8)                                 return super().get_unit_vector() * self.unit_size
358: (8)                                 def get_number_mobject(
359: (8)                                     self,
360: (8)                                     x: float,
361: (4)                                     direction: Sequence[float] | None = None,
362: (8)                                     buff: float | None = None,
363: (8)                                     font_size: float | None = None,
364: (8)                                     label_constructor: VMobject | None = None,
365: (8)                                     **number_config,
366: (8)                                     ) -> VMobject:
367: (12)                                """Generates a positioned :class:`~.DecimalNumber` mobject
368: (8)                                generated according to ``label_constructor``.
369: (12)                                Parameters
370: (8)
371: (12)                                     x
372: (8)                                     The x-value at which the mobject should be positioned.
373: (8)                                     direction
374: (8)                                     Determines the direction at which the label is positioned next to
375: (12)                                     the line.
376: (8)
377: (12)                                     buff
378: (8)                                     The distance of the label from the line.
379: (8)                                     font_size
380: (8)                                     The font size of the label mobject.
381: (12)                                     label_constructor
382: (8)                                     The :class:`~.VMobject` class that will be used to construct the
383: (8)                                     label.
384: (12)                                     Defaults to the ``label_constructor`` attribute of the number line
385: (12)                                     if not specified.
386: (8)                                 Returns
387: (8)
388: (8)                                     :class:`~.DecimalNumber`
389: (8)                                     The positioned mobject.
390: (12)
391: (8)

```

```

392: (12)           font_size = self.font_size
393: (8)            if label_constructor is None:
394: (12)              label_constructor = self.label_constructor
395: (8)            num_mob = DecimalNumber(
396: (12)                x, font_size=font_size, mob_class=label_constructor,
**number_config
397: (8)            )
398: (8)            num_mob.next_to(self.number_to_point(x), direction=direction,
buff=buf)
399: (8)            if x < 0 and self.label_direction[0] == 0:
400: (12)                num_mob.shift(num_mob[0].width * LEFT / 2)
401: (8)            return num_mob
402: (4)        def get_number_mobjects(self, *numbers, **kwargs) -> VGroup:
403: (8)            if len(numbers) == 0:
404: (12)                numbers = self.default_numbers_to_display()
405: (8)            return VGroup([self.get_number_mobject(number, **kwargs) for number in
numbers])
406: (4)        def get_labels(self) -> VGroup:
407: (8)            return self.get_number_mobjects()
408: (4)        def add_numbers(
409: (8)            self,
410: (8)            x_values: Iterable[float] | None = None,
411: (8)            excluding: Iterable[float] | None = None,
412: (8)            font_size: float | None = None,
413: (8)            label_constructor: VMobject | None = None,
414: (8)            **kwargs,
415: (4)        ):
416: (8)            """Adds :class:`~.DecimalNumber` mobjects representing their position
417: (8)            at each tick of the number line. The numbers can be accessed after
creation
418: (8)            via ``self.numbers``.
419: (8)        Parameters
420: (8)            -----
421: (8)            x_values
422: (12)                An iterable of the values used to position and create the labels.
423: (12)                Defaults to the output produced by
:meth:`~.NumberLine.get_tick_range`
424: (8)            excluding
425: (12)                A list of values to exclude from :attr:`x_values`.
426: (8)            font_size
427: (12)                The font size of the labels. Defaults to the ``font_size``
attribute
428: (12)                of the number line.
429: (8)            label_constructor
430: (12)                The :class:`~.VMobject` class that will be used to construct the
label.
431: (12)                Defaults to the ``label_constructor`` attribute of the number line
432: (12)                if not specified.
433: (8)        """
434: (8)        if x_values is None:
435: (12)            x_values = self.get_tick_range()
436: (8)        if excluding is None:
437: (12)            excluding = self.numbers_to_exclude
438: (8)        if font_size is None:
439: (12)            font_size = self.font_size
440: (8)        if label_constructor is None:
441: (12)            label_constructor = self.label_constructor
442: (8)        numbers = VGroup()
443: (8)        for x in x_values:
444: (12)            if x in excluding:
445: (16)                continue
446: (12)            numbers.add(
447: (16)                self.get_number_mobject(
448: (20)                    x,
449: (20)                    font_size=font_size,
450: (20)                    label_constructor=label_constructor,
451: (20)                    **kwargs,
452: (16)
453: (12)            )
)

```

```

454: (8)             self.add(numbers)
455: (8)             self.numbers = numbers
456: (8)             return self
457: (4)             def add_labels(
458: (8)                 self,
459: (8)                 dict_values: dict[float, str | float | VMobject],
460: (8)                 direction: Sequence[float] = None,
461: (8)                 buff: float | None = None,
462: (8)                 font_size: float | None = None,
463: (8)                 label_constructor: VMobject | None = None,
464: (4)             ):
465: (8)                 """Adds specifically positioned labels to the :class:`~.NumberLine`  

using a ``dict``.  

466: (8)             The labels can be accessed after creation via ``self.labels``.  

467: (8)             Parameters  

468: (8)             -----  

469: (8)                 dict_values  

470: (12)                 A dictionary consisting of the position along the number line and  

the mobject to be added:  

471: (12)                 ``{1: Tex("Monday"), 3: Tex("Tuesday")}``.  

:attr:`label_constructor` will be used  

472: (12)                 to construct the labels if the value is not a mobject (``str`` or  

``float``).  

473: (8)  

474: (12)                 direction  

475: (8)                     Determines the direction at which the label is positioned next to  

the line.  

476: (12)                 buff  

477: (8)                     The distance of the label from the line.  

478: (12)                 font_size  

479: (8)                     The font size of the mobject to be positioned.  

480: (12)                 label_constructor  

481: (8)                     The :class:`~.VMobject` class that will be used to construct the  

label.  

482: (12)                 Defaults to the ``label_constructor`` attribute of the number line  

if not specified.  

483: (8)             Raises  

484: (8)             -----  

485: (8)             AttributeError  

486: (12)                 If the label does not have a ``font_size`` attribute, an  

``AttributeError`` is raised.  

487: (8)             """
488: (8)                 direction = self.label_direction if direction is None else direction
489: (8)                 buff = self.line_to_number_buff if buff is None else buff
490: (8)                 font_size = self.font_size if font_size is None else font_size
491: (8)                 if label_constructor is None:
492: (12)                     label_constructor = self.label_constructor
493: (8)                 labels = VGroup()
494: (8)                 for x, label in dict_values.items():
495: (12)                     if isinstance(label, str) and label_constructor is MathTex:
496: (16)                         label = Tex(label)
497: (12)                     else:
498: (16)                         label = self._create_label_tex(label, label_constructor)
499: (12)                     if hasattr(label, "font_size"):
500: (16)                         label.font_size = font_size
501: (12)                     else:
502: (16)                         raise AttributeError(f"{label} is not compatible with  

add_labels.")
503: (12)                         label.next_to(self.number_to_point(x), direction=direction,
504: (8)                             labels.add(label)
505: (8)                         self.labels = labels
506: (8)                         self.add(labels)
507: (8)                         return self
508: (4)             def _create_label_tex(
509: (8)                 self,
510: (8)                 label_tex: str | float | VMobject,
511: (8)                 label_constructor: Callable | None = None,
512: (8)                 **kwargs,
513: (4)             ) -> VMobject:

```

```

514: (8)             """Checks if the label is a :class:`~.VMobject`, otherwise, creates a
515: (8)             label by passing ``label_tex`` to ``label_constructor``.
516: (8)             Parameters
517: (8)
518: (8)             label_tex
519: (12)             The label for which a mobject should be created. If the label
already
520: (12)             is a mobject, no new mobject is created.
521: (8)             label_constructor
522: (12)             Optional. A class or function returning a mobject when
523: (12)             passing ``label_tex`` as an argument. If ``None`` is passed
524: (12)             (the default), the label constructor from the
:attr:`.label_constructor` attribute is used.
525: (12)
526: (8)             Returns
527: (8)
528: (8)             :class:`~.VMobject`
529: (12)             The label.
530: (8)
531: (8)             if label_constructor is None:
532: (12)                 label_constructor = self.label_constructor
533: (8)
534: (12)                 if isinstance(label_tex, (VMobject, OpenGLVMobject)):
535: (8)                     return label_tex
536: (12)                 else:
537: (4)                     return label_constructor(label_tex, **kwargs)
538: (4)             @staticmethod
539: (8)             def _decimal_places_from_step(step) -> int:
540: (8)                 step = str(step)
541: (12)                 if "." not in step:
542: (8)                     return 0
543: (0)                 return len(step.split("."))[-1]
544: (4)             class UnitInterval(NumberLine):
545: (8)
546: (8)                 def __init__(
547: (8)                     self,
548: (8)                     unit_size=10,
549: (8)                     numbers_with_elongated_ticks=None,
550: (4)                     decimal_number_config=None,
551: (8)                     **kwargs,
552: (12)                 ):
553: (12)                     numbers_with_elongated_ticks = (
554: (12)                         [0, 1]
555: (8)                         if numbers_with_elongated_ticks is None
556: (8)                         else numbers_with_elongated_ticks
557: (12)                     )
558: (16)                     decimal_number_config = (
559: (12)                         {
560: (12)                             "num_decimal_places": 1,
561: (12)                         }
562: (8)                         if decimal_number_config is None
563: (8)                         else decimal_number_config
564: (12)                     )
565: (12)                     super().__init__(
566: (12)                         x_range=(0, 1, 0.1),
567: (12)                         unit_size=unit_size,
568: (12)                         numbers_with_elongated_ticks=numbers_with_elongated_ticks,
569: (8)                         decimal_number_config=decimal_number_config,
570: (8)                         **kwargs,
571: (8)                     )

```

## File 69 - probability.py:

```

1: (0)             """Mobjects representing objects from probability theory and statistics."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["SampleSpace", "BarChart"]
4: (0)             from typing import Iterable, MutableSequence, Sequence
5: (0)             import numpy as np
6: (0)             from manim import config, logger

```

```

7: (0)         from manim.constants import *
8: (0)         from manim.mobject.geometry.polygram import Rectangle
9: (0)         from manim.mobject.graphing.coordinate_systems import Axes
10: (0)        from manim.mobject.mobject import Mobject
11: (0)        from manim.mobject.opengl.opengl_mobject import OpenGLObject
12: (0)        from manim.mobject.svg.brace import Brace
13: (0)        from manim.mobject.text.tex_mobject import MathTex, Tex
14: (0)        from manim.mobject.types.vectorized_mobject import VGroup, VMobject
15: (0)        from manim.utils.color import (
16: (4)            BLUE_E,
17: (4)            DARK_GREY,
18: (4)            GREEN_E,
19: (4)            LIGHT_GREY,
20: (4)            MAROON_B,
21: (4)            YELLOW,
22: (4)            ParsableManimColor,
23: (4)            color_gradient,
24: (0)        )
25: (0)        from manim.utils.iterables import tuplify
26: (0)        EPSILON = 0.0001
27: (0)        class SampleSpace(Rectangle):
28: (4)            """A mobject representing a twodimensional rectangular
29: (4)            sampling space.
30: (4)            Examples
31: (4)            -----
32: (4)            .. manim:: ExampleSampleSpace
33: (8)                :save_last_frame:
34: (8)                class ExampleSampleSpace(Scene):
35: (12)                    def construct(self):
36: (16)                        poly1 = SampleSpace(stroke_width=15, fill_opacity=1)
37: (16)                        poly2 = SampleSpace(width=5, height=3, stroke_width=5,
fill_opacity=0.5)
38: (16)                        poly3 = SampleSpace(width=2, height=2, stroke_width=5,
fill_opacity=0.1)
39: (16)                        poly3.divide_vertically(p_list=np.array([0.37, 0.13, 0.5]),
colors=[BLACK, WHITE, GRAY], vect=RIGHT)
40: (16)                        poly_group = VGroup(poly1, poly2, poly3).arrange()
41: (16)                        self.add(poly_group)
42: (4)            """
43: (4)            def __init__(
44: (8)                self,
45: (8)                height=3,
46: (8)                width=3,
47: (8)                fill_color=DARK_GREY,
48: (8)                fill_opacity=1,
49: (8)                stroke_width=0.5,
50: (8)                stroke_color=LIGHT_GREY,
51: (8)                default_label_scale_val=1,
52: (4)            ):
53: (8)                super().__init__(
54: (12)                    height=height,
55: (12)                    width=width,
56: (12)                    fill_color=fill_color,
57: (12)                    fill_opacity=fill_opacity,
58: (12)                    stroke_width=stroke_width,
59: (12)                    stroke_color=stroke_color,
60: (8)                )
61: (8)                self.default_label_scale_val = default_label_scale_val
62: (4)            def add_title(self, title="Sample space", buff=MED_SMALL_BUFF):
63: (8)                title_mob = Tex(title)
64: (8)                if title_mob.width > self.width:
65: (12)                    title_mob.width = self.width
66: (8)                title_mob.next_to(self, UP, buff=buff)
67: (8)                self.title = title_mob
68: (8)                self.add(title_mob)
69: (4)            def add_label(self, label):
70: (8)                self.label = label
71: (4)            def complete_p_list(self, p_list):
72: (8)                new_p_list = list(tuplify(p_list))

```

```

73: (8)           remainder = 1.0 - sum(new_p_list)
74: (8)           if abs(remainder) > EPSILON:
75: (12)             new_p_list.append(remainder)
76: (8)           return new_p_list
77: (4) def get_division_along_dimension(self, p_list, dim, colors, vect):
78: (8)     p_list = self.complete_p_list(p_list)
79: (8)     colors = color_gradient(colors, len(p_list))
80: (8)     last_point = self.get_edge_center(-vect)
81: (8)     parts = VGroup()
82: (8)     for factor, color in zip(p_list, colors):
83: (12)       part = SampleSpace()
84: (12)       part.set_fill(color, 1)
85: (12)       part.replace(self, stretch=True)
86: (12)       part.stretch(factor, dim)
87: (12)       part.move_to(last_point, -vect)
88: (12)       last_point = part.get_edge_center(vect)
89: (12)       parts.add(part)
90: (8)   return parts
91: (4) def get_horizontal_division(self, p_list, colors=[GREEN_E, BLUE_E],
vect=DOWN):
92: (8)   return self.get_division_along_dimension(p_list, 1, colors, vect)
93: (4) def get_vertical_division(self, p_list, colors=[MAROON_B, YELLOW],
vect=RIGHT):
94: (8)   return self.get_division_along_dimension(p_list, 0, colors, vect)
95: (4) def divide_horizontally(self, *args, **kwargs):
96: (8)   self.horizontal_parts = self.get_horizontal_division(*args, **kwargs)
97: (8)   self.add(self.horizontal_parts)
98: (4) def divide_vertically(self, *args, **kwargs):
99: (8)   self.vertical_parts = self.get_vertical_division(*args, **kwargs)
100: (8)  self.add(self.vertical_parts)
101: (4) def get_subdivision_braces_and_labels(
102: (8)   self,
103: (8)   parts,
104: (8)   labels,
105: (8)   direction,
106: (8)   buff=SMALL_BUFF,
107: (8)   min_num_quads=1,
108: (4) ):
109: (8)   label_mobs = VGroup()
110: (8)   braces = VGroup()
111: (8)   for label, part in zip(labels, parts):
112: (12)     brace = Brace(part, direction, min_num_quads=min_num_quads,
buff=buff)
113: (12)
114: (16)
115: (12)
116: (16)
117: (16)
118: (12)
119: (12)
120: (12)
121: (8)
122: (8)
123: (8)
124: (12)
125: (12)
126: (12)
127: (8)
128: (8)
129: (4)   return VGroup(parts.braces, parts.labels)
130: (8) def get_side_braces_and_labels(self, labels, direction=LEFT, **kwargs):
131: (8)   assert hasattr(self, "horizontal_parts")
132: (8)   parts = self.horizontal_parts
133: (12)   return self.get_subdivision_braces_and_labels(
134: (8)     parts, labels, direction, **kwargs
135: (4) )
136: (8) def get_top_braces_and_labels(self, labels, **kwargs):
137: (8)   assert hasattr(self, "vertical_parts")
138: (8)   parts = self.vertical_parts

```

```

**kwargs)
139: (4)         def get_bottom_braces_and_labels(self, labels, **kwargs):
140: (8)             assert hasattr(self, "vertical_parts")
141: (8)             parts = self.vertical_parts
142: (8)             return self.get_subdivision_braces_and_labels(parts, labels, DOWN,
**kwargs)
143: (4)         def add_braces_and_labels(self):
144: (8)             for attr in "horizontal_parts", "vertical_parts":
145: (12)                 if not hasattr(self, attr):
146: (16)                     continue
147: (12)                 parts = getattr(self, attr)
148: (12)                 for subattr in "braces", "labels":
149: (16)                     if hasattr(parts, subattr):
150: (20)                         self.add(getattr(parts, subattr))
151: (4)         def __getitem__(self, index):
152: (8)             if hasattr(self, "horizontal_parts"):
153: (12)                 return self.horizontal_parts[index]
154: (8)             elif hasattr(self, "vertical_parts"):
155: (12)                 return self.vertical_parts[index]
156: (8)             return self.split()[index]
157: (0)     class BarChart(Axes):
158: (4)         """Creates a bar chart. Inherits from :class:`~.Axes`, so it shares its
methods
159: (4)         and attributes. Each axis inherits from :class:`~.NumberLine`, so pass in
``x_axis_config``/``y_axis_config``
160: (4)             to control their attributes.
161: (4)         Parameters
162: (4)             -----
163: (4)             values
164: (8)                 A sequence of values that determines the height of each bar. Accepts
negative values.
165: (4)             bar_names
166: (8)                 A sequence of names for each bar. Does not have to match the length of
``values``.
167: (4)             y_range
168: (8)                 The y_axis range of values. If ``None``, the range will be calculated
min/max of ``values`` and the step will be calculated based on
``y_length``.
169: (8)             x_length
170: (4)                 The length of the x-axis. If ``None``, it is automatically calculated
based on
171: (8)                 the number of values and the width of the screen.
172: (8)             y_length
173: (4)                 The length of the y-axis.
174: (8)             bar_colors
175: (4)                 The color for the bars. Accepts a sequence of colors (can contain just
one item).
176: (8)                 If the length of ``bar_colors`` does not match that of ``values``,
intermediate colors will be automatically determined.
177: (8)             bar_width
178: (8)                 The length of a bar. Must be between 0 and 1.
179: (4)             bar_fill_opacity
180: (8)                 The fill opacity of the bars.
181: (4)             bar_stroke_width
182: (8)                 The stroke width of the bars.
183: (4)             Examples
184: (4)             -----
185: (4)             .. manim:: BarChartExample
186: (4)                 :save_last_frame:
187: (4)                 class BarChartExample(Scene):
188: (8)                     def construct(self):
189: (12)                         chart = BarChart(
190: (16)                             values=[-5, 40, -10, 20, -3],
191: (20)                             bar_names=["one", "two", "three", "four", "five"],
192: (20)                             y_range=[-20, 50, 10],
193: (20)                             y_length=6,
194: (20)                             x_length=10,
195: (20)                             x_axis_config={"font_size": 36},
196: (20)
197: (20)

```

```

198: (16)                               )
199: (16)                         c_bar_lbls = chart.get_bar_labels(font_size=48)
200: (16)                         self.add(chart, c_bar_lbls)
201: (4)   """
202: (4)     def __init__(self,
203: (8)       values: MutableSequence[float],
204: (8)       bar_names: Sequence[str] | None = None,
205: (8)       y_range: Sequence[float] | None = None,
206: (8)       x_length: float | None = None,
207: (8)       y_length: float | None = None,
208: (8)       bar_colors: Iterable[str] = [
209: (12)         "#003f5c",
210: (12)         "#58508d",
211: (12)         "#bc5090",
212: (12)         "#ff6361",
213: (12)         "#ffa600",
214: (12)       ],
215: (8)       bar_width: float = 0.6,
216: (8)       bar_fill_opacity: float = 0.7,
217: (8)       bar_stroke_width: float = 3,
218: (8)       **kwargs,
219: (8)     ):
220: (4)       if isinstance(bar_colors, str):
221: (8)         logger.warning(
222: (12)           "Passing a string to `bar_colors` has been deprecated since
v0.15.2 and will be removed after v0.17.0, the parameter must be a list.  "
223: (16)           )
224: (12)         bar_colors = list(bar_colors)
225: (12)       y_length = y_length if y_length is not None else config.frame_height -
4
226: (8)       self.values = values
227: (8)       self.bar_names = bar_names
228: (8)       self.bar_colors = bar_colors
229: (8)       self.bar_width = bar_width
230: (8)       self.bar_fill_opacity = bar_fill_opacity
231: (8)       self.bar_stroke_width = bar_stroke_width
232: (8)       x_range = [0, len(self.values), 1]
233: (8)       if y_range is None:
234: (8)         y_range = [
235: (12)           min(0, min(self.values)),
236: (16)           max(0, max(self.values)),
237: (16)           round(max(self.values) / y_length, 2),
238: (16)         ]
239: (12)       elif len(y_range) == 2:
240: (8)         y_range = [*y_range, round(max(self.values) / y_length, 2)]
241: (12)       if x_length is None:
242: (8)         x_length = min(len(self.values), config.frame_width - 2)
243: (12)       x_axis_config = {"font_size": 24, "label_constructor": Tex}
244: (8)       self._update_default_configs(
245: (8)         (x_axis_config,), (kwargs.pop("x_axis_config", None),)
246: (12)       )
247: (8)       self.bars: VGroup = VGroup()
248: (8)       self.x_labels: VGroup | None = None
249: (8)       self.bar_labels: VGroup | None = None
250: (8)       super().__init__(
251: (8)         x_range=x_range,
252: (12)         y_range=y_range,
253: (12)         x_length=x_length,
254: (12)         y_length=y_length,
255: (12)         x_axis_config=x_axis_config,
256: (12)         tips=kwargs.pop("tips", False),
257: (12)         **kwargs,
258: (12)       )
259: (8)       self._add_bars()
260: (8)       if self.bar_names is not None:
261: (8)         self._add_x_axis_labels()
262: (12)       self.y_axis.add_numbers()
263: (8)     def _update_colors(self):

```

```

265: (8)             """Initialize the colors of the bars of the chart.
266: (8)             Sets the color of ``self.bars`` via ``self.bar_colors``.
267: (8)             Primarily used when the bars are initialized with ``self._add_bars``
268: (8)             or updated via ``self.change_bar_values``.
269: (8)
270: (8)             self.bars.set_color_by_gradient(*self.bar_colors)
271: (4)             def _add_x_axis_labels(self):
272: (8)                 """Essentially :meth`:`~.NumberLine.add_labels`, but differs in that
273: (8)                 the direction of the label with respect to the x_axis changes to UP or
DOWN
274: (8)                 depending on the value.
275: (8)                 UP for negative values and DOWN for positive values.
276: (8)
277: (8)                 val_range = np.arange(
278: (12)                     0.5, len(self.bar_names), 1
279: (8)                 ) # 0.5 shifted so that labels are centered, not on ticks
280: (8)                 labels = VGroup()
281: (8)                 for i, (value, bar_name) in enumerate(zip(val_range, self.bar_names)):
282: (12)                     if self.values[i] < 0:
283: (16)                         direction = UP
284: (12)                     else:
285: (16)                         direction = DOWN
286: (12)                     bar_name_label = self.x_axis.label_constructor(bar_name)
287: (12)                     bar_name_label.font_size = self.x_axis.font_size
288: (12)                     bar_name_label.next_to(
289: (16)                         self.x_axis.number_to_point(value),
290: (16)                         direction=direction,
291: (16)                         buff=self.x_axis.line_to_number_buff,
292: (12)                     )
293: (12)                     labels.add(bar_name_label)
294: (8)                     self.x_axis.labels = labels
295: (8)                     self.x_axis.add(labels)
296: (4)             def _create_bar(self, bar_number: int, value: float) -> Rectangle:
297: (8)                 """Creates a positioned bar on the chart.
298: (8)                 Parameters
299: (8)
300: (8)                 bar_number
301: (12)                     Determines the x-position of the bar.
302: (8)                 value
303: (12)                     The value that determines the height of the bar.
304: (8)                 Returns
305: (8)
306: (8)                 Rectangle
307: (12)                     A positioned rectangle representing a bar on the chart.
308: (8)
309: (8)                 bar_h = abs(self.c2p(0, value)[1] - self.c2p(0, 0)[1])
310: (8)                 bar_w = self.c2p(self.bar_width, 0)[0] - self.c2p(0, 0)[0]
311: (8)                 bar = Rectangle(
312: (12)                     height=bar_h,
313: (12)                     width=bar_w,
314: (12)                     stroke_width=self.bar_stroke_width,
315: (12)                     fill_opacity=self.bar_fill_opacity,
316: (8)
317: (8)                 pos = UP if (value >= 0) else DOWN
318: (8)                 bar.next_to(self.c2p(bar_number + 0.5, 0), pos, buff=0)
319: (8)
320: (4)             def _add_bars(self) -> None:
321: (8)                 for i, value in enumerate(self.values):
322: (12)                     tmp_bar = self._create_bar(bar_number=i, value=value)
323: (12)                     self.bars.add(tmp_bar)
324: (8)
325: (8)                     self._update_colors()
326: (4)                     self.add_to_back(self.bars)
327: (8)             def get_bar_labels(
328: (8)                 self,
329: (8)                 color: ParsableManimColor | None = None,
330: (8)                 font_size: float = 24,
331: (8)                 buff: float = MED_SMALL_BUFF,
332: (8)                 label_constructor: type[VMobject] = Tex,
332: (4)             ):

```

```

333: (8)                                     """Annotates each bar with its corresponding value. Use
``self.bar_labels`` to access the
334: (8)                                     labels after creation.
335: (8)                                     Parameters
336: (8)                                     -----
337: (8)                                     color
338: (12)                                     The color of each label. By default ``None`` and is based on the
parent's bar color.
339: (8)
340: (12)
341: (8)
342: (12)
343: (8)
344: (12)
:class:`~.Tex` .
345: (8)
346: (8)
347: (8)
348: (12)
349: (12)
350: (16)
351: (20)
y_range=[0, 10, 11]
352: (20)
353: (24)
354: (20)
355: (20)
356: (8)
357: (8)
358: (8)
359: (12)
360: (12)
361: (16)
362: (12)
363: (16)
364: (12)
365: (12)
366: (12)
367: (12)
368: (8)
369: (4)
True):
370: (8)                                     """Updates the height of the bars of the chart.
371: (8)
372: (8)
373: (8)
374: (12)
375: (12)
376: (8)
377: (12)
``self.bar_colors``.
378: (8)
379: (8)
380: (8)
381: (12)
382: (12)
383: (16)
384: (20)
385: (20)
386: (24)
387: (24)
388: (24)
389: (20)
390: (20)
391: (20)
392: (20)
393: (8)
394: (8)
395: (12)

    Parameters
    -----
    color
        The color of each label. By default ``None`` and is based on the
    font_size
        The font size of each label.
    buff
        The distance from each label to its bar. By default 0.4.
    label_constructor
        The Mobject class to construct the labels, by default
Examples
-----
.. manim:: GetBarLabelsExample
:save_last_frame:
class GetBarLabelsExample(Scene):
    def construct(self):
        chart = BarChart(values=[10, 9, 8, 7, 6, 5, 4, 3, 2, 1],
                          c_bar_lbls = chart.get_bar_labels(
                                color=WHITE, label_constructor=MathTex, font_size=36
                            )
                        self.add(chart, c_bar_lbls)
    """
    bar_labels = VGroup()
    for bar, value in zip(self.bars, self.values):
        bar_lbl = label_constructor(str(value))
        if color is None:
            bar_lbl.set_color(bar.get_fill_color())
        else:
            bar_lbl.set_color(color)
        bar_lbl.font_size = font_size
        pos = UP if (value >= 0) else DOWN
        bar_lbl.next_to(bar, pos, buff=buff)
        bar_labels.add(bar_lbl)
    return bar_labels
def change_bar_values(self, values: Iterable[float], update_colors: bool =
True):
    """Updates the height of the bars of the chart.
    Parameters
    -----
    values
        The values that will be used to update the height of the bars.
        Does not have to match the number of bars.
    update_colors
        Whether to re-initialize the colors of the bars based on
``self.bar_colors``.
Examples
-----
.. manim:: ChangeBarValuesExample
:save_last_frame:
class ChangeBarValuesExample(Scene):
    def construct(self):
        values=[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]
        chart = BarChart(
            values,
            y_range=[-10, 10, 2],
            y_axis_config={"font_size": 24},
        )
        self.add(chart)
        chart.change_bar_values(list(reversed(values)))
        self.add(chart.get_bar_labels(font_size=24))
    """
    for i, (bar, value) in enumerate(zip(self.bars, values)):
        chart_val = self.values[i]

```

```

396: (12)             if chart_val > 0:
397: (16)                 bar_lim = bar.get_bottom()
398: (16)                 aligned_edge = DOWN
399: (12)             else:
400: (16)                 bar_lim = bar.get_top()
401: (16)                 aligned_edge = UP
402: (12)             if chart_val != 0:
403: (16)                 quotient = value / chart_val
404: (16)                 if quotient < 0:
405: (20)                     aligned_edge = UP if chart_val > 0 else DOWN
406: (16)                 bar.stretch_to_fit_height(abs(quotient) * bar.height)
407: (12)             else:
408: (16)                 temp_bar = self._create_bar(i, value)
409: (16)                 self.bars.remove(bar)
410: (16)                 self.bars.insert(i, temp_bar)
411: (12)                 bar.move_to(bar_lim, aligned_edge)
412: (8)             if update_colors:
413: (12)                 self._update_colors()
414: (8)             self.values[: len(values)] = values

```

---

## File 70 - render\_options.py:

```

1: (0)         from __future__ import annotations
2: (0)         import re
3: (0)         from cloup import Choice, option, option_group
4: (0)         from manim.constants import QUALITIES, RendererType
5: (0)         from ... import logger
6: (0)         __all__ = ["render_options"]
7: (0)         def validate_scene_range(ctx, param, value):
8: (4)             try:
9: (8)                 start = int(value)
10: (8)                 return (start,)
11: (4)             except Exception:
12: (8)                 pass
13: (4)             if value:
14: (8)                 try:
15: (12)                     start, end = map(int, re.split(r"[;,\-]", value))
16: (12)                     return start, end
17: (8)                 except Exception:
18: (12)                     logger.error("Couldn't determine a range for -n option.")
19: (12)                     exit()
20: (0)             def validate_resolution(ctx, param, value):
21: (4)                 if value:
22: (8)                     try:
23: (12)                         start, end = map(int, re.split(r"[;,\-]", value))
24: (12)                         return (start, end)
25: (8)                     except Exception:
26: (12)                         logger.error("Resolution option is invalid.")
27: (12)                         exit()
28: (0)             render_options = option_group(
29: (4)                 "Render Options",
30: (4)                 option(
31: (8)                     "-n",
32: (8)                     "--from_animation_number",
33: (8)                     callback=validate_scene_range,
34: (8)                     help="Start rendering from n_0 until n_1. If n_1 is left unspecified,
"
35: (8)                         "renders all scenes after n_0.",
36: (8)                         default=None,
37: (4)                     ),
38: (4)                     option(
39: (8)                         "-a",
40: (8)                         "--write_all",
41: (8)                         is_flag=True,
42: (8)                         help="Render all scenes in the input file.",
43: (8)                         default=None,
44: (4)                     ),

```

```

45: (4)             option(
46: (8)             "--format",
47: (8)             type=Choice(["png", "gif", "mp4", "webm", "mov"],
48: (8)             default=None,
49: (4)             ),
50: (4)             option(
51: (8)             "-s",
52: (8)             "--save_last_frame",
53: (8)             default=None,
54: (8)             is_flag=True,
55: (8)             help="Render and save only the last frame of a scene as a PNG image.",
56: (4)             ),
57: (4)             option(
58: (8)             "-q",
59: (8)             "--quality",
60: (8)             default=None,
61: (8)             type=Choice(
62: (12)             list(reversed([q["flag"] for q in QUALITIES.values() if
63: (12)             q["flag"]])), # type: ignore
64: (8)             case_sensitive=False,
65: (8)             ),
66: (8)             help="Render quality at the follow resolution framerates,
respectively: "
67: (12)             +
68: (16)             ".join(
69: (20)             reversed([
70: (20)               f'{q["pixel_width"]}x{q["pixel_height"]}'
71: (20)             for q in QUALITIES.values()
72: (16)             if q["flag"]
73: (12)           ]
74: (8)             ),
75: (4)             ),
76: (4)             option(
77: (8)             "-r",
78: (8)             "--resolution",
79: (8)             callback=validate_resolution,
80: (8)             default=None,
81: (8)             help='Resolution in "W,H" for when 16:9 aspect ratio isn\'t
possible.',
82: (4)             ),
83: (4)             option(
84: (8)             "--fps",
85: (8)             "--frame_rate",
86: (8)             "frame_rate",
87: (8)             type=float,
88: (8)             default=None,
89: (8)             help="Render at this frame rate.",
90: (4)             ),
91: (4)             option(
92: (8)             "--renderer",
93: (8)             type=Choice(
94: (12)             [renderer_type.value for renderer_type in RendererType],
95: (12)             case_sensitive=False,
96: (8)             ),
97: (8)             help="Select a renderer for your Scene.",
98: (8)             default="cairo",
99: (4)             ),
100: (4)            option(
101: (8)             "-g",
102: (8)             "--save_pngs",
103: (8)             is_flag=True,
104: (8)             default=None,
105: (8)             help="Save each frame as png (Deprecated).",
106: (4)             ),
107: (4)            option(
108: (8)             "-i",

```

```

109: (8)             "--save_as_gif",
110: (8)             default=None,
111: (8)             is_flag=True,
112: (8)             help="Save as a gif (Deprecated).",
113: (4)         ),
114: (4)         option(
115: (8)             "--save_sections",
116: (8)             default=None,
117: (8)             is_flag=True,
118: (8)             help="Save section videos in addition to movie file.",
119: (4)         ),
120: (4)         option(
121: (8)             "-t",
122: (8)             "--transparent",
123: (8)             is_flag=True,
124: (8)             help="Render scenes with alpha channel.",
125: (4)         ),
126: (4)         option(
127: (8)             "--use_projection_fill_shaders",
128: (8)             is_flag=True,
129: (8)             help="Use shaders for OpenGLMobject fill which are compatible with
transformation matrices.",
130: (8)             default=None,
131: (4)         ),
132: (4)         option(
133: (8)             "--use_projection_stroke_shaders",
134: (8)             is_flag=True,
135: (8)             help="Use shaders for OpenGLMobject stroke which are compatible with
transformation matrices.",
136: (8)             default=None,
137: (4)         ),
138: (0)     )
-----
```

## File 71 - shape\_matchers.py:

```

1: (0)             """Mobjects used to mark and annotate other mobjects."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["SurroundingRectangle", "BackgroundRectangle", "Cross",
"Underline"]
4: (0)             from typing import Any
5: (0)             from typing_extensions import Self
6: (0)             from manim import config, logger
7: (0)             from manim.constants import *
8: (0)             from manim.mobject.geometry.line import Line
9: (0)             from manim.mobject.geometry.polygram import RoundedRectangle
10: (0)            from manim.mobject.mobject import Mobject
11: (0)            from manim.mobject.types.vectorized_mobject import VGroup
12: (0)            from manim.utils.color import BLACK, RED, YELLOW, ManimColor,
ParsableManimColor
13: (0)            class SurroundingRectangle(RoundedRectangle):
14: (4)                r"""A rectangle surrounding a :class:`~.Mobject`"""
15: (4)                Examples
16: (4)                -----
17: (4)                .. manim:: SurroundingRectExample
18: (8)                    :save_last_frame:
19: (8)                    class SurroundingRectExample(Scene):
20: (12)                        def construct(self):
21: (16)                            title = Title("A Quote from Newton")
22: (16)                            quote = Text(
23: (20)                                "If I have seen further than others, \n"
24: (20)                                "it is by standing upon the shoulders of giants.",
25: (20)                                color=BLUE,
26: (16)                                ).scale(0.75)
27: (16)                                box = SurroundingRectangle(quote, color=YELLOW,
buff=MED_LARGE_BUFF)
28: (16)                                t2 = Tex(r"Hello World").scale(1.5)
29: (16)                                box2 = SurroundingRectangle(t2, corner_radius=0.2)
```

```

30: (16)                                     mobjects = VGroup(VGroup(box, quote), VGroup(t2,
box2)).arrange(DOWN)
31: (16)                                         self.add(title, mobjects)
32: (4)                                          """
33: (4)                                          def __init__(
34: (8)                                              self,
35: (8)                                              mobobject: Mobject,
36: (8)                                              color: ParsableManimColor = YELLOW,
37: (8)                                              buff: float = SMALL_BUFF,
38: (8)                                              corner_radius: float = 0.0,
39: (8)                                              **kwargs,
40: (4)                                          ) -> None:
41: (8)                                              super().__init__(
42: (12)                                                 color=color,
43: (12)                                                 width=mobobject.width + 2 * buff,
44: (12)                                                 height=mobobject.height + 2 * buff,
45: (12)                                                 corner_radius=corner_radius,
46: (12)                                                 **kwargs,
47: (8)
48: (8)
49: (8)                                         self.move_to(mobobject)
50: (0)                                         class BackgroundRectangle(SurroundingRectangle):
51: (4)                                         """A background rectangle. Its default color is the background color
52: (4)                                         of the scene.
53: (4)                                         Examples
54: (4)                                         -----
55: (4)                                         .. manim:: ExampleBackgroundRectangle
56: (8)                                         :save_last_frame:
57: (8)                                         class ExampleBackgroundRectangle(Scene):
58: (12)                                         def construct(self):
59: (16)                                             circle = Circle().shift(LEFT)
60: (16)                                             circle.set_stroke(color=GREEN, width=20)
61: (16)                                             triangle = Triangle().shift(2 * RIGHT)
62: (16)                                             triangle.set_fill(PINK, opacity=0.5)
63: (16)                                             backgroundRectangle1 = BackgroundRectangle(circle,
color=WHITE, fill_opacity=0.15)
64: (16)                                             backgroundRectangle2 = BackgroundRectangle(triangle,
color=WHITE, fill_opacity=0.15)
65: (16)
66: (16)
67: (16)
68: (16)
69: (16)
70: (16)                                         self.add(backgroundRectangle1)
71: (4)                                         self.add(backgroundRectangle2)
72: (4)                                         self.add(circle)
73: (8)                                         self.add(triangle)
74: (8)                                         self.play(Rotate(backgroundRectangle1, PI / 4))
75: (8)                                         self.play(Rotate(backgroundRectangle2, PI / 2))
76: (4)
77: (4)
78: (4)
79: (4)
80: (4)
81: (4)
82: (4)
83: (12)
84: (8)
85: (12)
86: (12)
87: (12)
88: (12)
89: (12)
90: (12)
91: (12)
92: (8)
93: (8)
94: (4)
Self:

```

```

95: (8)                     self.set_fill(opacity=b * self.original_fill_opacity)
96: (8)                     return self
97: (4)             def set_style(self, fill_opacity: float, **kwargs) -> Self:
98: (8)                 super().set_style(
99: (12)                     stroke_color=BLACK,
100: (12)                     stroke_width=0,
101: (12)                     fill_color=BLACK,
102: (12)                     fill_opacity=fill_opacity,
103: (8)                 )
104: (8)                 if len(kwargs) > 0:
105: (12)                     logger.info(
106: (16)                         "Argument %s is ignored in BackgroundRectangle.set_style.", 
107: (16)                         kwargs,
108: (12)                     )
109: (8)                     return self
110: (4)             def get_fill_color(self) -> ManimColor:
111: (8)                 return self.color
112: (0)             class Cross(VGroup):
113: (4)                 """Creates a cross.
114: (4)                 Parameters
115: (4)                 -----
116: (4)                 mobject
117: (8)                     The mobject linked to this instance. It fits the mobject when
specified. Defaults to None.
118: (4)                     stroke_color
119: (8)                         Specifies the color of the cross lines. Defaults to RED.
120: (4)                     stroke_width
121: (8)                         Specifies the width of the cross lines. Defaults to 6.
122: (4)                     scale_factor
123: (8)                         Scales the cross to the provided units. Defaults to 1.
124: (4)             Examples
125: (4)             -----
126: (4)             .. manim:: ExampleCross
127: (8)                 :save_last_frame:
128: (8)                 class ExampleCross(Scene):
129: (12)                     def construct(self):
130: (16)                         cross = Cross()
131: (16)                         self.add(cross)
132: (4)             """
133: (4)             def __init__(
134: (8)                 self,
135: (8)                 mobject: Mobject | None = None,
136: (8)                 stroke_color: ParsableManimColor = RED,
137: (8)                 stroke_width: float = 6.0,
138: (8)                 scale_factor: float = 1.0,
139: (8)                 **kwargs,
140: (4)             ) -> None:
141: (8)                 super().__init__(
142: (12)                     Line(UP + LEFT, DOWN + RIGHT), Line(UP + RIGHT, DOWN + LEFT),
**kwargs
143: (8)                     )
144: (8)                     if mobject is not None:
145: (12)                         self.replace(mobject, stretch=True)
146: (8)                         self.scale(scale_factor)
147: (8)                         self.set_stroke(color=stroke_color, width=stroke_width)
148: (0)             class Underline(Line):
149: (4)                 """Creates an underline.
150: (4)                 Examples
151: (4)                 -----
152: (4)                 .. manim:: UnderLine
153: (8)                 :save_last_frame:
154: (8)                 class UnderLine(Scene):
155: (12)                     def construct(self):
156: (16)                         man = Tex("Manim") # Full Word
157: (16)                         ul = Underline(man) # Underlining the word
158: (16)                         self.add(man, ul)
159: (4)             """
160: (4)             def __init__(self, mobject: Mobject, buff: float = SMALL_BUFF, **kwargs) -
> None:

```

```

161: (8)             super().__init__(LEFT, RIGHT, buff=buff, **kwargs)
162: (8)             self.match_width(mobject)
163: (8)             self.next_to(mobject, DOWN, buff=self.buff)
-----
```

## File 72 - coordinate\_systems.py:

```

1: (0)             """Mobjects that represent coordinate systems."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "CoordinateSystem",
5: (4)                 "Axes",
6: (4)                 "ThreeDAxes",
7: (4)                 "NumberPlane",
8: (4)                 "PolarPlane",
9: (4)                 "ComplexPlane",
10: (0)            ]
11: (0)            import fractions as fr
12: (0)            import numbers
13: (0)            from typing import TYPE_CHECKING, Any, Callable, Iterable, Sequence, TypeVar,
overload
14: (0)            import numpy as np
15: (0)            from typing_extensions import Self
16: (0)            from manim import config
17: (0)            from manim.constants import *
18: (0)            from manim.mobject.geometry.arc import Circle, Dot
19: (0)            from manim.mobject.geometry.line import Arrow, DashedLine, Line
20: (0)            from manim.mobject.geometry.polygram import Polygon, Rectangle, RegularPolygon
21: (0)            from manim.mobject.graphing.functions import ImplicitFunction,
ParametricFunction
22: (0)            from manim.mobject.graphing.number_line import NumberLine
23: (0)            from manim.mobject.graphing.scale import LinearBase
24: (0)            from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
25: (0)            from manim.mobject.opengl.opengl_surface import OpenGLSurface
26: (0)            from manim.mobject.text.tex_mobject import MathTex
27: (0)            from manim.mobject.three_d.three_dimensions import Surface
28: (0)            from manim.mobject.types.vectorized_mobject import (
29: (4)                 VDict,
30: (4)                 VectorizedPoint,
31: (4)                 VGroup,
32: (4)                 VМobject,
33: (0)            )
34: (0)            from manim.utils.color import (
35: (4)                 BLACK,
36: (4)                 BLUE,
37: (4)                 BLUE_D,
38: (4)                 GREEN,
39: (4)                 WHITE,
40: (4)                 YELLOW,
41: (4)                 ManimColor,
42: (4)                 ParsableManimColor,
43: (4)                 color_gradient,
44: (4)                 invert_color,
45: (0)            )
46: (0)            from manim.utils.config_ops import merge_dicts_recursively,
update_dict_recursively
47: (0)            from manim.utils.simple_functions import binary_search
48: (0)            from manim.utils.space_ops import angle_of_vector
49: (0)            if TYPE_CHECKING:
50: (4)                 from manim.mobject.mobject import Mobject
51: (4)                 from manim.typing import ManimFloat, Point2D, Point3D, Vector3D
52: (4)                 LineType = TypeVar("LineType", bound=Line)
53: (0)            class CoordinateSystem:
54: (4)                 """Abstract base class for Axes and NumberPlane.
55: (4)                 Examples
56: (4)                 -----
57: (4)                 .. manim:: CoordSysExample
58: (8)                         :save_last_frame:
```

```

59: (8)             class CoordSysExample(Scene):
60: (12)             def construct(self):
61: (16)                 grid = Axes(
62: (20)                     x_range=[0, 1, 0.05], # step size determines
num_decimal_places.
63: (20)                     y_range=[0, 1, 0.05],
64: (20)                     x_length=9,
65: (20)                     y_length=5.5,
66: (20)                     axis_config={
67: (24)                         "numbers_to_include": np.arange(0, 1 + 0.1, 0.1),
68: (24)                         "font_size": 24,
69: (20)                     },
70: (20)                     tips=False,
71: (16)                 )
y_label = grid.get_y_axis_label("y", edge=LEFT,
72: (16)                     x_label = grid.get_x_axis_label("x")
grid_labels = VGroup(x_label, y_label)
graphs = VGroup()
for n in np.arange(1, 20 + 0.5, 0.5):
    graphs += grid.plot(lambda x: x ** n, color=WHITE)
    graphs += grid.plot(
        lambda x: x ** (1 / n), color=WHITE,
    )
graphs += grid.get_horizontal_line(grid.c2p(1, 1, 0),
73: (16)                     graphs += grid.get_vertical_line(grid.c2p(1, 1, 0),
74: (16)                     graphs += Dot(point=grid.c2p(1, 1, 0), color=YELLOW)
75: (16)                     graphs += Tex("(1,1)").scale(0.75).next_to(grid.c2p(1, 1, 0))
76: (16)                     title = Title(
77: (20)                         r"Graphs of $y=x^{\frac{1}{n}}$ and $y=x^n$",
78: (20)                         include_underline=False,
79: (24)                         font_size=40,
80: (20)                     )
81: (16)                     self.add(title, graphs, grid_labels)
82: (16)                     """
83: (16)                     self,
84: (16)                     x_range: Sequence[float] | None = None,
85: (16)                     y_range: Sequence[float] | None = None,
86: (20)                     x_length: float | None = None,
87: (20)                     y_length: float | None = None,
88: (20)                     dimension: int = 2,
89: (16)             ) -> None:
90: (16)                 self.dimension = dimension
91: (4)                 default_step = 1
92: (4)                 if x_range is None:
93: (8)                     x_range = [
94: (8)                         round(-config["frame_x_radius"]),
95: (8)                         round(config["frame_x_radius"]),
96: (8)                         default_step,
97: (8)                     ]
98: (8)                 elif len(x_range) == 2:
99: (12)                     x_range = [*x_range, default_step]
100: (8)                 if y_range is None:
101: (8)                     y_range = [
102: (8)                         round(-config["frame_y_radius"]),
103: (12)                         round(config["frame_y_radius"]),
104: (16)                         default_step,
105: (16)                     ]
106: (16)                 elif len(y_range) == 2:
107: (12)                     y_range = [*y_range, default_step]
108: (8)                 self.x_range = x_range
109: (12)                 self.y_range = y_range
110: (8)                 self.x_length = x_length
111: (12)                 self.y_length = y_length
112: (16)
113: (16)
114: (16)
115: (12)
116: (8)
117: (12)
118: (8)
119: (8)
120: (8)
121: (8)

```

```

122: (8)             self.num_sampled_graph_points_per_tick = 10
123: (4)             def coords_to_point(self, *coords: Sequence[ManimFloat]):
124: (8)                 raise NotImplementedError()
125: (4)             def point_to_coords(self, point: Point3D):
126: (8)                 raise NotImplementedError()
127: (4)             def polar_to_point(self, radius: float, azimuth: float) -> Point2D:
128: (8)                 r"""Gets a point from polar coordinates.
129: (8)                 Parameters
130: (8)
131: (8)                 radius
132: (12)                     The coordinate radius (:math:`r`).
133: (8)                 azimuth
134: (12)                     The coordinate azimuth (:math:`\theta`).
135: (8)             Returns
136: (8)
137: (8)             numpy.ndarray
138: (12)                     The point.
139: (8)             Examples
140: (8)
141: (8)             .. manim:: PolarToPointExample
142: (12)                 :ref_classes: PolarPlane Vector
143: (12)                 :save_last_frame:
144: (12)                 class PolarToPointExample(Scene):
145: (16)                     def construct(self):
146: (20)                         polarplane_pi = PolarPlane(azimuth_units="PI radians",
147: (20)                         size=6)
148: (20)                         polartopoint_vector =
149: (20)                         Vector(polarplane_pi.polar_to_point(3, PI/4))
150: (8)                         self.add(polarplane_pi)
151: (8)                         self.add(polartopoint_vector)
152: (8)                         """
153: (8)                     return self.coords_to_point(radius * np.cos(azimuth), radius *
154: (8)                         np.sin(azimuth))
155: (8)             def point_to_polar(self, point: np.ndarray) -> Point2D:
156: (8)                 r"""Gets polar coordinates from a point.
157: (8)                 Parameters
158: (8)                 point
159: (8)                     The point.
160: (8)             Returns
161: (12)                 Tuple[:class:`float`, :class:`float`]
162: (8)                     The coordinate radius (:math:`r`) and the coordinate azimuth
163: (8)                     (:math:`\theta`).
164: (8)
165: (4)             def c2p(
166: (8)                 self, *coords: float | Sequence[float] | Sequence[Sequence[float]] |
167: (4)             ) -> np.ndarray:
168: (8)                 """Abbreviation for :meth:`coords_to_point`"""
169: (8)                 return self.coords_to_point(*coords)
170: (4)             def p2c(self, point: Point3D):
171: (8)                 """Abbreviation for :meth:`point_to_coords`"""
172: (8)                 return self.point_to_coords(point)
173: (4)             def pr2pt(self, radius: float, azimuth: float) -> np.ndarray:
174: (8)                 """Abbreviation for :meth:`polar_to_point`"""
175: (8)                 return self.polar_to_point(radius, azimuth)
176: (4)             def pt2pr(self, point: np.ndarray) -> tuple[float, float]:
177: (8)                 """Abbreviation for :meth:`point_to_polar`"""
178: (8)                 return self.point_to_polar(point)
179: (4)             def get_axes(self):
180: (8)                 raise NotImplementedError()
181: (4)             def get_axis(self, index: int) -> Mobject:
182: (8)                 return self.get_axes()[index]
183: (4)             def get_origin(self) -> np.ndarray:
184: (8)                 """Gets the origin of :class:`~.Axes`.
185: (8)                 Returns

```

```

186: (8)           -----
187: (8)           np.ndarray
188: (12)          The center point.
189: (8)
190: (8)          return self.coords_to_point(0, 0)
191: (4)           def get_x_axis(self) -> Mobject:
192: (8)           return self.get_axis(0)
193: (4)           def get_y_axis(self) -> Mobject:
194: (8)           return self.get_axis(1)
195: (4)           def get_z_axis(self) -> Mobject:
196: (8)           return self.get_axis(2)
197: (4)           def get_x_unit_size(self) -> float:
198: (8)           return self.get_x_axis().get_unit_size()
199: (4)           def get_y_unit_size(self) -> float:
200: (8)           return self.get_y_axis().get_unit_size()
201: (4)           def get_x_axis_label(
202: (8)             self,
203: (8)             label: float | str | Mobject,
204: (8)             edge: Sequence[float] = UR,
205: (8)             direction: Sequence[float] = UR,
206: (8)             buff: float = SMALL_BUFF,
207: (8)             **kwargs,
208: (4)         ) -> Mobject:
209: (8)             """Generate an x-axis label.
210: (8)             Parameters
211: (8)             -----
212: (8)             label
213: (12)            The label. Defaults to :class:`~.MathTex` for ``str`` and
``float`` inputs.
214: (8)
215: (12)            default ``UR``.
216: (8)
217: (12)            direction
218: (8)              Allows for further positioning of the label from an edge, by
219: (12)            buff
220: (8)              The distance of the label from the line.
221: (8)
222: (8)            Returns
223: (12)            :class:`~.Mobject`
224: (8)              The positioned label.
225: (8)
226: (8)            Examples
227: (12)
228: (12)            .. manim:: GetXAxisLabelExample
229: (16)              :save_last_frame:
230: (20)              class GetXAxisLabelExample(Scene):
231: (20)                def construct(self):
232: (24)                  ax = Axes(x_range=(0, 8), y_range=(0, 5), x_length=8,
y_length=5)
233: (20)                  x_label = ax.get_x_axis_label(
234: (20)                    Tex("$x$-values").scale(0.65), edge=DOWN,
direction=DOWN, buff=0.5
235: (8)
236: (8)
237: (12)                  )
238: (8)
239: (4)                  self.add(ax, x_label)
240: (8)
241: (8)                  return self._get_axis_label(
242: (8)                    label, self.get_x_axis(), edge, direction, buff=buff, **kwargs
243: (8)
244: (8)                  )
245: (8)
246: (4)                  def get_y_axis_label(
247: (8)                    self,
248: (8)                    label: float | str | Mobject,
249: (8)                    edge: Sequence[float] = UR,

```

```

250: (8)           label
251: (12)          The label. Defaults to :class:`~.MathTex` for ``str`` and
``float`` inputs.
252: (8)           edge
253: (12)          The edge of the y-axis to which the label will be added, by
default ``UR``.
254: (8)           direction
255: (12)          Allows for further positioning of the label from an edge, by
default ``UR``.
256: (8)           buff
257: (12)          The distance of the label from the line.
258: (8)           Returns
259: (8)
260: (8)           -----
261: (12)          :class:`~.Mobject`
262: (8)          The positioned label.
263: (8)
264: (8)           Examples
265: (12)
266: (12)
267: (16)
268: (20)
y_length=5)
269: (20)
270: (24)
271: (24)
272: (24)
273: (24)
274: (20)
275: (20)
276: (8)
277: (8)
278: (12)
279: (8)
280: (4)
281: (8)
282: (8)
283: (8)
284: (8)
285: (8)
286: (8)
287: (4)
288: (8)
289: (8)
290: (8)
291: (8)
292: (12)           ``float`` inputs.
293: (8)
294: (12)
295: (8)
296: (12)
adds to the right side of the axis
297: (8)
298: (12)
299: (8)
300: (12)
301: (8)
302: (8)
303: (8)
304: (12)
305: (8)
306: (8)
307: (8)
buff=buff)
308: (8)
309: (8)
310: (4)
311: (8)
.. manim:: GetYAxisLabelExample
:save_last_frame:
class GetYAxisLabelExample(Scene):
    def construct(self):
        ax = Axes(x_range=(0, 8), y_range=(0, 5), x_length=8,
                  y_label = ax.get_y_axis_label(
                      Tex("$y$-values").scale(0.65).rotate(90 * DEGREES),
                      edge=LEFT,
                      direction=LEFT,
                      buff=0.3,
                  )
                  self.add(ax, y_label)
    """
    return self._get_axis_label(
        label, self.get_y_axis(), edge, direction, buff=buff, **kwargs
    )
def _get_axis_label(
    self,
    label: float | str | Mobject,
    axis: Mobject,
    edge: Sequence[float],
    direction: Sequence[float],
    buff: float = SMALL_BUFF,
) -> Mobject:
    """Gets the label for an axis.
    Parameters
    -----
    label
        The label. Defaults to :class:`~.MathTex` for ``str`` and
    axis
        The axis to which the label will be added.
    edge
        The edge of the axes to which the label will be added. ``RIGHT``
    adds to the right side of the axis
    direction
        Allows for further positioning of the label.
    buff
        The distance of the label from the line.
    Returns
    -----
    :class:`~.Mobject`
        The positioned label along the given axis.
    """
    label = self.x_axis._create_label_tex(label)
    label.next_to(axis.get_edge_center(edge), direction=direction,
                  label.shift_onto_screen(buff=MED_SMALL_BUFF)
    return label
def get_axis_labels(self):
    raise NotImplementedError()

```

```

312: (4)             def add_coordinates(
313: (8)             self,
314: (8)             *axes_numbers: Iterable[float] | None | dict[float, str | float |
315: (8)             **kwargs: Any,
316: (4)         ) -> Self:
317: (8)             """Adds labels to the axes. Use ``Axes.coordinate_labels`` to
318: (8)             access the coordinates after creation.
319: (8)             Parameters
320: (8)             -----
321: (8)             axes_numbers
322: (12)             The numbers to be added to the axes. Use ``None`` to represent an
axis with default labels.
323: (8)             Examples
324: (8)             -----
325: (8)             .. code-block:: python
326: (12)             ax = ThreeDAxes()
327: (12)             x_labels = range(-4, 5)
328: (12)             z_labels = range(-4, 4, 2)
329: (12)             ax.add_coordinates(x_labels, None, z_labels) # default y labels,
custom x & z labels
330: (12)             ax.add_coordinates(x_labels) # only x labels
331: (8)             You can also specifically control the position and value of the labels
using a dict.
332: (8)
333: (12)
334: (12)
335: (12)
336: (12) "Saturday", "Sunday"]
337: (12)
338: (8)
339: (8)
340: (8)
341: (12)
342: (8)
343: (12)
344: (16)
345: (16)
346: (12)
347: (16)
348: (16)
349: (20)
axis.scaling.get_custom_labels(tick_range)))
350: (16)
351: (16)
352: (12)
353: (16)
354: (16)
355: (12)
356: (8)
357: (4)
358: (4)
359: (8)
360: (8)
361: (8)
362: (8)
363: (8)
364: (8)
365: (4)
366: (4)
367: (4)
368: (8)
369: (8)
370: (8)
371: (8)
372: (8)
373: (8)
374: (8)
            self.coordinate_labels = VGroup()
            if not axes_numbers:
                axes_numbers = [None for _ in range(self.dimension)]
            for axis, values in zip(self.axes, axes_numbers):
                if isinstance(values, dict):
                    axis.add_labels(values, **kwargs)
                    labels = axis.labels
                elif values is None and axis.scaling.custom_labels:
                    tick_range = axis.get_tick_range()
                    axis.add_labels(
                        dict(zip(tick_range,
axis.scaling.get_custom_labels(tick_range))))
                    )
                    labels = axis.labels
                else:
                    axis.add_numbers(values, **kwargs)
                    labels = axis.numbers
                self.coordinate_labels.add(labels)
            return self
        @overload
        def get_line_from_axis_to_point(
            self,
            index: int,
            point: Sequence[float],
            line_config: dict | None = ...,
            color: ParsableManimColor | None = ...,
            stroke_width: float = ...,
        ) -> DashedLine: ...
        @overload
        def get_line_from_axis_to_point(
            self,
            index: int,
            point: Sequence[float],
            line_func: type[LineType],
            line_config: dict | None = ...,
            color: ParsableManimColor | None = ...,
            stroke_width: float = ...,
        ) -> Line: ...
    
```

```

375: (4)             ) -> LineType: ...
376: (4)         def get_line_from_axis_to_point( # type: ignore[no-untyped-def]
377: (8)             self,
378: (8)             index,
379: (8)             point,
380: (8)             line_func=DashedLine,
381: (8)             line_config=None,
382: (8)             color=None,
383: (8)             stroke_width=2,
384: (4):
385: (8)         """Returns a straight line from a given axis to a point in the scene.
386: (8)         Parameters
387: (8)         -----
388: (8)         index
389: (12)             Specifies the axis from which to draw the line. `0 = x_axis`, `1 =
y_axis`
390: (8)
391: (12)
392: (8)
393: (12)             line.
394: (8)
395: (12)
396: (8)
397: (12)
398: (8)
399: (12)
400: (8)
401: (8)
402: (8)
403: (12)
404: (8)
405: (12)
406: (12)
407: (8)
408: (8)
409: (8)
410: (12)
411: (8)
412: (8)
413: (8)
414: (8)
415: (8)
416: (4)             def get_vertical_line(self, point: Sequence[float], **kwargs: Any) ->
Line:
417: (8)                 """A vertical line from the x-axis to a given point in the scene.
418: (8)                 Parameters
419: (8)                 -----
420: (8)                 point
421: (12)                     The point to which the vertical line will be drawn.
422: (8)
423: (12)                     kwargs
424: (8)                         Additional parameters to be passed to
:class:`get_line_from_axis_to_point`.
425: (8)
426: (8)
427: (12)
428: (8)
429: (8)
430: (8)
431: (12)
432: (12)
433: (16)
434: (20)
435: (20)
436: (20)
437: (20)
{"dashed_ratio": 0.85})
438: (20)

```

```

439: (8)             """
440: (8)         return self.get_line_from_axis_to_point(0, point, **kwargs)
441: (4)     def get_horizontal_line(self, point: Sequence[float], **kwargs) -> Line:
442: (8)         """A horizontal line from the y-axis to a given point in the scene.
443: (8)         Parameters
444: (8)         -----
445: (8)         point
446: (12)             The point to which the horizontal line will be drawn.
447: (8)         kwargs
448: (12)             Additional parameters to be passed to
:class:`get_line_from_axis_to_point`.
449: (8)             Returns
450: (8)             -----
451: (8)             :class:`Line`
452: (12)             A horizontal line from the y-axis to the point.
453: (8)         Examples
454: (8)         -----
455: (8)         .. manim:: GetHorizontalLineExample
456: (12)             :save_last_frame:
457: (12)             class GetHorizontalLineExample(Scene):
458: (16)                 def construct(self):
459: (20)                     ax = Axes().add_coordinates()
460: (20)                     point = ax.c2p(-4, 1.5)
461: (20)                     dot = Dot(point)
462: (20)                     line = ax.get_horizontal_line(point, line_func=Line)
463: (20)                     self.add(ax, line, dot)
464: (8)                 """
465: (8)         return self.get_line_from_axis_to_point(1, point, **kwargs)
466: (4)     def get_lines_to_point(self, point: Sequence[float], **kwargs) -> VGroup:
467: (8)         """Generate both horizontal and vertical lines from the axis to a
point.
468: (8)         Parameters
469: (8)         -----
470: (8)         point
471: (12)             A point on the scene.
472: (8)         kwargs
473: (12)             Additional parameters to be passed to
:meth:`get_line_from_axis_to_point`
474: (8)             Returns
475: (8)             -----
476: (8)             :class:`~.VGroup`
477: (12)             A :class:`~.VGroup` of the horizontal and vertical lines.
478: (8)         .. seealso::
479: (12)             :meth:`~.CoordinateSystem.get_vertical_line`
480: (12)             :meth:`~.CoordinateSystem.get_horizontal_line`
481: (8)         Examples
482: (8)         -----
483: (8)         .. manim:: GetLinesToPointExample
484: (12)             :save_last_frame:
485: (12)             class GetLinesToPointExample(Scene):
486: (16)                 def construct(self):
487: (20)                     ax = Axes()
488: (20)                     circ = Circle(radius=0.5).move_to([-4, -1.5, 0])
489: (20)                     lines_1 = ax.get_lines_to_point(circ.get_right()),
color=GREEN_B)
490: (20)                     lines_2 = ax.get_lines_to_point(circ.get_corner(DL),
color=BLUE_B)
491: (20)                     self.add(ax, lines_1, lines_2, circ)
492: (8)                 """
493: (8)             return VGroup(
494: (12)                 self.get_horizontal_line(point, **kwargs),
495: (12)                 self.get_vertical_line(point, **kwargs),
496: (8)             )
497: (4)         def plot(
498: (8)             self,
499: (8)             function: Callable[[float], float],
500: (8)             x_range: Sequence[float] | None = None,
501: (8)             use_vectorized: bool = False,
502: (8)             **kwargs: Any,

```

```

503: (4)             ) -> ParametricFunction:
504: (8)             """Generates a curve based on a function.
505: (8)             Parameters
506: (8)             -----
507: (8)             function
508: (12)            The function used to construct the :class:`~.ParametricFunction`.
509: (8)             x_range
510: (12)            The range of the curve along the axes. ``x_range = [x_min, x_max,
x_step]``.
511: (8)             use_vectorized
512: (12)            Whether to pass in the generated t value array to the function.
Only use this if your function supports it.
513: (12)            Output should be a numpy array of shape ``[y_0, y_1, ...]``
514: (8)             kwargs
515: (12)            Additional parameters to be passed to
:class:`~.ParametricFunction`.
516: (8)             Returns
517: (8)             -----
518: (8)             :class:`~.ParametricFunction`
519: (12)            The plotted curve.
520: (8)             .. warning::
521: (12)            This method may not produce accurate graphs since Manim currently
relies on interpolation between
522: (12)            evenly-spaced samples of the curve, instead of intelligent
plotting.
523: (12)            See the example below for some solutions to this problem.
Examples
525: (8)            -----
526: (8)            .. manim:: PlotExample
527: (12)            :save_last_frame:
528: (12)            class PlotExample(Scene):
529: (16)            def construct(self):
530: (20)            ax_1 = Axes(
531: (24)                x_range=[0.001, 6],
532: (24)                y_range=[-8, 2],
533: (24)                x_length=5,
534: (24)                y_length=3,
535: (24)                tips=False,
536: (20)
537: (20)            )
538: (20)            ax_2 = ax_1.copy()
539: (20)            ax_3 = ax_1.copy()
540: (20)            ax_1.to_corner(UL)
541: (20)            ax_2.to_corner(UR)
542: (20)            ax_3.to_edge(DOWN)
543: (20)            axes = VGroup(ax_1, ax_2, ax_3)
544: (24)            def log_func(x):
545: (20)                return np.log(x)
546: (20)            curve_1 = ax_1.plot(log_func, color=PURE_RED)
547: (20)            curve_2 = ax_2.plot(log_func, use_smoothing=False,
color=ORANGE)
548: (24)
549: (20)
550: (20)
551: (20)
552: (8)
553: (8)            curves = VGroup(curve_1, curve_2, curve_3)
554: (8)            self.add(axes, curves)
555: (12)
556: (8)
557: (12)
558: (8)
559: (12)
560: (12)
561: (12)
562: (12)
563: (12)
564: (8)
565: (8)
      t_range = np.array(self.x_range, dtype=float)
      if x_range is not None:
          t_range[: len(x_range)] = x_range
      if x_range is None or len(x_range) < 3:
          t_range[2] /= self.num_sampled_graph_points_per_tick
      graph = ParametricFunction(
          lambda t: self.coords_to_point(t, function(t)),
          t_range=t_range,
          scaling=self.x_axis.scaling,
          use_vectorized=use_vectorized,
          **kwargs,
      )
      graph.underlying_function = function

```

```

566: (8)             return graph
567: (4)         def plot_implicit_curve(
568: (8)             self,
569: (8)             func: Callable[[float, float], float],
570: (8)             min_depth: int = 5,
571: (8)             max_quads: int = 1500,
572: (8)             **kwargs: Any,
573: (4)         ) -> ImplicitFunction:
574: (8)             """Creates the curves of an implicit function.
575: (8)             Parameters
576: (8)             -----
577: (8)             func
578: (12)                 The function to graph, in the form of  $f(x, y) = 0$ .
579: (8)             min_depth
580: (12)                 The minimum depth of the function to calculate.
581: (8)             max_quads
582: (12)                 The maximum number of quads to use.
583: (8)             kwargs
584: (12)                 Additional parameters to pass into :class:`ImplicitFunction`.
585: (8)             Examples
586: (8)             -----
587: (8)             .. manim:: ImplicitExample
588: (12)                 :save_last_frame:
589: (12)                 class ImplicitExample(Scene):
590: (16)                     def construct(self):
591: (20)                         ax = Axes()
592: (20)                         a = ax.plot_implicit_curve(
593: (24)                             lambda x, y: y * (x - y) ** 2 - 4 * x - 8, color=BLUE
594: (20)
595: (20)
596: (8)
597: (8)
598: (8)
599: (8)
600: (12)                     graph = ImplicitFunction(
601: (12)                         func=(lambda x, y: func(x_scale.function(x),
y_scale.function(y))),
602: (12)                         x_range=self.x_range[:2],
603: (12)                         y_range=self.y_range[:2],
604: (12)                         min_depth=min_depth,
605: (12)                         max_quads=max_quads,
606: (8)                         **kwargs,
607: (8)
608: (12)                     )
609: (12)                     (
610: (12)                         graph.stretch(self.get_x_unit_size(), 0, about_point=ORIGIN)
611: (8)                         .stretch(self.get_y_unit_size(), 1, about_point=ORIGIN)
612: (8)                         .shift(self.get_origin())
613: (4)                     )
614: (8)
615: (8)
616: (8)
617: (8)
618: (4)         ) -> ParametricFunction:
619: (8)             """A parametric curve.
620: (8)             Parameters
621: (8)             -----
622: (8)             function
623: (12)                 A parametric function mapping a number to a point in the
624: (12)                 coordinate system.
625: (8)             use_vectorized
626: (12)                 Whether to pass in the generated t value array to the function.
Only use this if your function supports it.
627: (8)             kwargs
628: (12)                 Any further keyword arguments are passed to
:class:`.ParametricFunction`.
629: (8)             Example
630: (8)             -----
631: (8)             .. manim:: ParametricCurveExample

```

```

632: (12)           :save_last_frame:
633: (12)           class ParametricCurveExample(Scene):
634: (16)             def construct(self):
635: (20)               ax = Axes()
636: (20)               cardioid = ax.plot_parametric_curve(
637: (24)                 lambda t: np.array([
638: (28)                   [
639: (32)                     np.exp(1) * np.cos(t) * (1 - np.cos(t)),
640: (32)                     np.exp(1) * np.sin(t) * (1 - np.cos(t)),
641: (32)                     0,
642: (28)                   ]
643: (24)                 ),
644: (24)                 t_range=[0, 2 * PI],
645: (24)                 color="#0FF1CE",
646: (20)               )
647: (20)             self.add(ax, cardioid)
648: (8)             """
649: (8)             dim = self.dimension
650: (8)             graph = ParametricFunction(
651: (12)               lambda t: self.coords_to_point(*function(t)[:dim]),
652: (12)               use_vectorized=use_vectorized,
653: (12)               **kwargs,
654: (8)             )
655: (8)             graph.underlying_function = function
656: (8)             return graph
657: (4)             def plot_polar_graph(
658: (8)               self,
659: (8)               r_func: Callable[[float], float],
660: (8)               theta_range: Sequence[float] | None = None,
661: (8)               **kwargs: Any,
662: (4)             ) -> ParametricFunction:
663: (8)             """A polar graph.
664: (8)             Parameters
665: (8)             -----
666: (8)             r_func
667: (12)               The function r of theta.
668: (8)             theta_range
669: (12)               The range of theta as ``theta_range = [theta_min, theta_max,
theta_step]``.
670: (8)             kwargs
671: (12)               Additional parameters passed to :class:`~.ParametricFunction`.
672: (8)             Examples
673: (8)             -----
674: (8)             .. manim:: PolarGraphExample
675: (12)               :ref_classes: PolarPlane
676: (12)               :save_last_frame:
677: (12)               class PolarGraphExample(Scene):
678: (16)                 def construct(self):
679: (20)                   plane = PolarPlane()
680: (20)                   r = lambda theta: 2 * np.sin(theta * 5)
681: (20)                   graph = plane.plot_polar_graph(r, [0, 2 * PI],
color=ORANGE)
682: (20)                   self.add(plane, graph)
683: (8)             """
684: (8)             theta_range = theta_range if theta_range is not None else [0, 2 * PI]
685: (8)             graph = ParametricFunction(
686: (12)               function=lambda th: self.pr2pt(r_func(th), th),
687: (12)               t_range=theta_range,
688: (12)               **kwargs,
689: (8)             )
690: (8)             graph.underlying_function = r_func
691: (8)             return graph
692: (4)             def plot_surface(
693: (8)               self,
694: (8)               function: Callable[[float], float],
695: (8)               u_range: Sequence[float] | None = None,
696: (8)               v_range: Sequence[float] | None = None,
697: (8)               colorscale: (
698: (12)                 Sequence[ParsableManimColor]

```

```

699: (12)                         | Sequence[tuple[ParsableManimColor, float]]
700: (12)                         | None
701: (8)                          ) = None,
702: (8)                          colorscale_axis: int = 2,
703: (8)                          **kwargs: Any,
704: (4)                          ) -> Surface | OpenGLSurface:
705: (8)                          """Generates a surface based on a function.
706: (8)                          Parameters
707: (8)
708: (8)
709: (12)                         function
710: (8)                           The function used to construct the :class:`~.Surface`.
711: (12)                         u_range
712: (8)                           The range of the ``u`` variable: ``(``u_min, u_max````).
713: (12)                         v_range
714: (8)                           The range of the ``v`` variable: ``(``v_min, v_max````).
715: (12)                         colorscale
716: (12)                           Colors of the surface. Passing a list of colors will color the
717: (12)                           surface by z-value.
718: (8)                           user-defined pivots
719: (12)                           2 = z), default
720: (12)                           2 = z-axis (2).
721: (8)                           kwargs
722: (12)                           Additional parameters to be passed to :class:`~.Surface`.
723: (8)                           Returns
724: (8)
725: (8)
726: (12)                         :class:`~.Surface`
727: (8)                           The plotted surface.
728: (8)
729: (8)
730: (12)
731: (12)
732: (16)
733: (20)
734: (20)
DEGREES)
735: (20)
z_range=(-5, 5, 1))
736: (20)
737: (24)
738: (24)
739: (24)
740: (24)
741: (20)
742: (24)
743: (24)
744: (24)
745: (24)
746: (24)
747: (24)
748: (20)
749: (8)
750: (8)
751: (12)
752: (16)
753: (16)
754: (16)
755: (16)
756: (12)
757: (12)
758: (16)
759: (20)
760: (20)
761: (20)
762: (16)

```

| Sequence[tuple[ParsableManimColor, float]]  
| None  
)= None,  
colorscale\_axis: int = 2,  
\*\*kwargs: Any,  
-> Surface | OpenGLSurface:  
"""Generates a surface based on a function.  
Parameters  
-----  
function  
The function used to construct the :class:`~.Surface`.  
u\_range  
The range of the ``u`` variable: ``(``u\_min, u\_max````).  
v\_range  
The range of the ``v`` variable: ``(``v\_min, v\_max````).  
colorscale  
Colors of the surface. Passing a list of colors will color the  
surface by z-value.  
user-defined pivots  
2 = z), default  
2 = z-axis (2).  
kwargs  
Additional parameters to be passed to :class:`~.Surface`.  
Returns  
-----  
:class:`~.Surface`  
The plotted surface.  
Examples  
-----  
. manim:: PlotSurfaceExample  
:save\_last\_frame:  
class PlotSurfaceExample(ThreeDScene):  
 def construct(self):  
 resolution\_fa = 16  
 self.set\_camera\_orientation(phi=75 \* DEGREES, theta=-60 \*  
 axes = ThreeDAxes(x\_range=(-3, 3, 1), y\_range=(-3, 3, 1),  
 def param\_trig(u, v):  
 x = u  
 y = v  
 z = 2 \* np.sin(x) + 2 \* np.cos(y)  
 return z  
 trig\_plane = axes.plot\_surface(  
 param\_trig,  
 resolution=(resolution\_fa, resolution\_fa),  
 u\_range = (-3, 3),  
 v\_range = (-3, 3),  
 colorscale = [BLUE, GREEN, YELLOW, ORANGE, RED],  
 )  
 self.add(axes, trig\_plane)  
 """  
 if config.renderer == RendererType.CAIRO:  
 surface = Surface(  
 lambda u, v: self.c2p(u, v, function(u, v)),  
 u\_range=u\_range,  
 v\_range=v\_range,  
 \*\*kwargs,  
 )  
 if colorscale:  
 surface.set\_fill\_by\_value(  
 axes=self.copy(),  
 colorscale=colorscale,  
 axis=colorscale\_axis,  
 )

```

763: (8)             elif config.renderer == RendererType.OPENGL:
764: (12)                 surface = OpenGLSurface(
765: (16)                     lambda u, v: self.c2p(u, v, function(u, v)),
766: (16)                     u_range=u_range,
767: (16)                     v_range=v_range,
768: (16)                     axes=self.copy(),
769: (16)                     colorscale=colorscale,
770: (16)                     colorscale_axis=colorscale_axis,
771: (16)                     **kwargs,
772: (12)                 )
773: (8)             return surface
774: (4)         def input_to_graph_point(
775: (8)             self,
776: (8)             x: float,
777: (8)             graph: ParametricFunction | VMOBJECT,
778: (4)         ) -> np.ndarray:
779: (8)             """Returns the coordinates of the point on a ``graph`` corresponding
to an ``x`` value.
780: (8)             Parameters
781: (8)             -----
782: (8)
783: (12)                 x
784: (8)                     The x-value of a point on the ``graph``.
785: (12)                 graph
786: (8)                     The :class:`~ParametricFunction` on which the point lies.
787: (8)             Returns
788: (8)             -----
789: (12)                 :class:`np.ndarray`
790: (8)                     The coordinates of the point on the :attr:`graph` corresponding to
the :attr:`x` value.
791: (8)
792: (8)
793: (12)             Raises
794: (8)             -----
795: (8)             :exc:`ValueError`
796: (8)                 When the target x is not in the range of the line graph.
797: (12)             Examples
798: (12)
799: (16)
800: (20)
801: (20)
802: (20)
803: (20)
804: (20)
805: (8)
806: (8)
807: (12)
808: (8)
809: (12)
810: (16)
self.point_to_coords(graph.point_from_proportion(a))[
811: (20)             0
812: (16)             ],
813: (16)             target=x,
814: (16)             lower_bound=0,
815: (16)             upper_bound=1,
816: (12)
817: (12)
818: (16)
819: (12)
820: (16)
821: (20)
([self.p2c(graph.get_start())[0]], {self.p2c(graph.get_end())[0]}),
822: (16)
823: (4)         def input_to_graph_coords(
824: (8)             self, x: float, graph: ParametricFunction
825: (4)         ) -> tuple[float, float]:
826: (8)             """Returns a tuple of the axis relative coordinates of the point
on the graph based on the x-value given.
827: (8)

```

```

828: (8)             Examples
829: (8)
830: (8)
831: (12)          -----
832: (12)          .. code-block:: pycon
833: (12)          >>> from manim import Axes
834: (12)          >>> ax = Axes()
835: (12)          >>> parabola = ax.plot(lambda x: x**2)
836: (12)          >>> ax.input_to_graph_coords(x=3, graph=parabola)
837: (8)          (3, 9)
838: (4)          """
839: (8)          return x, graph.underlying_function(x)
840: (8)      def i2gc(self, x: float, graph: ParametricFunction) -> tuple[float,
841: (4)          float]:
842: (8)              """
843: (8)              Alias for :meth:`input_to_graph_coords`."
844: (8)              return self.input_to_graph_coords(x, graph)
845: (4)      def i2gp(self, x: float, graph: ParametricFunction) -> np.ndarray:
846: (8)          """
847: (8)          Alias for :meth:`input_to_graph_point`."
848: (8)          return self.input_to_graph_point(x, graph)
849: (4)      def get_graph_label(
850: (8)          self,
851: (8)          graph: ParametricFunction,
852: (8)          label: float | str | Mobject = "f(x)",
853: (8)          x_val: float | None = None,
854: (8)          direction: Sequence[float] = RIGHT,
855: (8)          buff: float = MED_SMALL_BUFF,
856: (8)          color: ParsableManimColor | None = None,
857: (8)          dot: bool = False,
858: (8)          dot_config: dict[str, Any] | None = None,
859: (12)      ) -> Mobject:
860: (8)          """
861: (12)          """Creates a properly positioned label for the passed graph, with an
862: (8)          optional dot.
863: (8)          Parameters
864: (8)          -----
865: (12)          graph
866: (8)              The curve.
867: (12)          label
868: (8)              The label for the function's curve. Defaults to :class:`~.MathTex`
869: (12)          for ``str`` and ``float`` inputs.
870: (8)          x_val
871: (12)              The x_value along the curve that positions the label.
872: (8)          direction
873: (12)              The cartesian position, relative to the curve that the label will
874: (8)              be at --> ``LEFT``, ``RIGHT``.
875: (8)          buff
876: (8)              The distance between the curve and the label.
877: (12)          color
878: (8)              The color of the label. Defaults to the color of the curve.
879: (8)          dot
880: (8)              Whether to add a dot at the point on the graph.
881: (12)          dot_config
882: (12)              Additional parameters to be passed to :class:`~.Dot`.
883: (16)          Returns
884: (20)          -----
885: (20)          :class:`Mobject`
886: (20)          The positioned label and :class:`~.Dot`, if applicable.
887: (24)          Examples
888: (24)          -----
889: (24)          .. manim:: GetGraphLabelExample
890: (24)          :save_last_frame:
891: (24)          class GetGraphLabelExample(Scene):
892: (20)          def construct(self):
893: (20)              ax = Axes()
894: (20)              sin = ax.plot(lambda x: np.sin(x), color=PURPLE_B)
895: (20)              label = ax.get_graph_label(
896: (24)                  graph=sin,
897: (24)                  label= MathTex(r"\frac{\pi}{2}"),
898: (24)                  x_val=PI / 2,
899: (24)                  dot=True,
900: (24)                  direction=UR,
901: (24)              )

```

```

893: (20)                         self.add(ax, sin, label)
894: (8)
895: (8)
896: (12)
897: (8)
898: (12)
899: (8)
900: (8)
901: (12)
902: (16)
903: (16)
904: (20)
905: (8)
906: (12)
907: (8)
908: (8)
909: (8)
910: (12)
911: (12)
912: (12)
913: (8)
914: (4) def get_riemann_rectangles(
915: (8)     self,
916: (8)     graph: ParametricFunction,
917: (8)     x_range: Sequence[float] | None = None,
918: (8)     dx: float | None = 0.1,
919: (8)     input_sample_type: str = "left",
920: (8)     stroke_width: float = 1,
921: (8)     stroke_color: ParsableManimColor = BLACK,
922: (8)     fill_opacity: float = 1,
923: (8)     color: Iterable[ParsableManimColor] | ParsableManimColor = (BLUE,
GREEN),
924: (8)
925: (8)
926: (8)
927: (8)
928: (4)
929: (8)
curve.
930: (8)
931: (8)
932: (8)
933: (12) Parameters
934: (8)
935: (12) -----
936: (8) graph
937: (12)     The graph whose area will be approximated by Riemann rectangles.
938: (8) x_range
939: (12)     The minimum and maximum x-values of the rectangles. ``x_range =
where
940: (12) [x_min, x_max]``.
941: (12)
942: (8)
943: (12)
944: (8)
945: (12)
946: (8)
947: (12)
948: (8)
949: (12)
multiple colors are passed.
950: (8)
951: (12) show_signed_area
inverting its color.
952: (8)
953: (12)     Indicates negative area when the curve dips below the x-axis by
rectangles without clear separation.
954: (8)     blend
Sets the :attr:`stroke_color` to :attr:`fill_color`, blending the
boundaries without clear separation.
955: (8)     bounded_graph

```

```

955: (12)                                If a secondary graph is specified, encloses the area between the
two curves.
956: (8)                                 width_scale_factor
957: (12)                                The factor by which the width of the rectangles is scaled.
958: (8)                                 Returns
959: (8)
960: (8)
961: (12)                                :class:`~.VGroup`
962: (8)                                 A :class:`~.VGroup` containing the Riemann Rectangles.
963: (8)
964: (8)
965: (12)
966: (12)                                Examples
967: (16)
968: (20)
969: (20)
970: (20)
971: (24)
972: (24)
973: (24)
974: (24)
975: (24)
976: (20)
977: (20)
978: (24)
979: (20)
980: (20)
981: (24)
982: (20)
983: (20)
984: (24)
985: (24)
986: (24)
987: (24)
988: (24)
989: (24)
990: (20)
991: (20)
992: (24)
993: (20)
994: (8)
995: (8)
996: (12)
997: (16)
998: (12)
999: (16)
1000: (16)
1001: (16)
1002: (8)
1003: (8)
1004: (8)
1005: (8)
1006: (12)
1007: (8)
1008: (12)
1009: (8)
1010: (8)
1011: (12)
1012: (16)
1013: (12)
1014: (16)
1015: (12)
1016: (16)
1017: (12)
1018: (16)
1019: (12)
1020: (12)
1021: (16)

        width_scale_factor
        The factor by which the width of the rectangles is scaled.

    Returns
    ----
    :class:`~.VGroup`
        A :class:`~.VGroup` containing the Riemann Rectangles.

Examples
-----
.. manim:: GetRiemannRectanglesExample
    :save_last_frame:
    class GetRiemannRectanglesExample(Scene):
        def construct(self):
            ax = Axes(y_range=[-2, 10])
            quadratic = ax.plot(lambda x: 0.5 * x ** 2 - 0.5)
            rects_right = ax.get_riemann_rectangles(
                quadratic,
                x_range=[-4, -3],
                dx=0.25,
                color=(TEAL, BLUE_B, DARK_BLUE),
                input_sample_type="right",
            )
            rects_left = ax.get_riemann_rectangles(
                quadratic, x_range=[-1.5, 1.5], dx=0.15, color=YELLOW
            )
            bounding_line = ax.plot(
                lambda x: 1.5 * x, color=BLUE_B, x_range=[3.3, 6]
            )
            bounded_rects = ax.get_riemann_rectangles(
                bounding_line,
                bounded_graph=quadratic,
                dx=0.15,
                x_range=[4, 5],
                show_signed_area=False,
                color=(MAROON_A, RED_B, PURPLE_D),
            )
            self.add(
                ax, bounding_line, quadratic, rects_right, rects_left,
                bounded_rects
            )
        """
        if x_range is None:
            if bounded_graph is None:
                x_range = [graph.t_min, graph.t_max]
            else:
                x_min = max(graph.t_min, bounded_graph.t_min)
                x_max = min(graph.t_max, bounded_graph.t_max)
                x_range = [x_min, x_max]
        x_range = [*x_range[:2], dx]
        rectangles = VGroup()
        x_range = np.arange(*x_range)
        if isinstance(color, (list, tuple)):
            color = [ManimColor(c) for c in color]
        else:
            color = [ManimColor(color)]
        colors = color_gradient(color, len(x_range))
        for x, color in zip(x_range, colors):
            if input_sample_type == "left":
                sample_input = x
            elif input_sample_type == "right":
                sample_input = x + dx
            elif input_sample_type == "center":
                sample_input = x + 0.5 * dx
            else:
                raise ValueError("Invalid input sample type")
            graph_point = self.input_to_graph_point(sample_input, graph)
            if bounded_graph is None:
                y_point = self._origin_shift(self.y_range)

```

```

1022: (12)             else:
1023: (16)                 y_point = bounded_graph.underlying_function(x)
1024: (12)             points = VGroup(
1025: (16)                 *list(
1026: (20)                     map(
1027: (24)                         VectorizedPoint,
1028: (24)                         [
1029: (28)                             self.coords_to_point(x, y_point),
1030: (28)                             self.coords_to_point(x + width_scale_factor * dx,
y_point),
1031: (28)                         graph_point,
1032: (24)                     ],
1033: (20)                 ),
1034: (16)             )
1035: (12)         rect = Rectangle().replace(points, stretch=True)
1036: (12)         rectangles.add(rect)
1037: (12)         if self.p2c(graph_point)[1] < y_point and show_signed_area:
1038: (12)             color = invert_color(color)
1039: (16)         if blend:
1040: (12)             stroke_color = color
1041: (16)         rect.set_style(
1042: (12)             fill_color=color,
1043: (16)             fill_opacity=fill_opacity,
1044: (16)             stroke_color=stroke_color,
1045: (16)             stroke_width=stroke_width,
1046: (16)         )
1047: (12)
1048: (8)     return rectangles
def get_area(
1049: (4)             self,
1050: (8)
1051: (8)             graph: ParametricFunction,
1052: (8)             x_range: tuple[float, float] | None = None,
1053: (8)             color: ParsableManimColor | Iterable[ParsableManimColor] = (BLUE,
GREEN),
1054: (8)
1055: (8)
1056: (8)
1057: (4)
1058: (8)
passed.
1059: (8)             Parameters
1060: (8)             -----
1061: (8)             graph
1062: (12)                 The graph/curve for which the area needs to be gotten.
1063: (8)             x_range
1064: (12)                 The range of the minimum and maximum x-values of the area.
``x_range = [x_min, x_max]``.
1065: (8)
1066: (12)
provided.
1067: (8)
1068: (12)
1069: (8)
1070: (12)
between the two curves.
1071: (8)
1072: (12)
1073: (8)
1074: (8)
1075: (8)
1076: (12)
1077: (8)
1078: (8)
1079: (8)
1080: (12)
or bounded_graph's x_range).
1081: (8)
1082: (8)
1083: (8)             Examples
-----.
.. manim:: GetAreaExample

```

```

1084: (12)             :save_last_frame:
1085: (12)         class GetAreaExample(Scene):
1086: (16)             def construct(self):
1087: (20)                 ax = Axes().add_coordinates()
1088: (20)                 curve = ax.plot(lambda x: 2 * np.sin(x), color=DARK_BLUE)
1089: (20)                 area = ax.get_area(
1090: (24)                     curve,
1091: (24)                     x_range=(PI / 2, 3 * PI / 2),
1092: (24)                     color=(GREEN_B, GREEN_D),
1093: (24)                     opacity=1,
1094: (20)                 )
1095: (20)             self.add(ax, curve, area)
1096: (8)             """
1097: (8)             if x_range is None:
1098: (12)                 a = graph.t_min
1099: (12)                 b = graph.t_max
1100: (8)             else:
1101: (12)                 a, b = x_range
1102: (8)             if bounded_graph is not None:
1103: (12)                 if bounded_graph.t_min > b:
1104: (16)                     raise ValueError(
1105: (20)                         f"Ranges not matching: {bounded_graph.t_min} < {b}",
1106: (16)                     )
1107: (12)                 if bounded_graph.t_max < a:
1108: (16)                     raise ValueError(
1109: (20)                         f"Ranges not matching: {bounded_graph.t_max} > {a}",
1110: (16)                     )
1111: (12)                 a = max(a, bounded_graph.t_min)
1112: (12)                 b = min(b, bounded_graph.t_max)
1113: (8)             if bounded_graph is None:
1114: (12)                 points = (
1115: (16)                     [self.c2p(a), graph.function(a)]
1116: (16)                     + [p for p in graph.points if a <= self.p2c(p)[0] <= b]
1117: (16)                     + [graph.function(b), self.c2p(b)])
1118: (12)             )
1119: (8)             else:
1120: (12)                 graph_points, bounded_graph_points = (
1121: (16)                     [g.function(a)]
1122: (16)                     + [p for p in g.points if a <= self.p2c(p)[0] <= b]
1123: (16)                     + [g.function(b)]
1124: (16)                     for g in (graph, bounded_graph)
1125: (12)             )
1126: (12)                 points = graph_points + bounded_graph_points[::-1]
1127: (8)             return Polygon(*points,
**kwargs).set_opacity(opacity).set_color(color)
1128: (4)             def angle_of_tangent(
1129: (8)                 self,
1130: (8)                 x: float,
1131: (8)                 graph: ParametricFunction,
1132: (8)                 dx: float = 1e-8,
1133: (4)             ) -> float:
1134: (8)                 """Returns the angle to the x-axis of the tangent
1135: (8)                 to the plotted curve at a particular x-value.
1136: (8)                 Parameters
1137: (8)                 -----
1138: (8)                 x
1139: (12)                     The x-value at which the tangent must touch the curve.
1140: (8)             graph
1141: (12)                     The :class:`~.ParametricFunction` for which to calculate the
tangent.
1142: (8)             dx
1143: (12)                     The change in `x` used to determine the angle of the tangent to
the curve.
1144: (8)             Returns
1145: (8)             -----
1146: (8)             :class:`float`
1147: (12)                     The angle of the tangent to the curve.
1148: (8)             Examples
1149: (8)             -----

```

```

1150: (8)          .. code-block:: python
1151: (12)         ax = Axes()
1152: (12)         curve = ax.plot(lambda x: x**2)
1153: (12)         ax.angle_of_tangent(x=3, graph=curve)
1154: (8)         """
1155: (8)         p0 = np.array([*self.input_to_graph_coords(x, graph)])
1156: (8)         p1 = np.array([*self.input_to_graph_coords(x + dx, graph)])
1157: (8)         return angle_of_vector(p1 - p0)
1158: (4) def slope_of_tangent(
1159: (8)     self, x: float, graph: ParametricFunction, **kwargs: Any
1160: (4) ) -> float:
1161: (8)     """Returns the slope of the tangent to the plotted curve
1162: (8)     at a particular x-value.
1163: (8)     Parameters
1164: (8)     -----
1165: (8)     x
1166: (12)         The x-value at which the tangent must touch the curve.
1167: (8)     graph
1168: (12)         The :class:`~.ParametricFunction` for which to calculate the
tangent.
1169: (8)     Returns
1170: (8)     -----
1171: (8)     :class:`float`
1172: (12)         The slope of the tangent with the x axis.
1173: (8)     Examples
1174: (8)     -----
1175: (8)     .. code-block:: python
1176: (12)         ax = Axes()
1177: (12)         curve = ax.plot(lambda x: x**2)
1178: (12)         ax.slope_of_tangent(x=-2, graph=curve)
1179: (8)         """
1180: (8)         return np.tan(self.angle_of_tangent(x, graph, **kwargs))
1181: (4) def plot_derivative_graph(
1182: (8)     self, graph: ParametricFunction, color: ParsableManimColor = GREEN,
**kwargs
1183: (4) ) -> ParametricFunction:
1184: (8)     """Returns the curve of the derivative of the passed graph.
1185: (8)     Parameters
1186: (8)     -----
1187: (8)     graph
1188: (12)         The graph for which the derivative will be found.
1189: (8)     color
1190: (12)         The color of the derivative curve.
1191: (8)     kwargs
1192: (12)         Any valid keyword argument of :class:`~.ParametricFunction`.
1193: (8)     Returns
1194: (8)     -----
1195: (8)     :class:`~.ParametricFunction`
1196: (12)         The curve of the derivative.
1197: (8)     Examples
1198: (8)     -----
1199: (8)     .. manim:: DerivativeGraphExample
1200: (12)         :save_last_frame:
1201: (12)         class DerivativeGraphExample(Scene):
1202: (16)             def construct(self):
1203: (20)                 ax = NumberPlane(y_range=[-1, 7], background_line_style=
{"stroke_opacity": 0.4})
1204: (20)                 curve_1 = ax.plot(lambda x: x ** 2, color=PURPLE_B)
1205: (20)                 curve_2 = ax.plot_derivative_graph(curve_1)
1206: (20)                 curves = VGroup(curve_1, curve_2)
1207: (20)                 label_1 = ax.get_graph_label(curve_1, "x^2", x_val=-2,
direction=DL)
1208: (20)                 label_2 = ax.get_graph_label(curve_2, "2x", x_val=3,
direction=RIGHT)
1209: (20)                 labels = VGroup(label_1, label_2)
1210: (20)                 self.add(ax, curves, labels)
1211: (8)             """
1212: (8)             def deriv(x):
1213: (12)                 return self.slope_of_tangent(x, graph)

```

```

1214: (8)                                return self.plot(deriv, color=color, **kwargs)
1215: (4)                                def plot_antiderivative_graph(
1216: (8)                                self,
1217: (8)                                graph: ParametricFunction,
1218: (8)                                y_intercept: float = 0,
1219: (8)                                samples: int = 50,
1220: (8)                                use_vectorized: bool = False,
1221: (8)                                **kwargs: Any,
1222: (4)                                ) -> ParametricFunction:
1223: (8)                                """Plots an antiderivative graph.
1224: (8)                                Parameters
1225: (8)                                -----
1226: (8)                                graph
1227: (12)                                The graph for which the antiderivative will be found.
1228: (8)                                y_intercept
1229: (12)                                The y-value at which the graph intercepts the y-axis.
1230: (8)                                samples
1231: (12)                                The number of points to take the area under the graph.
1232: (8)                                use_vectorized
1233: (12)                                Whether to use the vectorized version of the antiderivative. This
means
1234: (12)                                to pass in the generated t value array to the function. Only use
this if your function supports it.
1235: (12)                                Output should be a numpy array of shape ``[y_0, y_1, ...]``
1236: (8)                                kwargs
1237: (12)                                Any valid keyword argument of :class:`~.ParametricFunction`.
1238: (8)                                Returns
1239: (8)                                -----
1240: (8)                                :class:`~.ParametricFunction`
1241: (12)                                The curve of the antiderivative.
1242: (8)                                .. note::
1243: (12)                                This graph is plotted from the values of area under the reference
graph.
1244: (12)                                The result might not be ideal if the reference graph contains
uncalculatable
1245: (12)                                areas from x=0.
1246: (8)                                Examples
1247: (8)                                -----
1248: (8)                                .. manim:: AntiderivativeExample
1249: (12)                                :save_last_frame:
1250: (12)                                class AntiderivativeExample(Scene):
1251: (16)                                def construct(self):
1252: (20)                                ax = Axes()
1253: (20)                                graph1 = ax.plot(
1254: (24)                                lambda x: (x ** 2 - 2) / 3,
1255: (24)                                color=RED,
1256: (20)                                )
1257: (20)                                graph2 = ax.plot_antiderivative_graph(graph1, color=BLUE)
1258: (20)                                self.add(ax, graph1, graph2)
1259: (8)                                """
1260: (8)                                def antideriv(x):
1261: (12)                                x_vals = np.linspace(0, x, samples, axis=1 if use_vectorized else
0)
1262: (12)                                f_vec = np.vectorize(graph.underlying_function)
1263: (12)                                y_vals = f_vec(x_vals)
1264: (12)                                return np.trapz(y_vals, x_vals) + y_intercept
1265: (8)                                return self.plot(antideriv, use_vectorized=use_vectorized, **kwargs)
1266: (4)                                def get_secant_slope_group(
1267: (8)                                self,
1268: (8)                                x: float,
1269: (8)                                graph: ParametricFunction,
1270: (8)                                dx: float | None = None,
1271: (8)                                dx_line_color: ParsableManimColor = YELLOW,
1272: (8)                                dy_line_color: ParsableManimColor | None = None,
1273: (8)                                dx_label: float | str | None = None,
1274: (8)                                dy_label: float | str | None = None,
1275: (8)                                include_secant_line: bool = True,
1276: (8)                                secant_line_color: ParsableManimColor = GREEN,
1277: (8)                                secant_line_length: float = 10,

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

1278: (4)             ) -> VGroup:
1279: (8)             """Creates two lines representing `dx` and `df`, the labels for `dx`
and `df`, and
1280: (9)             the secant to the curve at a particular x-value.
1281: (8)             Parameters
1282: (8)             -----
1283: (8)             x
1284: (12)             The x-value at which the secant intersects the graph for the first
time.
1285: (8)
1286: (12)             graph
1287: (8)             The curve for which the secant will be found.
1288: (12)             dx
1289: (8)             The change in `x` after which the secant exits.
1290: (12)             dx_line_color
1291: (8)             The color of the line that indicates the change in `x`.
1292: (12)             dy_line_color
1293: (8)             The color of the line that indicates the change in `y`. Defaults
to the color of :attr:`graph`.
1294: (12)             dx_label
1295: (8)             The label for the `dx` line. Defaults to :class:`~.MathTex` for
``str`` and ``float`` inputs.
1296: (12)             dy_label
1297: (8)             The label for the `dy` line. Defaults to :class:`~.MathTex` for
``str`` and ``float`` inputs.
1298: (12)             include_secant_line
1299: (12)             Whether to include the secant line in the graph,
or just the df/dx lines and labels.
1300: (8)             secant_line_color
1301: (12)             The color of the secant line.
1302: (8)             secant_line_length
1303: (12)             The length of the secant line.
1304: (8)
1305: (8)
1306: (8)
1307: (12)             Returns
1308: (12)
`secant_line`.

1309: (8)             Examples
1310: (8)
1311: (9)             ..
1312: (12)             manim:: GetSecantSlopeGroupExample
1313: (12)             :save_last_frame:
1314: (16)             class GetSecantSlopeGroupExample(Scene):
1315: (20)             def construct(self):
1316: (20)                 ax = Axes(y_range=[-1, 7])
1317: (20)                 graph = ax.plot(lambda x: 1 / 4 * x ** 2, color=BLUE)
1318: (24)                 slopes = ax.get_secant_slope_group(
1319: (24)                     x=2.0,
1320: (24)                     graph=graph,
1321: (24)                     dx=1.0,
1322: (24)                     dx_label=Tex("dx = 1.0"),
1323: (24)                     dy_label="dy",
1324: (24)                     dx_line_color=GREEN_B,
1325: (24)                     secant_line_length=4,
1326: (20)                     secant_line_color=RED_D,
1327: (20)
1328: (8)                     )
1329: (8)                     self.add(ax, graph, slopes)
1330: (8)
1331: (8)
1332: (8)
1333: (8)
1334: (8)
1335: (8)
1336: (8)
1337: (8)
1338: (8)
1339: (8)
1340: (12)

```

```

1341: (12)             labels.add(group.dx_label)
1342: (12)             group.add(group.dx_label)
1343: (8)              if dy_label is not None:
1344: (12)                  group.df_label = self.x_axis._create_label_tex(dy_label)
1345: (12)                  labels.add(group.df_label)
1346: (12)                  group.add(group.df_label)
1347: (8)              if len(labels) > 0:
1348: (12)                  max_width = 0.8 * group.dx_line.width
1349: (12)                  max_height = 0.8 * group.df_line.height
1350: (12)                  if labels.width > max_width:
1351: (16)                      labels.width = max_width
1352: (12)                      if labels.height > max_height:
1353: (16)                          labels.height = max_height
1354: (8)              if dx_label is not None:
1355: (12)                  group.dx_label.next_to(
1356: (16)                      group.dx_line,
1357: (16)                      np.sign(dx) * DOWN,
1358: (16)                      buff=group.dx_label.height / 2,
1359: (12)                  )
1360: (12)                  group.dx_label.set_color(group.dx_line.get_color())
1361: (8)              if dy_label is not None:
1362: (12)                  group.df_label.next_to(
1363: (16)                      group.df_line,
1364: (16)                      np.sign(dx) * RIGHT,
1365: (16)                      buff=group.df_label.height / 2,
1366: (12)                  )
1367: (12)                  group.df_label.set_color(group.df_line.get_color())
1368: (8)              if include_secant_line:
1369: (12)                  group.secant_line = Line(p1, p2, color=secant_line_color)
1370: (12)                  group.secant_line.scale(
1371: (16)                      secant_line_length / group.secant_line.get_length(),
1372: (12)                  )
1373: (12)                  group.add(group.secant_line)
1374: (8)              return group
1375: (4)          def get_vertical_lines_to_graph(
1376: (8)              self,
1377: (8)              graph: ParametricFunction,
1378: (8)              x_range: Sequence[float] | None = None,
1379: (8)              num_lines: int = 20,
1380: (8)              **kwargs: Any,
1381: (4)          ) -> VGroup:
1382: (8)              """Obtains multiple lines from the x-axis to the curve.
1383: (8)              Parameters
1384: (8)              -----
1385: (8)              graph
1386: (12)                  The graph along which the lines are placed.
1387: (8)              x_range
1388: (12)                  A list containing the lower and upper bounds of the lines:
``x_range = [x_min, x_max]``.
1389: (8)              num_lines
1390: (12)                  The number of evenly spaced lines.
1391: (8)              kwargs
1392: (12)                  Additional arguments to be passed to
:meth:`~.CoordinateSystem.get_vertical_line`.
1393: (8)          Returns
1394: (8)          -----
1395: (8)          :class:`~.VGroup`
1396: (12)              The :class:`~.VGroup` of the evenly spaced lines.
1397: (8)          Examples
1398: (8)          -----
1399: (8)          .. manim:: GetVerticalLinesToGraph
1400: (12)              :save_last_frame:
1401: (12)              class GetVerticalLinesToGraph(Scene):
1402: (16)                  def construct(self):
1403: (20)                      ax = Axes(
1404: (24)                          x_range=[0, 8.0, 1],
1405: (24)                          y_range=[-1, 1, 0.2],
1406: (24)                          axis_config={"font_size": 24},
1407: (20)                      ).add_coordinates()

```

```

1408: (20)                               curve = ax.plot(lambda x: np.sin(x) / np.e ** 2 * x)
1409: (20)                               lines = ax.get_vertical_lines_to_graph(
1410: (24)                                   curve, x_range=[0, 4], num_lines=30, color=BLUE
1411: (20)                               )
1412: (20)                               self.add(ax, curve, lines)
1413: (8)                               """
1414: (8)           x_range = x_range if x_range is not None else self.x_range
1415: (8)           return VGroup(
1416: (12)               *(

1417: (16)                   self.get_vertical_line(self.i2gp(x, graph), **kwargs)
1418: (16)                   for x in np.linspace(x_range[0], x_range[1], num_lines)
1419: (12)               )
1420: (8)           )
1421: (4)           def get_T_label(
1422: (8)               self,
1423: (8)               x_val: float,
1424: (8)               graph: ParametricFunction,
1425: (8)               label: float | str | Mobject | None = None,
1426: (8)               label_color: ParsableManimColor | None = None,
1427: (8)               triangle_size: float = MED_SMALL_BUFF,
1428: (8)               triangle_color: ParsableManimColor | None = WHITE,
1429: (8)               line_func: type[Line] = Line,
1430: (8)               line_color: ParsableManimColor = YELLOW,
1431: (4)           ) -> VGroup:
1432: (8)           """Creates a labelled triangle marker with a vertical line from the x-
axis
1433: (8)           to a curve at a given x-value.
1434: (8)           Parameters
1435: (8)           -----
1436: (8)           x_val
1437: (12)               The position along the curve at which the label, line and triangle
will be constructed.
1438: (8)           graph
1439: (12)               The :class:`~.ParametricFunction` for which to construct the
label.
1440: (8)           label
1441: (12)               The label of the vertical line and triangle.
1442: (8)           label_color
1443: (12)               The color of the label.
1444: (8)           triangle_size
1445: (12)               The size of the triangle.
1446: (8)           triangle_color
1447: (12)               The color of the triangle.
1448: (8)           line_func
1449: (12)               The function used to construct the vertical line.
1450: (8)           line_color
1451: (12)               The color of the vertical line.
1452: (8)           Returns
1453: (8)           -----
1454: (8)           :class:`~.VGroup`
1455: (12)               A :class:`~.VGroup` of the label, triangle and vertical line
mobjects.
1456: (8)           Examples
1457: (8)           -----
1458: (8)           .. manim:: TLabelExample
1459: (12)               :save_last_frame:
1460: (12)               class TLabelExample(Scene):
1461: (16)                   def construct(self):
1462: (20)                       axes = Axes(x_range=[-1, 10], y_range=[-1, 10],
x_length=9, y_length=6)
1463: (20)                       func = axes.plot(lambda x: x, color=BLUE)
1464: (20)                       t_label = axes.get_T_label(x_val=4, graph=func,
label=Tex("x-value"))
1465: (20)                           self.add(axes, func, t_label)
1466: (8)                           """
1467: (8)           T_label_group = VGroup()
1468: (8)           triangle = RegularPolygon(n=3, start_angle=np.pi / 2,
stroke_width=0).set_fill(
1469: (12)                           color=triangle_color,

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1470: (12)           opacity=1,
1471: (8)           )
1472: (8)           triangle.height = triangle_size
1473: (8)           triangle.move_to(self.coords_to_point(x_val, 0), UP)
1474: (8)           if label is not None:
1475: (12)             t_label = self.x_axis._create_label_tex(label, color=label_color)
1476: (12)             t_label.next_to(triangle, DOWN)
1477: (12)             T_label_group.add(t_label)
1478: (8)           v_line = self.get_vertical_line(
1479: (12)             self.i2gp(x_val, graph),
1480: (12)             color=line_color,
1481: (12)             line_func=line_func,
1482: (8)           )
1483: (8)           T_label_group.add(triangle, v_line)
1484: (8)           return T_label_group
1485: (0) class Axes(VGroup, CoordinateSystem, metaclass=ConvertToOpenGL):
1486: (4)     """Creates a set of axes.
1487: (4)     Parameters
1488: (4)     -----
1489: (4)     x_range
1490: (8)       The ``(``x_min, x_max, x_step)`` values of the x-axis.
1491: (4)     y_range
1492: (8)       The ``(``y_min, y_max, y_step)`` values of the y-axis.
1493: (4)     x_length
1494: (8)       The length of the x-axis.
1495: (4)     y_length
1496: (8)       The length of the y-axis.
1497: (4)     axis_config
1498: (8)       Arguments to be passed to :class:`~.NumberLine` that influences both
axes.
1499: (4)     x_axis_config
1500: (8)       Arguments to be passed to :class:`~.NumberLine` that influence the x-
axis.
1501: (4)     y_axis_config
1502: (8)       Arguments to be passed to :class:`~.NumberLine` that influence the y-
axis.
1503: (4)     tips
1504: (8)       Whether or not to include the tips on both axes.
1505: (4)     kwargs
1506: (8)       Additional arguments to be passed to :class:`CoordinateSystem` and
:class:`~.VGroup`.
1507: (4) Examples
1508: (4) -----
1509: (4) .. manim:: LogScalingExample
1510: (8)   :save_last_frame:
1511: (8)   class LogScalingExample(Scene):
1512: (12)     def construct(self):
1513: (16)       ax = Axes(
1514: (20)         x_range=[0, 10, 1],
1515: (20)         y_range=[-2, 6, 1],
1516: (20)         tips=False,
1517: (20)         axis_config={"include_numbers": True},
1518: (20)         y_axis_config={"scaling": LogBase(custom_labels=True)},
1519: (16)
1520: (16)
1521: (16)       use_smoothing=False)
1522: (16)       self.add(ax, graph)
1523: (4) Styling arguments can be passed to the underlying :class:`.NumberLine` 
mobjects that represent the axes:
1524: (4) .. manim:: AxesWithDifferentTips
1525: (8)   :save_last_frame:
1526: (8)   class AxesWithDifferentTips(Scene):
1527: (12)     def construct(self):
1528: (16)       ax = Axes(axis_config={'tip_shape': StealthTip})
1529: (16)       self.add(ax)
1530: (4) """
1531: (4)     def __init__(
1532: (8)       self,
1533: (8)       x_range: Sequence[float] | None = None,

```

```

1534: (8)             y_range: Sequence[float] | None = None,
1535: (8)             x_length: float | None = round(config.frame_width) - 2,
1536: (8)             y_length: float | None = round(config.frame_height) - 2,
1537: (8)             axis_config: dict | None = None,
1538: (8)             x_axis_config: dict | None = None,
1539: (8)             y_axis_config: dict | None = None,
1540: (8)             tips: bool = True,
1541: (8)             **kwargs: Any,
1542: (4)         ) -> None:
1543: (8)             VGroup.__init__(self, **kwargs)
1544: (8)             CoordinateSystem.__init__(self, x_range, y_range, x_length, y_length)
1545: (8)             self.axis_config = {
1546: (12)                 "include_tip": tips,
1547: (12)                 "numbers_to_exclude": [0],
1548: (8)             }
1549: (8)             self.x_axis_config = {}
1550: (8)             self.y_axis_config = {"rotation": 90 * DEGREES, "label_direction":
LEFT}
1551: (8)             self._update_default_configs(
1552: (12)                 (self.axis_config, self.x_axis_config, self.y_axis_config),
1553: (12)                 (axis_config, x_axis_config, y_axis_config),
1554: (8)             )
1555: (8)             self.x_axis_config = merge_dicts_recursively(
1556: (12)                 self.axis_config,
1557: (12)                 self.x_axis_config,
1558: (8)             )
1559: (8)             self.y_axis_config = merge_dicts_recursively(
1560: (12)                 self.axis_config,
1561: (12)                 self.y_axis_config,
1562: (8)             )
1563: (8)             if self.x_axis_config.get("scaling") is None or isinstance(
1564: (12)                 self.x_axis_config.get("scaling"), LinearBase
1565: (8)             ):
1566: (12)                 self.x_axis_config["exclude_origin_tick"] = True
1567: (8)             else:
1568: (12)                 self.x_axis_config["exclude_origin_tick"] = False
1569: (8)             if self.y_axis_config.get("scaling") is None or isinstance(
1570: (12)                 self.y_axis_config.get("scaling"), LinearBase
1571: (8)             ):
1572: (12)                 self.y_axis_config["exclude_origin_tick"] = True
1573: (8)             else:
1574: (12)                 self.y_axis_config["exclude_origin_tick"] = False
1575: (8)             self.x_axis = self._create_axis(self.x_range, self.x_axis_config,
1576: (8)             self.y_axis = self._create_axis(self.y_range, self.y_axis_config,
1577: (8)             self.axes = VGroup(self.x_axis, self.y_axis)
1578: (8)             self.add(*self.axes)
1579: (8)             lines_center_point = [
1580: (12)                 axis.scaling.function((axis.x_range[1] + axis.x_range[0]) / 2)
1581: (12)                 for axis in self.axes
1582: (8)             ]
1583: (8)             self.shift(-self.coords_to_point(*lines_center_point))
1584: (4)         @staticmethod
1585: (4)         def _update_default_configs(
1586: (8)             default_configs: tuple[dict[Any, Any]], passed_configs:
tuple[dict[Any, Any]]
1587: (4)         ) -> None:
1588: (8)             """Takes in two tuples of dicts and return modifies the first such
that values from
1589: (8)             ``passed_configs`` overwrite values in ``default_configs``. If a key
does not exist
1590: (8)             in default_configs, it is added to the dict.
1591: (8)             This method is useful for having defaults in a class and being able to
overwrite
1592: (8)             them with user-defined input.
1593: (8)             Parameters
1594: (8)             -----
1595: (8)             default_configs

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

1596: (12)             The dict that will be updated.
1597: (8)              passed_configs
1598: (12)             The dict that will be used to update.
1599: (8)              Examples
1600: (8)              -----
1601: (8)             To create a tuple with one dictionary, add a comma after the element:
1602: (8)             .. code-block:: python
1603: (12)             self._update_default_configs(
1604: (16)               (dict_1,)(
1605: (20)                 dict_2,
1606: (16)               )
1607: (12)             )
1608: (8)             """
1609: (8)             for default_config, passed_config in zip(default_configs,
passed_configs):
1610: (12)               if passed_config is not None:
1611: (16)                 update_dict_recursively(default_config, passed_config)
1612: (4)             def _create_axis(
1613: (8)               self,
1614: (8)               range_terms: Sequence[float],
1615: (8)               axis_config: dict[str, Any],
1616: (8)               length: float,
1617: (4)             ) -> NumberLine:
1618: (8)               """Creates an axis and dynamically adjusts its position depending on
where 0 is located on the line.
1619: (8)             Parameters
1620: (8)             -----
1621: (8)             range_terms
1622: (12)               The range of the the axis : ````(x_min, x_max, x_step)````.
1623: (8)             axis_config
1624: (12)               Additional parameters that are passed to :class:`~.NumberLine` .
1625: (8)             length
1626: (12)               The length of the axis.
1627: (8)             Returns
1628: (8)             -----
1629: (8)             :class:`NumberLine`
1630: (12)               Returns a number line based on ``range_terms``.
1631: (8)             """
1632: (8)             axis_config["length"] = length
1633: (8)             axis = NumberLine(range_terms, **axis_config)
1634: (8)             axis.shift(-axis.number_to_point(self._origin_shift([axis.x_min,
axis.x_max])))
1635: (8)             return axis
1636: (4)             def coords_to_point(
1637: (8)               self, *coords: float | Sequence[float] | Sequence[Sequence[float]] |
1638: (4)             ) -> np.ndarray:
1639: (8)               """Accepts coordinates from the axes and returns a point with respect
to the scene.
1640: (8)             Parameters
1641: (8)             -----
1642: (8)             coords
1643: (12)               The coordinates. Each coord is passed as a separate argument:
````ax.coords_to_point(1, 2, 3)````.
1644: (12)               Also accepts a list of coordinates
1645: (12)               ````ax.coords_to_point( [x_0, x_1, ...], [y_0, y_1, ...], ... )````
```

$$\text{````ax.coords\_to\_point( [x}_0, x_1, \dots], [y}_0, y_1, \dots], \dots )````}$$

$$\text{````ax.coords\_to\_point( [[x}_0, y_0, z_0], [x}_1, y_1, z_1] )````}$$

```

1646: (12)             Returns
1647: (8)             -----
1648: (8)             np.ndarray
1649: (8)               A point with respect to the scene's coordinate system.
1650: (12)               The shape of the array will be similar to the shape of the input.
1651: (12)             Examples
1652: (8)             -----
1653: (8)             .. code-block:: pycon
1654: (8)               >>> from manim import Axes
1655: (12)               >>> import numpy as np
1656: (12)               >>> ax = Axes()
1657: (12)               >>> np.around(ax.coords_to_point(1, 0, 0), 2)
1658: (12)
```

```

1659: (12)             array([0.86, 0. , 0. ])
1660: (12)             >>> np.around(ax.coords_to_point([[0, 1], [1, 1], [1, 0]]), 2)
1661: (12)             array([[0. , 0.75, 0. ],
1662: (19)                 [0.86, 0.75, 0. ],
1663: (19)                 [0.86, 0. , 0. ]])
1664: (12)             >>> np.around(
1665: (12)             ...     ax.coords_to_point([0, 1, 1], [1, 1, 0]), 2
1666: (12)             ... ) # Transposed version of the above
1667: (12)             array([[0. , 0.86, 0.86],
1668: (19)                 [0.75, 0.75, 0. ],
1669: (19)                 [0. , 0. , 0. ]])
1670: (8) .. manim:: CoordsToPointExample
1671: (12) :save_last_frame:
1672: (12) class CoordsToPointExample(Scene):
1673: (16)     def construct(self):
1674: (20)         ax = Axes().add_coordinates()
1675: (20)         dot_axes = Dot(ax.coords_to_point(2, 2), color=GREEN)
1676: (20)         lines = ax.get_lines_to_point(ax.c2p(2,2))
1677: (20)         plane = NumberPlane()
1678: (20)         dot_scene = Dot((2,2,0), color=RED)
1679: (20)         self.add(plane, dot_scene, ax, dot_axes, lines)
1680: (8)     """
1681: (8)     coords = np.asarray(coords)
1682: (8)     origin = self.x_axis.number_to_point(
1683: (12)         self._origin_shift([self.x_axis.x_min, self.x_axis.x_max]),
1684: (8)     )
1685: (8)     are_coordinates_transposed = False
1686: (8)     if coords.ndim == 3:
1687: (12)         coords = coords[0]
1688: (12)         if coords.shape[0] == 1:
1689: (16)             coords = coords[0]
1690: (12)         else:
1691: (16)             coords = coords.T
1692: (16)             are_coordinates_transposed = True
1693: (8)     points = self.x_axis.number_to_point(coords[0])
1694: (8)     other_axes = self.axes.submobjects[1:]
1695: (8)     for axis, nums in zip(other_axes, coords[1:]):
1696: (12)         points += axis.number_to_point(nums) - origin
1697: (8)     if are_coordinates_transposed or points.ndim == 1:
1698: (12)         return points
1699: (8)     return points.T
1700: (4) def point_to_coords(self, point: Sequence[float]) -> np.ndarray:
1701: (8) """Accepts a point from the scene and returns its coordinates with
respect to the axes.
1702: (8) Parameters
1703: (8) -----
1704: (8) point
1705: (12)     The point, i.e. ``RIGHT`` or ``[0, 1, 0]``.
1706: (12)     Also accepts a list of points as ``[RIGHT, [0, 1, 0]]``.
1707: (8) Returns
1708: (8) -----
1709: (8) np.ndarray[float]
1710: (12)     The coordinates on the axes, i.e. ``[4.0, 7.0]``.
1711: (12)     Or a list of coordinates if `point` is a list of points.
1712: (8) Examples
1713: (8) -----
1714: (8) .. code-block:: pycon
1715: (12)     >>> from manim import Axes, RIGHT
1716: (12)     >>> import numpy as np
1717: (12)     >>> ax = Axes(x_range=[0, 10, 2])
1718: (12)     >>> np.around(ax.point_to_coords(RIGHT), 2)
1719: (12)     array([5.83, 0. ])
1720: (12)     >>> np.around(ax.point_to_coords([[0, 0, 1], [1, 0, 0]]), 2)
1721: (12)     array([[5. , 0. ],
1722: (19)                 [5.83, 0. ]])
1723: (8) .. manim:: PointToCoordsExample
1724: (12) :save_last_frame:
1725: (12) class PointToCoordsExample(Scene):
1726: (16)     def construct(self):

```

```

1727: (20)                     ax = Axes(x_range=[0, 10, 2]).add_coordinates()
1728: (20)                     circ = Circle(radius=0.5).shift(UR * 2)
1729: (20)                     coords = np.around(ax.point_to_coords(circ.get_right())),
1730: (20)                     decimals=2)
1731: (24)                     label = (
1732: (20)                         Matrix([[coords[0]]],
1733: (20)                         [coords[1]]]).scale(0.75).next_to(circ, RIGHT)
1734: (8)                     )
1735: (8)                     self.add(ax, circ, label, Dot(circ.get_right()))
1736: (8)                     """
1737: (8)                     point = np.asarray(point)
1738: (12)                     result = np.asarray([axis.point_to_number(point) for axis in
1739: (8)                     self.get_axes()])
1740: (4)                     if point.ndim == 2:
1741: (8)                         return result.T
1742: (8)                     return result
1743: (8)                     def get_axes(self) -> VGroup:
1744: (8)                         """Gets the axes.
1745: (12)                         Returns
1746: (8)                         -----
1747: (8)                         :class:`~.VGroup`
1748: (4)                         A pair of axes.
1749: (8)                         """
1750: (8)                     return self.axes
1751: (8)                     def get_axis_labels(
1752: (4)                     self,
1753: (8)                         x_label: float | str | Mobject = "x",
1754: (8)                         y_label: float | str | Mobject = "y",
1755: (8)                     ) -> VGroup:
1756: (8)                         """Defines labels for the x-axis and y-axis of the graph.
1757: (8)                         For increased control over the position of the labels,
1758: (8)                         use :meth:`~.CoordinateSystem.get_x_axis_label` and
1759: (8)                         :meth:`~.CoordinateSystem.get_y_axis_label`.
1760: (12)                         Parameters
1761: (8)                         -----
1762: (12)                         x_label
1763: (8)                             The label for the x_axis. Defaults to :class:`~.MathTex` for
1764: (8)                             ``str`` and ``float`` inputs.
1765: (8)                         y_label
1766: (12)                             The label for the y_axis. Defaults to :class:`~.MathTex` for
1767: (8)                             ``str`` and ``float`` inputs.
1768: (8)                         Returns
1769: (8)                         -----
1770: (8)                         :class:`~.VGroup`
1771: (8)                         A :class:`~.VGroup` of the labels for the x_axis and y_axis.
1772: (8)                         .. seealso::
1773: (12)                             :meth:`~.CoordinateSystem.get_x_axis_label`
1774: (12)                             :meth:`~.CoordinateSystem.get_y_axis_label`
1775: (16)                         Examples
1776: (20)                         -----
1777: (20)                         .. manim:: GetAxisLabelsExample
1778: (24)                             :save_last_frame:
1779: (20)                             class GetAxisLabelsExample(Scene):
1780: (20)                                 def construct(self):
1781: (8)                                     ax = Axes()
1782: (8)                                     labels = ax.get_axis_labels(
1783: (12)   Tex("x-axis").scale(0.7), Tex("y-axis").scale(0.45)
1784: (12)   )
1785: (8)                                     self.add(ax, labels)
1786: (8)                                     """
1787: (4)                                     self.axis_labels = VGroup(
1788: (8)   self.get_x_axis_label(x_label),
1789: (8)   self.get_y_axis_label(y_label),
1790: (8)   )
1791: (8)                                     return self.axis_labels
1792: (4)                     def plot_line_graph(
1793: (8)                     self,
1794: (8)                         x_values: Iterable[float],
1795: (8)                         y_values: Iterable[float],
1796: (8)                         
```

```

1791: (8)             z_values: Iterable[float] | None = None,
1792: (8)             line_color: ParsableManimColor = YELLOW,
1793: (8)             add_vertex_dots: bool = True,
1794: (8)             vertex_dot_radius: float = DEFAULT_DOT_RADIUS,
1795: (8)             vertex_dot_style: dict[str, Any] | None = None,
1796: (8)             **kwargs: Any,
1797: (4)         ) -> VDict:
1798: (8)             """Draws a line graph.
1799: (8)             The graph connects the vertices formed from zipping
1800: (8)             ``x_values``, ``y_values`` and ``z_values``. Also adds :class:`Dots
<.Dot>` at the
1801: (8)             vertices if ``add_vertex_dots`` is set to ``True``.
1802: (8)             Parameters
1803: (8)             -----
1804: (8)             x_values
1805: (12)             Iterable of values along the x-axis.
1806: (8)             y_values
1807: (12)             Iterable of values along the y-axis.
1808: (8)             z_values
1809: (12)             Iterable of values (zeros if z_values is None) along the z-axis.
1810: (8)             line_color
1811: (12)             Color for the line graph.
1812: (8)             add_vertex_dots
1813: (12)             Whether or not to add :class:`~.Dot` at each vertex.
1814: (8)             vertex_dot_radius
1815: (12)             Radius for the :class:`~.Dot` at each vertex.
1816: (8)             vertex_dot_style
1817: (12)             Style arguments to be passed into :class:`~.Dot` at each vertex.
1818: (8)             kwargs
1819: (12)             Additional arguments to be passed into :class:`~.VMobject`.
1820: (8)             Returns
1821: (8)             -----
1822: (8)             :class:`~.VDict`
1823: (12)             A VDict containing both the line and dots (if specified). The line
can be accessed with: ``line_graph["line_graph"]``.
1824: (12)             The dots can be accessed with: ``line_graph["vertex_dots"]``.
1825: (8)             Examples
1826: (8)             -----
1827: (8)             .. manim:: LineGraphExample
1828: (12)             :save_last_frame:
1829: (12)             class LineGraphExample(Scene):
1830: (16)             def construct(self):
1831: (20)                 plane = NumberPlane(
1832: (24)                     x_range = (0, 7),
1833: (24)                     y_range = (0, 5),
1834: (24)                     x_length = 7,
1835: (24)                     axis_config={"include_numbers": True},
1836: (20)
1837: (20)
1838: (20)
1839: (24)
1840: (24)
1841: (24)
1842: (24)
fill_color=PURPLE),
1843: (24)
1844: (20)
1845: (20)
1846: (8)
1847: (8)
1848: (8)
1849: (12)
1850: (8)
1851: (8)
1852: (8)
1853: (12)
1854: (12)
1855: (8)
1856: (8)
             x_values, y_values = map(np.array, (x_values, y_values))
             if z_values is None:
                 z_values = np.zeros(x_values.shape)
             line_graph = VDict()
             graph = VGroup(color=line_color, **kwargs)
             vertices = [
                 self.coords_to_point(x, y, z)
                 for x, y, z in zip(x_values, y_values, z_values)
             ]
             graph.set_points_as_corners(vertices)

```

```

1857: (8)             line_graph["line_graph"] = graph
1858: (8)             if add_vertex_dots:
1859: (12)                 vertex_dot_style = vertex_dot_style or {}
1860: (12)                 vertex_dots = VGroup(
1861: (16)                     *(
1862: (20)                         Dot(point=vertex, radius=vertex_dot_radius,
1863: (20)                         for vertex in vertices
1864: (16)
1865: (12)
1866: (12)                     )
1867: (8)                 line_graph["vertex_dots"] = vertex_dots
1868: (8)             return line_graph
1869: (4) @staticmethod
1870: (4)             def _origin_shift(axis_range: Sequence[float]) -> float:
1871: (8)                 """Determines how to shift graph mobjects to compensate when 0 is not
1872: (8)                 on the axis.
1873: (8)             Parameters
1874: (8)                 -----
1875: (8)                 axis_range
1876: (8)                     The range of the axis : `` $(x_{\min}, x_{\max}, x_{\text{step}})$ ``.
1877: (8)                     """
1878: (8)                     if axis_range[0] > 0:
1879: (12)                         return axis_range[0]
1880: (8)                     if axis_range[1] < 0:
1881: (12)                         return axis_range[1]
1882: (8)                     else:
1883: (12)                         return 0
1884: (0) class ThreeDAxes(Axes):
1885: (4)                 """A 3-dimensional set of axes.
1886: (4)             Parameters
1887: (4)                 -----
1888: (4)                 x_range
1889: (8)                     The `` $[x_{\min}, x_{\max}, x_{\text{step}}]$ `` values of the x-axis.
1890: (4)                 y_range
1891: (8)                     The `` $[y_{\min}, y_{\max}, y_{\text{step}}]$ `` values of the y-axis.
1892: (4)                 z_range
1893: (8)                     The `` $[z_{\min}, z_{\max}, z_{\text{step}}]$ `` values of the z-axis.
1894: (4)                 x_length
1895: (8)                     The length of the x-axis.
1896: (4)                 y_length
1897: (8)                     The length of the y-axis.
1898: (4)                 z_length
1899: (8)                     The length of the z-axis.
1900: (4)                 z_axis_config
1901: (8)                     Arguments to be passed to :class:`~.NumberLine` that influence the z-
axis.
1902: (4)                 z_normal
1903: (8)                     The direction of the normal.
1904: (4)                 num_axis_pieces
1905: (8)                     The number of pieces used to construct the axes.
1906: (4)                 light_source
1907: (8)                     The direction of the light source.
1908: (4)                 depth
1909: (8)                     Currently non-functional.
1910: (4)                 gloss
1911: (8)                     Currently non-functional.
1912: (4)                 kwargs
1913: (4)                     Additional arguments to be passed to :class:`Axes`.
1914: (8)             """
1915: (4)             def __init__(
1916: (8)                 self,
1917: (8)                 x_range: Sequence[float] | None = (-6, 6, 1),
1918: (8)                 y_range: Sequence[float] | None = (-5, 5, 1),
1919: (8)                 z_range: Sequence[float] | None = (-4, 4, 1),
1920: (8)                 x_length: float | None = config.frame_height + 2.5,
1921: (8)                 y_length: float | None = config.frame_height + 2.5,
1922: (8)                 z_length: float | None = config.frame_height - 1.5,
1923: (8)                 z_axis_config: dict[str, Any] | None = None,
1924: (8)                 z_normal: Vector3D = DOWN,
```

```

1923: (8)             num_axis_pieces: int = 20,
1924: (8)             light_source: Sequence[float] = 9 * DOWN + 7 * LEFT + 10 * OUT,
1925: (8)             depth=None,
1926: (8)             gloss=0.5,
1927: (8)             **kwargs: dict[str, Any],
1928: (4)             ) -> None:
1929: (8)             super().__init__(
1930: (12)                 x_range=x_range,
1931: (12)                 x_length=x_length,
1932: (12)                 y_range=y_range,
1933: (12)                 y_length=y_length,
1934: (12)                 **kwargs,
1935: (8)
1936: (8)             )
1937: (8)             self.z_range = z_range
1938: (8)             self.z_length = z_length
1939: (8)             self.z_axis_config = {}
1940: (8)             self._update_default_configs((self.z_axis_config,), (z_axis_config,))
1941: (12)             self.z_axis_config = merge_dicts_recursively(
1942: (12)                 self.axis_config,
1943: (8)                 self.z_axis_config,
1944: (8)             )
1945: (8)             self.z_normal = z_normal
1946: (8)             self.num_axis_pieces = num_axis_pieces
1947: (8)             self.light_source = light_source
1948: (8)             self.dimension = 3
1949: (12)             if self.z_axis_config.get("scaling") is None or isinstance(
1950: (8)                 self.z_axis_config.get("scaling"), LinearBase
1951: (12)             ):
1952: (8)                 self.z_axis_config["exclude_origin_tick"] = True
1953: (8)             else:
1954: (8)                 self.z_axis_config["exclude_origin_tick"] = False
1955: (8)             z_axis = self._create_axis(self.z_range, self.z_axis_config,
1956: (8)             self.z_length)
1957: (8)             z_origin = self._origin_shift([z_axis.x_min, z_axis.x_max])
1958: (8)             z_axis.rotate_about_number(z_origin, -PI / 2, UP)
1959: (8)             z_axis.rotate_about_number(z_origin, angle_of_vector(self.z_normal))
1960: (12)             z_axis.shift(-z_axis.number_to_point(z_origin))
1961: (16)             z_axis.shift(
1962: (12)                 self.x_axis.number_to_point(
1963: (8)                     self._origin_shift([self.x_axis.x_min, self.x_axis.x_max]),
1964: (8)                 ),
1965: (8)
1966: (8)             self.axes.add(z_axis)
1967: (8)             self.add(z_axis)
1968: (12)             self.z_axis = z_axis
1969: (12)             if config.renderer == RendererType.CAIRO:
1970: (4)             self._add_3d_pieces()
1971: (8)             self._set_axis_shading()
def _add_3d_pieces(self) -> None:
1972: (12)             for axis in self.axes:
1973: (12)                 axis.pieces = VGroup(*axis.get_pieces(self.num_axis_pieces))
1974: (12)                 axis.add(axis.pieces)
1975: (12)                 axis.set_stroke(width=0, family=False)
1976: (4)                 axis.set_shade_in_3d(True)
def _set_axis_shading(self) -> None:
1977: (8)                 def make_func(axis):
1978: (12)                     vect = self.light_source
1979: (12)                     return lambda:
1980: (16)                         axis.get_edge_center(-vect),
1981: (16)                         axis.get_edge_center(vect),
1982: (12)                     )
1983: (8)                 for axis in self:
1984: (12)                     for submob in axis.family_members_with_points():
1985: (16)                         submob.get_gradient_start_and_end_points = make_func(axis)
1986: (16)                         submob.get_unit_normal = lambda a: np.ones(3)
1987: (16)                         submob.set_sheen(0.2)
def get_y_axis_label(
1988: (4)                 self,
1989: (8)                 label: float | str | Mobject,
1990: (8)

```

```

1991: (8)             edge: Sequence[float] = UR,
1992: (8)             direction: Sequence[float] = UR,
1993: (8)             buff: float = SMALL_BUFF,
1994: (8)             rotation: float = PI / 2,
1995: (8)             rotation_axis: Vector3D = OUT,
1996: (8)             **kwargs,
1997: (4)         ) -> Mobject:
1998: (8)             """Generate a y-axis label.
1999: (8)             Parameters
2000: (8)             -----
2001: (8)             label
2002: (12)             The label. Defaults to :class:`~.MathTex` for ``str`` and
``float`` inputs.
2003: (8)             edge
2004: (12)             The edge of the y-axis to which the label will be added, by
default ``UR``.
2005: (8)             direction
2006: (12)             Allows for further positioning of the label from an edge, by
default ``UR``.
2007: (8)             buff
2008: (12)             The distance of the label from the line, by default
``SMALL_BUFF``.
2009: (8)             rotation
2010: (12)             The angle at which to rotate the label, by default ``PI/2``.
2011: (8)             rotation_axis
2012: (12)             The axis about which to rotate the label, by default ``OUT``.
2013: (8)
2014: (8)
2015: (8)         Returns
2016: (12)             :class:`~.Mobject`
2017: (8)             The positioned label.
2018: (8)
2019: (8)         Examples
2020: (12)             .. manim:: GetYAxisLabelExample
2021: (12)             :save_last_frame:
2022: (16)             class GetYAxisLabelExample(ThreeDScene):
2023: (20)                 def construct(self):
2024: (20)                     ax = ThreeDAxes()
2025: (20)                     lab = ax.get_y_axis_label(Tex("$y$-label"))
2026: (20)                     self.set_camera_orientation(phi=2*PI/5, theta=PI/5)
2027: (8)                     self.add(ax, lab)
2028: (8)             """
2029: (12)             positioned_label = self._get_axis_label(
2030: (8)                 label, self.get_y_axis(), edge, direction, buff=buff, **kwargs
2031: (8)             )
2032: (8)             positioned_label.rotate(rotation, axis=rotation_axis)
2033: (4)             return positioned_label
2034: (8)
2035: (8)         def get_z_axis_label(
2036: (8)             self,
2037: (8)             label: float | str | Mobject,
2038: (8)             edge: Vector3D = OUT,
2039: (8)             direction: Vector3D = RIGHT,
2040: (8)             buff: float = SMALL_BUFF,
2041: (8)             rotation: float = PI / 2,
2042: (8)             rotation_axis: Vector3D = RIGHT,
2043: (8)             **kwargs: Any,
2044: (8)         ) -> Mobject:
2045: (8)             """Generate a z-axis label.
2046: (8)             Parameters
2047: (12)             -----
2048: (8)             label
2049: (12)             The label. Defaults to :class:`~.MathTex` for ``str`` and
``float`` inputs.
2050: (8)             edge
2051: (12)             The edge of the z-axis to which the label will be added, by
default ``OUT``.
2052: (8)             direction
2053: (8)             Allows for further positioning of the label from an edge, by
default ``RIGHT``.
2054: (8)             buff

```

```

2053: (12)                                     The distance of the label from the line, by default
``SMALL_BUFF``.

2054: (8)                                     rotation
2055: (12)                                     The angle at which to rotate the label, by default ``PI/2``.
2056: (8)                                     rotation_axis
2057: (12)                                     The axis about which to rotate the label, by default ``RIGHT``.
2058: (8)                                     Returns
2059: (8)
2060: (8)                                     -----
2061: (12)                                     :class:`~.Mobject`
2062: (8)                                     The positioned label.
2063: (8)
2064: (8)                                     Examples
2065: (12)
2066: (12)
2067: (16)
2068: (20)
2069: (20)
2070: (20)
2071: (20)
2072: (8)
2073: (8)
2074: (12)                                     .. manim:: GetZAxisLabelExample
2075: (8)                                     :save_last_frame:
2076: (12)                                     class GetZAxisLabelExample(ThreeDScene):
2077: (8)                                     def construct(self):
2078: (4)                                     ax = ThreeDAxes()
2079: (8)                                     lab = ax.get_z_axis_label(Tex("$z$-label"))
2080: (8)                                     self.set_camera_orientation(phi=2*PI/5, theta=PI/5)
2081: (8)                                     self.add(ax, lab)
2082: (8)
2083: (4)                                     """
2084: (8)                                     positioned_label = self._get_axis_label(
2085: (8)                                     label, self.get_z_axis(), edge, direction, buff=buf, **kwargs
2086: (8)                                     )
2087: (8)                                     positioned_label.rotate(rotation, axis=rotation_axis)
2088: (8)                                     return positioned_label
2089: (8)
2090: (8)                                     def get_axis_labels(
2091: (8)                                     self,
2092: (12)                                     x_label: float | str | Mobject = "x",
2093: (8)                                     y_label: float | str | Mobject = "y",
2094: (12)                                     z_label: float | str | Mobject = "z",
2095: (8)                                     ) -> VGroup:
2096: (8)                                     """Defines labels for the x_axis and y_axis of the graph.
2097: (8)                                     For increased control over the position of the labels,
2098: (8)                                     use :meth:`~.CoordinateSystem.get_x_axis_label`,
2099: (8)                                     :meth:`~.ThreeDAxes.get_y_axis_label`, and
2100: (12)                                     :meth:`~.ThreeDAxes.get_z_axis_label`.
2101: (8)                                     Parameters
2102: (12)
2103: (12)
2104: (12)
2105: (8)
2106: (8)
2107: (8)                                     x_label
2108: (12)                                     The label for the x_axis. Defaults to :class:`~.MathTex` for
2109: (12)                                     ``str`` and ``float`` inputs.
2110: (16)
2111: (20)
2112: (20)
2113: (20)
2114: (24)
2115: (20)                                     y_label
2116: (12)                                     The label for the y_axis. Defaults to :class:`~.MathTex` for
2117: (12)                                     ``str`` and ``float`` inputs.
2118: (16)
2119: (20)
2120: (20)
2121: (20)
2122: (24)
2123: (20)                                     z_label
2124: (12)                                     The label for the z_axis. Defaults to :class:`~.MathTex` for
2125: (12)                                     ``str`` and ``float`` inputs.
2126: (8)
2127: (8)                                     Returns
2128: (8)
2129: (8)                                     -----
2130: (8)                                     :class:`~.VGroup`
2131: (12)                                     A :class:`~.VGroup` of the labels for the x_axis, y_axis, and
2132: (12)                                     z_axis.
2133: (8)
2134: (8)
2135: (8)                                     .. seealso::
2136: (12)                                     :meth:`~.CoordinateSystem.get_x_axis_label`
2137: (12)                                     :meth:`~.ThreeDAxes.get_y_axis_label`
2138: (12)                                     :meth:`~.ThreeDAxes.get_z_axis_label`
2139: (8)
2140: (8)
2141: (8)                                     Examples
2142: (8)
2143: (8)                                     .. manim:: GetAxisLabelsExample
2144: (12)
2145: (12)                                     :save_last_frame:
2146: (12)                                     class GetAxisLabelsExample(ThreeDScene):
2147: (16)                                     def construct(self):
2148: (20)                                     self.set_camera_orientation(phi=2*PI/5, theta=PI/5)
2149: (20)                                     axes = ThreeDAxes()
2150: (20)                                     labels = axes.get_axis_labels(
2151: (24)                                     Text("x-axis").scale(0.7), Text("y-axis").scale(0.45),
2152: (24)                                     Text("z-axis").scale(0.45)
2153: (20)
2154: (20)
```

```

12/20/24, 4:24 AM manims_installed_to_implement_with_phenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2116: (20)                         self.add(axes, labels)
2117: (8)
2118: (8)
2119: (12)                         self.axis_labels = VGroup(
2120: (12)                             self.get_x_axis_label(x_label),
2121: (12)                             self.get_y_axis_label(y_label),
2122: (12)                             self.get_z_axis_label(z_label),
2123: (8)                         )
2124: (0)                         return self.axis_labels
class NumberPlane(Axes):
    """Creates a cartesian plane with background lines.
    Parameters
    -----
    x_range
        The ``[x_min, x_max, x_step]`` values of the plane in the horizontal
        direction.
    y_range
        The ``[y_min, y_max, y_step]`` values of the plane in the vertical
        direction.
    x_length
        The width of the plane.
    y_length
        The height of the plane.
    background_line_style
        Arguments that influence the construction of the background lines of
        the plane.
    faded_line_style
        Similar to :attr:`background_line_style`, affects the construction of
        the scene's background lines.
    faded_line_ratio
        Determines the number of boxes within the background lines: :code:`2`-
        = 4 boxes, :code:`3` = 9 boxes.
    make_smooth_after_applying_functions
        Currently non-functional.
    kwargs
        Additional arguments to be passed to :class:`Axes`.
    .. note::
        If :attr:`x_length` or :attr:`y_length` are not defined, they are
        automatically calculated such that
            one unit on each axis is one Manim unit long.
    Examples
    -----
    .. manim:: NumberPlaneExample
        :save_last_frame:
        class NumberPlaneExample(Scene):
            def construct(self):
                number_plane = NumberPlane(
                    background_line_style={
                        "stroke_color": TEAL,
                        "stroke_width": 4,
                        "stroke_opacity": 0.6
                    }
                )
                self.add(number_plane)
    .. manim:: NumberPlaneScaled
        :save_last_frame:
        class NumberPlaneScaled(Scene):
            def construct(self):
                number_plane = NumberPlane(
                    x_range=(-4, 11, 1),
                    y_range=(-3, 3, 1),
                    x_length=5,
                    y_length=2,
                    ).move_to(LEFT*3)
                number_plane_scaled_y = NumberPlane(
                    x_range=(-4, 11, 1),
                    x_length=5,
                    y_length=4,
                    ).move_to(RIGHT*3)
                self.add(number_plane)

```

```

2179: (16)                         self.add(number_plane_scaled_y)
2180: (4)                         """
2181: (4)                     def __init__(
2182: (8)                         self,
2183: (8)                         x_range: Sequence[float] | None = (
2184: (12)                             -config["frame_x_radius"],
2185: (12)                             config["frame_x_radius"],
2186: (12)                             1,
2187: (8) ),
2188: (8)                         y_range: Sequence[float] | None = (
2189: (12)                             -config["frame_y_radius"],
2190: (12)                             config["frame_y_radius"],
2191: (12)                             1,
2192: (8) ),
2193: (8)                         x_length: float | None = None,
2194: (8)                         y_length: float | None = None,
2195: (8)                         background_line_style: dict[str, Any] | None = None,
2196: (8)                         faded_line_style: dict[str, Any] | None = None,
2197: (8)                         faded_line_ratio: int = 1,
2198: (8)                         make_smooth_after_applying_functions: bool = True,
2199: (8)                         **kwargs: dict[str, Any],
2200: (4) ):                         self.axis_config = {
2201: (8)                         "stroke_width": 2,
2202: (12)                         "include_ticks": False,
2203: (12)                         "include_tip": False,
2204: (12)                         "line_to_number_buff": SMALL_BUFF,
2205: (12)                         "label_direction": DR,
2206: (12)                         "font_size": 24,
2207: (12)                     }
2208: (8)                         self.y_axis_config = {"label_direction": DR}
2209: (8)                         self.background_line_style = {
2210: (8)                             "stroke_color": BLUE_D,
2211: (12)                             "stroke_width": 2,
2212: (12)                             "stroke_opacity": 1,
2213: (12)                         }
2214: (8)                         self._update_default_configs(
2215: (8)                             (self.axis_config, self.y_axis_config,
2216: (12)                             self.background_line_style),
2217: (12)                         (
2218: (16)                             kwargs.pop("axis_config", None),
2219: (16)                             kwargs.pop("y_axis_config", None),
2220: (16)                             background_line_style,
2221: (12)                         ),
2222: (8)                         )
2223: (8)                         self.faded_line_style = faded_line_style
2224: (8)                         self.faded_line_ratio = faded_line_ratio
2225: (8)                         self.make_smooth_after_applying_functions =
make_smooth_after_applying_functions
2226: (8)                         super().__init__(
2227: (12)                             x_range=x_range,
2228: (12)                             y_range=y_range,
2229: (12)                             x_length=x_length,
2230: (12)                             y_length=y_length,
2231: (12)                             axis_config=self.axis_config,
2232: (12)                             y_axis_config=self.y_axis_config,
2233: (12)                             **kwargs,
2234: (8)                         )
2235: (8)                         self._init_background_lines()
2236: (4)                     def _init_background_lines(self) -> None:
2237: (8)                         """Will init all the lines of NumberPlanes (faded or not)"""
2238: (8)                         if self.faded_line_style is None:
2239: (12)                             style = dict(self.background_line_style)
2240: (12)                             for key in style:
2241: (16)                                 if isinstance(style[key], numbers.Number):
2242: (20)                                     style[key] *= 0.5
2243: (12)                             self.faded_line_style = style
2244: (8)                             self.background_lines, self.faded_lines = self._get_lines()
2245: (8)                             self.background_lines.set_style(

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```
2246: (12)             **self.background_line_style,
2247: (8)         )
2248: (8)         self.faded_lines.set_style(
2249: (12)             **self.faded_line_style,
2250: (8)         )
2251: (8)         self.add_to_back(
2252: (12)             self.faded_lines,
2253: (12)             self.background_lines,
2254: (8)         )
2255: (4)     def _get_lines(self) -> tuple[VGroup, VGroup]:
2256: (8)         """Generate all the lines, faded and not faded.
2257: (9)             Two sets of lines are generated: one parallel to the X-axis, and
parallel to the Y-axis.
2258: (8)         Returns
2259: (8)             -----
2260: (8)             Tuple[:class:`~.VGroup`, :class:`~.VGroup`]
2261: (12)             The first (i.e the non faded lines) and second (i.e the faded
lines) sets of lines, respectively.
2262: (8)             """
2263: (8)             x_axis = self.get_x_axis()
2264: (8)             y_axis = self.get_y_axis()
2265: (8)             x_lines1, x_lines2 = self._get_lines_parallel_to_axis(
2266: (12)                 x_axis,
2267: (12)                 y_axis,
2268: (12)                 self.y_axis.x_range[2],
2269: (12)                 self.faded_line_ratio,
2270: (8)             )
2271: (8)             y_lines1, y_lines2 = self._get_lines_parallel_to_axis(
2272: (12)                 y_axis,
2273: (12)                 x_axis,
2274: (12)                 self.x_axis.x_range[2],
2275: (12)                 self.faded_line_ratio,
2276: (8)             )
2277: (8)             self.x_lines = x_lines1
2278: (8)             self.y_lines = y_lines1
2279: (8)             lines1 = VGroup(*x_lines1, *y_lines1)
2280: (8)             lines2 = VGroup(*x_lines2, *y_lines2)
2281: (8)             return lines1, lines2
2282: (4)     def _get_lines_parallel_to_axis(
2283: (8)             self,
2284: (8)             axis_parallel_to: NumberLine,
2285: (8)             axis_perpendicular_to: NumberLine,
2286: (8)             freq: float,
2287: (8)             ratio_faded_lines: int,
2288: (4)         ) -> tuple[VGroup, VGroup]:
2289: (8)             """Generate a set of lines parallel to an axis.
2290: (8)             Parameters
2291: (8)             -----
2292: (8)             axis_parallel_to
2293: (12)                 The axis with which the lines will be parallel.
2294: (8)             axis_perpendicular_to
2295: (12)                 The axis with which the lines will be perpendicular.
2296: (8)             ratio_faded_lines
2297: (12)                 The ratio between the space between faded lines and the space
between non-faded lines.
2298: (8)             freq
2299: (12)                 Frequency of non-faded lines (number of non-faded lines per graph
unit).
2300: (8)             Returns
2301: (8)             -----
2302: (8)             Tuple[:class:`~.VGroup`, :class:`~.VGroup`]
2303: (12)                 The first (i.e the non-faded lines parallel to `axis_parallel_to`)
and second
2304: (13)                 (i.e the faded lines parallel to `axis_parallel_to`) sets of
lines, respectively.
2305: (8)             """
2306: (8)             line = Line(axis_parallel_to.get_start(), axis_parallel_to.get_end())
2307: (8)             if ratio_faded_lines == 0: # don't show faded lines
2308: (12)                 ratio_faded_lines = 1 # i.e. set ratio to 1
```

```

2309: (8)             step = (1 / ratio_faded_lines) * freq
2310: (8)             lines1 = VGroup()
2311: (8)             lines2 = VGroup()
2312: (8)             unit_vector_axis_perp_to = axis_perpendicular_to.get_unit_vector()
2313: (8)             x_min, x_max, _ = axis_perpendicular_to.x_range
2314: (8)             if axis_perpendicular_to.x_min > 0 and x_min < 0:
2315: (12)                x_min, x_max = (0, np.abs(x_min) + np.abs(x_max))
2316: (8)             ranges = (
2317: (12)                [0],
2318: (12)                np.arange(step, min(x_max - x_min, x_max), step),
2319: (12)                np.arange(-step, max(x_min - x_max, x_min), -step),
2320: (8)
2321: (8)             for inputs in ranges:
2322: (12)                 for k, x in enumerate(inputs):
2323: (16)                     new_line = line.copy()
2324: (16)                     new_line.shift(unit_vector_axis_perp_to * x)
2325: (16)                     if (k + 1) % ratio_faded_lines == 0:
2326: (20)                         lines1.add(new_line)
2327: (16)
2328: (20)                     else:
2329: (8)                         lines2.add(new_line)
2330: (4)             return lines1, lines2
def get_vector(self, coords: Sequence[ManimFloat], **kwargs: Any) ->
Arrow:
2331: (8)             kwargs["buff"] = 0
2332: (8)             return Arrow(
2333: (12)                 self.coords_to_point(0, 0), self.coords_to_point(*coords),
**kwargs
2334: (8)
2335: (4)             def prepare_for_nonlinear_transform(self, num_inserted_curves: int = 50) -
> Self:
2336: (8)                 for mob in self.family_members_with_points():
2337: (12)                     num_curves = mob.get_num_curves()
2338: (12)                     if num_inserted_curves > num_curves:
2339: (16)                         mob.insert_n_curves(num_inserted_curves - num_curves)
2340: (8)
2341: (0)             class PolarPlane(Axes):
2342: (4)                 """Creates a polar plane with background lines.
2343: (4)                 Parameters
2344: (4)                 -----
2345: (4)                 azimuth_step
2346: (8)                     The number of divisions in the azimuth (also known as the `angular
coordinate` or `polar angle`). If ``None`` is specified then it will use the default
2347: (8)                     specified by ``azimuth_units``:
2348: (8)                     - ``"PI radians"`` or ``"TAU radians"``: 20
2349: (8)                     - ``"degrees"``: 36
2350: (8)                     - ``"gradians"``: 40
2351: (8)                     - ``None``: 1
2352: (8)                     A non-integer value will result in a partial division at the end of
the circle.
2353: (4)
2354: (8)                     size
2355: (4)                     The diameter of the plane.
2356: (8)                     radius_step
2357: (4)                     The distance between faded radius lines.
2358: (8)                     radius_max
2359: (4)                     The maximum value of the radius.
azimuth_units
2360: (8)                     Specifies a default labelling system for the azimuth. Choices are:
2361: (8)                     - ``"PI radians"``: Fractional labels in the interval :math:`\left[0, 2\pi\right]` with :math:`\pi` as a constant.
2362: (8)                     - ``"TAU radians"``: Fractional labels in the interval :math:`\left[0, \tau\right]` (where :math:`\tau = 2\pi` with :math:`\tau` as a constant.
2363: (8)                     - ``"degrees"``: Decimal labels in the interval :math:`\left[0, 360\right]` with a degree (:math:`^\circ` symbol.
2364: (8)                     - ``"gradians"``: Decimal labels in the interval :math:`\left[0, 400\right]` with a superscript "g" (:math:`^g`).
2365: (8)                     - ``None``: Decimal labels in the interval :math:`\left[0, 1\right]`.
2366: (4)                     azimuth_compact_fraction
2367: (8)                     If the ``azimuth_units`` choice has fractional labels, choose whether
to

```

```

2368: (8)           combine the constant in a compact form :math:`\tfrac{xu}{y}` as
opposed to
2369: (8)           :math:`\tfrac{xy}{u}` , where :math:`u` is the constant.
2370: (4)           azimuth_offset
2371: (8)           The angle offset of the azimuth, expressed in radians.
2372: (4)           azimuth_direction
2373: (8)           The direction of the azimuth.
2374: (8)           - ``"CW"``: Clockwise.
2375: (8)           - ``"CCW"``: Anti-clockwise.
2376: (4)           azimuth_label_buff
2377: (8)           The buffer for the azimuth labels.
2378: (4)           azimuth_label_font_size
2379: (8)           The font size of the azimuth labels.
2380: (4)           radius_config
2381: (8)           The axis config for the radius.
2382: (4)           Examples
2383: (4)           -----
2384: (4)           .. manim:: PolarPlaneExample
2385: (8)           :ref_classes: PolarPlane
2386: (8)           :save_last_frame:
2387: (8)           class PolarPlaneExample(Scene):
2388: (12)           def construct(self):
2389: (16)               polarplane_pi = PolarPlane(
2390: (20)                   azimuth_units="PI radians",
2391: (20)                   size=6,
2392: (20)                   azimuth_label_font_size=33.6,
2393: (20)                   radius_config={"font_size": 33.6},
2394: (16)               ).add_coordinates()
2395: (16)               self.add(polarplane_pi)
2396: (4)           """
2397: (4)           def __init__(
2398: (8)               self,
2399: (8)               radius_max: float = config["frame_y_radius"],
2400: (8)               size: float | None = None,
2401: (8)               radius_step: float = 1,
2402: (8)               azimuth_step: float | None = None,
2403: (8)               azimuth_units: str | None = "PI radians",
2404: (8)               azimuth_compact_fraction: bool = True,
2405: (8)               azimuth_offset: float = 0,
2406: (8)               azimuth_direction: str = "CCW",
2407: (8)               azimuth_label_buff: float = SMALL_BUFF,
2408: (8)               azimuth_label_font_size: float = 24,
2409: (8)               radius_config: dict[str, Any] | None = None,
2410: (8)               background_line_style: dict[str, Any] | None = None,
2411: (8)               faded_line_style: dict[str, Any] | None = None,
2412: (8)               faded_line_ratio: int = 1,
2413: (8)               make_smooth_after_applying_functions: bool = True,
2414: (8)               **kwargs: Any,
2415: (4)           ) -> None:
2416: (8)               if azimuth_units in ["PI radians", "TAU radians", "degrees",
"gradians", None]:
2417: (12)                   self.azimuth_units = azimuth_units
2418: (8)               else:
2419: (12)                   raise ValueError(
2420: (16)                       "Invalid azimuth units. Expected one of: PI radians, TAU
radians, degrees, gradians or None.",
2421: (12)                           )
2422: (8)                   if azimuth_direction in ["CW", "CCW"]:
2423: (12)                       self.azimuth_direction = azimuth_direction
2424: (8)                   else:
2425: (12)                       raise ValueError("Invalid azimuth units. Expected one of: CW,
CCW.")
2426: (8)                   self.radius_config = {
2427: (12)                       "stroke_width": 2,
2428: (12)                       "include_ticks": False,
2429: (12)                       "include_tip": False,
2430: (12)                       "line_to_number_buff": SMALL_BUFF,
2431: (12)                       "label_direction": DL,
2432: (12)                       "font_size": 24,

```

```

2433: (8) }
2434: (8) self.background_line_style = {
2435: (12)     "stroke_color": BLUE_D,
2436: (12)     "stroke_width": 2,
2437: (12)     "stroke_opacity": 1,
2438: (8) }
2439: (8) self.azimuth_step = (
2440: (12)     (
2441: (16)         {
2442: (20)             "PI radians": 20,
2443: (20)             "TAU radians": 20,
2444: (20)             "degrees": 36,
2445: (20)             "gradians": 40,
2446: (20)             None: 1,
2447: (16)         }[azimuth_units]
2448: (12)     )
2449: (12)     if azimuth_step is None
2450: (12)     else azimuth_step
2451: (8) )
2452: (8) self._update_default_configs(
2453: (12)     (self.radius_config, self.background_line_style),
2454: (12)     (radius_config, background_line_style),
2455: (8) )
2456: (8) self.faded_line_style = faded_line_style
2457: (8) self.faded_line_ratio = faded_line_ratio
2458: (8) self.make_smooth_after_applying_functions =
make_smooth_after_applying_functions
2459: (8)     self.azimuth_offset = azimuth_offset
2460: (8)     self.azimuth_label_buff = azimuth_label_buff
2461: (8)     self.azimuth_label_font_size = azimuth_label_font_size
2462: (8)     self.azimuth_compact_fraction = azimuth_compact_fraction
2463: (8) super().__init__(
2464: (12)         x_range=np.array((-radius_max, radius_max, radius_step)),
2465: (12)         y_range=np.array((-radius_max, radius_max, radius_step)),
2466: (12)         x_length=size,
2467: (12)         y_length=size,
2468: (12)         axis_config=self.radius_config,
2469: (12)         **kwargs,
2470: (8)     )
2471: (8)     self._init_background_lines()
def _init_background_lines(self) -> None:
2472: (4)     """Will init all the lines of NumberPlanes (faded or not)"""
2473: (8)     if self.faded_line_style is None:
2474: (8)         style = dict(self.background_line_style)
2475: (12)         for key in style:
2476: (12)             if isinstance(style[key], numbers.Number):
2477: (16)                 style[key] *= 0.5
2478: (20)             self.faded_line_style = style
2479: (12)             self.background_lines, self.faded_lines = self._get_lines()
2480: (8)             self.background_lines.set_style(
2481: (8)                 **self.background_line_style,
2482: (12)             )
2483: (8)             self.faded_lines.set_style(
2484: (8)                 **self.faded_line_style,
2485: (12)             )
2486: (8)             self.add_to_back(
2487: (8)                 self.faded_lines,
2488: (12)                 self.background_lines,
2489: (12)             )
2490: (8)         )
def _get_lines(self) -> tuple[VGroup, VGroup]:
2491: (4)     """Generate all the lines and circles, faded and not faded.
2492: (8)     Returns
2493: (8)     -----
2494: (8)     Tuple[:class:`~.VGroup`, :class:`~.VGroup`]
2495: (8)     The first (i.e the non faded lines and circles) and second (i.e
2496: (12)     the faded lines and circles) sets of lines and circles, respectively.
2497: (8)     """
2498: (8)     center = self.get_origin()
2499: (8)     ratio_faded_lines = self.faded_line_ratio

```

```

2500: (8)             offset = self.azimuth_offset
2501: (8)             if ratio_faded_lines == 0: # don't show faded lines
2502: (12)             ratio_faded_lines = 1 # i.e. set ratio to 1
2503: (8)             rstep = (1 / ratio_faded_lines) * self.x_axis.x_range[2]
2504: (8)             astep = (1 / ratio_faded_lines) * (TAU * (1 / self.azimuth_step))
2505: (8)             rlines1 = VGroup()
2506: (8)             rlines2 = VGroup()
2507: (8)             alines1 = VGroup()
2508: (8)             alines2 = VGroup()
2509: (8)             rinput = np.arange(0, self.x_axis.x_range[1] + rstep, rstep)
2510: (8)             ainput = np.arange(0, TAU, astep)
2511: (8)             unit_vector = self.x_axis.get_unit_vector()[0]
2512: (8)             for k, x in enumerate(rinput):
2513: (12)                 new_line = Circle(radius=x * unit_vector)
2514: (12)                 if k % ratio_faded_lines == 0:
2515: (16)                     alines1.add(new_line)
2516: (12)                 else:
2517: (16)                     alines2.add(new_line)
2518: (8)             line = Line(center, self.get_x_axis().get_end())
2519: (8)             for k, x in enumerate(aiput):
2520: (12)                 new_line = line.copy()
2521: (12)                 new_line.rotate(x + offset, about_point=center)
2522: (12)                 if k % ratio_faded_lines == 0:
2523: (16)                     rlines1.add(new_line)
2524: (12)                 else:
2525: (16)                     rlines2.add(new_line)
2526: (8)             lines1 = VGroup(*rlines1, *alines1)
2527: (8)             lines2 = VGroup(*rlines2, *alines2)
2528: (8)             return lines1, lines2
2529: (4)             def get_axes(self) -> VGroup:
2530: (8)                 """Gets the axes.
2531: (8)                 Returns
2532: (8)                 -----
2533: (8)                 :class:`~.VGroup`
2534: (12)                 A pair of axes.
2535: (8)
2536: (8)
2537: (4)             def get_vector(self, coords: Sequence[ManimFloat], **kwargs: Any) ->
Arrow:
2538: (8)                 kwargs["buff"] = 0
2539: (8)                 return Arrow(
2540: (12)                     self.coords_to_point(0, 0), self.coords_to_point(*coords),
**kwargs
2541: (8)
2542: (4)             def prepare_for_nonlinear_transform(self, num_inserted_curves: int = 50) -
> Self:
2543: (8)                 for mob in self.family_members_with_points():
2544: (12)                     num_curves = mob.get_num_curves()
2545: (12)                     if num_inserted_curves > num_curves:
2546: (16)                         mob.insert_n_curves(num_inserted_curves - num_curves)
2547: (8)
2548: (4)             def get_coordinate_labels(
2549: (8)                 self,
2550: (8)                 r_values: Iterable[float] | None = None,
2551: (8)                 a_values: Iterable[float] | None = None,
2552: (8)                 **kwargs: Any,
2553: (4)             ) -> VDict:
2554: (8)                 """Gets labels for the coordinates
Parameters
-----
r_values
    Iterable of values along the radius, by default None.
a_values
    Iterable of values along the azimuth, by default None.
Returns
-----
VDict
    Labels for the radius and azimuth values.
"""

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2566: (8)           if r_values is None:
2567: (12)             r_values = [r for r in self.get_x_axis().get_tick_range() if r >=
0]
2568: (8)           if a_values is None:
2569: (12)             a_values = np.arange(0, 1, 1 / self.azimuth_step)
2570: (8)             r_mobs = self.get_x_axis().add_numbers(r_values)
2571: (8)             if self.azimuth_direction == "CCW":
2572: (12)               d = 1
2573: (8)             elif self.azimuth_direction == "CW":
2574: (12)               d = -1
2575: (8)             else:
2576: (12)               raise ValueError("Invalid azimuth direction. Expected one of: CW,
CCW")
2577: (8)             a_points = [
2578: (12)               {
2579: (16)                 "label": i,
2580: (16)                 "point": np.array(
2581: (20)                   [
2582: (24)                     self.get_right()[0]
2583: (24)                     * np.cos(d * (i * TAU) + self.azimuth_offset),
2584: (24)                     self.get_right()[0]
2585: (24)                     * np.sin(d * (i * TAU) + self.azimuth_offset),
2586: (24)                     0,
2587: (20)                   ],
2588: (16)                 ),
2589: (12)               }
2590: (12)             for i in a_values
2591: (8)           ]
2592: (8)           if self.azimuth_units == "PI radians" or self.azimuth_units == "TAU
radians":
2593: (12)             a_tex = [
2594: (16)               self.get_radian_label(
2595: (20)                 i["label"],
2596: (20)                 font_size=self.azimuth_label_font_size,
2597: (16)               ).next_to(
2598: (20)                 i["point"],
2599: (20)                 direction=i["point"],
2600: (20)                 aligned_edge=i["point"],
2601: (20)                 buff=self.azimuth_label_buff,
2602: (16)               )
2603: (16)               for i in a_points
2604: (12)             ]
2605: (8)             elif self.azimuth_units == "degrees":
2606: (12)               a_tex = [
2607: (16)                 MathTex(
2608: (20)                   f'{360 * i["label"]}:g}' + r"\circ",
2609: (20)                   font_size=self.azimuth_label_font_size,
2610: (16)                 ).next_to(
2611: (20)                   i["point"],
2612: (20)                   direction=i["point"],
2613: (20)                   aligned_edge=i["point"],
2614: (20)                   buff=self.azimuth_label_buff,
2615: (16)                 )
2616: (16)               for i in a_points
2617: (12)             ]
2618: (8)             elif self.azimuth_units == "gradians":
2619: (12)               a_tex = [
2620: (16)                 MathTex(
2621: (20)                   f'{400 * i["label"]}:g}' + r"\text{g}",
2622: (20)                   font_size=self.azimuth_label_font_size,
2623: (16)                 ).next_to(
2624: (20)                   i["point"],
2625: (20)                   direction=i["point"],
2626: (20)                   aligned_edge=i["point"],
2627: (20)                   buff=self.azimuth_label_buff,
2628: (16)                 )
2629: (16)               for i in a_points
2630: (12)             ]
2631: (8)             elif self.azimuth_units is None:

```

```

2632: (12)
2633: (16)
2634: (20)
2635: (20)
2636: (16)
2637: (20)
2638: (20)
2639: (20)
2640: (20)
2641: (16)
2642: (16)
2643: (12)
2644: (8)
2645: (8)
2646: (8)
2647: (4)
2648: (8)
2649: (8)
2650: (8)
2651: (4)
2652: (8)
2653: (8)
2654: (8)
2655: (8)
2656: (12)
2657: (8)
2658: (12)
2659: (8)
2660: (8)
2661: (8)
2662: (4)
> MathTex:
2663: (8)
2664: (12)
2665: (8)
2666: (8)
[self.azimuth_units]
2667: (8)
2668: (8)
2669: (12)
2670: (8)
2671: (12)
2672: (8)
2673: (12)
2674: (16)
2675: (20)
str(frac.denominator) + "}"
2676: (16)
2677: (12)
2678: (16)
constant_label
2679: (8)
2680: (12)
2681: (8)
2682: (12)
2683: (16)
2684: (20)
2685: (20)
2686: (20)
2687: (20)
2688: (20)
2689: (20)
2690: (16)
2691: (12)
2692: (16)
2693: (20)
2694: (20)
2695: (20)
2696: (20)

        a_tex = [
            MathTex(
                f'{i["label"]}:g',
                font_size=self.azimuth_label_font_size,
            ).next_to(
                i["point"],
                direction=i["point"],
                aligned_edge=i["point"],
                buff=self.azimuth_label_buff,
            )
            for i in a_points
        ]
        a_mobs = VGroup(*a_tex)
        self.coordinate_labels = VGroup(r_mobs, a_mobs)
        return self.coordinate_labels
    def add_coordinates(
        self,
        r_values: Iterable[float] | None = None,
        a_values: Iterable[float] | None = None,
    ) -> Self:
        """Adds the coordinates.
        Parameters
        -----
        r_values
            Iterable of values along the radius, by default None.
        a_values
            Iterable of values along the azimuth, by default None.
        """
        self.add(self.get_coordinate_labels(r_values, a_values))
        return self
    def get_radian_label(self, number, font_size: float = 24, **kwargs: Any) -
        constant_label = {"PI radians": r"\pi", "TAU radians": r"\tau"}[
            self.azimuth_units
        ]
        division = number * {"PI radians": 2, "TAU radians": 1}

        frac = fr.Fraction(division).limit_denominator(max_denominator=100)
        if frac.numerator == 0 & frac.denominator == 0:
            string = r"0"
        elif frac.numerator == 1 and frac.denominator == 1:
            string = constant_label
        elif frac.numerator == 1:
            if self.azimuth_compact_fraction:
                string = (
                    r"\tfrac{" + constant_label + r"}{"
                    +
                    str(frac.denominator) + "}"
                )
            else:
                string = r"\tfrac{1}{" + str(frac.denominator) + "}" +
        elif frac.denominator == 1:
            string = str(frac.numerator) + constant_label
        else:
            if self.azimuth_compact_fraction:
                string = (
                    r"\tfrac{"
                    +
                    str(frac.numerator)
                    +
                    constant_label
                    +
                    r"}{"
                    +
                    str(frac.denominator)
                    +
                    r"}"
                )
            else:
                string = (
                    r"\tfrac{"
                    +
                    str(frac.numerator)
                    +
                    r"}{"
                    +
                    str(frac.denominator)
                )

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2697: (20)
2698: (20)
2699: (16)
2700: (8)
2701: (0)
2702: (4)
2703: (4)
2704: (4)
2705: (4)
2706: (8)
2707: (8)
2708: (8)
2709: (12)
2710: (16)
2711: (16)
2712: (16)
2713: (16)
2714: (16)
2715: (16)
2716: (16)
2717: (20)
2718: (20)
2719: (20)
2720: (20)
2721: (16)
2722: (4)
2723: (4)
2724: (8)
2725: (12)
2726: (8)
2727: (4)
2728: (8)
the plane.
2729: (8)
2730: (8)
2731: (8)
2732: (12)
2733: (8)
2734: (8)
2735: (8)
2736: (12)
2737: (8)
2738: (8)
2739: (8)
2740: (4)
2741: (8)
2742: (8)
2743: (4)
2744: (8)
point on the plane.
2745: (8)
2746: (8)
2747: (8)
2748: (12)
2749: (8)
2750: (8)
2751: (8)
2752: (12)
2753: (8)
2754: (8)
2755: (8)
2756: (4)
2757: (8)
2758: (8)
2759: (4)
2760: (8)
labels.
2761: (8)
2762: (8)

        + r"}"
        + constant_label
    )
    return MathTex(string, font_size=font_size, **kwargs)
class ComplexPlane(NumberPlane):
    """A :class:`~.NumberPlane` specialized for use with complex numbers.
Examples
-----
.. manim:: ComplexPlaneExample
    :save_last_frame:
    :ref_classes: Dot MathTex
    class ComplexPlaneExample(Scene):
        def construct(self):
            plane = ComplexPlane().add_coordinates()
            self.add(plane)
            d1 = Dot(plane.n2p(2 + 1j), color=YELLOW)
            d2 = Dot(plane.n2p(-3 - 2j), color=YELLOW)
            label1 = MathTex("2+i").next_to(d1, UR, 0.1)
            label2 = MathTex("-3-2i").next_to(d2, UR, 0.1)
            self.add(
                d1,
                label1,
                d2,
                label2,
            )
    """
    def __init__(self, **kwargs: Any) -> None:
        super().__init__(
            **kwargs,
        )
    def number_to_point(self, number: float | complex) -> np.ndarray:
        """Accepts a float/complex number and returns the equivalent point on
Parameters
-----
number
    The number. Can be a float or a complex number.
Returns
-----
np.ndarray
    The point on the plane.
"""
        number = complex(number)
        return self.coords_to_point(number.real, number.imag)
    def n2p(self, number: float | complex) -> np.ndarray:
        """Abbreviation for :meth:`number_to_point`."""
        return self.number_to_point(number)
    def point_to_number(self, point: Point3D) -> complex:
        """Accepts a point and returns a complex number equivalent to that
Parameters
-----
point
    The point in manim's coordinate-system
Returns
-----
complex
    A complex number consisting of real and imaginary components.
"""
        x, y = self.point_to_coords(point)
        return complex(x, y)
    def p2n(self, point: Point3D) -> complex:
        """Abbreviation for :meth:`point_to_number`."""
        return self.point_to_number(point)
    def _get_default_coordinate_values(self) -> list[float | complex]:
        """Generate a list containing the numerical values of the plane's
Returns
-----

```

```

2763: (8)             List[float | complex]
2764: (12)             A list of floats representing the x-axis and complex numbers
representing the y-axis.
2765: (8)             """
2766: (8)             x_numbers = self.get_x_axis().get_tick_range()
2767: (8)             y_numbers = self.get_y_axis().get_tick_range()
2768: (8)             y_numbers = [complex(0, y) for y in y_numbers if y != 0]
2769: (8)             return [*x_numbers, *y_numbers]
2770: (4)             def get_coordinate_labels(
2771: (8)                 self, *numbers: Iterable[float | complex], **kwargs: Any
2772: (4)             ) -> VGroup:
2773: (8)                 """Generates the :class:`~.DecimalNumber` mobjects for the coordinates
of the plane.
2774: (8)             Parameters
2775: (8)             -----
2776: (8)             numbers
2777: (12)             An iterable of floats/complex numbers. Floats are positioned along
the x-axis, complex numbers along the y-axis.
2778: (8)             kwargs
2779: (12)             Additional arguments to be passed to
:meth:`~.NumberLine.get_number_mobject`, i.e. :class:`~.DecimalNumber`.
2780: (8)             Returns
2781: (8)             -----
2782: (8)             :class:`~.VGroup`
2783: (12)             A :class:`~.VGroup` containing the positioned label mobjects.
2784: (8)             """
2785: (8)             if len(numbers) == 0:
2786: (12)                 numbers = self._get_default_coordinate_values()
2787: (8)             self.coordinate_labels = VGroup()
2788: (8)             for number in numbers:
2789: (12)                 z = complex(number)
2790: (12)                 if abs(z.imag) > abs(z.real):
2791: (16)                     axis = self.get_y_axis()
2792: (16)                     value = z.imag
2793: (16)                     kwargs["unit"] = "i"
2794: (12)                 else:
2795: (16)                     axis = self.get_x_axis()
2796: (16)                     value = z.real
2797: (12)                     number_mob = axis.get_number_mobject(value, **kwargs)
2798: (12)                     self.coordinate_labels.add(number_mob)
2799: (8)             return self.coordinate_labels
2800: (4)             def add_coordinates(
2801: (8)                 self, *numbers: Iterable[float | complex], **kwargs: Any
2802: (4)             ) -> Self:
2803: (8)                 """Adds the labels produced from
:meth:`~.NumberPlane.get_coordinate_labels` to the plane.
2804: (8)             Parameters
2805: (8)             -----
2806: (8)             numbers
2807: (12)             An iterable of floats/complex numbers. Floats are positioned along
the x-axis, complex numbers along the y-axis.
2808: (8)             kwargs
2809: (12)             Additional arguments to be passed to
:meth:`~.NumberLine.get_number_mobject`, i.e. :class:`~.DecimalNumber`.
2810: (8)             """
2811: (8)             self.add(self.get_coordinate_labels(*numbers, **kwargs))
2812: (8)             return self

```

-----

## File 73 - opengl\_compatibility.py:

```

1: (0)             from __future__ import annotations
2: (0)             from abc import ABCMeta
3: (0)             from manim import config
4: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject
5: (0)             from manim.mobject.opengl.opengl_point_cloud_mobject import OpenGLPObject
6: (0)             from manim.mobject.opengl.opengl_three_dimensions import OpenGLSurface
7: (0)             from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLVObject

```

```

8: (0)         from ...constants import RenderType
9: (0)         __all__ = ["ConvertToOpenGL"]
10: (0)        class ConvertToOpenGL(ABCMeta):
11: (4)          """Metaclass for swapping (V)Mobject with its OpenGL counterpart at
12: (4)          runtime
13: (4)          depending on config.renderer. This metaclass should only need to be
14: (4)          on the lowest order inheritance classes such as Mobject and VMobject.
15: (4)          """
16: (4)          _converted_classes = []
17: (8)          def __new__(mcls, name, bases, namespace): # noqa: B902
18: (12)            if config.renderer == RenderType.OPENGL:
19: (16)              base_names_to_opengl = {
20: (16)                "Mobject": OpenGLMobject,
21: (16)                "VMobject": OpenGLVMobject,
22: (16)                "PMobject": OpenGLPMobject,
23: (16)                "Mobject1D": OpenGLPMobject,
24: (16)                "Mobject2D": OpenGLPMobject,
25: (16)                "Surface": OpenGLSurface,
26: (12)              }
27: (16)              bases = tuple(
28: (12)                base_names_to_opengl.get(base.__name__, base) for base in
29: (12)                bases
30: (8)              )
31: (4)              return super().__new__(mcls, name, bases, namespace)
32: (8)          def __init__(cls, name, bases, namespace): # noqa: B902
33: (8)            super().__init__(name, bases, namespace)
34: (8)            cls._converted_classes.append(cls)

```

-----

## File 74 - table.py:

```

1: (0)          r"""Mobjects representing tables.
2: (0)          Examples
3: (0)          -----
4: (0)          .. manim:: TableExamples
5: (4)            :save_last_frame:
6: (4)            class TableExamples(Scene):
7: (8)              def construct(self):
8: (12)                t0 = Table(
9: (16)                  [[["First", "Second"],
10: (16)                    ["Third", "Fourth"]],
11: (16)                    row_labels=[Text("R1"), Text("R2")],
12: (16)                    col_labels=[Text("C1"), Text("C2")],
13: (16)                    top_left_entry=Text("TOP"))
14: (12)                  t0.add_highlighted_cell((2,2), color=GREEN)
15: (12)                  x_vals = np.linspace(-2,2,5)
16: (12)                  y_vals = np.exp(x_vals)
17: (12)                  t1 = DecimalTable(
18: (16)                    [x_vals, y_vals],
19: (16)                    row_labels=[MathTex("x"), MathTex("f(x)")],
20: (16)                    include_outer_lines=True)
21: (12)                  t1.add(t1.get_cell((2,2), color=RED))
22: (12)                  t2 = MathTable(
23: (16)                    [[["+", 0, 5, 10],
24: (16)                      [0, 0, 5, 10],
25: (16)                      [2, 2, 7, 12],
26: (16)                      [4, 4, 9, 14]],
27: (16)                      include_outer_lines=True)
28: (12)                  t2.get_horizontal_lines()[:3].set_color(BLUE)
29: (12)                  t2.get_vertical_lines()[:3].set_color(BLUE)
30: (12)                  t2.get_horizontal_lines()[:3].set_z_index(1)
31: (12)                  cross = VGroup(
32: (16)                    Line(UP + LEFT, DOWN + RIGHT),
33: (16)                    Line(UP + RIGHT, DOWN + LEFT))
34: (12)                  a = Circle().set_color(RED).scale(0.5)
35: (12)                  b = cross.set_color(BLUE).scale(0.5)
36: (12)                  t3 = MobjectTable(

```

```

37: (16)                         [[a.copy(),b.copy(),a.copy()],
38: (16)                         [b.copy(),a.copy(),a.copy()],
39: (16)                         [a.copy(),b.copy(),b.copy()]])
40: (12)                         t3.add(Line(
41: (16)                           t3.get_corner(DL), t3.get_corner(UR)
42: (12)                         ).set_color(RED))
43: (12)                         vals = np.arange(1,21).reshape(5,4)
44: (12)                         t4 = IntegerTable(
45: (16)                           vals,
46: (16)                           include_outer_lines=True
47: (12)                         )
48: (12)                         g1 = Group(t0, t1).scale(0.5).arrange(buff=1).to_edge(UP, buff=1)
49: (12)                         g2 = Group(t2, t3, t4).scale(0.5).arrange(buff=1).to_edge(DOWN,
buff=1)
50: (12)                         self.add(g1, g2)
51: (0) """
52: (0)     from __future__ import annotations
53: (0)     __all__ = [
54: (4)       "Table",
55: (4)       "MathTable",
56: (4)       "MobjectTable",
57: (4)       "IntegerTable",
58: (4)       "DecimalTable",
59: (0)     ]
60: (0)     import itertools as it
61: (0)     from typing import Callable, Iterable, Sequence
62: (0)     from manim.mobject.geometry.line import Line
63: (0)     from manim.mobject.geometry.polygram import Polygon
64: (0)     from manim.mobject.geometry.shape_matchers import BackgroundRectangle
65: (0)     from manim.mobject.text.numbers import DecimalNumber, Integer
66: (0)     from manim.mobject.text.tex_mobject import MathTex
67: (0)     from manim.mobject.text.text_mobject import Paragraph
68: (0)     from ..animation.animation import Animation
69: (0)     from ..animation.composition import AnimationGroup
70: (0)     from ..animation.creation import Create, Write
71: (0)     from ..animation.fading import FadeIn
72: (0)     from ..mobject.types.vectorized_mobject import VGroup, VMobject
73: (0)     from ..utils.color import BLACK, YELLOW, ManimColor, ParsableManimColor
74: (0)     from ..utils import get_vectorized_mobject_class
75: (0)     class Table(VGroup):
76: (4)         """A mobject that displays a table on the screen.
77: (4)         Parameters
78: (4)         -----
79: (4)         table
80: (8)             A 2D array or list of lists. Content of the table has to be a valid
input
81: (8)             for the callable set in ``element_to_mobject``.
82: (4)         row_labels
83: (8)             List of :class:`~.VMobject` representing the labels of each row.
84: (4)         col_labels
85: (8)             List of :class:`~.VMobject` representing the labels of each column.
86: (4)         top_left_entry
87: (8)             The top-left entry of the table, can only be specified if row and
column labels are given.
88: (8)
89: (4)
90: (8)             Vertical buffer passed to :meth:`~.Mobject.arrange_in_grid`, by
default 0.8.
91: (4)
92: (8)             Horizontal buffer passed to :meth:`~.Mobject.arrange_in_grid`, by
default 1.3.
93: (4)
94: (8)             ``True`` if the table should include outer lines, by default False.
95: (4)
96: (8)             ``True`` if background rectangles should be added to entries, by
default ``False``.
97: (4)
98: (8)             entries_background_color
99: (8)             Background color of entries if
``add_background_rectangles_to_entries`` is ``True``.
99: (4)             include_background_rectangle

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

100: (8)             ``True`` if the table should have a background rectangle, by default
``False``.
101: (4)             background_rectangle_color
102: (8)             Background color of table if ``include_background_rectangle`` is
``True``.
103: (4)             element_to_mobject
104: (8)             The :class:`~.Mobject` class applied to the table entries. by default
:class:`~.Paragraph`. For common choices, see :mod:`~.text_mobject`/:mod:`~.tex_mobject`.
105: (4)             element_to_mobject_config
106: (8)             Custom configuration passed to :attr:`element_to_mobject`, by default
{}.
107: (4)             arrange_in_grid_config
108: (8)             Dict passed to :meth:`~.Mobject.arrange_in_grid`, customizes the
arrangement of the table.
109: (4)             line_config
110: (8)             Dict passed to :class:`~.Line`, customizes the lines of the table.
111: (4)             kwargs
112: (8)             Additional arguments to be passed to :class:`~.VGroup`.
113: (4)             Examples
114: (4)             -----
115: (4)             .. manim:: TableExamples
116: (8)             :save_last_frame:
117: (8)             class TableExamples(Scene):
118: (12)             def construct(self):
119: (16)                 t0 = Table(
120: (20)                     [["This", "is a"],
121: (20)                     ["simple", "Table in \\n Manim."]])
122: (16)                 t1 = Table(
123: (20)                     [["This", "is a"],
124: (20)                     ["simple", "Table."]],
125: (20)                     row_labels=[Text("R1"), Text("R2")],
126: (20)                     col_labels=[Text("C1"), Text("C2")])
127: (16)                 t1.add_highlighted_cell((2,2), color=YELLOW)
128: (16)                 t2 = Table(
129: (20)                     [["This", "is a"],
130: (20)                     ["simple", "Table."]],
131: (20)                     row_labels=[Text("R1"), Text("R2")],
132: (20)                     col_labels=[Text("C1"), Text("C2")],
133: (20)                     top_left_entry=Star().scale(0.3),
134: (20)                     include_outer_lines=True,
135: (20)                     arrange_in_grid_config={"cell_alignment": RIGHT})
136: (16)                 t2.add(t2.get_cell((2,2), color=RED))
137: (16)                 t3 = Table(
138: (20)                     [["This", "is a"],
139: (20)                     ["simple", "Table."]],
140: (20)                     row_labels=[Text("R1"), Text("R2")],
141: (20)                     col_labels=[Text("C1"), Text("C2")],
142: (20)                     top_left_entry=Star().scale(0.3),
143: (20)                     include_outer_lines=True,
144: (20)                     line_config={"stroke_width": 1, "color": YELLOW})
145: (16)                 t3.remove(*t3.get_vertical_lines())
146: (16)                 g = Group(
147: (20)                     t0,t1,t2,t3
148: (16)                     ).scale(0.7).arrange_in_grid(buff=1)
149: (16)                     self.add(g)
150: (4)             .. manim:: BackgroundRectanglesExample
151: (8)             :save_last_frame:
152: (8)             class BackgroundRectanglesExample(Scene):
153: (12)             def construct(self):
154: (16)                 background = Rectangle(height=6.5, width=13)
155: (16)                 background.set_fill(opacity=.5)
156: (16)                 background.set_color([TEAL, RED, YELLOW])
157: (16)                 self.add(background)
158: (16)                 t0 = Table(
159: (20)                     [["This", "is a"],
160: (20)                     ["simple", "Table."]],
161: (20)                     add_background_rectangles_to_entries=True)
162: (16)                 t1 = Table(
163: (20)                     [["This", "is a"],
```

```

164: (20)                     ["simple", "Table."]],
165: (20)                     include_background_rectangle=True)
166: (16)                     g = Group(t0, t1).scale(0.7).arrange(buff=0.5)
167: (16)                     self.add(g)
168: (4) """
169: (4)     def __init__(
170: (8)         self,
171: (8)         table: Iterable[Iterable[float | str | VMobject]],
172: (8)         row_labels: Iterable[VMobject] | None = None,
173: (8)         col_labels: Iterable[VMobject] | None = None,
174: (8)         top_left_entry: VMobject | None = None,
175: (8)         v_buff: float = 0.8,
176: (8)         h_buff: float = 1.3,
177: (8)         include_outer_lines: bool = False,
178: (8)         add_background_rectangles_to_entries: bool = False,
179: (8)         entries_background_color: ParsableManimColor = BLACK,
180: (8)         include_background_rectangle: bool = False,
181: (8)         background_rectangle_color: ParsableManimColor = BLACK,
182: (8)         element_to_mobject: Callable[
183: (12)             [float | str | VMobject],
184: (12)             VMobject,
185: (8)         ] = Paragraph,
186: (8)         element_to_mobject_config: dict = {},
187: (8)         arrange_in_grid_config: dict = {},
188: (8)         line_config: dict = {},
189: (8)         **kwargs,
190: (4)     ):
191: (8)         self.row_labels = row_labels
192: (8)         self.col_labels = col_labels
193: (8)         self.top_left_entry = top_left_entry
194: (8)         self.row_dim = len(table)
195: (8)         self.col_dim = len(table[0])
196: (8)         self.v_buff = v_buff
197: (8)         self.h_buff = h_buff
198: (8)         self.include_outer_lines = include_outer_lines
199: (8)         self.add_background_rectangles_to_entries =
add_background_rectangles_to_entries
200: (8)             self.entries_background_color = ManimColor(entries_background_color)
201: (8)             self.include_background_rectangle = include_background_rectangle
202: (8)             self.background_rectangle_color =
ManimColor(background_rectangle_color)
203: (8)             self.element_to_mobject = element_to_mobject
204: (8)             self.element_to_mobject_config = element_to_mobject_config
205: (8)             self.arrange_in_grid_config = arrange_in_grid_config
206: (8)             self.line_config = line_config
207: (8)             for row in table:
208: (12)                 if len(row) == len(table[0]):
209: (16)                     pass
210: (12)                 else:
211: (16)                     raise ValueError("Not all rows in table have the same
length.")
212: (8)             super().__init__(**kwargs)
213: (8)             mob_table = self._table_to_mob_table(table)
214: (8)             self.elements_without_labels = VGroup(*it.chain(*mob_table))
215: (8)             mob_table = self._add_labels(mob_table)
216: (8)             self._organize_mob_table(mob_table)
217: (8)             self.elements = VGroup(*it.chain(*mob_table))
218: (8)             if len(self.elements[0].get_all_points()) == 0:
219: (12)                 self.elements.remove(self.elements[0])
220: (8)             self.add(self.elements)
221: (8)             self.center()
222: (8)             self.mob_table = mob_table
223: (8)             self._add_horizontal_lines()
224: (8)             self._add_vertical_lines()
225: (8)             if self.add_background_rectangles_to_entries:
226: (12)                 self.add_background_to_entries(color=self.entries_background_color)
227: (8)                     if self.include_background_rectangle:
228: (12)

```

```

self.add_background_rectangle(color=self.background_rectangle_color)
229: (4)             def _table_to_mob_table(
230: (8)                 self,
231: (8)                 table: Iterable[Iterable[float | str | VMobject]],
232: (4)             ) -> list:
233: (8)                 """Initializes the entries of ``table`` as :class:`~.VMobject` .
234: (8)                 Parameters
235: (8)                 -----
236: (8)                 table
237: (12)                     A 2D array or list of lists. Content of the table has to be a
valid input
238: (12)                     for the callable set in ``element_to_mobobject`` .
239: (8)             Returns
240: (8)             -----
241: (8)             List
242: (12)                 List of :class:`~.VMobject` from the entries of ``table`` .
243: (8)             """
244: (8)             return [
245: (12)                 [
246: (16)                     self.element_to_mobobject(item,
**self.element_to_mobobject_config)
247: (16)                     for item in row
248: (12)                 ]
249: (12)                     for row in table
250: (8)                 ]
def _organize_mob_table(self, table: Iterable[Iterable[VMobject]]) ->
VGroup:
252: (8)             """Arranges the :class:`~.VMobject` of ``table`` in a grid.
253: (8)             Parameters
254: (8)             -----
255: (8)             table
256: (12)                 A 2D iterable object with :class:`~.VMobject` entries.
257: (8)             Returns
258: (8)             -----
259: (8)             :class:`~.VGroup`
260: (12)                 The :class:`~.VMobject` of the ``table`` in a :class:`~.VGroup` .
already
261: (12)                     arranged in a table-like grid.
262: (8)             """
263: (8)             help_table = VGroup()
264: (8)             for i, row in enumerate(table):
265: (12)                 for j, _ in enumerate(row):
266: (16)                     help_table.add(table[i][j])
267: (8)             help_table.arrange_in_grid(
268: (12)                 rows=len(table),
269: (12)                 cols=len(table[0]),
270: (12)                 buff=(self.h_buff, self.v_buff),
271: (12)                 **self.arrange_in_grid_config,
272: (8)             )
273: (8)             return help_table
def _add_labels(self, mob_table: VGroup) -> VGroup:
274: (4)             """Adds labels to an in a grid arranged :class:`~.VGroup` .
275: (8)             Parameters
276: (8)             -----
277: (8)             mob_table
278: (8)                 An in a grid organized class:`~.VGroup` .
279: (12)             Returns the ``mob_table`` with added labels.
280: (8)             Returns
281: (8)             -----
282: (8)             :class:`~.VGroup`
283: (12)             Returns the ``mob_table`` with added labels.
284: (8)             """
285: (8)             if self.row_labels is not None:
286: (12)                 for k in range(len(self.row_labels)):
287: (16)                     mob_table[k] = [self.row_labels[k]] + mob_table[k]
288: (8)             if self.col_labels is not None:
289: (12)                 if self.row_labels is not None:
290: (16)                     if self.top_left_entry is not None:
291: (20)                         col_labels = [self.top_left_entry] + self.col_labels
292: (20)                         mob_table.insert(0, col_labels)

```

```

293: (16)                                else:
294: (20)                                dummy_mobject = get_vectorized_mobject_class()()
295: (20)                                col_labels = [dummy_mobject] + self.col_labels
296: (20)                                mob_table.insert(0, col_labels)
297: (12)                                else:
298: (16)                                mob_table.insert(0, self.col_labels)
299: (8)                                return mob_table
300: (4) def _add_horizontal_lines(self) -> Table:
301: (8)    """Adds the horizontal lines to the table."""
302: (8)    anchor_left = self.get_left()[0] - 0.5 * self.h_buff
303: (8)    anchor_right = self.get_right()[0] + 0.5 * self.h_buff
304: (8)    line_group = VGroup()
305: (8)    if self.include_outer_lines:
306: (12)        anchor = self.get_rows()[0].get_top()[1] + 0.5 * self.v_buff
307: (12)        line = Line(
308: (16)            [anchor_left, anchor, 0], [anchor_right, anchor, 0],
309: (12)            **self.line_config
310: (12)        )
311: (12)        line_group.add(line)
312: (12)        self.add(line)
313: (12)        anchor = self.get_rows()[-1].get_bottom()[1] - 0.5 * self.v_buff
314: (16)        line = Line(
315: (12)            [anchor_left, anchor, 0], [anchor_right, anchor, 0],
316: (12)            **self.line_config
317: (12)        )
318: (8)        line_group.add(line)
319: (12)        self.add(line)
320: (16)        for k in range(len(self.mob_table) - 1):
321: (12)            anchor = self.get_rows()[k + 1].get_top()[1] + 0.5 * (
322: (12)                self.get_rows()[k].get_bottom()[1] - self.get_rows()[k +
323: (16)                    1].get_top()[1]
324: (12)            )
325: (12)            line = Line(
326: (12)                [anchor_left, anchor, 0], [anchor_right, anchor, 0],
327: (8)                **self.line_config
328: (8)            )
329: (4)            line_group.add(line)
330: (8)            self.add(line)
331: (8)            self.horizontal_lines = line_group
332: (8)            return self
333: (8) def _add_vertical_lines(self) -> Table:
334: (8)    """Adds the vertical lines to the table"""
335: (12)    anchor_top = self.get_rows().get_top()[1] + 0.5 * self.v_buff
336: (12)    anchor_bottom = self.get_rows().get_bottom()[1] - 0.5 * self.v_buff
337: (16)    line_group = VGroup()
338: (12)    if self.include_outer_lines:
339: (12)        anchor = self.get_columns()[0].get_left()[0] - 0.5 * self.h_buff
340: (12)        line = Line(
341: (12)            [anchor, anchor_top, 0], [anchor, anchor_bottom, 0],
342: (12)            **self.line_config
343: (16)            )
344: (12)            line_group.add(line)
345: (12)            self.add(line)
346: (12)            anchor = self.get_columns()[-1].get_right()[0] + 0.5 * self.h_buff
347: (8)            line = Line(
348: (12)                [anchor, anchor_top, 0], [anchor, anchor_bottom, 0],
349: (16)                **self.line_config
350: (16)                )
351: (12)                line_group.add(line)
352: (12)                self.add(line)
353: (16)                for k in range(len(self.mob_table[0]) - 1):
354: (12)                    anchor = self.get_columns()[k + 1].get_left()[0] + 0.5 * (

```

```

355: (12)                         line_group.add(line)
356: (12)                         self.add(line)
357: (8)                          self.vertical_lines = line_group
358: (8)                          return self
359: (4)                           def get_horizontal_lines(self) -> VGroup:
360: (8)                             """Return the horizontal lines of the table.
361: (8)                             Returns
362: (8)                             -----
363: (8)                             :class:`~.VGroup`
364: (12)                           :class:`~.VGroup` containing all the horizontal lines of the
table.
365: (8)                           Examples
366: (8)                           -----
367: (8)                           .. manim:: GetHorizontalLinesExample
368: (12)                           :save_last_frame:
369: (12)                           class GetHorizontalLinesExample(Scene):
370: (16)                           def construct(self):
371: (20)                           table = Table(
372: (24)                             [["First", "Second"],
373: (24)                             ["Third", "Fourth"]],
374: (24)                             row_labels=[Text("R1"), Text("R2")],
375: (24)                             col_labels=[Text("C1"), Text("C2")])
376: (20)                           table.get_horizontal_lines().set_color(RED)
377: (20)                           self.add(table)
378: (8)                           """
379: (8)                           return self.horizontal_lines
def get_vertical_lines(self) -> VGroup:
380: (4)                            """Return the vertical lines of the table.
381: (8)                            Returns
382: (8)                            -----
383: (8)                            :class:`~.VGroup`
384: (8)                            :class:`~.VGroup` containing all the vertical lines of the table.
Examples
-----
.. manim:: GetVerticalLinesExample
389: (12)                           :save_last_frame:
390: (12)                           class GetVerticalLinesExample(Scene):
391: (16)                           def construct(self):
392: (20)                           table = Table(
393: (24)                             [["First", "Second"],
394: (24)                             ["Third", "Fourth"]],
395: (24)                             row_labels=[Text("R1"), Text("R2")],
396: (24)                             col_labels=[Text("C1"), Text("C2")])
397: (20)                           table.get_vertical_lines()[0].set_color(RED)
398: (20)                           self.add(table)
399: (8)                           """
400: (8)                           return self.vertical_lines
def get_columns(self) -> VGroup:
401: (4)                            """Return columns of the table as a :class:`~.VGroup` of
402: (8)                            :class:`~.VGroup`.
:class:`~.VGroup` .
403: (8)                           Returns
404: (8)                           -----
405: (8)                           :class:`~.VGroup`
406: (12)                           :class:`~.VGroup` containing each column in a :class:`~.VGroup`.
Examples
-----
.. manim:: GetColumnsExample
410: (12)                           :save_last_frame:
411: (12)                           class GetColumnsExample(Scene):
412: (16)                           def construct(self):
413: (20)                           table = Table(
414: (24)                             [["First", "Second"],
415: (24)                             ["Third", "Fourth"]],
416: (24)                             row_labels=[Text("R1"), Text("R2")],
417: (24)                             col_labels=[Text("C1"), Text("C2")])
418: (20)                           table.add(SurroundingRectangle(table.get_columns()[1]))
419: (20)                           self.add(table)
420: (8)                           """
421: (8)                           return VGroup(

```

```

422: (12) *
423: (16)     VGroup(*(row[i] for row in self.mob_table))
424: (16)         for i in range(len(self.mob_table[0]))
425: (12)     )
426: (8)   )
427: (4) def get_rows(self) -> VGroup:
428: (8)     """Return the rows of the table as a :class:`~.VGroup` of
:class:`~.VGroup`'s.
429: (8)     Returns
430: (8)     -----
431: (8)     :class:`~.VGroup`'s
432: (12)     :class:`~.VGroup` containing each row in a :class:`~.VGroup`'.
433: (8) Examples
434: (8) -----
435: (8) .. manim:: GetRowsExample
436: (12)     :save_last_frame:
437: (12)     class GetRowsExample(Scene):
438: (16)         def construct(self):
439: (20)             table = Table(
440: (24)                 [["First", "Second"],
441: (24)                 ["Third", "Fourth"]],
442: (24)                 row_labels=[Text("R1"), Text("R2")],
443: (24)                 col_labels=[Text("C1"), Text("C2")])
444: (20)             table.add(SurroundingRectangle(table.get_rows()[1]))
445: (20)             self.add(table)
446: (8)     """
447: (8)     return VGroup(*(VGroup(*row) for row in self.mob_table))
448: (4) def set_column_colors(self, *colors: Iterable[ParsableManimColor]) ->
Table:
449: (8)     """Set individual colors for each column of the table.
450: (8) Parameters
451: (8) -----
452: (8) colors
453: (12)     An iterable of colors; each color corresponds to a column.
454: (8) Examples
455: (8) -----
456: (8) .. manim:: SetColumnColorsExample
457: (12)     :save_last_frame:
458: (12)     class SetColumnColorsExample(Scene):
459: (16)         def construct(self):
460: (20)             table = Table(
461: (24)                 [["First", "Second"],
462: (24)                 ["Third", "Fourth"]],
463: (24)                 row_labels=[Text("R1"), Text("R2")],
464: (24)                 col_labels=[Text("C1"), Text("C2")])
465: (20)             ).set_column_colors([RED,BLUE], GREEN)
466: (20)             self.add(table)
467: (8)     """
468: (8)     columns = self.get_columns()
469: (8)     for color, column in zip(colors, columns):
470: (12)         column.set_color(color)
471: (8)     return self
472: (4) def set_row_colors(self, *colors: Iterable[ParsableManimColor]) -> Table:
473: (8)     """Set individual colors for each row of the table.
474: (8) Parameters
475: (8) -----
476: (8) colors
477: (12)     An iterable of colors; each color corresponds to a row.
478: (8) Examples
479: (8) -----
480: (8) .. manim:: SetRowColorsExample
481: (12)     :save_last_frame:
482: (12)     class SetRowColorsExample(Scene):
483: (16)         def construct(self):
484: (20)             table = Table(
485: (24)                 [["First", "Second"],
486: (24)                 ["Third", "Fourth"]],
487: (24)                 row_labels=[Text("R1"), Text("R2")],
488: (24)                 col_labels=[Text("C1"), Text("C2")])

```

```

489: (20)                               ).set_row_colors([RED,BLUE], GREEN)
490: (20)                               self.add(table)
491: (8)      """
492: (8)      rows = self.get_rows()
493: (8)      for color, row in zip(colors, rows):
494: (12)          row.set_color(color)
495: (8)      return self
496: (4)      def get_entries(
497: (8)          self,
498: (8)          pos: Sequence[int] | None = None,
499: (4)      ) -> VMobject | VGroup:
500: (8)          """Return the individual entries of the table (including labels) or
one specific entry
501: (8)          if the parameter, ``pos``, is set.
502: (8)          Parameters
503: (8)          -----
504: (8)          pos
505: (12)              The position of a specific entry on the table. ``(1,1)`` being the
top left entry
506: (12)
507: (8)
508: (8)
509: (8)
510: (12)      Union[:class:`~.VMobject`, :class:`~.VGroup`]
511: (12)          :class:`~.VGroup` containing all entries of the table (including
labels)
512: (12)          or the :class:`~.VMobject` at the given position if ``pos`` is
set.
513: (8)
514: (8)
515: (12)
516: (12)
517: (16)
518: (20)
519: (24)
520: (24)
521: (24)
522: (24)
523: (20)
524: (20)
525: (24)
526: (20)
527: (20)
528: (8)
529: (8)
530: (12)
531: (16)
532: (16)
533: (16)
534: (12)
535: (16)
536: (16)
537: (12)
538: (16)
539: (16)
540: (8)
541: (12)
542: (4)      if pos is not None:
543: (8)          if (
544: (8)              self.row_labels is not None
545: (8)              and self.col_labels is not None
546: (8)              and self.top_left_entry is None
547: (8)          ):
548: (8)              index = len(self.mob_table[0]) * (pos[0] - 1) + pos[1] - 2
549: (8)              return self.elements[index]
550: (8)
551: (12)          else:
552: (8)              index = len(self.mob_table[0]) * (pos[0] - 1) + pos[1] - 1
553: (8)              return self.elements[index]
554: (8)
555: (12)
556: (4)      else:
557: (8)          return self.elements
558: (4)      def get_entries_without_labels(
559: (8)          self,
560: (8)          pos: Sequence[int] | None = None,
561: (4)      ) -> VMobject | VGroup:
562: (8)          """Return the individual entries of the table (without labels) or one
specific entry
563: (8)          if the parameter, ``pos``, is set.
564: (8)          Parameters
565: (8)          -----
566: (8)          pos
567: (8)              The position of a specific entry on the table. ``(1,1)`` being the
top left entry

```

```

552: (12)                                of the table (without labels).
553: (8)                                 Returns
554: (8)
555: (8)                                Union[:class:`~.VMobject`, :class:`~.VGroup`]
556: (12)                                :class:`~.VGroup` containing all entries of the table (without
labels)
557: (12)                                or the :class:`~.VMobject` at the given position if ``pos`` is
set.
558: (8)                                 Examples
559: (8)
560: (8)                                -----
561: (12)                                .. manim:: GetEntriesWithoutLabelsExample
562: (12)                                :save_last_frame:
563: (16)                                class GetEntriesWithoutLabelsExample(Scene):
564: (20)                                def construct(self):
565: (24)                                table = Table(
566: (24)                                [[ "First", "Second" ],
567: (24)                                [ "Third", "Fourth" ]],
568: (24)                                row_labels=[Text("R1"), Text("R2")],
569: (20)                                col_labels=[Text("C1"), Text("C2")])
570: (20)                                ent = table.get_entries_without_labels()
571: (20)                                colors = [BLUE, GREEN, YELLOW, RED]
572: (24)                                for k in range(len(colors)):
573: (20)                                ent[k].set_color(colors[k])
574: (20)                                table.get_entries_without_labels((2,2)).rotate(PI)
575: (8)                                self.add(table)
576: (8)
577: (12)                                """
578: (12)                                if pos is not None:
579: (8)                                index = self.col_dim * (pos[0] - 1) + pos[1] - 1
580: (12)                                return self.elements_without_labels[index]
581: (4)                                else:
582: (8)                                return self.elements_without_labels
583: (8)
584: (8)
585: (8)                                def get_row_labels(self) -> VGroup:
586: (12)                                """
587: (8)                                Return the row labels of the table.
588: (8)                                Returns
589: (8)
590: (12)                                :class:`~.VGroup` -
591: (12)                                :class:`~.VGroup` containing the row labels of the table.
592: (16)                                Examples
593: (20)
594: (24)
595: (24)
596: (24)
597: (24)
598: (20)
599: (20)
600: (24)
601: (20)
602: (8)
603: (8)
604: (4)                                def get_col_labels(self) -> VGroup:
605: (8)                                """
606: (8)                                Return the column labels of the table.
607: (8)
608: (8)
609: (12)                                Returns
610: (8)
611: (8)
612: (8)
613: (12)                                :class:`~.VGroup` -
614: (12)                                VGroup containing the column labels of the table.
615: (16)                                Examples
616: (20)
617: (24)
618: (24)

```

```

619: (24)                                row_labels=[Text("R1"), Text("R2")],
620: (24)                                col_labels=[Text("C1"), Text("C2")])
621: (20)                                lab = table.get_col_labels()
622: (20)                                for item in lab:
623: (24)                                    item.set_color(random_bright_color())
624: (20)                                self.add(table)
625: (8)                                """
626: (8)                                return VGroup(*self.col_labels)
627: (4) def get_labels(self) -> VGroup:
628: (8)    """Returns the labels of the table.
629: (8)    Returns
630: (8)    -----
631: (8)    :class:`~.VGroup`
632: (12)    :class:`~.VGroup` containing all the labels of the table.
633: (8) Examples
634: (8) -----
635: (8) .. manim:: GetLabelsExample
636: (12)    :save_last_frame:
637: (12)    class GetLabelsExample(Scene):
638: (16)        def construct(self):
639: (20)            table = Table(
640: (24)                [["First", "Second"],
641: (24)                ["Third", "Fourth"]],
642: (24)                row_labels=[Text("R1"), Text("R2")],
643: (24)                col_labels=[Text("C1"), Text("C2")])
644: (20)            lab = table.get_labels()
645: (20)            colors = [BLUE, GREEN, YELLOW, RED]
646: (20)            for k in range(len(colors)):
647: (24)                lab[k].set_color(colors[k])
648: (20)            self.add(table)
649: (8)        """
650: (8)        label_group = VGroup()
651: (8)        if self.top_left_entry is not None:
652: (12)            label_group.add(self.top_left_entry)
653: (8)        for label in (self.col_labels, self.row_labels):
654: (12)            if label is not None:
655: (16)                label_group.add(*label)
656: (8)        return label_group
657: (4) def add_background_to_entries(self, color: ParsableManimColor = BLACK) ->
Table:
658: (8)    """Adds a black :class:`~.BackgroundRectangle` to each entry of the
table."""
659: (8)
660: (12)
661: (8)
662: (4) def get_cell(self, pos: Sequence[int] = (1, 1), **kwargs) -> Polygon:
663: (8)    """Returns one specific cell as a rectangular :class:`~.Polygon` without the entry.
664: (8) Parameters
665: (8) -----
666: (8) pos
667: (12)    The position of a specific entry on the table. ``(1,1)`` being the
top left entry
668: (12)    of the table.
669: (8) kwargs
670: (12)    Additional arguments to be passed to :class:`~.Polygon`.
671: (8) Returns
672: (8) -----
673: (8) :class:`~.Polygon`
674: (12)    Polygon mimicking one specific cell of the Table.
675: (8) Examples
676: (8) -----
677: (8) .. manim:: GetCellExample
678: (12)    :save_last_frame:
679: (12)    class GetCellExample(Scene):
680: (16)        def construct(self):
681: (20)            table = Table(
682: (24)                [["First", "Second"],
683: (24)                ["Third", "Fourth"]],
```

```

684: (24)                                row_labels=[Text("R1"), Text("R2")],
685: (24)                                col_labels=[Text("C1"), Text("C2")])
686: (20)                                cell = table.get_cell((2,2), color=RED)
687: (20)                                self.add(table, cell)
688: (8)                                 """
689: (8)                                row = self.get_rows()[pos[0] - 1]
690: (8)                                col = self.get_columns()[pos[1] - 1]
691: (8)                                edge_UL = [
692: (12)                                  col.get_left()[0] - self.h_buff / 2,
693: (12)                                  row.get_top()[1] + self.v_buff / 2,
694: (12)                                  0,
695: (8) ]
696: (8)                                edge_UR = [
697: (12)                                  col.get_right()[0] + self.h_buff / 2,
698: (12)                                  row.get_top()[1] + self.v_buff / 2,
699: (12)                                  0,
700: (8) ]
701: (8)                                edge_DL = [
702: (12)                                  col.get_left()[0] - self.h_buff / 2,
703: (12)                                  row.get_bottom()[1] - self.v_buff / 2,
704: (12)                                  0,
705: (8) ]
706: (8)                                edge_DR = [
707: (12)                                  col.get_right()[0] + self.h_buff / 2,
708: (12)                                  row.get_bottom()[1] - self.v_buff / 2,
709: (12)                                  0,
710: (8) ]
711: (8)                                rec = Polygon(edge_UL, edge_UR, edge_DR, edge_DL, **kwargs)
712: (8)                                return rec
713: (4) def get_highlighted_cell(
714: (8)     self, pos: Sequence[int] = (1, 1), color: ParsableManimColor = YELLOW,
**kwargs
715: (4) )
716: (8)     -> BackgroundRectangle:
717: (8)         """Returns a :class:`~.BackgroundRectangle` of the cell at the given
position.
718: (8)
719: (8)
720: (12) top left entry
721: (12)     Parameters
722: (8)     -----
723: (12)     pos
724: (8)         The position of a specific entry on the table. ``(1,1)`` being the
725: (12)         of the table.
726: (8)     color
727: (8)         The color used to highlight the cell.
728: (8)     kwargs
729: (12)         Additional arguments to be passed to
730: (12)         :class:`~.BackgroundRectangle`.
731: (16) Examples
732: (20)     -----
733: (24)     .. manim:: GetHighlightedCellExample
734: (24)         :save_last_frame:
735: (24)         class GetHighlightedCellExample(Scene):
736: (24)             def construct(self):
737: (20)                 table = Table(
738: (20)                     [["First", "Second"],
739: (20)                         ["Third", "Fourth"]],
740: (20)                         row_labels=[Text("R1"), Text("R2")],
741: (20)                         col_labels=[Text("C1"), Text("C2")])
742: (20)                     highlight = table.get_highlighted_cell((2,2), color=GREEN)
743: (20)                     table.add_to_back(highlight)
744: (20)                     self.add(table)
745: (20)                     """
746: (4)             cell = self.get_cell(pos)
747: (4)             bg_cell = BackgroundRectangle(cell, color=ManimColor(color), **kwargs)
748: (4)             return bg_cell
749: (4) def add_highlighted_cell(
750: (8)     self, pos: Sequence[int] = (1, 1), color: ParsableManimColor = YELLOW,
**kwargs
751: (4) )
752: (8)     -> Table:
753: (8)         """Highlights one cell at a specific position on the table by adding a

```

```

:class:`~.BackgroundRectangle`.
748: (8)          Parameters
749: (8)          -----
750: (8)          pos
751: (12)         The position of a specific entry on the table. ``(1,1)`` being the
top left entry
752: (12)         of the table.
753: (8)
754: (12)         color
755: (8)         The color used to highlight the cell.
756: (12)         kwargs
757: (8)         Additional arguments to be passed to
:class:`~.BackgroundRectangle` .
758: (8)          Examples
759: (8)          -----
760: (12)         .. manim:: AddHighlightedCellExample
761: (12)             :save_last_frame:
762: (16)             class AddHighlightedCellExample(Scene):
763: (20)                 def construct(self):
764: (24)                     table = Table(
765: (24)                         [["First", "Second"],
766: (24)                         ["Third", "Fourth"]],
767: (24)                         row_labels=[Text("R1"), Text("R2")],
768: (20)                         col_labels=[Text("C1"), Text("C2")])
769: (20)                     table.add_highlighted_cell((2,2), color=GREEN)
770: (8)                     self.add(table)
771: (8)             """
772: (8)             bg_cell = self.get_highlighted_cell(pos, color=ManimColor(color),
773: (8)             **kwargs)
774: (8)             self.add_to_back(bg_cell)
775: (8)             entry = self.get_entries(pos)
776: (4)             entry.background_rectangle = bg_cell
777: (8)             return self
778: (4)             def create(
779: (8)                 self,
780: (8)                 lag_ratio: float = 1,
781: (8)                 line_animation: Callable[[VMobject | VGroup], Animation] = Create,
782: (8)                 label_animation: Callable[[VMobject | VGroup], Animation] = Write,
783: (8)                 element_animation: Callable[[VMobject | VGroup], Animation] = Create,
784: (4)                 entry_animation: Callable[[VMobject | VGroup], Animation] = FadeIn,
785: (8)                 **kwargs,
786: (8)             ) -> AnimationGroup:
787: (8)                 """Customized create-type function for tables.
788: (8)                 Parameters
789: (8)                 -----
790: (8)                 lag_ratio
791: (12)                 The lag ratio of the animation.
792: (8)                 line_animation
793: (12)                 The animation style of the table lines, see :mod:`~.creation` for
examples.
794: (8)                 label_animation
795: (12)                 The animation style of the table labels, see :mod:`~.creation` for
examples.
796: (8)                 element_animation
797: (12)                 The animation style of the table elements, see :mod:`~.creation` for
examples.
798: (8)                 entry_animation
799: (12)                 The entry animation of the table background, see :mod:`~.creation` for
examples.
800: (8)                 kwargs
801: (8)                 Further arguments passed to the creation animations.
802: (8)
803: (12)                 Returns
804: (8)                 -----
805: (8)                 :class:`~.AnimationGroup`
806: (8)                 AnimationGroup containing creation of the lines and of the
elements.
807: (12)                 Examples
808: (8)                 -----
809: (8)                 .. manim:: CreateTableExample
810: (12)                   class CreateTableExample(Scene):

```

```

808: (16)             def construct(self):
809: (20)                 table = Table(
810: (24)                     [["First", "Second"],
811: (24)                         ["Third", "Fourth"]],
812: (24)                     row_labels=[Text("R1"), Text("R2")],
813: (24)                     col_labels=[Text("C1"), Text("C2")],
814: (24)                     include_outer_lines=True)
815: (20)                     self.play(table.create())
816: (20)                     self.wait()
817: (8)                 """
818: (8)             animations: Sequence[Animation] = [
819: (12)                 line_animation(
820: (16)                     VGroup(self.vertical_lines, self.horizontal_lines),
821: (16)                     **kwargs,
822: (12)                 ),
823: (12)                 element_animation(self.elements_without_labels.set_z_index(2),
824: (8)                     **kwargs),
825: (8)             ]
826: (12)             if self.get_labels():
827: (16)                 animations += [
828: (12)                     label_animation(self.get_labels(), **kwargs),
829: (12)                 ]
830: (16)             if self.get_entries():
831: (12)                 for entry in self.elements_without_labels:
832: (16)                     try:
833: (20)                         animations += [
834: (28)                             entry_animation(
835: (28)                                 entry.background_rectangle,
836: (24)                                 **kwargs,
837: (20)                             )
838: (16)                         ]
839: (20)                     except AttributeError:
840: (16)                         continue
841: (4)             return AnimationGroup(*animations, lag_ratio=lag_ratio)
842: (8)         def scale(self, scale_factor: float, **kwargs):
843: (8)             self.h_buff *= scale_factor
844: (8)             self.v_buff *= scale_factor
845: (8)             super().scale(scale_factor, **kwargs)
846: (0)             return self
847: (4)         class MathTable(Table):
848: (4)             """A specialized :class:`~.Table` mobject for use with LaTeX.
849: (4)             Examples
850: (4)             -----
851: (8)             .. manim:: MathTableExample
852: (8)                 :save_last_frame:
853: (12)                 class MathTableExample(Scene):
854: (16)                     def construct(self):
855: (20)                         t0 = MathTable(
856: (20)                             [[ "+", 0, 5, 10],
857: (20)                               [0, 0, 5, 10],
858: (20)                               [2, 2, 7, 12],
859: (20)                               [4, 4, 9, 14]],
860: (16)                               include_outer_lines=True)
861: (4)                         self.add(t0)
862: (4)             """
863: (8)             def __init__(
864: (8)                 self,
865: (8)                 table: Iterable[Iterable[float | str]],
866: (8)                 element_to_mobject: Callable[[float | str], VMobject] = MathTex,
867: (8)                 **kwargs,
868: (4)             ):
869: (8)                 """
870: (8)             Special case of :class:`~.Table` with `element_to_mobject` set to
871: (8)             Every entry in `table` is set in a Latex `align` environment.
872: (8)             Parameters
873: (8)             -----
874: (12)                 table
875: (12)             A 2d array or list of lists. Content of the table have to be valid

```

```

input
875: (12)           for :class:`~.MathTex`.
876: (8)            element_to_mobject
877: (12)           The :class:`~.Mobject` class applied to the table entries. Set as
:class:`~.MathTex`.
878: (8)            kwargs
879: (12)           Additional arguments to be passed to :class:`~.Table`.
880: (8)            """
881: (8)            super().__init__(
882: (12)              table,
883: (12)              element_to_mobject=element_to_mobject,
884: (12)              **kwargs,
885: (8)            )
886: (0)             class MobjectTable(Table):
887: (4)               """A specialized :class:`~.Table` mobject for use with :class:`~.Mobject` .
888: (4)               Examples
889: (4)               -----
890: (4)               .. manim:: MobjectTableExample
891: (8)                 :save_last_frame:
892: (8)                 class MobjectTableExample(Scene):
893: (12)                   def construct(self):
894: (16)                     cross = VGroup(
895: (20)                       Line(UP + LEFT, DOWN + RIGHT),
896: (20)                       Line(UP + RIGHT, DOWN + LEFT),
897: (16)
898: (16)
899: (16)
900: (16)
901: (20)
902: (20)
903: (20)
904: (16)
905: (16)
906: (20)
907: (16)
908: (16)
909: (4)
910: (4)             """
911: (8)             def __init__(
912: (8)               self,
913: (8)               table: Iterable[Iterable[VMobject]],
914: (8)               element_to_mobject: Callable[[VMobject], VMobject] = lambda m: m,
915: (4)               **kwargs,
916: (8)             ):
917: (8)               """
918: (8)               identity function.
919: (8)
920: (8)
921: (8)
922: (12)
923: (8)
924: (12)
925: (8)
926: (12)
927: (8)
928: (8)
929: (0)             **kwargs)
930: (4)             class IntegerTable(Table):
931: (4)               """A specialized :class:`~.Table` mobject for use with :class:`~.Integer` .
932: (4)               Examples
933: (4)               -----
934: (4)               .. manim:: IntegerTableExample
935: (8)                 :save_last_frame:
936: (12)                 class IntegerTableExample(Scene):

```

```

937: (16)             t0 = IntegerTable(
938: (20)             [[0,30,45,60,90],
939: (20)             [90,60,45,30,0]],
940: (20)             col_labels=[
941: (24)                 MathTex("\\frac{\\sqrt{0}}{2}"),
942: (24)                 MathTex("\\frac{\\sqrt{1}}{2}"),
943: (24)                 MathTex("\\frac{\\sqrt{2}}{2}"),
944: (24)                 MathTex("\\frac{\\sqrt{3}}{2}"),
945: (24)                 MathTex("\\frac{\\sqrt{4}}{2}"]),
946: (20)             row_labels=[MathTex("\sin"), MathTex("\cos")],
947: (20)             h_buff=1,
948: (20)             element_to_mobject_config={"unit": "\circ")
949: (16)             self.add(t0)
950: (4)
951: (4)         """
952: (8)         def __init__(
953: (8)             self,
954: (8)             table: Iterable[Iterable[float | str]],
955: (8)             element_to_mobject: Callable[[float | str], VMobject] = Integer,
956: (8)             **kwargs,
957: (8)         ):
958: (8)             """
959: (8)             Special case of :class:`~.Table` with `element_to_mobject` set to
960: (8)             Will round if there are decimal entries in the table.
961: (8)             Parameters
962: (8)             -----
963: (12)             table
964: (12)                 A 2d array or list of lists. Content of the table has to be valid
965: (8)
966: (12)             for :class:`~.Integer`.
967: (8)             element_to_mobject
968: (12)                 The :class:`~.Mobject` class applied to the table entries. Set as
969: (8)
970: (8)             **kwargs
971: (0)             """
972: (4)             """
973: (4)             A specialized :class:`~.Table` mobject for use with
974: (4)             :class:`~.DecimalNumber` to display decimal entries.
975: (4)             Examples
976: (8)
977: (8)             .. manim:: DecimalTableExample
978: (12)             :save_last_frame:
979: (16)             class DecimalTableExample(Scene):
980: (16)                 def construct(self):
981: (16)                     x_vals = [-2,-1,0,1,2]
982: (20)                     y_vals = np.exp(x_vals)
983: (20)                     t0 = DecimalTable(
984: (20)                         [x_vals, y_vals],
985: (20)                         row_labels=[MathTex("x"), MathTex("f(x)=e^{x}")],
986: (16)                         h_buff=1,
987: (4)                         element_to_mobject_config={"num_decimal_places": 2})
988: (4)                     self.add(t0)
989: (8)
990: (8)             """
991: (8)             def __init__(
992: (8)                 self,
993: (8)                 table: Iterable[Iterable[float | str]],
994: (8)                 element_to_mobject: Callable[[float | str], VMobject] = DecimalNumber,
995: (8)                 element_to_mobject_config: dict = {"num_decimal_places": 1},
996: (8)                 **kwargs,
997: (8)             ):
998: (8)             """
999: (8)             Special case of :class:`~.Table` with ``element_to_mobject`` set to
999: (8)             :class:`~.DecimalNumber`.
999: (8)             By default, ``num_decimal_places`` is set to 1.
999: (8)             Will round/truncate the decimal places based on the provided
999: (8)             ``element_to_mobject_config``.
```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
999: (8)          Parameters
1000: (8)          -----
1001: (8)          table
1002: (12)         A 2D array, or a list of lists. Content of the table must be valid
input
1003: (12)         for :class:`~.DecimalNumber`.
1004: (8)          element_to_mobject
1005: (12)         The :class:`~.Mobject` class applied to the table entries. Set as
:class:`~.DecimalNumber`.
1006: (8)
1007: (12)         element_to_mobject_config
1008: (8)         Element to mobject config, here set as {"num_decimal_places": 1}.
1009: (12)         kwargs
1010: (8)         Additional arguments to be passed to :class:`~.Table`.
1011: (8)
1012: (12)         super().__init__(
1013: (12)             table,
1014: (12)             element_to_mobject=element_to_mobject,
1015: (12)             element_to_mobject_config=element_to_mobject_config,
1016: (8)             **kwargs,
)
-----
```

## File 75 - brace.py:

```
1: (0)          """Mobject representing curly braces."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = ["Brace", "BraceLabel", "ArcBrace", "BraceText",
"BraceBetweenPoints"]
4: (0)          from typing import Sequence
5: (0)          import numpy as np
6: (0)          import svgelements as se
7: (0)          from manim._config import config
8: (0)          from manim.mobject.geometry.arc import Arc
9: (0)          from manim.mobject.geometry.line import Line
10: (0)         from manim.mobject.mobject import Mobject
11: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
12: (0)         from manim.mobject.text.tex_mobject import MathTex, Tex
13: (0)         from ...animation.composition import AnimationGroup
14: (0)         from ...animation.fading import FadeIn
15: (0)         from ...animation.growing import GrowFromCenter
16: (0)         from ...constants import *
17: (0)         from ...mobject.types.vectorized_mobject import VMobject
18: (0)         from ...utils.color import BLACK
19: (0)         from ..svg.svg_mobject import VMobjectFromSVGPath
20: (0)         __all__ = ["Brace", "BraceBetweenPoints", "BraceLabel", "ArcBrace"]
21: (0)         class Brace(VMobjectFromSVGPath):
22: (4)             """Takes a mobject and draws a brace adjacent to it.
23: (4)             Passing a direction vector determines the direction from which the
24: (4)             brace is drawn. By default it is drawn from below.
25: (4)             Parameters
26: (4)             -----
27: (4)             mobject
28: (8)                 The mobject adjacent to which the brace is placed.
29: (4)             direction :
30: (8)                 The direction from which the brace faces the mobject.
31: (4)             See Also
32: (4)             -----
33: (4)             :class:`BraceBetweenPoints`
34: (4)             Examples
35: (4)             -----
36: (4)             .. manim:: BraceExample
37: (8)                 :save_last_frame:
38: (8)                 class BraceExample(Scene):
39: (12)                     def construct(self):
40: (16)                         s = Square()
41: (16)                         self.add(s)
42: (16)                         for i in np.linspace(0.1,1.0,4):
43: (20)                             br = Brace(s, sharpness=i)
```

```

44: (20)                     t = Text(f"sharpness= {i}").next_to(br, RIGHT)
45: (20)                     self.add(t)
46: (20)                     self.add(br)
47: (16)             VGroup(*self.mobjects).arrange(DOWN, buff=0.2)
48: (4)             """
49: (4)     def __init__(
50: (8)         self,
51: (8)         mobject: Mobject,
52: (8)         direction: Sequence[float] | None = DOWN,
53: (8)         buff=0.2,
54: (8)         sharpness=2,
55: (8)         stroke_width=0,
56: (8)         fill_opacity=1.0,
57: (8)         background_stroke_width=0,
58: (8)         background_stroke_color=BLACK,
59: (8)         **kwargs,
60: (4)     ):
61: (8)         path_string_template = (
62: (12)             "m0.01216 0c-0.01152 0-0.01216 6.103e-4 -0.01216
0.01311v0.007762c0.06776 "
63: (12)             "0.122 0.1799 0.1455 0.2307 0.1455h{0}c0.03046 3.899e-4 0.07964
0.00449 "
64: (12)             "0.1246 0.02636 0.0537 0.02695 0.07418 0.05816 0.08648 0.07769
0.001562 "
65: (12)             "0.002538 0.004539 0.002563 0.01098 0.002563 0.006444-2e-8
0.009421-2.47e-"
66: (12)             "5 0.01098-0.002563 0.0123-0.01953 0.03278-0.05074 0.08648-0.07769
0.04491"
67: (12)             "-0.02187 0.09409-0.02597 0.1246-0.02636h{0}c0.05077 0 0.1629-
0.02346 "
68: (12)             "0.2307-0.1455v-0.007762c-1.78e-6 -0.0125-6.365e-4 -0.01311-
0.01216-0.0131"
69: (12)             "1-0.006444-3.919e-8 -0.009348 2.448e-5 -0.01091 0.002563-0.0123
0.01953-"
70: (12)             "0.03278 0.05074-0.08648 0.07769-0.04491 0.02187-0.09416 0.02597-
0.1246 "
71: (12)             "0.02636h{1}c-0.04786 0-0.1502 0.02094-0.2185 0.1256-0.06833-
0.1046-0.1706"
72: (12)             "-0.1256-0.2185-0.1256h{1}c-0.03046-3.899e-4 -0.07972-0.004491-
0.1246-0.02"
73: (12)             "636-0.0537-0.02695-0.07418-0.05816-0.08648-0.07769-0.001562-
0.002538-"
74: (12)             "0.004467-0.002563-0.01091-0.002563z"
75: (8)         )
76: (8)         default_min_width = 0.90552
77: (8)         self.buff = buff
78: (8)         angle = -np.arctan2(*direction[:2]) + np.pi
79: (8)         mobject.rotate(-angle, about_point=ORIGIN)
80: (8)         left = mobject.get_corner(DOWN + LEFT)
81: (8)         right = mobject.get_corner(DOWN + RIGHT)
82: (8)         target_width = right[0] - left[0]
83: (8)         linear_section_length = max(
84: (12)             0,
85: (12)             (target_width * sharpness - default_min_width) / 2,
86: (8)         )
87: (8)         path = se.Path(
88: (12)             path_string_template.format(
89: (16)                 linear_section_length,
90: (16)                 -linear_section_length,
91: (12)             )
92: (8)         )
93: (8)         super().__init__(
94: (12)             path_obj=path,
95: (12)             stroke_width=stroke_width,
96: (12)             fill_opacity=fill_opacity,
97: (12)             background_stroke_width=background_stroke_width,
98: (12)             background_stroke_color=background_stroke_color,
99: (12)             **kwargs,
100: (8)         )

```

```

101: (8)                     self.flip(RIGHT)
102: (8)                     self.stretch_to_fit_width(target_width)
103: (8)                     self.shift(left - self.get_corner(UP + LEFT) + selfbuff * DOWN)
104: (8)                     for mob in mobject, self:
105: (12)                         mob.rotate(angle, about_point=ORIGIN)
106: (4)                     def put_at_tip(self, mob, use_next_to=True, **kwargs):
107: (8)                         if use_next_to:
108: (12)                             mob.next_to(self.get_tip(), np.round(self.get_direction())),
109: (8)                         else:
110: (12)                             mob.move_to(self.get_tip())
111: (12)                             buff = kwargs.get("buff", DEFAULT_MOBJECT_TO_MOBJECT_BUFFER)
112: (12)                             shift_distance = mob.width / 2.0 + buff
113: (12)                             mob.shift(self.get_direction() * shift_distance)
114: (8)                     return self
115: (4)                     def get_text(self, *text, **kwargs):
116: (8)                         text_mob = Tex(*text)
117: (8)                         self.put_at_tip(text_mob, **kwargs)
118: (8)                         return text_mob
119: (4)                     def get_tex(self, *tex, **kwargs):
120: (8)                         tex_mob = MathTex(*tex)
121: (8)                         self.put_at_tip(tex_mob, **kwargs)
122: (8)                         return tex_mob
123: (4)                     def get_tip(self):
124: (8)                         if config["renderer"] == "opengl":
125: (12)                             return self.points[34]
126: (8)                             return self.points[28] # = 7*4
127: (4)                     def get_direction(self):
128: (8)                         vect = self.get_tip() - self.get_center()
129: (8)                         return vect / np.linalg.norm(vect)
130: (0)                     class BraceLabel(VMobject, metaclass=ConvertToOpenGL):
131: (4)                         """Create a brace with a label attached.
132: (4)                         Parameters
133: (4)                         -----
134: (4)                         obj
135: (8)                             The mobject adjacent to which the brace is placed.
136: (4)                         text
137: (8)                             The label text.
138: (4)                         brace_direction
139: (8)                             The direction of the brace. By default ``DOWN``.
140: (4)                         label_constructor
141: (8)                             A class or function used to construct a mobject representing
142: (8)                             the label. By default :class:`~.MathTex` .
143: (4)                         font_size
144: (8)                             The font size of the label, passed to the ``label_constructor``.
145: (4)                         buff
146: (8)                             The buffer between the mobject and the brace.
147: (4)                         brace_config
148: (8)                             Arguments to be passed to :class:`.Brace` .
149: (4)                         kwargs
150: (8)                             Additional arguments to be passed to :class:`~.VMobject` .
151: (4)                         """
152: (4)                     def __init__(
153: (8)                         self,
154: (8)                         obj: Mobject,
155: (8)                         text: str,
156: (8)                         brace_direction: np.ndarray = DOWN,
157: (8)                         label_constructor: type = MathTex,
158: (8)                         font_size: float = DEFAULT_FONT_SIZE,
159: (8)                         buff: float = 0.2,
160: (8)                         brace_config: dict | None = None,
161: (8)                         **kwargs,
162: (4) ):
163: (8)                         self.label_constructor = label_constructor
164: (8)                         super().__init__(**kwargs)
165: (8)                         self.brace_direction = brace_direction
166: (8)                         if brace_config is None:
167: (12)                             brace_config = {}
168: (8)                         self.brace = Brace(obj, brace_direction, buff, **brace_config)

```

```

169: (8)             if isinstance(text, (tuple, list)):
170: (12)             self.label = self.label_constructor(font_size=font_size, *text,
**kwargs)
171: (8)         else:
172: (12)             self.label = self.label_constructor(str(text),
font_size=font_size)
173: (8)             self.brace.put_at_tip(self.label)
174: (8)             self.add(self.brace, self.label)
175: (4)         def creation_anim(self, label_anim=FadeIn, brace_anim=GrowFromCenter):
176: (8)             return AnimationGroup(brace_anim(self.brace), label_anim(self.label))
177: (4)         def shift_brace(self, obj, **kwargs):
178: (8)             if isinstance(obj, list):
179: (12)                 obj = self.get_group_class()(*obj)
180: (8)             self.brace = Brace(obj, self.brace_direction, **kwargs)
181: (8)             self.brace.put_at_tip(self.label)
182: (8)             return self
183: (4)         def change_label(self, *text, **kwargs):
184: (8)             self.label = self.label_constructor(*text, **kwargs)
185: (8)             self.brace.put_at_tip(self.label)
186: (8)             return self
187: (4)         def change_brace_label(self, obj, *text, **kwargs):
188: (8)             self.shift_brace(obj)
189: (8)             self.change_label(*text, **kwargs)
190: (8)             return self
191: (0)     class BraceText(BraceLabel):
192: (4)         def __init__(self, obj, text, label_constructor=Tex, **kwargs):
193: (8)             super().__init__(obj, text, label_constructor=label_constructor,
**kwargs)
194: (0)     class BraceBetweenPoints(Brace):
195: (4)         """Similar to Brace, but instead of taking a mobject it uses 2
196: (4)         points to place the brace.
197: (4)         A fitting direction for the brace is
198: (4)         computed, but it still can be manually overridden.
199: (4)         If the points go from left to right, the brace is drawn from below.
200: (4)         Swapping the points places the brace on the opposite side.
201: (4)         Parameters
202: (4)         -----
203: (4)         point_1 :
204: (8)             The first point.
205: (4)         point_2 :
206: (8)             The second point.
207: (4)         direction :
208: (8)             The direction from which the brace faces towards the points.
209: (4)         Examples
210: (4)         -----
211: (8)             .. manim:: BraceBPExample
212: (12)             class BraceBPExample(Scene):
213: (16)                 def construct(self):
214: (20)                     p1 = [0,0,0]
215: (20)                     p2 = [1,2,0]
216: (20)                     brace = BraceBetweenPoints(p1,p2)
217: (20)                     self.play(Create(NumberPlane()))
218: (20)                     self.play(Create(brace))
219: (20)                     self.wait(2)
220: (4)         """
221: (4)         def __init__(
222: (8)             self,
223: (8)             point_1: Sequence[float] | None,
224: (8)             point_2: Sequence[float] | None,
225: (8)             direction: Sequence[float] | None = ORIGIN,
226: (8)             **kwargs,
227: (4)         ):
228: (8)             if all(direction == ORIGIN):
229: (12)                 line_vector = np.array(point_2) - np.array(point_1)
230: (12)                 direction = np.array([line_vector[1], -line_vector[0], 0])
231: (8)                 super().__init__(Line(point_1, point_2), direction=direction,
**kwargs)
232: (0)         class ArcBrace(Brace):
233: (4)             """Creates a :class:`~Brace` that wraps around an :class:`~Arc`.
```

```

234: (4)             The direction parameter allows the brace to be applied
235: (4)             from outside or inside the arc.
236: (4)
237: (8)             .. warning::
238: (4)                 The :class:`ArcBrace` is smaller for arcs with smaller radii.
239: (8)             .. note::
240: (8)                 The :class:`ArcBrace` is initially a vertical :class:`Brace` defined
241: (8)                 length of the :class:`~.Arc`, but is scaled down to match the start
242: (8)                 angles. An exponential function is then applied after it is shifted
243: (8)                 the radius of the arc.
244: (8)                 The scaling effect is not applied for arcs with radii smaller than 1
245: (4)                 over-scaling.
246: (4)             Parameters
247: (4)             -----
248: (4)             arc
249: (4)                 The :class:`~.Arc` that wraps around the :class:`Brace` mobject.
250: (8)             direction
251: (8)                 The direction from which the brace faces the arc.
252: (8)                 ``LEFT`` for inside the arc, and ``RIGHT`` for the outside.
253: (4)             Example
254: (4)             -----
255: (12)             .. manim:: ArcBraceExample
256: (12)                 :save_last_frame:
257: (12)                 :ref_classes: Arc
258: (16)                 class ArcBraceExample(Scene):
259: (20)                     def construct(self):
300: (20)                         arc_1 =
301: (20)                         Arc(radius=1.5,start_angle=0,angle=2*PI/3).set_color(RED)
302: (20)                         brace_1 = ArcBrace(arc_1,LEFT)
303: (20)                         group_1 = VGroup(arc_1,brace_1)
304: (20)                         arc_2 =
305: (20)                         Arc(radius=3,start_angle=0,angle=5*PI/6).set_color(YELLOW)
306: (20)                         brace_2 = ArcBrace(arc_2)
307: (20)                         group_2 = VGroup(arc_2,brace_2)
308: (20)                         arc_3 =
309: (20)                         Arc(radius=0.5,start_angle=-0,angle=PI).set_color(BLUE)
310: (20)                         brace_3 = ArcBrace(arc_3)
311: (20)                         group_3 = VGroup(arc_3,brace_3)
312: (20)                         arc_4 =
313: (20)                         Arc(radius=0.2,start_angle=0,angle=3*PI/2).set_color(GREEN)
314: (20)                         brace_4 = ArcBrace(arc_4)
315: (20)                         group_4 = VGroup(arc_4,brace_4)
316: (20)                         arc_group = VGroup(group_1, group_2, group_3,
317: (20)                         group_4).arrange_in_grid(buff=1.5)
318: (20)                         self.add(arc_group.center())
319: (4)             """
320: (4)             def __init__(
321: (8)                 self,
322: (8)                 arc: Arc | None = None,
323: (8)                 direction: Sequence[float] = RIGHT,
324: (8)                 **kwargs,
325: (4)             ):
326: (8)                 if arc is None:
327: (12)                     arc = Arc(start_angle=-1, angle=2, radius=1)
328: (8)                     arc_end_angle = arc.start_angle + arc.angle
329: (8)                     line = Line(UP * arc.start_angle, UP * arc_end_angle)
330: (8)                     scale_shift = RIGHT * np.log(arc.radius)
331: (8)                     if arc.radius >= 1:
332: (12)                         line.scale(arc.radius, about_point=ORIGIN)
333: (12)                         super().__init__(line, direction=direction, **kwargs)
334: (12)                         self.scale(1 / (arc.radius), about_point=ORIGIN)
335: (8)                     else:
336: (12)                         super().__init__(line, direction=direction, **kwargs)
337: (8)                     if arc.radius >= 0.3:
338: (12)                         self.shift(scale_shift)
339: (8)                 else:

```

```

294: (12)           self.shift(RIGHT * np.log(0.3))
295: (8)            self.apply_complex_function(np.exp)
296: (8)            self.shift(arc.get_arc_center())

```

-----  
File 76 - numbers.py:

```

1: (0)      """Mobjects representing numbers."""
2: (0)      from __future__ import annotations
3: (0)      __all__ = ["DecimalNumber", "Integer", "Variable"]
4: (0)      from typing import Sequence
5: (0)      import numpy as np
6: (0)      from manim import config
7: (0)      from manim.constants import *
8: (0)      from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
9: (0)      from manim.mobject.text.tex_mobject import MathTex, SingleStringMathTex, Tex
10: (0)     from manim.mobject.text.text_mobject import Text
11: (0)     from manim.mobject.types.vectorized_mobject import VMobject
12: (0)     from manim.mobject.value_tracker import ValueTracker
13: (0)     string_to_mob_map = {}
14: (0)     __all__ = ["DecimalNumber", "Integer", "Variable"]
15: (0)     class DecimalNumber(VMobject, metaclass=ConvertToOpenGL):
16: (4)       """An mobject representing a decimal number.
17: (4)       Parameters
18: (4)       -----
19: (4)       number
20: (8)         The numeric value to be displayed. It can later be modified using
:meth:`.set_value`.
21: (4)       num_decimal_places
22: (8)         The number of decimal places after the decimal separator. Values are
automatically rounded.
23: (4)
24: (8)         mob_class
25: (4)           The class for rendering digits and units, by default
:class:`.MathTex`.
26: (8)         include_sign
27: (4)           Set to ``True`` to include a sign for positive numbers and zero.
28: (8)         group_with_commas
29: (4)           When ``True`` thousands groups are separated by commas for
readability.
30: (8)         digit_buff_per_font_unit
31: (4)           Additional spacing between digits. Scales with font size.
32: (8)         show_ellipsis
33: (4)           When a number has been truncated by rounding, indicate with an
ellipsis (``...``).
34: (8)         unit
35: (4)           A unit string which can be placed to the right of the numerical
values.
36: (8)         unit_buff_per_font_unit
37: (4)           An additional spacing between the numerical values and the unit. A
of ``unit_buff_per_font_unit=0.003`` gives a decent spacing. Scales
with font size.
38: (4)         include_background_rectangle
39: (8)           Adds a background rectangle to increase contrast on busy scenes.
40: (4)         edge_to_fix
41: (8)           Assuring right- or left-alignment of the full object.
42: (4)         font_size
43: (8)           Size of the font.
44: (4)         Examples
45: (4)         -----
46: (4)         .. manim:: MovingSquareWithUpdaters
47: (8)           class MovingSquareWithUpdaters(Scene):
48: (12)             def construct(self):
49: (16)               decimal = DecimalNumber(
50: (20)                 0,
51: (20)                 show_ellipsis=True,
52: (20)                 num_decimal_places=3,
53: (20)                 include_sign=True,

```

```

54: (20)                         unit=r"\text{M-Units}",
55: (20)                         unit_buff_per_font_unit=0.003
56: (16)
57: (16)
58: (16)
59: (16)
[1]))
60: (16)                     self.add(square, decimal)
61: (16)                     self.play(
62: (20)                         square.animate.to_edge(DOWN),
63: (20)                         rate_func=there_and_back,
64: (20)                         run_time=5,
65: (16)
66: (16)                     self.wait()
67: (4)
68: (4) """
69: (8)     def __init__(
70: (8)         self,
71: (8)         number: float = 0,
72: (8)         num_decimal_places: int = 2,
73: (8)         mob_class: VMobject = MathTex,
74: (8)         include_sign: bool = False,
75: (8)         group_with_commas: bool = True,
76: (8)         digit_buff_per_font_unit: float = 0.001,
77: (8)         show_ellipsis: bool = False,
78: (8)         unit: str | None = None, # Aligned to bottom unless it starts with
    "^^"
79: (8)         unit_buff_per_font_unit: float = 0,
80: (8)         include_background_rectangle: bool = False,
81: (8)         edge_to_fix: Sequence[float] = LEFT,
82: (8)         font_size: float = DEFAULT_FONT_SIZE,
83: (8)         stroke_width: float = 0,
84: (8)         fill_opacity: float = 1.0,
85: (4)         **kwargs,
86: (8)     ):
87: (8)         super().__init__(**kwargs, stroke_width=stroke_width)
88: (8)         self.number = number
89: (8)         self.num_decimal_places = num_decimal_places
90: (8)         self.include_sign = include_sign
91: (8)         self.mob_class = mob_class
92: (8)         self.group_with_commas = group_with_commas
93: (8)         self.digit_buff_per_font_unit = digit_buff_per_font_unit
94: (8)         self.show_ellipsis = show_ellipsis
95: (8)         self.unit = unit
96: (8)         self.unit_buff_per_font_unit = unit_buff_per_font_unit
97: (8)         self.include_background_rectangle = include_background_rectangle
98: (8)         self.edge_to_fix = edge_to_fix
99: (8)         self._font_size = font_size
100: (8)        self.fill_opacity = fill_opacity
101: (8)        self.initial_config = kwargs.copy()
102: (12)        self.initial_config.update(
103: (16)            {
104: (16)                "num_decimal_places": num_decimal_places,
105: (16)                "include_sign": include_sign,
106: (16)                "group_with_commas": group_with_commas,
107: (16)                "digit_buff_per_font_unit": digit_buff_per_font_unit,
108: (16)                "show_ellipsis": show_ellipsis,
109: (16)                "unit": unit,
110: (16)                "unit_buff_per_font_unit": unit_buff_per_font_unit,
111: (16)                "include_background_rectangle": include_background_rectangle,
112: (16)                "edge_to_fix": edge_to_fix,
113: (16)                "font_size": font_size,
114: (16)                "stroke_width": stroke_width,
115: (12)                "fill_opacity": fill_opacity,
116: (8)            },
117: (8)        )
118: (8)        self._set_submobjects_from_number(number)
119: (4)        self.init_colors()
120: (4)        @property
121: (4)        def font_size(self):

```

```

121: (8)             """The font size of the tex mobject."""
122: (8)             return self.height / self.initial_height * self._font_size
123: (4)             @font_size.setter
124: (4)             def font_size(self, font_val):
125: (8)                 if font_val <= 0:
126: (12)                     raise ValueError("font_size must be greater than 0.")
127: (8)                 elif self.height > 0:
128: (12)                     self.scale(font_val / self.font_size)
129: (4)             def _set_submobjects_from_number(self, number):
130: (8)                 self.number = number
131: (8)                 self.submobjects = []
132: (8)                 num_string = self._get_num_string(number)
133: (8)                 self.add(*(map(self._string_to_mob, num_string)))
134: (8)                 if self.show_ellipsis:
135: (12)                     self.add(
136: (16)                         self._string_to_mob("\dots", SingleStringMathTex,
color=self.color),
137: (12)                         )
138: (8)             self.arrange(
139: (12)                 buff=self.digit_buff_per_font_unit * self._font_size,
140: (12)                 aligned_edge=DOWN,
141: (8)             )
142: (8)             if self.unit is not None:
143: (12)                 self.unit_sign = self._string_to_mob(self.unit,
SingleStringMathTex)
144: (12)                 self.add(
145: (16)                     self.unit_sign.next_to(
146: (20)                         self,
147: (20)                         direction=RIGHT,
148: (20)                         buff=(self.unit_buff_per_font_unit +
self.digit_buff_per_font_unit)
149: (20)                         * self._font_size,
150: (20)                         aligned_edge=DOWN,
151: (16)                     )
152: (12)                     )
153: (8)             self.move_to(ORIGIN)
154: (8)             for i, c in enumerate(num_string):
155: (12)                 if c == "-" and len(num_string) > i + 1:
156: (16)                     self[i].align_to(self[i + 1], UP)
157: (16)                     self[i].shift(self[i + 1].height * DOWN / 2)
158: (12)                 elif c == ",":
159: (16)                     self[i].shift(self[i].height * DOWN / 2)
160: (8)                 if self.unit and self.unit.startswith("^"):
161: (12)                     self.unit_sign.align_to(self, UP)
162: (8)                     self.initial_height = self.height
163: (8)                     if self.include_background_rectangle:
164: (12)                         self.add_background_rectangle()
165: (4)             def _get_num_string(self, number):
166: (8)                 if isinstance(number, complex):
167: (12)                     formatter = self._get_complex_formatter()
168: (8)                 else:
169: (12)                     formatter = self._get_formatter()
170: (8)                 num_string = formatter.format(number)
171: (8)                 rounded_num = np.round(number, self.num_decimal_places)
172: (8)                 if num_string.startswith("-") and rounded_num == 0:
173: (12)                     if self.include_sign:
174: (16)                         num_string = "+" + num_string[1:]
175: (12)                     else:
176: (16)                         num_string = num_string[1:]
177: (8)                 return num_string
178: (4)             def _string_to_mob(self, string: str, mob_class: VMobject | None = None,
**kwargs):
179: (8)                 if mob_class is None:
180: (12)                     mob_class = self.mob_class
181: (8)                 if string not in string_to_mob_map:
182: (12)                     string_to_mob_map[string] = mob_class(string, **kwargs)
183: (8)                 mob = string_to_mob_map[string].copy()
184: (8)                 mob.font_size = self._font_size
185: (8)                 return mob

```

```

186: (4)         def _get_formatter(self, **kwargs):
187: (8)             """
188: (8)                 Configuration is based first off instance attributes,
189: (8)                 but overwritten by any keyword argument. Relevant
190: (8)                 key words:
191: (8)                     - include_sign
192: (8)                     - group_with_commas
193: (8)                     - num_decimal_places
194: (8)                     - field_name (e.g. 0 or 0.real)
195: (8)             """
196: (8)             config = {
197: (12)                 attr: getattr(self, attr)
198: (12)                 for attr in [
199: (16)                     "include_sign",
200: (16)                     "group_with_commas",
201: (16)                     "num_decimal_places",
202: (12)                 ]
203: (8)             }
204: (8)             config.update(kwargs)
205: (8)             return "".join(
206: (12)                 [
207: (16)                     "{",
208: (16)                     config.get("field_name", ""),
209: (16)                     ":" ,
210: (16)                     "+" if config["include_sign"] else "",
211: (16)                     "," if config["group_with_commas"] else "",
212: (16)                     ".",
213: (16)                     str(config["num_decimal_places"]),
214: (16)                     "f",
215: (16)                     "}",
216: (12)                 ],
217: (8)             )
218: (4)         def _get_complex_formatter(self):
219: (8)             return "".join(
220: (12)                 [
221: (16)                     self._get_formatter(field_name="0.real"),
222: (16)                     self._get_formatter(field_name="0.imag", include_sign=True),
223: (16)                     "i",
224: (12)                 ],
225: (8)             )
226: (4)         def set_value(self, number: float):
227: (8)             """Set the value of the :class:`~.DecimalNumber` to a new number.
228: (8)             Parameters
229: (8)             -----
230: (8)             number
231: (12)                 The value that will overwrite the current number of the
232: (8)                 :class:`~.DecimalNumber`.
233: (8)             """
234: (8)             old_family = self.get_family()
235: (8)             old_font_size = self.font_size
236: (8)             move_to_point = self.get_edge_center(self.edge_to_fix)
237: (8)             old_submobjects = self.submobjects
238: (8)             self._set_submobjects_from_number(number)
239: (8)             self.font_size = old_font_size
240: (8)             self.move_to(move_to_point, self.edge_to_fix)
241: (12)             for sm1, sm2 in zip(self.submobjects, old_submobjects):
242: (8)                 sm1.match_style(sm2)
243: (12)             if config.renderer == RendererType.CAIRO:
244: (16)                 for mob in old_family:
245: (8)                     mob.points[:] = 0
246: (8)             self.init_colors()
247: (4)             return self
248: (8)         def get_value(self):
249: (8)             return self.number
250: (4)             def increment_value(self, delta_t=1):
251: (8)                 self.set_value(self.get_value() + delta_t)
252: (0)         class Integer(DecimalNumber):
253: (4)             """A class for displaying Integers.
254: (4)             Examples

```

```

254: (4)          -----
255: (4)          .. manim:: IntegerExample
256: (8)          :save_last_frame:
257: (8)          class IntegerExample(Scene):
258: (12)         def construct(self):
259: (16)
self.add(Integer(number=2.5).set_color(ORANGE).scale(2.5).set_x(-0.5).set_y(0.8))
260: (16)           self.add(Integer(number=3.14159,
show_ellipsis=True).set_x(3).set_y(3.3).scale(3.14159))
261: (16)
self.add(Integer(number=42).set_x(2.5).set_y(-2.3).set_color_by_gradient(BLUE, TEAL).scale(1.7))
262: (16)
self.add(Integer(number=6.28).set_x(-1.5).set_y(-2).set_color(YELLOW).scale(1.4))
263: (4)         """
264: (4)         def __init__(self, number=0, num_decimal_places=0, **kwargs):
265: (8)           super().__init__(number=number, num_decimal_places=num_decimal_places,
**kwargs)
266: (4)         def get_value(self):
267: (8)           return int(np.round(super().get_value()))
268: (0)         class Variable(VMobject, metaclass=ConvertToOpenGL):
269: (4)           """A class for displaying text that shows "label = value" with
270: (4)             the value continuously updated from a :class:`~.ValueTracker`.
271: (4)             Parameters
272: (4)             -----
273: (4)             var
274: (8)               The initial value you need to keep track of and display.
275: (4)             label
276: (8)               The label for your variable. Raw strings are converted to
:class:`~.MathTex` objects.
277: (4)             var_type
278: (8)               The class used for displaying the number. Defaults to
:class:`~.DecimalNumber`.
279: (4)             num_decimal_places
280: (8)               The number of decimal places to display in your variable. Defaults to
2.
281: (8)               If `var_type` is an :class:`~.Integer`, this parameter is ignored.
282: (4)             kwargs
283: (12)            Other arguments to be passed to `~.Mobject`.
284: (4)             Attributes
285: (4)             -----
286: (4)             label : Union[:class:`str`, :class:`~.Tex`, :class:`~.MathTex`,
:class:`~.Text`, :class:`~.SingleStringMathTex`]
287: (8)               The label for your variable, for example ``x = ...``.
288: (4)             tracker : :class:`~.ValueTracker`
289: (8)               Useful in updating the value of your variable on-screen.
290: (4)             value : Union[:class:`~.DecimalNumber`, :class:`~.Integer`]
291: (8)               The tex for the value of your variable.
292: (4)             Examples
293: (4)             -----
294: (4)             Normal usage::
295: (8)               var = 0.5
296: (8)               on_screen_var = Variable(var, Text("var"), num_decimal_places=3)
297: (8)               int_var = 0
298: (8)               on_screen_int_var = Variable(int_var, Text("int_var"),
var_type=Integer)
299: (8)               on_screen_int_var = Variable(int_var, "{a}_{i}", var_type=Integer)
.. manim:: VariablesWithValueTracker
301: (8)             class VariablesWithValueTracker(Scene):
302: (12)             def construct(self):
303: (16)               var = 0.5
304: (16)               on_screen_var = Variable(var, Text("var"),
num_decimal_places=3)
305: (16)               on_screen_var.label.set_color(RED)
306: (16)               on_screen_var.value.set_color(GREEN)
307: (16)               self.play(Write(on_screen_var))
308: (16)               self.wait()
309: (16)               var_tracker = on_screen_var.tracker
310: (16)               var = 10.5
311: (16)               self.play(var_tracker.animate.set_value(var))

```

```

312: (16)                     self.wait()
313: (16)                     int_var = 0
314: (16)                     on_screen_int_var = Variable(
315: (20)                         int_var, Text("int_var"), var_type=Integer
316: (16)                     ).next_to(on_screen_var, DOWN)
317: (16)                     on_screen_int_var.label.set_color(RED)
318: (16)                     on_screen_int_var.value.set_color(GREEN)
319: (16)                     self.play(Write(on_screen_int_var))
320: (16)                     self.wait()
321: (16)                     var_tracker = on_screen_int_var.tracker
322: (16)                     var = 10.5
323: (16)                     self.play(var_tracker.animate.set_value(var))
324: (16)                     self.wait()
325: (16)                     subscript_label_var = 10
326: (16)                     on_screen_subscript_var = Variable(subscript_label_var, "
{a}_{i}").next_to(
327: (20)                         on_screen_int_var, DOWN
328: (16)                     )
329: (16)                     self.play(Write(on_screen_subscript_var))
330: (16)                     self.wait()
331: (4) .. manim:: VariableExample
332: (8)             class VariableExample(Scene):
333: (12)                 def construct(self):
334: (16)                     start = 2.0
335: (16)                     x_var = Variable(start, 'x', num_decimal_places=3)
336: (16)                     sqr_var = Variable(start**2, 'x^2', num_decimal_places=3)
337: (16)                     Group(x_var, sqr_var).arrange(DOWN)
338: (16)                     sqr_var.add_updater(lambda v:
v.tracker.set_value(x_var.tracker.get_value()**2))
339: (16)                         self.add(x_var, sqr_var)
340: (16)                         self.play(x_var.tracker.animate.set_value(5), run_time=2,
rate_func=linear)
341: (16)                     self.wait(0.1)
342: (4) """
343: (4)             def __init__(self,
344: (8)                 var: float,
345: (8)                 label: str | Tex | MathTex | Text | SingleStringMathTex,
346: (8)                 var_type: DecimalNumber | Integer = DecimalNumber,
347: (8)                 num_decimal_places: int = 2,
348: (8)                 **kwargs,
349: (8)             ):
350: (4)                 self.label = MathTex(label) if isinstance(label, str) else label
351: (8)                 equals = MathTex("=").next_to(self.label, RIGHT)
352: (8)                 self.label.add>equals)
353: (8)                 self.tracker = ValueTracker(var)
354: (8)                 if var_type == DecimalNumber:
355: (12)                     self.value = DecimalNumber(
356: (16)                         self.tracker.get_value(),
357: (16)                         num_decimal_places=num_decimal_places,
358: (16)                         )
359: (12)                 elif var_type == Integer:
360: (8)                     self.value = Integer(self.tracker.get_value())
361: (12)                     self.value.add_updater(lambda v:
362: (8)                         self.set_value(self.tracker.get_value()).next_to(
363: (12)                             self.label,
364: (12)                             RIGHT,
365: (8)                         )
366: (8)                         super().__init__(**kwargs)
367: (8)                         self.add(self.label, self.value)

```

---

### File 77 - \_\_init\_\_.py:

```

1: (0)         """Mobjects related to SVG images.
2: (0)         Modules
3: (0)         =====
4: (0)         .. autosummary::
```

```

5: (4)          :toctree: ../reference
6: (4)          ~brace
7: (4)          ~svg_mobject
8: (0)          """
-----
```

File 78 - \_\_init\_\_.py:

```

1: (0)          """Mobjects used to display Text using Pango or LaTeX.
2: (0)          Modules
3: (0)          =====
4: (0)          .. autosummary::
5: (4)          :toctree: ../reference
6: (4)          ~code_mobject
7: (4)          ~numbers
8: (4)          ~tex_mobject
9: (4)          ~text_mobject
10: (0)         """
-----
```

File 79 - \_\_init\_\_.py:

```

1: (0)          """Three-dimensional mobjects.
2: (0)          Modules
3: (0)          =====
4: (0)          .. autosummary::
5: (4)          :toctree: ../reference
6: (4)          ~polyhedra
7: (4)          ~three_d_utils
8: (4)          ~three_dimensions
9: (0)         """
-----
```

File 80 - \_\_init\_\_.py:

```

1: (0)          """Specialized mobject base classes.
2: (0)          Modules
3: (0)          =====
4: (0)          .. autosummary::
5: (4)          :toctree: ../reference
6: (4)          ~image_mobject
7: (4)          ~point_cloud_mobject
8: (4)          ~vectorized_mobject
9: (0)         """
-----
```

File 81 - polyhedra.py:

```

1: (0)          """General polyhedral class and platonic solids."""
2: (0)          from __future__ import annotations
3: (0)          from typing import TYPE_CHECKING
4: (0)          import numpy as np
5: (0)          from manim.mobject.geometry.polygram import Polygon
6: (0)          from manim.mobject.graph import Graph
7: (0)          from manim.mobject.three_d.three_dimensions import Dot3D
8: (0)          from manim.mobject.types.vectorized_mobject import VGroup
9: (0)          if TYPE_CHECKING:
10: (4)             from manim.mobject.mobject import Mobject
11: (0)             __all__ = ["Polyhedron", "Tetrahedron", "Octahedron", "Icosahedron",
"Dodecahedron"]
12: (0)             class Polyhedron(VGroup):
13: (4)                 """An abstract polyhedra class.
14: (4)                 In this implementation, polyhedra are defined with a list of vertex
coordinates in space, and a list
15: (4)                     of faces. This implementation mirrors that of a standard polyhedral data
-----
```

```

format (OFF, object file format).
16: (4)          Parameters
17: (4)          -----
18: (4)          vertex_coords
19: (8)          A list of coordinates of the corresponding vertices in the polyhedron.
Each coordinate will correspond to
20: (8)          a vertex. The vertices are indexed with the usual indexing of Python.
21: (4)          faces_list
22: (8)          A list of faces. Each face is a sublist containing the indices of the
vertices that form the corners of that face.
23: (4)          faces_config
24: (8)          Configuration for the polygons representing the faces of the
polyhedron.
25: (4)          graph_config
26: (8)          Configuration for the graph containing the vertices and edges of the
polyhedron.
27: (4)          Examples
28: (4)          -----
29: (4)          To understand how to create a custom polyhedra, let's use the example of a
rather simple one - a square pyramid.
30: (4)          .. manim:: SquarePyramidScene
31: (8)          :save_last_frame:
32: (8)          class SquarePyramidScene(ThreeDScene):
33: (12)          def construct(self):
34: (16)          self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
35: (16)          vertex_coords = [
36: (20)          [1, 1, 0],
37: (20)          [1, -1, 0],
38: (20)          [-1, -1, 0],
39: (20)          [-1, 1, 0],
40: (20)          [0, 0, 2]
41: (16)          ]
42: (16)          faces_list = [
43: (20)          [0, 1, 4],
44: (20)          [1, 2, 4],
45: (20)          [2, 3, 4],
46: (20)          [3, 0, 4],
47: (20)          [0, 1, 2, 3]
48: (16)          ]
49: (16)          pyramid = Polyhedron(vertex_coords, faces_list)
50: (16)          self.add(pyramid)
51: (4)          In defining the polyhedron above, we first defined the coordinates of the
vertices.
52: (4)          These are the corners of the square base, given as the first four
coordinates in the vertex list,
53: (4)          and the apex, the last coordinate in the list.
54: (4)          Next, we define the faces of the polyhedron. The triangular surfaces of
the pyramid are polygons
55: (4)          with two adjacent vertices in the base and the vertex at the apex as
corners. We thus define these
56: (4)          surfaces in the first four elements of our face list. The last element
defines the base of the pyramid.
57: (4)          The graph and faces of polyhedra can also be accessed and modified
directly, after instantiation.
58: (4)          They are stored in the `graph` and `faces` attributes respectively.
59: (4)          .. manim:: PolyhedronSubMobjects
60: (8)          :save_last_frame:
61: (8)          class PolyhedronSubMobjects(ThreeDScene):
62: (12)          def construct(self):
63: (16)          self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
64: (16)          octahedron = Octahedron(edge_length = 3)
65: (16)          octahedron.graph[0].set_color(RED)
66: (16)          octahedron.faces[2].set_color(YELLOW)
67: (16)          self.add(octahedron)
68: (4)          """
69: (4)          def __init__(
70: (8)          self,

```

```

71: (8)             vertex_coords: list[list[float] | np.ndarray],
72: (8)             faces_list: list[list[int]],
73: (8)             faces_config: dict[str, str | int | float | bool] = {},
74: (8)             graph_config: dict[str, str | int | float | bool] = {},
75: (4)         ):
76: (8)             super().__init__()
77: (8)             self.faces_config = dict(
78: (12)                 {"fill_opacity": 0.5, "shade_in_3d": True}, **faces_config
79: (8)
80: (8)             self.graph_config = dict(
81: (12)                 {
82: (16)                     "vertex_type": Dot3D,
83: (16)                     "edge_config": {
84: (20)                         "stroke_opacity": 0, # I find that having the edges
visible makes the polyhedra look weird
85: (16)                     },
86: (12)                 },
87: (12)             **graph_config,
88: (8)
89: (8)             self.vertex_coords = vertex_coords
90: (8)             self.vertex_indices = list(range(len(self.vertex_coords)))
91: (8)             self.layout = dict(enumerate(self.vertex_coords))
92: (8)             self.faces_list = faces_list
93: (8)             self.face_coords = [[self.layout[j] for j in i] for i in faces_list]
94: (8)             self.edges = self.get_edges(self.faces_list)
95: (8)             self.faces = self.create_faces(self.face_coords)
96: (8)             self.graph = Graph(
97: (12)                 self.vertex_indices, self.edges, layout=self.layout,
**self.graph_config
98: (8)
99: (8)
100: (8)         )
101: (4)     def get_edges(self, faces_list: list[list[int]]) -> list[tuple[int, int]]:
102: (8)         """Creates list of cyclic pairwise tuples."""
103: (8)         edges = []
104: (8)         for face in faces_list:
105: (12)             edges += zip(face, face[1:] + face[:1])
106: (8)         return edges
107: (4)     def create_faces(
108: (8)         self,
109: (8)         face_coords: list[list[list | np.ndarray]],
110: (4)     ) -> VGroup:
111: (8)         """Creates VGroup of faces from a list of face coordinates."""
112: (8)         face_group = VGroup()
113: (8)         for face in face_coords:
114: (12)             face_group.add(Polygon(*face, **self.faces_config))
115: (8)         return face_group
116: (4)     def update_faces(self, m: Mobject):
117: (8)         face_coords = self.extract_face_coords()
118: (8)         new_faces = self.create_faces(face_coords)
119: (8)         self.faces.match_points(new_faces)
120: (4)     def extract_face_coords(self) -> list[list[np.ndarray]]:
121: (8)         """Extracts the coordinates of the vertices in the graph.
Used for updating faces.
"""
122: (8)
123: (8)
124: (8)         new_vertex_coords = []
125: (8)         for v in self.graph.vertices:
126: (12)             new_vertex_coords.append(self.graph[v].get_center())
127: (8)         layout = dict(enumerate(new_vertex_coords))
128: (8)         return [[layout[j] for j in i] for i in self.faces_list]
129: (0)     class Tetrahedron(Polyhedron):
130: (4)         """A tetrahedron, one of the five platonic solids. It has 4 faces, 6
edges, and 4 vertices.
131: (4)             Parameters
132: (4)             -----
133: (4)             edge_length
134: (8)                 The length of an edge between any two vertices.
135: (4)             Examples
136: (4)             -----

```

```

137: (4)          .. manim:: TetrahedronScene
138: (8)          :save_last_frame:
139: (8)          class TetrahedronScene(ThreeDScene):
140: (12)         def construct(self):
141: (16)         self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
142: (16)         obj = Tetrahedron()
143: (16)         self.add(obj)
144: (4)         """
145: (4)         def __init__(self, edge_length: float = 1, **kwargs):
146: (8)         unit = edge_length * np.sqrt(2) / 4
147: (8)         super().__init__(
148: (12)         vertex_coords=[
149: (16)         np.array([unit, unit, unit]),
150: (16)         np.array([unit, -unit, -unit]),
151: (16)         np.array([-unit, unit, -unit]),
152: (16)         np.array([-unit, -unit, unit]),
153: (12)         ],
154: (12)         faces_list=[[0, 1, 2], [3, 0, 2], [0, 1, 3], [3, 1, 2]],
155: (12)         **kwargs,
156: (8)         )
157: (0)          class Octahedron(Polyhedron):
158: (4)          """An octahedron, one of the five platonic solids. It has 8 faces, 12
edges and 6 vertices.
159: (4)          Parameters
160: (4)          -----
161: (4)          edge_length
162: (8)          The length of an edge between any two vertices.
163: (4)          Examples
164: (4)          -----
165: (4)          .. manim:: OctahedronScene
166: (8)          :save_last_frame:
167: (8)          class OctahedronScene(ThreeDScene):
168: (12)         def construct(self):
169: (16)         self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
170: (16)         obj = Octahedron()
171: (16)         self.add(obj)
172: (4)         """
173: (4)         def __init__(self, edge_length: float = 1, **kwargs):
174: (8)         unit = edge_length * np.sqrt(2) / 2
175: (8)         super().__init__(
176: (12)         vertex_coords=[
177: (16)         np.array([unit, 0, 0]),
178: (16)         np.array([-unit, 0, 0]),
179: (16)         np.array([0, unit, 0]),
180: (16)         np.array([0, -unit, 0]),
181: (16)         np.array([0, 0, unit]),
182: (16)         np.array([0, 0, -unit]),
183: (12)         ],
184: (12)         faces_list=[
185: (16)         [2, 4, 1],
186: (16)         [0, 4, 2],
187: (16)         [4, 3, 0],
188: (16)         [1, 3, 4],
189: (16)         [3, 5, 0],
190: (16)         [1, 5, 3],
191: (16)         [2, 5, 1],
192: (16)         [0, 5, 2],
193: (12)         ],
194: (12)         **kwargs,
195: (8)         )
196: (0)          class Icosahedron(Polyhedron):
197: (4)          """An icosahedron, one of the five platonic solids. It has 20 faces, 30
edges and 12 vertices.
198: (4)          Parameters
199: (4)          -----
200: (4)          edge_length
201: (8)          The length of an edge between any two vertices.

```

```

202: (4)             Examples
203: (4)
204: (4)
205: (8)           .. manim:: IcosahedronScene
206: (8)           :save_last_frame:
207: (12)          class IcosahedronScene(ThreeDScene):
208: (16)          def construct(self):
209: (16)            self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
210: (16)            obj = Icosahedron()
211: (4)            self.add(obj)
212: (4)        """
213: (8)        def __init__(self, edge_length: float = 1, **kwargs):
214: (8)            unit_a = edge_length * ((1 + np.sqrt(5)) / 4)
215: (8)            unit_b = edge_length * (1 / 2)
216: (12)           super().__init__(
217: (16)             vertex_coords=[
218: (16)               np.array([0, unit_b, unit_a]),
219: (16)               np.array([0, -unit_b, unit_a]),
220: (16)               np.array([0, unit_b, -unit_a]),
221: (16)               np.array([0, -unit_b, -unit_a]),
222: (16)               np.array([unit_b, unit_a, 0]),
223: (16)               np.array([unit_b, -unit_a, 0]),
224: (16)               np.array([-unit_b, unit_a, 0]),
225: (16)               np.array([-unit_b, -unit_a, 0]),
226: (16)               np.array([unit_a, 0, unit_b]),
227: (16)               np.array([unit_a, 0, -unit_b]),
228: (16)               np.array([-unit_a, 0, unit_b]),
229: (16)               np.array([-unit_a, 0, -unit_b]),
230: (12)           ],
231: (16)             faces_list=[
232: (16)               [1, 8, 0],
233: (16)               [1, 5, 7],
234: (16)               [8, 5, 1],
235: (16)               [7, 3, 5],
236: (16)               [5, 9, 3],
237: (16)               [8, 9, 5],
238: (16)               [3, 2, 9],
239: (16)               [9, 4, 2],
240: (16)               [8, 4, 9],
241: (16)               [0, 4, 8],
242: (16)               [6, 4, 0],
243: (16)               [6, 2, 4],
244: (16)               [11, 2, 6],
245: (16)               [3, 11, 2],
246: (16)               [0, 6, 10],
247: (16)               [10, 1, 0],
248: (16)               [10, 7, 1],
249: (16)               [11, 7, 3],
250: (16)               [10, 11, 7],
251: (16)               [10, 11, 6],
252: (12)           ],
253: (8)           **kwargs,
254: (0)       )
255: (4)       class Dodecahedron(Polyhedron):
256: (8)           """A dodecahedron, one of the five platonic solids. It has 12 faces, 30
edges and 20 vertices.
257: (4)           Parameters
258: (4)           -----
259: (8)           edge_length
260: (8)           The length of an edge between any two vertices.
261: (4)           Examples
262: (4)
263: (4)           .. manim:: DodecahedronScene
264: (8)           :save_last_frame:
265: (8)           class DodecahedronScene(ThreeDScene):
266: (12)          def construct(self):
267: (16)            self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
268: (16)            obj = Dodecahedron()

```

```

268: (16)                     self.add(obj)
269: (4)             """
270: (4)             def __init__(self, edge_length: float = 1, **kwargs):
271: (8)                 unit_a = edge_length * ((1 + np.sqrt(5)) / 4)
272: (8)                 unit_b = edge_length * ((3 + np.sqrt(5)) / 4)
273: (8)                 unit_c = edge_length * (1 / 2)
274: (8)                 super().__init__(
275: (12)                     vertex_coords=[
276: (16)                         np.array([unit_a, unit_a, unit_a]),
277: (16)                         np.array([unit_a, unit_a, -unit_a]),
278: (16)                         np.array([unit_a, -unit_a, unit_a]),
279: (16)                         np.array([unit_a, -unit_a, -unit_a]),
280: (16)                         np.array([-unit_a, unit_a, unit_a]),
281: (16)                         np.array([-unit_a, unit_a, -unit_a]),
282: (16)                         np.array([-unit_a, -unit_a, unit_a]),
283: (16)                         np.array([-unit_a, -unit_a, -unit_a]),
284: (16)                         np.array([0, unit_c, unit_b]),
285: (16)                         np.array([0, unit_c, -unit_b]),
286: (16)                         np.array([0, -unit_c, -unit_b]),
287: (16)                         np.array([0, -unit_c, unit_b]),
288: (16)                         np.array([unit_c, unit_b, 0]),
289: (16)                         np.array([-unit_c, unit_b, 0]),
290: (16)                         np.array([unit_c, -unit_b, 0]),
291: (16)                         np.array([-unit_c, -unit_b, 0]),
292: (16)                         np.array([unit_b, 0, unit_c]),
293: (16)                         np.array([-unit_b, 0, unit_c]),
294: (16)                         np.array([unit_b, 0, -unit_c]),
295: (16)                         np.array([-unit_b, 0, -unit_c]),
296: (12)                     ],
297: (12)                     faces_list=[
298: (16)                         [18, 16, 0, 12, 1],
299: (16)                         [3, 18, 16, 2, 14],
300: (16)                         [3, 10, 9, 1, 18],
301: (16)                         [1, 9, 5, 13, 12],
302: (16)                         [0, 8, 4, 13, 12],
303: (16)                         [2, 16, 0, 8, 11],
304: (16)                         [4, 17, 6, 11, 8],
305: (16)                         [17, 19, 5, 13, 4],
306: (16)                         [19, 7, 15, 6, 17],
307: (16)                         [6, 15, 14, 2, 11],
308: (16)                         [19, 5, 9, 10, 7],
309: (16)                         [7, 10, 3, 14, 15],
310: (12)                     ],
311: (12)                     **kwargs,
312: (8)                 )
-----
```

**File 82 - svg\_mobject.py:**

```

1: (0)             """Mobjects generated from an SVG file."""
2: (0)             from __future__ import annotations
3: (0)             import os
4: (0)             from pathlib import Path
5: (0)             from xml.etree import ElementTree as ET
6: (0)             import numpy as np
7: (0)             import svgelements as se
8: (0)             from manim import config, logger
9: (0)             from ...constants import RIGHT
10: (0)             from ...utils.bezier import get_quadratic_approximation_of_cubic
11: (0)             from ...utils.images import get_full_vector_image_path
12: (0)             from ...utils.iterables import hash_obj
13: (0)             from ..geometry.arc import Circle
14: (0)             from ..geometry.line import Line
15: (0)             from ..geometry.polygram import Polygon, Rectangle, RoundedRectangle
16: (0)             from ..opengl.opengl_compatibility import ConvertToOpenGL
17: (0)             from ..types.vectorized_mobject import VMobject
18: (0)             __all__ = ["SVGMobject", "VMobjectFromSVGPath"]
19: (0)             SVG_HASH_TO_MOB_MAP: dict[int, VMobject] = {}
```

```

20: (0)         def _convert_point_to_3d(x: float, y: float) -> np.ndarray:
21: (4)             return np.array([x, y, 0.0])
22: (0)         class SVGMobject(VMobject, metaclass=ConvertToOpenGL):
23: (4)             """A vectorized mobject created from importing an SVG file.
24: (4)             Parameters
25: (4)             -----
26: (4)             file_name
27: (8)                 The path to the SVG file.
28: (4)             should_center
29: (8)                 Whether or not the mobject should be centered after
30: (8)                     being imported.
31: (4)             height
32: (8)                 The target height of the mobject, set to 2 Manim units by default.
33: (8)                 If the height and width are both set to ``None``, the mobject
34: (8)                     is imported without being scaled.
35: (4)             width
36: (8)                 The target width of the mobject, set to ``None`` by default. If
37: (8)                     the height and the width are both set to ``None``, the mobject
38: (8)                     is imported without being scaled.
39: (4)             color
40: (8)                 The color (both fill and stroke color) of the mobject. If
41: (8)                     ``None`` (the default), the colors set in the SVG file
42: (8)                     are used.
43: (4)             opacity
44: (8)                 The opacity (both fill and stroke opacity) of the mobject.
45: (8)                 If ``None`` (the default), the opacity set in the SVG file
46: (8)                     is used.
47: (4)             fill_color
48: (8)                 The fill color of the mobject. If ``None`` (the default),
49: (8)                     the fill colors set in the SVG file are used.
50: (4)             fill_opacity
51: (8)                 The fill opacity of the mobject. If ``None`` (the default),
52: (8)                     the fill opacities set in the SVG file are used.
53: (4)             stroke_color
54: (8)                 The stroke color of the mobject. If ``None`` (the default),
55: (8)                     the stroke colors set in the SVG file are used.
56: (4)             stroke_opacity
57: (8)                 The stroke opacity of the mobject. If ``None`` (the default),
58: (8)                     the stroke opacities set in the SVG file are used.
59: (4)             stroke_width
60: (8)                 The stroke width of the mobject. If ``None`` (the default),
61: (8)                     the stroke width values set in the SVG file are used.
62: (4)             svg_default
63: (8)                 A dictionary in which fallback values for unspecified
64: (8)                     properties of elements in the SVG file are defined. If
65: (8)                     ``None`` (the default), ``color``, ``opacity``, ``fill_color``
66: (8)                     ``fill_opacity``, ``stroke_color``, and ``stroke_opacity``
67: (8)                     are set to ``None``, and ``stroke_width`` is set to 0.
68: (4)             path_string_config
69: (8)                 A dictionary with keyword arguments passed to
70: (8)                     :class:`.VMobjectFromSVGPath` used for importing path elements.
71: (8)                 If ``None`` (the default), no additional arguments are passed.
72: (4)             use_svg_cache
73: (8)                 If True (default), the svg inputs (e.g. file_name, settings)
74: (8)                     will be used as a key and a copy of the created mobject will
75: (8)                     be saved using that key to be quickly retrieved if the same
76: (8)                     inputs need be processed later. For large SVGs which are used
77: (8)                     only once, this can be omitted to improve performance.
78: (4)             kwargs
79: (8)                 Further arguments passed to the parent class.
80: (4)             """
81: (4)         def __init__(
82: (8)             self,
83: (8)                 file_name: str | os.PathLike | None = None,
84: (8)                 should_center: bool = True,
85: (8)                 height: float | None = 2,
86: (8)                 width: float | None = None,
87: (8)                 color: str | None = None,
88: (8)                 opacity: float | None = None,

```

```

89: (8)                 fill_color: str | None = None,
90: (8)                 fill_opacity: float | None = None,
91: (8)                 stroke_color: str | None = None,
92: (8)                 stroke_opacity: float | None = None,
93: (8)                 stroke_width: float | None = None,
94: (8)                 svg_default: dict | None = None,
95: (8)                 path_string_config: dict | None = None,
96: (8)                 use_svg_cache: bool = True,
97: (8)                 **kwargs,
98: (4):
99: (8)                 super().__init__(color=None, stroke_color=None, fill_color=None,
**kwargs)
100: (8)                 self.file_name = Path(file_name) if file_name is not None else None
101: (8)                 self.should_center = should_center
102: (8)                 self.svg_height = height
103: (8)                 self.svg_width = width
104: (8)                 self.color = color
105: (8)                 self.opacity = opacity
106: (8)                 self.fill_color = fill_color
107: (8)                 self.fill_opacity = fill_opacity
108: (8)                 self.stroke_color = stroke_color
109: (8)                 self.stroke_opacity = stroke_opacity
110: (8)                 self.stroke_width = stroke_width
111: (8)                 if svg_default is None:
112: (12)                     svg_default = {
113: (16)                         "color": None,
114: (16)                         "opacity": None,
115: (16)                         "fill_color": None,
116: (16)                         "fill_opacity": None,
117: (16)                         "stroke_width": 0,
118: (16)                         "stroke_color": None,
119: (16)                         "stroke_opacity": None,
120: (12)                     }
121: (8)                     self.svg_default = svg_default
122: (8)                     if path_string_config is None:
123: (12)                         path_string_config = {}
124: (8)                         self.path_string_config = path_string_config
125: (8)                         self.init_svg_mobject(use_svg_cache=use_svg_cache)
126: (8)                         self.set_style(
127: (12)                             fill_color=fill_color,
128: (12)                             fill_opacity=fill_opacity,
129: (12)                             stroke_color=stroke_color,
130: (12)                             stroke_opacity=stroke_opacity,
131: (12)                             stroke_width=stroke_width,
132: (8)                         )
133: (8)                         self.move_into_position()
134: (4)             def init_svg_mobject(self, use_svg_cache: bool) -> None:
135: (8)                 """Checks whether the SVG has already been imported and
136: (8)                 generates it if not.
137: (8)                 See also
138: (8)                 -----
139: (8)                 :meth:`.SVGMOBJECT.generate_mobject`
140: (8)                 """
141: (8)                 if use_svg_cache:
142: (12)                     hash_val = hash_obj(self.hash_seed)
143: (12)                     if hash_val in SVG_HASH_TO_MOB_MAP:
144: (16)                         mob = SVG_HASH_TO_MOB_MAP[hash_val].copy()
145: (16)                         self.add(*mob)
146: (16)                         return
147: (8)                     self.generate_mobject()
148: (8)                     if use_svg_cache:
149: (12)                         SVG_HASH_TO_MOB_MAP[hash_val] = self.copy()
150: (4)             @property
151: (4)             def hash_seed(self) -> tuple:
152: (8)                 """A unique hash representing the result of the generated
153: (8)                 mobject points.
154: (8)                 Used as keys in the ``SVG_HASH_TO_MOB_MAP`` caching dictionary.
155: (8)                 """
156: (8)                 return (

```

```

157: (12)                         self.__class__.__name__,
158: (12)                         self.svg_default,
159: (12)                         self.path_string_config,
160: (12)                         self.file_name,
161: (12)                         config.renderer,
162: (8)
163: (4)     def generate_mobject(self) -> None:
164: (8)         """Parse the SVG and translate its elements to submobjects."""
165: (8)         file_path = self.get_file_path()
166: (8)         element_tree = ET.parse(file_path)
167: (8)         new_tree = self.modify_xml_tree(element_tree)
168: (8)         modified_file_path = file_path.with_name(f"
{file_path.stem}_{file_path.suffix}")
169: (8)             new_tree.write(modified_file_path)
170: (8)             svg = se.SVG.parse(modified_file_path)
171: (8)             modified_file_path.unlink()
172: (8)             mobjects = self.get_mobjects_from(svg)
173: (8)             self.add(*mobjects)
174: (8)             self.flip(RIGHT) # Flip y
175: (4)     def get_file_path(self) -> Path:
176: (8)         """Search for an existing file based on the specified file name."""
177: (8)         if self.file_name is None:
178: (12)             raise ValueError("Must specify file for SVGMobject")
179: (8)         return get_full_vector_image_path(self.file_name)
180: (4)     def modify_xml_tree(self, element_tree: ET.ElementTree) -> ET.ElementTree:
181: (8)         """Modifies the SVG element tree to include default
182: (8)         style information.
183: (8)         Parameters
184: (8)         -----
185: (8)         element_tree
186: (12)             The parsed element tree from the SVG file.
187: (8)
188: (8)         config_style_dict = self.generate_config_style_dict()
189: (8)         style_keys = (
190: (12)             "fill",
191: (12)             "fill-opacity",
192: (12)             "stroke",
193: (12)             "stroke-opacity",
194: (12)             "stroke-width",
195: (12)             "style",
196: (8)
197: (8)
198: (8)             root = element_tree.getroot()
199: (8)             root_style_dict = {k: v for k, v in root.attrib.items() if k in
style_keys}
200: (8)
201: (8)             new_root = ET.Element("svg", {})
202: (8)             config_style_node = ET.SubElement(new_root, "g", config_style_dict)
203: (8)             root_style_node = ET.SubElement(config_style_node, "g",
root_style_dict)
204: (4)     def generate_config_style_dict(self) -> dict[str, str]:
205: (8)         """Generate a dictionary holding the default style information."""
206: (8)         keys_converting_dict = {
207: (12)             "fill": ("color", "fill_color"),
208: (12)             "fill-opacity": ("opacity", "fill_opacity"),
209: (12)             "stroke": ("color", "stroke_color"),
210: (12)             "stroke-opacity": ("opacity", "stroke_opacity"),
211: (12)             "stroke-width": ("stroke_width",),
212: (8)
213: (8)
214: (8)             svg_default_dict = self.svg_default
215: (8)             result = {}
216: (12)             for svg_key, style_keys in keys_converting_dict.items():
217: (16)                 for style_key in style_keys:
218: (20)                     if svg_default_dict[style_key] is None:
219: (16)                         continue
220: (8)                     result[svg_key] = str(svg_default_dict[style_key])
221: (4)             return result
222: (8)             def get_mobjects_from(self, svg: se.SVG) -> list[VMobject]:
223: (8)                 """Convert the elements of the SVG to a list of mobjects.

```

```

223: (8)             Parameters
224: (8)             -----
225: (8)             svg
226: (12)            The parsed SVG file.
227: (8)
228: (8)             """
229: (8)             result = []
230: (12)            for shape in svg.elements():
231: (16)              if isinstance(shape, se.Group):
232: (12)                continue
233: (16)              elif isinstance(shape, se.Path):
234: (12)                mob = self.path_to_mobject(shape)
235: (16)              elif isinstance(shape, se.SimpleLine):
236: (12)                mob = self.line_to_mobject(shape)
237: (16)              elif isinstance(shape, se.Rect):
238: (12)                mob = self.rect_to_mobject(shape)
239: (16)              elif isinstance(shape, (se.Circle, se.Ellipse)):
240: (12)                mob = self.ellipse_to_mobject(shape)
241: (16)              elif isinstance(shape, se.Polygon):
242: (12)                mob = self.polygon_to_mobject(shape)
243: (16)              elif isinstance(shape, se.Polyline):
244: (12)                mob = self.polyline_to_mobject(shape)
245: (16)              elif isinstance(shape, se.Text):
246: (12)                mob = self.text_to_mobject(shape)
247: (16)              elif isinstance(shape, se.Use) or type(shape) is se.SVGElement:
248: (12)                continue
249: (16)              else:
250: (16)                logger.warning(f"Unsupported element type: {type(shape)}")
251: (12)                continue
252: (16)              if mob is None or not mob.has_points():
253: (12)                continue
254: (12)              self.apply_style_to_mobject(mob, shape)
255: (16)              if isinstance(shape, se.Transformable) and shape.apply:
256: (12)                self.handle_transform(mob, shape.transform)
257: (8)              result.append(mob)
258: (4)            return result
259: (4) @staticmethod
260: (8) def handle_transform(mob: VMobject, matrix: se.Matrix) -> VMobject:
261: (8)     """Apply SVG transformations to the converted mobject.
262: (8)     Parameters
263: (8)     -----
264: (12)     mob
265: (8)       The converted mobject.
266: (12)     matrix
267: (12)       The transformation matrix determined from the SVG
268: (8)       transformation.
269: (8)     """
270: (8)     mat = np.array([[matrix.a, matrix.c], [matrix.b, matrix.d]])
271: (8)     vec = np.array([matrix.e, matrix.f, 0.0])
272: (8)     mob.apply_matrix(mat)
273: (8)     mob.shift(vec)
274: (4)   return mob
275: (4) @staticmethod
276: (8) def apply_style_to_mobject(mob: VMobject, shape: se.GraphicObject) ->
277: (8) VMobject:
278: (8)     """Apply SVG style information to the converted mobject.
279: (8)     Parameters
280: (8)     -----
281: (8)     mob
282: (12)       The converted mobject.
283: (8)     shape
284: (12)       The parsed SVG element.
285: (8)     """
286: (12)     mob.set_style(
287: (12)       stroke_width=shape.stroke_width,
288: (12)       stroke_color=shape.stroke.hexrgb,
289: (12)       stroke_opacity=shape.stroke.opacity,
290: (12)       fill_color=shape.fill.hexrgb,
291: (12)       fill_opacity=shape.fill.opacity,
292: (8)     )

```

```

291: (8)             return mob
292: (4)         def path_to_mob(self, path: se.Path) -> VMobjectFromSVGPath:
293: (8)             """Convert a path element to a vectorized mob.
294: (8)             Parameters
295: (8)             -----
296: (8)             path
297: (12)                 The parsed SVG path.
298: (8)
299: (8)             return VMobjectFromSVGPath(path, **self.path_string_config)
300: (4)         @staticmethod
301: (4)         def line_to_mob(self, line: se.Line) -> Line:
302: (8)             """Convert a line element to a vectorized mob.
303: (8)             Parameters
304: (8)             -----
305: (8)             line
306: (12)                 The parsed SVG line.
307: (8)
308: (8)             return Line(
309: (12)                 start=_convert_point_to_3d(line.x1, line.y1),
310: (12)                 end=_convert_point_to_3d(line.x2, line.y2),
311: (8)             )
312: (4)         @staticmethod
313: (4)         def rect_to_mob(self, rect: se.Rect) -> Rectangle:
314: (8)             """Convert a rectangle element to a vectorized mob.
315: (8)             Parameters
316: (8)             -----
317: (8)             rect
318: (12)                 The parsed SVG rectangle.
319: (8)
320: (8)             if rect.rx == 0 or rect.ry == 0:
321: (12)                 mob = Rectangle(
322: (16)                     width=rect.width,
323: (16)                     height=rect.height,
324: (12)                 )
325: (8)             else:
326: (12)                 mob = RoundedRectangle(
327: (16)                     width=rect.width,
328: (16)                     height=rect.height * rect.rx / rect.ry,
329: (16)                     corner_radius=rect.rx,
330: (12)                 )
331: (12)                 mob.stretch_to_fit_height(rect.height)
332: (8)             mob.shift(
333: (12)                 _convert_point_to_3d(rect.x + rect.width / 2, rect.y + rect.height
334: (8) / 2)
335: (8)             )
336: (4)             return mob
337: (4)         @staticmethod
338: (4)         def ellipse_to_mob(self, ellipse: se.Ellipse | se.Circle) -> Circle:
339: (8)             """Convert an ellipse or circle element to a vectorized mob.
340: (8)             Parameters
341: (8)             -----
342: (12)             ellipse
343: (8)                 The parsed SVG ellipse or circle.
344: (8)
345: (8)             mob = Circle(radius=ellipse.rx)
346: (12)             if ellipse.rx != ellipse.ry:
347: (8)                 mob.stretch_to_fit_height(2 * ellipse.ry)
348: (8)             mob.shift(_convert_point_to_3d(ellipse.cx, ellipse.cy))
349: (4)             return mob
350: (4)         @staticmethod
351: (4)         def polygon_to_mob(self, polygon: se.Polygon) -> Polygon:
352: (8)             """Convert a polygon element to a vectorized mob.
353: (8)             Parameters
354: (8)             -----
355: (12)             polygon
356: (8)                 The parsed SVG polygon.
357: (8)
358: (8)             points = [_convert_point_to_3d(*point) for point in polygon]
            return Polygon(*points)

```

```

359: (4)             def polyline_to_mobject(self, polyline: se.Polyline) -> VMobject:
360: (8)                 """Convert a polyline element to a vectorized mobject.
361: (8)                 Parameters
362: (8)                 -----
363: (8)                 polyline
364: (12)                    The parsed SVG polyline.
365: (8)
366: (8)                 points = [_convert_point_to_3d(*point) for point in polyline]
367: (8)                 vmobject_class = self.get_mobject_type_class()
368: (8)                 return vmobject_class().set_points_as_corners(points)
369: (4)             @staticmethod
370: (4)             def text_to_mobject(text: se.Text):
371: (8)                 """Convert a text element to a vectorized mobject.
372: (8)                 .. warning::
373: (12)                     Not yet implemented.
374: (8)             Parameters
375: (8)
376: (8)             text
377: (12)                    The parsed SVG text.
378: (8)
379: (8)                 logger.warning(f"Unsupported element type: {type(text)}")
380: (8)             return
381: (4)             def move_into_position(self) -> None:
382: (8)                 """Scale and move the generated mobject into position."""
383: (8)                 if self.should_center:
384: (12)                     self.center()
385: (8)                 if self.svg_height is not None:
386: (12)                     self.set(height=self.svg_height)
387: (8)                 if self.svg_width is not None:
388: (12)                     self.set(width=self.svg_width)
389: (0)             class VMobjectFromSVGPath(VMobject, metaclass=ConvertToOpenGL):
390: (4)                 """A vectorized mobject representing an SVG path.
391: (4)                 .. note::
392: (8)                     The ``long_lines``, ``should_subdivide_sharp_curves``,
393: (8)                     and ``should_remove_null_curves`` keyword arguments are
394: (8)                     only respected with the OpenGL renderer.
395: (4)             Parameters
396: (4)
397: (4)             path_obj
398: (8)                 A parsed SVG path object.
399: (4)             long_lines
400: (8)                 Whether or not straight lines in the vectorized mobject
401: (8)                 are drawn in one or two segments.
402: (4)             should_subdivide_sharp_curves
403: (8)                 Whether or not to subdivide subcurves further in case
404: (8)                 two segments meet at an angle that is sharper than a
405: (8)                 given threshold.
406: (4)             should_remove_null_curves
407: (8)                 Whether or not to remove subcurves of length 0.
408: (4)             kwargs
409: (8)                 Further keyword arguments are passed to the parent
410: (8)                 class.
411: (4)
412: (4)             def __init__(
413: (8)                 self,
414: (8)                 path_obj: se.Path,
415: (8)                 long_lines: bool = False,
416: (8)                 should_subdivide_sharp_curves: bool = False,
417: (8)                 should_remove_null_curves: bool = False,
418: (8)                 **kwargs,
419: (4)             ):
420: (8)                 path_obj.approximate_arcs_with_quads()
421: (8)                 self.path_obj = path_obj
422: (8)                 self.long_lines = long_lines
423: (8)                 self.should_subdivide_sharp_curves = should_subdivide_sharp_curves
424: (8)                 self.should_remove_null_curves = should_remove_null_curves
425: (8)                 super().__init__(**kwargs)
426: (4)             def init_points(self) -> None:
427: (8)                 self.handle_commands()

```

```

428: (8)             if config.renderer == "opengl":
429: (12)             if self.should_subdivide_sharp_curves:
430: (16)                 self.subdivide_sharp_curves()
431: (12)             if self.should_remove_null_curves:
432: (16)                 self.set_points(self.get_points_without_null_curves())
433: (4)             generate_points = init_points
434: (4)         def handle_commands(self) -> None:
435: (8)             all_points: list[np.ndarray] = []
436: (8)             last_move = None
437: (8)             curve_start = None
438: (8)             last_true_move = None
439: (8)             def move_pen(pt, *, true_move: bool = False):
440: (12)                 nonlocal last_move, curve_start, last_true_move
441: (12)                 last_move = pt
442: (12)                 if curve_start is None:
443: (16)                     curve_start = last_move
444: (12)                 if true_move:
445: (16)                     last_true_move = last_move
446: (8)             if self.n_points_per_curve == 4:
447: (12)                 def add_cubic(start, cp1, cp2, end):
448: (16)                     nonlocal all_points
449: (16)                     assert len(all_points) % 4 == 0, len(all_points)
450: (16)                     all_points += [start, cp1, cp2, end]
451: (16)                     move_pen(end)
452: (12)                 def add_quad(start, cp, end):
453: (16)                     add_cubic(start, (start + cp + cp) / 3, (cp + cp + end) / 3,
454: (16)                         move_pen(end)
455: (12)                 def add_line(start, end):
456: (16)                     add_cubic(
457: (20)                         start, (start + start + end) / 3, (start + end + end) / 3,
458: (16)                         )
459: (16)                     move_pen(end)
460: (8)             else:
461: (12)                 def add_cubic(start, cp1, cp2, end):
462: (16)                     nonlocal all_points
463: (16)                     assert len(all_points) % 3 == 0, len(all_points)
464: (16)                     two_quads = get_quadratic_approximation_of_cubic(
465: (20)                         start,
466: (20)                         cp1,
467: (20)                         cp2,
468: (20)                         end,
469: (16)                     )
470: (16)                     all_points += two_quads[:3].tolist()
471: (16)                     all_points += two_quads[3:].tolist()
472: (16)                     move_pen(end)
473: (12)                 def add_quad(start, cp, end):
474: (16)                     nonlocal all_points
475: (16)                     assert len(all_points) % 3 == 0, len(all_points)
476: (16)                     all_points += [start, cp, end]
477: (16)                     move_pen(end)
478: (12)                 def add_line(start, end):
479: (16)                     add_quad(start, (start + end) / 2, end)
480: (16)                     move_pen(end)
481: (8)             for segment in self.path_obj:
482: (12)                 segment_class = segment.__class__
483: (12)                 if segment_class == se.Move:
484: (16)                     move_pen(_convert_point_to_3d(*segment.end), true_move=True)
485: (12)                 elif segment_class == se.Line:
486: (16)                     add_line(last_move, _convert_point_to_3d(*segment.end))
487: (12)                 elif segment_class == se.QuadraticBezier:
488: (16)                     add_quad(
489: (20)                         last_move,
490: (20)                         _convert_point_to_3d(*segment.control),
491: (20)                         _convert_point_to_3d(*segment.end),
492: (16)                     )
493: (12)                 elif segment_class == se.CubicBezier:
494: (16)                     add_cubic(

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
495: (20)                               last_move,
496: (20)                               _convert_point_to_3d(*segment.control1),
497: (20)                               _convert_point_to_3d(*segment.control2),
498: (20)                               _convert_point_to_3d(*segment.end),
499: (16)
500: (12)                           )
501: (16)                           elif segment_class == se.Close:
502: (20)                               if abs(np.linalg.norm(last_move - last_true_move)) > 0.0001:
503: (16)                                   add_line(last_move, last_true_move)
504: (12)                                   curve_start = None
505: (16)                           else:
506: (8)                               raise AssertionError(f"Not implemented: {segment_class}")
507: (8)                           self.points = np.array(all_points, ndmin=2, dtype="float64")
508: (12)                           if len(all_points) == 0:
509: (12)                               self.points = np.reshape(self.points, (0, 3))

-----
```

## File 83 - tex\_mobject.py:

```
1: (0)             r"""Mobjects representing text rendered using LaTeX.
2: (0)             .. important::
3: (3)                 See the corresponding tutorial :ref:`rendering-with-latex`  

4: (0)             .. note::
5: (3)                 Just as you can use :class:`~.Text` (from the module :mod:`~.text_mobject`)  

to add text to your videos, you can use :class:`~.Tex` and :class:`~.MathTex` to insert LaTeX.  

6: (0)             """
7: (0)             from __future__ import annotations
8: (0)             from manim.utils.color import ManimColor
9: (0)             __all__ = [
10: (4)                 "SingleStringMathTex",
11: (4)                 "MathTex",
12: (4)                 "Tex",
13: (4)                 "BulletedList",
14: (4)                 "Title",
15: (0)
16: (0)             ]
17: (0)             import itertools as it
18: (0)             import operator as op
19: (0)             import re
20: (0)             from functools import reduce
21: (0)             from textwrap import dedent
22: (0)             from typing import Iterable
23: (0)             from manim import config, logger
24: (0)             from manim.constants import *
25: (0)             from manim.mobject.geometry.line import Line
26: (0)             from manim.mobject.svg.svg_mobject import SVGObject
27: (0)             from manim.mobject.types.vectorized_mobject import VGroup, VMobject
28: (0)             from manim.utils.tex import TexTemplate
29: (0)             from manim.utils.tex_file_writing import tex_to_svg_file
30: (0)             tex_string_to_mob_map = {}
31: (0)             class SingleStringMathTex(SVGObject):
32: (4)                 """Elementary building block for rendering text with LaTeX.
33: (4)                 Tests
34: (4)                 -----
35: (4)                 Check that creating a :class:`~.SingleStringMathTex` object works::
36: (8)                     >>> SingleStringMathTex('Test') # doctest: +SKIP
37: (8)                     SingleStringMathTex('Test')
38: (4)
39: (8)                     def __init__(
40: (8)                         self,
41: (8)                         tex_string: str,
42: (8)                         stroke_width: float = 0,
43: (8)                         should_center: bool = True,
44: (8)                         height: float | None = None,
45: (8)                         organize_left_to_right: bool = False,
46: (8)                         tex_environment: str = "align*",
47: (8)                         tex_template: TexTemplate | None = None,
48: (8)                         font_size: float = DEFAULT_FONT_SIZE,
49: (8)                         **kwargs,
50: (4)                     ):
```

```

50: (8)             if kwargs.get("color") is None:
51: (12)             kwargs["color"] = VMobject().color
52: (8)             self._font_size = font_size
53: (8)             self.organize_left_to_right = organize_left_to_right
54: (8)             self.tex_environment = tex_environment
55: (8)             if tex_template is None:
56: (12)                 tex_template = config["tex_template"]
57: (8)             self.tex_template = tex_template
58: (8)             assert isinstance(tex_string, str)
59: (8)             self.tex_string = tex_string
60: (8)             file_name = tex_to_svg_file(
61: (12)                 self._get_modified_expression(tex_string),
62: (12)                 environment=self.tex_environment,
63: (12)                 tex_template=self.tex_template,
64: (8)             )
65: (8)             super().__init__(
66: (12)                 file_name=file_name,
67: (12)                 should_center=should_center,
68: (12)                 stroke_width=stroke_width,
69: (12)                 height=height,
70: (12)                 path_string_config={
71: (16)                     "should_subdivide_sharp_curves": True,
72: (16)                     "should_remove_null_curves": True,
73: (12)                 },
74: (12)                 **kwargs,
75: (8)             )
76: (8)             self.init_colors()
77: (8)             self.initial_height = self.height
78: (8)             if height is None:
79: (12)                 self.font_size = self._font_size
80: (8)             if self.organize_left_to_right:
81: (12)                 self._organize_submobjects_left_to_right()
82: (4)             def __repr__(self):
83: (8)                 return f"{type(self).__name__}({repr(self.tex_string)})"
84: (4)             @property
85: (4)             def font_size(self):
86: (8)                 """The font size of the tex mobject."""
87: (8)                 return self.height / self.initial_height / SCALE_FACTOR_PER_FONT_POINT
88: (4)             @font_size.setter
89: (4)             def font_size(self, font_val):
90: (8)                 if font_val <= 0:
91: (12)                     raise ValueError("font_size must be greater than 0.")
92: (8)                 elif self.height > 0:
93: (12)                     self.scale(font_val / self.font_size)
94: (4)             def _get_modified_expression(self, tex_string):
95: (8)                 result = tex_string
96: (8)                 result = result.strip()
97: (8)                 result = self._modify_special_strings(result)
98: (8)                 return result
99: (4)             def _modify_special_strings(self, tex):
100: (8)                 tex = tex.strip()
101: (8)                 should_add_filler = reduce(
102: (12)                     op.or_,
103: (12)                     [
104: (16)                         tex == "\\over",
105: (16)                         tex == "\\overline",
106: (16)                         tex == "\\sqrt",
107: (16)                         tex == "\\sqrt{",
108: (16)                         tex.endswith("_"),
109: (16)                         tex.endswith("^"),
110: (16)                         tex.endswith("dot"),
111: (12)                     ],
112: (8)                 )
113: (8)                 if should_add_filler:
114: (12)                     filler = "\\quad"
115: (12)                     tex += filler
116: (8)                     if tex == "\\substack":
117: (12)                         tex = "\\quad"
118: (8)                     if tex == "":

```

```

119: (12)           tex = "\quad"
120: (8)            if tex.startswith("\\\\"):
121: (12)              tex = tex.replace("\\\\", "\\quad\\\\")
122: (8)            num_lefts, num_rights = (
123: (12)                len([s for s in tex.split(substr)[1:] if s and s[0] in "(){}"
124: []|.\\\""])
125: (12)                    for substr in ("\\left", "\\right"))
126: (8)            )
127: (12)            if num_lefts != num_rights:
128: (12)                tex = tex.replace("\\left", "\\big")
129: (12)                tex = tex.replace("\\right", "\\big")
130: (8)            tex = self._remove_stray_braces(tex)
131: (12)            for context in ["array"]:
132: (12)                begin_in = ("\\begin{%s}" % context) in tex
133: (12)                end_in = ("\\end{%s}" % context) in tex
134: (16)                if begin_in ^ end_in:
135: (8)                    tex = ""
136: (4)            return tex
137: (8)        def _remove_stray_braces(self, tex):
138: (8)            r"""
139: Makes :class:`~.MathTex` resilient to unmatched braces.
140: This is important when the braces in the TeX code are spread over
141: multiple arguments as in, e.g., ``MathTex(r"e^{i", r"\tau} = 1")``.
142: """
143: (8)            num_lefts = tex.count("{") - tex.count("\\{") + tex.count("\\\\{")
144: (8)            num_rights = tex.count("}") - tex.count("\\}") + tex.count("\\\\}")
145: (12)            while num_rights > num_lefts:
146: (12)                tex = "{" + tex
147: (8)                num_lefts += 1
148: (12)            while num_lefts > num_rights:
149: (12)                tex = tex + "}"
150: (8)                num_rights += 1
151: (4)            return tex
152: (8)        def _organize_submobjects_left_to_right(self):
153: (8)            self.sort(lambda p: p[0])
154: (4)            return self
155: (8)        def get_tex_string(self):
156: (4)            return self.tex_string
157: (8)        def init_colors(self, propagate_colors=True):
158: (12)            if config.renderer == RendererType.OPENGL:
159: (8)                super().init_colors()
160: (12)            elif config.renderer == RendererType.CAIRO:
161: (0)                super().init_colors(propagate_colors=propagate_colors)
162: (4)    class MathTex(SingleStringMathTex):
163: (4)        r"""A string compiled with LaTeX in math mode.
164: Examples
165: -----
166: .. manim:: Formula
167: :save_last_frame:
168: class Formula(Scene):
169:     def construct(self):
170:         t = MathTex(r"\int_a^b f'(x) dx = f(b) - f(a)")
171:         self.add(t)
172: Tests
173: -----
174: Check that creating a :class:`~.MathTex` works::
175:     >>> MathTex('a^2 + b^2 = c^2') # doctest: +SKIP
176:     MathTex('a^2 + b^2 = c^2')
177: Check that double brace group splitting works correctly::
178:     >>> t1 = MathTex('{{ a }} + {{ b }} = {{ c }}') # doctest: +SKIP
179:     >>> len(t1.submobjects) # doctest: +SKIP
180:     5
181:     >>> t2 = MathTex(r"\frac{1}{a+b\sqrt{2}}") # doctest: +SKIP
182:     >>> len(t2.submobjects) # doctest: +SKIP
183:     1
184: """
185: (4)    def __init__(
186: (8)        self,
187: *tex_strings,

```

```

187: (8)             arg_separator: str = " ",
188: (8)             substrings_to_isolate: Iterable[str] | None = None,
189: (8)             tex_to_color_map: dict[str, ManimColor] = None,
190: (8)             tex_environment: str = "align*",
191: (8)             **kwargs,
192: (4):
193: (8)         self.tex_template = kwargs.pop("tex_template", config["tex_template"])
194: (8)         self.arg_separator = arg_separator
195: (8)         self.substrings_to_isolate = (
196: (12)             [] if substrings_to_isolate is None else substrings_to_isolate
197: (8)
198: (8)         )
199: (8)         self.tex_to_color_map = tex_to_color_map
200: (12)         if self.tex_to_color_map is None:
201: (8)             self.tex_to_color_map = {}
202: (8)         self.tex_environment = tex_environment
203: (8)         self.brace_notation_split_occurred = False
204: (8)         self.tex_strings = self._break_up_tex_strings(tex_strings)
205: (12)         try:
206: (16)             super().__init__(
207: (16)                 self.arg_separator.join(self.tex_strings),
208: (16)                 tex_environment=self.tex_environment,
209: (16)                 tex_template=self.tex_template,
210: (12)                 **kwargs,
211: (12)             )
212: (8)             self._break_up_by_substrings()
213: (12)         except ValueError as compilation_error:
214: (16)             if self.brace_notation_split_occurred:
215: (20)                 logger.error(
216: (24)                     dedent(
217: (24)                         """\
218: (24)                             A group of double braces, {{ ... }}, was detected in
219: (24)                             your string. Manim splits TeX strings at the double
220: (24)                             braces, which might have caused the current
221: (24)                             compilation error. If you didn't use the double brace
222: (24)                             split intentionally, add spaces between the braces to
223: (24)                             avoid the automatic splitting: {{ ... }} --> { { ... }
224: (20)                         },
225: (16)                     )
226: (12)                     raise compilation_error
227: (8)                     self.set_color_by_tex_to_color_map(self.tex_to_color_map)
228: (8)                     if self.organize_left_to_right:
229: (12)                         self._organize_submobjects_left_to_right()
230: (4)             def _break_up_tex_strings(self, tex_strings):
231: (8)                 pre_split_length = len(tex_strings)
232: (8)                 tex_strings = [re.split("{{(.*?)}", str(t)) for t in tex_strings]
233: (8)                 tex_strings = sum(tex_strings, [])
234: (8)                 if len(tex_strings) > pre_split_length:
235: (12)                     self.brace_notation_split_occurred = True
236: (8)                 patterns = []
237: (8)                 patterns.extend(
238: (12)                     [
239: (16)                         f"({re.escape(ss)})"
240: (16)                         for ss in it.chain(
241: (20)                             self.substrings_to_isolate,
242: (20)                             self.tex_to_color_map.keys(),
243: (16)                         )
244: (12)                     ],
245: (8)
246: (8)                     pattern = "|".join(patterns)
247: (8)                     if pattern:
248: (12)                         pieces = []
249: (12)                         for s in tex_strings:
250: (16)                             pieces.extend(re.split(pattern, s))
251: (8)
252: (12)                         pieces = tex_strings
253: (8)                     return [p for p in pieces if p]
254: (4)             def _break_up_by_substrings(self):

```

```

255: (8)             """
256: (8)             Reorganize existing submobjects one layer
257: (8)             deeper based on the structure of tex_strings (as a list
258: (8)             of tex_strings)
259: (8)             """
260: (8)             new_submobjects = []
261: (8)             curr_index = 0
262: (8)             for tex_string in self.tex_strings:
263: (12)                 sub_tex_mob = SingleStringMathTex(
264: (16)                     tex_string,
265: (16)                     tex_environment=self.tex_environment,
266: (16)                     tex_template=self.tex_template,
267: (12)             )
268: (12)             num_submobs = len(sub_tex_mob.submobjects)
269: (12)             new_index = (
270: (16)                 curr_index + num_submobs +
len("".join(self.arg_separator.split())))
271: (12)             )
272: (12)             if num_submobs == 0:
273: (16)                 last_submob_index = min(curr_index, len(self.submobjects) - 1)
274: (16)                 sub_tex_mob.move_to(self.submobjects[last_submob_index],
RIGHT)
275: (12)             else:
276: (16)                 sub_tex_mob.submobjects =
self.submobjects[curr_index:new_index]
277: (12)                     new_submobjects.append(sub_tex_mob)
278: (12)                     curr_index = new_index
279: (8)                     self.submobjects = new_submobjects
280: (8)             return self
281: (4)             def get_parts_by_tex(self, tex, substring=True, case_sensitive=True):
282: (8)                 def test(tex1, tex2):
283: (12)                     if not case_sensitive:
284: (16)                         tex1 = tex1.lower()
285: (16)                         tex2 = tex2.lower()
286: (12)                     if substring:
287: (16)                         return tex1 in tex2
288: (12)                     else:
289: (16)                         return tex1 == tex2
290: (8)             return VGroup(*[m for m in self.submobjects if test(tex,
m.get_tex_string())))
291: (4)             def get_part_by_tex(self, tex, **kwargs):
292: (8)                 all_parts = self.get_parts_by_tex(tex, **kwargs)
293: (8)                 return all_parts[0] if all_parts else None
294: (4)             def set_color_by_tex(self, tex, color, **kwargs):
295: (8)                 parts_to_color = self.get_parts_by_tex(tex, **kwargs)
296: (8)                 for part in parts_to_color:
297: (12)                     part.set_color(color)
298: (8)                 return self
299: (4)             def set_opacity_by_tex(
300: (8)                 self, tex: str, opacity: float = 0.5, remaining_opacity: float = None,
**kwargs
301: (4)             ):
302: (8)                 """
303: (8)                 Sets the opacity of the tex specified. If 'remaining_opacity' is
specified,
304: (8)                 then the remaining tex will be set to that opacity.
305: (8)                 Parameters
306: (8)                 -----
307: (8)                 tex
308: (12)                     The tex to set the opacity of.
309: (8)                 opacity
310: (12)                     Default 0.5. The opacity to set the tex to
311: (8)                 remaining_opacity
312: (12)                     Default None. The opacity to set the remaining tex to.
313: (12)                     If None, then the remaining tex will not be changed
314: (8)                     """
315: (8)                     if remaining_opacity is not None:
316: (12)                         self.set_opacity(opacity=remaining_opacity)
317: (8)                     for part in self.get_parts_by_tex(tex):

```

```

318: (12)           part.set_opacity(opacity)
319: (8)            return self
320: (4)            def set_color_by_tex_to_color_map(self, texs_to_color_map, **kwargs):
321: (8)              for texs, color in list(texs_to_color_map.items()):
322: (12)                try:
323: (16)                  texs + ""
324: (16)                  self.set_color_by_tex(texs, color, **kwargs)
325: (12)                except TypeError:
326: (16)                  for tex in texs:
327: (20)                      self.set_color_by_tex(tex, color, **kwargs)
328: (8)            return self
329: (4)            def index_of_part(self, part):
330: (8)              split_self = self.split()
331: (8)              if part not in split_self:
332: (12)                  raise ValueError("Trying to get index of part not in MathTex")
333: (8)              return split_self.index(part)
334: (4)            def index_of_part_by_tex(self, tex, **kwargs):
335: (8)              part = self.get_part_by_tex(tex, **kwargs)
336: (8)              return self.index_of_part(part)
337: (4)            def sort_alphabetically(self):
338: (8)              self.submobjects.sort(key=lambda m: m.get_tex_string())
339: (0)            class Tex(MathTex):
340: (4)              """A string compiled with LaTeX in normal mode.
341: (4)              Tests
342: (4)              -----
343: (4)              Check whether writing a LaTeX string works::
344: (8)                  >>> Tex('The horse does not eat cucumber salad.') # doctest: +SKIP
345: (8)                  Tex('The horse does not eat cucumber salad.')
346: (4)              """
347: (4)            def __init__(
348: (8)              self, *tex_strings, arg_separator="", tex_environment="center",
**kwargs
349: (4)
350: (8)
351: (12)
352: (12)
353: (12)
354: (12)
355: (8)
356: (0)
357: (4)
358: (4)
359: (4)
360: (4)
361: (8)
362: (8)
363: (12)
364: (16)
width=2)
365: (16)
366: (16)
367: (16)
368: (16)
369: (4)
370: (4)
371: (8)
372: (8)
373: (8)
374: (8)
375: (8)
376: (8)
377: (4)
378: (8)
379: (8)
380: (8)
381: (8)
382: (8)
383: (12)
384: (8)
):
    super().__init__(
        *tex_strings,
        arg_separator=arg_separator,
        tex_environment=tex_environment,
        **kwargs,
    )
class BulletedList(Tex):
    """A bulleted list.
Examples
-----
.. manim:: BulletedListExample
:save_last_frame:
    class BulletedListExample(Scene):
        def construct(self):
            blist = BulletedList("Item 1", "Item 2", "Item 3", height=2,
width=2)
            blist.set_color_by_tex("Item 1", RED)
            blist.set_color_by_tex("Item 2", GREEN)
            blist.set_color_by_tex("Item 3", BLUE)
            self.add(blist)
"""
    def __init__(
        self,
        *items,
        buff=MED_LARGE_BUFF,
        dot_scale_factor=2,
        tex_environment=None,
        **kwargs,
    ):
        self.buff = buff
        self.dot_scale_factor = dot_scale_factor
        self.tex_environment = tex_environment
        line_separated_items = [s + "\\\\" for s in items]
        super().__init__(
            *line_separated_items, tex_environment=tex_environment, **kwargs
        )

```

```

385: (8)             for part in self:
386: (12)             dot = MathTex("\cdot").scale(self.dot_scale_factor)
387: (12)             dot.next_to(part[0], LEFT, SMALL_BUFF)
388: (12)             part.add_to_back(dot)
389: (8)             self.arrange(DOWN, aligned_edge=LEFT, buff=self.buff)
390: (4)         def fade_all_but(self, index_or_string, opacity=0.5):
391: (8)             arg = index_or_string
392: (8)             if isinstance(arg, str):
393: (12)                 part = self.get_part_by_tex(arg)
394: (8)             elif isinstance(arg, int):
395: (12)                 part = self.submobjects[arg]
396: (8)             else:
397: (12)                 raise TypeError(f"Expected int or string, got {arg}")
398: (8)             for other_part in self.submobjects:
399: (12)                 if other_part is part:
400: (16)                     other_part.set_fill(opacity=1)
401: (12)                 else:
402: (16)                     other_part.set_fill(opacity=opacity)
403: (0)     class Title(Tex):
404: (4)         """A mobject representing an underlined title.
405: (4)         Examples
406: (4)         -----
407: (4)         .. manim:: TitleExample
408: (8)             :save_last_frame:
409: (8)             import manim
410: (8)             class TitleExample(Scene):
411: (12)                 def construct(self):
412: (16)                     banner = ManimBanner()
413: (16)                     title = Title(f"Manim version {manim.__version__}")
414: (16)                     self.add(banner, title)
415: (4)         """
416: (4)         def __init__(
417: (8)             self,
418: (8)             *text_parts,
419: (8)             include_underline=True,
420: (8)             match_underline_width_to_text=False,
421: (8)             underline_buff=MED_SMALL_BUFF,
422: (8)             **kwargs,
423: (4)         ):
424: (8)             self.include_underline = include_underline
425: (8)             self.match_underline_width_to_text = match_underline_width_to_text
426: (8)             self.underline_buff = underline_buff
427: (8)             super().__init__(*text_parts, **kwargs)
428: (8)             self.to_edge(UP)
429: (8)             if self.include_underline:
430: (12)                 underline_width = config["frame_width"] - 2
431: (12)                 underline = Line(LEFT, RIGHT)
432: (12)                 underline.next_to(self, DOWN, buff=self.underline_buff)
433: (12)                 if self.match_underline_width_to_text:
434: (16)                     underline.match_width(self)
435: (12)                 else:
436: (16)                     underline.width = underline_width
437: (12)                 self.add(underline)
438: (12)                 self.underline = underline

```

-----

## File 84 - code\_mobject.py:

```

1: (0)             """Mobject representing highlighted source code listings."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "Code",
5: (0)             ]
6: (0)             import html
7: (0)             import os
8: (0)             import re
9: (0)             from pathlib import Path
10: (0)            import numpy as np

```

```

11: (0)         from pygments import highlight
12: (0)         from pygments.formatters.html import HtmlFormatter
13: (0)         from pygments.lexers import get_lexer_by_name, guess_lexer_for_filename
14: (0)         from pygments.styles import get_all_styles
15: (0)         from manim import logger
16: (0)         from manim.constants import *
17: (0)         from manim.mobject.geometry.arc import Dot
18: (0)         from manim.mobject.geometry.polygram import RoundedRectangle
19: (0)         from manim.mobject.geometry.shape_matchers import SurroundingRectangle
20: (0)         from manim.mobject.text.text_mobject import Paragraph
21: (0)         from manim.mobject.types.vectorized_mobject import VGroup
22: (0)         from manim.utils.color import WHITE
23: (0)         __all__ = ["Code"]
24: (0) class Code(VGroup):
25: (4)     """A highlighted source code listing.
26: (4)     An object ``listing`` of :class:`.Code` is a :class:`.VGroup` consisting
27: (4)     of three objects:
28: (4)     - The background, ``listing.background_mobject``. This is either
29: (6)         a :class:`.Rectangle` (if the listing has been initialized with
30: (6)         ``background="rectangle"``), the default option) or a :class:`.VGroup`
31: (6)         resembling a window (if ``background="window"`` has been passed).
32: (4)     - The line numbers, ``listing.line_numbers`` (a :class:`.Paragraph`
33: (6)         object).
34: (4)     - The highlighted code itself, ``listing.code`` (a :class:`.Paragraph`
35: (6)         object).
36: (4) .. WARNING::
37: (8)     Using a :class:`.Transform` on text with leading whitespace (and in
38: (8)     this particular case: code) can look
39: (8)     `weird <https://github.com/3b1b/manim/issues/1067>`. Consider using
40: (8)     :meth:`remove_invisible_chars` to resolve this issue.
41: (4) Examples
42: (4) -----
43: (4) Normal usage::
44: (8)     listing = Code(
45: (12)         "helloworldcpp.cpp",
46: (12)         tab_width=4,
47: (12)         background_stroke_width=1,
48: (12)         background_stroke_color=WHITE,
49: (12)         insert_line_no=True,
50: (12)         style=Code.styles_list[15],
51: (12)         background="window",
52: (12)         language="cpp",
53: (8)
54: (4)     We can also render code passed as a string (but note that
55: (4)     the language has to be specified in this case):
56: (4) .. manim:: CodeFromString
57: (8)     :save_last_frame:
58: (8)     class CodeFromString(Scene):
59: (12)         def construct(self):
60: (16)             code = '''from manim import Scene, Square
61: (8)             class FadeInSquare(Scene):
62: (12)                 def construct(self):
63: (16)                     s = Square()
64: (16)                     self.play(FadeIn(s))
65: (16)                     self.play(s.animate.scale(2))
66: (16)                     self.wait()
67: (8)             ...
68: (16)             rendered_code = Code(code=code, tab_width=4,
69: (36)   language="Python", font="Monospace")
70: (16)             self.add(rendered_code)
71: (4)         Parameters
72: (4) -----
73: (4)         file_name
74: (8)             Name of the code file to display.
75: (4)         code
76: (8)             If ``file_name`` is not specified, a code string can be
77: (8)             passed directly.
78: (4)         tab_width

```

```

79: (8)           Number of space characters corresponding to a tab character. Defaults
to 3.
80: (4)           line_spacing
81: (8)           Amount of space between lines in relation to font size. Defaults to
0.3, which means 30% of font size.
82: (4)           font_size
83: (8)           A number which scales displayed code. Defaults to 24.
84: (4)           font
85: (8)           The name of the text font to be used. Defaults to ``"Monospace"``.
86: (8)           This is either a system font or one loaded with
`text.register_font()`. Note
87: (8)           that font family names may be different across operating systems.
88: (4)           stroke_width
89: (8)           Stroke width for text. 0 is recommended, and the default.
90: (4)           margin
91: (8)           Inner margin of text from the background. Defaults to 0.3.
92: (4)           indentation_chars
93: (8)           "Indentation chars" refers to the spaces/tabs at the beginning of a
given code line. Defaults to ``"    `` (spaces).
94: (4)           background
95: (8)           Defines the background's type. Currently supports only ``"rectangle"``.
(default) and ``"window"``.
96: (4)           background_stroke_width
97: (8)           Defines the stroke width of the background. Defaults to 1.
98: (4)           background_stroke_color
99: (8)           Defines the stroke color for the background. Defaults to ``WHITE``.
100: (4)          corner_radius
101: (8)          Defines the corner radius for the background. Defaults to 0.2.
102: (4)          insert_line_no
103: (8)          Defines whether line numbers should be inserted in displayed code.
Defaults to ``True``.
104: (4)          line_no_from
105: (8)          Defines the first line's number in the line count. Defaults to 1.
106: (4)          line_no_buff
107: (8)          Defines the spacing between line numbers and displayed code. Defaults
to 0.4.
108: (4)          style
109: (8)          Defines the style type of displayed code. You can see possible names
of styles in with :attr:`styles_list`. Defaults to ``"vim"``.
110: (4)          language
111: (8)          Specifies the programming language the given code was written in. If
``None``
112: (8)          (the default), the language will be automatically detected. For the
list of
113: (8)          possible options, visit https://pygments.org/docs/lexers/ and look for
114: (8)          'aliases or short names'.
115: (4)          generate_html_file
116: (8)          Defines whether to generate highlighted html code to the folder
`assets/codes/generated_html_files`. Defaults to `False`.
117: (4)          warn_missing_font
118: (8)          If True (default), Manim will issue a warning if the font does not
exist in the
119: (8)          (case-sensitive) list of fonts returned from
`manimpango.list_fonts()`.

120: (4)          Attributes
121: (4)          -----
122: (4)          background_mobject : :class:`~.VGroup`
123: (8)          The background of the code listing.
124: (4)          line_numbers : :class:`~.Paragraph`
125: (8)          The line numbers for the code listing. Empty, if
126: (8)          ``insert_line_no=False`` has been specified.
127: (4)          code : :class:`~.Paragraph`
128: (8)          The highlighted code.
129: (4)          """
130: (4)          styles_list = list(get_all_styles())
131: (4)          def __init__(
132: (8)              self,
133: (8)              file_name: str | os.PathLike | None = None,
134: (8)              code: str | None = None,

```

```

135: (8)                     tab_width: int = 3,
136: (8)                     line_spacing: float = 0.3,
137: (8)                     font_size: float = 24,
138: (8)                     font: str = "Monospace", # This should be in the font list on all
platforms.
139: (8)                     stroke_width: float = 0,
140: (8)                     margin: float = 0.3,
141: (8)                     indentation_chars: str = "    ",
142: (8)                     background: str = "rectangle", # or window
143: (8)                     background_stroke_width: float = 1,
144: (8)                     background_stroke_color: str = WHITE,
145: (8)                     corner_radius: float = 0.2,
146: (8)                     insert_line_no: bool = True,
147: (8)                     line_no_from: int = 1,
148: (8)                     line_no_buff: float = 0.4,
149: (8)                     style: str = "vim",
150: (8)                     language: str | None = None,
151: (8)                     generate_html_file: bool = False,
152: (8)                     warn_missing_font: bool = True,
153: (8)                     **kwargs,
154: (4)
155: (8)                 ):
156: (12)                     super().__init__(
157: (12)                         stroke_width=stroke_width,
158: (8)                         **kwargs,
159: (8)                     )
160: (8)                     self.background_stroke_color = background_stroke_color
161: (8)                     self.background_stroke_width = background_stroke_width
162: (8)                     self.tab_width = tab_width
163: (8)                     self.line_spacing = line_spacing
164: (8)                     self.warn_missing_font = warn_missing_font
165: (8)                     self.font = font
166: (8)                     self.font_size = font_size
167: (8)                     self.margin = margin
168: (8)                     self.indentation_chars = indentation_chars
169: (8)                     self.background = background
170: (8)                     self.corner_radius = corner_radius
171: (8)                     self.insert_line_no = insert_line_no
172: (8)                     self.line_no_from = line_no_from
173: (8)                     self.line_no_buff = line_no_buff
174: (8)                     self.style = style
175: (8)                     self.language = language
176: (8)                     self.generate_html_file = generate_html_file
177: (8)                     self.file_path = None
178: (8)                     self.file_name = file_name
179: (12)                     if self.file_name:
180: (12)                         self._ensure_valid_file()
181: (8)                         self.code_string = self.file_path.read_text(encoding="utf-8")
182: (12)                     elif code:
183: (8)                         self.code_string = code
184: (12)                     else:
185: (16)                         raise ValueError(
186: (12)                             "Neither a code file nor a code string have been specified.",
187: (8)                         )
188: (12)                     if isinstance(self.style, str):
189: (8)                         self.style = self.style.lower()
190: (8)                     self._gen_html_string()
191: (8)                     strati = self.html_string.find("background:")
192: (8)                     self.background_color = self.html_string[strati + 12 : strati + 19]
193: (8)                     self._gen_code_json()
194: (8)                     self.code = self._gen_colored_lines()
195: (12)                     if self.insert_line_no:
196: (12)                         self.line_numbers = self._gen_line_numbers()
buff=self.line_no_buff)                         self.line_numbers.next_to(self.code, direction=LEFT,
197: (8)                     if self.background == "rectangle":
198: (12)                         if self.insert_line_no:
199: (16)                             foreground = VGroup(self.code, self.line_numbers)
200: (12)                         else:
foreground = self.code

```

```

202: (12)             rect = SurroundingRectangle(
203: (16)               foreground,
204: (16)               buff=self.margin,
205: (16)               color=self.background_color,
206: (16)               fill_color=self.background_color,
207: (16)               stroke_width=self.background_stroke_width,
208: (16)               stroke_color=self.background_stroke_color,
209: (16)               fill_opacity=1,
210: (12)           )
211: (12)           rect.round_corners(self.corner_radius)
212: (12)           self.background_mobject = rect
213: (8) else:
214: (12)     if self.insert_line_no:
215: (16)       foreground = VGroup(self.code, self.line_numbers)
216: (12)     else:
217: (16)       foreground = self.code
218: (12)     height = foreground.height + 0.1 * 3 + 2 * self.margin
219: (12)     width = foreground.width + 0.1 * 3 + 2 * self.margin
220: (12)     rect = RoundedRectangle(
221: (16)       corner_radius=self.corner_radius,
222: (16)       height=height,
223: (16)       width=width,
224: (16)       stroke_width=self.background_stroke_width,
225: (16)       stroke_color=self.background_stroke_color,
226: (16)       color=self.background_color,
227: (16)       fill_opacity=1,
228: (12)   )
229: (12)   red_button = Dot(radius=0.1, stroke_width=0, color="#ff5f56")
230: (12)   red_button.shift(LEFT * 0.1 * 3)
231: (12)   yellow_button = Dot(radius=0.1, stroke_width=0, color="#ffbd2e")
232: (12)   green_button = Dot(radius=0.1, stroke_width=0, color="#27c93f")
233: (12)   green_button.shift(RIGHT * 0.1 * 3)
234: (12)   buttons = VGroup(red_button, yellow_button, green_button)
235: (12)   buttons.shift(
236: (16)     UP * (height / 2 - 0.1 * 2 - 0.05)
237: (16)     + LEFT * (width / 2 - 0.1 * 5 - self.corner_radius / 2 -
0.05),
238: (12)   )
239: (12)   self.background_mobject = VGroup(rect, buttons)
240: (12)   x = (height - foreground.height) / 2 - 0.1 * 3
241: (12)   self.background_mobject.shift(foreground.get_center())
242: (12)   self.background_mobject.shift(UP * x)
243: (8) if self.insert_line_no:
244: (12)     super().__init__(
245: (16)       self.background_mobject, self.line_numbers, self.code,
**kwargs
246: (12)     )
247: (8) else:
248: (12)     super().__init__(
249: (16)       self.background_mobject,
250: (16)       Dot(fill_opacity=0, stroke_opacity=0),
251: (16)       self.code,
252: (16)       **kwargs,
253: (12)     )
254: (8)     self.move_to(np.array([0, 0, 0]))
255: (4) def _ensure_valid_file(self):
256: (8)     """Function to validate file."""
257: (8)     if self.file_name is None:
258: (12)         raise Exception("Must specify file for Code")
259: (8)     possible_paths = [
260: (12)       Path() / "assets" / "codes" / self.file_name,
261: (12)       Path(self.file_name).expanduser(),
262: (8)     ]
263: (8)     for path in possible_paths:
264: (12)       if path.exists():
265: (16)         self.file_path = path
266: (16)         return
267: (8)     error = (
268: (12)       f"From: {Path.cwd()}, could not find {self.file_name} at either "

```

```

269: (12)          + f"of these locations: {list(map(str, possible_paths))}"
270: (8)
271: (8)
272: (4)      def _gen_line_numbers(self):
273: (8)          """Function to generate line_numbers.
274: (8)          Returns
275: (8)          -----
276: (8)          :class:`~.Paragraph`
277: (12)          The generated line_numbers according to parameters.
278: (8)
279: (8)          line_numbers_array = []
280: (8)          for line_no in range(0, self.code_json.__len__()):
281: (12)              number = str(self.line_no_from + line_no)
282: (12)              line_numbers_array.append(number)
283: (8)          line_numbers = Paragraph(
284: (12)              *list(line_numbers_array),
285: (12)              line_spacing=self.line_spacing,
286: (12)              alignment="right",
287: (12)              font_size=self.font_size,
288: (12)              font=self.font,
289: (12)              disable_ligatures=True,
290: (12)              stroke_width=self.stroke_width,
291: (12)              warn_missing_font=self.warn_missing_font,
292: (8)
293: (8)
294: (12)          for i in line_numbers:
295: (8)              i.set_color(self.default_color)
296: (8)          return line_numbers
297: (4)      def _gen_colored_lines(self):
298: (8)          """Function to generate code.
299: (8)          Returns
300: (8)
301: (12)          :class:`~.Paragraph`
302: (8)          The generated code according to parameters.
303: (8)
304: (8)          lines_text = []
305: (12)          for line_no in range(0, self.code_json.__len__()):
306: (12)              line_str = ""
307: (16)              for word_index in range(self.code_json[line_no].__len__()):
308: (12)                  line_str = line_str + self.code_json[line_no][word_index][0]
309: (8)                  lines_text.append(self.tab_spaces[line_no] * "\t" + line_str)
310: (12)          code = Paragraph(
311: (12)              *list(lines_text),
312: (12)              line_spacing=self.line_spacing,
313: (12)              tab_width=self.tab_width,
314: (12)              font_size=self.font_size,
315: (12)              font=self.font,
316: (12)              disable_ligatures=True,
317: (12)              stroke_width=self.stroke_width,
318: (8)              warn_missing_font=self.warn_missing_font,
319: (8)
320: (12)          for line_no in range(code.__len__()):
321: (12)              line = code.chars[line_no]
322: (12)              line_char_index = self.tab_spaces[line_no]
323: (16)              for word_index in range(self.code_json[line_no].__len__()):
324: (20)                  line[
325: (20)                      line_char_index : line_char_index
326: (16)                          + self.code_json[line_no][word_index][0].__len__()
327: (16)                          ].set_color(self.code_json[line_no][word_index][1])
328: (8)                          line_char_index += self.code_json[line_no][word_index]
329: (4)          [0].__len__())
330: (8)
331: (8)          return code
332: (4)      def _gen_html_string(self):
333: (8)          """Function to generate html string with code highlighted and stores
334: in variable html_string."""
335: (8)          self.html_string = _hilite_me(
336: (12)              self.code_string,
337: (12)              self.language,
338: (12)              self.style,
339: (12)              self.insert_line_no,

```

```

336: (12)                                     "border:solid gray; border-width:.1em .1em .1em .8em; padding:.2em
.6em;",
337: (12)                                     self.file_path,
338: (12)                                     self.line_no_from,
339: (8)                                      )
340: (8)                                     if self.generate_html_file:
341: (12)   output_folder = Path() / "assets" / "codes" /
342: (12)   output_folder.mkdir(parents=True, exist_ok=True)
343: (12)   (output_folder / f"
{self.file_name}.html").write_text(self.html_string)
344: (4)                                     def _gen_code_json(self):
345: (8)   """Function to background_color, generate code_json and tab_spaces
from html_string.
346: (8)   background_color is just background color of displayed code.
347: (8)   code_json is 2d array with rows as line numbers
348: (8)   and columns as a array with length 2 having text and text's color
value.
349: (8)   tab_spaces is 2d array with rows as line numbers
350: (8)   and columns as corresponding number of indentation_chars in front of
that line in code.
351: (8)   """
352: (8)                                     if (
353: (12)   self.background_color == "#111111"
354: (12)   or self.background_color == "#272822"
355: (12)   or self.background_color == "#202020"
356: (12)   or self.background_color == "#000000"
357: (8)                                     ):
358: (12)   self.default_color = "#ffffff"
359: (8)                                     else:
360: (12)   self.default_color = "#000000"
361: (8)                                     for i in range(3, -1, -1):
362: (12)   self.html_string = self.html_string.replace("</> " * i, "</>")
363: (8)                                     self.html_string = self.html_string.replace("<span></span>", "")
364: (8)                                     for i in range(10, -1, -1):
365: (12)   self.html_string = self.html_string.replace(
366: (16)   "</span>" + " " * i,
367: (16)   " " * i + "</span>",
368: (12)                                     )
369: (8)                                     self.html_string = self.html_string.replace("background-color:",
"background:")
370: (8)                                     if self.insert_line_no:
371: (12)   start_point = self.html_string.find("</td><td><pre>")
372: (12)   start_point = start_point + 9
373: (8)                                     else:
374: (12)   start_point = self.html_string.find("<pre>")
375: (8)   self.html_string = self.html_string[start_point:]
376: (8)   lines = self.html_string.split("\n")
377: (8)   lines = lines[0 : lines.__len__() - 2]
378: (8)   start_point = lines[0].find(">")
379: (8)   lines[0] = lines[0][start_point + 1 :]
380: (8)   self.code_json = []
381: (8)   self.tab_spaces = []
382: (8)   code_json_line_index = -1
383: (8)   for line_index in range(0, lines.__len__()):
384: (12)   self.code_json.append([])
385: (12)   code_json_line_index = code_json_line_index + 1
386: (12)   if lines[line_index].startswith(self.indentation_chars):
387: (16)   start_point = lines[line_index].find("<")
388: (16)   starting_string = lines[line_index][:start_point]
389: (16)   indentation_chars_count = lines[line_index]
[:start_point].count(
390: (20)   self.indentation_chars,
391: (16)
392: (16)
393: (20)
394: (20)
self.indentation_chars.__len__()
395: (16)
):

```

```

396: (20)                         lines[line_index] = (
397: (24)                           "\t" * indentation_chars_count
398: (24)                           + starting_string[
399: (28)                             starting_string.rfind(self.indentation_chars)
400: (28)                             + self.indentation_chars.__len__() :
401: (24)                           ]
402: (24)                           + lines[line_index][start_point:]
403: (20)                         )
404: (16)                         else:
405: (20)                           lines[line_index] = (
406: (24)                             "\t" * indentation_chars_count + lines[line_index]
407: (20)                           )
408: (12)                         indentation_chars_count = 0
409: (12)                         if lines[line_index]:
410: (16)                           while lines[line_index][indentation_chars_count] == "\t":
411: (20)                             indentation_chars_count = indentation_chars_count + 1
412: (12)                           self.tab_spaces.append(indentation_chars_count)
413: (12)                           lines[line_index] = self._correct_non_span(lines[line_index])
414: (12)                           words = lines[line_index].split("<span")
415: (12)                           for word_index in range(1, words.__len__()):
416: (16)                             color_index = words[word_index].find("color:")
417: (16)                             if color_index == -1:
418: (20)                               color = self.default_color
419: (16)                             else:
420: (20)                               starti = words[word_index][color_index:].find("#")
421: (20)                               color = words[word_index][
422: (24)                                 color_index + starti : color_index + starti + 7
423: (20)                               ]
424: (16)                               start_point = words[word_index].find(">")
425: (16)                               end_point = words[word_index].find("</span>")
426: (16)                               text = words[word_index][start_point + 1 : end_point]
427: (16)                               text = html.unescape(text)
428: (16)                               if text != "":
429: (20)                                 self.code_json[code_json_line_index].append([text, color])
430: (4)                           def _correct_non_span(self, line_str: str):
431: (8)                             """Function put text color to those strings that don't have one
according to background_color of displayed code.
432: (8)                               Parameters
433: (8)                               -----
434: (8)                               line_str
435: (12)                                 Takes a html element's string to put color to it according to
background_color of displayed code.
436: (8)                               Returns
437: (8)                               -----
438: (8)                               :class:`str`
439: (12)                                 The generated html element's string with having color attributes.
440: (8)                               """
441: (8)                               words = line_str.split("</span>")
442: (8)                               line_str = ""
443: (8)                               for i in range(0, words.__len__()):
444: (12)                                 if i != words.__len__() - 1:
445: (16)                                   j = words[i].find("<span")
446: (12)                                 else:
447: (16)                                   j = words[i].__len__()
448: (12)                                 temp = ""
449: (12)                                 starti = -1
450: (12)                                 for k in range(0, j):
451: (16)                                   if words[i][k] == "\t" and starti == -1:
452: (20)                                     continue
453: (16)                                   else:
454: (20)                                     if starti == -1:
455: (24)                                       starti = k
456: (20)                                       temp = temp + words[i][k]
457: (12)                                   if temp != "":
458: (16)                                     if i != words.__len__() - 1:
459: (20)                                       temp = (
460: (24)   '<span style="color: '
461: (24)   + self.default_color

```

```

462: (24)                                + '">' 
463: (24)                                + words[i][starti:j]
464: (24)                                + "</span>" 
465: (20)                               )
466: (16)      else:
467: (20)          temp = (
468: (24)              '<span style="color: '
469: (24)              + self.default_color
470: (24)              + '">' 
471: (24)              + words[i][starti:j]
472: (20)          ) 
473: (16)          temp = temp + words[i][j:]
474: (16)          words[i] = temp
475: (12)          if words[i] != "":
476: (16)              line_str = line_str + words[i] + "</span>" 
477: (8)          return line_str
478: (0)      def _hilite_me(
479: (4)          code: str,
480: (4)          language: str,
481: (4)          style: str,
482: (4)          insert_line_no: bool,
483: (4)          divstyles: str,
484: (4)          file_path: Path,
485: (4)          line_no_from: int,
486: (0)      ):
487: (4)          """Function to highlight code from string to html.
488: (4)          Parameters
489: (4)          -----
490: (4)          code
491: (8)              Code string.
492: (4)          language
493: (8)              The name of the programming language the given code was written in.
494: (4)          style
495: (8)              Code style name.
496: (4)          insert_line_no
497: (8)              Defines whether line numbers should be inserted in the html file.
498: (4)          divstyles
499: (8)              Some html css styles.
500: (4)          file_path
501: (8)              Path of code file.
502: (4)          line_no_from
503: (8)              Defines the first line's number in the line count.
504: (4)          """
505: (4)          style = style or "colorful"
506: (4)          defstyles = "overflow:auto; width: auto;"
507: (4)          formatter = HtmlFormatter(
508: (8)              style=style,
509: (8)              linenos=False,
510: (8)              noclasses=True,
511: (8)              cssclass="",
512: (8)              cssstyles=defstyles + divstyles,
513: (8)              prestyles="margin: 0",
514: (4)          )
515: (4)          if language is None and file_path:
516: (8)              lexer = guess_lexer_for_filename(file_path, code)
517: (8)              html = highlight(code, lexer, formatter)
518: (4)          elif language is None:
519: (8)              raise ValueError(
520: (12)                  "The code language has to be specified when rendering a code
string",
521: (8)                  )
522: (4)          else:
523: (8)              html = highlight(code, get_lexer_by_name(language, **{}), formatter)
524: (4)          if insert_line_no:
525: (8)              html = _insert_line_numbers_in_html(html, line_no_from)
526: (4)              html = "<!-- HTML generated by Code() -->" + html
527: (4)          return html
528: (0)      def _insert_line_numbers_in_html(html: str, line_no_from: int):
529: (4)          """Function that inserts line numbers in the highlighted HTML code.

```

```

530: (4)             Parameters
531: (4)             -----
532: (4)             html
533: (8)             html string of highlighted code.
534: (4)             line_no_from
535: (8)             Defines the first line's number in the line count.
536: (4)             Returns
537: (4)             -----
538: (4)             :class:`str`
539: (8)             The generated html string with having line numbers.
540: (4)             """
541: (4)             match = re.search("<pre[^>]*>(.*)</pre>", html, re.DOTALL)
542: (4)             if not match:
543: (8)                 return html
544: (4)             pre_open = match.group(1)
545: (4)             pre = match.group(2)
546: (4)             pre_close = match.group(3)
547: (4)             html = html.replace(pre_close, "</pre></td></tr></table>")
548: (4)             numbers = range(line_no_from, line_no_from + pre.count("\n") + 1)
549: (4)             format_lines = "%" + str(len(str(numbers[-1]))) + "i"
550: (4)             lines = "\n".join(format_lines % i for i in numbers)
551: (4)             html = html.replace(
552: (8)                 pre_open,
553: (8)                 "<table><tr><td>" + pre_open + lines + "</pre></td><td>" + pre_open,
554: (4)             )
555: (4)             return html
-----
```

#### File 85 - text\_mobject.py:

```

1: (0)             """Mobjects used for displaying (non-LaTeX) text.
2: (0)             .. note::
3: (3)             Just as you can use :class:`~.Tex` and :class:`~.MathTex` (from the module
:mod:`~.tex_mobject`)
4: (3)             to insert LaTeX to your videos, you can use :class:`~.Text` to add
normal text.
5: (0)             .. important::
6: (3)             See the corresponding tutorial :ref:`using-text-objects`, especially for
information about fonts.
7: (0)             The simplest way to add text to your animations is to use the :class:`~.Text`-
class. It uses the Pango library to render text.
8: (0)             With Pango, you are also able to render non-English alphabets like `你好` or
`こんにちは` or `안녕하세요` or `مرحبا بالعالم`.
9: (0)             Examples
10: (0)             -----
11: (0)             .. manim:: HelloWorld
12: (4)             :save_last_frame:
13: (4)             class HelloWorld(Scene):
14: (8)             def construct(self):
15: (12)             text = Text('Hello world').scale(3)
16: (12)             self.add(text)
17: (0)             .. manim:: TextAlignment
18: (4)             :save_last_frame:
19: (4)             class TextAlignment(Scene):
20: (8)             def construct(self):
21: (12)             title = Text("K-means clustering and Logistic Regression",
color=WHITE)
22: (12)             title.scale(0.75)
23: (12)             self.add(title.to_edge(UP))
24: (12)             t1 = Text("1. Measuring").set_color(WHITE)
25: (12)             t2 = Text("2. Clustering").set_color(WHITE)
26: (12)             t3 = Text("3. Regression").set_color(WHITE)
27: (12)             t4 = Text("4. Prediction").set_color(WHITE)
28: (12)             x = VGroup(t1, t2, t3, t4).arrange(direction=DOWN,
aligned_edge=LEFT).scale(0.7).next_to(ORIGIN,DR)
29: (12)             x.set_opacity(0.5)
30: (12)             x.submobjects[1].set_opacity(1)
31: (12)             self.add(x)
```

```

32: (0)         """
33: (0)         from __future__ import annotations
34: (0)         import functools
35: (0)         __all__ = ["Text", "Paragraph", "MarkupText", "register_font"]
36: (0)         import copy
37: (0)         import hashlib
38: (0)         import os
39: (0)         import re
40: (0)         from contextlib import contextmanager
41: (0)         from itertools import chain
42: (0)         from pathlib import Path
43: (0)         from typing import Iterable, Sequence
44: (0)         import manimpango
45: (0)         import numpy as np
46: (0)         from manimpango import MarkupUtils, PangoUtils, TextSetting
47: (0)         from manim import config, logger
48: (0)         from manim.constants import *
49: (0)         from manim.mobject.geometry.arc import Dot
50: (0)         from manim.mobject.svg.svg_mobject import SVGObject
51: (0)         from manim.mobject.types.vectorized_mobject import VGroup, VMobject
52: (0)         from manim.utils.color import ManimColor, ParsableManimColor, color_gradient
53: (0)         from manim.utils.deprecation import deprecated
54: (0)         TEXT_MOB_SCALE_FACTOR = 0.05
55: (0)         DEFAULT_LINE_SPACING_SCALE = 0.3
56: (0)         TEXT2SVG_ADJUSTMENT_FACTOR = 4.8
57: (0)         __all__ = ["Text", "Paragraph", "MarkupText", "register_font"]
58: (0)         def remove_invisible_chars(mobject: SVGObject) -> SVGObject:
59: (4)             """Function to remove unwanted invisible characters from some mobjects.
60: (4)             Parameters
61: (4)             -----
62: (4)             mobject
63: (8)                 Any SVGObject from which we want to remove unwanted invisible
characters.
64: (4)             Returns
65: (4)             -----
66: (4)             :class:`~.SVGObject`
67: (8)                 The SVGObject without unwanted invisible characters.
68: (4)             """
69: (4)             iscode = False
70: (4)             if mobject.__class__.__name__ == "Text":
71: (8)                 mobject = mobject[:]
72: (4)             elif mobject.__class__.__name__ == "Code":
73: (8)                 iscode = True
74: (8)                 code = mobject
75: (8)                 mobject = mobject.code
76: (4)                 mobject_without_dots = VGroup()
77: (4)                 if mobject[0].__class__ == VGroup:
78: (8)                     for i in range(len(mobject)):
79: (12)                         mobject_without_dots.add(VGroup())
80: (12)                         mobject_without_dots[i].add(*(k for k in mobject[i] if k.__class__
!= Dot))
81: (4)
82: (8)                 else:
83: (4)                     mobject_without_dots.add(*(k for k in mobject if k.__class__ != Dot))
84: (4)             if iscode:
85: (8)                 code.code = mobject_without_dots
86: (8)                 return code
87: (0)             return mobject_without_dots
88: (0)         class Paragraph(VGroup):
89: (4)             """Display a paragraph of text.
90: (4)             For a given :class:`.Paragraph` ``par``, the attribute ``par.chars`` is a
91: (4)             :class:`.VGroup` containing all the lines. In this context, every line is
92: (4)             constructed as a :class:`.VGroup` of characters contained in the line.
93: (4)             Parameters
94: (4)             -----
95: (8)             line_spacing
96: (8)                 Represents the spacing between lines. Defaults to -1, which means
auto.
97: (4)             alignment
97: (8)                 Defines the alignment of paragraph. Defaults to None. Possible values

```

```

are "left", "right" or "center".
98: (4)          Examples
99: (4)          -----
100: (4)          Normal usage::
101: (8)          paragraph = Paragraph('this is a awesome', 'paragraph',
102: (30)          'With \nNewlines', '\tWith Tabs',
103: (30)          ' With Spaces', 'With Alignments',
104: (30)          'center', 'left', 'right')
105: (4)          Remove unwanted invisible characters::
106: (8)          self.play(Transform(remove_invisible_chars(paragraph.chars[0:2]),
107: (28)          remove_invisible_chars(paragraph.chars[3][0:3])))
108: (4)
109: (4)          def __init__(
110: (8)          self,
111: (8)          *text: Sequence[str],
112: (8)          line_spacing: float = -1,
113: (8)          alignment: str | None = None,
114: (8)          **kwargs,
115: (4)          ) -> None:
116: (8)          self.line_spacing = line_spacing
117: (8)          self.alignment = alignment
118: (8)          self.consider_spaces_as_chars = kwargs.get("disable_ligatures", False)
119: (8)          super().__init__()
120: (8)          lines_str = "\n".join(list(text))
121: (8)          self.lines_text = Text(lines_str, line_spacing=line_spacing, **kwargs)
122: (8)          lines_str_list = lines_str.split("\n")
123: (8)          self.chars = self._gen_chars(lines_str_list)
124: (8)          self.lines = [list(self.chars), [self.alignment] * len(self.chars)]
125: (8)          self.lines_initial_positions = [line.get_center() for line in
self.lines[0]]
126: (8)
127: (8)
128: (8)
129: (12)          if self.alignment:
130: (4)          self._set_all_lines_alignments(self.alignment)
131: (8)          def _gen_chars(self, lines_str_list: list) -> VGroup:
132: (8)          """Function to convert a list of plain strings to a VGroup of VGroups
of chars.
133: (8)
134: (8)
135: (12)          Parameters
136: (8)          -----
137: (8)          lines_str_list
138: (8)          List of plain text strings.
139: (12)          Returns
140: (8)          -----
141: (8)          :class:`~.VGroup`
142: (8)          The generated 2d-VGroup of chars.
143: (8)
144: (12)
145: (12)
146: (16)
147: (12)
148: (16)
149: (16)
150: (20)
151: (24)
152: (12)
153: (12)
154: (16)
155: (20)
156: (16)
157: (12)
158: (12)
159: (12)
160: (16)
161: (8)
162: (4)          def _set_all_lines_alignments(self, alignment: str) -> Paragraph:
163: (8)          """Function to set all line's alignment to a specific value.

```

```

164: (8)             Parameters
165: (8)             -----
166: (8)             alignment
167: (12)            Defines the alignment of paragraph. Possible values are "left",
168: (8)             "right", "center".
169: (8)
170: (12)
171: (8)
172: (4)             def _set_line_alignment(self, alignment: str, line_no: int) -> Paragraph:
173: (8)               """Function to set one line's alignment to a specific value.
174: (8)               Parameters
175: (8)               -----
176: (8)               alignment
177: (12)            Defines the alignment of paragraph. Possible values are "left",
178: (8)             "right", "center".
179: (12)
180: (8)
181: (8)
182: (8)
183: (4)             def _set_all_lines_to_initial_positions(self) -> Paragraph:
184: (8)               """Set all lines to their initial positions."""
185: (8)               self.lines[1] = [None] * len(self.lines[0])
186: (8)               for line_no in range(len(self.lines[0])):
187: (12)                 self[line_no].move_to(
188: (16)                   self.get_center() + self.lines_initial_positions[line_no],
189: (12)                 )
190: (8)
191: (4)             def _set_line_to_initial_position(self, line_no: int) -> Paragraph:
192: (8)               """Function to set one line to initial positions.
193: (8)               Parameters
194: (8)               -----
195: (8)               line_no
196: (12)            Defines the line number for which we want to set given alignment.
197: (8)
198: (8)             self.lines[1][line_no] = None
199: (8)             self[line_no].move_to(self.get_center() +
self.lines_initial_positions[line_no])
200: (8)
201: (4)             def _change_alignment_for_a_line(self, alignment: str, line_no: int) ->
None:
202: (8)               """Function to change one line's alignment to a specific value.
203: (8)               Parameters
204: (8)               -----
205: (8)               alignment
206: (12)            Defines the alignment of paragraph. Possible values are "left",
207: (8)             "right", "center".
208: (12)
209: (8)
210: (8)             self.lines[1][line_no] = alignment
211: (8)             if self.lines[1][line_no] == "center":
212: (12)               self[line_no].move_to(
213: (16)                 np.array([self.get_center()[0], self[line_no].get_center()[1],
0]),
214: (12)               )
215: (8)               elif self.lines[1][line_no] == "right":
216: (12)                 self[line_no].move_to(
217: (16)                   np.array(
218: (20)                     [
219: (24)                       self.get_right()[0] - self[line_no].width / 2,
220: (24)                       self[line_no].get_center()[1],
221: (24)                       0,
222: (20)                     ],
223: (16)                   ),
224: (12)               )
225: (8)               elif self.lines[1][line_no] == "left":
226: (12)                 self[line_no].move_to(

```

```

227: (16)                     np.array(
228: (20)                         [
229: (24)                             self.get_left()[0] + self[line_no].width / 2,
230: (24)                             self[line_no].get_center()[1],
231: (24)                             0,
232: (20)                         ],
233: (16)
234: (12)                     ),
235: (0)             class Text(SVGMobject):
236: (4)                 r"""Display (non-LaTeX) text rendered using `Pango
<https://pango.gnome.org/>`_.
237: (4)                 Text objects behave like a :class:`.VGroup`-like iterable of all
characters
238: (4)                     in the given text. In particular, slicing is possible.
239: (4)             Parameters
240: (4)                 -----
241: (4)                 text
242: (8)                     The text that needs to be created as a mobject.
243: (4)                 font
244: (8)                     The font family to be used to render the text. This is either a system
font or
245: (8)                     one loaded with `register_font()`. Note that font family names may be
different
246: (8)
247: (4)
248: (8)                     If True (default), Manim will issue a warning if the font does not
exist in the
249: (8)                     (case-sensitive) list of fonts returned from
`manimpango.list_fonts()`.

250: (4)             Returns
251: (4)                 -----
252: (4)                 :class:`Text`
253: (8)                     The mobject-like :class:`.VGroup`.
254: (4)             Examples
255: (4)                 -----
256: (4)                 .. manim:: Example1Text
257: (8)                     :save_last_frame:
258: (8)                     class Example1Text(Scene):
259: (12)                         def construct(self):
260: (16)                             text = Text('Hello world').scale(3)
261: (16)                             self.add(text)
262: (4)                 .. manim:: TextColorExample
263: (8)                     :save_last_frame:
264: (8)                     class TextColorExample(Scene):
265: (12)                         def construct(self):
266: (16)                             text1 = Text('Hello world', color=BLUE).scale(3)
267: (16)                             text2 = Text('Hello world', gradient=(BLUE,
GREEN)).scale(3).next_to(text1, DOWN)
268: (16)                             self.add(text1, text2)
269: (4)                 .. manim:: TextItalicAndBoldExample
270: (8)                     :save_last_frame:
271: (8)                     class TextItalicAndBoldExample(Scene):
272: (12)                         def construct(self):
273: (16)                             text1 = Text("Hello world", slant=ITALIC)
274: (16)                             text2 = Text("Hello world", t2s={'world':ITALIC})
275: (16)                             text3 = Text("Hello world", weight=BOLD)
276: (16)                             text4 = Text("Hello world", t2w={'world':BOLD})
277: (16)                             text5 = Text("Hello world", t2c={'o':YELLOW},
disable_ligatures=True)
278: (16)
279: (20)                     text6 = Text(
280: (20)                         "Visit us at docs.manim.community",
281: (20)                         t2c={"docs.manim.community": YELLOW},
282: (15)                         disable_ligatures=True,
283: (16)                     )
284: (16)                     text6.scale(1.3).shift(DOWN)
285: (16)                     self.add(text1, text2, text3, text4, text5 , text6,
Group(*self.mobjects).arrange(DOWN,
buff=.8).set(height=config.frame_height-LARGE_BUFF)
286: (4)             .. manim:: TextMoreCustomization

```

```

287: (12)           :save_last_frame:
288: (12)           class TextMoreCustomization(Scene):
289: (16)             def construct(self):
290: (20)               text1 = Text(
291: (24)                 'Google',
292: (24)                 t2c={'[:1]': '#3174f0', '[1:2]': '#e53125',
293: (29)                   '[2:3]': '#ffbb003', '[3:4]': '#3174f0',
294: (29)                   '[4:5]': '#269a43', '[5:]: '#e53125'},
295: (20)               font_size=58).scale(3)
296: (4)               self.add(text1)
297: (4)             As :class:`Text` uses Pango to render text, rendering non-English
298: (4)             characters is easily possible:
299: (8)             .. manim:: MultipleFonts
300: (8)               :save_last_frame:
301: (12)             class MultipleFonts(Scene):
302: (16)               def construct(self):
303: (16)                 morning = Text("வணக்கம்", font="sans-serif")
304: (20)                 japanese = Text(
305: (16)                   "日本へようこそ", t2c={"日本": BLUE}
306: (16)                 ) # works same as ``Text``.
307: (16)                 mess = Text("Multi-Language", weight=BOLD)
308: (16)                 russ = Text("Здравствуйте मस नम म ", font="sans-serif")
309: (16)                 hin = Text("नमस्ते", font="sans-serif")
310: (20)                 arb = Text(
311: (16)                   "شرفت بمقابلتك \n صباح الخير", font="sans-serif"
312: (16)                 ) # don't mix RTL and LTR languages nothing shows up then ;-
313: (16)                 chinese = Text("臂猿「黛比」帶著孩子", font="sans-serif")
314: (16)                 self.add(morning, japanese, mess, russ, hin, arb, chinese)
315: (20)                 for i,mobj in enumerate(self.mobjects):
316: (4)                   mobj.shift(DOWN*(i-3))
317: (8)             .. manim:: PangoRender
318: (8)               :quality: low
319: (12)             class PangoRender(Scene):
320: (16)               def construct(self):
321: (16)                 morning = Text("வணக்கம்", font="sans-serif")
322: (16)                 self.play(Write(morning))
323: (4)                 self.wait(2)
324: (4)             Tests
325: (4)             -----
326: (8)             Check that the creation of :class:`~.Text` works::
327: (8)               >>> Text('The horse does not eat cucumber salad.')
328: (8)               Text('The horse does not eat cucumber salad.')
329: (4)               """
330: (4)               @staticmethod
331: (4)               @functools.lru_cache(maxsize=None)
332: (8)               def font_list() -> list[str]:
333: (4)                 return manimpango.list_fonts()
334: (8)               def __init__(
335: (8)                 self,
336: (8)                 text: str,
337: (8)                 fill_opacity: float = 1.0,
338: (8)                 stroke_width: float = 0,
339: (8)                 color: ParsableManimColor | None = None,
340: (8)                 font_size: float = DEFAULT_FONT_SIZE,
341: (8)                 line_spacing: float = -1,
342: (8)                 font: str = "",
343: (8)                 slant: str = NORMAL,
344: (8)                 weight: str = NORMAL,
345: (8)                 t2c: dict[str, str] = None,
346: (8)                 t2f: dict[str, str] = None,
347: (8)                 t2g: dict[str, tuple] = None,
348: (8)                 t2s: dict[str, str] = None,
349: (8)                 t2w: dict[str, str] = None,
350: (8)                 gradient: tuple = None,
351: (8)                 tab_width: int = 4,
352: (8)                 warn_missing_font: bool = True,
353: (8)                 height: float = None,
353: (8)                 width: float = None,

```

```

354: (8)             should_center: bool = True,
355: (8)             disable_ligatures: bool = False,
356: (8)             use_svg_cache: bool = False,
357: (8)             **kwargs,
358: (4)         ) -> None:
359: (8)             self.line_spacing = line_spacing
360: (8)             if font and warn_missing_font:
361: (12)                 fonts_list = Text.font_list()
362: (12)                 if font.lower() == "sans-serif":
363: (16)                     font = "sans"
364: (12)                 if font not in fonts_list:
365: (16)                     if font.capitalize() in fonts_list:
366: (20)                         font = font.capitalize()
367: (16)                     elif font.lower() in fonts_list:
368: (20)                         font = font.lower()
369: (16)                     elif font.title() in fonts_list:
370: (20)                         font = font.title()
371: (16)                 else:
372: (20)                     logger.warning(f"Font {font} not in {fonts_list}.")
373: (8)             self.font = font
374: (8)             self._font_size = float(font_size)
375: (8)             self.slant = slant
376: (8)             self.weight = weight
377: (8)             self.gradient = gradient
378: (8)             self.tab_width = tab_width
379: (8)             if t2c is None:
380: (12)                 t2c = {}
381: (8)             if t2f is None:
382: (12)                 t2f = {}
383: (8)             if t2g is None:
384: (12)                 t2g = {}
385: (8)             if t2s is None:
386: (12)                 t2s = {}
387: (8)             if t2w is None:
388: (12)                 t2w = {}
389: (8)             t2c = kwargs.pop("text2color", t2c)
390: (8)             t2f = kwargs.pop("text2font", t2f)
391: (8)             t2g = kwargs.pop("text2gradient", t2g)
392: (8)             t2s = kwargs.pop("text2slant", t2s)
393: (8)             t2w = kwargs.pop("text2weight", t2w)
394: (8)             self.t2c = {k: ManimColor(v).to_hex() for k, v in t2c.items()}
395: (8)             self.t2f = t2f
396: (8)             self.t2g = t2g
397: (8)             self.t2s = t2s
398: (8)             self.t2w = t2w
399: (8)             self.original_text = text
400: (8)             self.disable_ligatures = disable_ligatures
401: (8)             text_without_tabs = text
402: (8)             if text.find("\t") != -1:
403: (12)                 text_without_tabs = text.replace("\t", " " * self.tab_width)
404: (8)             self.text = text_without_tabs
405: (8)             if self.line_spacing == -1:
406: (12)                 self.line_spacing = (
407: (16)                     self._font_size + self._font_size * DEFAULT_LINE_SPACING_SCALE
408: (12)
409: (8)
410: (12)             self.line_spacing = self._font_size + self._font_size *
self.line_spacing
411: (8)             color: ManimColor = ManimColor(color) if color else VMOBJECT().color
412: (8)             file_name = self._text2svg(color.to_hex())
413: (8)             PangoUtils.remove_last_M(file_name)
414: (8)             super().__init__(
415: (12)                 file_name,
416: (12)                 fill_opacity=fill_opacity,
417: (12)                 stroke_width=stroke_width,
418: (12)                 height=height,
419: (12)                 width=width,
420: (12)                 should_center=should_center,
421: (12)                 use_svg_cache=use_svg_cache,

```

```

422: (12)                         **kwargs,
423: (8)                          )
424: (8)                          self.text = text
425: (8)                          if self.disable_ligatures:
426: (12)                            self.submobjects = [*self._gen_chars()]
427: (8)                          self.chars = self.get_group_class()(*self.submobjects)
428: (8)                          self.text = text_without_tabs.replace(" ", "").replace("\n", "")
429: (8)                          nppc = self.n_points_per_curve
430: (8)                          for each in self:
431: (12)                            if len(each.points) == 0:
432: (16)                              continue
433: (12)                            points = each.points
434: (12)                            curve_start = points[0]
435: (12)                            assert len(curve_start) == self.dim, curve_start
436: (12)                            closed_curve_points = []
437: (12)                            if nppc == 3: # RendererType.OPENGL
438: (16)                              def add_line_to(end):
439: (20)                                nonlocal closed_curve_points
440: (20)                                start = closed_curve_points[-1]
441: (20)                                closed_curve_points += [
442: (24)                                  start,
443: (24)                                  (start + end) / 2,
444: (24)                                  end,
445: (20)                                ]
446: (12)                            else: # RendererType.CAIRO
447: (16)                              def add_line_to(end):
448: (20)                                nonlocal closed_curve_points
449: (20)                                start = closed_curve_points[-1]
450: (20)                                closed_curve_points += [
451: (24)                                  start,
452: (24)                                  (start + start + end) / 3,
453: (24)                                  (start + end + end) / 3,
454: (24)                                  end,
455: (20)                                ]
456: (12)                            for index, point in enumerate(points):
457: (16)                              closed_curve_points.append(point)
458: (16)                              if (
459: (20)                                index != len(points) - 1
460: (20)                                and (index + 1) % nppc == 0
461: (20)                                and any(point != points[index + 1]))
462: (16)                            ):
463: (20)                              add_line_to(curve_start)
464: (20)                              curve_start = points[index + 1]
465: (12)                              add_line_to(curve_start)
466: (12)                              each.points = np.array(closed_curve_points, ndmin=2)
467: (8)                              if height is None and width is None:
468: (12)                                self.scale(TEXT_MOB_SCALE_FACTOR)
469: (8)                                self.initial_height = self.height
470: (4)                                def __repr__(self):
471: (8)                                  return f"Text({repr(self.original_text)})"
472: (4)                                @property
473: (4)                                def font_size(self):
474: (8)                                  return (
475: (12)                                    self.height
476: (12)                                    / self.initial_height
477: (12)                                    / TEXT_MOB_SCALE_FACTOR
478: (12)                                    * 2.4
479: (12)                                    * self._font_size
480: (12)                                    / DEFAULT_FONT_SIZE
481: (8)                                  )
482: (4)                                @font_size.setter
483: (4)                                def font_size(self, font_val):
484: (8)                                  if font_val <= 0:
485: (12)                                    raise ValueError("font_size must be greater than 0.")
486: (8)                                  else:
487: (12)                                    self.scale(font_val / self.font_size)
488: (4)                                def _gen_chars(self):
489: (8)                                  chars = self.get_group_class }()
490: (8)                                  submobjects_char_index = 0

```

```

491: (8)             for char_index in range(len(self.text)):
492: (12)             if self.text[char_index].isspace():
493: (16)                 space = Dot(radius=0, fill_opacity=0, stroke_opacity=0)
494: (16)                 if char_index == 0:
495: (20)
space.move_to(self.submobjects[submobjects_char_index].get_center())
496: (16)             else:
497: (20)                 space.move_to(
498: (24)                     self.submobjects[submobjects_char_index] -
1].get_center(),
499: (20)                 )
500: (16)             chars.add(space)
501: (12)         else:
502: (16)             chars.add(self.submobjects[submobjects_char_index])
503: (16)             submobjects_char_index += 1
504: (8)         return chars
505: (4)     def _find_indexes(self, word: str, text: str):
506: (8)         """Finds the indexes of ``text`` in ``word``."""
507: (8)         temp = re.match(r"\[([0-9\-\-]{0,}):([0-9\-\-]{0,})\]", word)
508: (8)         if temp:
509: (12)             start = int(temp.group(1)) if temp.group(1) != "" else 0
510: (12)             end = int(temp.group(2)) if temp.group(2) != "" else len(text)
511: (12)             start = len(text) + start if start < 0 else start
512: (12)             end = len(text) + end if end < 0 else end
513: (12)             return [(start, end)]
514: (8)     indexes = []
515: (8)     index = text.find(word)
516: (8)     while index != -1:
517: (12)         indexes.append((index, index + len(word)))
518: (12)         index = text.find(word, index + len(word))
519: (8)     return indexes
520: (4) @deprecated(
521: (8)     since="v0.14.0",
522: (8)     until="v0.15.0",
523: (8)     message="This was internal function, you shouldn't be using it
anyway.",
524: (4) )
525: (4)     def _set_color_by_t2c(self, t2c=None):
526: (8)         """Sets color for specified strings."""
527: (8)         t2c = t2c if t2c else self.t2c
528: (8)         for word, color in list(t2c.items()):
529: (12)             for start, end in self._find_indexes(word, self.text):
530: (16)                 self.chars[start:end].set_color(color)
531: (4) @deprecated(
532: (8)     since="v0.14.0",
533: (8)     until="v0.15.0",
534: (8)     message="This was internal function, you shouldn't be using it
anyway.",
535: (4) )
536: (4)     def _set_color_by_t2g(self, t2g=None):
537: (8)         """Sets gradient colors for specified
538: (8)             strings. Behaves similarly to ``set_color_by_t2c``."""
539: (8)         t2g = t2g if t2g else self.t2g
540: (8)         for word, gradient in list(t2g.items()):
541: (12)             for start, end in self._find_indexes(word, self.text):
542: (16)                 self.chars[start:end].set_color_by_gradient(*gradient)
543: (4)     def _text2hash(self, color: ManimColor):
544: (8)         """Generates ``sha256`` hash for file name."""
545: (8)         settings = (
546: (12)             "PANGO" + self.font + self.slant + self.weight + str(color)
547: (8)         ) # to differentiate Text and CairoText
548: (8)         settings += str(self.t2f) + str(self.t2s) + str(self.t2w) +
str(self.t2c)
549: (8)         settings += str(self.line_spacing) + str(self._font_size)
550: (8)         settings += str(self.disable_ligatures)
551: (8)         id_str = self.text + settings
552: (8)         hasher = hashlib.sha256()
553: (8)         hasher.update(id_str.encode())
554: (8)         return hasher.hexdigest()[:16]

```

```

555: (4)             def _merge_settings(
556: (8)               self,
557: (8)               left_setting: TextSetting,
558: (8)               right_setting: TextSetting,
559: (8)               default_args: dict[str, Iterable[str]],
560: (4)           ) -> TextSetting:
561: (8)               contained = right_setting.end < left_setting.end
562: (8)               new_setting = copy.copy(left_setting) if contained else
copy.copy(right_setting)
563: (8)               new_setting.start = right_setting.end if contained else
left_setting.end
564: (8)               left_setting.end = right_setting.start
565: (8)               if not contained:
566: (12)                   right_setting.end = new_setting.start
567: (8)               for arg in default_args:
568: (12)                   left = getattr(left_setting, arg)
569: (12)                   right = getattr(right_setting, arg)
570: (12)                   default = default_args[arg]
571: (12)                   if left != default and getattr(right_setting, arg) != default:
572: (16)                       raise ValueError(
573: (20)                           f"Ambiguous style for text
'{self.text[right_setting.start:right_setting.end]}':"
574: (20)                               + f"'{arg}' cannot be both '{left}' and '{right}'."
575: (16)                           )
576: (12)                   setattr(right_setting, arg, left if left != default else right)
577: (8)               return new_setting
578: (4)             def _get_settings_from_t2xs(
579: (8)               self,
580: (8)               t2xs: Sequence[tuple[dict[str, str], str]],
581: (8)               default_args: dict[str, Iterable[str]],
582: (4)           ) -> Sequence[TextSetting]:
583: (8)               settings = []
584: (8)               t2xwords = set(chain(*([*t2x.keys()] for t2x, _ in t2xs)))
585: (8)               for word in t2xwords:
586: (12)                   setting_args = {
587: (16)                       arg: str(t2x[word]) if word in t2x else default_args[arg]
588: (16)                       for t2x, arg in t2xs
589: (12)                   }
590: (12)                   for start, end in self._find_indexes(word, self.text):
591: (16)                       settings.append(TextSetting(start, end, **setting_args))
592: (8)               return settings
593: (4)             def _get_settings_from_gradient(
594: (8)               self, default_args: dict[str, Iterable[str]]
595: (4)           ) -> Sequence[TextSetting]:
596: (8)               settings = []
597: (8)               args = copy.copy(default_args)
598: (8)               if self.gradient:
599: (12)                   colors = color_gradient(self.gradient, len(self.text))
600: (12)                   for i in range(len(self.text)):
601: (16)                       args["color"] = colors[i].to_hex()
602: (16)                       settings.append(TextSetting(i, i + 1, **args))
603: (8)               for word, gradient in self.t2g.items():
604: (12)                   if isinstance(gradient, str) or len(gradient) == 1:
605: (16)                       color = gradient if isinstance(gradient, str) else gradient[0]
606: (16)                       gradient = [ManimColor(color)]
607: (12)                   colors = (
608: (16)                       color_gradient(gradient, len(word))
609: (16)                       if len(gradient) != 1
610: (16)                       else len(word) * gradient
611: (12)                   )
612: (12)                   for start, end in self._find_indexes(word, self.text):
613: (16)                       for i in range(start, end):
614: (20)                           args["color"] = colors[i - start].to_hex()
615: (20)                           settings.append(TextSetting(i, i + 1, **args))
616: (8)               return settings
617: (4)             def _text2settings(self, color: str):
618: (8)                 """Converts the texts and styles to a setting for parsing."""
619: (8)                 t2xs = [
620: (12)                     (self.t2f, "font"),

```

```

621: (12)                               (self.t2s, "slant"),
622: (12)                               (self.t2w, "weight"),
623: (12)                               (self.t2c, "color"),
624: (8) ]                               default_args = {
625: (8)                                 arg: getattr(self, arg) if arg != "color" else color for _, arg in
626: (12)                               }
627: (8) }                               settings = self._get_settings_from_t2xs(t2xs, default_args)
628: (8) settings.extend(self._get_settings_from_gradient(default_args))
629: (8) settings.sort(key=lambda setting: setting.start)
630: (8) for index, setting in enumerate(settings):
631: (8)   if index + 1 == len(settings):
632: (12)     break
633: (16)   next_setting = settings[index + 1]
634: (12)   if setting.end > next_setting.start:
635: (12)     new_setting = self._merge_settings(setting, next_setting,
636: (16)       new_index = index + 1
637: (16)       while (
638: (20)         new_index < len(settings)
639: (20)         and settings[new_index].start < new_setting.start
640: (16)       ):
641: (20)         new_index += 1
642: (16)         settings.insert(new_index, new_setting)
643: (16)       temp_settings = settings.copy()
644: (8)       start = 0
645: (8)       for setting in settings:
646: (8)         if setting.start != start:
647: (12)           temp_settings.append(TextSetting(start, setting.start,
648: (16)             start = setting.end
649: (12)           if start != len(self.text):
650: (8)             temp_settings.append(TextSetting(start, len(self.text),
651: (12)               settings = sorted(temp_settings, key=lambda setting: setting.start)
652: (8)               line_num = 0
653: (8)               if re.search(r"\n", self.text):
654: (8)                 for start, end in self._find_indexes("\n", self.text):
655: (12)                   for setting in settings:
656: (16)                     if setting.line_num == -1:
657: (20)                       setting.line_num = line_num
658: (24)                     if start < setting.end:
659: (20)                       line_num += 1
660: (24)                       new_setting = copy.copy(setting)
661: (24)                       setting.end = end
662: (24)                       new_setting.start = end
663: (24)                       new_setting.line_num = line_num
664: (24)                       settings.append(new_setting)
665: (24)                       settings.sort(key=lambda setting: setting.start)
666: (24)                     break
667: (24)               for setting in settings:
668: (8)                 if setting.line_num == -1:
669: (12)                   setting.line_num = line_num
670: (16)             return settings
671: (8)         def _text2svg(self, color: ManimColor):
672: (4)           """Convert the text to SVG using Pango."""
673: (8)           size = self._font_size
674: (8)           line_spacing = self.line_spacing
675: (8)           size /= TEXT2SVG_ADJUSTMENT_FACTOR
676: (8)           line_spacing /= TEXT2SVG_ADJUSTMENT_FACTOR
677: (8)           dir_name = config.get_dir("text_dir")
678: (8)           if not dir_name.is_dir():
679: (8)             dir_name.mkdir(parents=True)
680: (12)           hash_name = self._text2hash(color)
681: (8)           file_name = dir_name / (hash_name + ".svg")
682: (8)           if file_name.exists():
683: (8)             svg_file = str(file_name.resolve())
684: (12)           else:
685: (8)

```

```

686: (12)                     settings = self._text2settings(color)
687: (12)                     width = config["pixel_width"]
688: (12)                     height = config["pixel_height"]
689: (12)                     svg_file = manimpango.text2svg(
690: (16)                         settings,
691: (16)                         size,
692: (16)                         line_spacing,
693: (16)                         self.disable_ligatures,
694: (16)                         str(file_name.resolve())),
695: (16)                         START_X,
696: (16)                         START_Y,
697: (16)                         width,
698: (16)                         height,
699: (16)                         self.text,
700: (12)                     )
701: (8)                     return svg_file
702: (4)             def init_colors(self, propagate_colors=True):
703: (8)                 if config.renderer == RendererType.OPENGL:
704: (12)                     super().init_colors()
705: (8)                 elif config.renderer == RendererType.CAIRO:
706: (12)                     super().init_colors(propagate_colors=propagate_colors)
707: (0)             class MarkupText(SVGMobject):
708: (4)                 r"""Display (non-LaTeX) text rendered using `Pango
<https://pango.gnome.org/>`_.
709: (4)                     Text objects behave like a :class:`.VGroup`-like iterable of all
characters
710: (4)                     in the given text. In particular, slicing is possible.
711: (4)                     **What is PangоМаркап?**
712: (4)                     PangоМаркап is a small markup language like html and it helps you avoid
using
713: (4)                     "range of characters" while coloring or styling a piece a Text. You can
use
714: (4)                     this language with :class:`~.MarkupText`.
715: (4)                     A simple example of a marked-up string might be::
716: (8)                         <span foreground="blue" size="x-large">Blue text</span> is
<i>cool</i>!"
717: (4)                     and it can be used with :class:`~.MarkupText` as
718: (4)                         .. manim:: MarkupExample
719: (8)                             :save_last_frame:
720: (8)                             class MarkupExample(Scene):
721: (12)                             def construct(self):
722: (16)                                 text = MarkupText('<span foreground="blue" size="x-large">Blue
text</span> is <i>cool</i>!')
723: (16)                                 self.add(text)
724: (4)                     A more elaborate example would be:
725: (4)                         .. manim:: MarkupElaborateExample
726: (8)                             :save_last_frame:
727: (8)                             class MarkupElaborateExample(Scene):
728: (12)                             def construct(self):
729: (16)                                 text = MarkupText(
730: (20)                                     '<span foreground="purple">|</span><span
foreground="red"></span>''
731: (20)                                     '<span foreground="blue">|</span>&lt;span
foreground="red"></span>,''
732: (20)                                     '<span foreground="red"></span>|<span
foreground="red"></span>|'
733: (20)                                     '<span foreground="green"></span><span
foreground="red"></span>|'
734: (20)                                     '|<span foreground="blue"></span>'
735: (16)
736: (16)                         )
737: (16)                         self.add(text)
738: (4)                     PangоМаркап can also contain XML features such as numeric character
entities such as ``&#169;`` for ® can be used too.
739: (4)                     The most general markup tag is ``<span>``, then there are some
convenience tags.
740: (4)                     Here is a list of supported tags:
741: (4)                         - ``<b>bold</b>``, ``<i>italic</i>`` and ``<b><i>bold+italic</i></b>``
742: (4)                         - ``<ul>underline</ul>`` and ``<s>strike through</s>``
743: (4)                         - ``<tt>typewriter font</tt>``

```

```

745: (4)           - ``<big>bigger font</big>`` and ``<small>smaller font</small>``
746: (4)           - ``<sup>superscript</sup>`` and ``<sub>subscript</sub>``
747: (4)           - ``<span underline="double" underline_color="green">double
underline</span>``
748: (4)           - ``<span underline="error">error underline</span>``
749: (4)           - ``<span overline="single" overline_color="green">overline</span>``
750: (4)           - ``<span strikethrough="true"
strikethrough_color="red">strikethrough</span>``
751: (4)           - ``<span font_family="sans">temporary change of font</span>``
752: (4)           - ``<span foreground="red">temporary change of color</span>``
753: (4)           - ``<span fgcolor="red">temporary change of color</span>``
754: (4)           - ``<gradient from="YELLOW" to="RED">temporary gradient</gradient>``
755: (4)           For ``<span>`` markup, colors can be specified either as
756: (4)           hex triples like ``#aabbcc`` or as named CSS colors like
757: (4)           ``AliceBlue``.
758: (4)           The ``<gradient>`` tag is handled by Manim rather than
759: (4)           Pango, and supports hex triplets or Manim constants like
760: (4)           ``RED`` or ``RED_A``.
761: (4)           If you want to use Manim constants like ``RED_A`` together
762: (4)           with ``<span>``, you will need to use Python's f-String
763: (4)           syntax as follows::
764: (8)           MarkupText(f'<span foreground="{RED_A}">here you go</span>')
765: (4)           If your text contains ligatures, the :class:`MarkupText` class may
766: (4)           incorrectly determine the first and last letter when creating the
767: (4)           gradient. This is due to the fact that ``fl`` are two separate characters,
768: (4)           but might be set as one single glyph - a ligature. If your language
769: (4)           does not depend on ligatures, consider setting ``disable_ligatures``
770: (4)           to ``True``. If you must use ligatures, the ``gradient`` tag supports an
optional
771: (4)           attribute ``offset`` which can be used to compensate for that error.
772: (4)           For example:
773: (4)           - ``<gradient from="RED" to="YELLOW" offset="1">example</gradient>`` to
*start* the gradient one letter earlier
774: (4)           - ``<gradient from="RED" to="YELLOW" offset=",1">example</gradient>`` to
*end* the gradient one letter earlier
775: (4)           - ``<gradient from="RED" to="YELLOW" offset="2,1">example</gradient>`` to
*start* the gradient two letters earlier and *end* it one letter earlier
776: (4)           Specifying a second offset may be necessary if the text to be colored does
777: (4)           itself contain ligatures. The same can happen when using HTML entities for
778: (4)           special chars.
779: (4)           When using ``underline``, ``overline`` or ``strikethrough`` together with
780: (4)           ``<gradient>`` tags, you will also need to use the offset, because
781: (4)           underlines are additional paths in the final :class:`SVGMOobject`.
782: (4)           Check out the following example.
783: (4)           Escaping of special characters: ``>`` **should** be written as ``&gt;``
784: (4)           whereas ``<`` and ``&&`` *must* be written as ``&lt;`` and
785: (4)           ``&amp;``.
786: (4)           You can find more information about Pango markup formatting at the
787: (4)           corresponding documentation page:
788: (4)           `Pango Markup <https://docs.gtk.org/Pango/pango\_markup.html>`_.
789: (4)           Please be aware that not all features are supported by this class and that
790: (4)           the ``<gradient>`` tag mentioned above is not supported by Pango.
791: (4)           Parameters
792: (4)           -----
793: (4)           text
794: (8)           The text that needs to be created as mobject.
795: (4)           fill_opacity
796: (8)           The fill opacity, with 1 meaning opaque and 0 meaning transparent.
797: (4)           stroke_width
798: (8)           Stroke width.
799: (4)           font_size
800: (8)           Font size.
801: (4)           line_spacing
802: (8)           Line spacing.
803: (4)           font
804: (8)           Global font setting for the entire text. Local overrides are possible.
805: (4)           slant
806: (8)           Global slant setting, e.g. `NORMAL` or `ITALIC`. Local overrides are
possible.

```

```

807: (4)           weight
808: (8)             Global weight setting, e.g. `NORMAL` or `BOLD`. Local overrides are
possible.
809: (4)           gradient
810: (8)             Global gradient setting. Local overrides are possible.
811: (4)           warn_missing_font
812: (8)             If True (default), Manim will issue a warning if the font does not
exist in the
813: (8)               (case-sensitive) list of fonts returned from
`manimpango.list_fonts()`.

814: (4)           Returns
815: (4)           -----
816: (4)           :class:`MarkupText`
817: (8)             The text displayed in form of a :class:`.VGroup`-like mobject.
818: (4)           Examples
819: (4)           -----
820: (4)             .. manim:: BasicMarkupExample
821: (8)               :save_last_frame:
822: (8)                 class BasicMarkupExample(Scene):
823: (12)                   def construct(self):
824: (16)                     text1 = MarkupText("<b>foo</b> <i>bar</i> <b><i>foobar</i></b>")
825: (16)                     text2 = MarkupText("<s>foo</s> <u>bar</u> <big>big</big>")
826: (16)                     text3 = MarkupText("H<sub>2</sub>O and H<sub>3</sub>O<sup>+</sup>")
827: (16)
828: (16)
829: (20)               underline="error">bar</span>'           )
830: (16)               group = VGroup(text1, text2, text3, text4,
831: (16)                 self.add(group)
832: (16)             .. manim:: ColorExample
833: (4)               :save_last_frame:
834: (8)                 class ColorExample(Scene):
835: (8)                   def construct(self):
836: (12)                     text1 = MarkupText(
837: (16)                       f'all in red <span fgcolor="{YELLOW}">except this</span>',
838: (20)                     )
839: (16)                     text2 = MarkupText("nice gradient", gradient=(BLUE, GREEN))
840: (16)                     text3 = MarkupText(
841: (16)                         'nice <gradient from="RED"
842: (20)                           to="YELLOW">intermediate</gradient> gradient',
843: (20)                           gradient=(BLUE, GREEN),
844: (16)                         )
845: (16)                     text4 = MarkupText(
846: (20)                         'fl ligature <gradient from="RED" to="YELLOW">causing
trouble</gradient> here'
847: (16)                     )
848: (16)                     text5 = MarkupText(
849: (20)                         'fl ligature <gradient from="RED" to="YELLOW"
offset="1">defeated</gradient> with offset'
850: (16)                         )
851: (16)                     text6 = MarkupText(
852: (20)                         'fl ligature <gradient from="RED" to="YELLOW"
offset="1">floating</gradient> inside'
853: (16)                         )
854: (16)                     text7 = MarkupText(
855: (20)                         'fl ligature <gradient from="RED" to="YELLOW"
offset="1,1">floating</gradient> inside'
856: (16)                         )
857: (16)                     group = VGroup(text1, text2, text3, text4, text5, text6,
text7).arrange(DOWN)
858: (16)                     self.add(group)
859: (4)             .. manim:: UnderlineExample
860: (8)               :save_last_frame:

```

```

861: (8)             class UnderlineExample(Scene):
862: (12)             def construct(self):
863: (16)                 text1 = MarkupText(
864: (20)                     '<span underline="double"
underline_color="green">bla</span>'
865: (16)                 )
866: (16)                 text2 = MarkupText(
867: (20)                     '<span underline="single"
underline_color="green">xxx</span><gradient from="#ffff00" to="RED">aabb</gradient>y'
868: (16)                     )
869: (16)                     text3 = MarkupText(
870: (20)                         '<span underline="single"
underline_color="green">xxx</span><gradient from="#ffff00" to="RED" offset="-1">aabb</gradient>y'
871: (16)                         )
872: (16)                         text4 = MarkupText(
873: (20)                             '<span underline="double"
underline_color="green">xxx</span><gradient from="#ffff00" to="RED">aabb</gradient>y'
874: (16)                             )
875: (16)                             text5 = MarkupText(
876: (20)                                 '<span underline="double"
underline_color="green">xxx</span><gradient from="#ffff00" to="RED" offset="-2">aabb</gradient>y'
877: (16)                                 )
878: (16)                                 group = VGroup(text1, text2, text3, text4,
text5).arrange(DOWN)
879: (16)                                     self.add(group)
880: (4) .. manim:: FontExample
881: (8)     :save_last_frame:
882: (8)     class FontExample(Scene):
883: (12)         def construct(self):
884: (16)             text1 = MarkupText(
885: (20)                 'all in sans <span font_family="serif">except
this</span>', font="sans"
886: (16)             )
887: (16)             text2 = MarkupText(
888: (20)                 '<span font_family="serif">mixing</span> <span
font_family="sans">fonts</span> <span font_family="monospace">is ugly</span>'
889: (16)                 )
890: (16)                 text3 = MarkupText("special char > or &gt;")
891: (16)                 text4 = MarkupText("special char &lt; and &amp;")
892: (16)                 group = VGroup(text1, text2, text3, text4).arrange(DOWN)
893: (16)                 self.add(group)
894: (4) .. manim:: NewlineExample
895: (8)     :save_last_frame:
896: (8)     class NewlineExample(Scene):
897: (12)         def construct(self):
898: (16)             text = MarkupText('foooo<span
foreground="red">oo\nbaa</span>aar')
899: (16)             self.add(text)
900: (4) .. manim:: NoLigaturesExample
901: (8)     :save_last_frame:
902: (8)     class NoLigaturesExample(Scene):
903: (12)         def construct(self):
904: (16)             text1 = MarkupText('fl<gradient from="RED"
to="GREEN">oat</gradient>ing')
905: (16)             text2 = MarkupText('fl<gradient from="RED"
to="GREEN">oat</gradient>ing', disable_ligatures=True)
906: (16)                 group = VGroup(text1, text2).arrange(DOWN)
907: (16)                 self.add(group)
908: (4) As :class:`MarkupText` uses Pango to render text, rendering non-English
909: (4) characters is easily possible:
910: (4) .. manim:: MultiLanguage
911: (8)     :save_last_frame:
912: (8)     class MultiLanguage(Scene):
913: (12)         def construct(self):
914: (16)             morning = MarkupText("வணக்கம்", font="sans-serif")
915: (16)             japanese = MarkupText(
916: (20)                 '<span fgcolor="blue">日本</span>へようこそ'
917: (16)             ) # works as in ``Text``.
918: (16)             mess = MarkupText("Multi-Language", weight=BOLD)

```

```

919: (16)             russ = MarkupText("здравствуйте мэр нам м ", font="sans-serif")
920: (16)             hin = MarkupText("ନମ୍ରତେ", font="sans-serif")
921: (16)             chinese = MarkupText("臂猿「黛比」帶著孩子", font="sans-serif")
922: (16)             group = VGroup(morning, japanese, mess, russ, hin,
chinese).arrange(DOWN)
923: (16)                 self.add(group)
924: (4)             You can justify the text by passing :attr:`justify` parameter.
925: (4)             .. manim:: JustifyText
926: (8)                 class JustifyText(Scene):
927: (12)                     def construct(self):
928: (16)                         ipsum_text = (
929: (20)                             "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
930: (20)                             "Praesent feugiat metus sit amet iaculis pulvinar. Nulla
posuere "
931: (20)                             "quam a ex aliquam, eleifend consectetur tellus viverra.
Aliquam "
932: (20)                             "fermentum interdum justo, nec rutrum elit pretium ac. Nam
quis "
933: (20)                             "leo pulvinar, dignissim est at, venenatis nisi."
934: (16)
935: (16)                     justified_text = MarkupText(ipsum_text,
justify=True).scale(0.4)
936: (16)                     not_justified_text = MarkupText(ipsum_text,
justify=False).scale(0.4)
937: (16)
938: (16)
939: (16)
940: (16)                     just_title = Title("Justified")
941: (20)                     njust_title = Title("Not Justified")
942: (20)                     self.add(njust_title, not_justified_text)
943: (20)                     self.play(
944: (20)                         FadeOut(not_justified_text),
945: (16)                         FadeIn(justified_text),
946: (16)                         FadeOut(njust_title),
947: (4)                         FadeIn(just_title),
948: (4)                     )
949: (4)             self.wait(1)
Tests
-----
949: (4)             Check that the creation of :class:`~.MarkupText` works::
950: (8)                 >>> MarkupText('The horse does not eat cucumber salad.')
951: (8)                 MarkupText('The horse does not eat cucumber salad.')
"""
953: (4)             @staticmethod
954: (4)             @functools.lru_cache(maxsize=None)
955: (4)             def font_list() -> list[str]:
956: (8)                 return manimpango.list_fonts()
957: (4)             def __init__(
958: (8)                 self,
959: (8)                 text: str,
960: (8)                 fill_opacity: float = 1,
961: (8)                 stroke_width: float = 0,
962: (8)                 color: ParsableManimColor | None = None,
963: (8)                 font_size: float = DEFAULT_FONT_SIZE,
964: (8)                 line_spacing: int = -1,
965: (8)                 font: str = "",
966: (8)                 slant: str = NORMAL,
967: (8)                 weight: str = NORMAL,
968: (8)                 justify: bool = False,
969: (8)                 gradient: tuple = None,
970: (8)                 tab_width: int = 4,
971: (8)                 height: int = None,
972: (8)                 width: int = None,
973: (8)                 should_center: bool = True,
974: (8)                 disable_ligatures: bool = False,
975: (8)                 warn_missing_font: bool = True,
976: (8)                 **kwargs,
977: (4)             ) -> None:
978: (8)                 self.text = text
979: (8)                 self.line_spacing = line_spacing
980: (8)                 if font and warn_missing_font:
981: (12)                     fonts_list = Text.font_list()

```

```

982: (12)             if font.lower() == "sans-serif":
983: (16)                 font = "sans"
984: (12)             if font not in fonts_list:
985: (16)                 if font.capitalize() in fonts_list:
986: (20)                     font = font.capitalize()
987: (16)                 elif font.lower() in fonts_list:
988: (20)                     font = font.lower()
989: (16)                 elif font.title() in fonts_list:
990: (20)                     font = font.title()
991: (16)                 else:
992: (20)                     logger.warning(f"Font {font} not in {fonts_list}.")
993: (8)             self.font = font
994: (8)             self._font_size = float(font_size)
995: (8)             self.slant = slant
996: (8)             self.weight = weight
997: (8)             self.gradient = gradient
998: (8)             self.tab_width = tab_width
999: (8)             self.justify = justify
1000: (8)             self.original_text = text
1001: (8)             self.disable_ligatures = disable_ligatures
1002: (8)             text_without_tabs = text
1003: (8)             if "\t" in text:
1004: (12)                 text_without_tabs = text.replace("\t", " " * self.tab_width)
1005: (8)             colormap = self._extract_color_tags()
1006: (8)             if len(colormap) > 0:
1007: (12)                 logger.warning(
1008: (16)                     'Using <color> tags in MarkupText is deprecated. Please use
<span foreground="...> instead.')
1009: (12)             )
1010: (8)             gradientmap = self._extract_gradient_tags()
1011: (8)             validate_error = MarkupUtils.validate(self.text)
1012: (8)             if validate_error:
1013: (12)                 raise ValueError(validate_error)
1014: (8)             if self.line_spacing == -1:
1015: (12)                 self.line_spacing = (
1016: (16)                     self._font_size + self._font_size * DEFAULT_LINE_SPACING_SCALE
1017: (12)                 )
1018: (8)             else:
1019: (12)                 self.line_spacing = self._font_size + self._font_size *
self.line_spacing
1020: (8)             color: ManimColor = ManimColor(color) if color else VMobject().color
1021: (8)             file_name = self._text2svg(color)
1022: (8)             PangоФutils.remove_last_M(file_name)
1023: (8)             super().__init__(
1024: (12)                 file_name,
1025: (12)                 fill_opacity=fill_opacity,
1026: (12)                 stroke_width=stroke_width,
1027: (12)                 height=height,
1028: (12)                 width=width,
1029: (12)                 should_center=should_center,
1030: (12)                 **kwargs,
1031: (8)             )
1032: (8)             self.chars = self.get_group_class()(*self.submobjects)
1033: (8)             self.text = text_without_tabs.replace(" ", "").replace("\n", "")
1034: (8)             nppc = self.n_points_per_curve
1035: (8)             for each in self:
1036: (12)                 if len(each.points) == 0:
1037: (16)                     continue
1038: (12)                 points = each.points
1039: (12)                 curve_start = points[0]
1040: (12)                 assert len(curve_start) == self.dim, curve_start
1041: (12)                 closed_curve_points = []
1042: (12)                 if nppc == 3: # RendererType.OPENGL
1043: (16)                     def add_line_to(end):
1044: (20)                         nonlocal closed_curve_points
1045: (20)                         start = closed_curve_points[-1]
1046: (20)                         closed_curve_points += [
1047: (24)                             start,
1048: (24)                             (start + end) / 2,
```

```

1049: (24)                     end,
1050: (20)                 ]
1051: (12)             else: # RendererType.CAIRO
1052: (16)                 def add_line_to(end):
1053: (20)                     nonlocal closed_curve_points
1054: (20)                     start = closed_curve_points[-1]
1055: (20)                     closed_curve_points += [
1056: (24)                         start,
1057: (24)                         (start + start + end) / 3,
1058: (24)                         (start + end + end) / 3,
1059: (24)                         end,
1060: (20)                     ]
1061: (12)             for index, point in enumerate(points):
1062: (16)                 closed_curve_points.append(point)
1063: (16)             if (
1064: (20)                 index != len(points) - 1
1065: (20)                 and (index + 1) % nppc == 0
1066: (20)                 and any(point != points[index + 1])
1067: (16)             ):
1068: (20)                 add_line_to(curve_start)
1069: (20)                 curve_start = points[index + 1]
1070: (12)             add_line_to(curve_start)
1071: (12)             each.points = np.array(closed_curve_points, ndmin=2)
1072: (8)         if self.gradient:
1073: (12)             self.set_color_by_gradient(*self.gradient)
1074: (8)         for col in colormap:
1075: (12)             self.chars[
1076: (16)                 col["start"]
1077: (16)                 - col["start_offset"] : col["end"]
1078: (16)                 - col["start_offset"]
1079: (16)                 - col["end_offset"]
1080: (12)             ].set_color(self._parse_color(col["color"]))
1081: (8)         for grad in gradientmap:
1082: (12)             self.chars[
1083: (16)                 grad["start"]
1084: (16)                 - grad["start_offset"] : grad["end"]
1085: (16)                 - grad["start_offset"]
1086: (16)                 - grad["end_offset"]
1087: (12)             ].set_color_by_gradient(
1088: (16)                 *(self._parse_color(grad["from"]),
self._parse_color(grad["to"]))
1089: (12)             )
1090: (8)             if height is None and width is None:
1091: (12)                 self.scale(TEXT_MOB_SCALE_FACTOR)
1092: (8)             self.initial_height = self.height
1093: (4)         @property
1094: (4)             def font_size(self):
1095: (8)                 return (
1096: (12)                     self.height
1097: (12)                     / self.initial_height
1098: (12)                     / TEXT_MOB_SCALE_FACTOR
1099: (12)                     * 2.4
1100: (12)                     * self._font_size
1101: (12)                     / DEFAULT_FONT_SIZE
1102: (8)                 )
1103: (4)         @font_size.setter
1104: (4)             def font_size(self, font_val):
1105: (8)                 if font_val <= 0:
1106: (12)                     raise ValueError("font_size must be greater than 0.")
1107: (8)                 else:
1108: (12)                     self.scale(font_val / self.font_size)
1109: (4)             def _text2hash(self, color: ParsableManimColor):
1110: (8)                 """Generates ``sha256`` hash for file name."""
1111: (8)                 settings = (
1112: (12)                     "MARKUPPANGO"
1113: (12)                     + self.font
1114: (12)                     + self.slant
1115: (12)                     + self.weight
1116: (12)                     + ManimColor(color).to_hex().lower()

```

```

1117: (8) ) # to differentiate from classical Pango Text
1118: (8) settings += str(self.line_spacing) + str(self._font_size)
1119: (8) settings += str(self.disable_ligatures)
1120: (8) settings += str(self.justify)
1121: (8) id_str = self.text + settings
1122: (8) hasher = hashlib.sha256()
1123: (8) hasher.update(id_str.encode())
1124: (8) return hasher.hexdigest()[:16]
1125: (4) def _text2svg(self, color: ParsableManimColor | None):
1126: (8) """Convert the text to SVG using Pango."""
1127: (8) color = ManimColor(color)
1128: (8) size = self._font_size
1129: (8) line_spacing = self.line_spacing
1130: (8) size /= TEXT2SVG_ADJUSTMENT_FACTOR
1131: (8) line_spacing /= TEXT2SVG_ADJUSTMENT_FACTOR
1132: (8) dir_name = config.get_dir("text_dir")
1133: (8) if not dir_name.is_dir():
1134: (12)     dir_name.mkdir(parents=True)
1135: (8) hash_name = self._text2hash(color)
1136: (8) file_name = dir_name / (hash_name + ".svg")
1137: (8) if file_name.exists():
1138: (12)     svg_file = str(file_name.resolve())
1139: (8) else:
1140: (12)     final_text = (
1141: (16)         f'<span foreground="{color.to_hex()}">{self.text}</span>'
1142: (16)         if color is not None
1143: (16)         else self.text
1144: (12)     )
1145: (12)     logger.debug(f"Setting Text {self.text}")
1146: (12)     svg_file = MarkupUtils.text2svg(
1147: (16)         final_text,
1148: (16)         self.font,
1149: (16)         self.slant,
1150: (16)         self.weight,
1151: (16)         size,
1152: (16)         line_spacing,
1153: (16)         self.disable_ligatures,
1154: (16)         str(file_name.resolve()),
1155: (16)         START_X,
1156: (16)         START_Y,
1157: (16)         600, # width
1158: (16)         400, # height
1159: (16)         justify=self.justify,
1160: (16)         pango_width=500,
1161: (12)     )
1162: (8)     return svg_file
1163: (4) def _count_real_chars(self, s):
1164: (8) """Counts characters that will be displayed.
1165: (8) This is needed for partial coloring or gradients, because space
1166: (8) counts to the text's `len`, but has no corresponding character."""
1167: (8) count = 0
1168: (8) level = 0
1169: (8) s = re.sub("&[^;]+;", "x", s)
1170: (8) for c in s:
1171: (12)     if c == "<":
1172: (16)         level += 1
1173: (12)     if c == ">" and level > 0:
1174: (16)         level -= 1
1175: (12)     elif c != " " and c != "\t" and level == 0:
1176: (16)         count += 1
1177: (8) return count
1178: (4) def _extract_gradient_tags(self):
1179: (8) """Used to determine which parts (if any) of the string should be
formatted
1180: (8) with a gradient.
1181: (8) Removes the ``<gradient>`` tag, as it is not part of Pango's markup
and would cause an error.
1182: (8)
1183: (8) """
tags = re.finditer(

```

12/20/24, 4:24 AM manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

1184: (12) r'<gradient\s+from="([^\"]+)"\s+to="([^\"]+)"\s+offset="([^\"]+)"?>
1185: (12) (.+?)</gradient>',
1186: (12) self.original_text,
1187: (8) re.S,
1188: (8)
1189: (8)
1190: (12) gradientmap = []
1191: (12) for tag in tags:
1192: (12)     start = self._count_real_chars(self.original_text[: tag.start(0)])
1193: (12)     end = start + self._count_real_chars(tag.group(5))
1194: (12)     offsets = tag.group(4).split(",") if tag.group(4) else [0]
1195: (12)     start_offset = int(offsets[0]) if offsets[0] else 0
1196: (16)     end_offset = int(offsets[1]) if len(offsets) == 2 and offsets[1]
1197: (20) else 0
1198: (20)
1199: (20)
1200: (20)
1201: (20)
1202: (20)
1203: (16)
1204: (12)
1205: (8) self.text = re.sub("<gradient[^>]+>(.+?)</gradient>", r"\1",
1206: (8) self.text, 0, re.S)
1207: (4)
1208: (8)
1209: (8)
1210: (12)
1211: (8)
1212: (12)
1213: (4)
1214: (8)
1215: (8)
1216: (8)
1217: (8)
1218: (8)
1219: (8)
1220: (8)
1221: (12)
1222: (12)
1223: (12)
1224: (8)
1225: (8)
1226: (8)
1227: (12)
1228: (12)
1229: (12)
1230: (12)
1231: (12)
1232: (12)
1233: (16)
1234: (20)
1235: (20)
1236: (20)
1237: (20)
1238: (20)
1239: (16)
1240: (12)
1241: (8)
1242: (8)
1243: (4)
1244: (8)

def _parse_color(self, col):
    """Parse color given in ``<color>`` or ``<gradient>`` tags."""
    if re.match("#[0-9a-f]{6}", col):
        return col
    else:
        return ManimColor(col).to_hex()

def _extract_color_tags(self):
    """Used to determine which parts (if any) of the string should be
    with a custom color.
    Removes the ``<color>`` tag, as it is not part of Pango's markup and
    Note: Using the ``<color>`` tags is deprecated. As soon as the legacy
    syntax is gone, this function
    will be removed.
    """
    tags = re.finditer(
        r'<color\s+col="([^\"]+)"\s+offset="([^\"]+)"?>(.+?)</color>',
        self.original_text,
        re.S,
    )
    colormap = []
    for tag in tags:
        start = self._count_real_chars(self.original_text[: tag.start(0)])
        end = start + self._count_real_chars(tag.group(4))
        offsets = tag.group(3).split(",") if tag.group(3) else [0]
        start_offset = int(offsets[0]) if offsets[0] else 0
        end_offset = int(offsets[1]) if len(offsets) == 2 and offsets[1]
        colormap.append(
            {
                "start": start,
                "end": end,
                "color": tag.group(1),
                "start_offset": start_offset,
                "end_offset": end_offset,
            },
        )
    self.text = re.sub("<color[^>]+>(.+?)</color>", r"\1", self.text, 0, re.S)
    return colormap

def __repr__(self):
    return f"MarkupText({repr(self.original_text)})"

```

```

1245: (0)             @contextmanager
1246: (0)             def register_font(font_file: str | Path):
1247: (4)                 """Temporarily add a font file to Pango's search path.
1248: (4)                 This searches for the font_file at various places. The order it searches
it described below.
1249: (4)                 1. Absolute path.
1250: (4)                 2. In ``assets/fonts`` folder.
1251: (4)                 3. In ``font/`` folder.
1252: (4)                 4. In the same directory.
1253: (4)             Parameters
1254: (4)             -----
1255: (4)             font_file
1256: (8)                 The font file to add.
1257: (4)             Examples
1258: (4)             -----
1259: (4)                 Use ``with register_font(...)`` to add a font file to search
1260: (4)                 path.
1261: (4)                 .. code-block:: python
1262: (8)                     with register_font("path/to/font_file.ttf"):
1263: (12)                         a = Text("Hello", font="Custom Font Name")
1264: (4)             Raises
1265: (4)             -----
1266: (4)             FileNotFoundError:
1267: (8)                 If the font doesn't exists.
1268: (4)             AttributeError:
1269: (8)                 If this method is used on macOS.
1270: (4)             .. important :::
1271: (8)                 This method is available for macOS for ``ManimPango>=v0.2.3``. Using
this
1272: (8)                 method with previous releases will raise an :class:`AttributeError` on
macOS.
1273: (4)             """
1274: (4)             input_folder = Path(config.input_file).parent.resolve()
1275: (4)             possible_paths = [
1276: (8)                 Path(font_file),
1277: (8)                 input_folder / "assets/fonts" / font_file,
1278: (8)                 input_folder / "fonts" / font_file,
1279: (8)                 input_folder / font_file,
1280: (4)             ]
1281: (4)             for path in possible_paths:
1282: (8)                 path = path.resolve()
1283: (8)                 if path.exists():
1284: (12)                     file_path = path
1285: (12)                     logger.debug("Found file at %s", file_path.absolute())
1286: (12)                     break
1287: (4)             else:
1288: (8)                 error = f"Can't find {font_file}." f" Tried these : {possible_paths}"
1289: (8)                 raise FileNotFoundError(error)
1290: (4)             try:
1291: (8)                 assert manimpango.register_font(str(file_path))
1292: (8)                 yield
1293: (4)             finally:
1294: (8)                 manimpango.unregister_font(str(file_path))

```

## File 86 - three\_d\_utils.py:

```

1: (0)             """Utility functions for three-dimensional mobjects."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "get_3d_vmob_gradient_start_and_end_points",
5: (4)                 "get_3d_vmob_start_corner_index",
6: (4)                 "get_3d_vmob_end_corner_index",
7: (4)                 "get_3d_vmob_start_corner",
8: (4)                 "get_3d_vmob_end_corner",
9: (4)                 "get_3d_vmob_unit_normal",
10: (4)                "get_3d_vmob_start_corner_unit_normal",
11: (4)                "get_3d_vmob_end_corner_unit_normal",

```

```

12: (0) ]
13: (0)     from typing import TYPE_CHECKING, Literal
14: (0)     import numpy as np
15: (0)     from manim.constants import ORIGIN, UP
16: (0)     from manim.utils.space_ops import get_unit_normal
17: (0)     if TYPE_CHECKING:
18: (4)         from manim.typing import Point3D, Vector3D
19: (0) def get_3d_vmob_gradient_start_and_end_points(vmob) -> tuple[Point3D,
Point3D]:
20: (4)     return (
21: (8)         get_3d_vmob_start_corner(vmob),
22: (8)         get_3d_vmob_end_corner(vmob),
23: (4)     )
24: (0) def get_3d_vmob_start_corner_index(vmob) -> Literal[0]:
25: (4)     return 0
26: (0) def get_3d_vmob_end_corner_index(vmob) -> int:
27: (4)     return ((len(vmob.points) - 1) // 6) * 3
28: (0) def get_3d_vmob_start_corner(vmob) -> Point3D:
29: (4)     if vmob.get_num_points() == 0:
30: (8)         return np.array(ORIGIN)
31: (4)     return vmob.points[get_3d_vmob_start_corner_index(vmob)]
32: (0) def get_3d_vmob_end_corner(vmob) -> Point3D:
33: (4)     if vmob.get_num_points() == 0:
34: (8)         return np.array(ORIGIN)
35: (4)     return vmob.points[get_3d_vmob_end_corner_index(vmob)]
36: (0) def get_3d_vmob_unit_normal(vmob, point_index: int) -> Vector3D:
37: (4)     n_points = vmob.get_num_points()
38: (4)     if len(vmob.get_anchors()) <= 2:
39: (8)         return np.array(UP)
40: (4)     i = point_index
41: (4)     im3 = i - 3 if i > 2 else (n_points - 4)
42: (4)     ip3 = i + 3 if i < (n_points - 3) else 3
43: (4)     unit_normal = get_unit_normal(
44: (8)         vmob.points[ip3] - vmob.points[i],
45: (8)         vmob.points[im3] - vmob.points[i],
46: (4)     )
47: (4)     if np.linalg.norm(unit_normal) == 0:
48: (8)         return np.array(UP)
49: (4)     return unit_normal
50: (0) def get_3d_vmob_start_corner_unit_normal(vmob) -> Vector3D:
51: (4)     return get_3d_vmob_unit_normal(vmob, get_3d_vmob_start_corner_index(vmob))
52: (0) def get_3d_vmob_end_corner_unit_normal(vmob) -> Vector3D:
53: (4)     return get_3d_vmob_unit_normal(vmob, get_3d_vmob_end_corner_index(vmob))

```

---

## File 87 - image\_mobject.py:

```

1: (0)     """Mobjects representing raster images."""
2: (0)     from __future__ import annotations
3: (0)     __all__ = ["AbstractImageMobject", "ImageMobject", "ImageMobjectFromCamera"]
4: (0)     import pathlib
5: (0)     import numpy as np
6: (0)     from PIL import Image
7: (0)     from PIL.Image import Resampling
8: (0)     from manim.mobject.geometry.shape_matchers import SurroundingRectangle
9: (0)     from ... import config
10: (0)    from ...constants import *
11: (0)    from ...mobject.mobject import Mobject
12: (0)    from ...utils.bezier import interpolate
13: (0)    from ...utils.color import WHITE, ManimColor, color_to_int_rgb
14: (0)    from ...utils.images import change_to_rgba_array, get_full_raster_image_path
15: (0)    __all__ = ["ImageMobject", "ImageMobjectFromCamera"]
16: (0)    class AbstractImageMobject(Mobject):
17: (4)        """
18: (4)            Automatically filters out black pixels
19: (4)            Parameters
20: (4)            -----
21: (4)            scale_to_resolution

```

```

22: (8)                                At this resolution the image is placed pixel by pixel onto the screen,
so it
23: (8)                                will look the sharpest and best.
24: (8)                                This is a custom parameter of ImageMobject so that rendering a scene
with
25: (8)                                e.g. the ``--quality low`` or ``--quality medium`` flag for faster
rendering
26: (8)                                won't effect the position of the image on the screen.
27: (4)
28: (4)                                """
29: (8)                                def __init__(
30: (8)                                    self,
31: (8)                                    scale_to_resolution: int,
32: (8)                                    pixel_array_dtype="uint8",
33: (8)                                    resampling_algorithm=Resampling.BICUBIC,
34: (8)                                    **kwargs,
35: (8)                                ):
36: (8)                                    self.pixel_array_dtype = pixel_array_dtype
37: (8)                                    self.scale_to_resolution = scale_to_resolution
38: (8)                                    self.set_resampling_algorithm(resampling_algorithm)
39: (4)                                    super().__init__(**kwargs)
40: (8)                                def get_pixel_array(self):
41: (4)                                    raise NotImplementedError()
42: (8)                                def set_color(self, color, alpha=None, family=True):
43: (4)                                    pass
44: (8)                                def set_resampling_algorithm(self, resampling_algorithm: int):
45: (8)                                    """
46: (8)  Sets the interpolation method for upscaling the image. By default the
47: (8)  interpolated using bicubic algorithm. This method lets you change it.
48: (8)  Interpolation is done internally using Pillow, and the function
49: (8)  string constants describing the algorithm accepts the Pillow integer
50: (8)  Parameters
51: (8)  -----
52: (12)  resampling_algorithm
53: (12)  An integer constant described in the Pillow library,
54: (12)  or one from the RESAMPLING_ALGORITHMS global dictionary,
55: (12)  under the following keys:
56: (12)  * 'bicubic' or 'cubic'
57: (12)  * 'nearest' or 'none'
58: (12)  * 'box'
59: (12)  * 'bilinear' or 'linear'
60: (12)  * 'hamming'
61: (8)  * 'lanczos' or 'antialias'
62: (8)
63: (12)  if isinstance(resampling_algorithm, int):
64: (8)  self.resampling_algorithm = resampling_algorithm
65: (8)  else:
66: (16)  raise ValueError(
67: (16)  "resampling_algorithm has to be an int, one of the values
68: (16)  defined in "
69: (16)  "RESAMPLING_ALGORITHMS or a Pillow resampling filter constant.
68: (16)  "
70: (12)  "Available algorithms: 'bicubic', 'nearest', 'box',
71: (8)  "'hamming', 'lanczos'.",
72: (4)
73: (8)  )
74: (8)
75: (12)
76: (16)
77: (16)
78: (16)
79: (16)
80: (12)
81: (8)

```

```

82: (8)                     self.center()
83: (8)                     h, w = self.get_pixel_array().shape[:2]
84: (8)                     if self.scale_to_resolution:
85: (12)                         height = h / self.scale_to_resolution * config["frame_height"]
86: (8)                     else:
87: (12)                         height = 3 # this is the case for ImageMobjectFromCamera
88: (8)                         self.stretch_to_fit_height(height)
89: (8)                         self.stretch_to_fit_width(height * w / h)
90: (0)             class ImageMobject(AbstractImageMobject):
91: (4)                 """Displays an Image from a numpy array or a file.
92: (4)                 Parameters
93: (4)                 -----
94: (4)                 scale_to_resolution
95: (8)                     At this resolution the image is placed pixel by pixel onto the screen,
so it
96: (8)                     will look the sharpest and best.
97: (8)                     This is a custom parameter of ImageMobject so that rendering a scene
with
98: (8)                     e.g. the ``--quality low`` or ``--quality medium`` flag for faster
rendering
99: (8)                     won't effect the position of the image on the screen.
100: (4)             Example
101: (4)             -----
102: (4)             .. manim:: ImageFromArray
103: (8)                 :save_last_frame:
104: (8)             class ImageFromArray(Scene):
105: (12)                 def construct(self):
106: (16)                     image = ImageMobject(np.uint8([[0, 100, 30, 200],
107: (47)                                     [255, 0, 5, 33]]))
108: (16)                     image.height = 7
109: (16)                     self.add(image)
110: (4)             Changing interpolation style:
111: (4)             .. manim:: ImageInterpolationEx
112: (8)                 :save_last_frame:
113: (8)             class ImageInterpolationEx(Scene):
114: (12)                 def construct(self):
115: (16)                     img = ImageMobject(np.uint8([[63, 0, 0, 0],
116: (48)                                     [0, 127, 0, 0],
117: (48)                                     [0, 0, 191, 0],
118: (48)                                     [0, 0, 0, 255]
119: (48)                                     ]]))
120: (16)                     img.height = 2
121: (16)                     img1 = img.copy()
122: (16)                     img2 = img.copy()
123: (16)                     img3 = img.copy()
124: (16)                     img4 = img.copy()
125: (16)                     img5 = img.copy()
126: (16)


```

```

146: (8)                         **kwargs,
147: (4) :
148: (8)             self.fill_opacity = 1
149: (8)             self.stroke_opacity = 1
150: (8)             self.invert = invert
151: (8)             self.image_mode = image_mode
152: (8)             if isinstance(filename_or_array, (str, pathlib.PurePath)):
153: (12)                 path = get_full_raster_image_path(filename_or_array)
154: (12)                 image = Image.open(path).convert(self.image_mode)
155: (12)                 self.pixel_array = np.array(image)
156: (12)                 self.path = path
157: (8)             else:
158: (12)                 self.pixel_array = np.array(filename_or_array)
159: (8)             self.pixel_array_dtype = kwargs.get("pixel_array_dtype", "uint8")
160: (8)             self.pixel_array = change_to_rgba_array(
161: (12)                 self.pixel_array, self.pixel_array_dtype
162: (8)             )
163: (8)             if self.invert:
164: (12)                 self.pixel_array[:, :, :3] = (
165: (16)                     np.iinfo(self.pixel_array_dtype).max - self.pixel_array[:, :, :3]
166: (12)                 )
167: (8)             super().__init__(scale_to_resolution, **kwargs)
168: (4) def get_pixel_array(self):
169: (8)     """A simple getter method."""
170: (8)     return self.pixel_array
171: (4) def set_color(self, color, alpha=None, family=True):
172: (8)     rgb = color_to_int_rgb(color)
173: (8)     self.pixel_array[:, :, :3] = rgb
174: (8)     if alpha is not None:
175: (12)         self.pixel_array[:, :, 3] = int(255 * alpha)
176: (8)     for submob in self.submobjects:
177: (12)         submob.set_color(color, alpha, family)
178: (8)     self.color = color
179: (8)     return self
180: (4) def set_opacity(self, alpha: float):
181: (8)     """Sets the image's opacity.
182: (8)     Parameters
183: (8)     -----
184: (8)     alpha
185: (12)         The alpha value of the object, 1 being opaque and 0 being
186: (12)         transparent.
187: (8)     """
188: (8)     self.pixel_array[:, :, 3] = int(255 * alpha)
189: (8)     self.fill_opacity = alpha
190: (8)     self.stroke_opacity = alpha
191: (8)     return self
192: (4) def fade(self, darkness: float = 0.5, family: bool = True):
193: (8)     """Sets the image's opacity using a 1 - alpha relationship.
194: (8)     Parameters
195: (8)     -----
196: (8)     darkness
197: (12)         The alpha value of the object, 1 being transparent and 0 being
198: (12)         opaque.
199: (8)     family
200: (12)         Whether the submobjects of the ImageMobject should be affected.
201: (8)     """
202: (8)     self.set_opacity(1 - darkness)
203: (8)     super().fade(darkness, family)
204: (8)     return self
205: (4) def interpolate_color(
206: (8)     self, mobject1: ImageMobject, mobject2: ImageMobject, alpha: float
207: (4) ):
208: (8)     """Interpolates the array of pixel color values from one ImageMobject
209: (8)     into an array of equal size in the target ImageMobject.
210: (8)     Parameters
211: (8)     -----
212: (8)     mobject1
213: (12)         The ImageMobject to transform from.

```

```

214: (8)             mobject2
215: (12)            The ImageMobject to transform into.
216: (8)             alpha
217: (12)            Used to track the lerp relationship. Not opacity related.
218: (8)
219: (8)             """
220: (12)            assert mobject1.pixel_array.shape == mobject2.pixel_array.shape, (
221: (12)              f"Mobect pixel array shapes incompatible for interpolation.\n"
222: (12)              f"Mobect 1 ({mobject1}) : {mobject1.pixel_array.shape}\n"
223: (12)              f"Mobect 2 ({mobject2}) : {mobject2.pixel_array.shape}"
224: (8)            )
225: (12)            self.fill_opacity = interpolate(
226: (12)              mobject1.fill_opacity,
227: (12)              mobject2.fill_opacity,
228: (8)              alpha,
229: (8)            )
230: (12)            self.stroke_opacity = interpolate(
231: (12)              mobject1.stroke_opacity,
232: (12)              mobject2.stroke_opacity,
233: (8)              alpha,
234: (8)            )
235: (12)            self.pixel_array = interpolate(
236: (12)              mobject1.pixel_array,
237: (12)              mobject2.pixel_array,
238: (8)              alpha,
239: (4)            ).astype(self.pixel_array_dtype)
240: (8)        def get_style(self):
241: (12)          return {
242: (12)            "fill_color": ManimColor(self.color.get_rgb()).to_hex(),
243: (8)            "fill_opacity": self.fill_opacity,
244: (0)          }
245: (4)        class ImageMobjectFromCamera(AbstractImageMobject):
246: (8)          def __init__(self, camera, default_display_frame_config=None, **kwargs):
247: (8)            self.camera = camera
248: (12)            if default_display_frame_config is None:
249: (16)              default_display_frame_config = {
250: (16)                "stroke_width": 3,
251: (16)                "stroke_color": WHITE,
252: (12)                "buff": 0,
253: (8)              }
254: (8)            self.default_display_frame_config = default_display_frame_config
255: (8)            self.pixel_array = self.camera.pixel_array
256: (4)            super().__init__(scale_to_resolution=False, **kwargs)
257: (8)        def get_pixel_array(self):
258: (8)          self.pixel_array = self.camera.pixel_array
259: (8)          return self.pixel_array
260: (4)        def add_display_frame(self, **kwargs):
261: (8)          config = dict(self.default_display_frame_config)
262: (8)          config.update(kwargs)
263: (8)          self.display_frame = SurroundingRectangle(self, **config)
264: (8)          self.add(self.display_frame)
265: (4)          return self
266: (8)        def interpolate_color(self, mobject1, mobject2, alpha):
267: (12)          assert mobject1.pixel_array.shape == mobject2.pixel_array.shape, (
268: (12)            f"Mobect pixel array shapes incompatible for interpolation.\n"
269: (12)            f"Mobect 1 ({mobject1}) : {mobject1.pixel_array.shape}\n"
270: (8)            f"Mobect 2 ({mobject2}) : {mobject2.pixel_array.shape}"
271: (8)          )
272: (12)          self.pixel_array = interpolate(
273: (12)            mobject1.pixel_array,
274: (12)            mobject2.pixel_array,
275: (8)            alpha,
276: (8)          ).astype(self.pixel_array_dtype)

```

-----  
File 88 - opengl\_mobject.py:

```

1: (0)            from __future__ import annotations
2: (0)            import copy

```

```

3: (0)          import inspect
4: (0)          import itertools as it
5: (0)          import random
6: (0)          import sys
7: (0)          from functools import partialmethod, wraps
8: (0)          from math import ceil
9: (0)          from typing import Iterable, Sequence
10: (0)         import moderngl
11: (0)         import numpy as np
12: (0)         from manim import config, logger
13: (0)         from manim.constants import *
14: (0)         from manim.renderer.shader_wrapper import get_colormap_code
15: (0)         from manim.utils.bezier import integer_interpolate, interpolate
16: (0)         from manim.utils.color import (
17: (4)             WHITE,
18: (4)             ManimColor,
19: (4)             ParsableManimColor,
20: (4)             color_gradient,
21: (4)             color_to_rgb,
22: (4)             rgb_to_hex,
23: (0)         )
24: (0)         from manim.utils.config_ops import _Data, _Uniforms
25: (0)         from manim.utils.iterables import (
26: (4)             batch_by_property,
27: (4)             list_update,
28: (4)             listify,
29: (4)             make_even,
30: (4)             resize_array,
31: (4)             resize_preserving_order,
32: (4)             resize_with_interpolation,
33: (4)             uniq_chain,
34: (0)         )
35: (0)         from manim.utils.paths import straight_path
36: (0)         from manim.utils.space_ops import (
37: (4)             angle_between_vectors,
38: (4)             normalize,
39: (4)             rotation_matrix_transpose,
40: (0)     )
41: (0)     def affects_shader_info_id(func):
42: (4)         @wraps(func)
43: (4)         def wrapper(self):
44: (8)             for mob in self.get_family():
45: (12)                 func(mob)
46: (12)                 mob.refresh_shader_wrapper_id()
47: (8)             return self
48: (4)         return wrapper
49: (0)     __all__ = ["OpenGLObject", "OpenGLGroup", "OpenGLPoint", "_AnimationBuilder"]
50: (0)     class OpenGLObject:
51: (4)         """Mathematical Object: base class for objects that can be displayed on
screen.
52: (4)         Attributes
53: (4)         -----
54: (4)         subobjects : List[:class:`~OpenGLObject`]
55: (8)             The contained objects.
56: (4)         points : :class:`numpy.ndarray`
57: (8)             The points of the objects.
58: (8)             .. seealso::
59: (12)                 :class:`~.OpenGLMobject`
50: (4)         """
51: (4)         shader_dtype = [
52: (8)             ("point", np.float32, (3,)),
53: (4)         ]
54: (4)         shader_folder = ""
55: (4)         points = _Data()
56: (4)         bounding_box = _Data()
57: (4)         rgbs = _Data()
58: (4)         is_fixed_in_frame = _Uniforms()
59: (4)         is_fixed_orientation = _Uniforms()
70: (4)         fixed_orientation_center = _Uniforms() # for fixed orientation reference

```

```

71: (4)                      gloss = _Uniforms()
72: (4)                      shadow = _Uniforms()
73: (4)                      def __init__(self,
74: (8)                          color=WHITE,
75: (8)                          opacity=1,
76: (8)                          dim=3, # TODO, get rid of this
77: (8)                          gloss=0.0,
78: (8)                          shadow=0.0,
79: (8)                          render_primitive=moderngl.TRIANGLES,
80: (8)                          texture_paths=None,
81: (8)                          depth_test=False,
82: (8)                          is_fixed_in_frame=False,
83: (8)                          is_fixed_orientation=False,
84: (8)                          listen_to_events=False,
85: (8)                          model_matrix=None,
86: (8)                          should_render=True,
87: (8)                          name: str | None = None,
88: (8)                          **kwargs,
89: (8)
90: (4)                      ):
91: (8)                          self.name = self.__class__.__name__ if name is None else name
92: (8)                          self.data = getattr(self, "data", {})
93: (8)                          self.uniforms = getattr(self, "uniforms", {})
94: (8)                          self.opacity = opacity
95: (8)                          self.dim = dim # TODO, get rid of this
96: (8)                          self.gloss = gloss
97: (8)                          self.shadow = shadow
98: (8)                          self.render_primitive = render_primitive
99: (8)                          self.texture_paths = texture_paths
100: (8)                         self.depth_test = depth_test
101: (8)                         self.is_fixed_in_frame = float(is_fixed_in_frame)
102: (8)                         self.is_fixed_orientation = float(is_fixed_orientation)
103: (8)                         self.fixed_orientation_center = (0, 0, 0)
104: (8)                         self.listen_to_events = listen_to_events
105: (8)                         self._subobjects = []
106: (8)                         self.parents = []
107: (8)                         self.parent = None
108: (8)                         self.family = [self]
109: (8)                         self.locked_data_keys = set()
110: (8)                         self.needs_new_bounding_box = True
111: (8)                         if model_matrix is None:
112: (12)                             self.model_matrix = np.eye(4)
113: (8)                         else:
114: (12)                             self.model_matrix = model_matrix
115: (8)                         self.init_data()
116: (8)                         self.init_updaters()
117: (8)                         self.init_points()
118: (8)                         self.color = ManimColor.parse(color)
119: (8)                         self.init_colors()
120: (8)                         self.shader_indices = None
121: (8)                         if self.depth_test:
122: (12)                             self.apply_depth_test()
123: (8)                         self.should_render = should_render
124: (4)                         @classmethod
125: (4)                         def __init_subclass__(cls, **kwargs):
126: (8)                             super().__init_subclass__(**kwargs)
127: (8)                             cls._original_init_ = cls.__init__
128: (4)                         def __str__(self):
129: (8)                             return self.__class__.__name__
130: (4)                         def __repr__(self):
131: (8)                             return str(self.name)
132: (4)                         def __sub__(self, other):
133: (8)                             return NotImplemented
134: (4)                         def __isub__(self, other):
135: (8)                             return NotImplemented
136: (4)                         def __add__(self, mobject):
137: (8)                             return NotImplemented
138: (4)                         def __iadd__(self, mobject):
139: (8)                             return NotImplemented

```

```

140: (4) @classmethod
141: (4) def set_default(cls, **kwargs):
142: (8)     """Sets the default values of keyword arguments.
143: (8)     If this method is called without any additional keyword
144: (8)     arguments, the original default values of the initialization
145: (8)     method of this class are restored.
146: (8) Parameters
147: (8) -----
148: (8)     kwargs
149: (12)         Passing any keyword argument will update the default
150: (12)         values of the keyword arguments of the initialization
151: (12)         function of this class.
152: (8) Examples
153: (8) -----
154: (8) :::
155: (12)     >>> from manim import Square, GREEN
156: (12)     >>> Square.set_default(color=GREEN, fill_opacity=0.25)
157: (12)     >>> s = Square(); s.color, s.fill_opacity
158: (12)     (ManimColor('#83C167'), 0.25)
159: (12)     >>> Square.set_default()
160: (12)     >>> s = Square(); s.color, s.fill_opacity
161: (12)     (ManimColor('#FFFFFF'), 0.0)
162: (8) .. manim:: ChangedDefaultTextColor
163: (12)     :save_last_frame:
164: (12)     config.background_color = WHITE
165: (12)     class ChangedDefaultTextColor(Scene):
166: (16)         def construct(self):
167: (20)             Text.set_default(color=BLACK)
168: (20)             self.add(Text("Changing default values is easy!"))
169: (20)             Text.set_default(color=WHITE)
170: (8)     """
171: (8)     if kwargs:
172: (12)         cls.__init__ = partialmethod(cls.__init__, **kwargs)
173: (8)     else:
174: (12)         cls.__init__ = cls._original_init_
175: (4)     def init_data(self):
176: (8)         """Initializes the ``points``, ``bounding_box`` and ``rgbas``
177: (8) attributes and groups them into self.data.
178: (8) Subclasses can inherit and overwrite this method to extend
`self.data`.
179: (8)
180: (8)
181: (8)
182: (8)
183: (8)
184: (8)
185: (8)
186: (8)
187: (8)
188: (8)
189: (8)
190: (4)     def set(self, **kwargs) -> OpenGLObject:
191: (8)         """Sets attributes.
192: (8)         Mainly to be used along with :attr:`animate` to
193: (8)         animate setting attributes.
194: (8) Examples
195: (8) -----
196: (8) :::
197: (12)     >>> mob = OpenGLObject()
198: (12)     >>> mob.set(foo=0)
199: (12)     OpenGLObject
200: (12)     >>> mob.foo
201: (12)     0
202: (8)
203: (8)
204: (8)
205: (12)     Parameters
206: (8) -----
207: (8)     **kwargs
208: (8)         The attributes and corresponding values to set.

```

```

206: (8)             Returns
207: (8)             -----
208: (8)             :class:`OpenGLMobject`
209: (12)             ``self``
210: (8)             """
211: (8)             for attr, value in kwargs.items():
212: (12)                 setattr(self, attr, value)
213: (8)             return self
214: (4)             def set_data(self, data):
215: (8)                 for key in data:
216: (12)                     self.data[key] = data[key].copy()
217: (8)                 return self
218: (4)             def set_uniforms(self, uniforms):
219: (8)                 for key in uniforms:
220: (12)                     self.uniforms[key] = uniforms[key] # Copy?
221: (8)                 return self
222: (4)             @property
223: (4)             def animate(self):
224: (8)                 """Used to animate the application of a method.
225: (8)                 .. warning::
226: (12)                     Passing multiple animations for the same :class:`OpenGLMobject` in
one
227: (12)                     call to :meth:`~.Scene.play` is discouraged and will most likely
228: (12)                     not work properly. Instead of writing an animation like
229: (12)                     :::
230: (16)                         self.play(my_mobject.animate.shift(RIGHT),
my_mobject.animate.rotate(PI))
231: (12)                     make use of method chaining for ``animate``, meaning::
232: (16)                         self.play(my_mobject.animate.shift(RIGHT).rotate(PI))
Keyword arguments that can be passed to :meth:`.Scene.play` can be
passed
234: (8)             directly after accessing ``.animate``, like so::
235: (12)                 self.play(my_mobject.animate(rate_func=linear).shift(RIGHT))
This is especially useful when animating simultaneous ``.animate``

calls that
237: (8)             you want to behave differently::
238: (12)                 self.play(
239: (16)                     mobject1.animate(run_time=2).rotate(PI),
240: (16)                     mobject2.animate(rate_func=there_and_back).shift(RIGHT),
241: (12)                 )
..seealso::
242: (8)                 :func:`override_animate`
Examples
-----
.. manim:: AnimateExample
243: (12)             class AnimateExample(Scene):
244: (8)                 def construct(self):
245: (16)                     s = Square()
246: (20)                     self.play(Create(s))
247: (20)                     self.play(s.animate.shift(RIGHT))
248: (20)                     self.play(s.animate.scale(2))
249: (20)                     self.play(s.animate.rotate(PI / 2))
250: (20)                     self.play(Uncreate(s))
251: (20)
252: (20)
253: (20)
254: (20)
255: (8)             .. manim:: AnimateChainExample
256: (12)             class AnimateChainExample(Scene):
257: (16)                 def construct(self):
258: (20)                     s = Square()
259: (20)                     self.play(Create(s))
260: (20)                     self.play(s.animate.shift(RIGHT).scale(2).rotate(PI / 2))
261: (20)                     self.play(Uncreate(s))
262: (8)             .. manim:: AnimateWithArgsExample
263: (12)             class AnimateWithArgsExample(Scene):
264: (16)                 def construct(self):
265: (20)                     s = Square()
266: (20)                     c = Circle()
267: (20)                     VGroup(s, c).arrange(RIGHT, buff=2)
268: (20)                     self.add(s, c)
269: (20)                     self.play(
270: (24)                         s.animate(run_time=2).rotate(PI / 2),

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
271: (24)                                     c.animate(rate_func=there_and_back).shift(RIGHT),
272: (20)                                     )
273: (8) .. warning::
274: (12)           ```.animate```
275: (13)           will interpolate the :class:`~.OpenGLObject` between its points
prior to
276: (13)           ```.animate``` and its points after applying ```.animate``` to it.
This may
277: (13)           result in unexpected behavior when attempting to interpolate
along paths,
278: (13)           or rotations.
279: (13)           If you want animations to consider the points between, consider
using
280: (13)               :class:`~.ValueTracker` with updaters instead.
281: (8) """
282: (8)             return _AnimationBuilder(self)
283: (4) @property
284: (4) def width(self):
285: (8)     """The width of the mobject.
286: (8)     Returns
287: (8)     -----
288: (8)     :class:`float`
289: (8)     Examples
290: (8)     -----
291: (8)     .. manim:: WidthExample
292: (12)         class WidthExample(Scene):
293: (16)             def construct(self):
294: (20)                 decimal = DecimalNumber().to_edge(UP)
295: (20)                 rect = Rectangle(color=BLUE)
296: (20)                 rect_copy = rect.copy().set_stroke(GRAY, opacity=0.5)
297: (20)                 decimal.add_updater(lambda d: d.set_value(rect.width))
298: (20)                 self.add(rect_copy, rect, decimal)
299: (20)                 self.play(rect.animate.set(width=7))
300: (20)                 self.wait()
301: (8)             See also
302: (8)             -----
303: (8)             :meth:`length_over_dim`
304: (8)             """
305: (8)             return self.length_over_dim(0)
306: (4) @width.setter
307: (4) def width(self, value):
308: (8)     self.rescale_to_fit(value, 0, stretch=False)
309: (4) @property
310: (4) def height(self):
311: (8)     """The height of the mobject.
312: (8)     Returns
313: (8)     -----
314: (8)     :class:`float`
315: (8)     Examples
316: (8)     -----
317: (8)     .. manim:: HeightExample
318: (12)         class HeightExample(Scene):
319: (16)             def construct(self):
320: (20)                 decimal = DecimalNumber().to_edge(UP)
321: (20)                 rect = Rectangle(color=BLUE)
322: (20)                 rect_copy = rect.copy().set_stroke(GRAY, opacity=0.5)
323: (20)                 decimal.add_updater(lambda d: d.set_value(rect.height))
324: (20)                 self.add(rect_copy, rect, decimal)
325: (20)                 self.play(rect.animate.set(height=5))
326: (20)                 self.wait()
327: (8)             See also
328: (8)             -----
329: (8)             :meth:`length_over_dim`
330: (8)             """
331: (8)             return self.length_over_dim(1)
332: (4) @height.setter
333: (4) def height(self, value):
334: (8)     self.rescale_to_fit(value, 1, stretch=False)
335: (4) @property

```

```

336: (4)             def depth(self):
337: (8)                 """The depth of the mobject.
338: (8)                 Returns
339: (8)                 -----
340: (8)                 :class:`float`
341: (8)                 See also
342: (8)                 -----
343: (8)                 :meth:`length_over_dim`
344: (8)                 """
345: (8)             return self.length_over_dim(2)
346: (4) @depth.setter
347: (4)     def depth(self, value):
348: (8)         self.rescale_to_fit(value, 2, stretch=False)
349: (4)     def resize_points(self, new_length, resize_func=resize_array):
350: (8)         if new_length != len(self.points):
351: (12)             self.points = resize_func(self.points, new_length)
352: (8)         self.refresh_bounding_box()
353: (8)         return self
354: (4)     def set_points(self, points):
355: (8)         if len(points) == len(self.points):
356: (12)             self.points[:] = points
357: (8)         elif isinstance(points, np.ndarray):
358: (12)             self.points = points.copy()
359: (8)         else:
360: (12)             self.points = np.array(points)
361: (8)         self.refresh_bounding_box()
362: (8)         return self
363: (4)     def apply_over_attr_arrays(self, func):
364: (8)         for attr in self.get_array_attrs():
365: (12)             setattr(self, attr, func(getattr(self, attr)))
366: (8)         return self
367: (4)     def append_points(self, new_points):
368: (8)         self.points = np.vstack([self.points, new_points])
369: (8)         self.refresh_bounding_box()
370: (8)         return self
371: (4)     def reverse_points(self):
372: (8)         for mob in self.get_family():
373: (12)             for key in mob.data:
374: (16)                 mob.data[key] = mob.data[key][::-1]
375: (8)         return self
376: (4)     def get_midpoint(self) -> np.ndarray:
377: (8)         """Get coordinates of the middle of the path that forms the
:class:`~OpenGLMobject` .
378: (8)             Examples
379: (8)             -----
380: (8)             .. manim:: AngleMidPoint
381: (12)                 :save_last_frame:
382: (12)                 class AngleMidPoint(Scene):
383: (16)                     def construct(self):
384: (20)                         line1 = Line(ORIGIN, 2*RIGHT)
385: (20)                         line2 = Line(ORIGIN,
2*RIGHT).rotate_about_origin(80*DEGREES)
386: (20)                         a = Angle(line1, line2, radius=1.5, other_angle=False)
387: (20)                         d = Dot(a.get_midpoint()).set_color(RED)
388: (20)                         self.add(line1, line2, a, d)
389: (20)                         self.wait()
390: (8)                         """
391: (8)                     return self.point_from_proportion(0.5)
392: (4)     def apply_points_function(
393: (8)         self,
394: (8)         func,
395: (8)         about_point=None,
396: (8)         about_edge=ORIGIN,
397: (8)         works_on_bounding_box=False,
398: (4)     ):
399: (8)         if about_point is None and about_edge is not None:
400: (12)             about_point = self.get_bounding_box_point(about_edge)
401: (8)         for mob in self.get_family():
402: (12)             arrs = []

```

```

403: (12)             if mob.has_points():
404: (16)                 arrs.append(mob.points)
405: (12)             if works_on_bounding_box:
406: (16)                 arrs.append(mob.get_bounding_box())
407: (12)             for arr in arrs:
408: (16)                 if about_point is None:
409: (20)                     arr[:] = func(arr)
410: (16)                 else:
411: (20)                     arr[:] = func(arr - about_point) + about_point
412: (8)             if not works_on_bounding_box:
413: (12)                 self.refresh_bounding_box(recurse_down=True)
414: (8)             else:
415: (12)                 for parent in self.parents:
416: (16)                     parent.refresh_bounding_box()
417: (8)             return self
418: (4)         def match_points(self, mobject):
419: (8)             """Edit points, positions, and submobjects to be identical
420: (8)             to another :class:`~.OpenGLMobject`, while keeping the style
unchanged.
421: (8)             Examples
422: (8)             -----
423: (8)             .. manim:: MatchPointsScene
424: (12)             class MatchPointsScene(Scene):
425: (16)                 def construct(self):
426: (20)                     circ = Circle(fill_color=RED, fill_opacity=0.8)
427: (20)                     square = Square(fill_color=BLUE, fill_opacity=0.2)
428: (20)                     self.add(circ)
429: (20)                     self.wait(0.5)
430: (20)                     self.play(circ.animate.match_points(square))
431: (20)                     self.wait(0.5)
432: (8)                 """
433: (8)                 self.set_points(mobject.points)
434: (4)             def clear_points(self):
435: (8)                 self.points = np.empty((0, 3))
436: (4)             def get_num_points(self):
437: (8)                 return len(self.points)
438: (4)             def get_all_points(self):
439: (8)                 if self.submobjects:
440: (12)                     return np.vstack([sm.points for sm in self.get_family()])
441: (8)                 else:
442: (12)                     return self.points
443: (4)             def has_points(self):
444: (8)                 return self.get_num_points() > 0
445: (4)             def get_bounding_box(self):
446: (8)                 if self.needs_new_bounding_box:
447: (12)                     self.bounding_box = self.compute_bounding_box()
448: (12)                     self.needs_new_bounding_box = False
449: (8)                 return self.bounding_box
450: (4)             def compute_bounding_box(self):
451: (8)                 all_points = np.vstack(
452: (12)                     [
453: (16)                         self.points,
454: (16)                         *(
455: (20)                             mob.get_bounding_box()
456: (20)                             for mob in self.get_family()[1:]
457: (20)                             if mob.has_points()
458: (16)                         ),
459: (12)                     ],
460: (8)                 )
461: (8)                 if len(all_points) == 0:
462: (12)                     return np.zeros((3, self.dim))
463: (8)                 else:
464: (12)                     mins = all_points.min(0)
465: (12)                     maxs = all_points.max(0)
466: (12)                     mids = (mins + maxs) / 2
467: (12)                     return np.array([mins, mids, maxs])
468: (4)             def refresh_bounding_box(self, recurse_down=False, recurse_up=True):
469: (8)                 for mob in self.get_family(recurse_down):
470: (12)                     mob.needs_new_bounding_box = True

```

```

471: (8)             if recurse_up:
472: (12)             for parent in self.parents:
473: (16)                 parent.refresh_bounding_box()
474: (8)             return self
475: (4)         def is_point_touching(self, point, buff=MED_SMALL_BUFF):
476: (8)             bb = self.get_bounding_box()
477: (8)             mins = bb[0] - buff
478: (8)             maxs = bb[2] + buff
479: (8)             return (point >= mins).all() and (point <= maxs).all()
480: (4)         def __getitem__(self, value):
481: (8)             if isinstance(value, slice):
482: (12)                 GroupClass = self.get_group_class()
483: (12)                 return GroupClass(*self.split().__getitem__(value))
484: (8)             return self.split().__getitem__(value)
485: (4)         def __iter__(self):
486: (8)             return iter(self.split())
487: (4)         def __len__(self):
488: (8)             return len(self.split())
489: (4)         def split(self):
490: (8)             return self.submobjects
491: (4)         def assemble_family(self):
492: (8)             sub_families = (sm.get_family() for sm in self.submobjects)
493: (8)             self.family = [self, *uniq_chain(*sub_families)]
494: (8)             self.refresh_has_updater_status()
495: (8)             self.refresh_bounding_box()
496: (8)             for parent in self.parents:
497: (12)                 parent.assemble_family()
498: (8)             return self
499: (4)         def get_family(self, recurse=True):
500: (8)             if recurse and hasattr(self, "family"):
501: (12)                 return self.family
502: (8)             else:
503: (12)                 return [self]
504: (4)         def family_members_with_points(self):
505: (8)             return [m for m in self.get_family() if m.has_points()]
506: (4)         def add(
507: (8)             self, *mobjects: OpenGLMobject, update_parent: bool = False
508: (4)         ) -> OpenGLMobject:
509: (8)             """Add mobjects as submobjects.
510: (8)             The mobjects are added to :attr:`submobjects`.
511: (8)             Subclasses of mobject may implement ``+`` and ``+=`` dunder methods.
512: (8)             Parameters
513: (8)             -----
514: (8)             mobjects
515: (12)                 The mobjects to add.
516: (8)             Returns
517: (8)             -----
518: (8)             :class:`OpenGLMobject`
519: (12)                 ``self``
520: (8)             Raises
521: (8)             -----
522: (8)             :class:`ValueError`
523: (12)                 When a mobject tries to add itself.
524: (8)             :class:`TypeError`
525: (12)                 When trying to add an object that is not an instance of
526: (8)             :class:`OpenGLMobject`.
527: (8)             Notes
528: (8)             -----
529: (8)             A mobject cannot contain itself, and it cannot contain a submobject
530: (8)             more than once. If the parent mobject is displayed, the newly-added
531: (8)             submobjects will also be displayed (i.e. they are automatically added
532: (8)             to the parent Scene).
533: (8)             See Also
534: (8)             -----
535: (8)             :meth:`remove`
536: (8)             :meth:`add_to_back`
537: (8)             Examples
538: (8)             -----
539: (8)             ::
```

```

539: (12)             >>> outer = OpenGLMobject()
540: (12)             >>> inner = OpenGLMobject()
541: (12)             >>> outer.add(inner)
542: (8)              Duplicates are not added again::
543: (12)             >>> outer.add(inner)
544: (12)             >>> len(outer.submobjects)
545: (12)             1
546: (8)              Adding an object to itself raises an error::
547: (12)             >>> outer.add(outer)
548: (12)             Traceback (most recent call last):
549: (12)             ...
550: (12)             ValueError: OpenGLMobject cannot contain self
551: (8)              """
552: (8)             if update_parent:
553: (12)               assert len(mobjects) == 1, "Can't set multiple parents."
554: (12)               mobjects[0].parent = self
555: (8)             if self in mobjects:
556: (12)               raise ValueError("OpenGLMobject cannot contain self")
557: (8)             if any(mobjects.count(elem) > 1 for elem in mobjects):
558: (12)               logger.warning(
559: (16)                 "Attempted adding some Mobject as a child more than once, "
560: (16)                 "this is not possible. Repetitions are ignored.",
561: (12)               )
562: (8)             for mobject in mobjects:
563: (12)               if not isinstance(mobject, OpenGLMobject):
564: (16)                 raise TypeError("All submobjects must be of type
565: (12)                   OpenGLMobject")
566: (16)
567: (12)
568: (16)
569: (8)
570: (8)
571: (4)             def insert(self, index: int, mobject: OpenGLMobject, update_parent: bool =
False):
572: (8)               """Inserts a mobject at a specific position into self.submobjects
573: (8)               Effectively just calls ``self.submobjects.insert(index, mobject)``,
574: (8)               where ``self.submobjects`` is a list.
575: (8)               Highly adapted from ``OpenGLMobject.add``.
576: (8)               Parameters
577: (8)               -----
578: (8)               index
579: (12)                 The index at which
580: (8)               mobject
581: (12)                 The mobject to be inserted.
582: (8)               update_parent
583: (12)                 Whether or not to set ``mobject.parent`` to ``self``.
584: (8)               """
585: (8)
586: (12)
587: (8)
588: (12)
589: (8)
590: (12)
591: (8)
592: (12)
593: (8)
594: (12)
595: (8)
596: (8)
597: (4)             if update_parent:
598: (8)               mobject.parent = self
599: (4)
600: (8)
601: (8)
602: (8)
603: (8)
604: (8)
605: (8)

```

```

606: (12)           The mobjects to remove.
607: (8)           Returns
608: (8)
609: (8)
610: (12)
611: (8)
612: (8)
613: (8)
614: (8)
615: (8)
616: (12)
617: (12)
618: (8)
619: (12)
620: (16)
621: (12)
622: (16)
623: (8)
624: (8)
625: (4)
626: (8)
627: (8)
get moved
628: (8)
629: (8)
630: (8)
631: (8)
632: (12)
633: (8)
634: (8)
635: (8)
636: (12)
637: (8)
638: (12)
639: (12)
640: (12)
641: (12)
642: (8)
643: (8)
644: (8)
645: (12)
646: (8)
647: (12)
:class:`OpenGLObject` .
648: (8)
649: (8)
650: (8)
651: (8)
652: (8)
653: (8)
654: (8)
655: (8)
656: (8)
657: (8)
658: (8)
659: (8)
660: (8)
661: (4)
662: (8)
663: (8)
664: (12)
665: (8)
666: (8)
667: (8)
668: (4)
669: (8)
670: (8)
671: (8)
672: (8)

```

The mobjects to remove.

Returns

-----

:class:`OpenGLObject`  
` ``self` `

See Also

-----

:meth:`add`

"""

if update\_parent:  
assert len(mobjects) == 1, "Can't remove multiple parents."  
mobjects[0].parent = None

for mob in mobjects:  
if mob in self.submobjects:  
self.submobjects.remove(mob)  
if self in mob.parents:  
mob.parents.remove(self)

self.assemble\_family()  
return self

def add\_to\_back(self, \*mobjects: OpenGLObject) -> OpenGLObject:  
"""Add all passed mobjects to the back of the submobjects.  
If :attr:`submobjects` already contains the given mobjects, they just  
to the back instead.

Parameters

-----

mobjects  
The mobjects to add.

Returns

-----

:class:`OpenGLObject`  
` ``self` `

.. note::  
Technically, this is done by adding (or moving) the mobjects to  
the head of :attr:`submobjects`. The head of this list is rendered  
first, which places the corresponding mobjects behind the  
subsequent list members.

Raises

-----

:class:`ValueError`  
When a mob tries to add itself.

:class:`TypeError`  
When trying to add an object that is not an instance of

Notes

-----

A mob cannot contain itself, and it cannot contain a submob more than once. If the parent mob is displayed, the newly-added submobs will also be displayed (i.e. they are automatically added to the parent Scene).

See Also

-----

:meth:`remove`  
:meth:`add`

"""

self.submobjects = list\_update(mobjects, self.submobjects)  
return self

def replace\_submob(self, index, new\_submob):  
old\_submob = self.submobjects[index]  
if self in old\_submob.parents:  
old\_submob.parents.remove(self)  
self.submobjects[index] = new\_submob  
self.assemble\_family()  
return self

def invert(self, recursive=False):  
"""Inverts the list of :attr:`submobjects`.

Parameters

-----

recursive

```

673: (12)                                If ``True``, all submobject lists of this mobject's family are
inverted.
674: (8)                                 Examples
675: (8)                                 -----
676: (8)                                 .. manim:: InvertSumobjectsExample
677: (12)                                class InvertSumobjectsExample(Scene):
678: (16)                                def construct(self):
679: (20)                                s = VGroup(*[Dot().shift(i*0.1*RIGHT) for i in
range(-20,20)])
680: (20)                                s2 = s.copy()
681: (20)                                s2.invert()
682: (20)                                s2.shift(DOWN)
683: (20)                                self.play(Write(s), Write(s2))
684: (8)                                 """
685: (8)                                if recursive:
686: (12)                                for submob in self.submobjects:
687: (16)                                submob.invert(recursive=True)
688: (8)                                list.reverse(self.submobjects)
689: (8)                                self.assemble_family()
690: (4)                                 def arrange(self, direction=RIGHT, center=True, **kwargs):
691: (8)                                 """Sorts :class:`~.OpenGLMobject` next to each other on screen.
692: (8)                                 Examples
693: (8)                                 -----
694: (8)                                 .. manim:: Example
695: (12)                                :save_last_frame:
696: (12)                                class Example(Scene):
697: (16)                                def construct(self):
698: (20)                                s1 = Square()
699: (20)                                s2 = Square()
700: (20)                                s3 = Square()
701: (20)                                s4 = Square()
702: (20)                                x = OpenGLGroup(s1, s2, s3,
s4).set_x(0).arrange(buff=1.0)
703: (20)                                self.add(x)
704: (8)                                 """
705: (8)                                for m1, m2 in zip(self.submobjects, self.submobjects[1:]):
706: (12)                                m2.next_to(m1, direction, **kwargs)
707: (8)                                if center:
708: (12)                                self.center()
709: (8)                                return self
710: (4)                                 def arrange_in_grid(
711: (8)                                self,
712: (8)                                rows: int | None = None,
713: (8)                                cols: int | None = None,
714: (8)                                buff: float | tuple[float, float] = MED_SMALL_BUFF,
715: (8)                                cell_alignment: np.ndarray = ORIGIN,
716: (8)                                row_alignments: str | None = None, # "ucd"
717: (8)                                col_alignments: str | None = None, # "lcr"
718: (8)                                row_heights: Iterable[float | None] | None = None,
719: (8)                                col_widths: Iterable[float | None] | None = None,
720: (8)                                flow_order: str = "rd",
721: (8)                                **kwargs,
722: (4)                                ) -> OpenGLMobject:
723: (8)                                 """Arrange submobjects in a grid.
724: (8)                                 Parameters
725: (8)                                 -----
726: (8)                                 rows
727: (12)                                The number of rows in the grid.
728: (8)                                 cols
729: (12)                                The number of columns in the grid.
730: (8)                                 buff
731: (12)                                The gap between grid cells. To specify a different buffer in the
horizontal and
732: (12)                                vertical directions, a tuple of two values can be given - ``((row,
col))``.
733: (8)                                 cell_alignment
734: (12)                                The way each submobject is aligned in its grid cell.
735: (8)                                 row_alignments
736: (12)                                The vertical alignment for each row (top to bottom). Accepts the

```

```

following characters: ``"u``` - up, ``"c``` - center, ``"d``` - down.
737: (12) col_alignments
738: (8) The horizontal alignment for each column (left to right). Accepts
739: (12) the following characters ``"l``` - left,
740: (12) ``"c``` - center, ``"r``` - right.
741: (8) row_heights
742: (12) Defines a list of heights for certain rows (top to bottom). If the
list contains
743: (12) ``None```, the corresponding row will fit its height automatically
based
744: (12) on the highest element in that row.
745: (8) col_widths
746: (12) Defines a list of widths for certain columns (left to right). If
the list contains ``None```, the
747: (12) corresponding column will fit its width automatically based on the
widest element in that column.
748: (8) flow_order
749: (12) The order in which subobjects fill the grid. Can be one of the
following values:
750: (12) "rd", "dr", "ld", "dl", "ru", "ur", "lu", "ul". ("rd" -> fill
rightwards then downwards)
751: (8) Returns
752: (8) -----
753: (8) OpenGLMobject
754: (12) The mobject.
755: (8) NOTES
756: (8) -----
757: (8) If only one of ``cols`` and ``rows`` is set implicitly, the other one
will be chosen big
758: (8) enough to fit all subobjects. If neither is set, they will be chosen
to be about the same,
759: (8) tending towards ``cols`` > ``rows`` (simply because videos are wider
than they are high).
760: (8) If both ``cell_alignment`` and ``row_alignments`` / ``col_alignments``
are
761: (8) defined, the latter has higher priority.
762: (8) Raises
763: (8) -----
764: (8) ValueError
765: (12) If ``rows`` and ``cols`` are too small to fit all subobjects.
766: (8) ValueError
767: (12) If :code:`cols`, :code:`col_alignments` and :code:`col_widths` or
768: (12) :code:`row_alignments` and :code:`row_heights` have mismatching
:scode:`rows`,
sizes.
769: (8) Examples
770: (8) -----
771: (8) .. manim:: ExampleBoxes
772: (12) :save_last_frame:
773: (12) class ExampleBoxes(Scene):
774: (16)     def construct(self):
775: (20)         boxes=VGroup(*[Square() for s in range(0,6)])
776: (20)         boxes.arrange_in_grid(rows=2, buff=0.1)
777: (20)         self.add(boxes)
778: (8) .. manim:: ArrangeInGrid
779: (12) :save_last_frame:
780: (12) class ArrangeInGrid(Scene):
781: (16)     def construct(self):
782: (20)         np.random.seed(3)
783: (20)         boxes = VGroup([
784: (24)             Rectangle(WHITE, np.random.random()+.5,
np.random.random()+.5).add(Text(str(i+1)).scale(0.5))
785: (24)             for i in range(22)
786: (20)         ])
787: (20)         self.add(boxes)
788: (20)         boxes.arrange_in_grid(
789: (24)             buff=(0.25,0.5),
790: (24)             col_alignments="lcccn",

```

```

791: (24)                                row_alignments="uccd",
792: (24)                                col_widths=[2, *[None]*4, 2],
793: (24)                                flow_order="dr"
794: (20)                                )
795: (8)      """
796: (8)      from manim.mobject.geometry.line import Line
797: (8)      mobs = self.submobjects.copy()
798: (8)      start_pos = self.get_center()
799: (8)      def init_size(num, alignments, sizes):
800: (12)          if num is not None:
801: (16)              return num
802: (12)          if alignments is not None:
803: (16)              return len(alignments)
804: (12)          if sizes is not None:
805: (16)              return len(sizes)
806: (8)          cols = init_size(cols, col_alignments, col_widths)
807: (8)          rows = init_size(rows, row_alignments, row_heights)
808: (8)          if rows is None and cols is None:
809: (12)              cols = ceil(np.sqrt(len(mobs)))
810: (8)          if rows is None:
811: (12)              rows = ceil(len(mobs) / cols)
812: (8)          if cols is None:
813: (12)              cols = ceil(len(mobs) / rows)
814: (8)          if rows * cols < len(mobs):
815: (12)              raise ValueError("Too few rows and columns to fit all
submobjetc.")
816: (8)
817: (12)      if isinstance(buff, tuple):
818: (12)          buff_x = buff[0]
819: (8)          buff_y = buff[1]
820: (12)      else:
821: (8)          buff_x = buff_y = buff
822: (12)      def init_alignments(alignments, num, mapping, name, dir):
823: (16)          if alignments is None:
824: (12)              return [cell_alignment * dir] * num
825: (16)          if len(alignments) != num:
826: (12)              raise ValueError(f"{name}_alignments has a mismatching size.")
827: (12)          alignments = list(alignments)
828: (16)          for i in range(num):
829: (12)              alignments[i] = mapping[alignments[i]]
830: (8)          return alignments
831: (12)      row_alignments = init_alignments(
832: (12)          alignments,
833: (12)          rows,
834: (12)          {"u": UP, "c": ORIGIN, "d": DOWN},
835: (12)          "row",
836: (8)          RIGHT,
837: (8)
838: (12)      col_alignments = init_alignments(
839: (12)          alignments,
840: (12)          cols,
841: (12)          {"l": LEFT, "c": ORIGIN, "r": RIGHT},
842: (12)          "col",
843: (8)          UP,
844: (8)      mapper = {
845: (12)          "dr": lambda r, c: (rows - r - 1) + c * rows,
846: (12)          "dl": lambda r, c: (rows - r - 1) + (cols - c - 1) * rows,
847: (12)          "ur": lambda r, c: r + c * rows,
848: (12)          "ul": lambda r, c: r + (cols - c - 1) * rows,
849: (12)          "rd": lambda r, c: (rows - r - 1) * cols + c,
850: (12)          "ld": lambda r, c: (rows - r - 1) * cols + (cols - c - 1),
851: (12)          "ru": lambda r, c: r * cols + c,
852: (12)          "lu": lambda r, c: r * cols + (cols - c - 1),
853: (8)
854: (8)      if flow_order not in mapper:
855: (12)          raise ValueError(
856: (16)              'flow_order must be one of the following values: "dr", "rd",
857: (12)              "ld", "dl", "ru", "ur", "lu", "ul".')

```

```

858: (8)             flow_order = mapper[flow_order]
859: (8)             def reverse(maybe_list):
860: (12)                 if maybe_list is not None:
861: (16)                     maybe_list = list(maybe_list)
862: (16)                     maybe_list.reverse()
863: (16)                     return maybe_list
864: (8)             row_alignments = reverse(row_alignments)
865: (8)             row_heights = reverse(row_heights)
866: (8)             placeholder = OpenGLMobject()
867: (8)             mobs.extend([placeholder] * (rows * cols - len(mobs)))
868: (8)             grid = [[mobs[flow_order(r, c)] for c in range(cols)] for r in
range(rows)]
869: (8)
870: (12)             measured_heights = [
871: (8)                 max(grid[r][c].height for c in range(cols)) for r in range(rows)
872: ]
873: (8)             measured_widths = [
874: (12)                 max(grid[r][c].width for r in range(rows)) for c in range(cols)
875: ]
876: (8)             def init_sizes(sizes, num, measures, name):
877: (12)                 if sizes is None:
878: (16)                     sizes = [None] * num
879: (12)                 if len(sizes) != num:
880: (16)                     raise ValueError(f"{name} has a mismatching size.")
881: (12)                 return [
882: (16)                     sizes[i] if sizes[i] is not None else measures[i] for i in
range(num)
883: (12)             ]
884: (8)             heights = init_sizes(row_heights, rows, measured_heights,
885: (8)             widths = init_sizes(col_widths, cols, measured_widths, "col_widths")
886: (8)             x, y = 0, 0
887: (12)             for r in range(rows):
888: (12)                 x = 0
889: (16)                 for c in range(cols):
890: (20)                     if grid[r][c] is not placeholder:
891: (20)                         alignment = row_alignments[r] + col_alignments[c]
892: (24)                         line = Line(
893: (24)                             x * RIGHT + y * UP,
894: (20)                             (x + widths[c]) * RIGHT + (y + heights[r]) * UP,
895: (20)                         )
896: (16)                         grid[r][c].move_to(line, alignment)
897: (12)                         x += widths[c] + buff_x
898: (8)                         y += heights[r] + buff_y
899: (8)                         self.move_to(start_pos)
900: (4)             return self
901: (8)             def get_grid(self, n_rows, n_cols, height=None, **kwargs):
902: (8)                 """
903: (8)                     Returns a new mobject containing multiple copies of this one
904: (8)                     arranged in a grid
905: (8)                 """
906: (8)                 grid = self.duplicate(n_rows * n_cols)
907: (8)                 grid.arrange_in_grid(n_rows, n_cols, **kwargs)
908: (12)                 if height is not None:
909: (8)                     grid.set_height(height)
910: (4)                 return grid
911: (8)             def duplicate(self, n: int):
912: (8)                 """
913: (8)                     Returns an :class:`~.OpenGLVGGroup` containing ``n`` copies of the
914: (8)                     mobject.
915: (8)                 """
916: (12)                 return self.get_group_class()(*[self.copy() for _ in range(n)])
917: (8)             def sort(self, point_to_num_func=lambda p: p[0], submob_func=None):
918: (8)                 """
919: (8)                     Sorts the list of :attr:`subobjects` by a function defined by
920: (8)                     ``submob_func``.
921: (8)                 """
922: (8)                 if submob_func is not None:
923: (12)                     self.subobjects.sort(key=submob_func)
924: (8)                 else:
925: (12)                     self.subobjects.sort(key=lambda m:
point_to_num_func(m.get_center())))
926: (8)                 return self
927: (4)             def shuffle(self, recurse=False):

```

```

921: (8)                                     """Shuffles the order of :attr:`submobjects`  

922: (8)  

923: (8)  

924: (8)  

925: (12)  

926: (16)  

927: (20)  

range(-20,20)])  

928: (20)  

929: (20)  

930: (20)  

931: (20)  

932: (8)  

933: (8)  

934: (12)  

935: (16)  

936: (8)  

937: (8)  

938: (8)  

939: (4)  

940: (8)  

941: (8)  

942: (8)  

943: (8)  

944: (12) inverted.  

945: (8)  

946: (8)  

947: (8)  

948: (12)  

949: (16)  

950: (20)  

range(-20,20))  

951: (20)  

952: (20)  

953: (20)  

954: (20)  

955: (8)  

956: (8)  

957: (12)  

958: (16)  

959: (8)  

960: (4)  

961: (8) including all  

962: (8)  

963: (8)  

964: (8)  

965: (8)  

966: (12)  

967: (8)  

968: (8)  

969: (8)  

970: (12) Note  

971: (8)  

972: (8)  

973: (8) The clone is initially not visible in the Scene, even if the original  

was.  

974: (8)  

975: (8)  

976: (12)  

977: (8)  

978: (8)  

979: (8)  

980: (8)  

981: (8)  

982: (8)  

983: (12)  

984: (8)

```

Examples

```

921: (8)                                     .. manim:: ShuffleSubmobjectsExample
922: (8)   class ShuffleSubmobjectsExample(Scene):
923: (8)   def construct(self):
924: (12)   s= OpenGLGroup(*[Dot().shift(i*0.1*RIGHT) for i in
925: (16)   range(-20,20)])
926: (20)   s2= s.copy()
927: (20)   s2.shuffle()
928: (20)   s2.shift(DOWN)
929: (20)   self.play(Write(s), Write(s2))
930: (20)
931: (20)
932: (8)
933: (8)
934: (12)
935: (16)
936: (8)
937: (8)
938: (8)
939: (4)
940: (8)
941: (8)
942: (8)
943: (8)
944: (12) inverted.
945: (8)
946: (8)
947: (8)
948: (12)
949: (16)
950: (20)
range(-20,20))
```

Parameters

```

921: (8)   if recurse:
922: (8)   for submob in self.submobjects:
923: (8)   submob.shuffle(recurse=True)
924: (12)   random.shuffle(self.submobjects)
925: (16)   self.assemble_family()
926: (20)   return self
927: (20)
928: (20)
929: (20)
930: (20)
931: (20)
932: (8)
933: (8)
934: (12)
935: (16)
936: (8)
937: (8)
938: (8)
939: (4)
940: (8)
941: (8)
942: (8)
943: (8)
944: (12) inverted.
945: (8)
946: (8)
947: (8)
948: (12)
949: (16)
950: (20)
range(-20,20))
```

recursive

```

921: (8)   If ``True``, all submobject lists of this mobject's family are
922: (8)   inverted.
```

Examples

```

921: (8)   .. manim:: InvertSumobjectsExample
922: (8)   class InvertSumobjectsExample(Scene):
923: (8)   def construct(self):
924: (12)   s = VGroup(*[Dot().shift(i*0.1*RIGHT) for i in
925: (16)   range(-20,20)])
926: (20)   s2 = s.copy()
927: (20)   s2.invert()
928: (20)   s2.shift(DOWN)
929: (20)   self.play(Write(s), Write(s2))
930: (20)
931: (20)
932: (8)
933: (8)
934: (12)
935: (16)
936: (8)
937: (8)
938: (8)
939: (4)
940: (8)
941: (8)
942: (8)
943: (8)
944: (12) including all
945: (8)
946: (8)
947: (8)
948: (12)
949: (16)
950: (20)
range(-20,20))
```

Parameters

```

921: (8)   if recursive:
922: (8)   for submob in self.submobjects:
923: (8)   submob.invert(recursive=True)
924: (12)   list.reverse(self.submobjects)
925: (16)
926: (8)
927: (4)   def copy(self, shallow: bool = False):
928: (8)   """Create and return an identical copy of the :class:`OpenGLMobject`  

929: (12)   :attr:`submobjects`.  

930: (16)   Returns
931: (20)   :class:`OpenGLMobject`  

932: (20)   The copy.
933: (20)   Parameters
934: (20)   shallow
935: (20)   Controls whether a shallow copy is returned.
936: (20)   Note
937: (20)   The clone is initially not visible in the Scene, even if the original
938: (20)   was.
939: (20)
940: (20)
941: (20)
942: (20)
943: (20)
944: (20)
945: (20)
946: (20)
947: (20)
948: (20)
949: (20)
950: (20)
range(-20,20))
```

shallow

```

921: (8)   Note
922: (8)   The clone is initially not visible in the Scene, even if the original
923: (8)   was.
924: (8)
925: (8)
926: (12)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
range(-20,20))
```

Note

```

921: (8)   The clone is initially not visible in the Scene, even if the original
922: (8)   was.
923: (8)
924: (8)
925: (12)
926: (8)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
range(-20,20))
```

The clone is initially not visible in the Scene, even if the original was.

Parameters

```

921: (8)   if not shallow:
922: (8)   return self.deepcopy()
923: (8)
924: (12)   parents = self.parents
925: (16)   self.parents = []
926: (8)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
range(-20,20))
```

parents

```

921: (8)   parents = self.parents
922: (8)   self.parents = []
923: (8)
924: (12)   copy_mobject = copy.copy(self)
925: (16)   self.parents = parents
926: (8)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
range(-20,20))
```

copy\_mobject

```

921: (8)   copy_mobject = copy.copy(self)
922: (8)   self.parents = parents
923: (8)
924: (12)   copy_mobject.data = dict(self.data)
925: (16)   self.parents = parents
926: (8)
927: (8)
928: (8)
929: (8)
930: (8)
931: (8)
932: (8)
933: (8)
934: (8)
935: (8)
936: (8)
937: (8)
938: (8)
939: (8)
940: (8)
941: (8)
942: (8)
943: (8)
944: (8)
945: (8)
946: (8)
947: (8)
948: (8)
949: (8)
950: (8)
range(-20,20))
```

copy\_mobject.data

```

921: (8)   copy_mobject.data[key] = self.data[key].copy()
922: (8)
923: (12)   copy_mobject.uniforms = dict(self.uniforms)
924: (16)
925: (8)
```

```

985: (8)             copy_mobject.submobjects = []
986: (8)             copy_mobject.add(*(sm.copy() for sm in self.submobjects))
987: (8)             copy_mobject.match_updaters(self)
988: (8)             copy_mobject.needs_new_bounding_box = self.needs_new_bounding_box
989: (8)             family = self.get_family()
990: (8)             for attr, value in list(self.__dict__.items()):
991: (12)                 if (
992: (16)                     isinstance(value, OpenGLMobject)
993: (16)                     and value in family
994: (16)                     and value is not self
995: (12)                 ):
996: (16)                     setattr(copy_mobject, attr, value.copy())
997: (12)             if isinstance(value, np.ndarray):
998: (16)                 setattr(copy_mobject, attr, value.copy())
999: (8)             return copy_mobject
1000: (4)             def deepcopy(self):
1001: (8)                 parents = self.parents
1002: (8)                 self.parents = []
1003: (8)                 result = copy.deepcopy(self)
1004: (8)                 self.parents = parents
1005: (8)                 return result
1006: (4)             def generate_target(self, use_deepcopy: bool = False):
1007: (8)                 self.target = None # Prevent exponential explosion
1008: (8)                 if use_deepcopy:
1009: (12)                     self.target = self.deepcopy()
1010: (8)                 else:
1011: (12)                     self.target = self.copy()
1012: (8)                 return self.target
1013: (4)             def save_state(self, use_deepcopy: bool = False):
1014: (8)                 """Save the current state (position, color & size). Can be restored
with :meth:`~.OpenGLMobject.restore`."""
1015: (8)                 if hasattr(self, "saved_state"):
1016: (12)                     self.saved_state = None
1017: (8)                 if use_deepcopy:
1018: (12)                     self.saved_state = self.deepcopy()
1019: (8)                 else:
1020: (12)                     self.saved_state = self.copy()
1021: (8)                 return self
1022: (4)             def restore(self):
1023: (8)                 """Restores the state that was previously saved with
:meth:`~.OpenGLMobject.save_state`."""
1024: (8)                 if not hasattr(self, "saved_state") or self.saved_state is None:
1025: (12)                     raise Exception("Trying to restore without having saved")
1026: (8)                 self.become(self.saved_state)
1027: (8)                 return self
1028: (4)             def init_updaters(self):
1029: (8)                 self.time_based_updaters = []
1030: (8)                 self.non_time_updaters = []
1031: (8)                 self.has_updaters = False
1032: (8)                 self.updating_suspended = False
1033: (4)             def update(self, dt=0, recurse=True):
1034: (8)                 if not self.has_updaters or self.updating_suspended:
1035: (12)                     return self
1036: (8)                 for updater in self.time_based_updaters:
1037: (12)                     updater(self, dt)
1038: (8)                 for updater in self.non_time_updaters:
1039: (12)                     updater(self)
1040: (8)                 if recurse:
1041: (12)                     for submob in self.submobjects:
1042: (16)                         submob.update(dt, recurse)
1043: (8)                     return self
1044: (4)             def get_time_based_updaters(self):
1045: (8)                 return self.time_based_updaters
1046: (4)             def has_time_based_updater(self):
1047: (8)                 return len(self.time_based_updaters) > 0
1048: (4)             def get_updaters(self):
1049: (8)                 return self.time_based_updaters + self.non_time_updaters
1050: (4)             def get_family_updaters(self):
1051: (8)                 return list(it.chain(*sm.get_updaters() for sm in

```

```

self.get_family()))
1052: (4)     def add_updater(self, update_function, index=None, call_updater=False):
1053: (8)         if "dt" in inspect.signature(update_function).parameters:
1054: (12)             updater_list = self.time_based_updaters
1055: (8)         else:
1056: (12)             updater_list = self.non_time_updaters
1057: (8)         if index is None:
1058: (12)             updater_list.append(update_function)
1059: (8)         else:
1060: (12)             updater_list.insert(index, update_function)
1061: (8)         self.refresh_has_updater_status()
1062: (8)         if call_updater:
1063: (12)             self.update()
1064: (8)     return self
1065: (4)     def remove_updater(self, update_function):
1066: (8)         for updater_list in [self.time_based_updaters,
self.non_time_updaters]:
1067: (12)             while update_function in updater_list:
1068: (16)                 updater_list.remove(update_function)
1069: (8)             self.refresh_has_updater_status()
1070: (8)         return self
1071: (4)     def clear_updaters(self, recurse=True):
1072: (8)         self.time_based_updaters = []
1073: (8)         self.non_time_updaters = []
1074: (8)         self.refresh_has_updater_status()
1075: (8)         if recurse:
1076: (12)             for submob in self.submobjects:
1077: (16)                 submob.clear_updaters()
1078: (8)         return self
1079: (4)     def match_updaters(self, mobject):
1080: (8)         self.clear_updaters()
1081: (8)         for updater in mobject.get_updaters():
1082: (12)             self.add_updater(updater)
1083: (8)         return self
1084: (4)     def suspend_updating(self, recurse=True):
1085: (8)         self.updating_suspended = True
1086: (8)         if recurse:
1087: (12)             for submob in self.submobjects:
1088: (16)                 submob.suspend_updating(recurse)
1089: (8)         return self
1090: (4)     def resume_updating(self, recurse=True, call_updater=True):
1091: (8)         self.updating_suspended = False
1092: (8)         if recurse:
1093: (12)             for submob in self.submobjects:
1094: (16)                 submob.resume_updating(recurse)
1095: (8)         for parent in self.parents:
1096: (12)             parent.resume_updating(recurse=False, call_updater=False)
1097: (8)         if call_updater:
1098: (12)             self.update(dt=0, recurse=recurse)
1099: (8)         return self
1100: (4)     def refresh_has_updater_status(self):
1101: (8)         self.has_updaters = any(mob.get_updaters() for mob in
self.get_family())
1102: (8)         return self
1103: (4)     def shift(self, vector):
1104: (8)         self.apply_points_function(
1105: (12)             lambda points: points + vector,
1106: (12)             about_edge=None,
1107: (12)             works_on_bounding_box=True,
1108: (8)
1109: (8)
1110: (4)         )
1111: (8)
1112: (8)
1113: (8)
1114: (8)
1115: (8)
1116: (4)
1117: (8)         )
1118: (4)     ) -> OpenGLMobject:
1119: (8)         r"""Scale the size by a factor.

```

```

1118: (8)                               Default behavior is to scale about the mobject.
1119: (8)                               The argument about_edge can be a vector, indicating which side of
1120: (8)                               the mobject to scale about, e.g., mob.scale(about_edge = RIGHT)
1121: (8)                               scales about mob.get_right().
1122: (8)                               Otherwise, if about_point is given a value, scaling is done with
1123: (8)                               respect to that point.
1124: (8)                               Parameters
1125: (8)
1126: (8)
1127: (12)                             scale_factor
1128: (12)                             the mobject
1129: (12)                             Furthermore,
1130: (8)                               if :math:`\alpha < 0`, the mobject is also flipped.
1131: (12)                             kwargs
1132: (12)                             Additional keyword arguments passed to
1133: (8)                               :meth:`apply_points_function_about_point`.
1134: (8)
1135: (8)
1136: (12)                             Returns
1137: (8)
1138: (8)
1139: (8)
1140: (12)                             OpenGLMobject
1141: (12)                             The scaled mobject.
1142: (16)                             Examples
1143: (20)
1144: (20)
1145: (20)
1146: (20)
1147: (20)
1148: (20)
1149: (8)                               .. manim:: MobjectScaleExample
1150: (8)                               :save_last_frame:
1151: (8)                               class MobjectScaleExample(Scene):
1152: (8)                                   def construct(self):
1153: (8)                                       f1 = Text("F")
1154: (12)                                       f2 = Text("F").scale(2)
1155: (12)                                       f3 = Text("F").scale(0.5)
1156: (12)                                       f4 = Text("F").scale(-1)
1157: (12)                                       vgroup = VGroup(f1, f2, f3, f4).arrange(6 * RIGHT)
1158: (12)                                       self.add(vgroup)
1159: (8)                               See also
1160: (8)
1161: (4)                               -----
1162: (8)                               :meth:`move_to`
1163: (12)
1164: (12)
1165: (8)
1166: (8)
1167: (4)
1168: (8)
1169: (4)
1170: (8)
1171: (8)
1172: (8)
1173: (8)
1174: (8)
1175: (4)
1176: (8)
1177: (8)
1178: (8)
1179: (12)
1180: (12)
1181: (12)
1182: (8)
1183: (8)
1184: (4)                               """Rotates the :class:`~.OpenGLMobject` about a certain point."""
1185: (8)                               rot_matrix_T = rotation_matrix_transpose(angle, axis)
1186: (8)                               self.apply_points_function(
1187: (12)                                   lambda points: np.dot(points, rot_matrix_T),
1188: (12)                                   about_point=about_point,
1189: (12)                                   **kwargs,
1190: (8)                               )
1191: (8)                               return self
1192: (4)                               def flip(self, axis=UP, **kwargs):

```

```

1185: (8)          """Flips/Mirrors an mobject about its center.
1186: (8)
1187: (8)
1188: (8)
1189: (12)
1190: (12)
1191: (16)
1192: (20)
1193: (20)
1194: (20)
1195: (20)
1196: (8)
1197: (8)
1198: (4)
1199: (8)
1200: (12)
1201: (8)
1202: (12)
1203: (8)
1204: (8)
1205: (4)
1206: (8)
1207: (8)
1208: (4)
1209: (8)
1210: (12)
1211: (8)
1212: (4)
1213: (8)
1214: (12)
1215: (8)
1216: (8)
1217: (8)
1218: (8)
1219: (12)
1220: (8)
1221: (8)
1222: (4)
1223: (8)
1224: (8)
respectively.
1225: (8)
1226: (8)
1227: (8)
1228: (12)
1229: (16)
1230: (20)
1231: (20)
1232: (20)
1233: (24)
1234: (20)
1235: (20)
1236: (20)
1237: (24)
x.become(circ_ref.copy().apply_complex_function(
1238: (28)          lambda x: np.exp(x+t.get_value()*1j)
1239: (24)          ).set_color(BLUE)
1240: (20)          )
1241: (20)          self.add(circ_ref)
1242: (20)          self.play(TransformFromCopy(circ_ref, circ))
1243: (20)          self.play(t.animate.set_value(TAU), run_time=3)
1244: (8)
1245: (8)
1246: (12)
1247: (12)
1248: (12)
1249: (8)
1250: (4)
1251: (8)
def R3_func(point):
    x, y, z = point
    xy_complex = function(complex(x, y))
    return [xy_complex.real, xy_complex.imag, z]
return self.apply_function(R3_func)
def hierarchical_matrix(self):
    if self.parent is None:
        .. manim:: FlipExample
            :save_last_frame:
            class FlipExample(Scene):
                def construct(self):
                    s= Line(LEFT, RIGHT+UP).shift(4*LEFT)
                    self.add(s)
                    s2= s.copy().flip()
                    self.add(s2)
                """
                return self.rotate(TAU / 2, axis, **kwargs)
def apply_function(self, function, **kwargs):
    if len(kwargs) == 0:
        kwargs["about_point"] = ORIGIN
    self.apply_points_function(
        lambda points: np.array([function(p) for p in points]), **kwargs
    )
    return self
def apply_function_to_position(self, function):
    self.move_to(function(self.get_center()))
    return self
def apply_function_to_submobject_positions(self, function):
    for submob in self.submobjects:
        submob.apply_function_to_position(function)
    return self
def apply_matrix(self, matrix, **kwargs):
    if ("about_point" not in kwargs) and ("about_edge" not in kwargs):
        kwargs["about_point"] = ORIGIN
    full_matrix = np.identity(self.dim)
    matrix = np.array(matrix)
    full_matrix[: matrix.shape[0], : matrix.shape[1]] = matrix
    self.apply_points_function(
        lambda points: np.dot(points, full_matrix.T), **kwargs
    )
    return self
def apply_complex_function(self, function, **kwargs):
    """Applies a complex function to a :class:`OpenGLMobject`.
The x and y coordinates correspond to the real and imaginary parts
Example
-----
.. manim:: ApplyFuncExample
    class ApplyFuncExample(Scene):
        def construct(self):
            circ = Circle().scale(1.5)
            circ_ref = circ.copy()
            circ.apply_complex_function(
                lambda x: np.exp(x*1j)
            )
            t = ValueTracker(0)
            circ.add_updater(
                lambda x:
x.become(circ_ref.copy().apply_complex_function(
1238: (28)          lambda x: np.exp(x+t.get_value()*1j)
1239: (24)          ).set_color(BLUE)
1240: (20)          )
1241: (20)          self.add(circ_ref)
1242: (20)          self.play(TransformFromCopy(circ_ref, circ))
1243: (20)          self.play(t.animate.set_value(TAU), run_time=3)
1244: (8)
1245: (8)
1246: (12)
1247: (12)
1248: (12)
1249: (8)
1250: (4)
1251: (8)
def R3_func(point):
    x, y, z = point
    xy_complex = function(complex(x, y))
    return [xy_complex.real, xy_complex.imag, z]
return self.apply_function(R3_func)
def hierarchical_matrix(self):
    if self.parent is None:
```

```

1252: (12)             return self.model_matrix
1253: (8)             model_matrices = [self.model_matrix]
1254: (8)             current_object = self
1255: (8)             while current_object.parent is not None:
1256: (12)                 model_matrices.append(current_object.parent.model_matrix)
1257: (12)                 current_object = current_object.parent
1258: (8)             return np.linalg.multi_dot(list(reversed(model_matrices)))
1259: (4)             def wag(self, direction=RIGHT, axis=DOWN, wag_factor=1.0):
1260: (8)                 for mob in self.family_members_with_points():
1261: (12)                     alphas = np.dot(mob.points, np.transpose(axis))
1262: (12)                     alphas -= min(alphas)
1263: (12)                     alphas /= max(alphas)
1264: (12)                     alphas = alphas**wag_factor
1265: (12)                     mob.set_points(
1266: (16)                         mob.points
1267: (16)                         + np.dot(
1268: (20)                             alphas.reshape((len(alphas), 1)),
1269: (20)                             np.array(direction).reshape((1, mob.dim)),
1270: (16)                         ),
1271: (12)                     )
1272: (8)             return self
1273: (4)             def center(self):
1274: (8)                 """Moves the mobject to the center of the Scene."""
1275: (8)                 self.shift(-self.get_center())
1276: (8)                 return self
1277: (4)             def align_on_border(self, direction, buff=DEFAULT_MOBJECT_TO_EDGE_BUFFER):
1278: (8)                 """
1279: (8)                     Direction just needs to be a vector pointing towards side or
1280: (8)                     corner in the 2d plane.
1281: (8)
1282: (8)                     target_point = np.sign(direction) * (
1283: (12)                         config["frame_x_radius"],
1284: (12)                         config["frame_y_radius"],
1285: (12)                         0,
1286: (8)                     )
1287: (8)                     point_to_align = self.get_bounding_box_point(direction)
1288: (8)                     shift_val = target_point - point_to_align - buff * np.array(direction)
1289: (8)                     shift_val = shift_val * abs(np.sign(direction))
1290: (8)                     self.shift(shift_val)
1291: (8)                     return self
1292: (4)             def to_corner(self, corner=LEFT + DOWN,
buff=DEFAULT_MOBJECT_TO_EDGE_BUFFER):
1293: (8)                 return self.align_on_border(corner, buff)
1294: (4)             def to_edge(self, edge=LEFT, buff=DEFAULT_MOBJECT_TO_EDGE_BUFFER):
1295: (8)                 return self.align_on_border(edge, buff)
1296: (4)             def next_to(
1297: (8)                 self,
1298: (8)                 mobject_or_point,
1299: (8)                 direction=RIGHT,
1300: (8)                 buff=DEFAULT_MOBJECT_TO_MOBJECT_BUFFER,
1301: (8)                 aligned_edge=ORIGIN,
1302: (8)                 submobject_to_align=None,
1303: (8)                 index_of_submobject_to_align=None,
1304: (8)                 coor_mask=np.array([1, 1, 1]),
1305: (4)             ):
1306: (8)                 """Move this :class:`~OpenGLMobject` next to another's
:class:`~OpenGLMobject` or coordinate.
1307: (8)                 Examples
1308: (8)                 -----
1309: (8)                 .. manim:: GeometricShapes
1310: (12)                     :save_last_frame:
1311: (12)                     class GeometricShapes(Scene):
1312: (16)                         def construct(self):
1313: (20)                             d = Dot()
1314: (20)                             c = Circle()
1315: (20)                             s = Square()
1316: (20)                             t = Triangle()
1317: (20)                             d.next_to(c, RIGHT)
1318: (20)                             s.next_to(c, LEFT)

```

```

1319: (20)                     t.next_to(c, DOWN)
1320: (20)                     self.add(d, c, s, t)
1321: (8)                     """
1322: (8)             if isinstance(mobject_or_point, OpenGLObject):
1323: (12)                 mob = mobject_or_point
1324: (12)                 if index_of_submobject_to_align is not None:
1325: (16)                     target_aligner = mob[index_of_submobject_to_align]
1326: (12)                 else:
1327: (16)                     target_aligner = mob
1328: (12)                     target_point = target_aligner.get_bounding_box_point(
1329: (16)                         aligned_edge + direction,
1330: (12)                     )
1331: (8)             else:
1332: (12)                 target_point = mobject_or_point
1333: (8)             if submobject_to_align is not None:
1334: (12)                 aligner = submobject_to_align
1335: (8)             elif index_of_submobject_to_align is not None:
1336: (12)                 aligner = self[index_of_submobject_to_align]
1337: (8)             else:
1338: (12)                 aligner = self
1339: (8)             point_to_align = aligner.get_bounding_box_point(aligned_edge -
1340: (8)             direction)
1341: (8)             self.shift((target_point - point_to_align + buff * direction) *
1342: (4)             return self
1343: (8)         def shift_onto_screen(self, **kwargs):
1344: (8)             space_lengths = [config["frame_x_radius"], config["frame_y_radius"]]
1345: (12)             for vect in UP, DOWN, LEFT, RIGHT:
1346: (12)                 dim = np.argmax(np.abs(vect))
1347: (12)                 buff = kwargs.get("buff", DEFAULT_MOBJECT_TO_EDGE_BUFFER)
1348: (12)                 max_val = space_lengths[dim] - buff
1349: (12)                 edge_center = self.get_edge_center(vect)
1350: (16)                 if np.dot(edge_center, vect) > max_val:
1351: (8)                     self.to_edge(vect, **kwargs)
1352: (4)             return self
1353: (8)         def is_off_screen(self):
1354: (12)             if self.get_left()[0] > config.frame_x_radius:
1355: (8)                 return True
1356: (12)             if self.get_right()[0] < config.frame_x_radius:
1357: (8)                 return True
1358: (12)             if self.get_bottom()[1] > config.frame_y_radius:
1359: (8)                 return True
1360: (12)             if self.get_top()[1] < -config.frame_y_radius:
1361: (8)                 return True
1362: (8)             return False
1363: (4)         def stretch_about_point(self, factor, dim, point):
1364: (8)             return self.stretch(factor, dim, about_point=point)
1365: (4)         def rescale_to_fit(self, length, dim, stretch=False, **kwargs):
1366: (8)             old_length = self.length_over_dim(dim)
1367: (12)             if old_length == 0:
1368: (8)                 return self
1369: (12)             if stretch:
1370: (8)                 self.stretch(length / old_length, dim, **kwargs)
1371: (12)             else:
1372: (8)                 self.scale(length / old_length, **kwargs)
1373: (4)             return self
1374: (8)         def stretch_to_fit_width(self, width, **kwargs):
1375: (8)             """Stretches the :class:`~OpenGLObject` to fit a width, not keeping
height/depth proportional.
1376: (8)             Returns
1377: (8)             -----
1378: (12)             :class:`OpenGLObject`
1379: (8)             ``self``
1380: (8)             Examples
1381: (8)             -----
1382: (12)             ::

1383: (12)                 >>> from manim import *
1384: (12)                 >>> sq = Square()
1385: (12)                 >>> sq.height

```

```

1385: (12)                      2.0
1386: (12)                      >>> sq.stretch_to_fit_width(5)
1387: (12)                      Square
1388: (12)                      >>> sq.width
1389: (12)                      5.0
1390: (12)                      >>> sq.height
1391: (12)                      2.0
1392: (8)                      """
1393: (8)                      return self.rescale_to_fit(width, 0, stretch=True, **kwargs)
1394: (4)                      def stretch_to_fit_height(self, height, **kwargs):
1395: (8)                          """Stretches the :class:`~.OpenGLMobject` to fit a height, not keeping
width/height proportional."""
1396: (8)                      return self.rescale_to_fit(height, 1, stretch=True, **kwargs)
1397: (4)                      def stretch_to_fit_depth(self, depth, **kwargs):
1398: (8)                          """Stretches the :class:`~.OpenGLMobject` to fit a depth, not keeping
width/height proportional."""
1399: (8)                      return self.rescale_to_fit(depth, 1, stretch=True, **kwargs)
1400: (4)                      def set_width(self, width, stretch=False, **kwargs):
1401: (8)                          """Scales the :class:`~.OpenGLMobject` to fit a width while keeping
height/depth proportional.
1402: (8)                      Returns
1403: (8)                      -----
1404: (8)                      :class:`OpenGLMobject`  

1405: (12)                      ``self``
1406: (8)                      Examples
1407: (8)                      -----
1408: (8)                      ::  

1409: (12)                      >>> from manim import *
1410: (12)                      >>> sq = Square()
1411: (12)                      >>> sq.height
1412: (12)                      2.0
1413: (12)                      >>> sq.scale_to_fit_width(5)
1414: (12)                      Square
1415: (12)                      >>> sq.width
1416: (12)                      5.0
1417: (12)                      >>> sq.height
1418: (12)                      5.0
1419: (8)                      """
1420: (8)                      return self.rescale_to_fit(width, 0, stretch=stretch, **kwargs)
1421: (4)                      scale_to_fit_width = set_width
1422: (4)                      def set_height(self, height, stretch=False, **kwargs):
1423: (8)                          """Scales the :class:`~.OpenGLMobject` to fit a height while keeping
width/depth proportional."""
1424: (8)                      return self.rescale_to_fit(height, 1, stretch=stretch, **kwargs)
1425: (4)                      scale_to_fit_height = set_height
1426: (4)                      def set_depth(self, depth, stretch=False, **kwargs):
1427: (8)                          """Scales the :class:`~.OpenGLMobject` to fit a depth while keeping
width/height proportional."""
1428: (8)                      return self.rescale_to_fit(depth, 2, stretch=stretch, **kwargs)
1429: (4)                      scale_to_fit_depth = set_depth
1430: (4)                      def set_coord(self, value, dim, direction=ORIGIN):
1431: (8)                          curr = self.get_coord(dim, direction)
1432: (8)                          shift_vect = np.zeros(self.dim)
1433: (8)                          shift_vect[dim] = value - curr
1434: (8)                          self.shift(shift_vect)
1435: (8)                          return self
1436: (4)                      def set_x(self, x, direction=ORIGIN):
1437: (8)                          """Set x value of the center of the :class:`~.OpenGLMobject` (`int`  

or `float`)`"""
1438: (8)                      return self.set_coord(x, 0, direction)
1439: (4)                      def set_y(self, y, direction=ORIGIN):
1440: (8)                          """Set y value of the center of the :class:`~.OpenGLMobject` (`int`  

or `float`)`"""
1441: (8)                      return self.set_coord(y, 1, direction)
1442: (4)                      def set_z(self, z, direction=ORIGIN):
1443: (8)                          """Set z value of the center of the :class:`~.OpenGLMobject` (`int`  

or `float`)`"""
1444: (8)                      return self.set_coord(z, 2, direction)
1445: (4)                      def space_out_submobjects(self, factor=1.5, **kwargs):

```

```

1446: (8)             self.scale(factor, **kwargs)
1447: (8)             for submob in self.submobjects:
1448: (12)                 submob.scale(1.0 / factor)
1449: (8)             return self
1450: (4)             def move_to(
1451: (8)                 self,
1452: (8)                 point_or_mobject,
1453: (8)                 aligned_edge=ORIGIN,
1454: (8)                 coor_mask=np.array([1, 1, 1]),
1455: (4)             ):
1456: (8)                 """Move center of the :class:`~.OpenGLMobject` to certain
1457: (8)                 coordinate."""
1458: (12)             if isinstance(point_or_mobject, OpenGLMobject):
1459: (8)                 target = point_or_mobject.get_bounding_box_point(aligned_edge)
1460: (12)             else:
1461: (8)                 target = point_or_mobject
1462: (8)             point_to_align = self.get_bounding_box_point(aligned_edge)
1463: (8)             self.shift((target - point_to_align) * coor_mask)
1464: (8)             return self
1465: (4)             def replace(self, mobject, dim_to_match=0, stretch=False):
1466: (8)                 if not mobject.get_num_points() and not mobject.submobjects:
1467: (12)                     self.scale(0)
1468: (8)                     return self
1469: (12)                 if stretch:
1470: (16)                     for i in range(self.dim):
1471: (8)                         self.rescale_to_fit(mobject.length_over_dim(i), i,
stretch=True)
1472: (12)             else:
1473: (16)                 self.rescale_to_fit(
1474: (16)                     mobject.length_over_dim(dim_to_match),
1475: (16)                     dim_to_match,
1476: (12)                     stretch=False,
1477: (8)                 )
1478: (8)                 self.shift(mobject.get_center() - self.get_center())
1479: (8)             return self
1480: (4)             def surround(
1481: (8)                 self,
1482: (8)                 mobject: OpenGLMobject,
1483: (8)                 dim_to_match: int = 0,
1484: (8)                 stretch: bool = False,
1485: (4)                 buff: float = MED_SMALL_BUFF,
1486: (8)             ):
1487: (8)                 self.replace(mobject, dim_to_match, stretch)
1488: (8)                 length = mobject.length_over_dim(dim_to_match)
1489: (8)                 self.scale((length + buff) / length)
1490: (8)             return self
1491: (4)             def put_start_and_end_on(self, start, end):
1492: (8)                 curr_start, curr_end = self.get_start_and_end()
1493: (8)                 curr_vect = curr_end - curr_start
1494: (12)                 if np.all(curr_vect == 0):
1495: (8)                     raise Exception("Cannot position endpoints of closed loop")
1496: (8)                 target_vect = np.array(end) - np.array(start)
1497: (12)                 axis = (
1498: (12)                     normalize(np.cross(curr_vect, target_vect))
1499: (12)                     if np.linalg.norm(np.cross(curr_vect, target_vect)) != 0
else OUT
1500: (8)                 )
1501: (8)                 self.scale(
1502: (12)                     np.linalg.norm(target_vect) / np.linalg.norm(curr_vect),
1503: (12)                     about_point=curr_start,
1504: (8)                 )
1505: (8)                 self.rotate(
1506: (12)                     angle_between_vectors(curr_vect, target_vect),
1507: (12)                     about_point=curr_start,
1508: (12)                     axis=axis,
1509: (8)                 )
1510: (8)                 self.shift(start - curr_start)
1511: (8)             return self
1512: (4)             def set_rgba_array(self, color=None, opacity=None, name="rgbas",

```

```

recurse=True):
    1513: (8)           if color is not None:
    1514: (12)          rgbs = np.array([color_to_rgb(c) for c in listify(color)])
    1515: (8)           if opacity is not None:
    1516: (12)          opacities = listify(opacity)
    1517: (8)           if color is not None and opacity is None:
    1518: (12)          for mob in self.get_family(recurse):
    1519: (16)            mob.data[name] = resize_array(
    1520: (20)              mob.data[name] if name in mob.data else np.empty((1, 3)),
len(rgbss)
    1521: (16)            )
    1522: (16)          mob.data[name][:, :3] = rgbs
    1523: (8)           if color is None and opacity is not None:
    1524: (12)          for mob in self.get_family(recurse):
    1525: (16)            mob.data[name] = resize_array(
    1526: (20)              mob.data[name] if name in mob.data else np.empty((1, 3)),
    1527: (20)              len(opacities),
    1528: (16)            )
    1529: (16)          mob.data[name][:, 3] = opacities
    1530: (8)           if color is not None and opacity is not None:
    1531: (12)          rgbs = np.array([[*rgb, o] for rgb, o in zip(*make_even(rgbss,
opacities))])
    1532: (12)          for mob in self.get_family(recurse):
    1533: (16)            mob.data[name] = rgbs.copy()
    1534: (8)          return self
    1535: (4)          def set_rgba_array_direct(self, rgbs: np.ndarray, name="rgbs",
recurve=True):
    1536: (8)            """Directly set rgba data from `rgbs` and optionally do the same
recursively
    1537: (8)            with submobjects. This can be used if the `rgbs` have already been
generated
    1538: (8)            with the correct shape and simply need to be set.
    1539: (8)            Parameters
    1540: (8)            -----
    1541: (8)            rgbs
    1542: (12)              the rgba to be set as data
    1543: (8)            name
    1544: (12)              the name of the data attribute to be set
    1545: (8)            recurse
    1546: (12)              set to true to recursively apply this method to submobjects
    1547: (8)
    1548: (8)            for mob in self.get_family(recurse):
    1549: (12)              mob.data[name] = rgbs.copy()
    1550: (4)          def set_color(self, color: ParsableManimColor | None, opacity=None,
recurve=True):
    1551: (8)            self.set_rgba_array(color, opacity, recurse=False)
    1552: (8)            if color is not None:
    1553: (12)              self.color: ManimColor = ManimColor.parse(color)
    1554: (8)            if opacity is not None:
    1555: (12)              self.opacity = opacity
    1556: (8)            if recurse:
    1557: (12)              for submob in self.submobjects:
    1558: (16)                submob.set_color(color, recurse=True)
    1559: (8)            return self
    1560: (4)          def set_opacity(self, opacity, recurse=True):
    1561: (8)            self.set_rgba_array(color=None, opacity=opacity, recurse=False)
    1562: (8)            if recurse:
    1563: (12)              for submob in self.submobjects:
    1564: (16)                submob.set_opacity(opacity, recurse=True)
    1565: (8)            return self
    1566: (4)          def get_color(self):
    1567: (8)            return rgb_to_hex(self.rgbss[0, :3])
    1568: (4)          def get_opacity(self):
    1569: (8)            return self.rgbss[0, 3]
    1570: (4)          def set_color_by_gradient(self, *colors):
    1571: (8)            self.set_submobject_colors_by_gradient(*colors)
    1572: (8)            return self
    1573: (4)          def set_submobject_colors_by_gradient(self, *colors):
    1574: (8)            if len(colors) == 0:

```

```

1575: (12)           raise Exception("Need at least one color")
1576: (8)            elif len(colors) == 1:
1577: (12)              return self.set_color(*colors)
1578: (8)            mobs = self.submobjects
1579: (8)            new_colors = color_gradient(colors, len(mobs))
1580: (8)            for mob, color in zip(mobs, new_colors):
1581: (12)                mob.set_color(color)
1582: (8)            return self
1583: (4)            def fade(self, darkness=0.5, recurse=True):
1584: (8)                self.set_opacity(1.0 - darkness, recurse=recurse)
1585: (4)            def get_gloss(self):
1586: (8)                return self.gloss
1587: (4)            def set_gloss(self, gloss, recurse=True):
1588: (8)                for mob in self.get_family(recurse):
1589: (12)                    mob.gloss = gloss
1590: (8)                return self
1591: (4)            def get_shadow(self):
1592: (8)                return self.shadow
1593: (4)            def set_shadow(self, shadow, recurse=True):
1594: (8)                for mob in self.get_family(recurse):
1595: (12)                    mob.shadow = shadow
1596: (8)                return self
1597: (4)            def add_background_rectangle(
1598: (8)                self, color: ParsableManimColor | None = None, opacity: float = 0.75,
**kwargs
1599: (4)            ):
1600: (8)                """Add a BackgroundRectangle as submobject.
1601: (8)                The BackgroundRectangle is added behind other submobjects.
1602: (8)                This can be used to increase the mobjects visibility in front of a
noisy background.
1603: (8)            Parameters
1604: (8)                -----
1605: (8)                color
1606: (12)                    The color of the BackgroundRectangle
1607: (8)                opacity
1608: (12)                    The opacity of the BackgroundRectangle
1609: (8)                kwargs
1610: (12)                    Additional keyword arguments passed to the BackgroundRectangle
constructor
1611: (8)            Returns
1612: (8)                -----
1613: (8)                :class:`~.BackgroundRectangle`
1614: (12)                    ``self``
1615: (8)            See Also
1616: (8)                -----
1617: (8)                :meth:`add_to_back`
1618: (8)                :class:`~.BackgroundRectangle`
1619: (8)                """
1620: (8)            from manim.mobject.geometry.shape_matchers import BackgroundRectangle
1621: (8)            self.background_rectangle = BackgroundRectangle(
1622: (12)                self, color=color, fill_opacity=opacity, **kwargs
1623: (8)
1624: (8)
1625: (8)
1626: (4)            )
1627: (8)            self.add_to_back(self.background_rectangle)
1628: (12)            return self
1629: (8)
1630: (4)            def add_background_rectangle_to_submobjects(self, **kwargs):
1631: (8)                for submobject in self.submobjects:
1632: (12)                    submobject.add_background_rectangle(**kwargs)
1633: (8)
1634: (4)            def add_background_rectangle_to_family_members_with_points(self,
**kwargs):
1635: (8)
1636: (8)
1637: (8)
1638: (4)
1639: (8)                for mob in self.family_members_with_points():
1640: (12)                    mob.add_background_rectangle(**kwargs)
1641: (8)
1642: (4)            return self
1643: (8)
1644: (4)            def get_bounding_box_point(self, direction):
1645: (8)                bb = self.get_bounding_box()
1646: (8)                indices = (np.sign(direction) + 1).astype(int)
1647: (8)                return np.array([bb[indices[i]][i] for i in range(3)])
1648: (4)            def get_edge_center(self, direction) -> np.ndarray:
1649: (8)                """Get edge coordinates for certain direction."""

```

```

1640: (8)             return self.get_bounding_box_point(direction)
1641: (4)             def get_corner(self, direction) -> np.ndarray:
1642: (8)                 """Get corner coordinates for certain direction."""
1643: (8)                 return self.get_bounding_box_point(direction)
1644: (4)             def get_center(self) -> np.ndarray:
1645: (8)                 """Get center coordinates."""
1646: (8)                 return self.get_bounding_box()[1]
1647: (4)             def get_center_of_mass(self):
1648: (8)                 return self.get_all_points().mean(0)
1649: (4)             def get_boundary_point(self, direction):
1650: (8)                 all_points = self.get_all_points()
1651: (8)                 boundary_directions = all_points - self.get_center()
1652: (8)                 norms = np.linalg.norm(boundary_directions, axis=1)
1653: (8)                 boundary_directions /= np.repeat(norms, 3).reshape(len(norms), 3)
1654: (8)                 index = np.argmax(np.dot(boundary_directions, np.array(direction).T))
1655: (8)                 return all_points[index]
1656: (4)             def get_continuous_bounding_box_point(self, direction):
1657: (8)                 dl, center, ur = self.get_bounding_box()
1658: (8)                 corner_vect = ur - center
1659: (8)                 return center + direction / np.max(
1660: (12)                     np.abs(
1661: (16)                         np.true_divide(
1662: (20)                             direction,
1663: (20)                             corner_vect,
1664: (20)                             out=np.zeros(len(direction)),
1665: (20)                             where=((corner_vect) != 0),
1666: (16)                         ),
1667: (12)                     ),
1668: (8)                 )
1669: (4)             def get_top(self) -> np.ndarray:
1670: (8)                 """Get top coordinates of a box bounding the
:class:`~.OpenGLMobject`"""
1671: (8)                 return self.get_edge_center(UP)
1672: (4)             def get_bottom(self) -> np.ndarray:
1673: (8)                 """Get bottom coordinates of a box bounding the
:class:`~.OpenGLMobject`"""
1674: (8)                 return self.get_edge_center(DOWN)
1675: (4)             def get_right(self) -> np.ndarray:
1676: (8)                 """Get right coordinates of a box bounding the
:class:`~.OpenGLMobject`"""
1677: (8)                 return self.get_edge_center(RIGHT)
1678: (4)             def get_left(self) -> np.ndarray:
1679: (8)                 """Get left coordinates of a box bounding the
:class:`~.OpenGLMobject`"""
1680: (8)                 return self.get_edge_center(LEFT)
1681: (4)             def get_zenith(self) -> np.ndarray:
1682: (8)                 """Get zenith coordinates of a box bounding a 3D
:class:`~.OpenGLMobject`."""
1683: (8)                 return self.get_edge_center(OUT)
1684: (4)             def get_nadir(self) -> np.ndarray:
1685: (8)                 """Get nadir (opposite the zenith) coordinates of a box bounding a 3D
:class:`~.OpenGLMobject`."""
1686: (8)                 return self.get_edge_center(IN)
1687: (4)             def length_over_dim(self, dim):
1688: (8)                 bb = self.get_bounding_box()
1689: (8)                 return abs((bb[2] - bb[0])[dim])
1690: (4)             def get_width(self):
1691: (8)                 """Returns the width of the mobject."""
1692: (8)                 return self.length_over_dim(0)
1693: (4)             def get_height(self):
1694: (8)                 """Returns the height of the mobject."""
1695: (8)                 return self.length_over_dim(1)
1696: (4)             def get_depth(self):
1697: (8)                 """Returns the depth of the mobject."""
1698: (8)                 return self.length_over_dim(2)
1699: (4)             def get_coord(self, dim: int, direction=ORIGIN):
1700: (8)                 """Meant to generalize ``get_x``, ``get_y`` and ``get_z``"""
1701: (8)                 return self.get_bounding_box_point(direction)[dim]
1702: (4)             def get_x(self, direction=ORIGIN) -> np.float64:

```

```

1703: (8)             """Returns x coordinate of the center of the :class:`~.OpenGLMobject` as ``float``"""
1704: (8)             return self.get_coord(0, direction)
1705: (4)             def get_y(self, direction=ORIGIN) -> np.float64:
1706: (8)                 """Returns y coordinate of the center of the :class:`~.OpenGLMobject` as ``float``"""
1707: (8)                 return self.get_coord(1, direction)
1708: (4)             def get_z(self, direction=ORIGIN) -> np.float64:
1709: (8)                 """Returns z coordinate of the center of the :class:`~.OpenGLMobject` as ``float``"""
1710: (8)                 return self.get_coord(2, direction)
1711: (4)             def get_start(self):
1712: (8)                 """Returns the point, where the stroke that surrounds the :class:`~.OpenGLMobject` starts."""
1713: (8)                 self.throw_error_if_no_points()
1714: (8)                 return np.array(self.points[0])
1715: (4)             def get_end(self):
1716: (8)                 """Returns the point, where the stroke that surrounds the :class:`~.OpenGLMobject` ends."""
1717: (8)                 self.throw_error_if_no_points()
1718: (8)                 return np.array(self.points[-1])
1719: (4)             def get_start_and_end(self):
1720: (8)                 """Returns starting and ending point of a stroke as a ``tuple``."""
1721: (8)                 return self.get_start(), self.get_end()
1722: (4)             def point_from_proportion(self, alpha):
1723: (8)                 points = self.points
1724: (8)                 i, subalpha = integer_interpolate(0, len(points) - 1, alpha)
1725: (8)                 return interpolate(points[i], points[i + 1], subalpha)
1726: (4)             def pfp(self, alpha):
1727: (8)                 """Abbreviation for point_from_proportion"""
1728: (8)                 return self.point_from_proportion(alpha)
1729: (4)             def get_pieces(self, n_pieces):
1730: (8)                 template = self.copy()
1731: (8)                 template.subobjects = []
1732: (8)                 alphas = np.linspace(0, 1, n_pieces + 1)
1733: (8)                 return OpenGLGroup(
1734: (12)                     *
1735: (16)                         template.copy().pointwise_become_partial(self, a1, a2)
1736: (16)                         for a1, a2 in zip(alphas[:-1], alphas[1:])
1737: (12)                     )
1738: (8)                 )
1739: (4)             def get_z_index_reference_point(self):
1740: (8)                 z_index_group = getattr(self, "z_index_group", self)
1741: (8)                 return z_index_group.get_center()
1742: (4)             def match_color(self, mobject: OpenGLMobject):
1743: (8)                 """Match the color with the color of another :class:`~.OpenGLMobject`."""
1744: (8)                 return self.set_color(mobject.get_color())
1745: (4)             def match_dim_size(self, mobject: OpenGLMobject, dim, **kwargs):
1746: (8)                 """Match the specified dimension with the dimension of another :class:`~.OpenGLMobject`."""
1747: (8)                 return self.rescale_to_fit(mobject.length_over_dim(dim), dim,
1748: (**kwargs)
1749: (4)             def match_width(self, mobject: OpenGLMobject, **kwargs):
1750: (8)                 """Match the width with the width of another :class:`~.OpenGLMobject`."""
1751: (8)                 return self.match_dim_size(mobject, 0, **kwargs)
1752: (4)             def match_height(self, mobject: OpenGLMobject, **kwargs):
1753: (8)                 """Match the height with the height of another :class:`~.OpenGLMobject`."""
1754: (8)                 return self.match_dim_size(mobject, 1, **kwargs)
1755: (4)             def match_depth(self, mobject: OpenGLMobject, **kwargs):
1756: (8)                 """Match the depth with the depth of another :class:`~.OpenGLMobject`."""
1757: (8)                 return self.match_dim_size(mobject, 2, **kwargs)
1758: (4)             def match_coord(self, mobject: OpenGLMobject, dim, direction=ORIGIN):
1759: (8)                 """Match the coordinates with the coordinates of another :class:`~.OpenGLMobject`."""

```

```

1760: (12)             mobject.get_coord(dim, direction),
1761: (12)             dim=dim,
1762: (12)             direction=direction,
1763: (8)
1764: (4)         def match_x(self, mobject, direction=ORIGIN):
1765: (8)             """Match x coord. to the x coord. of another
:class:`~.OpenGLMobject`."""
1766: (8)             return self.match_coord(mobject, 0, direction)
1767: (4)         def match_y(self, mobject, direction=ORIGIN):
1768: (8)             """Match y coord. to the x coord. of another
:class:`~.OpenGLMobject`."""
1769: (8)             return self.match_coord(mobject, 1, direction)
1770: (4)         def match_z(self, mobject, direction=ORIGIN):
1771: (8)             """Match z coord. to the x coord. of another
:class:`~.OpenGLMobject`."""
1772: (8)             return self.match_coord(mobject, 2, direction)
1773: (4)         def align_to(
1774: (8)             self,
1775: (8)             mobject_or_point: OpenGLMobject | Sequence[float],
1776: (8)             direction=ORIGIN,
1777: (4)         ):
1778: (8)             """
1779: (8)             Examples:
1780: (8)             mob1.align_to(mob2, UP) moves mob1 vertically so that its
1781: (8)             top edge lines up with mob2's top edge.
1782: (8)             mob1.align_to(mob2, alignment_vect = RIGHT) moves mob1
1783: (8)             horizontally so that it's center is directly above/below
1784: (8)             the center of mob2
1785: (8)             """
1786: (8)             if isinstance(mobject_or_point, OpenGLMobject):
1787: (12)                 point = mobject_or_point.get_bounding_box_point(direction)
1788: (8)             else:
1789: (12)                 point = mobject_or_point
1790: (8)             for dim in range(self.dim):
1791: (12)                 if direction[dim] != 0:
1792: (16)                     self.set_coord(point[dim], dim, direction)
1793: (8)             return self
1794: (4)         def get_group_class(self):
1795: (8)             return OpenGLGroup
1796: (4)         @staticmethod
1797: (4)         def get_mobject_type_class():
1798: (8)             """Return the base class of this mobject type."""
1799: (8)             return OpenGLMobject
1800: (4)         def align_data_and_family(self, mobject):
1801: (8)             self.align_family(mobject)
1802: (8)             self.align_data(mobject)
1803: (4)         def align_data(self, mobject):
1804: (8)             for mob1, mob2 in zip(self.get_family(), mobject.get_family()):
1805: (12)                 mob1.align_points(mob2)
1806: (12)                 for key in mob1.data.keys() & mob2.data.keys():
1807: (16)                     if key == "points":
1808: (20)                         continue
1809: (16)
1810: (16)
1811: (16)
1812: (20)
1813: (16)
1814: (20)
1815: (4)         def align_points(self, mobject):
1816: (8)             max_len = max(self.get_num_points(), mobject.get_num_points())
1817: (8)             for mob in (self, mobject):
1818: (12)                 mob.resize_points(max_len, resize_func=resize_preserving_order)
1819: (8)             return self
1820: (4)         def align_family(self, mobject):
1821: (8)             mob1 = self
1822: (8)             mob2 = mobject
1823: (8)             n1 = len(mob1)
1824: (8)             n2 = len(mob2)
1825: (8)             if n1 != n2:

```

```

1826: (12)             mob1.add_n_more_submobjects(max(0, n2 - n1))
1827: (12)             mob2.add_n_more_submobjects(max(0, n1 - n2))
1828: (8)              for sm1, sm2 in zip(mob1.submobjects, mob2.submobjects):
1829: (12)                  sm1.align_family(sm2)
1830: (8)              return self
1831: (4)              def push_self_into_submobjects(self):
1832: (8)                  copy = self.deepcopy()
1833: (8)                  copy.submobjects = []
1834: (8)                  self.resize_points(0)
1835: (8)                  self.add(copy)
1836: (8)              return self
1837: (4)              def add_n_more_submobjects(self, n):
1838: (8)                  if n == 0:
1839: (12)                      return self
1840: (8)                  curr = len(self.submobjects)
1841: (8)                  if curr == 0:
1842: (12)                      null_mob = self.copy()
1843: (12)                      null_mob.set_points([self.get_center()])
1844: (12)                      self.submobjects = [null_mob.copy() for k in range(n)]
1845: (12)                  return self
1846: (8)                  target = curr + n
1847: (8)                  repeat_indices = (np.arange(target) * curr) // target
1848: (8)                  split_factors = [(repeat_indices == i).sum() for i in range(curr)]
1849: (8)                  new_submobs = []
1850: (8)                  for submob, sf in zip(self.submobjects, split_factors):
1851: (12)                      new_submobs.append(submob)
1852: (12)                      for _ in range(1, sf):
1853: (16)                          new_submob = submob.copy()
1854: (16)                          if submob.get_opacity() < 1:
1855: (20)                              new_submob.set_opacity(0)
1856: (16)                          new_submobs.append(new_submob)
1857: (8)                  self.submobjects = new_submobs
1858: (8)              return self
1859: (4)              def interpolate(self, mobject1, mobject2, alpha,
path_func=straight_path()):
1860: (8)                  """Turns this :class:`~.OpenGLMobject` into an interpolation between
``mobject1``
1861: (8)                  and ``mobject2``.
1862: (8)                  Examples
1863: (8)                  -----
1864: (8)                  .. manim:: DotInterpolation
1865: (12)                      :save_last_frame:
1866: (12)                      class DotInterpolation(Scene):
1867: (16)                          def construct(self):
1868: (20)                              dotR = Dot(color=DARK_GREY)
1869: (20)                              dotR.shift(2 * RIGHT)
1870: (20)                              dotL = Dot(color=WHITE)
1871: (20)                              dotL.shift(2 * LEFT)
1872: (20)                              dotMiddle = OpenGLMobject().interpolate(dotL, dotR,
alpha=0.3)
1873: (20)
1874: (8)
1875: (8)
1876: (12)
1877: (16)
1878: (12)
1879: (16)
1880: (12)
1881: (16)
1882: (12)
1883: (16)
1884: (12)
1885: (16)
1886: (12)
1887: (8)
1888: (12)
1889: (16)
1890: (20)
alpha)
1887: (8)                  self.data[key][:] = func(mobject1.data[key], mobject2.data[key],
1888: (12)                      alpha)
1889: (16)
1890: (20)
for key in self.uniforms:
    if key != "fixed_orientation_center":
        self.uniforms[key] = interpolate(
            mobject1.uniforms[key],
```

```

1891: (20)                         mobject2.uniforms[key],
1892: (20)                         alpha,
1893: (16)                     )
1894: (12)                 else:
1895: (16)                         self.uniforms["fixed_orientation_center"] = tuple(
1896: (20)                             interpolate(
1897: (24)
np.array(mobject1.uniforms["fixed_orientation_center"]),
1898: (24)
np.array(mobject2.uniforms["fixed_orientation_center"]),
1899: (24)                         alpha,
1900: (20)                     )
1901: (16)                 )
1902: (8)             return self
1903: (4)         def pointwise_become_partial(self, mobject, a, b):
1904: (8)             """
1905: (8)                 Set points in such a way as to become only
1906: (8)                 part of mobject.
1907: (8)                 Inputs 0 <= a < b <= 1 determine what portion
1908: (8)                 of mobject to become.
1909: (8)             """
1910: (8)             pass # To implement in subclass
1911: (4)         def become(
1912: (8)             self,
1913: (8)             mobject: OpenGLMobject,
1914: (8)             match_height: bool = False,
1915: (8)             match_width: bool = False,
1916: (8)             match_depth: bool = False,
1917: (8)             match_center: bool = False,
1918: (8)             stretch: bool = False,
1919: (4)         ):
1920: (8)             """Edit all data and subobjects to be identical
1921: (8)             to another :class:`~.OpenGLMobject`
1922: (8)             .. note::
1923: (12)                 If both match_height and match_width are ``True`` then the
transformed :class:`~.OpenGLMobject`  

1924: (12)                         will match the height first and then the width
1925: (8)             Parameters
1926: (8)             -----
1927: (8)                 match_height
1928: (12)                     If ``True``, then the transformed :class:`~.OpenGLMobject` will
match the height of the original
1929: (8)                     match_width
1930: (12)                     If ``True``, then the transformed :class:`~.OpenGLMobject` will
match the width of the original
1931: (8)                     match_depth
1932: (12)                     If ``True``, then the transformed :class:`~.OpenGLMobject` will
match the depth of the original
1933: (8)                     match_center
1934: (12)                     If ``True``, then the transformed :class:`~.OpenGLMobject` will
match the center of the original
1935: (8)                     stretch
1936: (12)                     If ``True``, then the transformed :class:`~.OpenGLMobject` will
stretch to fit the proportions of the original
1937: (8)             Examples
1938: (8)             -----
1939: (8)             .. manim:: BecomeScene
1940: (12)                 class BecomeScene(Scene):
1941: (16)                     def construct(self):
1942: (20)                         circ = Circle(fill_color=RED, fill_opacity=0.8)
1943: (20)                         square = Square(fill_color=BLUE, fill_opacity=0.2)
1944: (20)                         self.add(circ)
1945: (20)                         self.wait(0.5)
1946: (20)                         circ.become(square)
1947: (20)                         self.wait(0.5)
1948: (8)                     """
1949: (8)                     if stretch:
1950: (12)                         mobject.stretch_to_fit_height(self.height)
1951: (12)                         mobject.stretch_to_fit_width(self.width)

```

```

1952: (12)             mobject.stretch_to_fit_depth(self.depth)
1953: (8)              else:
1954: (12)                if match_height:
1955: (16)                  mobject.match_height(self)
1956: (12)                if match_width:
1957: (16)                  mobject.match_width(self)
1958: (12)                if match_depth:
1959: (16)                  mobject.match_depth(self)
1960: (8)                if match_center:
1961: (12)                  mobject.move_to(self.get_center())
1962: (8)                self.align_family(mobject)
1963: (8)                for sm1, sm2 in zip(self.get_family(), mobject.get_family()):
1964: (12)                  sm1.set_data(sm2.data)
1965: (12)                  sm1.set_uniforms(sm2.uniforms)
1966: (8)                  self.refresh_bounding_box(recurse_down=True)
1967: (8)                return self
1968: (4)              def lock_data(self, keys):
1969: (8)                """
1970: (8)                  To speed up some animations, particularly transformations,
1971: (8)                  it can be handy to acknowledge which pieces of data
1972: (8)                  won't change during the animation so that calls to
1973: (8)                  interpolate can skip this, and so that it's not
1974: (8)                  read into the shader_wrapper objects needlessly
1975: (8)                """
1976: (8)                if self.has_updaters:
1977: (12)                  return
1978: (8)                self.refresh_shader_data()
1979: (8)                self.locked_data_keys = set(keys)
1980: (4)              def lock_matching_data(self, mobject1, mobject2):
1981: (8)                for sm, sm1, sm2 in zip(
1982: (12)                  self.get_family(),
1983: (12)                  mobject1.get_family(),
1984: (12)                  mobject2.get_family(),
1985: (8)                ):
1986: (12)                  keys = sm.data.keys() & sm1.data.keys() & sm2.data.keys()
1987: (12)                  sm.lock_data(
1988: (16)                    list(
1989: (20)                      filter(
1990: (24)                        lambda key: np.all(sm1.data[key] == sm2.data[key]),
1991: (24)                        keys,
1992: (20)                      ),
1993: (16)                    ),
1994: (12)                  )
1995: (8)                return self
1996: (4)              def unlock_data(self):
1997: (8)                for mob in self.get_family():
1998: (12)                  mob.locked_data_keys = set()
1999: (4)              @affects_shader_info_id
2000: (4)              def fix_in_frame(self):
2001: (8)                self.is_fixed_in_frame = 1.0
2002: (8)                return self
2003: (4)              @affects_shader_info_id
2004: (4)              def fix_orientation(self):
2005: (8)                self.is_fixed_orientation = 1.0
2006: (8)                self.fixed_orientation_center = tuple(self.get_center())
2007: (8)                self.depth_test = True
2008: (8)                return self
2009: (4)              @affects_shader_info_id
2010: (4)              def unfix_from_frame(self):
2011: (8)                self.is_fixed_in_frame = 0.0
2012: (8)                return self
2013: (4)              @affects_shader_info_id
2014: (4)              def unfix_orientation(self):
2015: (8)                self.is_fixed_orientation = 0.0
2016: (8)                self.fixed_orientation_center = (0, 0, 0)
2017: (8)                self.depth_test = False
2018: (8)                return self
2019: (4)              @affects_shader_info_id
2020: (4)              def apply_depth_test(self):

```

```

2021: (8)           self.depth_test = True
2022: (8)           return self
2023: (4)           @affects_shader_info_id
2024: (4)           def deactivate_depth_test(self):
2025: (8)             self.depth_test = False
2026: (8)             return self
2027: (4)           def replace_shader_code(self, old, new):
2028: (8)             for wrapper in self.get_shader_wrapper_list():
2029: (12)               wrapper.replace_code(old, new)
2030: (8)             return self
2031: (4)           def set_color_by_code(self, glsl_code):
2032: (8)             """
2033: (8)               Takes a snippet of code and inserts it into a
2034: (8)               context which has the following variables:
2035: (8)               vec4 color, vec3 point, vec3 unit_normal.
2036: (8)               The code should change the color variable
2037: (8)             """
2038: (8)             self.replace_shader_code("//// INSERT COLOR FUNCTION HERE ////",
2039: (8)             glsl_code)
2040: (4)           return self
2041: (8)           def set_color_by_xyz_func(
2042: (8)             self,
2043: (8)             glsl_snippet,
2044: (8)             min_value=-5.0,
2045: (8)             max_value=5.0,
2046: (8)             colormap="viridis",
2047: (4)           ):
2048: (8)             """
2049: (8)               Pass in a glsl expression in terms of x, y and z which returns
2050: (8)               a float.
2051: (8)             """
2052: (12)            for char in "xyz":
2053: (8)               glsl_snippet = glsl_snippet.replace(char, "point." + char)
2054: (8)            rgb_list = get_colormap_list(colormap)
2055: (12)            self.set_color_by_code(
2056: (16)              "color.rgb = float_to_color({}, {}, {}, {});".format(
2057: (16)                glsl_snippet,
2058: (16)                float(min_value),
2059: (16)                float(max_value),
2060: (12)                  get_colormap_code(rgb_list),
2061: (8)                ),
2062: (8)            )
2063: (4)            return self
2064: (8)           def refresh_shader_wrapper_id(self):
2065: (8)             self.get_shader_wrapper().refresh_id()
2066: (4)           return self
2067: (8)           def get_shader_wrapper(self):
2068: (8)             from manim.renderer.shader_wrapper import ShaderWrapper
2069: (12)             self.shader_wrapper = ShaderWrapper(
2070: (12)               vert_data=self.get_shader_data(),
2071: (12)               vert_indices=self.get_shader_vert_indices(),
2072: (12)               uniforms=self.get_shader_uniforms(),
2073: (12)               depth_test=self.depth_test,
2074: (12)               texture_paths=self.texture_paths,
2075: (12)               render_primitive=self.render_primitive,
2076: (8)                 shader_folder=self.__class__.shader_folder,
2077: (8)             )
2078: (4)             return self.shader_wrapper
2079: (8)           def get_shader_wrapper_list(self):
2080: (12)             shader_wrappers = it.chain(
2081: (12)               [self.get_shader_wrapper()],
2082: (8)                 *(sm.get_shader_wrapper_list() for sm in self.subobjects),
2083: (8)             )
2084: (8)             batches = batch_by_property(shader_wrappers, lambda sw: sw.get_id())
2085: (8)             result = []
2086: (12)             for wrapper_group, _ in batches:
2087: (12)               shader_wrapper = wrapper_group[0]
2088: (16)                 if not shader_wrapper.is_valid():
2088: (16)                   continue

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2089: (12)                     shader_wrapper.combine_with(*wrapper_group[1:])
2090: (12)                     if len(shader_wrapper.vert_data) > 0:
2091: (16)                         result.append(shader_wrapper)
2092: (8)                     return result
2093: (4)             def check_data_alignment(self, array, data_key):
2094: (8)                 d_len = len(self.data[data_key])
2095: (8)                 if d_len != 1 and d_len != len(array):
2096: (12)                     self.data[data_key] = resize_with_interpolation(
2097: (16)                         self.data[data_key],
2098: (16)                         len(array),
2099: (12)                     )
2100: (8)                     return self
2101: (4)             def get_resized_shader_data_array(self, length):
2102: (8)                 points = self.points
2103: (8)                 shader_data = np.zeros(len(points), dtype=self.shader_dtype)
2104: (8)                 return shader_data
2105: (4)             def read_data_to_shader(self, shader_data, shader_data_key, data_key):
2106: (8)                 if data_key in self.locked_data_keys:
2107: (12)                     return
2108: (8)                 self.check_data_alignment(shader_data, data_key)
2109: (8)                 shader_data[shader_data_key] = self.data[data_key]
2110: (4)             def get_shader_data(self):
2111: (8)                 shader_data =
self.get_resized_shader_data_array(self.get_num_points())
2112: (8)                     self.read_data_to_shader(shader_data, "point", "points")
2113: (8)                     return shader_data
2114: (4)             def refresh_shader_data(self):
2115: (8)                 self.get_shader_data()
2116: (4)             def get_shader_uniforms(self):
2117: (8)                 return self.uniforms
2118: (4)             def get_shader_vert_indices(self):
2119: (8)                 return self.shader_indices
2120: (4)             @property
2121: (4)             def subobjects(self):
2122: (8)                 return self._subobjects if hasattr(self, "_subobjects") else []
2123: (4)             @subobjects.setter
2124: (4)             def subobjects(self, subobject_list):
2125: (8)                 self.remove(*self.subobjects)
2126: (8)                 self.add(*subobject_list)
2127: (4)             def throw_error_if_no_points(self):
2128: (8)                 if not self.has_points():
2129: (12)                     message = (
2130: (16)                         "Cannot call OpenGLObject.{ } " + "for a OpenGLObject with no
points"
2131: (12)
2132: (12)
2133: (12)
2134: (0)             caller_name = sys._getframe(1).f_code.co_name
2135: (4)             raise Exception(message.format(caller_name))
2136: (8)         class OpenGLGroup(OpenGLMobject):
2137: (12)             def __init__(self, *mobjects, **kwargs):
2138: (8)                 if not all(isinstance(m, OpenGLMobject) for m in mobjects):
2139: (8)                     raise Exception("All subobjects must be of type OpenGLMobject")
2140: (0)                 super().__init__(**kwargs)
2141: (4)                 self.add(*mobjects)
2142: (8)         class OpenGLPoint(OpenGLMobject):
2143: (**kwargs)
2144: (4)             def __init__(
2145: (8)                 self, location=ORIGIN, artificial_width=1e-6, artificial_height=1e-6,
2146: (8)
2147: (8)
2148: (8)
2149: (8)
2150: (8)
2151: (8)
2152: (8)
2153: (8)
2154: (8)                 self.artificial_width = artificial_width
2155: (8)                 self.artificial_height = artificial_height
2156: (8)                 super().__init__(**kwargs)
2157: (8)                 self.set_location(location)
2158: (4)             def get_width(self):
2159: (8)                 return self.artificial_width
2160: (4)             def get_height(self):
2161: (8)                 return self.artificial_height
2162: (4)             def get_location(self):
2163: (8)                 return self.points[0].copy()
2164: (4)             def get_bounding_box_point(self, *args, **kwargs):

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
2155: (8)             return self.get_location()
2156: (4)         def set_location(self, new_loc):
2157: (8)             self.set_points(np.array(new_loc, ndmin=2, dtype=float))
2158: (0)     class _AnimationBuilder:
2159: (4)         def __init__(self, mobject):
2160: (8)             self.mobject = mobject
2161: (8)             self.mobject.generate_target()
2162: (8)             self.overridden_animation = None
2163: (8)             self.is_chaining = False
2164: (8)             self.methods = []
2165: (8)             self.cannot_pass_args = False
2166: (8)             self.anim_args = {}
2167: (4)         def __call__(self, **kwargs):
2168: (8)             if self.cannot_pass_args:
2169: (12)                 raise ValueError(
2170: (16)                     "Animation arguments must be passed before accessing methods
and can only be passed once",
2171: (12)                         )
2172: (8)                     self.anim_args = kwargs
2173: (8)                     self.cannot_pass_args = True
2174: (8)                     return self
2175: (4)         def __getattr__(self, method_name):
2176: (8)             method = getattr(self.mobject.target, method_name)
2177: (8)             has_overridden_animation = hasattr(method, "_override_animate")
2178: (8)             if (self.is_chaining and has_overridden_animation) or
self.overridden_animation:
2179: (12)                 raise NotImplementedError(
2180: (16)                     "Method chaining is currently not supported for "
2181: (16)                     "overridden animations",
2182: (12)                         )
2183: (8)             def update_target(*method_args, **method_kwargs):
2184: (12)                 if has_overridden_animation:
2185: (16)                     self.overridden_animation = method._override_animate(
2186: (20)                         self.mobject,
2187: (20)                         *method_args,
2188: (20)                         anim_args=self.anim_args,
2189: (20)                         **method_kwargs,
2190: (16)                     )
2191: (12)                 else:
2192: (16)                     self.methods.append([method, method_args, method_kwargs])
2193: (16)                     method(*method_args, **method_kwargs)
2194: (12)                     return self
2195: (8)                     self.is_chaining = True
2196: (8)                     self.cannot_pass_args = True
2197: (8)                     return update_target
2198: (4)             def build(self):
2199: (8)                 from manim.animation.transform import _MethodAnimation
2200: (8)                 if self.overridden_animation:
2201: (12)                     anim = self.overridden_animation
2202: (8)                 else:
2203: (12)                     anim = _MethodAnimation(self.mobject, self.methods)
2204: (8)                 for attr, value in self.anim_args.items():
2205: (12)                     setattr(anim, attr, value)
2206: (8)                 return anim
2207: (0)             def override_animate(method):
2208: (4)                 """Decorator for overriding method animations.
2209: (4)                 This allows to specify a method (returning an :class:`~.Animation`)
2210: (4)                 which is called when the decorated method is used with the ``.animate`````
syntax
2211: (4)                 for animating the application of a method.
2212: (4)                 .. seealso::
2213: (8)                     :attr:`OpenGLMobject.animate`_
2214: (4)                 .. note::
2215: (8)                     Overridden methods cannot be combined with normal or other overridden
2216: (8)                     methods using method chaining with the ``.animate```` syntax.
2217: (4)                 Examples
2218: (4)                 -----
2219: (4)                     .. manim:: AnimationOverrideExample
2220: (8)                         class CircleWithContent(VGroup):
```

```

2221: (12)             def __init__(self, content):
2222: (16)                 super().__init__()
2223: (16)                 self.circle = Circle()
2224: (16)                 self.content = content
2225: (16)                 self.add(self.circle, content)
2226: (16)                 content.move_to(self.circle.get_center())
2227: (12)             def clear_content(self):
2228: (16)                 self.remove(self.content)
2229: (16)                 self.content = None
2230: (12)             @override_animate(clear_content)
2231: (12)             def _clear_content_animation(self, anim_args=None):
2232: (16)                 if anim_args is None:
2233: (20)                     anim_args = {}
2234: (16)                 anim = Uncreate(self.content, **anim_args)
2235: (16)                 self.clear_content()
2236: (16)                 return anim
2237: (8)             class AnimationOverrideExample(Scene):
2238: (12)                 def construct(self):
2239: (16)                     t = Text("hello!")
2240: (16)                     my_mobject = CircleWithContent(t)
2241: (16)                     self.play(Create(my_mobject))
2242: (16)                     self.play(my_mobject.animate.clear_content())
2243: (16)                     self.wait()
2244: (4)             """
2245: (4)             def decorator(animation_method):
2246: (8)                 method._override_animate = animation_method
2247: (8)                 return animation_method
2248: (4)             return decorator

```

---

## File 89 - opengl\_surface.py:

```

1: (0)             from __future__ import annotations
2: (0)             from pathlib import Path
3: (0)             from typing import Iterable
4: (0)             import moderngl
5: (0)             import numpy as np
6: (0)             from PIL import Image
7: (0)             from manim.constants import *
8: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject
9: (0)             from manim.utils.bezier import integer_interpolate, interpolate
10: (0)             from manim.utils.color import *
11: (0)             from manim.utils.config_ops import _Data, _Uniforms
12: (0)             from manim.utils.deprecation import deprecated
13: (0)             from manim.utils.images import change_to_rgba_array,
get_full_raster_image_path
14: (0)             from manim.utils.iterables import listify
15: (0)             from manim.utils.space_ops import normalize_along_axis
16: (0)                 __all__ = ["OpenGLSurface", "OpenGLTexturedSurface"]
17: (0)             class OpenGLSurface(OpenGLMobject):
18: (4)                 r"""Creates a Surface.
19: (4)                 Parameters
20: (4)                 -----
21: (4)                 uv_func
22: (8)                     The function that defines the surface.
23: (4)                 u_range
24: (8)                     The range of the ``u`` variable: ``(``u_min, u_max``)``.
25: (4)                 v_range
26: (8)                     The range of the ``v`` variable: ``(``v_min, v_max``)``.
27: (4)                 resolution
28: (8)                     The number of samples taken of the surface.
29: (4)                 axes
30: (8)                     Axes on which the surface is to be drawn. Optional
31: (8)                     parameter used when coloring a surface by z-value.
32: (4)                 color
33: (8)                     Color of the surface. Defaults to grey.
34: (4)                 colorscale
35: (8)                     Colors of the surface. Optional parameter used when

```

```

36: (8)             coloring a surface by values. Passing a list of
37: (8)             colors and an axes will color the surface by z-value.
38: (8)             Passing a list of tuples in the form ``(color, pivot)``
39: (8)             allows user-defined pivots where the color transitions.
40: (4)             colorscale_axis
41: (8)                 Defines the axis on which the colorscale is applied
42: (8)                 ( $0 = x$ ,  $1 = y$ ,  $2 = z$ ), default is z-axis (2).
43: (4)             opacity
44: (8)                 Opacity of the surface from 0 being fully transparent
45: (8)                 to 1 being fully opaque. Defaults to 1.
46: (4)
47: (4)             """
48: (8)             shader_dtype = [
49: (8)                 ("point", np.float32, (3,)),
50: (8)                 ("du_point", np.float32, (3,)),
51: (8)                 ("dv_point", np.float32, (3,)),
52: (8)                 ("color", np.float32, (4,)),
53: (4)             ]
54: (4)             shader_folder = "surface"
55: (4)             def __init__(
56: (8)                 self,
57: (8)                 uv_func=None,
58: (8)                 u_range=None,
59: (8)                 v_range=None,
60: (8)                 resolution=None,
61: (8)                 axes=None,
62: (8)                 color=GREY,
63: (8)                 colorscale=None,
64: (8)                 colorscale_axis=2,
65: (8)                 opacity=1.0,
66: (8)                 gloss=0.3,
67: (8)                 shadow=0.4,
68: (8)                 prefered_creation_axis=1,
69: (8)                 epsilon=1e-5,
70: (8)                 render_primitive=moderngl.TRIANGLES,
71: (8)                 depth_test=True,
72: (8)                 shader_folder=None,
73: (4)                 **kwargs,
74: (8)             ):
75: (8)                 self.passed_uv_func = uv_func
76: (8)                 self.u_range = u_range if u_range is not None else (0, 1)
77: (8)                 self.v_range = v_range if v_range is not None else (0, 1)
78: (8)                 self.resolution = resolution if resolution is not None else (101, 101)
79: (8)                 self.axes = axes
80: (8)                 self.colors = colorscale
81: (8)                 self.colors_axis = colorscale_axis
82: (8)                 self.prefered_creation_axis = prefered_creation_axis
83: (8)                 self.epsilon = epsilon
84: (8)                 self.triangle_indices = None
85: (12)                 super().__init__(
86: (12)                     color=color,
87: (12)                     opacity=opacity,
88: (12)                     gloss=gloss,
89: (12)                     shadow=shadow,
90: (12)                     shader_folder=shader_folder if shader_folder is not None else
91: (12)                         render_primitive=render_primitive,
92: (12)                         depth_test=depth_test,
93: (8)                         **kwargs,
94: (8)                     )
95: (4)                     self.compute_triangle_indices()
96: (8)             def uv_func(self, u, v):
97: (12)                 if self.passed_uv_func:
98: (8)                     return self.passed_uv_func(u, v)
99: (4)             def init_points(self):
100: (8)                 dim = self.dim
101: (8)                 nu, nv = self.resolution
102: (8)                 u_range = np.linspace(*self.u_range, nu)
103: (8)                 v_range = np.linspace(*self.v_range, nv)

```

```

104: (8)             point_lists = []
105: (8)             for du, dv in [(0, 0), (self.epsilon, 0), (0, self.epsilon)]:
106: (12)                 uv_grid = np.array([[u + du, v + dv] for v in v_range] for u in
107: (12)                     u_range)
108: (12)             point_grid = np.apply_along_axis(lambda p: self.uv_func(*p), 2,
109: (8)                 point_lists.append(point_grid.reshape((nu * nv, dim)))
110: (4)             self.set_points(np.vstack(point_lists))
111: (8)         def compute_triangle_indices(self):
112: (8)             nu, nv = self.resolution
113: (12)             if nu == 0 or nv == 0:
114: (12)                 self.triangle_indices = np.zeros(0, dtype=int)
115: (8)             return
116: (8)             index_grid = np.arange(nu * nv).reshape((nu, nv))
117: (8)             indices = np.zeros(6 * (nu - 1) * (nv - 1), dtype=int)
118: (8)             indices[0::6] = index_grid[:-1, :-1].flatten() # Top left
119: (8)             indices[1::6] = index_grid[+1:, :-1].flatten() # Bottom left
120: (8)             indices[2::6] = index_grid[:-1, +1:].flatten() # Top right
121: (8)             indices[3::6] = index_grid[+1:, +1:].flatten() # Top right
122: (8)             indices[4::6] = index_grid[+1:, :-1].flatten() # Bottom left
123: (8)             indices[5::6] = index_grid[+1:, +1:].flatten() # Bottom right
124: (4)             self.triangle_indices = indices
125: (8)         def get_triangle_indices(self):
126: (4)             return self.triangle_indices
127: (8)         def get_surface_points_and_nudged_points(self):
128: (8)             points = self.points
129: (8)             k = len(points) // 3
130: (4)             return points[:k], points[k : 2 * k], points[2 * k :]
131: (8)         def get_unit_normals(self):
132: (8)             s_points, du_points, dv_points =
133: (12)             self.get_surface_points_and_nudged_points()
134: (12)             normals = np.cross(
135: (8)                 (du_points - s_points) / self.epsilon,
136: (8)                 (dv_points - s_points) / self.epsilon,
137: (4)             )
138: (8)             return normalize_along_axis(normals, 1)
139: (8)         def pointwiseBecomePartial(self, smobject, a, b, axis=None):
140: (12)             assert isinstance(smobject, OpenGLSurface)
141: (8)             if axis is None:
142: (12)                 axis = self.preferred_creation_axis
143: (12)             if a <= 0 and b >= 1:
144: (8)                 self.match_points(smobject)
145: (8)             nu, nv = smobject.resolution
146: (12)             self.set_points(
147: (16)                 np.vstack(
148: (20)                     [
149: (24)                         self.get_partial_points_array(
150: (24)                             arr.copy(),
151: (24)                             a,
152: (24)                             b,
153: (24)                             (nu, nv, 3),
154: (20)                             axis=axis,
155: (20)                         )
156: (16)                     ],
157: (12)                 ),
158: (8)             )
159: (8)             return self
160: (4)         def get_partial_points_array(self, points, a, b, resolution, axis):
161: (8)             if len(points) == 0:
162: (12)                 return points
163: (8)             nu, nv = resolution[:2]
164: (8)             points = points.reshape(resolution)
165: (8)             max_index = resolution[axis] - 1
166: (8)             lower_index, lower_residue = integer_interpolate(0, max_index, a)
167: (8)             upper_index, upper_residue = integer_interpolate(0, max_index, b)
168: (8)             if axis == 0:
169: (12)                 points[:lower_index] = interpolate(

```

```

170: (16)             points[lower_index],
171: (16)             points[lower_index + 1],
172: (16)             lower_residue,
173: (12)         )
174: (12)         points[upper_index + 1 :] = interpolate(
175: (16)             points[upper_index],
176: (16)             points[upper_index + 1],
177: (16)             upper_residue,
178: (12)         )
179: (8)     else:
180: (12)         shape = (nu, 1, resolution[2])
181: (12)         points[:, :lower_index] = interpolate(
182: (16)             points[:, lower_index],
183: (16)             points[:, lower_index + 1],
184: (16)             lower_residue,
185: (12)         ).reshape(shape)
186: (12)         points[:, upper_index + 1 :] = interpolate(
187: (16)             points[:, upper_index],
188: (16)             points[:, upper_index + 1],
189: (16)             upper_residue,
190: (12)         ).reshape(shape)
191: (8)     return points.reshape((nu * nv, *resolution[2:]))
192: (4) def sort_faces_back_to_front(self, vect=OUT):
193: (8)     tri_is = self.triangle_indices
194: (8)     indices = list(range(len(tri_is) // 3))
195: (8)     points = self.points
196: (8)     def index_dot(index):
197: (12)         return np.dot(points[tri_is[3 * index]], vect)
198: (8)     indices.sort(key=index_dot)
199: (8)     for k in range(3):
200: (12)         tri_is[k::3] = tri_is[k::3][indices]
201: (8)     return self
202: (4) def get_shader_data(self):
203: (8)     """Called by parent Mobject to calculate and return
204: (8)     the shader data.
205: (8)     Returns
206: (8)     -----
207: (8)     shader_dtype
208: (12)         An array containing the shader data (vertices and
209: (12)             color of each vertex)
210: (8)     """
211: (8)     s_points, du_points, dv_points =
self.get_surface_points_and_nudged_points()
212: (8)     shader_data = np.zeros(len(s_points), dtype=self.shader_dtype)
213: (8)     if "points" not in self.locked_data_keys:
214: (12)         shader_data["point"] = s_points
215: (12)         shader_data["du_point"] = du_points
216: (12)         shader_data["dv_point"] = dv_points
217: (12)         if self.colorscale:
218: (16)             if not hasattr(self, "color_by_val"):
219: (20)                 self.color_by_val = self._get_color_by_value(s_points)
220: (16)             shader_data["color"] = self.color_by_val
221: (12)         else:
222: (16)             self.fill_in_shader_color_info(shader_data)
223: (8)     return shader_data
224: (4) def fill_in_shader_color_info(self, shader_data):
225: (8)     """Fills in the shader color data when the surface
226: (8)     is all one color.
227: (8)     Parameters
228: (8)     -----
229: (8)     shader_data
230: (12)         The vertices of the surface.
231: (8)     Returns
232: (8)     -----
233: (8)     shader_dtype
234: (12)         An array containing the shader data (vertices and
235: (12)             color of each vertex)
236: (8)     """
237: (8)     self.read_data_to_shader(shader_data, "color", "rgbas")

```

```

238: (8)             return shader_data
239: (4)         def _get_color_by_value(self, s_points):
240: (8)             """Matches each vertex to a color associated to it's z-value.
241: (8)             Parameters
242: (8)             -----
243: (8)             s_points
244: (11)                 The vertices of the surface.
245: (8)             Returns
246: (8)             -----
247: (8)             List
248: (12)                 A list of colors matching the vertex inputs.
249: (8)             """
250: (8)         if type(self.colors[0]) in (list, tuple):
251: (12)             new_colors, pivots = [
252: (16)                 [i for i, j in self.colors],
253: (16)                 [j for i, j in self.colors],
254: (12)             ]
255: (8)         else:
256: (12)             new_colors = self.colors
257: (12)             pivot_min = self.axes.z_range[0]
258: (12)             pivot_max = self.axes.z_range[1]
259: (12)             pivot_frequency = (pivot_max - pivot_min) / (len(new_colors) - 1)
260: (12)             pivots = np.arange(
261: (16)                 start=pivot_min,
262: (16)                 stop=pivot_max + pivot_frequency,
263: (16)                 step=pivot_frequency,
264: (12)             )
265: (8)         return_colors = []
266: (8)         for point in s_points:
267: (12)             axis_value = self.axes.point_to_coords(point)
268: (12)             if axis_value <= pivots[0]:
269: (16)                 return_colors.append(color_to_rgba(new_colors[0],
270: (12) self.opacity))
271: (16)             elif axis_value >= pivots[-1]:
272: (12)                 return_colors.append(color_to_rgba(new_colors[-1],
273: (16) self.opacity))
274: (20)             else:
275: (24)                 for i, pivot in enumerate(pivots):
276: (28)                     if pivot > axis_value:
277: (24)                         color_index = (axis_value - pivots[i - 1]) / (
278: (24)                             pivots[i] - pivots[i - 1]
279: (24)                         )
280: (28)                         color_index = max(min(color_index, 1), 0)
281: (28)                         temp_color = interpolate_color(
282: (28)                             new_colors[i - 1],
283: (24)                             new_colors[i],
284: (24)                             color_index,
285: (16)                         )
286: (8)                         break
287: (4)                     return_colors.append(color_to_rgba(temp_color, self.opacity))
288: (8)             return return_colors
289: (0)         def get_shader_vert_indices(self):
290: (4)             return self.get_triangle_indices()
291: (8)         class OpenGLSurfaceGroup(OpenGLSurface):
292: (8)             def __init__(self, *parametric_surfaces, resolution=None, **kwargs):
293: (8)                 self.resolution = (0, 0) if resolution is None else resolution
294: (8)                 super().__init__(uv_func=None, **kwargs)
295: (8)                 self.add(*parametric_surfaces)
296: (0)             def init_points(self):
297: (4)                 pass # Needed?
298: (0)             class OpenGLTexturedSurface(OpenGLSurface):
299: (4)                 shader_dtype = [
300: (8)                     ("point", np.float32, (3,)),
301: (8)                     ("du_point", np.float32, (3,)),
302: (8)                     ("dv_point", np.float32, (3,)),
303: (8)                     ("im_coords", np.float32, (2,)),
303: (4)                     ("opacity", np.float32, (1,)),
303: (4)                 ]

```

```

304: (4)             shader_folder = "textured_surface"
305: (4)             im_coords = _Data()
306: (4)             opacity = _Data()
307: (4)             num_textures = _Uniforms()
308: (4)             def __init__(
309: (8)                 self,
310: (8)                 uv_surface: OpenGLSurface,
311: (8)                 image_file: str | Path,
312: (8)                 dark_image_file: str | Path = None,
313: (8)                 image_mode: str | Iterable[str] = "RGBA",
314: (8)                 shader_folder: str | Path = None,
315: (8)                 **kwargs,
316: (4)             ):
317: (8)                 self.uniforms = {}
318: (8)                 if not isinstance(uv_surface, OpenGLSurface):
319: (12)                     raise Exception("uv_surface must be of type OpenGLSurface")
320: (8)                 if isinstance(image_file, np.ndarray):
321: (12)                     image_file = change_to_rgba_array(image_file)
322: (8)                 if isinstance(image_mode, (str, Path)):
323: (12)                     image_mode = [image_mode] * 2
324: (8)                 image_mode_light, image_mode_dark = image_mode
325: (8)                 texture_paths = {
326: (12)                     "LightTexture": self.get_image_from_file(
327: (16)                         image_file,
328: (16)                         image_mode_light,
329: (12)                     ),
330: (12)                     "DarkTexture": self.get_image_from_file(
331: (16)                         dark_image_file or image_file,
332: (16)                         image_mode_dark,
333: (12)                     ),
334: (8)                 }
335: (8)                 if dark_image_file:
336: (12)                     self.num_textures = 2
337: (8)                 self.uv_surface = uv_surface
338: (8)                 self.uv_func = uv_surface.uv_func
339: (8)                 self.u_range = uv_surface.u_range
340: (8)                 self.v_range = uv_surface.v_range
341: (8)                 self.resolution = uv_surface.resolution
342: (8)                 self.gloss = self.uv_surface.gloss
343: (8)                 super().__init__(texture_paths=texture_paths, **kwargs)
344: (4)             def get_image_from_file(
345: (8)                 self,
346: (8)                 image_file: str | Path,
347: (8)                 image_mode: str,
348: (4)             ):
349: (8)                 image_file = get_full_raster_image_path(image_file)
350: (8)                 return Image.open(image_file).convert(image_mode)
351: (4)             def init_data(self):
352: (8)                 super().init_data()
353: (8)                 self.im_coords = np.zeros((0, 2))
354: (8)                 self.opacity = np.zeros((0, 1))
355: (4)             def init_points(self):
356: (8)                 nu, nv = self.uv_surface.resolution
357: (8)                 self.set_points(self.uv_surface.points)
358: (8)                 self.im_coords = np.array(
359: (12)                     [
360: (16)                         [u, v]
361: (16)                         for u in np.linspace(0, 1, nu)
362: (16)                         for v in np.linspace(1, 0, nv) # Reverse y-direction
363: (12)                     ],
364: (8)                 )
365: (4)             def init_colors(self):
366: (8)                 self.opacity = np.array([self.uv_surface.rgbas[:, 3]])
367: (4)             def set_opacity(self, opacity, recurse=True):
368: (8)                 for mob in self.get_family(recurse):
369: (12)                     mob.opacity = np.array([[o] for o in listify(opacity)])
370: (8)                 return self
371: (4)             def pointwiseBecomePartial(self, tsmobject, a, b, axis=1):
372: (8)                 super().pointwiseBecomePartial(tsmobject, a, b, axis)

```

```

373: (8)             im_coords = self.im_coords
374: (8)             im_coords[:] = tsmobject.im_coords
375: (8)             if a <= 0 and b >= 1:
376: (12)             return self
377: (8)             nu, nv = tsmobject.resolution
378: (8)             im_coords[:] = self.get_partial_points_array(im_coords, a, b, (nu, nv,
2), axis)
379: (8)             return self
380: (4)         def fill_in_shader_color_info(self, shader_data):
381: (8)             self.read_data_to_shader(shader_data, "opacity", "opacity")
382: (8)             self.read_data_to_shader(shader_data, "im_coords", "im_coords")
383: (8)             return shader_data
-----
```

## File 90 - opengl\_geometry.py:

```

1: (0)             from __future__ import annotations
2: (0)             import numpy as np
3: (0)             from manim.constants import *
4: (0)             from manim.mobject.mobject import Mobject
5: (0)             from manim.mobject.opengl.opengl_vectorized_mobject import (
6: (4)                 OpenGLDashedVMobject,
7: (4)                 OpenGLGroup,
8: (4)                 OpenGLVMobject,
9: (0)             )
10: (0)            from manim.utils.color import *
11: (0)            from manim.utils.iterables import adjacent_n_tuples, adjacent_pairs
12: (0)            from manim.utils.simple_functions import clip
13: (0)            from manim.utils.space_ops import (
14: (4)                angle_between_vectors,
15: (4)                angle_of_vector,
16: (4)                compass_directions,
17: (4)                find_intersection,
18: (4)                normalize,
19: (4)                rotate_vector,
20: (4)                rotation_matrix_transpose,
21: (0)            )
22: (0)            DEFAULT_DOT_RADIUS = 0.08
23: (0)            DEFAULT_DASH_LENGTH = 0.05
24: (0)            DEFAULT_ARROW_TIP_LENGTH = 0.35
25: (0)            DEFAULT_ARROW_TIP_WIDTH = 0.35
26: (0)            __all__ = [
27: (4)                "OpenGLTipableVMobject",
28: (4)                "OpenGLArc",
29: (4)                "OpenGLArcBetweenPoints",
30: (4)                "OpenGLCurvedArrow",
31: (4)                "OpenGLCurvedDoubleArrow",
32: (4)                "OpenGLCircle",
33: (4)                "OpenGLDot",
34: (4)                "OpenGLEllipse",
35: (4)                "OpenGLAnnularSector",
36: (4)                "OpenGLSector",
37: (4)                "OpenGLAnnulus",
38: (4)                "OpenGLLine",
39: (4)                "OpenGLDashedLine",
40: (4)                "OpenGLTangentLine",
41: (4)                "OpenGLElbow",
42: (4)                "OpenGLArrow",
43: (4)                "OpenGLVector",
44: (4)                "OpenGLDoubleArrow",
45: (4)                "OpenGLCubicBezier",
46: (4)                "OpenGLPolygon",
47: (4)                "OpenGLRegularPolygon",
48: (4)                "OpenGLTriangle",
49: (4)                "OpenGLArrowTip",
50: (0)            ]
51: (0)            class OpenGLTipableVMobject(OpenGLVMobject):
52: (4)                """

```

```

53: (4)               Meant for shared functionality between Arc and Line.
54: (4)               Functionality can be classified broadly into these groups:
55: (8)                 * Adding, Creating, Modifying tips
56: (12)                - add_tip calls create_tip, before pushing the new tip
57: (16)                into the TipableVMobject's list of subobjects
58: (12)                - stylistic and positional configuration
59: (8)                 * Checking for tips
60: (12)                - Boolean checks for whether the TipableVMobject has a tip
61: (16)                and a starting tip
62: (8)                 * Getters
63: (12)                - Straightforward accessors, returning information pertaining
64: (16)                to the TipableVMobject instance's tip(s), its length etc
65: (4)
66: (4)               """
67: (8)               def __init__(
68: (8)                   self,
69: (8)                   tip_length=DEFAULT_ARROW_TIP_LENGTH,
70: (8)                   normal_vector=OUT,
71: (8)                   tip_config={},
72: (8)                   **kwargs,
73: (4):
74: (8)                   self.tip_length = tip_length
75: (8)                   self.normal_vector = normal_vector
76: (8)                   self.tip_config = tip_config
77: (4)                   super().__init__(**kwargs)
78: (8)               def add_tip(self, at_start=False, **kwargs):
79: (8)                   """
80: (8)                   Adds a tip to the TipableVMobject instance, recognising
81: (8)                   that the endpoints might need to be switched if it's
82: (8)                   a 'starting tip' or not.
83: (8)                   """
84: (8)                   tip = self.create_tip(at_start, **kwargs)
85: (8)                   self.reset_endpoints_based_on_tip(tip, at_start)
86: (8)                   self.assign_tip_attr(tip, at_start)
87: (8)                   self.add(tip)
88: (4)                   return self
89: (8)               def create_tip(self, at_start=False, **kwargs):
90: (8)                   """
91: (8)                   Stylises the tip, positions it spacially, and returns
92: (8)                   the newly instantiated tip to the caller.
93: (8)                   """
94: (8)                   tip = self.get_unpositioned_tip(**kwargs)
95: (8)                   self.position_tip(tip, at_start)
96: (8)                   return tip
97: (4)               def get_unpositioned_tip(self, **kwargs):
98: (8)                   """
99: (8)                   Returns a tip that has been stylistically configured,
100: (8)                  but has not yet been given a position in space.
101: (8)                   """
102: (8)                   config = {}
103: (8)                   config.update(self.tip_config)
104: (8)                   config.update(kwargs)
105: (4)                   return OpenGLArrowTip(**config)
106: (8)               def position_tip(self, tip, at_start=False):
107: (12)                 if at_start:
108: (12)                     anchor = self.get_start()
109: (8)                     handle = self.get_first_handle()
110: (12)                 else:
111: (12)                     handle = self.get_last_handle()
112: (8)                     anchor = self.get_end()
113: (8)                     tip.rotate(angle_of_vector(handle - anchor) - PI - tip.get_angle())
114: (8)                     tip.shift(anchor - tip.get_tip_point())
115: (4)                     return tip
116: (8)               def reset_endpoints_based_on_tip(self, tip, at_start):
117: (12)                 if self.get_length() == 0:
118: (8)                     return self
119: (12)                 if at_start:
120: (12)                     start = tip.get_base()
121: (8)                     end = self.get_end()

```

```

122: (12)                      start = self.get_start()
123: (12)                      end = tip.get_base()
124: (8)                       self.put_start_and_end_on(start, end)
125: (8)                       return self
126: (4) def assign_tip_attr(self, tip, at_start):
127: (8)   if at_start:
128: (12)     self.start_tip = tip
129: (8)   else:
130: (12)     self.tip = tip
131: (8)   return self
132: (4) def has_tip(self):
133: (8)   return hasattr(self, "tip") and self.tip in self
134: (4) def has_start_tip(self):
135: (8)   return hasattr(self, "start_tip") and self.start_tip in self
136: (4) def pop_tips(self):
137: (8)   start, end = self.get_start_and_end()
138: (8)   result = OpenGLGroup()
139: (8)   if self.has_tip():
140: (12)     result.add(self.tip)
141: (12)     self.remove(self.tip)
142: (8)   if self.has_start_tip():
143: (12)     result.add(self.start_tip)
144: (12)     self.remove(self.start_tip)
145: (8)   self.put_start_and_end_on(start, end)
146: (8)   return result
147: (4) def get_tips(self):
148: (8) """
149: (8)     Returns a VGroup (collection of VMobjects) containing
150: (8)     the TipableVMObject instance's tips.
151: (8) """
152: (8)   result = OpenGLGroup()
153: (8)   if hasattr(self, "tip"):
154: (12)     result.add(self.tip)
155: (8)   if hasattr(self, "start_tip"):
156: (12)     result.add(self.start_tip)
157: (8)   return result
158: (4) def get_tip(self):
159: (8) """
160: (8)     Returns the TipableVMObject instance's (first) tip,
161: (8)     otherwise throws an exception."""
162: (8)   tips = self.get_tips()
163: (12)   if len(tips) == 0:
164: (8)     raise Exception("tip not found")
165: (12)   else:
166: (4)     return tips[0]
167: (8) def get_default_tip_length(self):
168: (4)   return self.tip_length
169: (8) def get_first_handle(self):
170: (4)   return self.points[1]
171: (8) def get_last_handle(self):
172: (4)   return self.points[-2]
173: (8) def get_end(self):
174: (12)   if self.has_tip():
175: (8)     return self.tip.get_start()
176: (12)   else:
177: (4)     return super().get_end()
178: (8) def get_start(self):
179: (12)   if self.has_start_tip():
180: (8)     return self.start_tip.get_start()
181: (12)   else:
182: (4)     return super().get_start()
183: (8) def get_length(self):
184: (8)   start, end = self.get_start_and_end()
185: (0)   return np.linalg.norm(start - end)
186: (4) class OpenGLArc(OpenGLTipableVMobject):
187: (8)   def __init__(
188: (8)     self,
189: (8)     start_angle=0,
190: (8)     angle=TAU / 4,

```

```

191: (8)             n_components=8,
192: (8)             arc_center=ORIGIN,
193: (8)             **kwargs,
194: (4)             ):
195: (8)             self.start_angle = start_angle
196: (8)             self.angle = angle
197: (8)             self.radius = radius
198: (8)             self.n_components = n_components
199: (8)             self.arc_center = arc_center
200: (8)             super().__init__(self, **kwargs)
201: (8)             self.orientation = -1
202: (4)             def init_points(self):
203: (8)                 self.set_points(
204: (12)                     OpenGLArc.create_quadratic_bezier_points(
205: (16)                         angle=self.angle,
206: (16)                         start_angle=self.start_angle,
207: (16)                         n_components=self.n_components,
208: (12)                         ),
209: (8)                     )
210: (8)                     self.scale(self.radius, about_point=ORIGIN)
211: (8)                     self.shift(self.arc_center)
212: (4)             @staticmethod
213: (4)             def create_quadratic_bezier_points(angle, start_angle=0, n_components=8):
214: (8)                 samples = np.array(
215: (12)                     [
216: (16)                         [np.cos(a), np.sin(a), 0]
217: (16)                         for a in np.linspace(
218: (20)                             start_angle,
219: (20)                             start_angle + angle,
220: (20)                             2 * n_components + 1,
221: (16)                         )
222: (12)                     ],
223: (8)                 )
224: (8)                 theta = angle / n_components
225: (8)                 samples[1::2] /= np.cos(theta / 2)
226: (8)                 points = np.zeros((3 * n_components, 3))
227: (8)                 points[0::3] = samples[0:-1:2]
228: (8)                 points[1::3] = samples[1::2]
229: (8)                 points[2::3] = samples[2::2]
230: (8)                 return points
231: (4)             def get_arc_center(self):
232: (8)                 """
233: (8)                     Looks at the normals to the first two
234: (8)                     anchors, and finds their intersection points
235: (8)                 """
236: (8)                 a1, h, a2 = self.points[:3]
237: (8)                 t1 = h - a1
238: (8)                 t2 = h - a2
239: (8)                 n1 = rotate_vector(t1, TAU / 4)
240: (8)                 n2 = rotate_vector(t2, TAU / 4)
241: (8)                 return find_intersection(a1, n1, a2, n2)
242: (4)             def get_start_angle(self):
243: (8)                 angle = angle_of_vector(self.get_start() - self.get_arc_center())
244: (8)                 return angle % TAU
245: (4)             def get_stop_angle(self):
246: (8)                 angle = angle_of_vector(self.get_end() - self.get_arc_center())
247: (8)                 return angle % TAU
248: (4)             def move_arc_center_to(self, point):
249: (8)                 self.shift(point - self.get_arc_center())
250: (8)                 return self
251: (0)             class OpenGLArcBetweenPoints(OpenGLArc):
252: (4)                 def __init__(self, start, end, angle=TAU / 4, **kwargs):
253: (8)                     super().__init__(angle=angle, **kwargs)
254: (8)                     if angle == 0:
255: (12)                         self.set_points_as_corners([LEFT, RIGHT])
256: (8)                         self.put_start_and_end_on(start, end)
257: (0)             class OpenGLCurvedArrow(OpenGLArcBetweenPoints):
258: (4)                 def __init__(self, start_point, end_point, **kwargs):
259: (8)                     super().__init__(start_point, end_point, **kwargs)

```

```

260: (8)             self.add_tip()
261: (0)         class OpenGLCurvedDoubleArrow(OpenGLCurvedArrow):
262: (4)             def __init__(self, start_point, end_point, **kwargs):
263: (8)                 super().__init__(start_point, end_point, **kwargs)
264: (8)                 self.add_tip(at_start=True)
265: (0)         class OpenGLCircle(OpenGLArc):
266: (4)             def __init__(self, color=RED, **kwargs):
267: (8)                 super().__init__(0, TAU, color=color, **kwargs)
268: (4)             def surround(self, mobject, dim_to_match=0, stretch=False,
buff=MED_SMALL_BUFF):
269: (8)                     self.replace(mobject, dim_to_match, stretch)
270: (8)                     self.stretch((self.get_width() + 2 * buff) / self.get_width(), 0)
271: (8)                     self.stretch((self.get_height() + 2 * buff) / self.get_height(), 1)
272: (4)             def point_at_angle(self, angle):
273: (8)                 start_angle = self.get_start_angle()
274: (8)                 return self.point_from_proportion((angle - start_angle) / TAU)
275: (0)         class OpenGLDot(OpenGLCircle):
276: (4)             def __init__(
277: (8)                 self,
278: (8)                 point=ORIGIN,
279: (8)                 radius=DEFAULT_DOT_RADIUS,
280: (8)                 stroke_width=0,
281: (8)                 fill_opacity=1.0,
282: (8)                 color=WHITE,
283: (8)                 **kwargs,
284: (4)             ):
285: (8)                 super().__init__(
286: (12)                     arc_center=point,
287: (12)                     radius=radius,
288: (12)                     stroke_width=stroke_width,
289: (12)                     fill_opacity=fill_opacity,
290: (12)                     color=color,
291: (12)                     **kwargs,
292: (8)             )
293: (0)         class OpenGLEllipse(OpenGLCircle):
294: (4)             def __init__(self, width=2, height=1, **kwargs):
295: (8)                 super().__init__(**kwargs)
296: (8)                 self.set_width(width, stretch=True)
297: (8)                 self.set_height(height, stretch=True)
298: (0)         class OpenGLAnnularSector(OpenGLArc):
299: (4)             def __init__(
300: (8)                 self,
301: (8)                 inner_radius=1,
302: (8)                 outer_radius=2,
303: (8)                 angle=TAU / 4,
304: (8)                 start_angle=0,
305: (8)                 fill_opacity=1,
306: (8)                 stroke_width=0,
307: (8)                 color=WHITE,
308: (8)                 **kwargs,
309: (4)             ):
310: (8)                 self.inner_radius = inner_radius
311: (8)                 self.outer_radius = outer_radius
312: (8)                 super().__init__(
313: (12)                     start_angle=start_angle,
314: (12)                     angle=angle,
315: (12)                     fill_opacity=fill_opacity,
316: (12)                     stroke_width=stroke_width,
317: (12)                     color=color,
318: (12)                     **kwargs,
319: (8)             )
320: (4)             def init_points(self):
321: (8)                 inner_arc, outer_arc = (
322: (12)                     OpenGLArc(
323: (16)                         start_angle=self.start_angle,
324: (16)                         angle=self.angle,
325: (16)                         radius=radius,
326: (16)                         arc_center=self.arc_center,
327: (12)                     )

```

```

328: (12)             for radius in (self.inner_radius, self.outer_radius)
329: (8)         )
330: (8)         outer_arc.reverse_points()
331: (8)         self.append_points(inner_arc.points)
332: (8)         self.add_line_to(outer_arc.points[0])
333: (8)         self.append_points(outer_arc.points)
334: (8)         self.add_line_to(inner_arc.points[0])
335: (0)     class OpenGLSector(OpenGLAnnularSector):
336: (4)         def __init__(self, outer_radius=1, inner_radius=0, **kwargs):
337: (8)             super().__init__(inner_radius=inner_radius, outer_radius=outer_radius,
338: (0)             **kwargs)
339: (4)     class OpenGLAnnulus(OpenGLCircle):
340: (8)         def __init__(
341: (8)             self,
342: (8)             inner_radius=1,
343: (8)             outer_radius=2,
344: (8)             fill_opacity=1,
345: (8)             stroke_width=0,
346: (8)             color=WHITE,
347: (8)             mark_paths_closed=False,
348: (4)             **kwargs,
349: (8)         ):
350: (8)             self.mark_paths_closed = mark_paths_closed # is this even used?
351: (8)             self.inner_radius = inner_radius
352: (8)             self.outer_radius = outer_radius
353: (12)             super().__init__(
354: (8)                 fill_opacity=fill_opacity, stroke_width=stroke_width, color=color,
355: (4)             **kwargs
356: (8)         )
357: (8)         def init_points(self):
358: (8)             self.radius = self.outer_radius
359: (8)             outer_circle = OpenGLCircle(radius=self.outer_radius)
360: (8)             inner_circle = OpenGLCircle(radius=self.inner_radius)
361: (8)             inner_circle.reverse_points()
362: (8)             self.append_points(outer_circle.points)
363: (0)             self.append_points(inner_circle.points)
364: (8)             self.shift(self.arc_center)
365: (0)     class OpenGLLine(OpenGLTipableVMOBJECT):
366: (4)         def __init__(self, start=LEFT, end=RIGHT, buff=0, path_arc=0, **kwargs):
367: (8)             self.dim = 3
368: (8)             self.buff = buff
369: (8)             self.path_arc = path_arc
370: (4)             self.set_start_and_end_attrs(start, end)
371: (8)             super().__init__(**kwargs)
372: (4)         def init_points(self):
373: (8)             self.set_points_by_ends(self.start, self.end, self.buff,
374: (12)             self.path_arc)
375: (12)             self.set_points(OpenGLArc.create_quadratic_bezier_points(path_arc))
376: (8)             self.put_start_and_end_on(start, end)
377: (12)             else:
378: (8)                 self.set_points_as_corners([start, end])
379: (4)             self.account_for_buff(self.buff)
380: (8)         def set_path_arc(self, new_value):
381: (8)             self.path_arc = new_value
382: (4)             self.init_points()
383: (8)         def account_for_buff(self, buff):
384: (12)             if buff == 0:
385: (8)                 return
386: (12)             if self.path_arc == 0:
387: (8)                 length = self.get_length()
388: (12)             else:
389: (8)                 length = self.get_arc_length()
390: (12)             if length < 2 * buff:
391: (8)                 return
392: (8)             buff_prop = buff / length

```

```

393: (8)             return self
394: (4)         def set_start_and_end_attrs(self, start, end):
395: (8)             rough_start = self.pointify(start)
396: (8)             rough_end = self.pointify(end)
397: (8)             vect = normalize(rough_end - rough_start)
398: (8)             self.start = self.pointify(start, vect) + self.buff * vect
399: (8)             self.end = self.pointify(end, -vect) - self.buff * vect
400: (4)         def pointify(self, mob_or_point, direction=None):
401: (8)             """
402: (8)             Take an argument passed into Line (or subclass) and turn
403: (8)             it into a 3d point.
404: (8)             """
405: (8)             if isinstance(mob_or_point, Mobject):
406: (12)                 mob = mob_or_point
407: (12)                 if direction is None:
408: (16)                     return mob.get_center()
409: (12)                 else:
410: (16)                     return mob.get_continuous_bounding_box_point(direction)
411: (8)             else:
412: (12)                 point = mob_or_point
413: (12)                 result = np.zeros(self.dim)
414: (12)                 result[: len(point)] = point
415: (12)                 return result
416: (4)         def put_start_and_end_on(self, start, end):
417: (8)             curr_start, curr_end = self.get_start_and_end()
418: (8)             if (curr_start == curr_end).all():
419: (12)                 self.set_points_by_ends(start, end, self.path_arc)
420: (8)             return super().put_start_and_end_on(start, end)
421: (4)         def get_vector(self):
422: (8)             return self.get_end() - self.get_start()
423: (4)         def get_unit_vector(self):
424: (8)             return normalize(self.get_vector())
425: (4)         def get_angle(self):
426: (8)             return angle_of_vector(self.get_vector())
427: (4)         def get_projection(self, point):
428: (8)             """
429: (8)             Return projection of a point onto the line
430: (8)             """
431: (8)             unit_vect = self.get_unit_vector()
432: (8)             start = self.get_start()
433: (8)             return start + np.dot(point - start, unit_vect) * unit_vect
434: (4)         def get_slope(self):
435: (8)             return np.tan(self.get_angle())
436: (4)         def set_angle(self, angle, about_point=None):
437: (8)             if about_point is None:
438: (12)                 about_point = self.get_start()
439: (8)             self.rotate(
440: (12)                 angle - self.get_angle(),
441: (12)                 about_point=about_point,
442: (8)             )
443: (8)             return self
444: (4)         def set_length(self, length):
445: (8)             self.scale(length / self.get_length())
446: (0)     class OpenGLDashedLine(OpenGLLine):
447: (4)         def __init__(
448: (8)             self, *args, dash_length=DEFAULT_DASH_LENGTH, dashed_ratio=0.5,
449: (4)             **kwargs
450: (8)         ):
451: (8)             self.dashed_ratio = dashed_ratio
452: (8)             self.dash_length = dash_length
453: (8)             super().__init__(*args, **kwargs)
454: (8)             dashed_ratio = self.dashed_ratio
455: (8)             num_dashes = self.calculate_num_dashes(dashed_ratio)
456: (12)             dashes = OpenGLDashedVMobject(
457: (12)                 self,
458: (12)                 num_dashes=num_dashes,
459: (8)                 dashed_ratio=dashed_ratio,
460: (8)             )

```

```

461: (8)                     self.add(*dashes)
462: (4)             def calculate_num_dashes(self, dashed_ratio):
463: (8)                 return max(
464: (12)                     2,
465: (12)                     int(np.ceil((self.get_length() / self.dash_length) *
dashed_ratio)),
466: (8)                 )
467: (4)             def get_start(self):
468: (8)                 if len(self.submobjects) > 0:
469: (12)                     return self.submobjects[0].get_start()
470: (8)                 else:
471: (12)                     return super().get_start()
472: (4)             def get_end(self):
473: (8)                 if len(self.submobjects) > 0:
474: (12)                     return self.submobjects[-1].get_end()
475: (8)                 else:
476: (12)                     return super().get_end()
477: (4)             def get_first_handle(self):
478: (8)                 return self.submobjects[0].points[1]
479: (4)             def get_last_handle(self):
480: (8)                 return self.submobjects[-1].points[-2]
481: (0)         class OpenGLTangentLine(OpenGLLine):
482: (4)             def __init__(self, vmob, alpha, length=1, d_alpha=1e-6, **kwargs):
483: (8)                 self.length = length
484: (8)                 self.d_alpha = d_alpha
485: (8)                 da = self.d_alpha
486: (8)                 a1 = clip(alpha - da, 0, 1)
487: (8)                 a2 = clip(alpha + da, 0, 1)
488: (8)                 super().__init__(vmob.pfp(a1), vmob.pfp(a2), **kwargs)
489: (8)                 self.scale(self.length / self.get_length())
490: (0)         class OpenGLElbow(OpenGLVMOobject):
491: (4)             def __init__(self, width=0.2, angle=0, **kwargs):
492: (8)                 self.angle = angle
493: (8)                 super().__init__(**kwargs)
494: (8)                 self.set_points_as_corners([UP, UP + RIGHT, RIGHT])
495: (8)                 self.set_width(width, about_point=ORIGIN)
496: (8)                 self.rotate(self.angle, about_point=ORIGIN)
497: (0)         class OpenGLArrow(OpenGLLine):
498: (4)             def __init__(
499: (8)                 self,
500: (8)                 start=LEFT,
501: (8)                 end=RIGHT,
502: (8)                 path_arc=0,
503: (8)                 fill_color=GREY_A,
504: (8)                 fill_opacity=1,
505: (8)                 stroke_width=0,
506: (8)                 buff=MED_SMALL_BUFF,
507: (8)                 thickness=0.05,
508: (8)                 tip_width_ratio=5,
509: (8)                 tip_angle=PI / 3,
510: (8)                 max_tip_length_to_length_ratio=0.5,
511: (8)                 max_width_to_length_ratio=0.1,
512: (8)                 **kwargs,
513: (4)             ):
514: (8)                 self.thickness = thickness
515: (8)                 self.tip_width_ratio = tip_width_ratio
516: (8)                 self.tip_angle = tip_angle
517: (8)                 self.max_tip_length_to_length_ratio = max_tip_length_to_length_ratio
518: (8)                 self.max_width_to_length_ratio = max_width_to_length_ratio
519: (8)                 super().__init__(
520: (12)                     start=start,
521: (12)                     end=end,
522: (12)                     buff=buff,
523: (12)                     path_arc=path_arc,
524: (12)                     fill_color=fill_color,
525: (12)                     fill_opacity=fill_opacity,
526: (12)                     stroke_width=stroke_width,
527: (12)                     **kwargs,
528: (8)             )

```

```

529: (4)             def set_points_by_ends(self, start, end, buff=0, path_arc=0):
530: (8)                 vect = end - start
531: (8)                 length = max(np.linalg.norm(vect), 1e-8)
532: (8)                 thickness = self.thickness
533: (8)                 w_ratio = self.max_width_to_length_ratio / (thickness / length)
534: (8)                 if w_ratio < 1:
535: (12)                     thickness *= w_ratio
536: (8)                 tip_width = self.tip_width_ratio * thickness
537: (8)                 tip_length = tip_width / (2 * np.tan(self.tip_angle / 2))
538: (8)                 t_ratio = self.max_tip_length_to_length_ratio / (tip_length / length)
539: (8)                 if t_ratio < 1:
540: (12)                     tip_length *= t_ratio
541: (12)                     tip_width *= t_ratio
542: (8)             if path_arc == 0:
543: (12)                 points1 = (length - tip_length) * np.array([RIGHT, 0.5 * RIGHT,
ORIGIN])
544: (12)             points1 += thickness * UP / 2
545: (12)             points2 = points1[::-1] + thickness * DOWN
546: (8)         else:
547: (12)             a = 2 * (1 - np.cos(path_arc))
548: (12)             b = -2 * tip_length * np.sin(path_arc)
549: (12)             c = tip_length**2 - length**2
550: (12)             R = (-b + np.sqrt(b**2 - 4 * a * c)) / (2 * a)
551: (12)             points1 = OpenGLArc.create_quadratic_bezier_points(path_arc)
552: (12)             points2 = np.array(points1[::-1])
553: (12)             points1 *= R + thickness / 2
554: (12)             points2 *= R - thickness / 2
555: (12)             if path_arc < 0:
556: (16)                 tip_length *= -1
557: (12)             rot_T = rotation_matrix_transpose(PI / 2 - path_arc, OUT)
558: (12)             for points in points1, points2:
559: (16)                 points[:] = np.dot(points, rot_T)
560: (16)                 points += R * DOWN
561: (8)             self.set_points(points1)
562: (8)             self.add_line_to(tip_width * UP / 2)
563: (8)             self.add_line_to(tip_length * LEFT)
564: (8)             self.tip_index = len(self.points) - 1
565: (8)             self.add_line_to(tip_width * DOWN / 2)
566: (8)             self.add_line_to(points2[0])
567: (8)             self.append_points(points2)
568: (8)             self.add_line_to(points1[0])
569: (8)             if length > 0:
570: (12)                 super().scale(length / self.get_length())
571: (8)             self.rotate(angle_of_vector(vect) - self.get_angle())
572: (8)             self.rotate(
573: (12)                 PI / 2 - np.arccos(normalize(vect)[2]),
574: (12)                 axis=rotate_vector(self.get_unit_vector(), -PI / 2),
575: (8)             )
576: (8)             self.shift(start - self.get_start())
577: (8)             self.refresh_triangulation()
578: (4)         def reset_points_around_ends(self):
579: (8)             self.set_points_by_ends(
580: (12)                 self.get_start(),
581: (12)                 self.get_end(),
582: (12)                 path_arc=self.path_arc,
583: (8)             )
584: (8)             return self
585: (4)         def get_start(self):
586: (8)             nppc = self.n_points_per_curve
587: (8)             points = self.points
588: (8)             return (points[0] + points[-nppc]) / 2
589: (4)         def get_end(self):
590: (8)             return self.points[self.tip_index]
591: (4)         def put_start_and_end_on(self, start, end):
592: (8)             self.set_points_by_ends(start, end, buff=0, path_arc=self.path_arc)
593: (8)             return self
594: (4)         def scale(self, *args, **kwargs):
595: (8)             super().scale(*args, **kwargs)
596: (8)             self.reset_points_around_ends()

```

```

597: (8)             return self
598: (4)         def set_thickness(self, thickness):
599: (8)             self.thickness = thickness
600: (8)             self.reset_points_around_ends()
601: (8)             return self
602: (4)         def set_path_arc(self, path_arc):
603: (8)             self.path_arc = path_arc
604: (8)             self.reset_points_around_ends()
605: (8)             return self
606: (0)     class OpenGLVector(OpenGLArrow):
607: (4)         def __init__(self, direction=RIGHT, buff=0, **kwargs):
608: (8)             self.buff = buff
609: (8)             if len(direction) == 2:
610: (12)                 direction = np.hstack([direction, 0])
611: (8)             super().__init__(ORIGIN, direction, buff=buff, **kwargs)
612: (0)     class OpenGLDoubleArrow(OpenGLArrow):
613: (4)         def __init__(self, *args, **kwargs):
614: (8)             super().__init__(*args, **kwargs)
615: (8)             self.add_tip(at_start=True)
616: (0)     class OpenGLCubicBezier(OpenGLMobject):
617: (4)         def __init__(self, a0, h0, h1, a1, **kwargs):
618: (8)             super().__init__(**kwargs)
619: (8)             self.add_cubic_bezier_curve(a0, h0, h1, a1)
620: (0)     class OpenGLPolygon(OpenGLMobject):
621: (4)         def __init__(self, *vertices, **kwargs):
622: (8)             self.vertices = vertices
623: (8)             super().__init__(**kwargs)
624: (4)         def init_points(self):
625: (8)             verts = self.vertices
626: (8)             self.set_points_as_corners([*verts, verts[0]])
627: (4)         def get_vertices(self):
628: (8)             return self.get_start_anchors()
629: (4)         def round_corners(self, radius=0.5):
630: (8)             vertices = self.get_vertices()
631: (8)             arcs = []
632: (8)             for v1, v2, v3 in adjacent_n_tuples(vertices, 3):
633: (12)                 vect1 = v2 - v1
634: (12)                 vect2 = v3 - v2
635: (12)                 unit_vect1 = normalize(vect1)
636: (12)                 unit_vect2 = normalize(vect2)
637: (12)                 angle = angle_between_vectors(vect1, vect2)
638: (12)                 angle *= np.sign(radius)
639: (12)                 cut_off_length = radius * np.tan(angle / 2)
640: (12)                 sign = np.sign(np.cross(vect1, vect2)[2])
641: (12)                 arc = OpenGLArcBetweenPoints(
642: (16)                     v2 - unit_vect1 * cut_off_length,
643: (16)                     v2 + unit_vect2 * cut_off_length,
644: (16)                     angle=sign * angle,
645: (16)                     n_components=2,
646: (12)                 )
647: (12)                 arcs.append(arc)
648: (8)             self.clear_points()
649: (8)             arcs = [arcs[-1], *arcs[:-1]]
650: (8)             for arc1, arc2 in adjacent_pairs(arcs):
651: (12)                 self.append_points(arc1.points)
652: (12)                 line = OpenGLLine(arc1.get_end(), arc2.get_start())
653: (12)                 len_ratio = line.get_length() / arc1.get_arc_length()
654: (12)                 line.insert_n_curves(int(arc1.get_num_curves() * len_ratio))
655: (12)                 self.append_points(line.points)
656: (8)             return self
657: (0)     class OpenGLRegularPolygon(OpenGLPolygon):
658: (4)         def __init__(self, n=6, start_angle=None, **kwargs):
659: (8)             self.start_angle = start_angle
660: (8)             if self.start_angle is None:
661: (12)                 if n % 2 == 0:
662: (16)                     self.start_angle = 0
663: (12)                 else:
664: (16)                     self.start_angle = 90 * DEGREES
665: (8)             start_vect = rotate_vector(RIGHT, self.start_angle)

```

```

666: (8)             vertices = compass_directions(n, start_vect)
667: (8)             super().__init__(vertices, **kwargs)
668: (0)         class OpenGLTriangle(OpenGLRegularPolygon):
669: (4)             def __init__(self, **kwargs):
670: (8)                 super().__init__(n=3, **kwargs)
671: (0)         class OpenGLArrowTip(OpenGLTriangle):
672: (4)             def __init__(
673: (8)                 self,
674: (8)                 fill_opacity=1,
675: (8)                 fill_color=WHITE,
676: (8)                 stroke_width=0,
677: (8)                 width=DEFAULT_ARROW_TIP_WIDTH,
678: (8)                 length=DEFAULT_ARROW_TIP_LENGTH,
679: (8)                 angle=0,
680: (8)                 **kwargs,
681: (4)             ):
682: (8)                 super().__init__(
683: (12)                     start_angle=0,
684: (12)                     fill_opacity=fill_opacity,
685: (12)                     fill_color=fill_color,
686: (12)                     stroke_width=stroke_width,
687: (12)                     **kwargs,
688: (8)             )
689: (8)             self.set_width(width, stretch=True)
690: (8)             self.set_height(length, stretch=True)
691: (4)         def get_base(self):
692: (8)             return self.point_from_proportion(0.5)
693: (4)         def get_tip_point(self):
694: (8)             return self.points[0]
695: (4)         def get_vector(self):
696: (8)             return self.get_tip_point() - self.get_base()
697: (4)         def get_angle(self):
698: (8)             return angle_of_vector(self.get_vector())
699: (4)         def get_length(self):
700: (8)             return np.linalg.norm(self.get_vector())
701: (0)     class OpenGLRectangle(OpenGLPolygon):
702: (4)         def __init__(self, color=WHITE, width=4.0, height=2.0, **kwargs):
703: (8)             super().__init__(UR, UL, DL, DR, color=color, **kwargs)
704: (8)             self.set_width(width, stretch=True)
705: (8)             self.set_height(height, stretch=True)
706: (0)     class OpenGLSquare(OpenGLRectangle):
707: (4)         def __init__(self, side_length=2.0, **kwargs):
708: (8)             self.side_length = side_length
709: (8)             super().__init__(height=side_length, width=side_length, **kwargs)
710: (0)     class OpenGLRoundedRectangle(OpenGLRectangle):
711: (4)         def __init__(self, corner_radius=0.5, **kwargs):
712: (8)             self.corner_radius = corner_radius
713: (8)             super().__init__(**kwargs)
714: (8)             self.round_corners(self.corner_radius)

```

-----

## File 91 - three\_dimensions.py:

```

1: (0)             """Three-dimensional mobjects."""
2: (0)             from __future__ import annotations
3: (0)             from manim.typing import Point3D, Vector3D
4: (0)             from manim.utils.color import BLUE, BLUE_D, BLUE_E, LIGHT_GREY, WHITE,
interpolate_color
5: (0)             __all__ = [
6: (4)                 "ThreeDVMobject",
7: (4)                 "Surface",
8: (4)                 "Sphere",
9: (4)                 "Dot3D",
10: (4)                "Cube",
11: (4)                "Prism",
12: (4)                "Cone",
13: (4)                "Arrow3D",
14: (4)                "Cylinder",

```

```

15: (4)             "Line3D",
16: (4)             "Torus",
17: (0)
18: (0)         from typing import Any, Callable, Iterable, Sequence
19: (0)         import numpy as np
20: (0)         from typing_extensions import Self
21: (0)         from manim import config, logger
22: (0)         from manim.constants import *
23: (0)         from manim.mobject.geometry.arc import Circle
24: (0)         from manim.mobject.geometry.polygram import Square
25: (0)         from manim.mobject.mobject import *
26: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
27: (0)         from manim.mobject.opengl.opengl_mobject import OpenGLMobject
28: (0)         from manim.mobject.types.vectorized_mobject import VGroup, VМобјект
29: (0)         from manim.utils.color import (
30: (4)             BLUE,
31: (4)             BLUE_D,
32: (4)             BLUE_E,
33: (4)             LIGHT_GREY,
34: (4)             WHITE,
35: (4)             ManimColor,
36: (4)             ParsableManimColor,
37: (4)             interpolate_color,
38: (0)
39: (0)
40: (0)
z_to_vector
41: (0)     class ThreeDVMobject(VMobject, metaclass=ConvertToOpenGL):
42: (4)         def __init__(self, shade_in_3d: bool = True, **kwargs):
43: (8)             super().__init__(shade_in_3d=shade_in_3d, **kwargs)
44: (0)     class Surface(VGroup, metaclass=ConvertToOpenGL):
45: (4)         """Creates a Parametric Surface using a checkerboard pattern.
46: (4)         Parameters
47: (4)         -----
48: (4)         func
49: (8)             The function defining the :class:`Surface`.
50: (4)         u_range
51: (8)             The range of the ``u`` variable: ``

```

```

82: (16)                     surface = Surface(
83: (20)                         lambda u, v: axes.c2p(*self.func(u, v)),
84: (20)                         u_range=[-PI, PI],
85: (20)                         v_range=[0, TAU],
86: (20)                         resolution=8,
87: (16)                     )
88: (16)             self.set_camera_orientation(theta=70 * DEGREES, phi=75 *
DEGREES)
89: (16)                 self.add(axes, surface)
90: (4) """
91: (4)     def __init__(
92: (8)         self,
93: (8)         func: Callable[[float, float], np.ndarray],
94: (8)         u_range: Sequence[float] = [0, 1],
95: (8)         v_range: Sequence[float] = [0, 1],
96: (8)         resolution: Sequence[int] = 32,
97: (8)         surface_piece_config: dict = {},
98: (8)         fill_color: ParsableManimColor = BLUE_D,
99: (8)         fill_opacity: float = 1.0,
100: (8)         checkerboard_colors: Sequence[ParsableManimColor] | bool = [BLUE_D,
BLUE_E],
101: (8)         stroke_color: ParsableManimColor = LIGHT_GREY,
102: (8)         stroke_width: float = 0.5,
103: (8)         should_make_jagged: bool = False,
104: (8)         pre_function_handle_to_anchor_scale_factor: float = 0.00001,
105: (8)         **kwargs: Any,
106: (4)     ) -> None:
107: (8)         self.u_range = u_range
108: (8)         self.v_range = v_range
109: (8)         super().__init__(**kwargs)
110: (8)         self.resolution = resolution
111: (8)         self.surface_piece_config = surface_piece_config
112: (8)         self.fill_color: ManimColor = ManimColor(fill_color)
113: (8)         self.fill_opacity = fill_opacity
114: (8)         if checkerboard_colors:
115: (12)             self.checkerboard_colors: list[ManimColor] = [
116: (16)                 ManimColor(x) for x in checkerboard_colors
117: (12)             ]
118: (8)         else:
119: (12)             self.checkerboard_colors = checkerboard_colors
120: (8)             self.stroke_color: ManimColor = ManimColor(stroke_color)
121: (8)             self.stroke_width = stroke_width
122: (8)             self.should_make_jagged = should_make_jagged
123: (8)             self.pre_function_handle_to_anchor_scale_factor = (
124: (12)                 pre_function_handle_to_anchor_scale_factor
125: (8)             )
126: (8)             self._func = func
127: (8)             self._setup_in_uv_space()
128: (8)             self.apply_function(lambda p: func(p[0], p[1]))
129: (8)             if self.should_make_jagged:
130: (12)                 self.make_jagged()
131: (4)     def func(self, u: float, v: float) -> np.ndarray:
132: (8)         return self._func(u, v)
133: (4)     def _get_u_values_and_v_values(self) -> tuple[np.ndarray, np.ndarray]:
134: (8)         res = tuplify(self.resolution)
135: (8)         if len(res) == 1:
136: (12)             u_res = v_res = res[0]
137: (8)         else:
138: (12)             u_res, v_res = res
139: (8)             u_values = np.linspace(*self.u_range, u_res + 1)
140: (8)             v_values = np.linspace(*self.v_range, v_res + 1)
141: (8)             return u_values, v_values
142: (4)     def _setup_in_uv_space(self) -> None:
143: (8)         u_values, v_values = self._get_u_values_and_v_values()
144: (8)         faces = VGroup()
145: (8)         for i in range(len(u_values) - 1):
146: (12)             for j in range(len(v_values) - 1):
147: (16)                 u1, u2 = u_values[i : i + 2]
148: (16)                 v1, v2 = v_values[j : j + 2]

```

```

149: (16)                         face = ThreeDVMobject()
150: (16)                         face.set_points_as_corners(
151: (20)                           [
152: (24)                             [u1, v1, 0],
153: (24)                             [u2, v1, 0],
154: (24)                             [u2, v2, 0],
155: (24)                             [u1, v2, 0],
156: (24)                             [u1, v1, 0],
157: (20)                           ],
158: (16)                         )
159: (16)                         faces.add(face)
160: (16)                         face.u_index = i
161: (16)                         face.v_index = j
162: (16)                         face.u1 = u1
163: (16)                         face.u2 = u2
164: (16)                         face.v1 = v1
165: (16)                         face.v2 = v2
166: (8)                          faces.set_fill(color=self.fill_color, opacity=self.fill_opacity)
167: (8)                          faces.set_stroke(
168: (12)                            color=self.stroke_color,
169: (12)                            width=self.stroke_width,
170: (12)                            opacity=self.stroke_opacity,
171: (8)                         )
172: (8)                         self.add(*faces)
173: (8)                         if self.checkerboard_colors:
174: (12)                           self.set_fill_by_checkerboard(*self.checkerboard_colors)
175: (4)                          def set_fill_by_checkerboard(
176: (8)                            self, *colors: Iterable[ParsableManimColor], opacity: float | None =
None
177: (4)                          ) -> Self:
178: (8)                            """Sets the fill_color of each face of :class:`Surface` in
179: (8)                            an alternating pattern.
180: (8)                            Parameters
181: (8)                            -----
182: (8)                            colors
183: (12)                           List of colors for alternating pattern.
184: (8)                            opacity
185: (12)                           The fill_opacity of :class:`Surface`, from 0 being fully
transparent
186: (12)                           to 1 being fully opaque.
187: (8)                          Returns
188: (8)                          -----
189: (8)                          :class:`~.Surface`
190: (12)                           The parametric surface with an alternating pattern.
191: (8)                          """
192: (8)                          n_colors = len(colors)
193: (8)                          for face in self:
194: (12)                            c_index = (face.u_index + face.v_index) % n_colors
195: (12)                            face.set_fill(colors[c_index], opacity=opacity)
196: (8)                          return self
197: (4)                          def set_fill_by_value(
198: (8)                            self,
199: (8)                            axes: Mobject,
200: (8)                            colorscale: list[ParsableManimColor] | ParsableManimColor | None =
None,
201: (8)                            axis: int = 2,
202: (8)                            **kwargs,
203: (4)                          ) -> Self:
204: (8)                            """Sets the color of each mobject of a parametric surface to a color
205: (8)                            relative to its axis-value.
206: (8)                            Parameters
207: (8)                            -----
208: (8)                            axes
209: (12)                           The axes for the parametric surface, which will be used to map
210: (12)                           axis-values to colors.
211: (8)                            colorscale
212: (12)                           A list of colors, ordered from lower axis-values to higher axis-
values.
213: (12)                           If a list of tuples is passed containing colors paired with

```

```

numbers,
214: (12)           then those numbers will be used as the pivots.
215: (8)
216: (12)           axis
217: (8)             The chosen axis to use for the color mapping. (0 = x, 1 = y, 2 =
z)
218: (8)
219: (8)
220: (12)           Returns
221: (8)             -----
222: (8)             :class:`~.Surface`
223: (8)               The parametric surface with a gradient applied by value. For
chaining.
224: (8)
225: (12)
226: (16)
227: (20)
228: (20)
DEGREES)
229: (20)
z_range=(-1, 1, 0.5))
230: (20)
231: (24)
232: (24)
233: (24)
234: (24)
235: (20)
236: (24)
237: (24)
238: (24)
239: (24)
240: (24)
241: (20)
242: (20)
[RED, -0.5), (YELLOW, 0), (GREEN, 0.5)], axis=2)
243: (20)           self.add(axes, surface_plane)
244: (8)
245: (8)           """
246: (12)           if "colors" in kwargs and colorscale is None:
247: (12)             colorscale = kwargs.pop("colors")
248: (16)             if kwargs:
249: (20)               raise ValueError(
250: (20)                 "Unsupported keyword argument(s): "
251: (16)                   f"{{', '.join(str(key) for key in kwargs)}}"
252: (8)
253: (12)
254: (16)
"
255: (16)
256: (12)
257: (12)
258: (8)
259: (8)
260: (12)
261: (16)
262: (16)
263: (12)
264: (8)
265: (12)
266: (12)
267: (12)
268: (12)
269: (12)
270: (16)
271: (16)
272: (16)
273: (12)
274: (8)
275: (12)
for mob in self.family_members_with_points():
    axis_value = axes.point_to_coords(mob.get_midpoint())[axis]

```

```

276: (12)             if axis_value <= pivots[0]:
277: (16)                 mob.set_color(new_colors[0])
278: (12)             elif axis_value >= pivots[-1]:
279: (16)                 mob.set_color(new_colors[-1])
280: (12)         else:
281: (16)             for i, pivot in enumerate(pivots):
282: (20)                 if pivot > axis_value:
283: (24)                     color_index = (axis_value - pivots[i - 1]) / (
284: (28)                         pivots[i] - pivots[i - 1]
285: (24)                     )
286: (24)                     color_index = min(color_index, 1)
287: (24)                     mob_color = interpolate_color(
288: (28)                         new_colors[i - 1],
289: (28)                         new_colors[i],
290: (28)                         color_index,
291: (24)                     )
292: (24)                     if config.renderer == RendererType.OPENGL:
293: (28)                         mob.set_color(mob_color, recurse=False)
294: (24)                     elif config.renderer == RendererType.CAIRO:
295: (28)                         mob.set_color(mob_color, family=False)
296: (24)                     break
297: (8)             return self
298: (0) class Sphere(Surface):
299: (4)     """A three-dimensional sphere.
300: (4)     Parameters
301: (4)     -----
302: (4)     center
303: (8)         Center of the :class:`Sphere`.
304: (4)     radius
305: (8)         The radius of the :class:`Sphere`.
306: (4)     resolution
307: (8)         The number of samples taken of the :class:`Sphere`. A tuple can be
used
308: (8)         to define different resolutions for ``u`` and ``v`` respectively.
309: (4)     u_range
310: (8)         The range of the ``u`` variable: ````(u_min, u_max)````.
311: (4)     v_range
312: (8)         The range of the ``v`` variable: ````(v_min, v_max)````.
313: (4) Examples
314: (4) -----
315: (4) .. manim:: ExampleSphere
316: (8)     :save_last_frame:
317: (8)     class ExampleSphere(ThreeDScene):
318: (12)         def construct(self):
319: (16)             self.set_camera_orientation(phi=PI / 6, theta=PI / 6)
320: (16)             sphere1 = Sphere(
321: (20)                 center=(3, 0, 0),
322: (20)                 radius=1,
323: (20)                 resolution=(20, 20),
324: (20)                 u_range=[0.001, PI - 0.001],
325: (20)                 v_range=[0, TAU]
326: (16)             )
327: (16)             sphere1.set_color(RED)
328: (16)             self.add(sphere1)
329: (16)             sphere2 = Sphere(center=(-1, -3, 0), radius=2, resolution=(18,
18))
330: (16)             sphere2.set_color(GREEN)
331: (16)             self.add(sphere2)
332: (16)             sphere3 = Sphere(center=(-1, 2, 0), radius=2, resolution=(16,
16))
333: (16)             sphere3.set_color(BLUE)
334: (16)             self.add(sphere3)
335: (4)         """
336: (4)         def __init__(
337: (8)             self,
338: (8)             center: Point3D = ORIGIN,
339: (8)             radius: float = 1,
340: (8)             resolution: Sequence[int] | None = None,
341: (8)             u_range: Sequence[float] = (0, TAU),

```

```

342: (8)             v_range: Sequence[float] = (0, PI),
343: (8)             **kwargs,
344: (4)         ) -> None:
345: (8)             if config.renderer == RendererType.OPENGL:
346: (12)                 res_value = (101, 51)
347: (8)             elif config.renderer == RendererType.CAIRO:
348: (12)                 res_value = (24, 12)
349: (8)
350: (12)             else:
351: (8)                 raise Exception("Unknown renderer")
352: (8)             resolution = resolution if resolution is not None else res_value
353: (8)             self.radius = radius
354: (12)             super().__init__(
355: (12)                 self.func,
356: (12)                 resolution=resolution,
357: (12)                 u_range=u_range,
358: (12)                 v_range=v_range,
359: (8)                 **kwargs,
360: (8)             )
361: (4)             self.shift(center)
362: (8)         def func(self, u: float, v: float) -> np.ndarray:
363: (8)             """The z values defining the :class:`Sphere` being plotted.
364: (8)             Returns
365: (8)             -----
366: (12)                 :class:`numpy.array`
367: (8)                 The z values defining the :class:`Sphere`.
368: (8)             """
369: (12)             return self.radius * np.array(
370: (8)                 [np.cos(u) * np.sin(v), np.sin(u) * np.sin(v), -np.cos(v)],
371: (0)             )
372: (4)         class Dot3D(Sphere):
373: (4)             """A spherical dot.
374: (4)             Parameters
375: (4)             -----
376: (8)             point
377: (4)                 The location of the dot.
378: (8)             radius
379: (4)                 The radius of the dot.
380: (8)             color
381: (4)                 The color of the :class:`Dot3D`.
382: (8)             resolution
383: (8)                 The number of samples taken of the :class:`Dot3D`. A tuple can be
384: (8)                 used to define different resolutions for ``u`` and ``v`` respectively.
385: (4)             Examples
386: (4)             -----
387: (8)             .. manim:: Dot3DExample
388: (8)                 :save_last_frame:
389: (12)             class Dot3DExample(ThreeDScene):
390: (16)                 def construct(self):
391: (16)                     self.set_camera_orientation(phi=75*DEGREES, theta=-45*DEGREES)
392: (16)                     axes = ThreeDAxes()
393: (16)                     dot_1 = Dot3D(point=axes.coords_to_point(0, 0, 1), color=RED)
394: (16)                     dot_2 = Dot3D(point=axes.coords_to_point(2, 0, 0), radius=0.1,
395: (16)                         dot_3 = Dot3D(point=[0, 0, 0], radius=0.1, color=ORANGE)
396: (4)                         self.add(axes, dot_1, dot_2, dot_3)
397: (4)             """
398: (8)             def __init__(
399: (8)                 self,
400: (8)                 point: list | np.ndarray = ORIGIN,
401: (8)                 radius: float = DEFAULT_DOT_RADIUS,
402: (8)                 color: ParsableManimColor = WHITE,
403: (8)                 resolution: tuple[int, int] = (8, 8),
404: (8)                 **kwargs,
405: (4)             ) -> None:
406: (8)                 super().__init__(center=point, radius=radius, resolution=resolution,
407: (0)                     self.set_color(color)
408: (4)             """
409: (8)             :class:`VGroup`
410: (8)             """A three-dimensional cube.

```

```

409: (4)             Parameters
410: (4)             -----
411: (4)             side_length
412: (8)             Length of each side of the :class:`Cube`.
413: (4)             fill_opacity
414: (8)             The opacity of the :class:`Cube`, from 0 being fully transparent to 1
being
415: (8)             fully opaque. Defaults to 0.75.
416: (4)             fill_color
417: (8)             The color of the :class:`Cube`.
418: (4)             stroke_width
419: (8)             The width of the stroke surrounding each face of the :class:`Cube`.
420: (4)             Examples
421: (4)             -----
422: (4)             .. manim:: CubeExample
423: (8)             :save_last_frame:
424: (8)             class CubeExample(ThreeDScene):
425: (12)             def construct(self):
426: (16)                 self.set_camera_orientation(phi=75*DEGREES, theta=-45*DEGREES)
427: (16)                 axes = ThreeDAxes()
428: (16)                 cube = Cube(side_length=3, fill_opacity=0.7, fill_color=BLUE)
429: (16)                 self.add(cube)
430: (4)             """
431: (4)             def __init__(
432: (8)                 self,
433: (8)                 side_length: float = 2,
434: (8)                 fill_opacity: float = 0.75,
435: (8)                 fill_color: ParsableManimColor = BLUE,
436: (8)                 stroke_width: float = 0,
437: (8)                 **kwargs,
438: (4)             ) -> None:
439: (8)                 self.side_length = side_length
440: (8)                 super().__init__(
441: (12)                     fill_color=fill_color,
442: (12)                     fill_opacity=fill_opacity,
443: (12)                     stroke_width=stroke_width,
444: (12)                     **kwargs,
445: (8)             )
446: (4)             def generate_points(self) -> None:
447: (8)                 """Creates the sides of the :class:`Cube`."""
448: (8)                 for vect in IN, OUT, LEFT, RIGHT, UP, DOWN:
449: (12)                     face = Square(
450: (16)                         side_length=self.side_length,
451: (16)                         shade_in_3d=True,
452: (12)                     )
453: (12)                     face.flip()
454: (12)                     face.shift(self.side_length * OUT / 2.0)
455: (12)                     face.apply_matrix(z_to_vector(vect))
456: (12)                     self.add(face)
457: (4)             init_points = generate_points
458: (0)             class Prism(Cube):
459: (4)                 """A right rectangular prism (or rectangular cuboid).
460: (4)                 Defined by the length of each side in ``[x, y, z]`` format.
461: (4)                 Parameters
462: (4)                 -----
463: (4)                 dimensions
464: (8)                     Dimensions of the :class:`Prism` in ``[x, y, z]`` format.
465: (4)                 Examples
466: (4)                 -----
467: (4)                 .. manim:: ExamplePrism
468: (8)                 :save_last_frame:
469: (8)                 class ExamplePrism(ThreeDScene):
470: (12)                 def construct(self):
471: (16)                     self.set_camera_orientation(phi=60 * DEGREES, theta=150 *
DEGREES)
472: (16)                     prismSmall = Prism(dimensions=[1, 2, 3]).rotate(PI / 2)
473: (16)                     prismLarge = Prism(dimensions=[1.5, 3, 4.5]).move_to([2, 0,
0])
474: (16)                     self.add(prismSmall, prismLarge)

```

```

475: (4)             """
476: (4)         def __init__(self, dimensions: tuple[float, float, float] | np.ndarray = [3, 2, 1], **kwargs):
477: (8)             self.dimensions = dimensions
478: (4)             super().__init__(**kwargs)
479: (8)         def generate_points(self) -> None:
480: (8)             """Creates the sides of the :class:`Prism`."""
481: (4)             super().generate_points()
482: (8)             for dim, value in enumerate(self.dimensions):
483: (8)                 self.rescale_to_fit(value, dim, stretch=True)
484: (12)         class Cone(Surface):
485: (0)             """A circular cone.
486: (4)             Can be defined using 2 parameters: its height, and its base radius.
487: (4)             The polar angle, theta, can be calculated using arctan(base_radius / height) The spherical radius, r, is calculated using the pythagorean theorem.
488: (4)             Parameters
489: (4)             -----
490: (4)             base_radius
491: (8)                 The base radius from which the cone tapers.
492: (4)             height
493: (8)                 The height measured from the plane formed by the base_radius to the apex of the cone.
494: (4)             direction
495: (8)                 The direction of the apex.
496: (4)             show_base
497: (8)                 Whether to show the base plane or not.
498: (4)             v_range
499: (8)                 The azimuthal angle to start and end at.
500: (4)             u_min
501: (8)                 The radius at the apex.
502: (4)             checkerboard_colors
503: (8)                 Show checkerboard grid texture on the cone.
504: (4)             Examples
505: (4)             -----
506: (4)             .. manim:: ExampleCone
507: (8)                 :save_last_frame:
508: (4)                 class ExampleCone(ThreeDScene):
509: (8)                     def construct(self):
510: (12)                         axes = ThreeDAxes()
511: (16)                         cone = Cone(direction=X_AXIS+Y_AXIS+2*Z_AXIS, resolution=8)
512: (16)                         self.set_camera_orientation(phi=5*PI/11, theta=PI/9)
513: (16)                         self.add(axes, cone)
514: (12)                     """
515: (16)                     def __init__(self, base_radius: float = 1, height: float = 1, direction: np.ndarray = Z_AXIS, show_base: bool = False, v_range: Sequence[float] = [0, TAU], u_min: float = 0, checkerboard_colors: bool = False, **kwargs: Any):
516: (12)             self.func,
517: (12)             v_range=v_range,
518: (12)             u_range=[u_min, np.sqrt(base_radius**2 + height**2)],
519: (12)             checkerboard_colors=checkerboard_colors,
520: (12)             **kwargs,
521: (12)         ) -> None:
522: (12)             self.direction = direction
523: (12)             self.theta = PI - np.arctan(base_radius / height)
524: (12)             super().__init__(self.func,
525: (12)             v_range=v_range,
526: (12)             u_range=[u_min, np.sqrt(base_radius**2 + height**2)],
527: (12)             checkerboard_colors=checkerboard_colors,
528: (12)             **kwargs,
529: (12)         )
530: (12)             self._current_theta = 0
531: (12)             self._current_phi = 0
532: (12)             if show_base:
533: (12)
534: (12)
535: (12)
536: (12)
537: (12)
538: (12)
539: (8)
540: (8)
541: (8)
542: (8)

```

```

543: (12)                     self.base_circle = Circle(
544: (16)                       radius=base_radius,
545: (16)                       color=self.fill_color,
546: (16)                       fill_opacity=self.fill_opacity,
547: (16)                       stroke_width=0,
548: (12)                   )
549: (12)                   self.base_circle.shift(height * IN)
550: (12)                   self.add(self.base_circle)
551: (8)                   self._rotate_to_direction()
552: (4) def func(self, u: float, v: float) -> np.ndarray:
553: (8)     """Converts from spherical coordinates to cartesian.
554: (8)     Parameters
555: (8)     -----
556: (8)     u
557: (12)         The radius.
558: (8)     v
559: (12)         The azimuthal angle.
560: (8)     Returns
561: (8)     -----
562: (8)     :class:`numpy.array`
563: (12)         Points defining the :class:`Cone`.
564: (8)     """
565: (8)     r = u
566: (8)     phi = v
567: (8)     return np.array(
568: (12)         [
569: (16)             r * np.sin(self.theta) * np.cos(phi),
570: (16)             r * np.sin(self.theta) * np.sin(phi),
571: (16)             r * np.cos(self.theta),
572: (12)         ],
573: (8)     )
574: (4) def _rotate_to_direction(self) -> None:
575: (8)     x, y, z = self.direction
576: (8)     r = np.sqrt(x**2 + y**2 + z**2)
577: (8)     if r > 0:
578: (12)         theta = np.arccos(z / r)
579: (8)     else:
580: (12)         theta = 0
581: (8)     if x == 0:
582: (12)         if y == 0: # along the z axis
583: (16)             phi = 0
584: (12)         else:
585: (16)             phi = np.arctan(np.inf)
586: (16)             if y < 0:
587: (20)                 phi += PI
588: (8)         else:
589: (12)             phi = np.arctan(y / x)
590: (8)     if x < 0:
591: (12)         phi += PI
592: (8)     self.rotate(-self._current_phi, Z_AXIS, about_point=ORIGIN)
593: (8)     self.rotate(-self._current_theta, Y_AXIS, about_point=ORIGIN)
594: (8)     self.rotate(theta, Y_AXIS, about_point=ORIGIN)
595: (8)     self.rotate(phi, Z_AXIS, about_point=ORIGIN)
596: (8)     self._current_theta = theta
597: (8)     self._current_phi = phi
598: (4) def set_direction(self, direction: np.ndarray) -> None:
599: (8)     """Changes the direction of the apex of the :class:`Cone`.
600: (8)     Parameters
601: (8)     -----
602: (8)     direction
603: (12)         The direction of the apex.
604: (8)     """
605: (8)     self.direction = direction
606: (8)     self._rotate_to_direction()
607: (4) def get_direction(self) -> np.ndarray:
608: (8)     """Returns the current direction of the apex of the :class:`Cone`.
609: (8)     Returns
610: (8)     -----
611: (8)     direction : :class:`numpy.array`
```

```

612: (12)           The direction of the apex.
613: (8)
614: (8)           return self.direction
615: (0)           class Cylinder(Surface):
616: (4)             """A cylinder, defined by its height, radius and direction,
617: (4)             Parameters
618: (4)             -----
619: (4)             radius
620: (8)               The radius of the cylinder.
621: (4)             height
622: (8)               The height of the cylinder.
623: (4)             direction
624: (8)               The direction of the central axis of the cylinder.
625: (4)             v_range
626: (8)               The height along the height axis (given by direction) to start and end
on.
627: (4)             show_ends
628: (8)               Whether to show the end caps or not.
629: (4)             resolution
630: (8)               The number of samples taken of the :class:`Cylinder`. A tuple can be
used
631: (8)               to define different resolutions for ``u`` and ``v`` respectively.
632: (4)             Examples
633: (4)             -----
634: (4)               .. manim:: ExampleCylinder
635: (8)                 :save_last_frame:
636: (8)                 class ExampleCylinder(ThreeDScene):
637: (12)                   def construct(self):
638: (16)                     axes = ThreeDAxes()
639: (16)                     cylinder = Cylinder(radius=2, height=3)
640: (16)                     self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
641: (16)                     self.add(axes, cylinder)
642: (4)               """
643: (4)               def __init__(
644: (8)                 self,
645: (8)                 radius: float = 1,
646: (8)                 height: float = 2,
647: (8)                 direction: np.ndarray = Z_AXIS,
648: (8)                 v_range: Sequence[float] = [0, TAU],
649: (8)                 show_ends: bool = True,
650: (8)                 resolution: Sequence[int] = (24, 24),
651: (8)                 **kwargs,
652: (4)             ) -> None:
653: (8)               self._height = height
654: (8)               self.radius = radius
655: (8)               super().__init__(
656: (12)                 self.func,
657: (12)                 resolution=resolution,
658: (12)                 u_range=[-self._height / 2, self._height / 2],
659: (12)                 v_range=v_range,
660: (12)                 **kwargs,
661: (8)             )
662: (8)             if show_ends:
663: (12)               self.add_bases()
664: (8)             self._current_phi = 0
665: (8)             self._current_theta = 0
666: (8)             self.set_direction(direction)
667: (4)             def func(self, u: float, v: float) -> np.ndarray:
668: (8)               """Converts from cylindrical coordinates to cartesian.
669: (8)               Parameters
670: (8)               -----
671: (8)               u
672: (12)                 The height.
673: (8)               v
674: (12)                 The azimuthal angle.
675: (8)               Returns
676: (8)               -----
677: (8)                 :class:`numpy.ndarray`
```

```

678: (12)             Points defining the :class:`Cylinder` .
679: (8)
680: (8)
681: (8)
682: (8)
683: (8)
684: (4)             """
685: (8)             height = u
686: (8)             phi = v
687: (12)             r = self.radius
688: (12)             return np.array([r * np.cos(phi), r * np.sin(phi), height])
689: (8)
690: (12)             def add_bases(self) -> None:
691: (12)             """Adds the end caps of the cylinder."""
692: (12)             if config.renderer == RendererType.OPENGL:
693: (12)                 color = self.color
694: (12)                 opacity = self.opacity
695: (12)             elif config.renderer == RendererType.CAIRO:
696: (12)                 color = self.fill_color
697: (12)                 opacity = self.fill_opacity
698: (8)             self.base_top = Circle(
699: (12)                 radius=self.radius,
700: (8)                 color=color,
701: (12)                 fill_opacity=opacity,
702: (12)                 shade_in_3d=True,
703: (12)                 stroke_width=0,
704: (12)
705: (12)
706: (8)
707: (8)
708: (8)
709: (4)             self.base_top.shift(self.u_range[1] * IN)
710: (8)             self.base_bottom = Circle(
711: (12)                 radius=self.radius,
712: (12)                 color=color,
713: (12)                 fill_opacity=opacity,
714: (12)                 shade_in_3d=True,
715: (12)                 stroke_width=0,
716: (8)
717: (12)
718: (16)
719: (12)
720: (16)
721: (16)
722: (20)
723: (8)
724: (12)
725: (8)
726: (12)
727: (8)
728: (8)
729: (8)
730: (8)
731: (8)
732: (8)
733: (4)             self.rotate(-self._current_phi, Z_AXIS, about_point=ORIGIN)
734: (8)             self.rotate(-self._current_theta, Y_AXIS, about_point=ORIGIN)
735: (8)             self.rotate(theta, Y_AXIS, about_point=ORIGIN)
736: (8)             self.rotate(phi, Z_AXIS, about_point=ORIGIN)
737: (8)             self._current_theta = theta
738: (12)             self._current_phi = phi
739: (8)
740: (8)
741: (8)
742: (4)             def set_direction(self, direction: np.ndarray) -> None:
743: (8)             """Sets the direction of the central axis of the :class:`Cylinder` .
744: (8)             Parameters
745: (8)
746: (8)             direction : :class:`numpy.array`             The direction of the central axis of the :class:`Cylinder` .
747: (8)
748: (8)             self.direction = direction
749: (8)             self._rotate_to_direction()
750: (4)             def get_direction(self) -> np.ndarray:
751: (8)             """Returns the direction of the central axis of the :class:`Cylinder` .
752: (8)             Returns
753: (8)
754: (8)             direction : :class:`numpy.array`
```

```

747: (12)             The direction of the central axis of the :class:`Cylinder`.
748: (8)
749: (8)             return self.direction
750: (0)             class Line3D(Cylinder):
751: (4)             """A cylindrical line, for use in ThreeDScene.
752: (4)             Parameters
753: (4)             -----
754: (4)             start
755: (8)             The start point of the line.
756: (4)             end
757: (8)             The end point of the line.
758: (4)             thickness
759: (8)             The thickness of the line.
760: (4)             color
761: (8)             The color of the line.
762: (4)             Examples
763: (4)             -----
764: (4)             .. manim:: ExampleLine3D
765: (8)             :save_last_frame:
766: (8)             class ExampleLine3D(ThreeDScene):
767: (12)             def construct(self):
768: (16)             axes = ThreeDAxes()
769: (16)             line = Line3D(start=np.array([0, 0, 0]), end=np.array([2, 2,
2]))
770: (16)             self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
771: (16)             self.add(axes, line)
772: (4)             """
773: (4)             def __init__(
774: (8)                 self,
775: (8)                 start: np.ndarray = LEFT,
776: (8)                 end: np.ndarray = RIGHT,
777: (8)                 thickness: float = 0.02,
778: (8)                 color: ParsableManimColor | None = None,
779: (8)                 **kwargs,
780: (4)             ):
781: (8)                 self.thickness = thickness
782: (8)                 self.set_start_and_end_attrs(start, end, **kwargs)
783: (8)                 if color is not None:
784: (12)                     self.set_color(color)
785: (4)             def set_start_and_end_attrs(
786: (8)                 self, start: np.ndarray, end: np.ndarray, **kwargs
787: (4)             ) -> None:
788: (8)                 """Sets the start and end points of the line.
789: (8)                 If either ``start`` or ``end`` are :class:`Mobjects <.Mobject>` ,
790: (8)                 this gives their centers.
791: (8)                 Parameters
792: (8)                 -----
793: (8)                 start
794: (12)                     Starting point or :class:`Mobject` .
795: (8)                 end
796: (12)                     Ending point or :class:`Mobject` .
797: (8)                 """
798: (8)                 rough_start = self.pointify(start)
799: (8)                 rough_end = self.pointify(end)
800: (8)                 self.vect = rough_end - rough_start
801: (8)                 self.length = np.linalg.norm(self.vect)
802: (8)                 self.direction = normalize(self.vect)
803: (8)                 self.start = self.pointify(start, self.direction)
804: (8)                 self.end = self.pointify(end, -self.direction)
805: (8)                 super().__init__(
806: (12)                     height=np.linalg.norm(self.vect),
807: (12)                     radius=self.thickness,
808: (12)                     direction=self.direction,
809: (12)                     **kwargs,
810: (8)                 )
811: (8)                 self.shift((self.start + self.end) / 2)
812: (4)             def pointify(
813: (8)                 self,

```

```

814: (8)             mob_or_point: Mobject | Point3D,
815: (8)             direction: Vector3D = None,
816: (4)
817: (8)         ) -> np.ndarray:
818: (8)             """Gets a point representing the center of the :class:`Mobjects
819: <.Mobject>`.
820: (8)
821: (12)             Parameters
822: (8)             -----
823: (12)             mob_or_point
824: (8)                 :class:`Mobjects <.Mobject>` or point whose center should be
825: (8)                 returned.
826: (8)
827: (12)             direction
828: (8)                 If an edge of a :class:`Mobjects <.Mobject>` should be returned,
829: (8)             the direction of the edge.
830: (8)
831: (12)             Returns
832: (8)             -----
833: (12)             :class:`numpy.array`
834: (8)                 Center of the :class:`Mobjects <.Mobject>` or point, or edge if
835: (8)                 direction is given.
836: (8)
837: (12)             if isinstance(mob_or_point, (Mobject, OpenGLMobject)):
838: (8)                 mob = mob_or_point
839: (8)                 if direction is None:
840: (8)                     return mob.get_center()
841: (8)                 else:
842: (8)                     return mob.get_boundary_point(direction)
843: (8)
844: (4)             return np.array(mob_or_point)
845: (8)
846: (4)         def get_start(self) -> np.ndarray:
847: (8)             """Returns the starting point of the :class:`Line3D`.
848: (8)
849: (12)             Returns
850: (8)
851: (8)             start : :class:`numpy.array`
852: (8)                 Starting point of the :class:`Line3D`.
853: (8)
854: (4)             """
855: (8)
856: (8)
857: (8)
858: (8)
859: (4)
860: (8)
861: (8)         def get_end(self) -> np.ndarray:
862: (8)             """Returns the ending point of the :class:`Line3D`.
863: (8)
864: (8)
865: (12)             Returns
866: (8)
867: (12)             @classmethod
868: (8)
869: (12)             def parallel_to(
870: (8)                 cls,
871: (8)                 line: Line3D,
872: (8)                 point: Vector3D = ORIGIN,
873: (8)                 length: float = 5,
874: (8)                 **kwargs,
875: (4)
876: (8)
877: (8)
878: (8)             ) -> Line3D:
879: (8)                 """Returns a line parallel to another line going through
880: (8)                 a given point.
881: (8)
882: (8)                 Parameters
883: (8)
884: (8)                 line
885: (8)                     The line to be parallel to.
886: (8)
887: (12)                 point
888: (8)                     The point to pass through.
889: (8)
890: (12)                 length
891: (8)                     Length of the parallel line.
892: (8)
893: (12)                 kwargs
894: (8)                     Additional parameters to be passed to the class.
895: (8)
896: (12)                 Returns
897: (8)
898: (12)                 :class:`Line3D`
899: (8)                     Line parallel to ``line``.
900: (8)
901: (12)                 Examples
902: (8)
903: (12)                 .. manim:: ParallelLineExample

```

```

879: (12)           :save_last_frame:
880: (12)           class ParallelLineExample(ThreeDScene):
881: (16)             def construct(self):
882: (20)               self.set_camera_orientation(PI / 3, -PI / 4)
883: (20)               ax = ThreeDAxes((-5, 5), (-5, 5), (-5, 5), 10, 10, 10)
884: (20)               line1 = Line3D(RIGHT * 2, UP + OUT, color=RED)
885: (20)               line2 = Line3D.parallel_to(line1, color=YELLOW)
886: (20)               self.add(ax, line1, line2)
887: (8)             """
888: (8)             point = np.array(point)
889: (8)             vect = normalize(line.vect)
890: (8)             return cls(
891: (12)               point + vect * length / 2,
892: (12)               point - vect * length / 2,
893: (12)               **kwargs,
894: (8)             )
895: (4)             @classmethod
896: (4)             def perpendicular_to(
897: (8)               cls,
898: (8)               line: Line3D,
899: (8)               point: Vector3D = ORIGIN,
900: (8)               length: float = 5,
901: (8)               **kwargs,
902: (4)             ) -> Line3D:
903: (8)               """Returns a line perpendicular to another line going through
904: (8)               a given point.
905: (8)               Parameters
906: (8)               -----
907: (8)               line
908: (12)                 The line to be perpendicular to.
909: (8)               point
910: (12)                 The point to pass through.
911: (8)               length
912: (12)                 Length of the perpendicular line.
913: (8)               kwargs
914: (12)                 Additional parameters to be passed to the class.
915: (8)               Returns
916: (8)               -----
917: (8)               :class:`Line3D`  
Line perpendicular to ``line``.
918: (12)               Examples
919: (8)               -----
920: (8)               .. manim:: PerpLineExample
921: (8)                   :save_last_frame:
922: (12)                   class PerpLineExample(ThreeDScene):
923: (12)                     def construct(self):
924: (16)                       self.set_camera_orientation(PI / 3, -PI / 4)
925: (20)                       ax = ThreeDAxes((-5, 5), (-5, 5), (-5, 5), 10, 10, 10)
926: (20)                       line1 = Line3D(RIGHT * 2, UP + OUT, color=RED)
927: (20)                       line2 = Line3D.perpendicular_to(line1, color=BLUE)
928: (20)                       self.add(ax, line1, line2)
929: (20)                     """
930: (8)
931: (8)                     point = np.array(point)
932: (8)                     norm = np.cross(line.vect, point - line.start)
933: (8)                     if all(np.linalg.norm(norm) == np.zeros(3)):
934: (12)                       raise ValueError("Could not find the perpendicular.")
935: (8)                     start, end = perpendicular_bisector([line.start, line.end], norm)
936: (8)                     vect = normalize(end - start)
937: (8)                     return cls(
938: (12)                       point + vect * length / 2,
939: (12)                       point - vect * length / 2,
940: (12)                       **kwargs,
941: (8)                     )
942: (0)             class Arrow3D(Line3D):
943: (4)               """An arrow made out of a cylindrical line and a conical tip.
944: (4)               Parameters
945: (4)               -----
946: (4)               start
947: (8)                 The start position of the arrow.

```

```

948: (4)           end
949: (8)             The end position of the arrow.
950: (4)           thickness
951: (8)             The thickness of the arrow.
952: (4)           height
953: (8)             The height of the conical tip.
954: (4)           base_radius
955: (8)             The base radius of the conical tip.
956: (4)           color
957: (8)             The color of the arrow.
958: (4)           Examples
959: (4)           -----
960: (4)             .. manim:: ExampleArrow3D
961: (8)               :save_last_frame:
962: (8)               class ExampleArrow3D(ThreeDScene):
963: (12)                 def construct(self):
964: (16)                   axes = ThreeDAxes()
965: (16)                   arrow = Arrow3D(
966: (20)                     start=np.array([0, 0, 0]),
967: (20)                     end=np.array([2, 2, 2]),
968: (20)                     resolution=8
969: (16)                   )
970: (16)                   self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
971: (16)                   self.add(axes, arrow)
972: (4)           """
973: (4)             def __init__(
974: (8)               self,
975: (8)                 start: np.ndarray = LEFT,
976: (8)                 end: np.ndarray = RIGHT,
977: (8)                 thickness: float = 0.02,
978: (8)                 height: float = 0.3,
979: (8)                 base_radius: float = 0.08,
980: (8)                 color: ParsableManimColor = WHITE,
981: (8)                 **kwargs,
982: (4)             ) -> None:
983: (8)               super().__init__(
984: (12)                 start=start, end=end, thickness=thickness, color=color, **kwargs
985: (8)
986: (8)               )
987: (8)               self.length = np.linalg.norm(self.vect)
988: (12)               self.set_start_and_end_attrs(
989: (12)                 self.start,
990: (12)                 self.end - height * self.direction,
**kwargs,
991: (8)
992: (8)
993: (12)               self.cone = Cone(
994: (8)                 direction=self.direction, base_radius=base_radius, height=height,
**kwargs
995: (8)
996: (8)
997: (8)
998: (0)           class Torus(Surface):
999: (4)             """A torus.
1000: (4)             Parameters
1001: (4)             -----
1002: (4)             major_radius
1003: (8)               Distance from the center of the tube to the center of the torus.
1004: (4)             minor_radius
1005: (8)               Radius of the tube.
1006: (4)             u_range
1007: (8)               The range of the ``u`` variable: ``

```

```

1015: (4)          .. manim :: ExampleTorus
1016: (8)          :save_last_frame:
1017: (8)          class ExampleTorus(ThreeDScene):
1018: (12)         def construct(self):
1019: (16)         axes = ThreeDAxes()
1020: (16)         torus = Torus()
1021: (16)         self.set_camera_orientation(phi=75 * DEGREES, theta=30 *
DEGREES)
1022: (16)             self.add(axes, torus)
1023: (4)         """
1024: (4)         def __init__(
1025: (8)             self,
1026: (8)             major_radius: float = 3,
1027: (8)             minor_radius: float = 1,
1028: (8)             u_range: Sequence[float] = (0, TAU),
1029: (8)             v_range: Sequence[float] = (0, TAU),
1030: (8)             resolution: tuple[int, int] | None = None,
1031: (8)             **kwargs,
1032: (4)         ) -> None:
1033: (8)             if config.renderer == RendererType.OPENGL:
1034: (12)                 res_value = (101, 101)
1035: (8)             elif config.renderer == RendererType.CAIRO:
1036: (12)                 res_value = (24, 24)
1037: (8)             resolution = resolution if resolution is not None else res_value
1038: (8)             self.R = major_radius
1039: (8)             self.r = minor_radius
1040: (8)             super().__init__(
1041: (12)                 self.func,
1042: (12)                 u_range=u_range,
1043: (12)                 v_range=v_range,
1044: (12)                 resolution=resolution,
1045: (12)                 **kwargs,
1046: (8)             )
1047: (4)         def func(self, u: float, v: float) -> np.ndarray:
1048: (8)             """The z values defining the :class:`Torus` being plotted.
1049: (8)             Returns
1050: (8)             -----
1051: (8)             :class:`numpy.ndarray`
1052: (12)                 The z values defining the :class:`Torus`.
1053: (8)             """
1054: (8)             P = np.array([np.cos(u), np.sin(u), 0])
1055: (8)             return (self.R - self.r * np.cos(v)) * P - self.r * np.sin(v) * OUT
-----
```

## File 92 - point\_cloud\_mobject.py:

```

1: (0)         """Mobjects representing point clouds."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = ["PMobject", "Mobject1D", "Mobject2D", "PGroup", "PointCloudDot",
"Point"]
4: (0)         import numpy as np
5: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
6: (0)         from manim.mobject.opengl.opengl_point_cloud_mobject import OpenGLMobject
7: (0)         from ...constants import *
8: (0)         from ...mobject.mobject import Mobject
9: (0)         from ...utils.bezier import interpolate
10: (0)        from ...utils.color import (
11: (4)             BLACK,
12: (4)             WHITE,
13: (4)             YELLOW,
14: (4)             ManimColor,
15: (4)             color_gradient,
16: (4)             color_to_rgba,
17: (4)             rgba_to_color,
18: (0)         )
19: (0)         from ...utils.iterables import stretch_array_to_length
20: (0)         __all__ = ["PMobject", "Mobject1D", "Mobject2D", "PGroup", "PointCloudDot",
"Point"]
```

```

21: (0)         class PMobject(Mobject, metaclass=ConvertToOpenGL):
22: (4)             """A disc made of a cloud of Dots
23: (4)             Examples
24: (4)             -----
25: (4)             .. manim:: PMobjectExample
26: (8)                 :save_last_frame:
27: (8)                 class PMobjectExample(Scene):
28: (12)                     def construct(self):
29: (16)                         pG = PGroup() # This is just a collection of PMobject's
30: (16)                         for sf in range(1, 9 + 1):
31: (20)                             p = PointCloudDot(density=20, radius=1).thin_out(sf)
32: (20)                             pG.add(p)
33: (16)                         pG.arrange_in_grid()
34: (16)                         self.add(pG)
35: (4)             """
36: (4)             def __init__(self, stroke_width=DEFAULT_STROKE_WIDTH, **kwargs):
37: (8)                 self.stroke_width = stroke_width
38: (8)                 super().__init__(**kwargs)
39: (4)             def reset_points(self):
40: (8)                 self.rgbas = np.zeros((0, 4))
41: (8)                 self.points = np.zeros((0, 3))
42: (8)                 return self
43: (4)             def get_array_attrs(self):
44: (8)                 return super().get_array_attrs() + ["rgbas"]
45: (4)             def add_points(self, points, rgbas=None, color=None, alpha=1):
46: (8)                 """Add points.
47: (8)                 Points must be a Nx3 numpy array.
48: (8)                 Rgbas must be a Nx4 numpy array if it is not None.
49: (8)                 """
50: (8)                 if not isinstance(points, np.ndarray):
51: (12)                     points = np.array(points)
52: (8)                     num_new_points = len(points)
53: (8)                     self.points = np.append(self.points, points, axis=0)
54: (8)                     if rgbas is None:
55: (12)                         color = ManimColor(color) if color else self.color
56: (12)                         rgbas = np.repeat([color_to_rgba(color, alpha)], num_new_points,
axis=0)
57: (8)                 elif len(rgbas) != len(points):
58: (12)                     raise ValueError("points and rgbas must have same length")
59: (8)                     self.rgbas = np.append(self.rgbas, rgbas, axis=0)
60: (8)                     return self
61: (4)             def set_color(self, color=YELLOW, family=True):
62: (8)                 rgba = color_to_rgba(color)
63: (8)                 mobs = self.family_members_with_points() if family else [self]
64: (8)                 for mob in mobs:
65: (12)                     mob.rgbas[:, :] = rgba
66: (8)                     self.color = color
67: (8)                     return self
68: (4)             def get_stroke_width(self):
69: (8)                 return self.stroke_width
70: (4)             def set_stroke_width(self, width, family=True):
71: (8)                 mobs = self.family_members_with_points() if family else [self]
72: (8)                 for mob in mobs:
73: (12)                     mob.stroke_width = width
74: (8)                     return self
75: (4)             def set_color_by_gradient(self, *colors):
76: (8)                 self.rgbas = np.array(
77: (12)                     list(map(color_to_rgba, color_gradient(*colors,
len(self.points)))),
78: (8)                         )
79: (8)                     return self
80: (4)             def set_colors_by_radial_gradient(
81: (8)                 self,
82: (8)                 center=None,
83: (8)                 radius=1,
84: (8)                 inner_color=WHITE,
85: (8)                 outer_color=BLACK,
86: (4)                 ):
87: (8)                 start_rgba, end_rgba = list(map(color_to_rgba, [inner_color,

```

```

outer_color)))
88: (8)         if center is None:
89: (12)           center = self.get_center()
90: (8)         for mob in self.family_members_with_points():
91: (12)           distances = np.abs(self.points - center)
92: (12)           alphas = np.linalg.norm(distances, axis=1) / radius
93: (12)           mob.rgbas = np.array(
94: (16)             np.array(
95: (20)               [interpolate(start_rgba, end_rgba, alpha) for alpha in
alphas],
96: (16)             ),
97: (12)           )
98: (8)         return self
99: (4)     def match_colors(self, mobject):
100: (8)       Mobject.align_data(self, mobject)
101: (8)       self.rgbas = np.array(mobject.rgbas)
102: (8)       return self
103: (4)     def filter_out(self, condition):
104: (8)       for mob in self.family_members_with_points():
105: (12)         to_eliminate = ~np.apply_along_axis(condition, 1, mob.points)
106: (12)         mob.points = mob.points[to_eliminate]
107: (12)         mob.rgbas = mob.rgbas[to_eliminate]
108: (8)       return self
109: (4)     def thin_out(self, factor=5):
110: (8)       """
111: (8)         Removes all but every nth point for n = factor
112: (8)       """
113: (8)       for mob in self.family_members_with_points():
114: (12)         num_points = self.get_num_points()
115: (12)         mob.apply_over_attr_arrays(
116: (16)           lambda arr: arr[np.arange(0, num_points, factor)],
117: (12)         )
118: (8)       return self
119: (4)     def sort_points(self, function=lambda p: p[0]):
120: (8)       """
121: (8)         Function is any map from R^3 to R
122: (8)       """
123: (8)       for mob in self.family_members_with_points():
124: (12)         indices = np.argsort(np.apply_along_axis(function, 1, mob.points))
125: (12)         mob.apply_over_attr_arrays(lambda arr: arr[indices])
126: (8)       return self
127: (4)     def fade_to(self, color, alpha, family=True):
128: (8)       self.rgbas = interpolate(self.rgbas, color_to_rgba(color), alpha)
129: (8)       for mob in self.submobjects:
130: (12)         mob.fade_to(color, alpha, family)
131: (8)       return self
132: (4)     def get_all_rgbas(self):
133: (8)       return self.get_merged_array("rgbas")
134: (4)     def ingest_submobjects(self):
135: (8)       attrs = self.get_array_attrs()
136: (8)       arrays = list(map(self.get_merged_array, attrs))
137: (8)       for attr, array in zip(attrs, arrays):
138: (12)         setattr(self, attr, array)
139: (8)       self.submobjects = []
140: (8)       return self
141: (4)     def get_color(self):
142: (8)       return rgba_to_color(self.rgbas[0, :])
143: (4)     def point_from_proportion(self, alpha):
144: (8)       index = alpha * (self.get_num_points() - 1)
145: (8)       return self.points[index]
146: (4)     @staticmethod
147: (4)     def get_mobject_type_class():
148: (8)       return PMobject
149: (4)     def align_points_with_larger(self, larger_mobject):
150: (8)       assert isinstance(larger_mobject, PMobject)
151: (8)       self.apply_over_attr_arrays(
152: (12)         lambda a: stretch_array_to_length(a,
larger_mobject.get_num_points()),
153: (8)       )

```

```

154: (4)             def get_point_mobject(self, center=None):
155: (8)                 if center is None:
156: (12)                     center = self.get_center()
157: (8)                     return Point(center)
158: (4)             def interpolate_color(self, mobject1, mobject2, alpha):
159: (8)                 self.rgbas = interpolate(mobject1.rgbas, mobject2.rgbas, alpha)
160: (8)                 self.set_stroke_width(
161: (12)                     interpolate(
162: (16)                         mobject1.get_stroke_width(),
163: (16)                         mobject2.get_stroke_width(),
164: (16)                         alpha,
165: (12)                     ),
166: (8)
167: (8)                 )
168: (4)             return self
169: (8)             def pointwise_become_partial(self, mobject, a, b):
170: (8)                 lower_index, upper_index = (int(x * mobject.get_num_points()) for x in
171: (a, b))
172: (8)                 for attr in self.get_array_attrs():
173: (12)                     full_array = getattr(mobject, attr)
174: (12)                     partial_array = full_array[lower_index:upper_index]
175: (12)                     setattr(self, attr, partial_array)
176: (0)             class Mobject1D(PMobject, metaclass=ConvertToOpenGL):
177: (4)                 def __init__(self, density=DEFAULT_POINT_DENSITY_1D, **kwargs):
178: (8)                     self.density = density
179: (8)                     self.epsilon = 1.0 / self.density
180: (8)                     super().__init__(**kwargs)
181: (8)                 def add_line(self, start, end, color=None):
182: (8)                     start, end = list(map(np.array, [start, end]))
183: (8)                     length = np.linalg.norm(end - start)
184: (12)                     if length == 0:
185: (12)                         points = [start]
186: (12)                     else:
187: (12)                         epsilon = self.epsilon / length
188: (12)                         points = [interpolate(start, end, t) for t in np.arange(0, 1,
189: (epsilon)]
190: (8)                     self.add_points(points, color=color)
191: (0)             class Mobject2D(PMobject, metaclass=ConvertToOpenGL):
192: (4)                 def __init__(self, density=DEFAULT_POINT_DENSITY_2D, **kwargs):
193: (8)                     self.density = density
194: (8)                     self.epsilon = 1.0 / self.density
195: (8)                     super().__init__(**kwargs)
196: (0)             class PGroup(PMobject):
197: (4)                 """A group for several point mobjects.
198: (4)                 Examples
199: (4)                 -----
200: (12)                 .. manim:: PgroupExample
201: (16)                     :save_last_frame:
202: (16)                     class PgroupExample(Scene):
203: (16)                         def construct(self):
204: (16)                             p1 = PointCloudDot(radius=1, density=20, color=BLUE)
205: (16)                             p1.move_to(4.5 * LEFT)
206: (16)                             p2 = PointCloudDot()
207: (16)                             p3 = PointCloudDot(radius=1.5, stroke_width=2.5, color=PINK)
208: (16)                             p3.move_to(4.5 * RIGHT)
209: (16)                             pList = PGroup(p1, p2, p3)
210: (16)                             self.add(pList)
211: (4)                 """
212: (4)                 def __init__(self, *pmobs, **kwargs):
213: (8)                     if not all(isinstance(m, (PMobject, OpenGLMobject)) for m in pmobs):
214: (12)                         raise ValueError(
215: (16)                             "All submobjects must be of type PMobject or OpenGLMobject"
216: (16)                             " if using the opengl renderer",
217: (12)                         )
218: (8)                     super().__init__(**kwargs)
219: (8)                     self.add(*pmobs)
220: (4)                 def fade_to(self, color, alpha, family=True):
221: (8)                     if family:
222: (12)                         for mob in self.submobjects:
223: (16)                             mob.fade_to(color, alpha, family)

```

```

221: (0)         class PointCloudDot(Mobject1D):
222: (4)             """A disc made of a cloud of dots.
223: (4)             Examples
224: (4)             -----
225: (4)             .. manim:: PointCloudDotExample
226: (8)                 :save_last_frame:
227: (8)                 class PointCloudDotExample(Scene):
228: (12)                     def construct(self):
229: (16)                         cloud_1 = PointCloudDot(color=RED)
230: (16)                         cloud_2 = PointCloudDot(stroke_width=4, radius=1)
231: (16)                         cloud_3 = PointCloudDot(density=15)
232: (16)                         group = Group(cloud_1, cloud_2, cloud_3).arrange()
233: (16)                         self.add(group)
234: (4)             .. manim:: PointCloudDotExample2
235: (8)                 class PointCloudDotExample2(Scene):
236: (12)                     def construct(self):
237: (16)                         plane = ComplexPlane()
238: (16)                         cloud = PointCloudDot(color=RED)
239: (16)                         self.add(
240: (20)                             plane, cloud
241: (16)                         )
242: (16)                         self.wait()
243: (16)                         self.play(
244: (20)                             cloud.animate.apply_complex_function(lambda z: np.exp(z))
245: (16)                         )
246: (4)             """
247: (4)             def __init__(
248: (8)                 self,
249: (8)                 center=ORIGIN,
250: (8)                 radius=2.0,
251: (8)                 stroke_width=2,
252: (8)                 density=DEFAULT_POINT_DENSITY_1D,
253: (8)                 color=YELLOW,
254: (8)                 **kwargs,
255: (4)             ):
256: (8)                 self.radius = radius
257: (8)                 self.epsilon = 1.0 / density
258: (8)                 super().__init__(
259: (12)                     stroke_width=stroke_width, density=density, color=color, **kwargs
260: (8)                 )
261: (8)                 self.shift(center)
262: (4)             def init_points(self):
263: (8)                 self.reset_points()
264: (8)                 self.generate_points()
265: (4)             def generate_points(self):
266: (8)                 self.add_points(
267: (12)                     [
268: (16)                         r * (np.cos(theta) * RIGHT + np.sin(theta) * UP)
269: (16)                         for r in np.arange(self.epsilon, self.radius, self.epsilon)
270: (16)                         for theta in np.linspace(
271: (20)                             0,
272: (20)                             2 * np.pi,
273: (20)                             num=int(2 * np.pi * (r + self.epsilon) / self.epsilon),
274: (16)                         )
275: (12)                     ],
276: (8)                 )
277: (0)             class Point(PMobject):
278: (4)                 """A mobject representing a point.
279: (4)                 Examples
280: (4)                 -----
281: (4)                 .. manim:: ExamplePoint
282: (8)                     :save_last_frame:
283: (8)                     class ExamplePoint(Scene):
284: (12)                         def construct(self):
285: (16)                             colorList = [RED, GREEN, BLUE, YELLOW]
286: (16)                             for i in range(200):
287: (20)                                 point = Point(location=[0.63 * np.random.randint(-4, 4),
0.37 * np.random.randint(-4, 4), 0], color=np.random.choice(colorList))
288: (20)                                 self.add(point)

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
289: (16)                     for i in range(200):
290: (20)                         point = Point(location=[0.37 * np.random.randint(-4, 4),
0.63 * np.random.randint(-4, 4), 0], color=np.random.choice(colorList))
291: (20)                             self.add(point)
292: (16)                             self.add(point)
293: (4) """
294: (4)     def __init__(self, location=ORIGIN, color=BLACK, **kwargs):
295: (8)         self.location = location
296: (8)         super().__init__(color=color, **kwargs)
297: (4)     def init_points(self):
298: (8)         self.reset_points()
299: (8)         self.generate_points()
300: (8)         self.set_points([self.location])
301: (4)     def generate_points(self):
302: (8)         self.add_points([self.location])
```

---

## File 93 - opengl\_image\_mobject.py:

```
1: (0)             from __future__ import annotations
2: (0)             __all__ = [
3: (4)                 "OpenGLImageMobject",
4: (0)             ]
5: (0)             from pathlib import Path
6: (0)             import numpy as np
7: (0)             from PIL import Image
8: (0)             from PIL.Image import Resampling
9: (0)             from manim.mobject.opengl.opengl_surface import OpenGLSurface,
OpenGLTexturedSurface
10: (0)             from manim.utils.images import get_full_raster_image_path
11: (0)             __all__ = ["OpenGLImageMobject"]
12: (0)             class OpenGLImageMobject(OpenGLTexturedSurface):
13: (4)                 def __init__(
14: (8)                     self,
15: (8)                     filename_or_array: str | Path | np.ndarray,
16: (8)                     width: float = None,
17: (8)                     height: float = None,
18: (8)                     image_mode: str = "RGBA",
19: (8)                     resampling_algorithm: int = Resampling.BICUBIC,
20: (8)                     opacity: float = 1,
21: (8)                     gloss: float = 0,
22: (8)                     shadow: float = 0,
23: (8)                     **kwargs,
24: (4)                 ):
25: (8)                     self.image = filename_or_array
26: (8)                     self.resampling_algorithm = resampling_algorithm
27: (8)                     if isinstance(filename_or_array, np.ndarray):
28: (12)                         self.size = self.image.shape[1::-1]
29: (8)                     elif isinstance(filename_or_array, (str, Path)):
30: (12)                         path = get_full_raster_image_path(filename_or_array)
31: (12)                         self.size = Image.open(path).size
32: (8)                     if width is None and height is None:
33: (12)                         width = 4 * self.size[0] / self.size[1]
34: (12)                         height = 4
35: (8)                     if height is None:
36: (12)                         height = width * self.size[1] / self.size[0]
37: (8)                     if width is None:
38: (12)                         width = height * self.size[0] / self.size[1]
39: (8)                     surface = OpenGLSurface(
40: (12)                         lambda u, v: np.array([u, v, 0]),
41: (12)                         [-width / 2, width / 2],
42: (12)                         [-height / 2, height / 2],
43: (12)                         opacity=opacity,
44: (12)                         gloss=gloss,
45: (12)                         shadow=shadow,
46: (8)                     )
47: (8)                     super().__init__(
48: (12)                         surface,
```

```

49: (12)                     self.image,
50: (12)                     image_mode=image_mode,
51: (12)                     opacity=opacity,
52: (12)                     gloss=gloss,
53: (12)                     shadow=shadow,
54: (12)                     **kwargs,
55: (8)                 )
56: (4)             def get_image_from_file(
57: (8)                 self,
58: (8)                 image_file: str | Path | np.ndarray,
59: (8)                 image_mode: str,
60: (4)             ):
61: (8)                 if isinstance(image_file, (str, Path)):
62: (12)                     return super().get_image_from_file(image_file, image_mode)
63: (8)                 else:
64: (12)                     return (
65: (16)                         Image.fromarray(image_file.astype("uint8"))
66: (16)                         .convert(image_mode)
67: (16)                         .resize(
68: (20)                             np.array(image_file.shape[:2])
69: (20)                             * 200, # assumption of 200 ppmu (pixels per manim unit)
would suffice
70: (20)                         resample=self.resampling_algorithm,
71: (16)
72: (12)                     )
-----
```

## File 94 - opengl\_three\_dimensions.py:

```

1: (0)                     from __future__ import annotations
2: (0)                     import numpy as np
3: (0)                     from manim.mobject.opengl.opengl_surface import OpenGLSurface
4: (0)                     from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLGroup,
OpenGLVMObject
5: (0)                     __all__ = ["OpenGLSurfaceMesh"]
6: (0)             class OpenGLSurfaceMesh(OpenGLGroup):
7: (4)                 def __init__(
8: (8)                     self,
9: (8)                     uv_surface,
10: (8)                     resolution=None,
11: (8)                     stroke_width=1,
12: (8)                     normal_nudge=1e-2,
13: (8)                     depth_test=True,
14: (8)                     flat_stroke=False,
15: (8)                     **kwargs,
16: (4)                 ):
17: (8)                     if not isinstance(uv_surface, OpenGLSurface):
18: (12)                         raise Exception("uv_surface must be of type OpenGLSurface")
19: (8)                     self.uv_surface = uv_surface
20: (8)                     self.resolution = resolution if resolution is not None else (21, 21)
21: (8)                     self.normal_nudge = normal_nudge
22: (8)                     super().__init__(
23: (12)                         stroke_width=stroke_width,
24: (12)                         depth_test=depth_test,
25: (12)                         flat_stroke=flat_stroke,
26: (12)                         **kwargs,
27: (8)                     )
28: (4)             def init_points(self):
29: (8)                 uv_surface = self.uv_surface
30: (8)                 full_nu, full_nv = uv_surface.resolution
31: (8)                 part_nu, part_nv = self.resolution
32: (8)                 u_indices = np.linspace(0, full_nu, part_nu).astype(int)
33: (8)                 v_indices = np.linspace(0, full_nv, part_nv).astype(int)
34: (8)                 points, du_points, dv_points =
uv_surface.get_surface_points_and_nudged_points()
35: (8)                 normals = uv_surface.get_unit_normals()
36: (8)                 nudged_points = points + self.normal_nudge * normals
37: (8)                 for ui in u_indices:
```

```

38: (12)                     path = OpenGLMobject()
39: (12)                     full_ui = full_nv * ui
40: (12)                     path.set_points_smoothly(nudged_points[full_ui : full_ui +
full_nv])
41: (12)                     self.add(path)
42: (8)                      for vi in v_indices:
43: (12)                          path = OpenGLMobject()
44: (12)                          path.set_points_smoothly(nudged_points[vi::full_nv])
45: (12)                          self.add(path)
-----
```

## File 95 - opengl\_vectorized\_mobject.py:

```

1: (0)                  from __future__ import annotations
2: (0)                  import itertools as it
3: (0)                  import operator as op
4: (0)                  from functools import reduce, wraps
5: (0)                  from typing import Callable, Iterable, Sequence
6: (0)                  import moderngl
7: (0)                  import numpy as np
8: (0)                  from manim import config
9: (0)                  from manim.constants import *
10: (0)                 from manim.mobject.opengl_mobject import OpenGLMobject, OpenGLPoint
11: (0)                 from manim.renderer.shader_wrapper import ShaderWrapper
12: (0)                 from manim.utils.bezier import (
13: (4)                     bezier,
14: (4)                     get_quadratic_approximation_of_cubic,
15: (4)                     get_smooth_cubic_bezier_handle_points,
16: (4)                     integer_interpolate,
17: (4)                     interpolate,
18: (4)                     partial_quadratic_bezier_points,
19: (4)                     proportions_along_bezier_curve_for_point,
20: (4)                     quadratic_bezier_remap,
21: (0)
22: (0)                 )
23: (0)                 from manim.utils.color import BLACK, WHITE, ManimColor, ParsableManimColor
24: (0)                 from manim.utils.config_ops import _Data
25: (0)                 from manim.utils.iterables import listify, make_even,
26: (0)                 resize_with_interpolation
27: (0)                 from manim.utils.space_ops import (
28: (4)                     angle_between_vectors,
29: (4)                     cross2d,
30: (4)                     earclip_triangulation,
31: (4)                     get_unit_normal,
32: (4)                     shoelace_direction,
33: (4)                     z_to_vector,
34: (0)             )
35: (0)             __all__ = [
36: (4)                 "triggers_refreshed_triangulation",
37: (4)                 "OpenGLMobject",
38: (4)                 "OpenGLGroup",
39: (4)                 "OpenGLVectorizedPoint",
40: (4)                 "OpenGLCurvesAsSubmobjects",
41: (4)                 "OpenGLDashedVMobject",
42: (0)
43: (0)             def triggers_refreshed_triangulation(func):
44: (4)                 @wraps(func)
45: (4)                 def wrapper(self, *args, **kwargs):
46: (8)                     old_points = np.empty((0, 3))
47: (8)                     for mob in self.family_members_with_points():
48: (12)                         old_points = np.concatenate((old_points, mob.points), axis=0)
49: (8)                         func(self, *args, **kwargs)
50: (8)                         new_points = np.empty((0, 3))
51: (8)                         for mob in self.family_members_with_points():
52: (12)                             new_points = np.concatenate((new_points, mob.points), axis=0)
53: (8)                             if not np.array_equal(new_points, old_points):
54: (12)                                 self.refresh_triangulation()
55: (12)                                 self.refresh_unit_normal()
56: (8)             return self

```

```

55: (4)             return wrapper
56: (0)         class OpenGLVMobject(OpenGLMobject):
57: (4)             """A vectorized mobject."""
58: (4)             fill_dtype = [
59: (8)                 ("point", np.float32, (3,)),
60: (8)                 ("unit_normal", np.float32, (3,)),
61: (8)                 ("color", np.float32, (4,)),
62: (8)                 ("vert_index", np.float32, (1,)),
63: (4)             ]
64: (4)             stroke_dtype = [
65: (8)                 ("point", np.float32, (3,)),
66: (8)                 ("prev_point", np.float32, (3,)),
67: (8)                 ("next_point", np.float32, (3,)),
68: (8)                 ("unit_normal", np.float32, (3,)),
69: (8)                 ("stroke_width", np.float32, (1,)),
70: (8)                 ("color", np.float32, (4,)),
71: (4)             ]
72: (4)             stroke_shader_folder = "quadratic_bezier_stroke"
73: (4)             fill_shader_folder = "quadratic_bezier_fill"
74: (4)             fill_rgba = _Data()
75: (4)             stroke_rgba = _Data()
76: (4)             stroke_width = _Data()
77: (4)             unit_normal = _Data()
78: (4)             def __init__(
79: (8)                 self,
80: (8)                 fill_color: ParsableManimColor | None = None,
81: (8)                 fill_opacity: float = 0.0,
82: (8)                 stroke_color: ParsableManimColor | None = None,
83: (8)                 stroke_opacity: float = 1.0,
84: (8)                 stroke_width: float = DEFAULT_STROKE_WIDTH,
85: (8)                 draw_stroke_behind_fill: bool = False,
86: (8)                 pre_function_handle_to_anchor_scale_factor: float = 0.01,
87: (8)                 make_smooth_after_applying_functions: float = False,
88: (8)                 background_image_file: str | None = None,
89: (8)                 tolerance_for_point_equality: float = 1e-8,
90: (8)                 n_points_per_curve: int = 3,
91: (8)                 long_lines: bool = False,
92: (8)                 should_subdivide_sharp_curves: bool = False,
93: (8)                 should_remove_null_curves: bool = False,
94: (8)                 joint_type: LineJointType | None = None,
95: (8)                 flat_stroke: bool = True,
96: (8)                 render_primitive=moderngl.TRIANGLES,
97: (8)                 triangulation_locked: bool = False,
98: (8)                 **kwargs,
99: (4)             ):
100: (8)                 self.data = {}
101: (8)                 self.fill_opacity = fill_opacity
102: (8)                 self.stroke_opacity = stroke_opacity
103: (8)                 self.stroke_width = stroke_width
104: (8)                 self.draw_stroke_behind_fill = draw_stroke_behind_fill
105: (8)                 self.pre_function_handle_to_anchor_scale_factor =
106: (12)                     pre_function_handle_to_anchor_scale_factor
107: (8)
108: (8)                 self.make_smooth_after_applying_functions =
make_smooth_after_applying_functions
109: (8)                     self.background_image_file = background_image_file
110: (8)                     self.tolerance_for_point_equality = tolerance_for_point_equality
111: (8)                     self.n_points_per_curve = n_points_per_curve
112: (8)                     self.long_lines = long_lines
113: (8)                     self.should_subdivide_sharp_curves = should_subdivide_sharp_curves
114: (8)                     self.should_remove_null_curves = should_remove_null_curves
115: (8)                     if joint_type is None:
116: (12)                         joint_type = LineJointType.AUTO
117: (8)                     self.joint_type = joint_type
118: (8)                     self.flat_stroke = flat_stroke
119: (8)                     self.render_primitive = render_primitive
120: (8)                     self.triangulation_locked = triangulation_locked
121: (8)                     self.needs_new_triangulation = True
122: (8)                     self.triangulation = np.zeros(0, dtype="i4")

```

```

123: (8)           self.orientation = 1
124: (8)           self.fill_data = None
125: (8)           self.stroke_data = None
126: (8)           self.fill_shader_wrapper = None
127: (8)           self.stroke_shader_wrapper = None
128: (8)           self.init_shader_data()
129: (8)           super().__init__(**kwargs)
130: (8)           self.refresh_unit_normal()
131: (8)           if fill_color is not None:
132: (12)             self.fill_color = ManimColor.parse(fill_color)
133: (8)           if stroke_color is not None:
134: (12)             self.stroke_color = ManimColor.parse(stroke_color)
135: (4)           def get_group_class(self):
136: (8)             return OpenGLGroup
137: (4)           @staticmethod
138: (4)           def get_mobject_type_class():
139: (8)             return OpenGLMobject
140: (4)           def init_data(self):
141: (8)             super().init_data()
142: (8)             self.data.pop("rgbas")
143: (8)             self.fill_rgba = np.zeros((1, 4))
144: (8)             self.stroke_rgba = np.zeros((1, 4))
145: (8)             self.unit_normal = np.zeros((1, 3))
146: (4)           def init_colors(self):
147: (8)             self.set_fill(
148: (12)               color=self.fill_color or self.color,
149: (12)               opacity=self.fill_opacity,
150: (8)             )
151: (8)             self.set_stroke(
152: (12)               color=self.stroke_color or self.color,
153: (12)               width=self.stroke_width,
154: (12)               opacity=self.stroke_opacity,
155: (12)               background=self.draw_stroke_behind_fill,
156: (8)             )
157: (8)             self.set_gloss(self.gloss)
158: (8)             self.set_flat_stroke(self.flat_stroke)
159: (8)             return self
160: (4)           def set_fill(
161: (8)             self,
162: (8)             color: ParsableManimColor | None = None,
163: (8)             opacity: float | None = None,
164: (8)             recurse: bool = True,
165: (4)           ) -> OpenGLMobject:
166: (8)             """Set the fill color and fill opacity of a :class:`OpenGLMobject` .
167: (8)             Parameters
168: (8)             -----
169: (8)             color
170: (12)               Fill color of the :class:`OpenGLMobject` .
171: (8)             opacity
172: (12)               Fill opacity of the :class:`OpenGLMobject` .
173: (8)             recurse
174: (12)               If ``True``, the fill color of all submobjects is also set.
175: (8)
176: (8)
177: (8)
178: (12)
179: (8)
180: (8)
181: (8)             .. manim:: SetFill
182: (12)               :save_last_frame:
183: (12)
184: (16)               class SetFill(Scene):
185: (20)                 def construct(self):
186: (20)                   square = Square().scale(2).set_fill(WHITE,1)
187: (20)                   circle1 = Circle().set_fill(GREEN,0.8)
188: (20)                   circle2 = Circle().set_fill(YELLOW) # No fill_opacity
189: (20)                   circle3 = Circle().set_fill(color = '#FF2135', opacity =
0.2)
190: (20)                   group = Group(circle1,circle2,circle3).arrange()
190: (20)                   self.add(square)

```

```

191: (20)                                     self.add(group)
192: (8)          See Also
193: (8)
194: (8)          -----
195: (8)          :meth:`~.OpenGLMobject.set_style`
196: (8)          """
197: (12)          if opacity is not None:
198: (8)              self.fill_opacity = opacity
199: (12)          if recurse:
200: (16)              for submobject in self.submobjects:
201: (8)                  submobject.set_fill(color, opacity, recurse)
202: (8)          self.set_rgba_array(color, opacity, "fill_rgba", recurse)
203: (8)          return self
204: (4)      def set_stroke(
205: (8)          self,
206: (8)          color=None,
207: (8)          width=None,
208: (8)          opacity=None,
209: (8)          background=None,
210: (4)          recurse=True,
211: (8)      ):
212: (12)          if opacity is not None:
213: (8)              self.stroke_opacity = opacity
214: (12)          if recurse:
215: (16)              for submobject in self.submobjects:
216: (20)                  submobject.set_stroke(
217: (20)                      color=color,
218: (20)                      width=width,
219: (20)                      opacity=opacity,
220: (20)                      background=background,
221: (16)                      recurse=recurse,
222: (8)                  )
223: (8)          self.set_rgba_array(color, opacity, "stroke_rgba", recurse)
224: (12)          if width is not None:
225: (16)              for mob in self.get_family(recurse):
226: (8)                  mob.stroke_width = np.array([[width] for width in
listify(width)])
227: (8)
228: (16)
229: (8)
230: (4)      def set_style(
231: (8)          self,
232: (8)          fill_color=None,
233: (8)          fill_opacity=None,
234: (8)          fill_rgba=None,
235: (8)          stroke_color=None,
236: (8)          stroke_opacity=None,
237: (8)          stroke_rgba=None,
238: (8)          stroke_width=None,
239: (8)          gloss=None,
240: (8)          shadow=None,
241: (8)          recurse=True,
242: (4)      ):
243: (8)          if fill_rgba is not None:
244: (12)              self.fill_rgba = resize_with_interpolation(fill_rgba,
245: (8)          else:
246: (12)              self.set_fill(color=fill_color, opacity=fill_opacity,
247: (8)          if stroke_rgba is not None:
248: (12)              self.stroke_rgba = resize_with_interpolation(stroke_rgba,
249: (12)                  self.set_stroke(width=stroke_width)
250: (8)          else:
251: (12)              self.set_stroke(
252: (16)                  color=stroke_color,
253: (16)                  width=stroke_width,
254: (16)                  opacity=stroke_opacity,
255: (16)                  recurse=recurse,

```

```

256: (12) )
257: (8)     if gloss is not None:
258: (12)         self.set_gloss(gloss, recurse=recurse)
259: (8)     if shadow is not None:
260: (12)         self.set_shadow(shadow, recurse=recurse)
261: (8)     return self
262: (4) def get_style(self):
263: (8)     return {
264: (12)         "fill_rgba": self.fill_rgba,
265: (12)         "stroke_rgba": self.stroke_rgba,
266: (12)         "stroke_width": self.stroke_width,
267: (12)         "gloss": self.gloss,
268: (12)         "shadow": self.shadow,
269: (8)     }
270: (4) def match_style(self, vmobject, recurse=True):
271: (8)     vmobject_style = vmobject.get_style()
272: (8)     if config.renderer == RendererType.OPENGL:
273: (12)         vmobject_style["stroke_width"] = vmobject_style["stroke_width"][0]
274: [0]
275: (8)     self.set_style(**vmobject_style, recurse=False)
276: (8)     if recurse:
277: (12)         submobs1, submobs2 = self.submobjects, vmobject.submobjects
278: (12)         if len(submobs1) == 0:
279: (16)             return self
280: (12)         elif len(submobs2) == 0:
281: (16)             submobs2 = [vmobject]
282: (12)         for sm1, sm2 in zip(*make_even(submobs1, submobs2)):
283: (16)             sm1.match_style(sm2)
284: (8)     return self
285: (4) def set_color(self, color, opacity=None, recurse=True):
286: (8)     if opacity is not None:
287: (12)         self.opacity = opacity
288: (8)         self.set_fill(color, opacity=opacity, recurse=recurse)
289: (8)         self.set_stroke(color, opacity=opacity, recurse=recurse)
290: (8)     return self
291: (4) def set_opacity(self, opacity, recurse=True):
292: (8)     self.set_fill(opacity=opacity, recurse=recurse)
293: (8)     self.set_stroke(opacity=opacity, recurse=recurse)
294: (8)     return self
295: (4) def fade(self, darkness=0.5, recurse=True):
296: (8)     factor = 1.0 - darkness
297: (12)     self.set_fill(
298: (12)         opacity=factor * self.get_fill_opacity(),
299: (8)         recurse=False,
300: (8)     )
301: (12)     self.set_stroke(
302: (12)         opacity=factor * self.get_stroke_opacity(),
303: (8)         recurse=False,
304: (8)     )
305: (8)     super().fade(darkness, recurse)
306: (8)     return self
307: (4) def get_fill_colors(self):
308: (8)     return [ManimColor.from_rgb(rgb[:3]) for rgb in self.fill_rgba]
309: (4) def get_fill_opacities(self):
310: (8)     return self.fill_rgba[:, 3]
311: (4) def get_stroke_colors(self):
312: (8)     return [ManimColor.from_rgb(rgb[:3]) for rgb in self.stroke_rgba]
313: (4) def get_stroke_opacities(self):
314: (8)     return self.stroke_rgba[:, 3]
315: (4) def get_stroke_widths(self):
316: (8)     return self.stroke_width
317: (4) def get_fill_color(self):
318: (8)     """
319: (8)         If there are multiple colors (for gradient)
320: (8)         this returns the first one
321: (8)     """
322: (4)     return self.get_fill_colors()[0]
323: (4) def get_fill_opacity(self):
324: (8)     """

```

```

324: (8)             If there are multiple opacities, this returns the
325: (8)             first
326: (8)
327: (8)             """
328: (4)             return self.get_fill_opacities()[0]
329: (8)         def get_stroke_color(self):
330: (4)             return self.get_stroke_colors()[0]
331: (8)         def get_stroke_width(self):
332: (4)             return self.get_stroke_widths()[0]
333: (8)         def get_stroke_opacity(self):
334: (4)             return self.get_stroke_opacities()[0]
335: (8)         def get_color(self):
336: (12)             if self.has_stroke():
337: (8)                 return self.get_stroke_color()
338: (4)             return self.get_fill_color()
339: (8)         def get_colors(self):
340: (12)             if self.has_stroke():
341: (8)                 return self.get_stroke_colors()
342: (4)             return self.get_fill_colors()
343: (4)             stroke_color = property(get_stroke_color, set_stroke)
344: (4)             color = property(get_color, set_color)
345: (4)             fill_color = property(get_fill_color, set_fill)
346: (8)         def has_stroke(self):
347: (8)             stroke_widths = self.get_stroke_widths()
348: (8)             stroke_opacities = self.get_stroke_opacities()
349: (12)             return (
350: (12)                 stroke_widths is not None
351: (12)                 and stroke_opacities is not None
352: (12)                 and any(stroke_widths)
353: (12)                 and any(stroke_opacities)
354: (8)             )
355: (4)             def has_fill(self):
356: (8)                 fill_opacities = self.get_fill_opacities()
357: (8)                 return fill_opacities is not None and any(fill_opacities)
358: (4)         def get_opacity(self):
359: (8)             if self.has_fill():
360: (8)                 return self.get_fill_opacity()
361: (4)             return self.get_stroke_opacity()
362: (8)         def set_flat_stroke(self, flat_stroke=True, recurse=True):
363: (12)             for mob in self.get_family(recurse):
364: (8)                 mob.flat_stroke = flat_stroke
365: (8)             return self
366: (4)         def get_flat_stroke(self):
367: (8)             return self.flat_stroke
368: (4)         def set_anchors_and_handles(self, anchors1, handles, anchors2):
369: (8)             assert len(anchors1) == len(handles) == len(anchors2)
370: (8)             nppc = self.n_points_per_curve
371: (8)             new_points = np.zeros((nppc * len(anchors1), self.dim))
372: (8)             arrays = [anchors1, handles, anchors2]
373: (12)             for index, array in enumerate(arrays):
374: (8)                 new_points[index::nppc] = array
375: (8)             self.set_points(new_points)
376: (8)             return self
377: (4)         def start_new_path(self, point):
378: (8)             assert self.get_num_points() % self.n_points_per_curve == 0
379: (8)             self.append_points([point])
380: (8)             return self
381: (4)         def add_cubic_bezier_curve(self, anchor1, handle1, handle2, anchor2):
382: (8)             new_points = get_quadratic_approximation_of_cubic(
383: (12)                 anchor1,
384: (12)                 handle1,
385: (12)                 handle2,
386: (12)                 anchor2,
387: (8)             )
388: (4)             self.append_points(new_points)
389: (8)         def add_cubic_bezier_curve_to(self, handle1, handle2, anchor):
390: (8)             """
391: (8)             Add cubic bezier curve to the path.
392: (8)             """

```

```

393: (8)             quadratic_approx = get_quadratic_approximation_of_cubic(
394: (12)             self.get_last_point(),
395: (12)             handle1,
396: (12)             handle2,
397: (12)             anchor,
398: (8)         )
399: (8)         if self.has_new_path_started():
400: (12)             self.append_points(quadratic_approx[1:])
401: (8)
402: (12)         else:
403: (4)             self.append_points(quadratic_approx)
404: (8)
405: (8)
406: (12)     def add_quadratic_bezier_curve_to(self, handle, anchor):
407: (8)         self.throw_error_if_no_points()
408: (12)         if self.has_new_path_started():
409: (12)             self.append_points([handle, anchor])
410: (8)
411: (8)         Parameters
412: (8)         -----
413: (8)         point
414: (12)             end of the straight line.
415: (8)
416: (8)         end = self.points[-1]
417: (8)         alphas = np.linspace(0, 1, self.n_points_per_curve)
418: (8)         if self.long_lines:
419: (12)             halfway = interpolate(end, point, 0.5)
420: (12)             points = [interpolate(end, halfway, a) for a in alphas] + [
421: (16)                 interpolate(halfway, point, a) for a in alphas
422: (12)             ]
423: (8)
424: (12)         else:
425: (8)             points = [interpolate(end, point, a) for a in alphas]
426: (12)         if self.has_new_path_started():
427: (8)             points = points[1:]
428: (8)         self.append_points(points)
429: (8)
430: (12)     def add_smooth_curve_to(self, point):
431: (8)         if self.has_new_path_started():
432: (8)             self.add_line_to(point)
433: (8)
434: (12)         else:
435: (12)             self.throw_error_if_no_points()
436: (8)             new_handle = self.get_reflection_of_last_handle()
437: (12)             self.add_quadratic_bezier_curve_to(new_handle, point)
438: (8)
439: (12)     def add_smooth_cubic_curve_to(self, handle, point):
440: (8)         self.throw_error_if_no_points()
441: (12)         new_handle = self.get_reflection_of_last_handle()
442: (8)         self.add_cubic_bezier_curve_to(new_handle, handle, point)
443: (8)
444: (12)     def has_new_path_started(self):
445: (8)         return self.get_num_points() % self.n_points_per_curve == 1
446: (8)
447: (12)     def get_last_point(self):
448: (8)         return self.points[-1]
449: (8)
450: (12)     def get_reflection_of_last_handle(self):
451: (8)         points = self.points
452: (8)         return 2 * points[-1] - points[-2]
453: (8)
454: (12)     def close_path(self):
455: (8)         if not self.is_closed():
456: (12)             self.add_line_to(self.get_subpaths()[-1][0])
457: (12)
458: (16)     def is_closed(self):
459: (8)         return self.consider_points_equals(self.points[0], self.points[-1])
460: (8)
461: (12)     def subdivide_sharp_curves(self, angle_threshold=30 * DEGREES,
462: (8)         recurse=True):
463: (8)         vmobs = [vm for vm in self.get_family(recurse) if vm.has_points()]
464: (8)         for vmob in vmobs:
465: (12)             new_points = []
466: (12)             for tup in vmob.get_bezier_tuples():
467: (16)                 angle = angle_between_vectors(tup[1] - tup[0], tup[2] -
468: (16)                   tup[1])

```

```

459: (16)             if angle > angle_threshold:
460: (20)                 n = int(np.ceil(angle / angle_threshold))
461: (20)                 alphas = np.linspace(0, 1, n + 1)
462: (20)                 new_points.extend(
463: (24)                     [
464: (28)                         partial_quadratic_bezier_points(tup, a1, a2)
465: (28)                         for a1, a2 in zip(alphas, alphas[1:])
466: (24)                     ],
467: (20)                 )
468: (16)             else:
469: (20)                 new_points.append(tup)
470: (12)             vmob.set_points(np.vstack(new_points))
471: (8)         return self
472: (4)     def add_points_as_corners(self, points):
473: (8)         for point in points:
474: (12)             self.add_line_to(point)
475: (8)         return points
476: (4)     def set_points_as_corners(self, points: Iterable[float]) ->
OpenGLMobject:
477: (8)         """Given an array of points, set them as corner of the vmobject.
478: (8)             To achieve that, this algorithm sets handles aligned with the anchors
such that the resultant bezier curve will be the segment
479: (8)             between the two anchors.
480: (8)         Parameters
481: (8)             -----
482: (8)             points
483: (12)                 Array of points that will be set as corners.
484: (8)         Returns
485: (8)             -----
486: (8)             OpenGLMobject
487: (12)                 self. For chaining purposes.
488: (8)             """
489: (8)             nppc = self.n_points_per_curve
490: (8)             points = np.array(points)
491: (8)             self.set_anchors_and_handles(
492: (12)                 *(interpolate(points[:-1], points[1:], a) for a in np.linspace(0,
1, nppc)))
493: (8)
494: (8)             return self
495: (4)     def set_points_smoothly(self, points, true_smooth=False):
496: (8)         self.set_points_as_corners(points)
497: (8)         self.make_smooth()
498: (8)         return self
499: (4)     def change_anchor_mode(self, mode):
500: (8)         """Changes the anchor mode of the bezier curves. This will modify the
handles.
501: (8)             There can be only three modes, "jagged", "approx_smooth" and
"true_smooth".
502: (8)         Returns
503: (8)             -----
504: (8)             OpenGLMobject
505: (12)                 For chaining purposes.
506: (8)             """
507: (8)             assert mode in ("jagged", "approx_smooth", "true_smooth")
508: (8)             nppc = self.n_points_per_curve
509: (8)             for submob in self.family_members_with_points():
510: (12)                 subpaths = submob.get_subpaths()
511: (12)                 submob.clear_points()
512: (12)                 for subpath in subpaths:
513: (16)                     anchors = np.vstack([subpath[::-nppc], subpath[-1:]])
514: (16)                     new_subpath = np.array(subpath)
515: (16)                     if mode == "approx_smooth":
516: (20)                         new_subpath[1::nppc] =
get_smooth_quadratic_bezier_handle_points(
517: (24)                             anchors,
518: (20)                         )
519: (16)                     elif mode == "true_smooth":
520: (20)                         h1, h2 = get_smooth_cubic_bezier_handle_points(anchors)
521: (20)                         new_subpath = get_quadratic_approximation_of_cubic(

```

```

522: (24)                                     anchors[:-1],
523: (24)                                     h1,
524: (24)                                     h2,
525: (24)                                     anchors[1:],
526: (20)                                     )
527: (16)                                     elif mode == "jagged":
528: (20)   new_subpath[1::nppc] = 0.5 * (anchors[:-1] + anchors[1:])
529: (16)   submob.append_points(new_subpath)
530: (12)   submob.refresh_triangulation()
531: (8)   return self
532: (4) def make_smooth(self):
533: (8)     """
534: (8)         This will double the number of points in the mobject,
535: (8)         so should not be called repeatedly. It also means
536: (8)         transforming between states before and after calling
537: (8)         this might have strange artifacts
538: (8)     """
539: (8)         self.change_anchor_mode("true_smooth")
540: (8)         return self
541: (4) def make_approximately_smooth(self):
542: (8)     """
543: (8)         Unlike make_smooth, this will not change the number of
544: (8)         points, but it also does not result in a perfectly smooth
545: (8)         curve. It's most useful when the points have been
546: (8)         sampled at a not-too-low rate from a continuous function,
547: (8)         as in the case of ParametricCurve
548: (8)     """
549: (8)         self.change_anchor_mode("approx_smooth")
550: (8)         return self
551: (4) def make_jagged(self):
552: (8)     self.change_anchor_mode("jagged")
553: (8)     return self
554: (4) def add_subpath(self, points):
555: (8)     assert len(points) % self.n_points_per_curve == 0
556: (8)     self.append_points(points)
557: (8)     return self
558: (4) def append_vectorized_mobject(self, vectorized_mobject):
559: (8)     new_points = list(vectorized_mobject.points)
560: (8)     if self.has_new_path_started():
561: (12)         self.resize_data(len(self.points) - 1))
562: (8)     self.append_points(new_points)
563: (8)     return self
564: (4) def consider_points_equals(self, p0, p1):
565: (8)     return np.linalg.norm(p1 - p0) < self.tolerance_for_point_equality
566: (4) def force_direction(self, target_direction: str):
567: (8)     """Makes sure that points are either directed clockwise or
568: (8)     counterclockwise.
569: (8)     Parameters
570: (8)     -----
571: (8)     target_direction
572: (12)         Either ``"CW"`` or ``"CCW"``.
573: (8)     """
574: (8)     if target_direction not in ("CW", "CCW"):
575: (12)         raise ValueError('Invalid input for force_direction. Use "CW" or
576: (8)         "CCW"')
577: (12)
578: (8)
579: (4) def reverse_direction(self):
580: (8)     """Reverts the point direction by inverting the point order.
581: (8)     Returns
582: (8)     -----
583: (8)     :class:`OpenGLMobject`
584: (12)         Returns self.
585: (8)     Examples
586: (8)     -----
587: (8)     .. manim:: ChangeOfDirection
588: (12)         class ChangeOfDirection(Scene):
589: (16)             def construct(self):

```

```

590: (20)                               ccw = RegularPolygon(5)
591: (20)                               ccw.shift(LEFT)
592: (20)                               cw = RegularPolygon(5)
593: (20)                               cw.shift(RIGHT).reverse_direction()
594: (20)                               self.play(Create(ccw), Create(cw),
595: (20)   run_time=4)
596: (8)                                """
597: (8)                                self.set_points(self.points[::-1])
598: (8)                                return self
599: (4) def get_bezier_tuples_from_points(self, points):
600: (8)     nppc = self.n_points_per_curve
601: (8)     remainder = len(points) % nppc
602: (8)     points = points[: len(points) - remainder]
603: (8)     return points.reshape((-1, nppc, 3))
604: (4) def get_bezier_tuples(self):
605: (8)     return self.get_bezier_tuples_from_points(self.points)
606: (4) def get_subpaths_from_points(self, points):
607: (8)     nppc = self.n_points_per_curve
608: (8)     diffs = points[nppc - 1 : -1 : nppc] - points[nppc::nppc]
609: (8)     splits = (diffs * diffs).sum(1) > self.tolerance_for_point_equality
610: (8)     split_indices = np.arange(nppc, len(points), nppc, dtype=int)[splits]
611: (8)     split_indices = [0, *split_indices, len(points)]
612: (8)     return [
613: (12)         points[i1:i2]
614: (12)         for i1, i2 in zip(split_indices, split_indices[1:])
615: (12)         if (i2 - i1) >= nppc
616: (8)     ]
617: (4) def get_subpaths(self):
618: (8)     """Returns subpaths formed by the curves of the OpenGLVObject.
619: (8)     Subpaths are ranges of curves with each pair of consecutive
620: (8)     curves having their end/start points coincident.
621: (8)     Returns
622: (8)     -----
623: (8)     Tuple
624: (12)     subpaths.
625: (8)     """
626: (8)     return self.get_subpaths_from_points(self.points)
627: (4) def get_nth_curve_points(self, n: int) -> np.ndarray:
628: (8)     """Returns the points defining the nth curve of the vobject.
629: (8)     Parameters
629: (8)     -----
630: (8)     n
631: (8)         index of the desired bezier curve.
632: (12)     Returns
633: (8)     -----
634: (8)     np.ndarray
635: (8)         points defininf the nth bezier curve (anchors, handles)
636: (12)         """
637: (8)         assert n < self.get_num_curves()
638: (8)         nppc = self.n_points_per_curve
639: (8)         return self.points[nppc * n : nppc * (n + 1)]
640: (8) def get_nth_curve_function(self, n: int) -> Callable[[float], np.ndarray]:
641: (4)     """Returns the expression of the nth curve.
642: (8)     Parameters
643: (8)     -----
644: (8)     n
645: (8)         index of the desired curve.
646: (12)     Returns
647: (8)     -----
648: (8)     typing.Callable[float]
649: (8)         expression of the nth bezier curve.
650: (12)         """
651: (8)         return bezier(self.get_nth_curve_points(n))
652: (8) def get_nth_curve_function_with_length(
653: (4)     self,
654: (8)     n: int,
655: (8)     sample_points: int | None = None,
656: (8) ) -> tuple[Callable[[float], np.ndarray], float]:
657: (4)     """Returns the expression of the nth curve along with its
658: (8)         """

```

```

(approximate) length.
659: (8)             Parameters
660: (8)             -----
661: (8)             n
662: (12)            The index of the desired curve.
663: (8)             sample_points
664: (12)            The number of points to sample to find the length.
665: (8)             Returns
666: (8)             -----
667: (8)             curve : Callable[[float], np.ndarray]
668: (12)            The function for the nth curve.
669: (8)             length : :class:`float`
670: (12)            The length of the nth curve.
671: (8)             """
672: (8)             if sample_points is None:
673: (12)                sample_points = 10
674: (8)                curve = self.get_nth_curve_function(n)
675: (8)                norms = self.get_nth_curve_length_pieces(n, sample_points)
676: (8)                length = np.sum(norms)
677: (8)                return curve, length
678: (4)             def get_num_curves(self) -> int:
679: (8)               """Returns the number of curves of the vobject.
680: (8)               Returns
681: (8)               -----
682: (8)               int
683: (12)              number of curves. of the vobject.
684: (8)               """
685: (8)               return self.get_num_points() // self.n_points_per_curve
686: (4)             def get_nth_curve_length(
687: (8)               self,
688: (8)               n: int,
689: (8)               sample_points: int | None = None,
690: (4)             ) -> float:
691: (8)               """Returns the (approximate) length of the nth curve.
692: (8)               Parameters
693: (8)               -----
694: (8)               n
695: (12)              The index of the desired curve.
696: (8)               sample_points
697: (12)              The number of points to sample to find the length.
698: (8)               Returns
699: (8)               -----
700: (8)               length : :class:`float`
701: (12)              The length of the nth curve.
702: (8)               """
703: (8)               _, length = self.get_nth_curve_function_with_length(n, sample_points)
704: (8)               return length
705: (4)             def get_curve_functions(
706: (8)               self,
707: (4)             ) -> Iterable[Callable[[float], np.ndarray]]:
708: (8)               """Gets the functions for the curves of the mobject.
709: (8)               Returns
710: (8)               -----
711: (8)               Iterable[Callable[[float], np.ndarray]]
712: (12)              The functions for the curves.
713: (8)               """
714: (8)               num_curves = self.get_num_curves()
715: (8)               for n in range(num_curves):
716: (12)                 yield self.get_nth_curve_function(n)
717: (4)             def get_nth_curve_length_pieces(
718: (8)               self,
719: (8)               n: int,
720: (8)               sample_points: int | None = None,
721: (4)             ) -> np.ndarray:
722: (8)               """Returns the array of short line lengths used for length
approximation.
723: (8)             Parameters
724: (8)             -----
725: (8)             n

```

```

726: (12)                      The index of the desired curve.
727: (8)                       sample_points
728: (12)                      The number of points to sample to find the length.
729: (8)                       Returns
730: (8)                       -----
731: (8)                       np.ndarray
732: (12)                      The short length-pieces of the nth curve.
733: (8)                       """
734: (8)                       if sample_points is None:
735: (12)                         sample_points = 10
736: (8)                         curve = self.get_nth_curve_function(n)
737: (8)                         points = np.array([curve(a) for a in np.linspace(0, 1,
sample_points)])
738: (8)                         diffs = points[1:] - points[:-1]
739: (8)                         norms = np.apply_along_axis(np.linalg.norm, 1, diffs)
740: (8)                         return norms
741: (4)  def get_curve_functions_with_lengths(
742: (8)    self, **kwargs
743: (4)  ) -> Iterable[tuple[Callable[[float], np.ndarray], float]]:
744: (8)    """Gets the functions and lengths of the curves for the mobobject.
745: (8)    Parameters
746: (8)    -----
747: (8)    **kwargs
748: (12)      The keyword arguments passed to
:meth:`get_nth_curve_function_with_length`
749: (8)    Returns
750: (8)    -----
751: (8)    Iterable[Tuple[Callable[[float], np.ndarray], float]]
752: (12)      The functions and lengths of the curves.
753: (8)    """
754: (8)    num_curves = self.get_num_curves()
755: (8)    for n in range(num_curves):
756: (12)      yield self.get_nth_curve_function_with_length(n, **kwargs)
757: (4)  def point_from_proportion(self, alpha: float) -> np.ndarray:
758: (8)    """Gets the point at a proportion along the path of the
:class:`OpenGLMobject`.
759: (8)    Parameters
760: (8)    -----
761: (8)    alpha
762: (12)      The proportion along the the path of the :class:`OpenGLMobject`.
763: (8)    Returns
764: (8)    -----
765: (8)    :class:`numpy.ndarray`
766: (12)      The point on the :class:`OpenGLMobject`.
767: (8)    Raises
768: (8)    -----
769: (8)    :exc:`ValueError`
770: (12)      If ``alpha`` is not between 0 and 1.
771: (8)    :exc:`Exception`
772: (12)      If the :class:`OpenGLMobject` has no points.
773: (8)    """
774: (8)    if alpha < 0 or alpha > 1:
775: (12)      raise ValueError(f"Alpha {alpha} not between 0 and 1.")
776: (8)    self.throw_error_if_no_points()
777: (8)    if alpha == 1:
778: (12)      return self.points[-1]
779: (8)    curves_and_lengths = tuple(self.get_curve_functions_with_lengths())
780: (8)    target_length = alpha * np.sum(
781: (12)      np.fromiter((length for _, length in curves_and_lengths),
dtype=np.float64)
782: (8)    )
783: (8)    current_length = 0
784: (8)    for curve, length in curves_and_lengths:
785: (12)      if current_length + length >= target_length:
786: (16)        if length != 0:
787: (20)          residue = (target_length - current_length) / length
788: (16)        else:
789: (20)          residue = 0
790: (16)    return curve(residue)

```

```

791: (12)                     current_length += length
792: (4)          def proportion_from_point(
793: (8)            self,
794: (8)            point: Iterable[float | int],
795: (4)          ) -> float:
796: (8)            """Returns the proportion along the path of the
:class:`OpenGLMobject`  

797: (8)            a particular given point is at.
798: (8)          Parameters
799: (8)            -----
800: (8)            point
801: (12)              The Cartesian coordinates of the point which may or may not lie on
the :class:`OpenGLMobject`  

802: (8)          Returns
803: (8)            -----
804: (8)            float
805: (12)              The proportion along the path of the :class:`OpenGLMobject`.  

806: (8)          Raises
807: (8)            -----
808: (8)            :exc:`ValueError`
809: (12)              If ``point`` does not lie on the curve.
810: (8)            :exc:`Exception`
811: (12)              If the :class:`OpenGLMobject` has no points.
812: (8)
813: (8)            """
814: (8)            self.throw_error_if_no_points()
815: (8)            num_curves = self.get_num_curves()
816: (8)            total_length = self.get_arc_length()
817: (8)            target_length = 0
818: (12)            for n in range(num_curves):
819: (12)              control_points = self.get_nth_curve_points(n)
820: (12)              length = self.get_nth_curve_length(n)
proportions_along_bezier_curve_for_point(
821: (16)                proportions_along_bezier =
822: (16)                  point,
823: (12)                  control_points,
824: (12)                  )
825: (16)                  if len(proportions_along_bezier) > 0:
826: (16)                    proportion_along_nth_curve = max(proportions_along_bezier)
827: (16)                    target_length += length * proportion_along_nth_curve
828: (12)                    break
829: (8)                  target_length += length
else:
830: (12)                    raise ValueError(f"Point {point} does not lie on this curve.")
831: (8)            alpha = target_length / total_length
832: (8)            return alpha
def get_anchors_and_handles(self) -> Iterable[np.ndarray]:
833: (4)            """
834: (8)              Returns anchors1, handles, anchors2,
835: (8)              where (anchors1[i], handles[i], anchors2[i])
836: (8)              will be three points defining a quadratic bezier curve
837: (8)              for any i in range(0, len(anchors1))
838: (8)
839: (8)
840: (8)            nppc = self.n_points_per_curve
841: (8)            points = self.points
842: (8)            return [points[i::nppc] for i in range(nppc)]
def get_start_anchors(self) -> np.ndarray:
843: (4)            """
844: (8)              Returns the start anchors of the bezier curves.
845: (8)
846: (8)
847: (8)
848: (12)              np.ndarray
849: (8)                  Starting anchors
850: (8)
851: (4)            return self.points[0 :: self.n_points_per_curve]
def get_end_anchors(self) -> np.ndarray:
852: (8)            """
853: (8)              Returns the starting anchors of the bezier curves.
854: (8)
855: (8)
856: (12)              np.ndarray
Starting anchors

```

```

857: (8)             """
858: (8)             nppc = self.n_points_per_curve
859: (8)             return self.points[nppc - 1 :: nppc]
860: (4)             def get_anchors(self) -> Iterable[np.ndarray]:
861: (8)                 """Returns the anchors of the curves forming the OpenGLMobject.
862: (8)                 Returns
863: (8)                 -----
864: (8)                 Iterable[np.ndarray]
865: (12)                The anchors.
866: (8)                """
867: (8)                points = self.points
868: (8)                if len(points) == 1:
869: (12)                   return points
870: (8)                s = self.get_start_anchors()
871: (8)                e = self.get_end_anchors()
872: (8)                return list(it.chain.from_iterable(zip(s, e)))
873: (4)                def get_points_without_null_curves(self, atol=1e-9):
874: (8)                    nppc = self.n_points_per_curve
875: (8)                    points = self.points
876: (8)                    distinct_curves = reduce(
877: (12)                      op.or_,
878: (12)                      [
879: (16)                        (abs(points[i::nppc] - points[0::nppc]) > atol).any(1)
880: (16)                        for i in range(1, nppc)
881: (12)                      ],
882: (8)                    )
883: (8)                    return points[distinct_curves.repeat(nppc)]
884: (4)                def get_arc_length(self, sample_points_per_curve: int | None = None) ->
float:
885: (8)                    """Return the approximated length of the whole curve.
886: (8)                    Parameters
887: (8)                    -----
888: (8)                    sample_points_per_curve
889: (12)                     Number of sample points per curve used to approximate the length.
More points result in a better approximation.
890: (8)                    Returns
891: (8)                    -----
892: (8)                    float
893: (12)                     The length of the :class:`OpenGLMobject`.
894: (8)                     """
895: (8)                return np.sum(
896: (12)                  length
897: (12)                  for _, length in self.get_curve_functions_with_lengths(
898: (16)                      sample_points=sample_points_per_curve,
899: (12)                  )
900: (8)                )
901: (4)                def get_area_vector(self):
902: (8)                    if not self.has_points():
903: (12)                        return np.zeros(3)
904: (8)                    nppc = self.n_points_per_curve
905: (8)                    points = self.points
906: (8)                    p0 = points[0::nppc]
907: (8)                    p1 = points[nppc - 1 :: nppc]
908: (8)                    return 0.5 * np.array(
909: (12)                      [
910: (16)                        sum(
911: (20)                          (p0[:, 1] + p1[:, 1]) * (p1[:, 2] - p0[:, 2]),
912: (16)                          # Add up (y1 + y2)*(z2 - z1)
913: (16)                          sum(
914: (20)                            (p0[:, 2] + p1[:, 2]) * (p1[:, 0] - p0[:, 0]),
915: (16)                            # Add up (z1 + z2)*(x2 - x1)
916: (16)                            sum(
917: (20)                              (p0[:, 0] + p1[:, 0]) * (p1[:, 1] - p0[:, 1]),
918: (16)                              # Add up (x1 + x2)*(y2 - y1)
919: (12)                            ),
920: (8)                        )
921: (4)                    def get_direction(self):
922: (8)                        """Uses :func:`~.space_ops.shoelace_direction` to calculate the
direction.

```

```

923: (8)             The direction of points determines in which direction the
924: (8)             object is drawn, clockwise or counterclockwise.
925: (8)             Examples
926: (8)
927: (8)             -----
928: (12)            The default direction of a :class:`~.Circle` is counterclockwise::
929: (12)            >>> from manim import Circle
930: (12)            >>> Circle().get_direction()
931: (8)            'CCW'
932: (8)            Returns
933: (8)
934: (12)            -----
935: (8)            :class:`str`
936: (8)            Either ``"CW"`` or ``"CCW"``.
937: (4)             """
938: (8)             return shoelace_direction(self.get_start_anchors())
939: (12)            def get_unit_normal(self, recompute=False):
940: (8)             if not recompute:
941: (12)                 return self.unit_normal[0]
942: (8)             if len(self.points) < 3:
943: (12)                 return OUT
944: (8)             area_vect = self.get_area_vector()
945: (12)             area = np.linalg.norm(area_vect)
946: (8)             if area > 0:
947: (12)                 return area_vect / area
948: (12)             else:
949: (16)                 points = self.points
950: (16)                 return get_unit_normal(
951: (12)                     points[1] - points[0],
952: (4)                     points[2] - points[1],
953: (8)                 )
954: (12)            def refresh_unit_normal(self):
955: (8)             for mob in self.get_family():
956: (12)                 mob.unit_normal[:] = mob.get_unit_normal(recompute=True)
957: (8)             return self
958: (12)            def align_points(self, vmobject):
959: (8)             if self.get_num_points() == len(vmobject.points):
960: (12)                 return
961: (16)             for mob in self, vmobject:
962: (12)                 if not mob.has_points():
963: (16)                     mob.start_new_path(mob.get_center())
964: (8)                 if mob.has_new_path_started():
965: (8)                     mob.add_line_to(mob.points[0])
966: (8)             subpaths1 = self.get_subpaths()
967: (8)             subpaths2 = vmobject.get_subpaths()
968: (8)             n_subpaths = max(len(subpaths1), len(subpaths2))
969: (8)             new_subpaths1 = []
970: (8)             new_subpaths2 = []
971: (12)             nppc = self.n_points_per_curve
972: (16)             def get_nth_subpath(path_list, n):
973: (12)                 if n >= len(path_list):
974: (12)                     return [path_list[-1][-1]] * nppc
975: (16)                 path = path_list[n]
976: (20)                 while len(path) > nppc:
977: (16)                     if self.consider_points_equals(path[-nppc:], path[-nppc - 1]):
978: (20)                         path = path[:-nppc]
979: (12)                     else:
980: (8)                         break
981: (8)                 return path
982: (12)             for n in range(n_subpaths):
983: (12)                 sp1 = get_nth_subpath(subpaths1, n)
984: (12)                 sp2 = get_nth_subpath(subpaths2, n)
985: (12)                 diff1 = max(0, (len(sp2) - len(sp1)) // nppc)
986: (12)                 diff2 = max(0, (len(sp1) - len(sp2)) // nppc)
987: (12)                 sp1 = self.insert_n_curves_to_point_list(diff1, sp1)
988: (12)                 sp2 = self.insert_n_curves_to_point_list(diff2, sp2)
989: (8)                 new_subpaths1.append(sp1)
990: (8)                 new_subpaths2.append(sp2)
991: (8)             self.set_points(np.vstack(new_subpaths1))

```

vmobject.set\_points(np.vstack(new\_subpaths2))

return self

```

992: (4)             def insert_n_curves(self, n: int, recurse=True) -> OpenGLMobject:
993: (8)                 """Inserts n curves to the bezier curves of the vmobject.
994: (8)                 Parameters
995: (8)                 -----
996: (8)                 n
997: (12)                     Number of curves to insert.
998: (8)                 Returns
999: (8)                 -----
1000: (8)                 OpenGLMobject
1001: (12)                     for chaining.
1002: (8)                 """
1003: (8)             for mob in self.get_family(recurse):
1004: (12)                 if mob.get_num_curves() > 0:
1005: (16)                     new_points = mob.insert_n_curves_to_point_list(n, mob.points)
1006: (16)                     if mob.has_new_path_started():
1007: (20)                         new_points = np.vstack([new_points, mob.get_last_point()])
1008: (16)                     mob.set_points(new_points)
1009: (8)                 return self
1010: (4)             def insert_n_curves_to_point_list(self, n: int, points: np.ndarray) ->
np.ndarray:
1011: (8)                 """Given an array of k points defining a bezier curves
1012: (9)                     (anchors and handles), returns points defining exactly
1013: (8)                     k + n bezier curves.
1014: (8)                 Parameters
1015: (8)                 -----
1016: (8)                 n
1017: (12)                     Number of desired curves.
1018: (8)                 points
1019: (12)                     Starting points.
1020: (8)                 Returns
1021: (8)                 -----
1022: (8)                 np.ndarray
1023: (12)                     Points generated.
1024: (8)                 """
1025: (8)             nppc = self.n_points_per_curve
1026: (8)             if len(points) == 1:
1027: (12)                 return np.repeat(points, nppc * n, 0)
1028: (8)             bezier_groups = self.get_bezier_tuples_from_points(points)
1029: (8)             norms = np.array([np.linalg.norm(bg[nppc - 1] - bg[0]) for bg in
bezier_groups])
1030: (8)             total_norm = sum(norms)
1031: (8)             if total_norm < 1e-6:
1032: (12)                 ipc = [n] + [0] * (len(bezier_groups) - 1)
1033: (8)             else:
1034: (12)                 ipc = np.round(n * norms / sum(norms)).astype(int)
1035: (8)             diff = n - sum(ipc)
1036: (8)             for _ in range(diff):
1037: (12)                 ipc[np.argmin(ipc)] += 1
1038: (8)             for _ in range(-diff):
1039: (12)                 ipc[np.argmax(ipc)] -= 1
1040: (8)             new_length = sum(x + 1 for x in ipc)
1041: (8)             new_points = np.empty((new_length, nppc, 3))
1042: (8)             i = 0
1043: (8)             for group, n_inserts in zip(bezier_groups, ipc):
1044: (12)                 alphas = np.linspace(0, 1, n_inserts + 2)
1045: (12)                 for a1, a2 in zip(alphas, alphas[1:]):
1046: (16)                     new_points[i] = partial_quadratic_bezier_points(group, a1, a2)
1047: (16)                     i = i + 1
1048: (8)             return np.vstack(new_points)
1049: (4)             def interpolate(self, mobject1, mobject2, alpha, *args, **kwargs):
1050: (8)                 super().interpolate(mobject1, mobject2, alpha, *args, **kwargs)
1051: (8)                 if config["use_projection_fill_shaders"]:
1052: (12)                     self.refresh_triangulation()
1053: (8)                 else:
1054: (12)                     if self.has_fill():
1055: (16)                         tri1 = mobject1.get_triangulation()
1056: (16)                         tri2 = mobject2.get_triangulation()
1057: (16)                         if len(tri1) != len(tri2) or not np.all(tri1 == tri2):
1058: (20)                             self.refresh_triangulation()

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1059: (8)             return self
1060: (4)             def pointwise_become_partial(
1061: (8)                 self, vmobject: OpenGLVMOBJECT, a: float, b: float, remap: bool = True
1062: (4)             ) -> OpenGLVMOBJECT:
1063: (8)                 """Given two bounds a and b, transforms the points of the self
vmobject into the points of the vmobject
1064: (8)             passed as parameter with respect to the bounds. Points here stand for
control points of the bezier curves (anchors and handles)
1065: (8)             Parameters
1066: (8)             -----
1067: (8)             vmobject
1068: (12)                 The vmobject that will serve as a model.
1069: (8)             a
1070: (12)                 upper-bound.
1071: (8)             b
1072: (12)                 lower-bound
1073: (8)             remap
1074: (12)                 if the point amount should be kept the same (True)
1075: (12)                 This option should be manually set to False if keeping the number
of points is not needed
1076: (8)             """
1077: (8)             assert isinstance(vmobject, OpenGLVMOBJECT)
1078: (8)             if a <= 0 and b >= 1:
1079: (12)                 self.set_points(vmobject.points)
1080: (12)                 return self
1081: (8)             bezier_triplets = vmobject.get_bezier_tuples()
1082: (8)             num_quadratics = len(bezier_triplets)
1083: (8)             lower_index, lower_residue = integer_interpolate(0, num_quadratics, a)
1084: (8)             upper_index, upper_residue = integer_interpolate(0, num_quadratics, b)
1085: (8)             self.clear_points()
1086: (8)             if num_quadratics == 0:
1087: (12)                 return self
1088: (8)             if lower_index == upper_index:
1089: (12)                 self.append_points(
1090: (16)                     partial_quadratic_bezier_points(
1091: (20)                         bezier_triplets[lower_index],
1092: (20)                         lower_residue,
1093: (20)                         upper_residue,
1094: (16)                     ),
1095: (12)                 )
1096: (8)             else:
1097: (12)                 self.append_points(
1098: (16)                     partial_quadratic_bezier_points(
1099: (20)                         bezier_triplets[lower_index], lower_residue, 1
1100: (16)                     ),
1101: (12)                 )
1102: (12)                 inner_points = bezier_triplets[lower_index + 1 : upper_index]
1103: (12)                 if len(inner_points) > 0:
1104: (16)                     if remap:
1105: (20)                         new_triplets = quadratic_bezier_remap(
1106: (24)                             inner_points, num_quadratics - 2
1107: (20)                         )
1108: (16)                     else:
1109: (20)                         new_triplets = bezier_triplets
1110: (16)                         self.append_points(np.asarray(new_triplets).reshape(-1, 3))
1111: (12)                     self.append_points(
1112: (16)                         partial_quadratic_bezier_points(
1113: (20)                             bezier_triplets[upper_index], 0, upper_residue
1114: (16)                         ),
1115: (12)                     )
1116: (8)                 return self
1117: (4)             def get_subcurve(self, a: float, b: float) -> OpenGLVMOBJECT:
1118: (8)                 """Returns the subcurve of the OpenGLVMOBJECT between the interval [a,
b].
1119: (8)                 The curve is a OpenGLVMOBJECT itself.
1120: (8)                 Parameters
1121: (8)                 -----
1122: (8)                 a
1123: (12)                     The lower bound.

```

```

1124: (8)                                b
1125: (12)                               The upper bound.
1126: (8)                                Returns
1127: (8)                               -----
1128: (8)                                OpenGLVMOBJECT
1129: (12)                               The subcurve between of [a, b]
1130: (8)                               """
1131: (8)                                vmob = self.copy()
1132: (8)                                vmob.pointwise_become_partial(self, a, b)
1133: (8)                                return vmob
1134: (4)                                def refresh_triangulation(self):
1135: (8)                                for mob in self.get_family():
1136: (12)                                  mob.needs_new_triangulation = True
1137: (8)                                return self
1138: (4)                                def get_triangulation(self, normal_vector=None):
1139: (8)                                if normal_vector is None:
1140: (12)                                  normal_vector = self.get_unit_normal()
1141: (8)                                if not self.needs_new_triangulation:
1142: (12)                                  return self.triangulation
1143: (8)                                points = self.points
1144: (8)                                if len(points) <= 1:
1145: (12)                                  self.triangulation = np.zeros(0, dtype="i4")
1146: (12)                                  self.needs_new_triangulation = False
1147: (12)                                  return self.triangulation
1148: (8)                                if not np.isclose(normal_vector, OUT).all():
1149: (12)                                  points = np.dot(points, z_to_vector(normal_vector))
1150: (8)                                indices = np.arange(len(points), dtype=int)
1151: (8)                                b0s = points[0::3]
1152: (8)                                b1s = points[1::3]
1153: (8)                                b2s = points[2::3]
1154: (8)                                v01s = b1s - b0s
1155: (8)                                v12s = b2s - b1s
1156: (8)                                crosses = cross2d(v01s, v12s)
1157: (8)                                convexities = np.sign(crosses)
1158: (8)                                atol = self.tolerance_for_point_equality
1159: (8)                                end_of_loop = np.zeros(len(b0s), dtype=bool)
1160: (8)                                end_of_loop[:-1] = (np.abs(b2s[:-1] - b0s[1:]) > atol).any(1)
1161: (8)                                end_of_loop[-1] = True
1162: (8)                                concave_parts = convexities < 0
1163: (8)                                inner_vert_indices = np.hstack(
1164: (12)                                  [
1165: (16)                                    indices[0::3],
1166: (16)                                    indices[1::3][concave_parts],
1167: (16)                                    indices[2::3][end_of_loop],
1168: (12)                                  ],
1169: (8)                                )
1170: (8)                                inner_vert_indices.sort()
1171: (8)                                rings = np.arange(1, len(inner_vert_indices) + 1)[inner_vert_indices %
3 == 2]
1172: (8)                                inner_verts = points[inner_vert_indices]
1173: (8)                                inner_tri_indices = inner_vert_indices[
1174: (12)                                  earclip_triangulation(inner_verts, rings)
1175: (8)                                ]
1176: (8)                                tri_indices = np.hstack([indices, inner_tri_indices])
1177: (8)                                self.triangulation = tri_indices
1178: (8)                                self.needs_new_triangulation = False
1179: (8)                                return tri_indices
1180: (4)                                @triggers_refreshed_triangulation
1181: (4)                                def set_points(self, points):
1182: (8)                                  super().set_points(points)
1183: (8)                                  return self
1184: (4)                                @triggers_refreshed_triangulation
1185: (4)                                def set_data(self, data):
1186: (8)                                  super().set_data(data)
1187: (8)                                  return self
1188: (4)                                @triggers_refreshed_triangulation
1189: (4)                                def apply_function(self, function, make_smooth=False, **kwargs):
1190: (8)                                  super().apply_function(function, **kwargs)
1191: (8)                                  if self.make_smooth_after_applying_functions or make_smooth:

```

```

1192: (12)                     self.make_approximately_smooth()
1193: (8)                      return self
1194: (4) @triggers_refreshed_triangulation
1195: (4) def apply_points_function(self, *args, **kwargs):
1196: (8)     super().apply_points_function(*args, **kwargs)
1197: (8)     return self
1198: (4) @triggers_refreshed_triangulation
1199: (4) def flip(self, *args, **kwargs):
1200: (8)     super().flip(*args, **kwargs)
1201: (8)     return self
1202: (4) def init_shader_data(self):
1203: (8)     self.fill_data = np.zeros(0, dtype=self.fill_dtype)
1204: (8)     self.stroke_data = np.zeros(0, dtype=self.stroke_dtype)
1205: (8)     self.fill_shader_wrapper = ShaderWrapper(
1206: (12)         vert_data=self.fill_data,
1207: (12)         vert_indices=np.zeros(0, dtype="i4"),
1208: (12)         shader_folder=self.fill_shader_folder,
1209: (12)         render_primitive=self.render_primitive,
1210: (8)     )
1211: (8)     self.stroke_shader_wrapper = ShaderWrapper(
1212: (12)         vert_data=self.stroke_data,
1213: (12)         shader_folder=self.stroke_shader_folder,
1214: (12)         render_primitive=self.render_primitive,
1215: (8)     )
1216: (4) def refresh_shader_wrapper_id(self):
1217: (8)     for wrapper in [self.fill_shader_wrapper, self.stroke_shader_wrapper]:
1218: (12)         wrapper.refresh_id()
1219: (8)     return self
1220: (4) def get_fill_shader_wrapper(self):
1221: (8)     self.update_fill_shader_wrapper()
1222: (8)     return self.fill_shader_wrapper
1223: (4) def update_fill_shader_wrapper(self):
1224: (8)     self.fill_shader_wrapper.vert_data = self.get_fill_shader_data()
1225: (8)     self.fill_shader_wrapper.vert_indices = self.get_triangulation()
1226: (8)     self.fill_shader_wrapper.uniforms = self.get_fill_uniforms()
1227: (8)     self.fill_shader_wrapper.depth_test = self.depth_test
1228: (4) def get_stroke_shader_wrapper(self):
1229: (8)     self.update_stroke_shader_wrapper()
1230: (8)     return self.stroke_shader_wrapper
1231: (4) def update_stroke_shader_wrapper(self):
1232: (8)     self.stroke_shader_wrapper.vert_data = self.get_stroke_shader_data()
1233: (8)     self.stroke_shader_wrapper.uniforms = self.get_stroke_uniforms()
1234: (8)     self.stroke_shader_wrapper.depth_test = self.depth_test
1235: (4) def get_shader_wrapper_list(self):
1236: (8)     fill_shader_wrappers = []
1237: (8)     stroke_shader_wrappers = []
1238: (8)     back_stroke_shader_wrappers = []
1239: (8)     for submob in self.family_members_with_points():
1240: (12)         if submob.has_fill() and not
config["use_projection_fill_shaders"]:
1241: (16)             fill_shader_wrappers.append(submob.get_fill_shader_wrapper())
1242: (12)             if submob.has_stroke() and not
config["use_projection_stroke_shaders"]:
1243: (16)                 ssw = submob.get_stroke_shader_wrapper()
1244: (16)                 if submob.draw_stroke_behind_fill:
1245: (20)                     back_stroke_shader_wrappers.append(ssw)
1246: (16)                 else:
1247: (20)                     stroke_shader_wrappers.append(ssw)
1248: (8)             wrapper_lists = [
1249: (12)                 back_stroke_shader_wrappers,
1250: (12)                 fill_shader_wrappers,
1251: (12)                 stroke_shader_wrappers,
1252: (8)             ]
1253: (8)             result = []
1254: (8)             for wlist in wrapper_lists:
1255: (12)                 if wlist:
1256: (16)                     wrapper = wlist[0]
1257: (16)                     wrapper.combine_with(*wlist[1:])
1258: (16)                     result.append(wrapper)

```

```

1259: (8)             return result
1260: (4)         def get_stroke_uniforms(self):
1261: (8)             result = dict(super().get_shader_uniforms())
1262: (8)             result["joint_type"] = self.joint_type.value
1263: (8)             result["flat_stroke"] = float(self.flat_stroke)
1264: (8)             return result
1265: (4)         def get_fill_uniforms(self):
1266: (8)             return {
1267: (12)                 "is_fixed_in_frame": float(self.is_fixed_in_frame),
1268: (12)                 "is_fixed_orientation": float(self.is_fixed_orientation),
1269: (12)                 "fixed_orientation_center": self.fixed_orientation_center,
1270: (12)                 "gloss": self.gloss,
1271: (12)                 "shadow": self.shadow,
1272: (8)             }
1273: (4)         def get_stroke_shader_data(self):
1274: (8)             points = self.points
1275: (8)             if len(self.stroke_data) != len(points):
1276: (12)                 self.stroke_data = np.zeros(len(points),
dtype=OpenGLMobject.stroke_dtype)
1277: (8)             if "points" not in self.locked_data_keys:
1278: (12)                 nppc = self.n_points_per_curve
1279: (12)                 self.stroke_data["point"] = points
1280: (12)                 self.stroke_data["prev_point"][:nppc] = points[-nppc:]
1281: (12)                 self.stroke_data["prev_point"][nppc:] = points[:-nppc]
1282: (12)                 self.stroke_data["next_point"][:-nppc] = points[nppc:]
1283: (12)                 self.stroke_data["next_point"][-nppc:] = points[:nppc]
1284: (8)             self.read_data_to_shader(self.stroke_data, "color", "stroke_rgba")
1285: (8)             self.read_data_to_shader(self.stroke_data, "stroke_width",
"stroke_width")
1286: (8)             self.read_data_to_shader(self.stroke_data, "unit_normal",
"unit_normal")
1287: (8)             return self.stroke_data
1288: (4)         def get_fill_shader_data(self):
1289: (8)             points = self.points
1290: (8)             if len(self.fill_data) != len(points):
1291: (12)                 self.fill_data = np.zeros(len(points),
dtype=OpenGLMobject.fill_dtype)
1292: (12)                 self.fill_data["vert_index"][:, 0] = range(len(points))
1293: (8)                 self.read_data_to_shader(self.fill_data, "point", "points")
1294: (8)                 self.read_data_to_shader(self.fill_data, "color", "fill_rgba")
1295: (8)                 self.read_data_to_shader(self.fill_data, "unit_normal", "unit_normal")
1296: (8)             return self.fill_data
1297: (4)         def refresh_shader_data(self):
1298: (8)             self.get_fill_shader_data()
1299: (8)             self.get_stroke_shader_data()
1300: (4)         def get_fill_shader_vert_indices(self):
1301: (8)             return self.get_triangulation()
1302: (0)     class OpenGLGroup(OpenGLMobject):
1303: (4)         """A group of vectorized mobjects.
1304: (4)         This can be used to group multiple :class:`~.OpenGLMobject` instances
together
1305: (4)             in order to scale, move, ... them together.
1306: (4)             Examples
1307: (4)             -----
1308: (4)             To add :class:`~.OpenGLMobject`'s to a :class:`~.OpenGLGroup`, you can
1309: (4)             :meth:`~.OpenGLGroup.add` method, or use the `+` and `+=` operators.
1310: (4)             can subtract elements of a OpenGLGroup via :meth:`~.OpenGLGroup.remove``,
1311: (4)             ``-`` and ``-=`` operators:
1312: (4)             .. doctest::
1313: (8)                 >>> from manim import config
1314: (8)                 >>> original_renderer = config.renderer
1315: (8)                 >>> config.renderer = "opengl"
1316: (8)                 >>> from manim import Triangle, Square
1317: (8)                 >>> from manim.opengl import OpenGLGroup
1318: (8)                 >>> config.renderer
1319: (8)                 <RendererType.OPENGL: 'opengl'>

```

```

1320: (8)             >>> vg = OpenGLGroup()
1321: (8)             >>> triangle, square = Triangle(), Square()
1322: (8)             >>> vg.add(triangle)
1323: (8)             OpenGLGroup(Triangle)
1324: (8)             >>> vg + square # a new OpenGLGroup is constructed
1325: (8)             OpenGLGroup(Triangle, Square)
1326: (8)             >>> vg # not modified
1327: (8)             OpenGLGroup(Triangle)
1328: (8)             >>> vg += square # modifies vg
1329: (8)             >>> vg
1330: (8)             OpenGLGroup(Triangle, Square)
1331: (8)             >>> vg.remove(triangle)
1332: (8)             OpenGLGroup(Square)
1333: (8)             >>> vg - square # a new OpenGLGroup is constructed
1334: (8)             OpenGLGroup()
1335: (8)             >>> vg # not modified
1336: (8)             OpenGLGroup(Square)
1337: (8)             >>> vg -= square # modifies vg
1338: (8)             >>> vg
1339: (8)             OpenGLGroup()
1340: (8)             >>> config.renderer = original_renderer
1341: (4) .. manim:: ArcShapeIris
1342: (8)             :save_last_frame:
1343: (8)             class ArcShapeIris(Scene):
1344: (12)                 def construct(self):
1345: (16)                     colors = [DARK_BROWN, BLUE_E, BLUE_D, BLUE_A, TEAL_B, GREEN_B,
1346: (16)                         YELLOW_E]
1347: (16)                     radius = [1 + rad * 0.1 for rad in range(len(colors))]
1348: (16)                     circles_group = OpenGLGroup()
1349: (36)                     circles_group.add(*[Circle(radius=rad, stroke_width=10,
1350: (16)                                     for rad, col in zip(radius, colors))])
1351: (4)                     self.add(circles_group)
1352: (4) """
1353: (8)             def __init__(self, *vmobjects, **kwargs):
1354: (12)                 if not all(isinstance(m, OpenGLMobject) for m in vmobjects):
1355: (8)                     raise Exception("All submobjects must be of type OpenGLMobject")
1356: (8)                 super().__init__(**kwargs)
1357: (4)                 self.add(*vmobjects)
1358: (8)             def __repr__(self):
1359: (12)                 return (
1360: (12)                     self.__class__.__name__
1361: (12)                     + "("
1362: (12)                     + ", ".join(str(mob) for mob in self.submobjects)
1363: (8)                     + ")"
1364: (4)             def __str__(self):
1365: (8)                 return (
1366: (12)                     f"{self.__class__.__name__} of {len(self.submobjects)} "
1367: (12)                     f"subobject{'s' if len(self.submobjects) > 0 else ''}"
1368: (8)                 )
1369: (4)             def add(self, *vmobjects: OpenGLMobject):
1370: (8)                 """Checks if all passed elements are an instance of OpenGLMobject and
then add them to submobjects
1371: (8)             Parameters
1372: (8)             -----
1373: (8)             vmobjects
1374: (12)                 List of OpenGLMobject to add
1375: (8)             Returns
1376: (8)             -----
1377: (8)             :class:`OpenGLGroup`
1378: (8)             Raises
1379: (8)             -----
1380: (8)             TypeError
1381: (12)                 If one element of the list is not an instance of OpenGLMobject
1382: (8)             Examples
1383: (8)             -----
1384: (8)             .. manim:: AddToOpenGLGroup
1385: (12)                 class AddToOpenGLGroup(Scene):
```

```

1386: (16)             def construct(self):
1387: (20)                 circle_red = Circle(color=RED)
1388: (20)                 circle_green = Circle(color=GREEN)
1389: (20)                 circle_blue = Circle(color=BLUE)
1390: (20)                 circle_red.shift(LEFT)
1391: (20)                 circle_blue.shift(RIGHT)
1392: (20)                 gr = OpenGLGroup(circle_red, circle_green)
1393: (20)                 gr2 = OpenGLGroup(circle_blue) # Constructor uses add
directly
1394: (20)                     self.add(gr,gr2)
1395: (20)                     self.wait()
1396: (20)                     gr += gr2 # Add group to another
1397: (20)                     self.play(
1398: (24)                         gr.animate.shift(DOWN),
1399: (20)                     )
1400: (20)                     gr -= gr2 # Remove group
1401: (20)                     self.play( # Animate groups separately
1402: (24)                         gr.animate.shift(LEFT),
1403: (24)                         gr2.animate.shift(UP),
1404: (20)                     )
1405: (20)                     self.play( #Animate groups without modification
1406: (24)                         (gr+gr2).animate.shift(RIGHT)
1407: (20)                     )
1408: (20)                     self.play( # Animate group without component
1409: (24)                         (gr-circle_red).animate.shift(RIGHT)
1410: (20)                     )
1411: (8)                     """
1412: (8)             if not all(isinstance(m, OpenGLMobject) for m in vmobjects):
1413: (12)                 raise TypeError("All submobjects must be of type OpenGLMobject")
1414: (8)             return super().add(*vmobjects)
1415: (4)             def __add__(self, vmobject):
1416: (8)                 return OpenGLGroup(*self.submobjects, vmobject)
1417: (4)             def __iadd__(self, vmobject):
1418: (8)                 return self.add(vmobject)
1419: (4)             def __sub__(self, vmobject):
1420: (8)                 copy = OpenGLGroup(*self.submobjects)
1421: (8)                 copy.remove(vmobject)
1422: (8)                 return copy
1423: (4)             def __isub__(self, vmobject):
1424: (8)                 return self.remove(vmobject)
1425: (4)             def __setitem__(self, key: int, value: OpenGLMobject | Sequence[OpenGLMobject]):
Sequence[OpenGLMobject])::
1426: (8)                 """Override the [] operator for item assignment.
1427: (8)                 Parameters
1428: (8)                 -----
1429: (8)                 key
1430: (12)                     The index of the submobject to be assigned
1431: (8)                 value
1432: (12)                     The vmobject value to assign to the key
1433: (8)                 Returns
1434: (8)                 -----
1435: (8)                 None
1436: (8)                 Tests
1437: (8)                 -----
1438: (8)                 .. doctest::
1439: (12)                     >>> from manim import config
1440: (12)                     >>> original_renderer = config.renderer
1441: (12)                     >>> config.renderer = "opengl"
1442: (12)                     >>> vgroup = OpenGLGroup(OpenGLMobject())
1443: (12)                     >>> new_obj = OpenGLMobject()
1444: (12)                     >>> vgroup[0] = new_obj
1445: (12)                     >>> config.renderer = original_renderer
1446: (8)                     """
1447: (8)                 if not all(isinstance(m, OpenGLMobject) for m in value):
1448: (12)                     raise TypeError("All submobjects must be of type OpenGLMobject")
1449: (8)                     self.submobjects[key] = value
1450: (0)             class OpenGLVectorizedPoint(OpenGLPoint, OpenGLMobject):
1451: (4)                 def __init__(
1452: (8)                     self,

```

```

1453: (8)             location=ORIGIN,
1454: (8)             color=BLACK,
1455: (8)             fill_opacity=0,
1456: (8)             stroke_width=0,
1457: (8)             artificial_width=0.01,
1458: (8)             artificial_height=0.01,
1459: (8)             **kwargs,
1460: (4)             ):
1461: (8)             self.artificial_width = artificial_width
1462: (8)             self.artificial_height = artificial_height
1463: (8)             super().__init__(
1464: (12)                 color=color, fill_opacity=fill_opacity, stroke_width=stroke_width,
**kwargs
1465: (8)             )
1466: (8)             self.set_points(np.array([location]))
1467: (0)             class OpenGLCurvesAsSubmobjects(OpenGLGroup):
1468: (4)                 """Convert a curve's elements to submobjects.
1469: (4)                 Examples
1470: (4)                 -----
1471: (4)                 .. manim:: LineGradientExample
1472: (8)                   :save_last_frame:
1473: (8)                   class LineGradientExample(Scene):
1474: (12)                     def construct(self):
1475: (16)                         curve = ParametricFunction(lambda t: [t, np.sin(t), 0],
t_range=[-PI, PI, 0.01], stroke_width=10)
1476: (16)                         new_curve = CurvesAsSubmobjects(curve)
1477: (16)                         new_curve.set_color_by_gradient(BLUE, RED)
1478: (16)                         self.add(new_curve.shift(UP), curve)
1479: (4)             """
1480: (4)             def __init__(self, vmobject, **kwargs):
1481: (8)                 super().__init__(**kwargs)
1482: (8)                 for tup in vmobject.get_bezier_tuples():
1483: (12)                     part = OpenGLMobject()
1484: (12)                     part.set_points(tup)
1485: (12)                     part.match_style(vmobject)
1486: (12)                     self.add(part)
1487: (0)             class OpenGLDashedVMobject(OpenGLMobject):
1488: (4)                 """A :class:`OpenGLMobject` composed of dashes instead of lines.
1489: (4)                 Examples
1490: (4)                 -----
1491: (4)                 .. manim:: DashedVMobjectExample
1492: (8)                   :save_last_frame:
1493: (8)                   class DashedVMobjectExample(Scene):
1494: (12)                     def construct(self):
1495: (16)                         r = 0.5
1496: (16)                         top_row = OpenGLGroup() # Increasing num_dashes
1497: (16)                         for dashes in range(2, 12):
1498: (20)                             circ = DashedVMobject(Circle(radius=r, color=WHITE),
num_dashes=dashes)
1499: (20)
1500: (16)
1501: (16)
1502: (20)
1503: (24)
1504: (20)
1505: (20)
1506: (16)
1507: (16)
1508: (16)
1509: (16)
1510: (16)
1511: (16)
1512: (16)
bottom_row).arrange(DOWN, buff=1)
1513: (16)                     self.add(everything)
1514: (4)             """
1515: (4)             def __init__(
1516: (8)                 self,
1517: (8)                 vmobject: OpenGLMobject,

```

```

1518: (8)             num_dashes: int = 15,
1519: (8)             dashed_ratio: float = 0.5,
1520: (8)             color: ParsableManimColor = WHITE,
1521: (8)             **kwargs,
1522: (4)             ):
1523: (8)             self.dashed_ratio = dashed_ratio
1524: (8)             self.num_dashes = num_dashes
1525: (8)             super().__init__(color=color, **kwargs)
1526: (8)             r = self.dashed_ratio
1527: (8)             n = self.num_dashes
1528: (8)             if num_dashes > 0:
1529: (12)                 dash_len = r / n
1530: (12)                 if vmobject.is_closed():
1531: (16)                     void_len = (1 - r) / n
1532: (12)                 else:
1533: (16)                     void_len = (1 - r) / (n - 1)
1534: (12)                 self.add(
1535: (16)                     *
1536: (20)                         vmobject.get_subcurve(
1537: (24)                             i * (dash_len + void_len),
1538: (24)                             i * (dash_len + void_len) + dash_len,
1539: (20)                         )
1540: (20)                         for i in range(n)
1541: (16)                     )
1542: (12)                 )
1543: (8)             self.match_style(vmobject, recurse=False)
-----
```

## File 96 - opengl\_point\_cloud\_mobject.py:

```

1: (0)         from __future__ import annotations
2: (0)         __all__ = ["OpenGLMobject", "OpenGLGroup", "OpenGLPoint"]
3: (0)         import moderngl
4: (0)         import numpy as np
5: (0)         from manim.constants import *
6: (0)         from manim.mobject.opengl.opengl_mobject import OpenGLMobject
7: (0)         from manim.utils.bezier import interpolate
8: (0)         from manim.utils.color import BLACK, WHITE, YELLOW, color_gradient,
color_to_rgba
9: (0)         from manim.utils.config_ops import _Uniforms
10: (0)        from manim.utils.iterables import resize_with_interpolation
11: (0)        __all__ = ["OpenGLMobject", "OpenGLGroup", "OpenGLPoint"]
12: (0)        class OpenGLMobject(OpenGLMobject):
13: (4)            shader_folder = "true_dot"
14: (4)            OPENGL_POINT_RADIUS_SCALE_FACTOR = 0.01
15: (4)            shader_dtype = [
16: (8)                ("point", np.float32, (3,)),
17: (8)                ("color", np.float32, (4,)),
18: (4)            ]
19: (4)            point_radius = _Uniforms()
20: (4)            def __init__(
21: (8)                self, stroke_width=2.0, color=YELLOW,
render_primitive=moderngl.POINTS, **kwargs
22: (4)            ):
23: (8)                self.stroke_width = stroke_width
24: (8)                super().__init__(color=color, render_primitive=render_primitive,
**kwargs)
25: (8)                self.point_radius = (
26: (12)                    self.stroke_width *
OpenGLMobject.OPENGL_POINT_RADIUS_SCALE_FACTOR
27: (8)                )
28: (4)                def reset_points(self):
29: (8)                    self.rgbas = np.zeros((1, 4))
30: (8)                    self.points = np.zeros((0, 3))
31: (8)                    return self
32: (4)                def get_array_attrs(self):
33: (8)                    return ["points", "rgbas"]
34: (4)                def add_points(self, points, rgbas=None, color=None, opacity=None):
```

```

35: (8)                      """Add points.
36: (8)                      Points must be a Nx3 numpy array.
37: (8)                      Rgbas must be a Nx4 numpy array if it is not None.
38: (8)
39: (8)                      if rgbas is None and color is None:
40: (12)                          color = YELLOW
41: (8)                      self.append_points(points)
42: (8)                      if color is not None:
43: (12)                          if opacity is None:
44: (16)                              opacity = self.rgbas[-1, 3]
45: (12)                          new_rgbas = np.repeat([color_to_rgba(color, opacity)],

len(points), axis=0)
46: (8)
47: (12)
48: (8)
49: (12)                      raise ValueError("points and rgbs must have same length")
50: (8)                      self.rgbas = np.append(self.rgbas, new_rgbas, axis=0)
51: (8)                      return self
52: (4)                      def thin_out(self, factor=5):
53: (8)
54: (8)                          """Removes all but every nth point for n = factor
55: (8)
56: (8)                          for mob in self.family_members_with_points():
57: (12)                              num_points = mob.get_num_points()
58: (12)                              def thin_func():
59: (16)                                  return np.arange(0, num_points, factor)
60: (12)                              if len(mob.points) == len(mob.rgbas):
61: (16)                                  mob.set_rgba_array_direct(mob.rgbas[thin_func()])
62: (12)                                  mob.set_points(mob.points[thin_func()])
63: (8)                          return self
64: (4)                      def set_color_by_gradient(self, *colors):
65: (8)                          self.rgbas = np.array(
66: (12)                              list(map(color_to_rgba, color_gradient(*colors,
self.get_num_points()))),
67: (8)
68: (8)                          )
69: (4)                          return self
70: (8)                      def set_colors_by_radial_gradient(
71: (8)                          self,
72: (8)                          center=None,
73: (8)                          radius=1,
74: (8)                          inner_color=WHITE,
75: (8)                          outer_color=BLACK,
76: (4)                          ):
77: (8)                          start_rgba, end_rgba = list(map(color_to_rgba, [inner_color,
outer_color]))
78: (12)
79: (8)
80: (12)
81: (12)
82: (12)
83: (16)
84: (20)
alphas],
85: (16)
86: (12)
87: (8)
88: (4)
89: (8)
self.get_num_points())
90: (8)
91: (4)
92: (8)
93: (8)
94: (12)
95: (8)
96: (8)
97: (4)
98: (8)
                      self.rgbas[:] = resize_with_interpolation(pmobject.rgbas,
return self
def fade_to(self, color, alpha, family=True):
    rgbs = interpolate(self.rgbas, color_to_rgba(color), alpha)
    for mob in self.subobjects:
        mob.fade_to(color, alpha, family)
    self.set_rgba_array_direct(rgbs)
    return self
def filter_out(self, condition):
    for mob in self.family_members_with_points():

```

```

99: (12)                     to_keep = ~np.apply_along_axis(condition, 1, mob.points)
100: (12)                     for key in mob.data:
101: (16)                         mob.data[key] = mob.data[key][to_keep]
102: (8)                     return self
103: (4)             def sort_points(self, function=lambda p: p[0]):
104: (8)                 """
105: (8)                     function is any map from R^3 to R
106: (8)                 """
107: (8)                 for mob in self.family_members_with_points():
108: (12)                     indices = np.argsort(np.apply_along_axis(function, 1, mob.points))
109: (12)                     for key in mob.data:
110: (16)                         mob.data[key] = mob.data[key][indices]
111: (8)                     return self
112: (4)             def ingest_submobjects(self):
113: (8)                 for key in self.data:
114: (12)                     self.data[key] = np.vstack([sm.data[key] for sm in
self.get_family()])
115: (8)                     return self
116: (4)             def point_from_proportion(self, alpha):
117: (8)                 index = alpha * (self.get_num_points() - 1)
118: (8)                 return self.points[int(index)]
119: (4)             def pointwiseBecomePartial(self, pmobject, a, b):
120: (8)                 lower_index = int(a * pmobject.get_num_points())
121: (8)                 upper_index = int(b * pmobject.get_num_points())
122: (8)                 for key in self.data:
123: (12)                     self.data[key] = pmobject.data[key][lower_index:upper_index]
124: (8)                     return self
125: (4)             def get_shader_data(self):
126: (8)                 shader_data = np.zeros(len(self.points), dtype=self.shader_dtype)
127: (8)                 self.read_data_to_shader(shader_data, "point", "points")
128: (8)                 self.read_data_to_shader(shader_data, "color", "rgbas")
129: (8)                 return shader_data
130: (4)             @staticmethod
131: (4)             def get_mobject_type_class():
132: (8)                 return OpenGLMobject
133: (0)         class OpenGLGroup(OpenGLMobject):
134: (4)             def __init__(self, *pmobs, **kwargs):
135: (8)                 if not all(isinstance(m, OpenGLMobject) for m in pmobs):
136: (12)                     raise Exception("All submobjects must be of type OpenGLMObject")
137: (8)                 super().__init__(**kwargs)
138: (8)                 self.add(*pmobs)
139: (4)             def fade_to(self, color, alpha, family=True):
140: (8)                 if family:
141: (12)                     for mob in self.submobjects:
142: (16)                         mob.fade_to(color, alpha, family)
143: (0)         class OpenGLPoint(OpenGLMobject):
144: (4)             def __init__(self, location=ORIGIN, stroke_width=4.0, **kwargs):
145: (8)                 self.location = location
146: (8)                 super().__init__(stroke_width=stroke_width, **kwargs)
147: (4)             def init_points(self):
148: (8)                 self.points = np.array([self.location], dtype=np.float32)

```

---

## File 97 - vectorized\_mobject.py:

```

1: (0)             """Mobjects that use vector graphics."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "VMobject",
5: (4)                 "VGroup",
6: (4)                 "VDict",
7: (4)                 "VectorizedPoint",
8: (4)                 "CurvesAsSubmobjects",
9: (4)                 "DashedVMobject",
10: (0)                ]
11: (0)                import itertools as it
12: (0)                import sys
13: (0)                from typing import (

```

```

14: (4)             TYPE_CHECKING,
15: (4)             Callable,
16: (4)             Generator,
17: (4)             Hashable,
18: (4)             Iterable,
19: (4)             Literal,
20: (4)             Mapping,
21: (4)             Sequence,
22: (0)
23: (0)         import numpy as np
24: (0)         from PIL.Image import Image
25: (0)         from manim import config
26: (0)         from manim.constants import *
27: (0)         from manim.mobject.mobject import Mobject
28: (0)         from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
29: (0)         from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLVMOBJECT
30: (0)         from manim.mobject.three_d.three_d_utils import (
31: (4)             get_3d_vmob_gradient_start_and_end_points,
32: (0)
33: (0)     )
34: (4)     from manim.utils.bezier import (
35: (4)         bezier,
36: (4)         get_smooth_handle_points,
37: (4)         integer_interpolate,
38: (4)         interpolate,
39: (4)         partial_bezier_points,
40: (0)         proportions_along_bezier_curve_for_point,
41: (0)     )
42: (0)     from manim.utils.color import BLACK, WHITE, ManimColor, ParsableManimColor
43: (0)     from manim.utils.iterables import (
44: (4)         make_even,
45: (4)         resize_array,
46: (4)         stretch_array_to_length,
47: (0)         tuplify,
48: (0)     )
49: (0)     from manim.utils.space_ops import rotate_vector, shoelace_direction
50: (0)     if TYPE_CHECKING:
51: (4)         import numpy.typing as npt
52: (4)         from typing_extensions import Self
53: (4)         from manim.typing import (
54: (8)             BezierPoints,
55: (8)             CubicBezierPoints,
56: (8)             ManimFloat,
57: (8)             MappingFunction,
58: (8)             Point2D,
59: (8)             Point3D,
60: (8)             Point3D_Array,
61: (8)             QuadraticBezierPoints,
62: (8)             RGBA_Array_Float,
63: (8)             Vector3D,
64: (8)             Zeros,
65: (0)         )
66: (4)         __all__ = [
67: (4)             "VMobject",
68: (4)             "VGroup",
69: (4)             "VDict",
70: (4)             "VectorizedPoint",
71: (4)             "CurvesAsSubmobjects",
72: (4)             "VectorizedPoint",
73: (0)             "DashedVMobject",
74: (0)         ]
75: (0)         class VMobject(Mobject):
76: (4)             """A vectorized mobject.
77: (4)             Parameters
78: (4)             -----
79: (8)             background_stroke_color
80: (8)                 The purpose of background stroke is to have something
81: (8)                 that won't overlap fill, e.g. For text against some
82: (8)                 textured background.
83: (4)             sheen_factor

```

```

83: (8)             When a color c is set, there will be a second color
84: (8)             computed based on interpolating c to WHITE by with
85: (8)             sheen_factor, and the display will gradient to this
86: (8)             secondary color in the direction of sheen_direction.
87: (4)             close_new_points
88: (8)                 Indicates that it will not be displayed, but
89: (8)                 that it should count in parent mobject's path
90: (4)             tolerance_for_point_equality
91: (8)                 This is within a pixel
92: (4)             joint_type
93: (8)                 The line joint type used to connect the curve segments
94: (8)                 of this vectorized mobject. See :class:`.LineJointType`
95: (8)                 for options.
96: (4)             """
97: (4)             sheen_factor = 0.0
98: (4)             def __init__(
99: (8)                 self,
100: (8)                 fill_color: ParsableManimColor | None = None,
101: (8)                 fill_opacity: float = 0.0,
102: (8)                 stroke_color: ParsableManimColor | None = None,
103: (8)                 stroke_opacity: float = 1.0,
104: (8)                 stroke_width: float = DEFAULT_STROKE_WIDTH,
105: (8)                 background_stroke_color: ParsableManimColor | None = BLACK,
106: (8)                 background_stroke_opacity: float = 1.0,
107: (8)                 background_stroke_width: float = 0,
108: (8)                 sheen_factor: float = 0.0,
109: (8)                 joint_type: LineJointType | None = None,
110: (8)                 sheen_direction: Vector3D = UL,
111: (8)                 close_new_points: bool = False,
112: (8)                 pre_function_handle_to_anchor_scale_factor: float = 0.01,
113: (8)                 make_smooth_after_applying_functions: bool = False,
114: (8)                 background_image: Image | str | None = None,
115: (8)                 shade_in_3d: bool = False,
116: (8)                 tolerance_for_point_equality: float = 1e-6,
117: (8)                 n_points_per_cubic_curve: int = 4,
118: (8)                 cap_style: CapStyleType = CapStyleType.AUTO,
119: (8)                 **kwargs,
120: (4)             ):
121: (8)                 self.fill_opacity = fill_opacity
122: (8)                 self.stroke_opacity = stroke_opacity
123: (8)                 self.stroke_width = stroke_width
124: (8)                 if background_stroke_color is not None:
125: (12)                     self.background_stroke_color: ManimColor = ManimColor(
126: (16)                         background_stroke_color
127: (12)                     )
128: (8)                     self.background_stroke_opacity: float = background_stroke_opacity
129: (8)                     self.background_stroke_width: float = background_stroke_width
130: (8)                     self.sheen_factor: float = sheen_factor
131: (8)                     self.joint_type: LineJointType = (
132: (12)                         LineJointType.AUTO if joint_type is None else joint_type
133: (8)                     )
134: (8)                     self.sheen_direction: Vector3D = sheen_direction
135: (8)                     self.close_new_points: bool = close_new_points
136: (8)                     self.pre_function_handle_to_anchor_scale_factor: float = (
137: (12)                         pre_function_handle_to_anchor_scale_factor
138: (8)                     )
139: (8)                     self.make_smooth_after_applying_functions: bool = (
140: (12)                         make_smooth_after_applying_functions
141: (8)                     )
142: (8)                     self.background_image: Image | str | None = background_image
143: (8)                     self.shade_in_3d: bool = shade_in_3d
144: (8)                     self.tolerance_for_point_equality: float =
tolerance_for_point_equality
145: (8)                         self.n_points_per_cubic_curve: int = n_points_per_cubic_curve
146: (8)                         self.cap_style: CapStyleType = cap_style
147: (8)                         super().__init__(**kwargs)
148: (8)                         self.submobjects: list[VMobject]
149: (8)                         if fill_color is not None:
150: (12)                             self.fill_color = ManimColor.parse(fill_color)

```

```

151: (8)             if stroke_color is not None:
152: (12)             self.stroke_color = ManimColor.parse(stroke_color)
153: (4)             @property
154: (4)             def n_points_per_curve(self) -> int:
155: (8)                 return self.n_points_per_cubic_curve
156: (4)             def get_group_class(self) -> type[VGroup]:
157: (8)                 return VGroup
158: (4)             @staticmethod
159: (4)             def get_mobject_type_class() -> type[VMobject]:
160: (8)                 return VMobject
161: (4)             def init_colors(self, propagate_colors: bool = True) -> Self:
162: (8)                 self.set_fill(
163: (12)                     color=self.fill_color,
164: (12)                     opacity=self.fill_opacity,
165: (12)                     family=propagate_colors,
166: (8)                 )
167: (8)                 self.set_stroke(
168: (12)                     color=self.stroke_color,
169: (12)                     width=self.stroke_width,
170: (12)                     opacity=self.stroke_opacity,
171: (12)                     family=propagate_colors,
172: (8)                 )
173: (8)                 self.set_background_stroke(
174: (12)                     color=self.background_stroke_color,
175: (12)                     width=self.background_stroke_width,
176: (12)                     opacity=self.background_stroke_opacity,
177: (12)                     family=propagate_colors,
178: (8)                 )
179: (8)                 self.set_sheen(
180: (12)                     factor=self.sheen_factor,
181: (12)                     direction=self.sheen_direction,
182: (12)                     family=propagate_colors,
183: (8)                 )
184: (8)             if not propagate_colors:
185: (12)                 for submobject in self.submobjects:
186: (16)                     submobject.init_colors(propagate_colors=False)
187: (8)             return self
188: (4)             def generate_rgbsas_array(
189: (8)                 self, color: ManimColor | list[ManimColor], opacity: float |
Iterable[float]
190: (4)             ) -> RGBA_Array_Float:
191: (8)                 """
192: (8)                     First arg can be either a color, or a tuple/list of colors.
193: (8)                     Likewise, opacity can either be a float, or a tuple of floats.
194: (8)                     If self.sheen_factor is not zero, and only
195: (8)                     one color was passed in, a second slightly light color
196: (8)                     will automatically be added for the gradient
197: (8)                 """
198: (8)                 colors: list[ManimColor] = [
199: (12)                     ManimColor(c) if (c is not None) else BLACK for c in
tuplify(color)
200: (8)                 ]
201: (8)                 opacities: list[float] = [
202: (12)                     o if (o is not None) else 0.0 for o in tuplify(opacity)
203: (8)                 ]
204: (8)                 rgbsas: npt.NDArray[RGBA_Array_Float] = np.array(
205: (12)                     [c.to_rgba_with_alpha(o) for c, o in zip(*make_even(colors,
opacities))],
206: (8)                 )
207: (8)                 sheen_factor = self.get_sheen_factor()
208: (8)                 if sheen_factor != 0 and len(rgbsas) == 1:
209: (12)                     light_rgbsas = np.array(rgbsas)
210: (12)                     light_rgbsas[:, :3] += sheen_factor
211: (12)                     np.clip(light_rgbsas, 0, 1, out=light_rgbsas)
212: (12)                     rgbsas = np.append(rgbsas, light_rgbsas, axis=0)
213: (8)                     return rgbsas
214: (4)             def update_rgbsas_array(
215: (8)                 self,
array_name: str,

```

```

217: (8)             color: ManimColor | None = None,
218: (8)             opacity: float | None = None,
219: (4)         ) -> Self:
220: (8)             rgbs = self.generate_rgbs_array(color, opacity)
221: (8)             if not hasattr(self, array_name):
222: (12)                 setattr(self, array_name, rgbs)
223: (12)             return self
224: (8)             curr_rgbs = getattr(self, array_name)
225: (8)             if len(curr_rgbs) < len(rgbs):
226: (12)                 curr_rgbs = stretch_array_to_length(curr_rgbs, len(rgbs))
227: (12)                 setattr(self, array_name, curr_rgbs)
228: (8)             elif len(rgbs) < len(curr_rgbs):
229: (12)                 rgbs = stretch_array_to_length(rgbs, len(curr_rgbs))
230: (8)             if color is not None:
231: (12)                 curr_rgbs[:, :3] = rgbs[:, :3]
232: (8)             if opacity is not None:
233: (12)                 curr_rgbs[:, 3] = rgbs[:, 3]
234: (8)             return self
235: (4)         def set_fill(
236: (8)             self,
237: (8)             color: ParsableManimColor | None = None,
238: (8)             opacity: float | None = None,
239: (8)             family: bool = True,
240: (4)         ) -> Self:
241: (8)             """Set the fill color and fill opacity of a :class:`VMobject`.
242: (8)             Parameters
243: (8)             -----
244: (8)             color
245: (12)                 Fill color of the :class:`VMobject`.
246: (8)             opacity
247: (12)                 Fill opacity of the :class:`VMobject`.
248: (8)             family
249: (12)                 If ``True``, the fill color of all subobjects is also set.
250: (8)             Returns
251: (8)             -----
252: (8)             :class:`VMobject`
253: (12)                 ``self``
254: (8)             Examples
255: (8)             -----
256: (8)             .. manim:: SetFill
257: (12)                 :save_last_frame:
258: (12)                 class SetFill(Scene):
259: (16)                     def construct(self):
260: (20)                         square = Square().scale(2).set_fill(WHITE,1)
261: (20)                         circle1 = Circle().set_fill(GREEN,0.8)
262: (20)                         circle2 = Circle().set_fill(YELLOW) # No fill_opacity
263: (20)                         circle3 = Circle().set_fill(color = '#FF2135', opacity =
0.2)
264: (20)                         group = Group(circle1,circle2,circle3).arrange()
265: (20)                         self.add(square)
266: (20)                         self.add(group)
267: (8)             See Also
268: (8)             -----
269: (8)             :meth:`~.VMobject.set_style`
270: (8)             """
271: (8)             if family:
272: (12)                 for subobject in self.subobjects:
273: (16)                     subobject.set_fill(color, opacity, family)
274: (8)                     self.update_rgbs_array("fill_rgbs", color, opacity)
275: (8)                     self.fill_rgbs: RGBA_Array_Float
276: (8)                     if opacity is not None:
277: (12)                         self.fill_opacity = opacity
278: (8)                     return self
279: (4)             def set_stroke(
280: (8)                 self,
281: (8)                 color: ParsableManimColor = None,
282: (8)                 width: float | None = None,
283: (8)                 opacity: float | None = None,
284: (8)                 background=False,

```

```

285: (8)             family: bool = True,
286: (4)         ) -> Self:
287: (8)         if family:
288: (12)             for submobject in self.submobjects:
289: (16)                 submobject.set_stroke(color, width, opacity, background,
family)
290: (8)             if background:
291: (12)                 array_name = "background_stroke_rgbs"
292: (12)                 width_name = "background_stroke_width"
293: (12)                 opacity_name = "background_stroke_opacity"
294: (8)             else:
295: (12)                 array_name = "stroke_rgbs"
296: (12)                 width_name = "stroke_width"
297: (12)                 opacity_name = "stroke_opacity"
298: (8)             self.update_rgbs_array(array_name, color, opacity)
299: (8)             if width is not None:
300: (12)                 setattr(self, width_name, width)
301: (8)             if opacity is not None:
302: (12)                 setattr(self, opacity_name, opacity)
303: (8)             if color is not None and background:
304: (12)                 if isinstance(color, (list, tuple)):
305: (16)                     self.background_stroke_color = ManimColor.parse(color)
306: (12)                 else:
307: (16)                     self.background_stroke_color = ManimColor(color)
308: (8)             return self
309: (4)         def set_cap_style(self, cap_style: CapStyleType) -> Self:
310: (8)             """
311: (8)             Sets the cap style of the :class:`VMobject` .
312: (8)             Parameters
313: (8)             -----
314: (8)             cap_style
315: (12)                 The cap style to be set. See :class:`CapStyleType` for options.
316: (8)             Returns
317: (8)             -----
318: (8)             :class:`VMobject`
319: (12)                 ``self``
320: (8)             Examples
321: (8)             -----
322: (8)             .. manim:: CapStyleExample
323: (12)                 :save_last_frame:
324: (12)                 class CapStyleExample(Scene):
325: (16)                     def construct(self):
326: (20)                         line = Line(LEFT, RIGHT, color=YELLOW, stroke_width=20)
327: (20)                         line.set_cap_style(CapStyleType.ROUND)
328: (20)                         self.add(line)
329: (8)                     """
330: (8)                     self.cap_style = cap_style
331: (8)                     return self
332: (4)         def set_background_stroke(self, **kwargs) -> Self:
333: (8)             kwargs["background"] = True
334: (8)             self.set_stroke(**kwargs)
335: (8)             return self
336: (4)         def set_style(
337: (8)             self,
338: (8)             fill_color: ParsableManimColor | None = None,
339: (8)             fill_opacity: float | None = None,
340: (8)             stroke_color: ParsableManimColor | None = None,
341: (8)             stroke_width: float | None = None,
342: (8)             stroke_opacity: float | None = None,
343: (8)             background_stroke_color: ParsableManimColor | None = None,
344: (8)             background_stroke_width: float | None = None,
345: (8)             background_stroke_opacity: float | None = None,
346: (8)             sheen_factor: float | None = None,
347: (8)             sheen_direction: Vector3D | None = None,
348: (8)             background_image: Image | str | None = None,
349: (8)             family: bool = True,
350: (4)         ) -> Self:
351: (8)             self.set_fill(color=fill_color, opacity=fill_opacity, family=family)
352: (8)             self.set_stroke(

```

```

353: (12)                               color=stroke_color,
354: (12)                               width=stroke_width,
355: (12)                               opacity=stroke_opacity,
356: (12)                               family=family,
357: (8)                                )
358: (8)                                self.set_background_stroke(
359: (12)                               color=background_stroke_color,
360: (12)                               width=background_stroke_width,
361: (12)                               opacity=background_stroke_opacity,
362: (12)                               family=family,
363: (8)                                )
364: (8)                                if sheen_factor:
365: (12)                               self.set_sheen(
366: (16)                               factor=sheen_factor,
367: (16)                               direction=sheen_direction,
368: (16)                               family=family,
369: (12)                                )
370: (8)                                if background_image:
371: (12)                               self.color_using_background_image(background_image)
372: (8)                                return self
373: (4)                                 def get_style(self, simple: bool = False) -> dict:
374: (8)                               ret = {
375: (12)                               "stroke_opacity": self.get_stroke_opacity(),
376: (12)                               "stroke_width": self.get_stroke_width(),
377: (8)                                }
378: (8)                                if simple:
379: (12)                               ret["fill_color"] = self.get_fill_color()
380: (12)                               ret["fill_opacity"] = self.get_fill_opacity()
381: (12)                               ret["stroke_color"] = self.get_stroke_color()
382: (8)                                else:
383: (12)                               ret["fill_color"] = self.get_fill_colors()
384: (12)                               ret["fill_opacity"] = self.get_fill_opacities()
385: (12)                               ret["stroke_color"] = self.get_stroke_colors()
386: (12)                               ret["background_stroke_color"] =
self.get_stroke_colors(background=True)
387: (12)                               ret["background_stroke_width"] =
self.get_stroke_width(background=True)
388: (12)                               ret["background_stroke_opacity"] =
self.get_stroke_opacity(background=True)
389: (12)                               ret["sheen_factor"] = self.get_sheen_factor()
390: (12)                               ret["sheen_direction"] = self.get_sheen_direction()
391: (12)                               ret["background_image"] = self.get_background_image()
392: (8)                                return ret
393: (4)                                 def match_style(self, vmobject: VMobject, family: bool = True) -> Self:
394: (8)                               self.set_style(**vmobject.get_style(), family=False)
395: (8)                                if family:
396: (12)                               submobs1, submobs2 = self.submobjects, vmobject.submobjects
397: (12)                               if len(submobs1) == 0:
398: (16)                               return self
399: (12)                               elif len(submobs2) == 0:
400: (16)                               submobs2 = [vmobject]
401: (12)                               for sm1, sm2 in zip(*make_even(submobs1, submobs2)):
402: (16)                               sm1.match_style(sm2)
403: (8)                                return self
404: (4)                                 def set_color(self, color: ParsableManimColor, family: bool = True) ->
Self:
405: (8)                               self.set_fill(color, family=family)
406: (8)                               self.set_stroke(color, family=family)
407: (8)                                return self
408: (4)                                 def set_opacity(self, opacity: float, family: bool = True) -> Self:
409: (8)                               self.set_fill(opacity=opacity, family=family)
410: (8)                               self.set_stroke(opacity=opacity, family=family)
411: (8)                               self.set_stroke(opacity=opacity, family=family, background=True)
412: (8)                                return self
413: (4)                                 def fade(self, darkness: float = 0.5, family: bool = True) -> Self:
414: (8)                               factor = 1.0 - darkness
415: (8)                               self.set_fill(opacity=factor * self.get_fill_opacity(), family=False)
416: (8)                               self.set_stroke(opacity=factor * self.get_stroke_opacity(),
family=False)

```

```

417: (8)             self.set_background_stroke(
418: (12)             opacity=factor * self.get_stroke_opacity(background=True),
419: (12)             family=False,
420: (8)         )
421: (8)         super().fade(darkness, family)
422: (8)         return self
423: (4)     def get_fill_rgbs(self) -> RGBA_Array_Float | Zeros:
424: (8)         try:
425: (12)             return self.fill_rgbs
426: (8)         except AttributeError:
427: (12)             return np.zeros((1, 4))
428: (4)     def get_fill_color(self) -> ManimColor:
429: (8)         """
430: (8)             If there are multiple colors (for gradient)
431: (8)             this returns the first one
432: (8)         """
433: (8)         return self.get_fill_colors()[0]
434: (4)     fill_color = property(get_fill_color, set_fill)
435: (4)     def get_fill_opacity(self) -> ManimFloat:
436: (8)         """
437: (8)             If there are multiple opacities, this returns the
438: (8)             first
439: (8)         """
440: (8)         return self.get_fill_opacities()[0]
441: (4)     def get_fill_colors(self) -> list[ManimColor | None]:
442: (8)         return [
443: (12)             ManimColor(rgb[:3]) if rgb.any() else None
444: (12)             for rgb in self.get_fill_rgbs()
445: (8)         ]
446: (4)     def get_fill_opacities(self) -> npt.NDArray[ManimFloat]:
447: (8)         return self.get_fill_rgbs()[:, 3]
448: (4)     def get_stroke_rgbs(self, background: bool = False) -> RGBA_Array_float |
Zeros:
449: (8)         try:
450: (12)             if background:
451: (16)                 self.background_stroke_rgbs: RGBA_Array_Float
452: (16)                 rgbs = self.background_stroke_rgbs
453: (12)             else:
454: (16)                 self.stroke_rgbs: RGBA_Array_Float
455: (16)                 rgbs = self.stroke_rgbs
456: (12)             return rgbs
457: (8)             except AttributeError:
458: (12)                 return np.zeros((1, 4))
459: (4)     def get_stroke_color(self, background: bool = False) -> ManimColor | None:
460: (8)         return self.get_stroke_colors(background)[0]
461: (4)     stroke_color = property(get_stroke_color, set_stroke)
462: (4)     def get_stroke_width(self, background: bool = False) -> float:
463: (8)         if background:
464: (12)             self.background_stroke_width: float
465: (12)             width = self.background_stroke_width
466: (8)         else:
467: (12)             width = self.stroke_width
468: (12)             if isinstance(width, str):
469: (16)                 width = int(width)
470: (8)             return max(0.0, width)
471: (4)     def get_stroke_opacity(self, background: bool = False) -> ManimFloat:
472: (8)         return self.get_stroke_opacities(background)[0]
473: (4)     def get_stroke_colors(self, background: bool = False) -> list[ManimColor | None]:
474: (8)         return [
475: (12)             ManimColor(rgb[:3]) if rgb.any() else None
476: (12)             for rgb in self.get_stroke_rgbs(background)
477: (8)         ]
478: (4)     def get_stroke_opacities(self, background: bool = False) ->
npt.NDArray[ManimFloat]:
479: (8)         return self.get_stroke_rgbs(background)[:, 3]
480: (4)     def get_color(self) -> ManimColor:
481: (8)         if np.all(self.get_fill_opacities() == 0):
482: (12)             return self.get_stroke_color()

```

```

483: (8)             return self.get_fill_color()
484: (4)             color = property(get_color, set_color)
485: (4)             def set_sheen_direction(self, direction: Vector3D, family: bool = True) ->
Self:
486: (8)                 """Sets the direction of the applied sheen.
487: (8)                 Parameters
488: (8)                 -----
489: (8)                 direction
490: (12)                   Direction from where the gradient is applied.
491: (8)                 Examples
492: (8)                 -----
493: (8)                 Normal usage:::
494: (12)                   Circle().set_sheen_direction(UP)
495: (8)                 See Also
496: (8)                 -----
497: (8)                 :meth:`~.VMobject.set_sheen`
498: (8)                 :meth:`~.VMobject.rotate_sheen_direction`  

499: (8)                 """
500: (8)                 direction = np.array(direction)
501: (8)                 if family:
502: (12)                   for submob in self.get_family():
503: (16)                     submob.sheen_direction = direction
504: (8)                 else:
505: (12)                   self.sheen_direction: Vector3D = direction
506: (8)                 return self
507: (4)             def rotate_sheen_direction(
508: (8)               self, angle: float, axis: Vector3D = OUT, family: bool = True
509: (4) ) -> Self:
510: (8)                 """Rotates the direction of the applied sheen.
511: (8)                 Parameters
512: (8)                 -----
513: (8)                 angle
514: (12)                   Angle by which the direction of sheen is rotated.
515: (8)                 axis
516: (12)                   Axis of rotation.
517: (8)                 Examples
518: (8)                 -----
519: (8)                 Normal usage:::
520: (12)                   Circle().set_sheen_direction(UP).rotate_sheen_direction(PI)
521: (8)                 See Also
522: (8)                 -----
523: (8)                 :meth:`~.VMobject.set_sheen_direction`  

524: (8)                 """
525: (8)                 if family:
526: (12)                   for submob in self.get_family():
527: (16)                     submob.sheen_direction = rotate_vector(
528: (20)                       submob.sheen_direction,
529: (20)                       angle,
530: (20)                       axis,
531: (16)                     )
532: (8)                 else:
533: (12)                   self.sheen_direction = rotate_vector(self.sheen_direction, angle,
axis)
534: (8)                 return self
535: (4)             def set_sheen(
536: (8)               self, factor: float, direction: Vector3D | None = None, family: bool =
True
537: (4) ) -> Self:
538: (8)                 """Applies a color gradient from a direction.
539: (8)                 Parameters
540: (8)                 -----
541: (8)                 factor
542: (12)                   The extent of lustre/gradient to apply. If negative, the gradient
543: (12)                     starts from black, if positive the gradient starts from white and
544: (12)                     changes to the current color.
545: (8)                 direction
546: (12)                   Direction from where the gradient is applied.
547: (8)                 Examples
548: (8)                 -----

```

```

549: (8)          .. manim:: SetSheen
550: (12)         :save_last_frame:
551: (12)         class SetSheen(Scene):
552: (16)           def construct(self):
553: (20)             circle = Circle(fill_opacity=1).set_sheen(-0.3, DR)
554: (20)             self.add(circle)
555: (8)           """
556: (8)           if family:
557: (12)             for submob in self.submobjects:
558: (16)               submob.set_sheen(factor, direction, family)
559: (8)           self.sheen_factor: float = factor
560: (8)           if direction is not None:
561: (12)             self.set_sheen_direction(direction, family=False)
562: (8)           if factor != 0:
563: (12)             self.set_stroke(self.get_stroke_color(), family=family)
564: (12)             self.set_fill(self.get_fill_color(), family=family)
565: (8)           return self
566: (4)           def get_sheen_direction(self) -> Vector3D:
567: (8)             return np.array(self.sheen_direction)
568: (4)           def get_sheen_factor(self) -> float:
569: (8)             return self.sheen_factor
570: (4)           def get_gradient_start_and_end_points(self) -> tuple[Point3D, Point3D]:
571: (8)             if self.shade_in_3d:
572: (12)               return get_3d_vmob_gradient_start_and_end_points(self)
573: (8)             else:
574: (12)               direction = self.get_sheen_direction()
575: (12)               c = self.get_center()
576: (12)               bases = np.array(
577: (16)                 [self.get_edge_center(vect) - c for vect in [RIGHT, UP, OUT]],
578: (12)               ).transpose()
579: (12)               offset = np.dot(bases, direction)
580: (12)               return (c - offset, c + offset)
581: (4)           def color_using_background_image(self, background_image: Image | str) ->
Self:
582: (8)             self.background_image: Image | str = background_image
583: (8)             self.set_color(WHITE)
584: (8)             for submob in self.submobjects:
585: (12)               submob.color_using_background_image(background_image)
586: (8)             return self
587: (4)           def get_background_image(self) -> Image | str:
588: (8)             return self.background_image
589: (4)           def match_background_image(self, vmobject: VMobject) -> Self:
590: (8)             self.color_using_background_image(vmobject.get_background_image())
591: (8)             return self
592: (4)           def set_shade_in_3d(
593: (8)             self, value: bool = True, z_index_as_group: bool = False
594: (4)           ) -> Self:
595: (8)             for submob in self.get_family():
596: (12)               submob.shade_in_3d = value
597: (12)               if z_index_as_group:
598: (16)                 submob.z_index_group = self
599: (8)             return self
600: (4)           def set_points(self, points: Point3D_Array) -> Self:
601: (8)             self.points: Point3D_Array = np.array(points)
602: (8)             return self
603: (4)           def resize_points(
604: (8)             self,
605: (8)             new_length: int,
606: (8)             resize_func: Callable[[Point3D, int], Point3D] = resize_array,
607: (4)           ) -> Self:
608: (8)             """Resize the array of anchor points and handles to have
609: (8)             the specified size.
610: (8)             Parameters
611: (8)             -----
612: (8)             new_length
613: (12)               The new (total) number of points.
614: (8)             resize_func
615: (12)               A function mapping a Numpy array (the points) and an integer
616: (12)               (the target size) to a Numpy array. The default implementation

```

```

617: (12)           is based on Numpy's ``resize`` function.
618: (8)
619: (8)
620: (12)
621: (8)
622: (4) def set_anchors_and_handles(
623: (8)     self,
624: (8)     anchors1: CubicBezierPoints,
625: (8)     handles1: CubicBezierPoints,
626: (8)     handles2: CubicBezierPoints,
627: (8)     anchors2: CubicBezierPoints,
628: (4) ) -> Self:
629: (8)     """Given two sets of anchors and handles, process them to set them as
anchors
630: (8)     and handles of the VMobject.
631: (8)     anchors1[i], handles1[i], handles2[i] and anchors2[i] define the i-th
bezier
632: (8)     curve of the vmobject. There are four hardcoded parameters and this is
a
633: (8)     problem as it makes the number of points per cubic curve unchangeable
from 4
634: (8)     (two anchors and two handles).
635: (8)     Returns
636: (8)     -----
637: (8)     :class:`VMobject`  

638: (12)     ``self``
639: (8)
640: (8)     assert len(anchors1) == len(handles1) == len(handles2) ==
len(anchors2)
641: (8)     nppcc = self.n_points_per_cubic_curve # 4
642: (8)     total_len = nppcc * len(anchors1)
643: (8)     self.points = np.zeros((total_len, self.dim))
644: (8)     arrays = [anchors1, handles1, handles2, anchors2]
645: (8)     for index, array in enumerate(arrays):
646: (12)         self.points[index::nppcc] = array
647: (8)     return self
648: (4) def clear_points(self) -> None:
649: (8)     self.points = np.zeros((0, self.dim))
650: (4) def append_points(self, new_points: Point3D_Array) -> Self:
651: (8)     self.points = np.append(self.points, new_points, axis=0)
652: (8)     return self
653: (4) def start_new_path(self, point: Point3D) -> Self:
654: (8)     if len(self.points) % 4 != 0:
655: (12)         last_anchor = self.get_start_anchors()[-1]
656: (12)         for _ in range(4 - (len(self.points) % 4)):
657: (16)             self.append_points([last_anchor])
658: (8)         self.append_points([point])
659: (8)     return self
660: (4) def add_cubic_bezier_curve(
661: (8)     self,
662: (8)     anchor1: CubicBezierPoints,
663: (8)     handle1: CubicBezierPoints,
664: (8)     handle2: CubicBezierPoints,
665: (8)     anchor2: CubicBezierPoints,
666: (4) ) -> None:
667: (8)     self.append_points([anchor1, handle1, handle2, anchor2])
668: (4) def add_cubic_bezier_curves(self, curves) -> None:
669: (8)     self.append_points(curves.flatten())
670: (4) def add_cubic_bezier_curve_to(
671: (8)     self,
672: (8)     handle1: CubicBezierPoints,
673: (8)     handle2: CubicBezierPoints,
674: (8)     anchor: CubicBezierPoints,
675: (4) ) -> Self:
676: (8)     """Add cubic bezier curve to the path.
677: (8)     NOTE : the first anchor is not a parameter as by default the end of
the last sub-path!
678: (8)     Parameters
679: (8)     -----

```

```

680: (8)           handle1
681: (12)          first handle
682: (8)           handle2
683: (12)          second handle
684: (8)           anchor
685: (12)          anchor
686: (8)           Returns
687: (8)           -----
688: (8)           :class:`VMobject`  
    ``self``
689: (12)          """
690: (8)
691: (8)           self.throw_error_if_no_points()
692: (8)           new_points = [handle1, handle2, anchor]
693: (8)           if self.has_new_path_started():
694: (12)             self.append_points(new_points)
695: (8)           else:
696: (12)             self.append_points([self.get_last_point()] + new_points)
697: (8)           return self
698: (4)            def add_quadratic_bezier_curve_to(
699: (8)              self,
700: (8)              handle: QuadraticBezierPoints,
701: (8)              anchor: QuadraticBezierPoints,
702: (4)            ) -> Self:
703: (8)              """Add Quadratic bezier curve to the path.
704: (8)              Returns
705: (8)              -----
706: (8)              :class:`VMobject`  
    ``self``
707: (12)            """
708: (8)
709: (8)           self.add_cubic_bezier_curve_to(
710: (12)             2 / 3 * handle + 1 / 3 * self.get_last_point(),
711: (12)             2 / 3 * handle + 1 / 3 * anchor,
712: (12)             anchor,
713: (8)
714: (8)           )
715: (4)            return self
716: (8)            def add_line_to(self, point: Point3D) -> Self:
717: (8)              """Add a straight line from the last point of VMobject to the given
718: (8)              point.
719: (8)
720: (12)            Parameters
721: (8)            -----
722: (8)            point
723: (8)              end of the straight line.
724: (12)
725: (8)
726: (8)            nppcc = self.n_points_per_cubic_curve
727: (8)            self.add_cubic_bezier_curve_to(
728: (12)              *(
729: (16)                interpolate(self.get_last_point(), point, a)
730: (16)                for a in np.linspace(0, 1, nppcc)[1:]
731: (12)
732: (8)
733: (8)            )
734: (4)            return self
735: (8)            def add_smooth_curve_to(self, *points: Point3D) -> Self:
736: (8)              """Creates a smooth curve from given points and add it to the
737: (8)              VMobject. If two points are passed in, the first is interpreted
738: (8)              as a handle, the second as an anchor.
739: (8)
740: (12)            Parameters
741: (8)            -----
742: (8)            points
743: (8)              Points (anchor and handle, or just anchor) to add a smooth curve
744: (12)
745: (8)            Returns
746: (8)            -----
747: (8)            :class:`VMobject`  
    ``self``
748: (12)
749: (8)            Raises

```

```

746: (8)           -----
747: (8)           ValueError
748: (12)          If 0 or more than 2 points are given.
749: (8)
750: (8)
751: (12)
752: (12)
753: (8)
754: (12)
755: (8)
756: (12)
757: (12)
758: (8)
759: (12)
760: (8)
761: (12)
762: (12)
763: (12)
764: (12)
765: (12)
766: (16)
767: (16)
axis=to_anchor_vect)
768: (16)
769: (12)
770: (8)
771: (4)
772: (8)
773: (8)
774: (4)
775: (8)
776: (4)
777: (8)
778: (4)
779: (8)
780: (12)
781: (4)
Iterable[Point3D]:
782: (8)
783: (12)
784: (8)
785: (4)
786: (8)
787: (8)
such that the resultant bezier curve will be the segment
788: (8)
between the two anchors.
789: (8)
Parameters
790: (8)
791: (8)
792: (12)
793: (8)
794: (8)
795: (8)
796: (12)
797: (8)
798: (8)
799: (8)
800: (12)
801: (12)
802: (16)
803: (20)
804: (24)
805: (24)
806: (24)
807: (24)
808: (24)
809: (20)
810: (20)
811: (20)
-----ValueError
If 0 or more than 2 points are given.
"""
if len(points) == 1:
    handle2 = None
    new_anchor = points[0]
elif len(points) == 2:
    handle2, new_anchor = points
else:
    name = sys._getframe(0).f_code.co_name
    raise ValueError(f"Only call {name} with 1 or 2 points")
if self.has_new_path_started():
    self.add_line_to(new_anchor)
else:
    self.throw_error_if_no_points()
last_h2, last_a2 = self.points[-2:]
last_tangent = last_a2 - last_h2
handle1 = last_a2 + last_tangent
if handle2 is None:
    to_anchor_vect = new_anchor - last_a2
    new_tangent = rotate_vector(last_tangent, PI,
                                handle2 = new_anchor - new_tangent
                                self.append_points([last_a2, handle1, handle2, new_anchor])
return self
def has_new_path_started(self) -> bool:
    nppcc = self.n_points_per_cubic_curve # 4
    return len(self.points) % nppcc == 1
def get_last_point(self) -> Point3D:
    return self.points[-1]
def is_closed(self) -> bool:
    return self.consider_points_equals(self.points[0], self.points[-1])
def close_path(self) -> None:
    if not self.is_closed():
        self.add_line_to(self.get_subpaths()[-1][0])
def add_points_as_corners(self, points: Iterable[Point3D]) ->
    for point in points:
        self.add_line_to(point)
    return points
def set_points_as_corners(self, points: Point3D_Array) -> Self:
    """
    Given an array of points, set them as corner of the vmobject.
    To achieve that, this algorithm sets handles aligned with the anchors
    Parameters
    -----
    points
        Array of points that will be set as corners.
    Returns
    -----
    :class:`VMobject`  

    ``self``
Examples
-----
.. manim:: PointsAsCornersExample
    :save_last_frame:
    class PointsAsCornersExample(Scene):
        def construct(self):
            corners = (
                UR, UL,
                DL, DR,
                UR,
                DL, UL,
                DR
            )
            vmob = VMobject(stroke_color=RED)
            vmob.set_points_as_corners(corners).scale(2)

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
812: (20)                         self.add(vmob)
813: (8)                           """
814: (8)                           nppcc = self.n_points_per_cubic_curve
815: (8)                           points = np.array(points)
816: (8)                           self.set_anchors_and_handles(
817: (12)                             *(interpolate(points[:-1], points[1:], a) for a in np.linspace(0,
1, nppcc)))
818: (8)                           )
819: (8)                           return self
820: (4)                           def set_points_smoothly(self, points: Point3D_Array) -> Self:
821: (8)                             self.set_points_as_corners(points)
822: (8)                             self.make_smooth()
823: (8)                             return self
824: (4)                           def change_anchor_mode(self, mode: Literal["jagged", "smooth"]) -> Self:
825: (8)                             """Changes the anchor mode of the bezier curves. This will modify the
handles.
826: (8)                             There can be only two modes, "jagged", and "smooth".
827: (8)                             Returns
828: (8)                             -----
829: (8)                             :class:`VMobject`  

830: (12)                               ``self``
831: (8)                               """
832: (8)                               assert mode in ["jagged", "smooth"], 'mode must be either "jagged" or
"smooth"
833: (8)                               nppcc = self.n_points_per_cubic_curve
834: (8)                               for submob in self.family_members_with_points():
835: (12)                                 subpaths = submob.get_subpaths()
836: (12)                                 submob.clear_points()
837: (12)                                 for subpath in subpaths:
838: (16)                                   anchors = np.append(subpath[::-nppcc], subpath[-1:], 0)
839: (16)                                   if mode == "smooth":
840: (20)                                     h1, h2 = get_smooth_handle_points(anchors)
841: (16)                                   else: # mode == "jagged"
842: (20)                                     a1 = anchors[:-1]
843: (20)                                     a2 = anchors[1:]
844: (20)                                     h1 = interpolate(a1, a2, 1.0 / 3)
845: (20)                                     h2 = interpolate(a1, a2, 2.0 / 3)
846: (16)                                     new_subpath = np.array(subpath)
847: (16)                                     new_subpath[1::nppcc] = h1
848: (16)                                     new_subpath[2::nppcc] = h2
849: (16)                                     submob.append_points(new_subpath)
850: (8)                           return self
851: (4)                           def make_smooth(self) -> Self:
852: (8)                             return self.change_anchor_mode("smooth")
853: (4)                           def make_jagged(self) -> Self:
854: (8)                             return self.change_anchor_mode("jagged")
855: (4)                           def add_subpath(self, points: Point3D_Array) -> Self:
856: (8)                             assert len(points) % 4 == 0
857: (8)                             self.points: Point3D_Array = np.append(self.points, points, axis=0)
858: (8)                             return self
859: (4)                           def append_vectorized_mobject(self, vectorized_mobject: VMobject) -> None:
860: (8)                             new_points = list(vectorized_mobject.points)
861: (8)                             if self.has_new_path_started():
862: (12)                               self.points = self.points[:-1]
863: (8)                               self.append_points(new_points)
864: (4)                           def apply_function(self, function: MappingFunction) -> Self:
865: (8)                             factor = self.pre_function_handle_to_anchor_scale_factor
866: (8)                             self.scale_handle_to_anchor_distances(factor)
867: (8)                             super().apply_function(function)
868: (8)                             self.scale_handle_to_anchor_distances(1.0 / factor)
869: (8)                             if self.make_smooth_after_applying_functions:
870: (12)                               self.make_smooth()
871: (8)                             return self
872: (4)                           def rotate(
873: (8)                             self,
874: (8)                             angle: float,
875: (8)                             axis: Vector3D = OUT,
876: (8)                             about_point: Point3D | None = None,
877: (8)                             **kwargs,

```

```

878: (4)             ) -> Self:
879: (8)             self.rotate_sheen_direction(angle, axis)
880: (8)             super().rotate(angle, axis, about_point, **kwargs)
881: (8)             return self
882: (4)             def scale_handle_to_anchor_distances(self, factor: float) -> Self:
883: (8)             """If the distance between a given handle point H and its associated
884: (8)             anchor point A is d, then it changes H to be a distances factor*d
885: (8)             away from A, but so that the line from A to H doesn't change.
886: (8)             This is mostly useful in the context of applying a (differentiable)
887: (8)             function, to preserve tangency properties. One would pull all the
888: (8)             handles closer to their anchors, apply the function then push them out
889: (8)             again.
890: (8)             Parameters
891: (8)             -----
892: (8)             factor
893: (12)             The factor used for scaling.
894: (8)             Returns
895: (8)             -----
896: (8)             :class:`VMobject`  

897: (12)             ``self``
898: (8)             """
899: (8)             for submob in self.family_members_with_points():
900: (12)             if len(submob.points) < self.n_points_per_cubic_curve:
901: (16)                 continue
902: (12)             a1, h1, h2, a2 = submob.get_anchors_and_handles()
903: (12)             a1_to_h1 = h1 - a1
904: (12)             a2_to_h2 = h2 - a2
905: (12)             new_h1 = a1 + factor * a1_to_h1
906: (12)             new_h2 = a2 + factor * a2_to_h2
907: (12)             submob.set_anchors_and_handles(a1, new_h1, new_h2, a2)
908: (8)             return self
909: (4)             def consider_points_equals(self, p0: Point3D, p1: Point3D) -> bool:
910: (8)             return np.allclose(p0, p1, atol=self.tolerance_for_point_equality)
911: (4)             def consider_points_equals_2d(self, p0: Point2D, p1: Point2D) -> bool:
912: (8)             """Determine if two points are close enough to be considered equal.
913: (8)             This uses the algorithm from np.isclose(), but expanded here for the
914: (8)             2D point case. NumPy is overkill for such a small question.
915: (8)             Parameters
916: (8)             -----
917: (8)             p0
918: (12)             first point
919: (8)             p1
920: (12)             second point
921: (8)             Returns
922: (8)             -----
923: (8)             bool
924: (12)             whether two points considered close.
925: (8)             """
926: (8)             rtol = 1.0e-5 # default from np.isclose()
927: (8)             atol = self.tolerance_for_point_equality
928: (8)             if abs(p0[0] - p1[0]) > atol + rtol * abs(p1[0]):
929: (12)                 return False
930: (8)             if abs(p0[1] - p1[1]) > atol + rtol * abs(p1[1]):
931: (12)                 return False
932: (8)             return True
933: (4)             def get_cubic_bezier_tuples_from_points(
934: (8)                 self, points: Point3D_Array
935: (4)             ) -> npt.NDArray[Point3D_Array]:
936: (8)                 return np.array(self.gen_cubic_bezier_tuples_from_points(points))
937: (4)             def gen_cubic_bezier_tuples_from_points(
938: (8)                 self, points: Point3D_Array
939: (4)             ) -> tuple[Point3D_Array]:
940: (8)                 """Returns the bezier tuples from an array of points.
941: (8)                 self.points is a list of the anchors and handles of the bezier curves
of the mobject (ie [anchor1, handle1, handle2, anchor2, anchor3 ...])
942: (8)                 This algorithm basically retrieve them by taking an element every n,
where n is the number of control points
943: (8)                         of the bezier curve.
944: (8)             Parameters

```

```

945: (8)           -----
946: (8)           points
947: (12)         Points from which control points will be extracted.
948: (8)           Returns
949: (8)           -----
950: (8)           tuple
951: (12)         Bezier control points.
952: (8)           """
953: (8)           nppcc = self.n_points_per_cubic_curve
954: (8)           remainder = len(points) % nppcc
955: (8)           points = points[: len(points) - remainder]
956: (8)           return tuple(points[i : i + nppcc] for i in range(0, len(points),
nppcc))
957: (4)           def get_cubic_bezier_tuples(self) -> npt.NDArray[Point3D_Array]:
958: (8)             return self.get_cubic_bezier_tuples_from_points(self.points)
959: (4)           def _gen_subpaths_from_points(
960: (8)             self,
961: (8)             points: Point3D_Array,
962: (8)             filter_func: Callable[[int], bool],
963: (4)           ) -> Generator[Point3D_Array]:
964: (8)             """Given an array of points defining the bezier curves of the
vmobject, return subpaths formed by these points.
965: (8)             Here, Two bezier curves form a path if at least two of their anchors
are evaluated True by the relation defined by filter_func.
966: (8)             The algorithm every bezier tuple (anchors and handles) in
``self.points`` (by regrouping each n elements, where
967: (8)               n is the number of points per cubic curve)), and evaluate the relation
between two anchors with filter_func.
968: (8)             NOTE : The filter_func takes an int n as parameter, and will evaluate
the relation between points[n] and points[n - 1]. This should probably be changed so
969: (8)               the function takes two points as parameters.
970: (8)             Parameters
971: (8)             -----
972: (8)             points
973: (12)           points defining the bezier curve.
974: (8)             filter_func
975: (12)           Filter-func defining the relation.
976: (8)             Returns
977: (8)             -----
978: (8)             Generator[Point3D_Array]
979: (12)           subpaths formed by the points.
980: (8)           """
981: (8)           nppcc = self.n_points_per_cubic_curve
982: (8)           filtered = filter(filter_func, range(nppcc, len(points), nppcc))
983: (8)           split_indices = [0] + list(filtered) + [len(points)]
984: (8)           return (
985: (12)             points[i1:i2]
986: (12)             for i1, i2 in zip(split_indices, split_indices[1:])
987: (12)             if (i2 - i1) >= nppcc
988: (8)           )
989: (4)           def get_subpaths_from_points(self, points: Point3D_Array) ->
list[Point3D_Array]:
990: (8)             return list(
991: (12)               self._gen_subpaths_from_points(
992: (16)                 points,
993: (16)                 lambda n: not self.consider_points_equals(points[n - 1],
points[n]),
994: (12)               ),
995: (8)             )
996: (4)             def gen_subpaths_from_points_2d(
997: (8)               self, points: Point3D_Array
998: (4)             ) -> Generator[Point3D_Array]:
999: (8)               return self._gen_subpaths_from_points(
1000: (12)                 points,
1001: (12)                 lambda n: not self.consider_points_equals_2d(points[n - 1],
points[n]),
1002: (8)               )
1003: (4)             def get_subpaths(self) -> list[Point3D_Array]:
1004: (8)               """Returns subpaths formed by the curves of the VMobject.

```

```

1005: (8)           Subpaths are ranges of curves with each pair of consecutive curves
having their end/start points coincident.
1006: (8)           Returns
1007: (8)           -----
1008: (8)           list[Point3D_Array]
1009: (12)          subpaths.
1010: (8)          """
1011: (8)          return self.get_subpaths_from_points(self.points)
1012: (4) def get_nth_curve_points(self, n: int) -> Point3D_Array:
1013: (8)         """Returns the points defining the nth curve of the vobject.
1014: (8)         Parameters
1015: (8)         -----
1016: (8)         n
1017: (12)          index of the desired bezier curve.
1018: (8)         Returns
1019: (8)         -----
1020: (8)         Point3D_Array
1021: (12)          points defining the nth bezier curve (anchors, handles)
1022: (8)         """
1023: (8)         assert n < self.get_num_curves()
1024: (8)         nppcc = self.n_points_per_cubic_curve
1025: (8)         return self.points[nppcc * n : nppcc * (n + 1)]
1026: (4) def get_nth_curve_function(self, n: int) -> Callable[[float], Point3D]:
1027: (8)         """Returns the expression of the nth curve.
1028: (8)         Parameters
1029: (8)         -----
1030: (8)         n
1031: (12)          index of the desired curve.
1032: (8)         Returns
1033: (8)         -----
1034: (8)         Callable[float, Point3D]
1035: (12)          expression of the nth bezier curve.
1036: (8)         """
1037: (8)         return bezier(self.get_nth_curve_points(n))
1038: (4) def get_nth_curve_length_pieces(
1039: (8)         self,
1040: (8)         n: int,
1041: (8)         sample_points: int | None = None,
1042: (4) ) -> npt.NDArray[ManimFloat]:
1043: (8)         """Returns the array of short line lengths used for length
approximation.
1044: (8)         Parameters
1045: (8)         -----
1046: (8)         n
1047: (12)          The index of the desired curve.
1048: (8)         sample_points
1049: (12)          The number of points to sample to find the length.
1050: (8)         Returns
1051: (8)         -----
1052: (12)          The short length-pieces of the nth curve.
1053: (8)         """
1054: (8)         if sample_points is None:
1055: (12)             sample_points = 10
1056: (8)         curve = self.get_nth_curve_function(n)
1057: (8)         points = np.array([curve(a) for a in np.linspace(0, 1,
sample_points)])
1058: (8)         diffs = points[1:] - points[:-1]
1059: (8)         norms = np.linalg.norm(diffs, axis=1)
1060: (8)         return norms
1061: (4) def get_nth_curve_length(
1062: (8)         self,
1063: (8)         n: int,
1064: (8)         sample_points: int | None = None,
1065: (4) ) -> float:
1066: (8)         """Returns the (approximate) length of the nth curve.
1067: (8)         Parameters
1068: (8)         -----
1069: (8)         n
1070: (12)          The index of the desired curve.

```

```

1071: (8)           sample_points
1072: (12)         The number of points to sample to find the length.
1073: (8)           Returns
1074: (8)         -----
1075: (8)           length : :class:`float`
1076: (12)         The length of the nth curve.
1077: (8)         """
1078: (8)           _, length = self.get_nth_curve_function_with_length(n, sample_points)
1079: (8)           return length
1080: (4) def get_nth_curve_function_with_length(
1081: (8)     self,
1082: (8)     n: int,
1083: (8)     sample_points: int | None = None,
1084: (4) ) -> tuple[Callable[[float], Point3D], float]:
1085: (8)     """Returns the expression of the nth curve along with its
(approximate) length.
1086: (8)           Parameters
1087: (8)           -----
1088: (8)           n
1089: (12)         The index of the desired curve.
1090: (8)           sample_points
1091: (12)         The number of points to sample to find the length.
1092: (8)           Returns
1093: (8)           -----
1094: (8)           curve : Callable[[float], Point3D]
1095: (12)         The function for the nth curve.
1096: (8)           length : :class:`float`
1097: (12)         The length of the nth curve.
1098: (8)         """
1099: (8)           curve = self.get_nth_curve_function(n)
1100: (8)           norms = self.get_nth_curve_length_pieces(n,
sample_points=sample_points)
1101: (8)           length = np.sum(norms)
1102: (8)           return curve, length
1103: (4) def get_num_curves(self) -> int:
1104: (8)     """Returns the number of curves of the vobject.
1105: (8)           Returns
1106: (8)           -----
1107: (8)           int
1108: (12)         number of curves of the vobject.
1109: (8)         """
1110: (8)           nppcc = self.n_points_per_cubic_curve
1111: (8)           return len(self.points) // nppcc
1112: (4) def get_curve_functions(
1113: (8)     self,
1114: (4) ) -> Generator[Callable[[float], Point3D]]:
1115: (8)     """Gets the functions for the curves of the mobject.
1116: (8)           Returns
1117: (8)           -----
1118: (8)           Generator[Callable[[float], Point3D]]
1119: (12)         The functions for the curves.
1120: (8)         """
1121: (8)           num_curves = self.get_num_curves()
1122: (8)           for n in range(num_curves):
1123: (12)             yield self.get_nth_curve_function(n)
1124: (4) def get_curve_functions_with_lengths(
1125: (8)     self, **kwargs
1126: (4) ) -> Generator[tuple[Callable[[float], Point3D], float]]:
1127: (8)     """Gets the functions and lengths of the curves for the mobject.
1128: (8)           Parameters
1129: (8)           -----
1130: (8)           **kwargs
1131: (12)             The keyword arguments passed to
:meth:`get_nth_curve_function_with_length`"
1132: (8)           Returns
1133: (8)           -----
1134: (8)           Generator[tuple[Callable[[float], Point3D], float]]
1135: (12)         The functions and lengths of the curves.
1136: (8)         """

```

```

1137: (8)             num_curves = self.get_num_curves()
1138: (8)             for n in range(num_curves):
1139: (12)                 yield self.get_nth_curve_function_with_length(n, **kwargs)
1140: (4)             def point_from_proportion(self, alpha: float) -> Point3D:
1141: (8)                 """Gets the point at a proportion along the path of the
:class:`VMobject`.
1142: (8)             Parameters
1143: (8)             -----
1144: (8)                 alpha
1145: (12)                     The proportion along the the path of the :class:`VMobject`.
1146: (8)             Returns
1147: (8)             -----
1148: (8)                 :class:`numpy.ndarray`
1149: (12)                     The point on the :class:`VMobject`.
1150: (8)             Raises
1151: (8)             -----
1152: (8)                 :exc:`ValueError`
1153: (12)                     If ``alpha`` is not between 0 and 1.
1154: (8)                 :exc:`Exception`
1155: (12)                     If the :class:`VMobject` has no points.
1156: (8)             Example
1157: (8)             -----
1158: (8)                 .. manim:: PointFromProportion
1159: (12)                     :save_last_frame:
1160: (12)                     class PointFromProportion(Scene):
1161: (16)                         def construct(self):
1162: (20)                             line = Line(2*DL, 2*UR)
1163: (20)                             self.add(line)
1164: (20)                             colors = (RED, BLUE, YELLOW)
1165: (20)                             proportions = (1/4, 1/2, 3/4)
1166: (20)                             for color, proportion in zip(colors, proportions):
1167: (24)                                 self.add(Dot(color=color).move_to(
1168: (32)                                     line.point_from_proportion(proportion)
1169: (24)                               ))
1170: (8)                         """
1171: (8)                         if alpha < 0 or alpha > 1:
1172: (12)                             raise ValueError(f"Alpha {alpha} not between 0 and 1.")
1173: (8)                         self.throw_error_if_no_points()
1174: (8)                         if alpha == 1:
1175: (12)                             return self.points[-1]
1176: (8)
1177: (8)
curves_and_lengths)
1178: (8)             curves_and_lengths = tuple(self.get_curve_functions_with_lengths())
1179: (8)             target_length = alpha * sum(length for _, length in
1180: (12)                 curves_and_lengths)
1181: (16)             current_length = 0
1182: (20)             for curve, length in curves_and_lengths:
1183: (16)                 if current_length + length >= target_length:
1184: (20)                     if length != 0:
1185: (16)                         residue = (target_length - current_length) / length
1186: (12)                     else:
1187: (8)                         residue = 0
1188: (12)                     return curve(residue)
1189: (8)                     current_length += length
raise Exception(
    "Not sure how you reached here, please file a bug report at
https://github.com/ManimCommunity/manim/issues/new/choose")
1190: (4)             )
1191: (8)             def proportion_from_point(
1192: (8)                 self,
1193: (4)                 point: Iterable[float | int],
) -> float:
1194: (8)                 """Returns the proportion along the path of the :class:`VMobject`
1195: (8)                 a particular given point is at.
1196: (8)             Parameters
1197: (8)             -----
1198: (8)                 point
1199: (12)                     The Cartesian coordinates of the point which may or may not lie on
the :class:`VMobject`.
1200: (8)             Returns
1201: (8)             -----

```

```

1202: (8)           float
1203: (12)          The proportion along the path of the :class:`VMobject`.
1204: (8)          Raises
1205: (8)          -----
1206: (8)          :exc:`ValueError`
1207: (12)          If ``point`` does not lie on the curve.
1208: (8)          :exc:`Exception`
1209: (12)          If the :class:`VMobject` has no points.
1210: (8)          """
1211: (8)          self.throw_error_if_no_points()
1212: (8)          num_curves = self.get_num_curves()
1213: (8)          total_length = self.get_arc_length()
1214: (8)          target_length = 0
1215: (8)          for n in range(num_curves):
1216: (12)              control_points = self.get_nth_curve_points(n)
1217: (12)              length = self.get_nth_curve_length(n)
1218: (12)              proportions_along_bezier =
proportions_along_bezier_curve_for_point(
1219: (16)                  point,
1220: (16)                  control_points,
1221: (12)              )
1222: (12)              if len(proportions_along_bezier) > 0:
1223: (16)                  proportion_along_nth_curve = max(proportions_along_bezier)
1224: (16)                  target_length += length * proportion_along_nth_curve
1225: (16)                  break
1226: (12)              target_length += length
1227: (8)          else:
1228: (12)              raise ValueError(f"Point {point} does not lie on this curve.")
1229: (8)          alpha = target_length / total_length
1230: (8)          return alpha
1231: (4)          def get_anchors_and_handles(self) -> list[Point3D_Array]:
1232: (8)              """Returns anchors1, handles1, handles2, anchors2,
1233: (8)              where (anchors1[i], handles1[i], handles2[i], anchors2[i])
1234: (8)              will be four points defining a cubic bezier curve
1235: (8)              for any i in range(0, len(anchors1))
1236: (8)          Returns
1237: (8)          -----
1238: (8)          `list[Point3D_Array]`
1239: (12)          Iterable of the anchors and handles.
1240: (8)          """
1241: (8)          nppcc = self.n_points_per_cubic_curve
1242: (8)          return [self.points[i::nppcc] for i in range(nppcc)]
1243: (4)          def get_start_anchors(self) -> Point3D_Array:
1244: (8)              """Returns the start anchors of the bezier curves.
1245: (8)          Returns
1246: (8)          -----
1247: (8)          Point3D_Array
1248: (12)          Starting anchors
1249: (8)          """
1250: (8)          return self.points[: self.n_points_per_cubic_curve]
1251: (4)          def get_end_anchors(self) -> Point3D_Array:
1252: (8)              """Return the end anchors of the bezier curves.
1253: (8)          Returns
1254: (8)          -----
1255: (8)          Point3D_Array
1256: (12)          Starting anchors
1257: (8)          """
1258: (8)          nppcc = self.n_points_per_cubic_curve
1259: (8)          return self.points[nppcc - 1 :: nppcc]
1260: (4)          def get_anchors(self) -> Point3D_Array:
1261: (8)              """Returns the anchors of the curves forming the VMobject.
1262: (8)          Returns
1263: (8)          -----
1264: (8)          Point3D_Array
1265: (12)          The anchors.
1266: (8)          """
1267: (8)          if self.points.shape[0] == 1:
1268: (12)              return self.points
1269: (8)          s = self.get_start_anchors()

```

manims\_installed\_to\_implement\_with\_qhenomenology\_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

1270: (8)             e = self.get_end_anchors()
1271: (8)             return list(it.chain.from_iterable(zip(s, e)))
1272: (4)             def get_points_defining_boundary(self) -> Point3D_Array:
1273: (8)                 return np.array(
1274: (12)                     tuple(it.chain(*[sm.get_anchors() for sm in self.get_family()])))
1275: (8)
1276: (4)             def get_arc_length(self, sample_points_per_curve: int | None = None) ->
1277: (8)                 """Return the approximated length of the whole curve.
1278: (8)                 Parameters
1279: (8)                 -----
1280: (8)                 sample_points_per_curve
1281: (12)                     Number of sample points per curve used to approximate the length.

```

More points result in a better approximation.

```

1282: (8)             Returns
1283: (8)             -----
1284: (8)             float
1285: (12)                 The length of the :class:`VMobject`.
1286: (8)
1287: (8)             """
1288: (12)             return sum(
1289: (12)                 length
1290: (16)                 for _, length in self.get_curve_functions_with_lengths(
1291: (12)                     sample_points=sample_points_per_curve,
1292: (8)
1293: (4)             def align_points(self, vmobject: VMobject) -> Self:
1294: (8)                 """Adds points to self and vmobject so that they both have the same

```

number of subpaths, with

```

1295: (8)             corresponding subpaths each containing the same number of points.
1296: (8)             Points are added either by subdividing curves evenly along the
subpath, or by creating new subpaths consisting
1297: (8)                 of a single point repeated.
1298: (8)
1299: (8)
1300: (8)
1301: (12)             Parameters
1302: (8)
1303: (8)
1304: (8)
1305: (11)
1306: (8)
1307: (8)
1308: (8)
1309: (12)
1310: (8)
1311: (12)
1312: (16)
1313: (12)
1314: (16)
1315: (8)
1316: (8)
1317: (8)
1318: (8)
1319: (8)
1320: (8)
1321: (8)
1322: (12)
1323: (16)
1324: (12)
1325: (12)
1326: (16)

```

1327: (20) The object to align points with.

1328: (16) Returns

1329: (8)

1330: (8) :class:`VMobject`

1331: (11) ``self``

1332: (8)

1333: (8) self.align\_rgbs(vmobject)

1334: (8) if self.get\_num\_points() == vmobject.get\_num\_points():

1335: (12) return

1336: (8) for mob in self, vmobject:

1337: (12) if mob.has\_no\_points():

1338: (16) mob.start\_new\_path(mob.get\_center())

1339: (12) if mob.has\_new\_path\_started():

1340: (16) mob.add\_line\_to(mob.get\_last\_point())

1341: (8) subpaths1 = self.get\_subpaths()

1342: (8) subpaths2 = vmobject.get\_subpaths()

1343: (8) n\_subpaths = max(len(subpaths1), len(subpaths2))

1344: (8) new\_path1 = np.zeros((0, self.dim))

1345: (8) new\_path2 = np.zeros((0, self.dim))

1346: (8) nppcc = self.n\_points\_per\_cubic\_curve

1347: (8) def get\_nth\_subpath(path\_list, n):

1348: (12) if n >= len(path\_list):

1349: (16) return [path\_list[-1][-1]] \* nppcc

1350: (12) path = path\_list[n]

1351: (12) while len(path) > nppcc:

1352: (16) if self.consider\_points\_equals(path[-nppcc:], path[-nppcc - 1]):

1353: (12) path = path[:-nppcc]

1354: (8) else:

1355: (12) break

1356: (12) return path

1357: (8) for n in range(n\_subpaths):

1358: (12) sp1 = get\_nth\_subpath(subpaths1, n)

1359: (12) sp2 = get\_nth\_subpath(subpaths2, n)

```

1334: (12)             diff1 = max(0, (len(sp2) - len(sp1)) // nppcc)
1335: (12)             diff2 = max(0, (len(sp1) - len(sp2)) // nppcc)
1336: (12)             sp1 = self.insert_n_curves_to_point_list(diff1, sp1)
1337: (12)             sp2 = self.insert_n_curves_to_point_list(diff2, sp2)
1338: (12)             new_path1 = np.append(new_path1, sp1, axis=0)
1339: (12)             new_path2 = np.append(new_path2, sp2, axis=0)
1340: (8)              self.set_points(new_path1)
1341: (8)              vmobject.set_points(new_path2)
1342: (8)              return self
1343: (4)  def insert_n_curves(self, n: int) -> Self:
1344: (8)      """Inserts n curves to the bezier curves of the vmobject.
1345: (8)      Parameters
1346: (8)      -----
1347: (8)      n
1348: (12)          Number of curves to insert.
1349: (8)      Returns
1350: (8)      -----
1351: (8)      :class:`VMobject`  
`self`  
"""
1352: (12)
1353: (8)
1354: (8)      new_path_point = None
1355: (8)      if self.has_new_path_started():
1356: (12)          new_path_point = self.get_last_point()
1357: (8)      new_points = self.insert_n_curves_to_point_list(n, self.points)
1358: (8)      self.set_points(new_points)
1359: (8)      if new_path_point is not None:
1360: (12)          self.append_points([new_path_point])
1361: (8)      return self
1362: (4)  def insert_n_curves_to_point_list(
1363: (8)      self, n: int, points: Point3D_Array
1364: (4) ) -> npt.NDArray[BezierPoints]:
1365: (8)      """Given an array of k points defining a bezier curves (anchors and
handles), returns points defining exactly k + n bezier curves.
1366: (8)      Parameters
1367: (8)      -----
1368: (8)      n
1369: (12)          Number of desired curves.
1370: (8)      points
1371: (12)          Starting points.
1372: (8)      Returns
1373: (8)      -----
1374: (12)          Points generated.
1375: (8)      """
1376: (8)      if len(points) == 1:
1377: (12)          nppcc = self.n_points_per_cubic_curve
1378: (12)          return np.repeat(points, nppcc * n, 0)
1379: (8)      bezier_quads = self.get_cubic_bezier_tuples_from_points(points)
1380: (8)      curr_num = len(bezier_quads)
1381: (8)      target_num = curr_num + n
1382: (8)      repeat_indices = (np.arange(target_num, dtype="i") * curr_num) //
target_num
1383: (8)      split_factors = np.zeros(curr_num, dtype="i")
1384: (8)      for val in repeat_indices:
1385: (12)          split_factors[val] += 1
1386: (8)      new_points = np.zeros((0, self.dim))
1387: (8)      for quad, sf in zip(bezier_quads, split_factors):
1388: (12)          alphas = np.linspace(0, 1, sf + 1)
1389: (12)          for a1, a2 in zip(alphas, alphas[1:]):
1390: (16)              new_points = np.append(
1391: (20)                  new_points,
1392: (20)                  partial_bezier_points(quad, a1, a2),
1393: (20)                  axis=0,
1394: (16)              )
1395: (8)      return new_points
1396: (4)  def align_rgbs(self, vmobject: VMobject) -> Self:
1397: (8)      attrs = ["fill_rgbs", "stroke_rgbs", "background_stroke_rgbs"]
1398: (8)      for attr in attrs:
1399: (12)          a1 = getattr(self, attr)
1400: (12)          a2 = getattr(vmobject, attr)

```

```

1401: (12)             if len(a1) > len(a2):
1402: (16)                 new_a2 = stretch_array_to_length(a2, len(a1))
1403: (16)                 setattr(vmobject, attr, new_a2)
1404: (12)             elif len(a2) > len(a1):
1405: (16)                 new_a1 = stretch_array_to_length(a1, len(a2))
1406: (16)                 setattr(self, attr, new_a1)
1407: (8)             return self
1408: (4)         def get_point_mobject(self, center: Point3D | None = None) ->
VectorizedPoint:
1409: (8)             if center is None:
1410: (12)                 center = self.get_center()
1411: (8)             point = VectorizedPoint(center)
1412: (8)             point.match_style(self)
1413: (8)             return point
1414: (4)         def interpolate_color(
1415: (8)             self, mobject1: VMobject, mobject2: VMobject, alpha: float
1416: (4)         ) -> None:
1417: (8)             attrs = [
1418: (12)                 "fill_rgbs",
1419: (12)                 "stroke_rgbs",
1420: (12)                 "background_stroke_rgbs",
1421: (12)                 "stroke_width",
1422: (12)                 "background_stroke_width",
1423: (12)                 "sheen_direction",
1424: (12)                 "sheen_factor",
1425: (8)
1426: (8)             ]
1427: (12)             for attr in attrs:
1428: (16)                 setattr(
1429: (16)                     self,
1430: (16)                     attr,
1431: (12)                     interpolate(getattr(mobject1, attr), getattr(mobject2, attr),
alpha),
1432: (12)                     )
1433: (16)                     if alpha == 1.0:
1434: (16)                         val = getattr(mobject2, attr)
1435: (20)                         if isinstance(val, np.ndarray):
1436: (16)                             val = val.copy()
1437: (4)                         setattr(self, attr, val)
1438: (8)             def pointwise_become_partial(
1439: (8)                 self,
1440: (8)                 vmobject: VMobject,
1441: (8)                 a: float,
1442: (8)                 b: float,
1443: (4)             ) -> Self:
1444: (8)                 """Given two bounds a and b, transforms the points of the self
vmobject into the points of the vmobject
control points of the bezier curves (anchors and handles)
passed as parameter with respect to the bounds. Points here stand for
control points of the bezier curves (anchors and handles)
Parameters
-----
vmobject
    The vmobject that will serve as a model.
a
    upper-bound.
b
    lower-bound
Returns
-----
:class:`VMobject`
    ``self``
"""
1445: (8)
1446: (8)
1447: (8)
1448: (12)
1449: (8)
1450: (12)
1451: (8)
1452: (12)
1453: (8)
1454: (8)
1455: (8)
1456: (12)
1457: (8)
1458: (8)
1459: (8)
1460: (12)
1461: (12)
1462: (8)
1463: (8)
1464: (8)
1465: (8)
assert isinstance(vmobject, VMobject)
if a <= 0 and b >= 1:
    self.set_points(vmobject.points)
    return self
bezier_quads = vmobject.get_cubic_bezier_tuples()
num_cubics = len(bezier_quads)
lower_index, lower_residue = integer_interpolate(0, num_cubics, a)
upper_index, upper_residue = integer_interpolate(0, num_cubics, b)

```

```

1466: (8)             self.clear_points()
1467: (8)             if num_cubics == 0:
1468: (12)                 return self
1469: (8)             if lower_index == upper_index:
1470: (12)                 self.append_points(
1471: (16)                     partial_bezier_points(
1472: (20)                         bezier_quads[lower_index],
1473: (20)                         lower_residue,
1474: (20)                         upper_residue,
1475: (16)                     ),
1476: (12)
1477: (8)             else:
1478: (12)                 self.append_points(
1479: (16)                     partial_bezier_points(bezier_quads[lower_index],
lower_residue, 1),
1480: (12)
1481: (12)                     )
1482: (16)                     for quad in bezier_quads[lower_index + 1 : upper_index]:
1483: (12)                         self.append_points(quad)
1484: (16)                     self.append_points(
1485: (12)                         partial_bezier_points(bezier_quads[upper_index], 0,
upper_residue),
1486: (12)
1487: (8)             return self
1488: (4)             def get_subcurve(self, a: float, b: float) -> Self:
1489: (8)                 """Returns the subcurve of the VMobject between the interval [a, b].
1490: (8)                 The curve is a VMobject itself.
1491: (8)                 Parameters
1492: (8)                 -----
1493: (12)                     a
1494: (8)                         The lower bound.
1495: (12)                     b
1496: (8)                         The upper bound.
1497: (8)                 Returns
1498: (8)                 -----
1499: (12)                     VMobject
1500: (8)                         The subcurve between of [a, b]
1501: (8)
1502: (12)             if self.is_closed() and a > b:
1503: (12)                 vmob = self.copy()
1504: (12)                 vmob.pointwise_become_partial(self, a, 1)
1505: (12)                 vmob2 = self.copy()
1506: (12)                 vmob2.pointwise_become_partial(self, 0, b)
1507: (12)                 vmob.append_vectorized_mobject(vmob2)
1508: (12)
1509: (12)             else:
1510: (8)                 vmob = self.copy()
1511: (4)                 vmob.pointwise_become_partial(self, a, b)
1512: (8)             return vmob
1513: (8)             def get_direction(self) -> Literal["CW", "CCW"]:
1514: (8)                 """Uses :func:`~.space_ops.shoelace_direction` to calculate the
1515: (8)                 direction.
1516: (8)                 Examples
1517: (8)                 -----
1518: (12)                     The default direction of a :class:`~.Circle` is counterclockwise::
1519: (12)                         >>> from manim import Circle
1520: (12)                         >>> Circle().get_direction()
1521: (12)                         'CCW'
1522: (8)                 Returns
1523: (8)                 -----
1524: (12)                     :class:`str`
1525: (8)                         Either ``"CW"`` or ``"CCW"``.
1526: (8)
1527: (4)             def reverse_direction(self) -> Self:
1528: (8)                 """Reverts the point direction by inverting the point order.
1529: (8)             Returns
1530: (8)
1531: (8)                 :class:`VMobject`
```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
   Returns self.

1532: (12)                                     Examples
1533: (8)
1534: (8)
1535: (8)
1536: (12)                                     -----
1537: (16)                                     .. manim:: ChangeOfDirection
1538: (20)                                     class ChangeOfDirection(Scene):
1539: (20)                                     def construct(self):
1540: (20)                                     ccw = RegularPolygon(5)
1541: (20)                                     ccw.shift(LEFT)
1542: (20)                                     cw = RegularPolygon(5)
1543: (20)                                     cw.shift(RIGHT).reverse_direction()
1544: (20)                                     self.play(Create(ccw), Create(cw),
1545: (8)                                       run_time=4)
1546: (8)                                     """
1547: (4)                                       self.points = self.points[::-1]
1548: (8)                                     return self
1549: (8)                                     def force_direction(self, target_direction: Literal["CW", "CCW"]) -> Self:
1550: (8)                                       """Makes sure that points are either directed clockwise or
1551: (8)                                       counterclockwise.
1552: (8)                                       Parameters
1553: (12)                                     target_direction
1554: (8)                                       Either ``"CW"`` or ``"CCW"``.
1555: (8)
1556: (12)                                     if target_direction not in ("CW", "CCW"):
1557: (8)                                       raise ValueError('Invalid input for force_direction. Use "CW" or
1558: (12)                                       "CCW"')
1559: (8)
1560: (0)                                     class VGroup(VMobject, metaclass=ConvertToOpenGL):
1561: (4)                                       """A group of vectorized mobjects.
1562: (4)                                       This can be used to group multiple :class:`~.VMobject` instances together
1563: (4)                                       in order to scale, move, ... them together.
1564: (4)                                     Notes
1565: (4)
1566: (4)                                       When adding the same mobject more than once, repetitions are ignored.
1567: (4)                                       Use :meth:`.Mobject.copy` to create a separate copy which can then
1568: (4)                                       be added to the group.
1569: (4)                                     Examples
1570: (4)
1571: (4)                                     To add :class:`~.VMobject`'s to a :class:`~.VGroup`, you can either use the
1572: (4)                                       :meth:`~.VGroup.add` method, or use the `+` and `+=` operators. Similarly,
you
1573: (4)                                       you can subtract elements of a VGroup via :meth:`~.VGroup.remove` method, or
1574: (4)                                       `--` and `-=` operators:
1575: (8)                                     >>> from manim import Triangle, Square, VGroup
1576: (8)                                     >>> vg = VGroup()
1577: (8)                                     >>> triangle, square = Triangle(), Square()
1578: (8)                                     >>> vg.add(triangle)
1579: (8)                                     VGroup(Triangle)
1580: (8)                                     >>> vg + square  # a new VGroup is constructed
1581: (8)                                     VGroup(Triangle, Square)
1582: (8)                                     >>> vg           # not modified
1583: (8)                                     VGroup(Triangle)
1584: (8)                                     >>> vg += square; vg  # modifies vg
1585: (8)                                     VGroup(Triangle, Square)
1586: (8)                                     >>> vg.remove(triangle)
1587: (8)                                     VGroup(Square)
1588: (8)                                     >>> vg - square; # a new VGroup is constructed
1589: (8)                                     VGroup()
1590: (8)                                     >>> vg           # not modified
1591: (8)                                     VGroup(Square)
1592: (8)                                     >>> vg -= square; vg # modifies vg
1593: (8)                                     VGroup()
1594: (4)                                     ..
1595: (8)                                     manim:: ArcShapeIris
1596: (8)                                     :save_last_frame:
1597: (12)                                     class ArcShapeIris(Scene):
1598: (16)                                       def construct(self):
1599: (16)   colors = [DARK_BROWN, BLUE_E, BLUE_D, BLUE_A, TEAL_B, GREEN_B,
```

```

YELLOW_E]
1599: (16)           radius = [1 + rad * 0.1 for rad in range(len(colors))]
1600: (16)           circles_group = VGroup()
1601: (16)           circles_group.add(*[Circle(radius=rad, stroke_width=10,
color=col)
1602: (36)                   for rad, col in zip(radius, colors)])
1603: (16)           self.add(circles_group)
1604: (4) """
1605: (4)     def __init__(self, *vmobjects, **kwargs):
1606: (8)         super().__init__(**kwargs)
1607: (8)         self.add(*vmobjects)
1608: (4)     def __repr__(self) -> str:
1609: (8)         return f'{self.__class__.__name__}({", ".join(str(mob) for mob in
self.submobjects)})'
1610: (4)
1611: (8)
1612: (12)
1613: (12)
1614: (8)
1615: (4)
1616: (8)
add them to submobjects
1617: (8)
1618: (8)
1619: (8)
1620: (12)
1621: (8)
1622: (8)
1623: (8)
1624: (8)
1625: (8)
1626: (8)
1627: (12)
1628: (8)
1629: (8)
1630: (8)
1631: (12)
1632: (16)
1633: (20)
1634: (20)
1635: (20)
1636: (20)
1637: (20)
1638: (20)
1639: (20)
1640: (20)
1641: (20)
1642: (20)
1643: (20)
1644: (24)
1645: (20)
1646: (20)
1647: (20)
1648: (24)
1649: (24)
1650: (20)
1651: (20)
1652: (24)
1653: (20)
1654: (20)
1655: (24)
1656: (20)
1657: (8)
1658: (8)
1659: (12)
1660: (16)
1661: (20)
type VMobject. "
1662: (20)

    for m in vmobjects:
        if not isinstance(m, (VMobject, OpenGLVMobject)):
            raise TypeError(
                f"All submobjects of {self.__class__.__name__} must be of
                f"Got {repr(m)} ({type(m).__name__}) instead. "
"""
def __str__(self) -> str:
    return (
        f"{self.__class__.__name__} of {len(self.submobjects)}"
        f"submobject{'s' if len(self.submobjects) > 0 else ''}"
    )
def add(self, *vmobjects: VMobject) -> Self:
    """Checks if all passed elements are an instance of VMobject and then
    Parameters
    -----
    vmobjects
        List of VMobject to add
    Returns
    -----
    :class:`VGroup`
    Raises
    -----
    TypeError
        If one element of the list is not an instance of VMobject
    Examples
    -----
    .. manim:: AddToVGroup
        class AddToVGroup(Scene):
            def construct(self):
                circle_red = Circle(color=RED)
                circle_green = Circle(color=GREEN)
                circle_blue = Circle(color=BLUE)
                circle_red.shift(LEFT)
                circle_blue.shift(RIGHT)
                gr = VGroup(circle_red, circle_green)
                gr2 = VGroup(circle_blue) # Constructor uses add directly
                self.add(gr,gr2)
                self.wait()
                gr += gr2 # Add group to another
                self.play(
                    gr.animate.shift(DOWN),
                )
                gr -= gr2 # Remove group
                self.play( # Animate groups separately
                    gr.animate.shift(LEFT),
                    gr2.animate.shift(UP),
                )
                self.play( #Animate groups without modification
                    (gr+gr2).animate.shift(RIGHT)
                )
                self.play( # Animate group without component
                    (gr-circle_red).animate.shift(RIGHT)
                )
"""
for m in vmobjects:
    if not isinstance(m, (VMobject, OpenGLVMobject)):
        raise TypeError(
            f"All submobjects of {self.__class__.__name__} must be of
            f"Got {repr(m)} ({type(m).__name__}) instead. "
"""

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
   "You can try using `Group` instead."
   )
   return super().add(*vmobjects)
def __add__(self, vmobject: VMobject) -> Self:
    return VGroup(*self.submobjects, vmobject)
def __iadd__(self, vmobject: VMobject) -> Self:
    return self.add(vmobject)
def __sub__(self, vmobject: VMobject) -> Self:
    copy = VGroup(*self.submobjects)
    copy.remove(vmobject)
    return copy
def __isub__(self, vmobject: VMobject) -> Self:
    return self.remove(vmobject)
def __setitem__(self, key: int, value: VMobject | Sequence[VMobject]) ->
None:
1677: (8)           """Override the [] operator for item assignment.
1678: (8)           Parameters
1679: (8)           -----
1680: (8)           key
1681: (12)          The index of the submobject to be assigned
1682: (8)           value
1683: (12)          The vmobject value to assign to the key
1684: (8)           Returns
1685: (8)           -----
1686: (8)           None
1687: (8)           Tests
1688: (8)           -----
1689: (8)           Check that item assignment does not raise error::
1690: (12)          >>> vgroup = VGroup(VMobject())
1691: (12)          >>> new_obj = VMobject()
1692: (12)          >>> vgroup[0] = new_obj
1693: (8)           """
1694: (8)           if not all(isinstance(m, (VMobject, OpenGLVMobject)) for m in value):
1695: (12)             raise TypeError("All submobjects must be of type VMobject")
1696: (8)           self.submobjects[key] = value
class VDict(VMobject, metaclass=ConvertToOpenGL):
1698: (4)           """A VGroup-like class, also offering submobject access by
1699: (4)           key, like a python dict
1700: (4)           Parameters
1701: (4)           -----
1702: (4)           mapping_or_iterable
1703: (12)          The parameter specifying the key-value mapping of keys and
mobjects.
1704: (4)           show_keys
1705: (12)             Whether to also display the key associated with
1706: (12)             the mobject. This might be useful when debugging,
1707: (12)             especially when there are a lot of mobjects in the
1708: (12)             :class:`VDict`. Defaults to False.
1709: (4)           kwargs
1710: (12)             Other arguments to be passed to `Mobject`.
1711: (4)           Attributes
1712: (4)           -----
1713: (4)           show_keys : :class:`bool`
1714: (12)             Whether to also display the key associated with
1715: (12)             the mobject. This might be useful when debugging,
1716: (12)             especially when there are a lot of mobjects in the
1717: (12)             :class:`VDict`. When displayed, the key is towards
1718: (12)             the left of the mobject.
1719: (12)             Defaults to False.
1720: (4)           submob_dict : :class:`dict`
1721: (12)             Is the actual python dictionary that is used to bind
1722: (12)             the keys to the mobjects.
1723: (4)           Examples
1724: (4)           -----
1725: (4)           .. manim:: ShapesWithVDict
1726: (8)             class ShapesWithVDict(Scene):
1727: (12)               def construct(self):
1728: (16)                 square = Square().set_color(RED)
1729: (16)                 circle = Circle().set_color(YELLOW).next_to(square, UP)

```

```

1730: (16)             pairs = [("s", square), ("c", circle)]
1731: (16)             my_dict = VDict(pairs, show_keys=True)
1732: (16)             self.play(Create(my_dict))
1733: (16)             self.wait()
1734: (16)             text = Tex("Some text").set_color(GREEN).next_to(square, DOWN)
1735: (16)             my_dict.add([("t", text)])
1736: (16)             self.wait()
1737: (16)             rect = Rectangle().next_to(text, DOWN)
1738: (16)             my_dict["r"] = rect
1739: (16)             my_dict["t"].set_color(PURPLE)
1740: (16)             self.play(my_dict["t"].animate.scale(3))
1741: (16)             self.wait()
1742: (16)             my_dict["t"] = Tex("Some other text").set_color(BLUE)
1743: (16)             self.wait()
1744: (16)             my_dict.remove("t")
1745: (16)             self.wait()
1746: (16)             self.play(Uncreate(my_dict["s"]))
1747: (16)             self.wait()
1748: (16)             self.play(FadeOut(my_dict["c"]))
1749: (16)             self.wait()
1750: (16)             self.play(FadeOut(my_dict["r"], shift=DOWN))
1751: (16)             self.wait()
1752: (16)             plain_dict = {
1753: (20)                 1: Integer(1).shift(DOWN),
1754: (20)                 2: Integer(2).shift(2 * DOWN),
1755: (20)                 3: Integer(3).shift(3 * DOWN),
1756: (16)             }
1757: (16)             vdict_from_plain_dict = VDict(plain_dict)
1758: (16)             vdict_from_plain_dict.shift(1.5 * (UP + LEFT))
1759: (16)             self.play(Create(vdict_from_plain_dict))
1760: (16)             vdict_using_zip = VDict(zip(["s", "c", "r"], [Square(),
Circle(), Rectangle()]))
1761: (16)             vdict_using_zip.shift(1.5 * RIGHT)
1762: (16)             self.play(Create(vdict_using_zip))
1763: (16)             self.wait()
1764: (4)             """
1765: (4)             def __init__(
1766: (8)                 self,
1767: (8)                 mapping_or_iterable: (
1768: (12)                     Mapping[Hashable, VMobject] | Iterable[tuple[Hashable, VMobject]]
1769: (8)                 ) = {},
1770: (8)                 show_keys: bool = False,
1771: (8)                 **kwargs,
1772: (4)             ) -> None:
1773: (8)                 super().__init__(**kwargs)
1774: (8)                 self.show_keys = show_keys
1775: (8)                 self.submob_dict = {}
1776: (8)                 self.add(mapping_or_iterable)
1777: (4)             def __repr__(self) -> str:
1778: (8)                 return f"{self.__class__.__name__}({repr(self.submob_dict)})"
1779: (4)             def add(
1780: (8)                 self,
1781: (8)                 mapping_or_iterable: (
1782: (12)                     Mapping[Hashable, VMobject] | Iterable[tuple[Hashable, VMobject]]
1783: (8)                 ),
1784: (4)             ) -> Self:
1785: (8)                 """Adds the key-value pairs to the :class:`VDict` object.
1786: (8)                 Also, it internally adds the value to the `subobjects` :class:`list`
1787: (8)                 of :class:`~.Mobject`, which is responsible for actual on-screen
display.
1788: (8)             Parameters
1789: (8)             -----
1790: (8)             mapping_or_iterable
1791: (12)                 The parameter specifying the key-value mapping of keys and
mobjects.
1792: (8)             Returns
1793: (8)             -----
1794: (8)             :class:`VDict`
1795: (12)                 Returns the :class:`VDict` object on which this method was called.

```

```

1796: (8)             Examples
1797: (8)             -----
1798: (8)             Normal usage:::
1799: (12)             square_obj = Square()
1800: (12)             my_dict.add([('s', square_obj)])
1801: (8)
1802: (8)
1803: (12)
1804: (8)
1805: (4)
1806: (8)
`key`              Also, it internally removes the mobject from the `submobjects` of :class:`~.Mobject`, (which is responsible for removing it from the
1807: (8)               screen)
:class:`list`       Parameters
1808: (8)
screen)            -----
1809: (8)
1810: (8)
1811: (8)
1812: (12)           key
1813: (8)             The key of the submobject to be removed.
1814: (8)             Returns
1815: (8)             -----
1816: (12)             :class:`VDict`
1817: (8)             Returns the :class:`VDict` object on which this method was called.
1818: (8)
1819: (8)
1820: (12)           Examples
1821: (8)
1822: (8)
1823: (12)           Normal usage:::
str(key))          my_dict.remove('square')
1824: (8)
1825: (8)
1826: (8)
1827: (4)             """
1828: (8)             Override the [] operator for item retrieval.
1829: (8)             Parameters
1830: (8)
1831: (8)
1832: (11)            key
1833: (8)             The key of the submobject to be accessed
1834: (8)             Returns
1835: (8)             -----
1836: (11)             :class:`VMobject`
1837: (8)             The submobject corresponding to the key `key`
1838: (8)
1839: (8)
1840: (11)           Examples
1841: (8)
1842: (8)
1843: (8)
1844: (4)             Normal usage:::
1845: (8)             self.play(Create(my_dict['s']))
1846: (8)
1847: (8)
1848: (8)
1849: (12)             """
1850: (8)             submob = self.submob_dict[key]
1851: (12)             return submob
1852: (8)
1853: (8)
1854: (8)
1855: (8)
1856: (8)
1857: (8)             def __setitem__(self, key: Hashable, value: VMobject) -> None:
1858: (12)             """
1859: (12)             """Override the [] operator for item assignment.
1860: (8)             Parameters
-----             key
-----             The key of the submobject to be assigned
value              The submobject to bind the key to
-----             Returns
-----             None
-----             Examples
-----             Normal usage:::
-----             square_obj = Square()
-----             my_dict['sq'] = square_obj
"""

```

```

1861: (8)             if key in self.submob_dict:
1862: (12)             self.remove(key)
1863: (8)             self.add([(key, value)])
1864: (4)         def __delitem__(self, key: Hashable):
1865: (8)             """Override the del operator for deleting an item.
1866: (8)             Parameters
1867: (8)             -----
1868: (8)             key
1869: (12)                 The key of the submoject to be deleted
1870: (8)             Returns
1871: (8)             -----
1872: (8)             None
1873: (8)         Examples
1874: (8)             -----
1875: (8)             :::
1876: (12)                 >>> from manim import *
1877: (12)                 >>> my_dict = VDict({'sq': Square()})
1878: (12)                 >>> 'sq' in my_dict
1879: (12)                     True
1880: (12)                 >>> del my_dict['sq']
1881: (12)                 >>> 'sq' in my_dict
1882: (12)                     False
1883: (8)         Notes
1884: (8)             -----
1885: (8)             Removing an item from a VDict does not remove that item from any Scene
1886: (8)             that the VDict is part of.
1887: (8)             """
1888: (8)             del self.submob_dict[key]
1889: (4)         def __contains__(self, key: Hashable):
1890: (8)             """Override the in operator.
1891: (8)             Parameters
1892: (8)             -----
1893: (8)             key
1894: (12)                 The key to check membership of.
1895: (8)             Returns
1896: (8)             -----
1897: (8)             :class:`bool`
1898: (8)         Examples
1899: (8)             -----
1900: (8)             :::
1901: (12)                 >>> from manim import *
1902: (12)                 >>> my_dict = VDict({'sq': Square()})
1903: (12)                 >>> 'sq' in my_dict
1904: (12)                     True
1905: (8)             """
1906: (8)             return key in self.submob_dict
1907: (4)         def get_all_submobjects(self) -> list[list]:
1908: (8)             """To get all the submobjects associated with a particular
1909: (8)             :class:`VDict` object
1910: (8)             Returns
1911: (8)             -----
1912: (12)             :class:`dict_values`  

1913: (8)                 All the submobjects associated with the :class:`VDict` object
1914: (8)             Examples
1915: (8)             -----
1916: (12)             Normal usage:::
1917: (16)                 for submob in my_dict.get_all_submobjects():
1918: (8)                     self.play(Create(submob))
1919: (8)             """
1920: (8)             submobjects = self.submob_dict.values()
1921: (4)         def add_key_value_pair(self, key: Hashable, value: VMobject) -> None:
1922: (8)             """A utility function used by :meth:`add` to add the key-value pair
1923: (8)             to :attr:`submob_dict`. Not really meant to be used externally.
1924: (8)             Parameters
1925: (8)             -----
1926: (8)             key
1927: (12)                 The key of the submobject to be added.
1928: (8)             value

```

```

1929: (12)           The mobject associated with the key
1930: (8)            Returns
1931: (8)            -----
1932: (8)            None
1933: (8)            Raises
1934: (8)            -----
1935: (8)            TypeError
1936: (12)           If the value is not an instance of VMobject
1937: (8)            Examples
1938: (8)            -----
1939: (8)            Normal usage:::
1940: (12)           square_obj = Square()
1941: (12)           self.add_key_value_pair('s', square_obj)
1942: (8)           """
1943: (8)           if not isinstance(value, (VMobject, OpenGLMobject)):
1944: (12)             raise TypeError("All submobjects must be of type VMobject")
1945: (8)             mob = value
1946: (8)             if self.show_keys:
1947: (12)               from manim.mobject.text.tex_mobject import Tex
1948: (12)               key_text = Tex(str(key)).next_to(value, LEFT)
1949: (12)               mob.add(key_text)
1950: (8)               self.submob_dict[key] = mob
1951: (8)               super().add(value)
1952: (0)             class VectorizedPoint(VMobject, metaclass=ConvertToOpenGL):
1953: (4)               def __init__(
1954: (8)                 self,
1955: (8)                 location: Point3D = ORIGIN,
1956: (8)                 color: ManimColor = BLACK,
1957: (8)                 fill_opacity: float = 0,
1958: (8)                 stroke_width: float = 0,
1959: (8)                 artificial_width: float = 0.01,
1960: (8)                 artificial_height: float = 0.01,
1961: (8)                 **kwargs,
1962: (4)               ) -> None:
1963: (8)                 self.artificial_width = artificial_width
1964: (8)                 self.artificial_height = artificial_height
1965: (8)                 super().__init__(
1966: (12)                   color=color,
1967: (12)                   fill_opacity=fill_opacity,
1968: (12)                   stroke_width=stroke_width,
1969: (12)                   **kwargs,
1970: (8)               )
1971: (8)                 self.set_points(np.array([location]))
1972: (4)             basecls = OpenGLMobject if config.renderer == RenderType.OPENGL else
VMobject
1973: (4)             @basecls.width.getter
1974: (4)             def width(self) -> float:
1975: (8)               return self.artificial_width
1976: (4)             @basecls.height.getter
1977: (4)             def height(self) -> float:
1978: (8)               return self.artificial_height
1979: (4)             def get_location(self) -> Point3D:
1980: (8)               return np.array(self.points[0])
1981: (4)             def set_location(self, new_loc: Point3D):
1982: (8)               self.set_points(np.array([new_loc]))
1983: (0)             class CurvesAsSubmobjects(VGroup):
1984: (4)               """Convert a curve's elements to submobjects.
1985: (4)               Examples
1986: (4)               -----
1987: (4)               .. manim:: LineGradientExample
1988: (8)                 :save_last_frame:
1989: (8)                 class LineGradientExample(Scene):
1990: (12)                   def construct(self):
1991: (16)                     curve = ParametricFunction(lambda t: [t, np.sin(t), 0],
t_range=[-PI, PI, 0.01], stroke_width=10)
1992: (16)                     new_curve = CurvesAsSubmobjects(curve)
1993: (16)                     new_curve.set_color_by_gradient(BLUE, RED)
1994: (16)                     self.add(new_curve.shift(UP), curve)
1995: (4)               """

```

```

1996: (4)             def __init__(self, vmobject: VMobject, **kwargs) -> None:
1997: (8)                 super().__init__(**kwargs)
1998: (8)                 tuples = vmobject.get_cubic_bezier_tuples()
1999: (8)                 for tup in tuples:
2000: (12)                     part = VMobject()
2001: (12)                     part.set_points(tup)
2002: (12)                     part.match_style(vmobject)
2003: (12)                     self.add(part)
2004: (4)             def point_from_proportion(self, alpha: float) -> Point3D:
2005: (8)                 """Gets the point at a proportion along the path of the
:class:`CurvesAsSubmobjects`.
2006: (8)             Parameters
2007: (8)                 -----
2008: (8)                 alpha
2009: (12)                 The proportion along the the path of the
:class:`CurvesAsSubmobjects`.
2010: (8)             Returns
2011: (8)                 -----
2012: (8)                 :class:`numpy.ndarray`
2013: (12)                 The point on the :class:`CurvesAsSubmobjects`.
2014: (8)             Raises
2015: (8)                 -----
2016: (8)                 :exc:`ValueError`
2017: (12)                 If ``alpha`` is not between 0 and 1.
2018: (8)                 :exc:`Exception`
2019: (12)                 If the :class:`CurvesAsSubmobjects` has no submobjects, or no
submobject has points.
2020: (8)             """
2021: (8)                 if alpha < 0 or alpha > 1:
2022: (12)                     raise ValueError(f"Alpha {alpha} not between 0 and 1.")
2023: (8)                 self._throw_error_if_no_submobjects()
2024: (8)                 submobs_with_pts = self._get_submobjects_with_points()
2025: (8)                 if alpha == 1:
2026: (12)                     return submobs_with_pts[-1].points[-1]
2027: (8)                 submobs_arc_lengths = tuple(
2028: (12)                     part.get_arc_length() for part in submobs_with_pts
2029: (8)                 )
2030: (8)                 total_length = sum(submobs_arc_lengths)
2031: (8)                 target_length = alpha * total_length
2032: (8)                 current_length = 0
2033: (8)                 for i, part in enumerate(submobs_with_pts):
2034: (12)                     part_length = submobs_arc_lengths[i]
2035: (12)                     if current_length + part_length >= target_length:
2036: (16)                         residue = (target_length - current_length) / part_length
2037: (16)                         return part.point_from_proportion(residue)
2038: (12)                     current_length += part_length
2039: (4)             def _throw_error_if_no_submobjects(self):
2040: (8)                 if len(self.submobjects) == 0:
2041: (12)                     caller_name = sys._getframe(1).f_code.co_name
2042: (12)                     raise Exception(
2043: (16)                         f"Cannot call CurvesAsSubmobjects. {caller_name} for a
CurvesAsSubmobject with no submobjects"
2044: (12)                     )
2045: (4)             def _get_submobjects_with_points(self):
2046: (8)                 submobs_with_pts = tuple(
2047: (12)                     part for part in self.submobjects if len(part.points) > 0
2048: (8)                 )
2049: (8)                 if len(submobs_with_pts) == 0:
2050: (12)                     caller_name = sys._getframe(1).f_code.co_name
2051: (12)                     raise Exception(
2052: (16)                         f"Cannot call CurvesAsSubmobjects. {caller_name} for a
CurvesAsSubmobject whose submobjects have no points"
2053: (12)                     )
2054: (8)                     return submobs_with_pts
2055: (0)             class DashedVMobject(VMobject, metaclass=ConvertToOpenGL):
2056: (4)                 """A :class:`VMobject` composed of dashes instead of lines.
2057: (4)             Parameters
2058: (4)                 -----
2059: (8)                     vmobject

```

```

2060: (12)           The object that will get dashed
2061: (8)
2062: (12)           num_dashes
2063: (8)             Number of dashes to add.
2064: (12)           dashed_ratio
2065: (8)             Ratio of dash to empty space.
2066: (12)           dash_offset
2067: (12)             Shifts the starting point of dashes along the
2068: (8)               path. Value 1 shifts by one full dash length.
2069: (12)           equal_lengths
2070: (12)             If ``True``, dashes will be (approximately) equally long.
2071: (12)             If ``False``, dashes will be split evenly in the curve's
2072: (4)              input t variable (legacy behavior).
2073: (4)
2074: (4)           Examples
2075: (8)
2076: (8)
2077: (12)
2078: (16)
2079: (16)
2080: (16)
2081: (20)
num_dashes=dashes)
2082: (20)
2083: (16)
2084: (16)
2085: (20)
2086: (24)
2087: (20)
2088: (20)
2089: (16)
2090: (16)
DashedVMOBJECT(func1,num_dashes=6,equal_lengths=True)
2091: (16)
equal_lengths=False)
2092: (16)
2093: (16)
2094: (16)
2095: (16)
2096: (16)
bottom_row).arrange(DOWN, buff=1)
2097: (16)
self.add(everything)
2098: (4)
2099: (4)           """
2100: (8)             def __init__(
2101: (8)               self,
2102: (8)                 vMobject: VMOBJECT,
2103: (8)                 num_dashes: int = 15,
2104: (8)                 dashed_ratio: float = 0.5,
2105: (8)                 dash_offset: float = 0,
2106: (8)                 color: ManimColor = WHITE,
2107: (8)                 equal_lengths: bool = True,
2108: (4)                   **kwargs,
2109: (8)             ) -> None:
2110: (8)               self.dashed_ratio = dashed_ratio
2111: (8)               self.num_dashes = num_dashes
2112: (8)               super().__init__(color=color, **kwargs)
2113: (8)               r = self.dashed_ratio
2114: (8)               n = self.num_dashes
2115: (12)               if n > 0:
2116: (12)                 dash_len = r / n
2117: (16)                 if vMobject.is_closed():
2118: (12)                   void_len = (1 - r) / n
2119: (16)                 else:
2120: (20)                   if n == 1:
2121: (16)                     void_len = 1 - r
2122: (20)                   else:
2123: (12)                     void_len = (1 - r) / (n - 1)
2124: (12)               period = dash_len + void_len
phase_shift = (dash_offset % 1) * period

```

```

2125: (12)             if vmobject.is_closed():
2126: (16)                 pattern_len = 1
2127: (12)             else:
2128: (16)                 pattern_len = 1 + void_len
2129: (12)             dash_starts = [((i * period + phase_shift) % pattern_len) for i in
range(n)]
2130: (12)             dash_ends = [
2131: (16)                 ((i * period + dash_len + phase_shift) % pattern_len) for i in
range(n)
2132: (12)             ]
2133: (12)             if not vmobject.is_closed():
2134: (16)                 if dash_ends[-1] > 1 and dash_starts[-1] > 1:
2135: (20)                     dash_ends.pop()
2136: (20)                     dash_starts.pop()
2137: (16)                 elif dash_ends[-1] < dash_len: # if it overflowed
2138: (20)                     if (
2139: (24)                         dash_starts[-1] < 1
2140: (20)                     ): # if the beginning of the piece is still in range
2141: (24)                         dash_starts.append(0)
2142: (24)                         dash_ends.append(dash_ends[-1])
2143: (24)                         dash_ends[-2] = 1
2144: (20)                     else:
2145: (24)                         dash_starts[-1] = 0
2146: (16)                 elif dash_starts[-1] > (1 - dash_len):
2147: (20)                     dash_ends[-1] = 1
2148: (12)             if equal_lengths:
2149: (16)                 norms = np.array(0)
2150: (16)                 for k in range(vmobject.get_num_curves()):
2151: (20)                     norms = np.append(norms,
vmobject.get_nth_curve_length_pieces(k))
2152: (16)                     length_vals = np.cumsum(norms)
2153: (16)                     ref_points = np.linspace(0, 1, length_vals.size)
2154: (16)                     curve_length = length_vals[-1]
2155: (16)                     self.add(
2156: (20)                         *
2157: (24)                             vmobject.get_subcurve(
2158: (28)                               np.interp(
2159: (32)                                 dash_starts[i] * curve_length,
2160: (32)                                 length_vals,
2161: (32)                                 ref_points,
2162: (28)
2163: (28)
2164: (32)
2165: (32)
2166: (32)
2167: (28)
2168: (24)
2169: (24)
2170: (20)
2171: (16)
2172: (12)
2173: (16)
2174: (20)
2175: (24)
2176: (28)
2177: (28)
2178: (24)
2179: (24)
2180: (20)
2181: (16)
2182: (8)
2183: (12)
2184: (8)
2185: (12)
-----
```

File 98 - utils.py:

```

1: (0)         """Utilities for working with mobjects."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = [
4: (4)             "get_mobject_class",
5: (4)             "get_point_mobject_class",
6: (4)             "get_vectorized_mobject_class",
7: (0)
8: (0)         from ..config import config
9: (0)         from ..constants import RendererType
10: (0)        from .mobject import Mobject
11: (0)        from .opengl.opengl_mobject import OpenGLMobject
12: (0)        from .opengl.opengl_point_cloud_mobject import OpenGLPobject
13: (0)        from .opengl.opengl_vectorized_mobject import OpenGLVobject
14: (0)        from .types.point_cloud_mobject import Pobject
15: (0)        from .types.vectorized_mobject import Vobject
16: (0)
17: (4)        def get_mobject_class() -> type:
18: (4)            """Gets the base mobject class, depending on the currently active
19: (4)            renderer.
20: (4)            .. NOTE::
21: (4)                This method is intended to be used in the code base of Manim itself
22: (4)                or in plugins where code should work independent of the selected
23: (4)                renderer.
24: (4)            Examples
25: (4)            -----
26: (4)            The function has to be explicitly imported. We test that
27: (4)            the name of the returned class is one of the known mobject
28: (4)            base classes:
29: (8)                >>> from manim.mobject.utils import get_mobject_class
30: (8)                >>> get_mobject_class().__name__ in ['Mobject', 'OpenGLMobject']
31: (8)                True
32: (4)
33: (4)            if config.renderer == RendererType.CAIRO:
34: (4)                return Mobject
35: (4)            if config.renderer == RendererType.OPENGL:
36: (4)                return OpenGLMobject
37: (4)            raise NotImplementedError(
38: (4)                "Base mobjects are not implemented for the active renderer."
39: (4)
40: (4)        def get_vectorized_mobject_class() -> type:
41: (4)            """Gets the vectorized mobject class, depending on the currently
42: (4)            active renderer.
43: (4)            .. NOTE::
44: (4)                This method is intended to be used in the code base of Manim itself
45: (4)                or in plugins where code should work independent of the selected
46: (4)                renderer.
47: (4)            Examples
48: (4)            -----
49: (4)            The function has to be explicitly imported. We test that
50: (4)            the name of the returned class is one of the known mobject
51: (4)            base classes:
52: (8)                >>> from manim.mobject.utils import get_vectorized_mobject_class
53: (8)                >>> get_vectorized_mobject_class().__name__ in ['Vobject',
54: (8)                    'OpenGLVobject']
55: (8)
56: (4)
57: (4)            if config.renderer == RendererType.CAIRO:
58: (4)                return Vobject
59: (4)            if config.renderer == RendererType.OPENGL:
60: (4)                return OpenGLVobject
61: (4)            raise NotImplementedError(
62: (4)                "Vectorized mobjects are not implemented for the active renderer."
63: (4)
64: (4)        def get_point_mobject_class() -> type:
65: (4)            """Gets the point cloud mobject class, depending on the currently
66: (4)            active renderer.
67: (4)            .. NOTE::
68: (4)                This method is intended to be used in the code base of Manim itself
69: (4)                or in plugins where code should work independent of the selected
70: (4)                renderer.

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
68: (4)          Examples
69: (4)
70: (4)          -----
71: (4)          The function has to be explicitly imported. We test that
72: (4)          the name of the returned class is one of the known mobject
73: (8)          base classes::
74: (8)          >>> from manim.mobject.utils import get_point_mobject_class
75: (8)          >>> get_point_mobject_class().__name__ in ['PMobject',
76: (8)          'OpenGLMobject']
77: (4)          True
78: (4)
79: (4)          """
80: (8)          if config.renderer == RendererType.CAIRO:
81: (8)              return PMobject
82: (8)          if config.renderer == RendererType.OPENGL:
83: (8)              return OpenGLMobject
84: (4)          raise NotImplementedError(
85: (8)              "Point cloud mobjects are not implemented for the active renderer."
86: (4)      )
-----
```

## File 99 - shader.py:

```
1: (0)          from __future__ import annotations
2: (0)          import inspect
3: (0)          import re
4: (0)          import textwrap
5: (0)          from pathlib import Path
6: (0)          import moderngl
7: (0)          import numpy as np
8: (0)          from .. import config
9: (0)          from ..utils import opengl
10: (0)         SHADER_FOLDER = Path(__file__).parent / "shaders"
11: (0)         shader_program_cache: dict = {}
12: (0)         file_path_to_code_map: dict = {}
13: (0)         __all__ = [
14: (4)             "Object3D",
15: (4)             "Mesh",
16: (4)             "Shader",
17: (4)             "FullScreenQuad",
18: (0)         ]
19: (0)         def get_shader_code_from_file(file_path: Path) -> str:
20: (4)             if file_path in file_path_to_code_map:
21: (8)                 return file_path_to_code_map[file_path]
22: (4)             source = file_path.read_text()
23: (4)             include_lines = re.finditer(
24: (8)                 r"^\#include (?P<include_path>\.*\.glsl)$",
25: (8)                 source,
26: (8)                 flags=re.MULTILINE,
27: (4)             )
28: (4)             for match in include_lines:
29: (8)                 include_path = match.group("include_path")
30: (8)                 included_code = get_shader_code_from_file(
31: (12)                     file_path.parent / include_path,
32: (8)                 )
33: (8)                 source = source.replace(match.group(0), included_code)
34: (4)             file_path_to_code_map[file_path] = source
35: (4)             return source
36: (0)         def filter_attributes(unfiltered_attributes, attributes):
37: (4)             filtered_attributes_dtype = []
38: (4)             for i, dtype_name in enumerate(unfiltered_attributes.dtype.names):
39: (8)                 if dtype_name in attributes:
40: (12)                     filtered_attributes_dtype.append(
41: (16)                         (
42: (20)                             dtype_name,
43: (20)                             unfiltered_attributes.dtype[i].subtype[0].str,
44: (20)                             unfiltered_attributes.dtype[i].shape,
45: (16)                         ),
46: (12)                     )
47: (4)             filtered_attributes = np.zeros(
```

```

48: (8)                 unfiltered_attributes[unfiltered_attributes.dtype.names[0]].shape[0],
49: (8)                 dtype=filtered_attributes_dtype,
50: (4)             )
51: (4)             for dtype_name in unfiltered_attributes.dtype.names:
52: (8)                 if dtype_name in attributes:
53: (12)                     filtered_attributes[dtype_name] =
unfiltered_attributes[dtype_name]
54: (4)                     return filtered_attributes
55: (0)             class Object3D:
56: (4)                 def __init__(self, *children):
57: (8)                     self.model_matrix = np.eye(4)
58: (8)                     self.normal_matrix = np.eye(4)
59: (8)                     self.children = []
60: (8)                     self.parent = None
61: (8)                     self.add(*children)
62: (8)                     self.init_updaters()
63: (4)                 def interpolate(self, start, end, alpha, _):
64: (8)                     self.model_matrix = (1 - alpha) * start.model_matrix + alpha *
end.model_matrix
65: (8)
66: (12)             self.normal_matrix = (
67: (8)                 1 - alpha
68: (4)             ) * start.normal_matrix + alpha * end.normal_matrix
69: (8)             def single_copy(self):
70: (8)                 copy = Object3D()
71: (8)                 copy.model_matrix = self.model_matrix.copy()
72: (8)                 copy.normal_matrix = self.normal_matrix.copy()
73: (8)                 return copy
74: (4)             def copy(self):
75: (8)                 node_to_copy = {}
76: (8)                 bfs = [self]
77: (8)                 while bfs:
78: (12)                     node = bfs.pop(0)
79: (12)                     bfs.extend(node.children)
80: (12)                     node_copy = node.single_copy()
81: (12)                     node_to_copy[node] = node_copy
82: (12)                     if node.parent is not None and node is not self:
83: (16)                         node_to_copy[node.parent].add(node_copy)
84: (8)             return node_to_copy[self]
85: (4)             def add(self, *children):
86: (8)                 for child in children:
87: (12)                     if child.parent is not None:
88: (16)                         raise Exception(
89: (20)                             "Attempt to add child that's already added to another
Object3D",
90: (16)                         )
91: (8)                     self.remove(*children, current_children_only=False)
92: (8)                     self.children.extend(children)
93: (12)                     for child in children:
94: (8)                         child.parent = self
95: (4)             def remove(self, *children, current_children_only=True):
96: (8)                 if current_children_only:
97: (12)                     for child in children:
98: (16)                         if child.parent != self:
99: (20)                             raise Exception(
100: (24)                                 "Attempt to remove child that isn't added to this
Object3D",
101: (20)                                 )
102: (8)                     self.children = list(filter(lambda child: child not in children,
103: (12)                         for child in children:
104: (16)                             child.parent = None
105: (8)                         def get_position(self):
106: (4)                             return self.model_matrix[:, 3][:3]
107: (8)                         def set_position(self, position):
108: (12)                             self.model_matrix[:, 3][:3] = position
109: (8)                             return self
110: (8)                         def get_meshes(self):
111: (12)                             dfs = [self]

```

```

112: (12)             parent = dfs.pop()
113: (12)             if isinstance(parent, Mesh):
114: (16)                 yield parent
115: (12)             dfs.extend(parent.children)
116: (4)             def get_family(self):
117: (8)                 dfs = [self]
118: (8)                 while dfs:
119: (12)                     parent = dfs.pop()
120: (12)                     yield parent
121: (12)                     dfs.extend(parent.children)
122: (4)             def align_data_and_family(self, _):
123: (8)                 pass
124: (4)             def hierarchical_model_matrix(self):
125: (8)                 if self.parent is None:
126: (12)                     return self.model_matrix
127: (8)                 model_matrices = [self.model_matrix]
128: (8)                 current_object = self
129: (8)                 while current_object.parent is not None:
130: (12)                     model_matrices.append(current_object.parent.model_matrix)
131: (12)                     current_object = current_object.parent
132: (8)                 return np.linalg.multi_dot(list(reversed(model_matrices)))
133: (4)             def hierarchical_normal_matrix(self):
134: (8)                 if self.parent is None:
135: (12)                     return self.normal_matrix[:3, :3]
136: (8)                 normal_matrices = [self.normal_matrix]
137: (8)                 current_object = self
138: (8)                 while current_object.parent is not None:
139: (12)                     normal_matrices.append(current_object.parent.model_matrix)
140: (12)                     current_object = current_object.parent
141: (8)                 return np.linalg.multi_dot(list(reversed(normal_matrices)))[::3, ::3]
142: (4)             def init_updaters(self):
143: (8)                 self.time_based_updaters = []
144: (8)                 self.non_time_updaters = []
145: (8)                 self.has_updaters = False
146: (8)                 self.updating_suspended = False
147: (4)             def update(self, dt=0):
148: (8)                 if not self.has_updaters or self.updating_suspended:
149: (12)                     return self
150: (8)                 for updater in self.time_based_updaters:
151: (12)                     updater(self, dt)
152: (8)                 for updater in self.non_time_updaters:
153: (12)                     updater(self)
154: (8)                 return self
155: (4)             def get_time_based_updaters(self):
156: (8)                 return self.time_based_updaters
157: (4)             def has_time_based_updater(self):
158: (8)                 return len(self.time_based_updaters) > 0
159: (4)             def get_updaters(self):
160: (8)                 return self.time_based_updaters + self.non_time_updaters
161: (4)             def add_updater(self, update_function, index=None, call_updater=True):
162: (8)                 if "dt" in inspect.signature(update_function).parameters:
163: (12)                     updater_list = self.time_based_updaters
164: (8)                 else:
165: (12)                     updater_list = self.non_time_updaters
166: (8)                 if index is None:
167: (12)                     updater_list.append(update_function)
168: (8)                 else:
169: (12)                     updater_list.insert(index, update_function)
170: (8)                 self.refresh_has_updater_status()
171: (8)                 if call_updater:
172: (12)                     self.update()
173: (8)                 return self
174: (4)             def remove_updater(self, update_function):
175: (8)                 for updater_list in [self.time_based_updaters,
176: (12)                     self.non_time_updaters]:
177: (16)                     while update_function in updater_list:
178: (16)                         updater_list.remove(update_function)
179: (8)                     self.refresh_has_updater_status()
179: (8)                     return self

```

```

180: (4)             def clear_updaters(self):
181: (8)                 self.time_based_updaters = []
182: (8)                 self.non_time_updaters = []
183: (8)                 self.refresh_has_updater_status()
184: (8)                 return self
185: (4)             def match_updaters(self, mobject):
186: (8)                 self.clear_updaters()
187: (8)                 for updater in mobject.get_updaters():
188: (12)                     self.add_updater(updater)
189: (8)                 return self
190: (4)             def suspend_updating(self):
191: (8)                 self.updating_suspended = True
192: (8)                 return self
193: (4)             def resume_updating(self, call_updater=True):
194: (8)                 self.updating_suspended = False
195: (8)                 if call_updater:
196: (12)                     self.update(dt=0)
197: (8)                 return self
198: (4)             def refresh_has_updater_status(self):
199: (8)                 self.has_updaters = len(self.get_updaters()) > 0
200: (8)                 return self
201: (0)             class Mesh(Object3D):
202: (4)                 def __init__(self,
203: (8)                         shader=None,
204: (8)                         attributes=None,
205: (8)                         geometry=None,
206: (8)                         material=None,
207: (8)                         indices=None,
208: (8)                         use_depth_test=True,
209: (8)                         primitive=moderngl.TRIANGLES,
210: (8)                         ):
211: (4)                     super().__init__()
212: (8)                     if shader is not None and attributes is not None:
213: (12)                         self.shader = shader
214: (12)                         self.attributes = attributes
215: (12)                         self.indices = indices
216: (12)                     elif geometry is not None and material is not None:
217: (12)                         self.shader = material
218: (12)                         self.attributes = geometry.attributes
219: (12)                         self.indices = geometry.index
220: (12)                     else:
221: (8)                         raise Exception(
222: (12)                             "Mesh requires either attributes and a Shader or a Geometry
223: (16) and a "
224: (16)                             "Material",
225: (12)                         )
226: (8)                     self.use_depth_test = use_depth_test
227: (8)                     self.primitive = primitive
228: (8)                     self.skip_render = False
229: (8)                     self.init_updaters()
230: (4)             def single_copy(self):
231: (8)                 copy = Mesh(
232: (12)                     attributes=self.attributes.copy(),
233: (12)                     shader=self.shader,
234: (12)                     indices=self.indices.copy() if self.indices is not None else None,
235: (12)                     use_depth_test=self.use_depth_test,
236: (12)                     primitive=self.primitive,
237: (8)                 )
238: (8)                 copy.skip_render = self.skip_render
239: (8)                 copy.model_matrix = self.model_matrix.copy()
240: (8)                 copy.normal_matrix = self.normal_matrix.copy()
241: (8)                 return copy
242: (4)             def set_uniforms(self, renderer):
243: (8)                 self.shader.set_uniform(
244: (12)                     "u_model_matrix",
245: (12)                     opengl.matrix_to_shader_input(self.model_matrix),
246: (8)                 )
247: (8)                 self.shader.set_uniform("u_view_matrix",

```

```

renderer.camera.formatted_view_matrix)
248: (8)           self.shader.set_uniform(
249: (12)             "u_projection_matrix",
250: (12)             renderer.camera.projection_matrix,
251: (8)         )
252: (4)     def render(self):
253: (8)       if self.skip_render:
254: (12)         return
255: (8)       if self.use_depth_test:
256: (12)         self.shader.context.enable(moderngl.DEPTH_TEST)
257: (8)     else:
258: (12)       self.shader.context.disable(moderngl.DEPTH_TEST)
259: (8)   from moderngl import Attribute
260: (8)   shader_attributes = []
261: (8)   for k, v in self.shader.shader_program._members.items():
262: (12)     if isinstance(v, Attribute):
263: (16)       shader_attributes.append(k)
264: (8)   shader_attributes = filter_attributes(self.attributes,
265: (8)     shader_attributes)
266: (8)   self.shader.context.buffer(shader_attributes.tobytes())
267: (12)     if self.indices is None:
268: (8)       index_buffer_object = None
269: (12)     else:
270: (12)       vert_index_data = self.indices.astype("i4").tobytes()
271: (16)       if vert_index_data:
272: (8)         index_buffer_object =
273: (12)         self.shader.context.simple_vertex_array(
274: (12)           self.shader.shader_program,
275: (12)           vertex_buffer_object,
276: (12)           *shader_attributes.dtype.names,
277: (12)           index_buffer=index_buffer_object,
278: (12)         )
279: (8)       vertex_array_object.render(self.primitive)
280: (8)       vertex_buffer_object.release()
281: (8)       vertex_array_object.release()
282: (8)       if index_buffer_object is not None:
283: (8)         index_buffer_object.release()
284: (12)   class Shader:
285: (0)     def __init__(
286: (4)       self,
287: (8)       context,
288: (8)       name=None,
289: (8)       source=None,
290: (8)     ):
291: (4)       global shader_program_cache
292: (8)       self.context = context
293: (8)       self.name = name
294: (8)       if (
295: (12)         self.name in shader_program_cache
296: (12)         and shader_program_cache[self.name].ctx == self.context
297: (12)       ):
298: (8)         self.shader_program = shader_program_cache[self.name]
299: (12)       elif source is not None:
300: (8)         self.shader_program = context.program(**source)
301: (12)       else:
302: (8)         source_dict = {}
303: (12)         source_dict_key = {
304: (12)           "vert": "vertex_shader",
305: (16)           "frag": "fragment_shader",
306: (16)           "geom": "geometry_shader",
307: (16)         }
308: (12)         shader_folder = SHADER_FOLDER / name
309: (12)         for shader_file in shader_folder.iterdir():
310: (12)           shader_file_path = shader_folder / shader_file
311: (16)           shader_source = get_shader_code_from_file(shader_file_path)
312: (16)

```

```

313: (16)                     source_dict[source_dict_key[shader_file_path.stem]] =
 shader_source
314: (12)             self.shader_program = context.program(**source_dict)
315: (8)             if name is not None and name not in shader_program_cache:
316: (12)                 shader_program_cache[self.name] = self.shader_program
317: (4)             def set_uniform(self, name, value):
318: (8)                 try:
319: (12)                     self.shader_program[name] = value
320: (8)                 except KeyError:
321: (12)                     pass
322: (0)             class FullScreenQuad(Mesh):
323: (4)                 def __init__(self,
324: (8)                     context,
325: (8)                     fragment_shader_source=None,
326: (8)                     fragment_shader_name=None,
327: (8)                 ):
328: (4)                     if fragment_shader_source is None and fragment_shader_name is None:
329: (8)                         raise Exception("Must either pass shader name or shader source.")
330: (12)                     if fragment_shader_name is not None:
331: (8)                         shader_file_path = SHADER_FOLDER / f"{fragment_shader_name}.frag"
332: (12)                         fragment_shader_source =
333: (12)                         get_shader_code_from_file(shader_file_path)
334: (8)                         elif fragment_shader_source is not None:
335: (12)                             fragment_shader_source =
textwrap.dedent(fragment_shader_source.lstrip())
336: (8)                         shader = Shader(
337: (12)                             context,
338: (12)                             source={
339: (16)                                 "vertex_shader": """
340: (16)                                     in vec4 in_vert;
341: (16)                                     uniform mat4 u_model_view_matrix;
342: (16)                                     uniform mat4 u_projection_matrix;
343: (16)                                     void main() {{
344: (20)   vec4 camera_space_vertex = u_model_view_matrix * in_vert;
345: (20)   vec4 clip_space_vertex = u_projection_matrix *
camera_space_vertex;
346: (20)   gl_Position = clip_space_vertex;
347: (16)                                     }}
348: (16)                                 """,
349: (16)                                 "fragment_shader": fragment_shader_source,
350: (12)                             },
351: (8)                         )
352: (8)                         attributes = np.zeros(6, dtype=[("in_vert", np.float32, (4,))])
353: (8)                         attributes["in_vert"] = np.array(
354: (12)                             [
355: (16)                                 [-config["frame_x_radius"], -config["frame_y_radius"], 0, 1],
356: (16)                                 [-config["frame_x_radius"], config["frame_y_radius"], 0, 1],
357: (16)                                 [config["frame_x_radius"], config["frame_y_radius"], 0, 1],
358: (16)                                 [-config["frame_x_radius"], -config["frame_y_radius"], 0, 1],
359: (16)                                 [config["frame_x_radius"], -config["frame_y_radius"], 0, 1],
360: (16)                                 [config["frame_x_radius"], config["frame_y_radius"], 0, 1],
361: (12)                             ],
362: (8)                         )
363: (8)                         shader.set_uniform("u_model_view_matrix", opengl.view_matrix())
364: (8)                         shader.set_uniform(
365: (12)                             "u_projection_matrix",
366: (12)                             opengl.orthographic_projection_matrix(),
367: (8)                         )
368: (8)                         super().__init__(shader, attributes)
369: (4)             def render(self):
370: (8)                 super().render()

```

-----  
File 100 - \_\_init\_\_.py:

```

1: (0)             from __future__ import annotations
2: (0)             try:

```

```

3: (4)          from dearpygui import dearpygui as dpg
4: (0)          except ImportError:
5: (4)          pass
6: (0)          from manim.mobject.opengl.dot_cloud import *
7: (0)          from manim.mobject.opengl.opengl_image_mobject import *
8: (0)          from manim.mobject.opengl.opengl_mobject import *
9: (0)          from manim.mobject.opengl.opengl_point_cloud_mobject import *
10: (0)         from manim.mobject.opengl.opengl_surface import *
11: (0)         from manim.mobject.opengl.opengl_three_dimensions import *
12: (0)         from manim.mobject.opengl.opengl_vectorized_mobject import *
13: (0)         from ..renderer.shader import *
14: (0)         from ..utils.opengl import *

```

-----

## File 101 - \_\_init\_\_.py:

```

1: (0)          from __future__ import annotations
2: (0)          from manim import config, logger
3: (0)          from .plugins_flags import get_plugins, list_plugins
4: (0)          __all__ = [
5: (4)          "get_plugins",
6: (4)          "list_plugins",
7: (0)
8: (0)          requested_plugins: set[str] = set(config["plugins"])
9: (0)          missing_plugins = requested_plugins - set(get_plugins().keys())
10: (0)         if missing_plugins:
11: (4)             logger.warning("Missing Plugins: %s", missing_plugins)

```

-----

## File 102 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 103 - vector\_field.py:

```

1: (0)          """Mobjects representing vector fields."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = [
4: (4)          "VectorField",
5: (4)          "ArrowVectorField",
6: (4)          "StreamLines",
7: (0)
8: (0)          import itertools as it
9: (0)          import random
10: (0)         from math import ceil, floor
11: (0)         from typing import Callable, Iterable, Sequence
12: (0)         import numpy as np
13: (0)         from PIL import Image
14: (0)         from manim.animation.updaters.update import UpdateFromAlphaFunc
15: (0)         from manim.mobject.geometry.line import Vector
16: (0)         from manim.mobject.graphing.coordinate_systems import CoordinateSystem
17: (0)         from .. import config
18: (0)         from ..animation.composition import AnimationGroup, Succession
19: (0)         from ..animation.creation import Create
20: (0)         from ..animation.indication import ShowPassingFlash
21: (0)         from ..constants import OUT, RIGHT, UP, RendererType
22: (0)         from ..mobject.mobject import Mobject
23: (0)         from ..mobject.types.vectorized_mobject import VGroup
24: (0)         from ..mobject.utils import get_vectorized_mobject_class
25: (0)         from ..utils.bezier import interpolate, inverse_interpolate
26: (0)         from ..utils.color import (
27: (4)             BLUE_E,
28: (4)             GREEN,
29: (4)             RED,
30: (4)             YELLOW,

```

```

31: (4)             ManimColor,
32: (4)             ParsableManimColor,
33: (4)             color_to_rgb,
34: (4)             rgb_to_color,
35: (0)
36: (0)         from ..utils.rate_functions import ease_out_sine, linear
37: (0)         from ..utils.simple_functions import sigmoid
38: (0)         DEFAULT_SCALAR_FIELD_COLORS: list = [BLUE_E, GREEN, YELLOW, RED]
39: (0)     class VectorField(VGroup):
40: (4)             """A vector field.
41: (4)             Vector fields are based on a function defining a vector at every position.
42: (4)             This class does by default not include any visible elements but provides
43: (4)             methods to move other :class:`~.Mobject`'s along the vector field.
44: (4)             Parameters
45: (4)             -----
46: (4)             func
47: (8)                 The function defining the rate of change at every position of the
`VectorField`.
48: (4)             color
49: (8)                 The color of the vector field. If set, position-specific coloring is
disabled.
50: (4)             color_scheme
51: (8)                 A function mapping a vector to a single value. This value gives the
position in the color gradient defined using `min_color_scheme_value`, `max_color_scheme_value`
and `colors`.
52: (4)             min_color_scheme_value
53: (8)                 The value of the color_scheme function to be mapped to the first color
in `colors`. Lower values also result in the first color of the gradient.
54: (4)             max_color_scheme_value
55: (8)                 The value of the color_scheme function to be mapped to the last color
in `colors`. Higher values also result in the last color of the gradient.
56: (4)             colors
57: (8)                 The colors defining the color gradient of the vector field.
58: (4)             kwargs
59: (8)                 Additional arguments to be passed to the :class:`~.VGroup` constructor
"""
60: (4)
61: (4)     def __init__(
62: (8)             self,
63: (8)             func: Callable[[np.ndarray], np.ndarray],
64: (8)             color: ParsableManimColor | None = None,
65: (8)             color_scheme: Callable[[np.ndarray], float] | None = None,
66: (8)             min_color_scheme_value: float = 0,
67: (8)             max_color_scheme_value: float = 2,
68: (8)             colors: Sequence[ParsableManimColor] = DEFAULT_SCALAR_FIELD_COLORS,
69: (8)             **kwargs,
70: (4)     ):
71: (8)             super().__init__(**kwargs)
72: (8)             self.func = func
73: (8)             if color is None:
74: (12)                 self.single_color = False
75: (12)                 if color_scheme is None:
76: (16)                     def color_scheme(p):
77: (20)                         return np.linalg.norm(p)
78: (12)                     self.color_scheme = color_scheme # TODO maybe other default for
direction?
79: (12)                     self.rgb = np.array(list(map(color_to_rgb, colors)))
80: (12)                     def pos_to_rgb(pos: np.ndarray) -> tuple[float, float, float,
float]:
81: (16)                         vec = self.func(pos)
82: (16)                         color_value = np.clip(
83: (20)                             self.color_scheme(vec),
84: (20)                             min_color_scheme_value,
85: (20)                             max_color_scheme_value,
86: (16)                         )
87: (16)                         alpha = inverse_interpolate(
88: (20)                             min_color_scheme_value,
89: (20)                             max_color_scheme_value,
90: (20)                             color_value,
91: (16)                         )

```

```

92: (16)                         alpha *= len(self.rgbss) - 1
93: (16)                         c1 = self.rgbss[int(alpha)]
94: (16)                         c2 = self.rgbss[min(int(alpha + 1), len(self.rgbss) - 1)]
95: (16)                         alpha %= 1
96: (16)                         return interpolate(c1, c2, alpha)
97: (12)                         self.pos_to_rgb = pos_to_rgb
98: (12)                         self.pos_to_color = lambda pos: rgb_to_color(self.pos_to_rgb(pos))
99: (8)                          else:
100: (12)                         self.single_color = True
101: (12)                         self.color = ManimColor.parse(color)
102: (8)                          self.submob_movement_updater = None
103: (4) @staticmethod
104: (4) def shift_func(
105: (8)     func: Callable[[np.ndarray], np.ndarray],
106: (8)     shift_vector: np.ndarray,
107: (4) ) -> Callable[[np.ndarray], np.ndarray]:
108: (8)     """Shift a vector field function.
109: (8)     Parameters
110: (8)     -----
111: (8)     func
112: (12)         The function defining a vector field.
113: (8)     shift_vector
114: (12)         The shift to be applied to the vector field.
115: (8)     Returns
116: (8)     -----
117: (8)     `Callable[[np.ndarray], np.ndarray]`  

118: (12)         The shifted vector field function.
119: (8)     """
120: (8)     return lambda p: func(p - shift_vector)
121: (4) @staticmethod
122: (4) def scale_func(
123: (8)     func: Callable[[np.ndarray], np.ndarray],
124: (8)     scalar: float,
125: (4) ) -> Callable[[np.ndarray], np.ndarray]:
126: (8)     """Scale a vector field function.
127: (8)     Parameters
128: (8)     -----
129: (8)     func
130: (12)         The function defining a vector field.
131: (8)     scalar
132: (12)         The scalar to be applied to the vector field.
133: (8)     Examples
134: (8)     -----
135: (8)     .. manim:: ScaleVectorFieldFunction
136: (12)         class ScaleVectorFieldFunction(Scene):
137: (16)             def construct(self):
138: (20)                 func = lambda pos: np.sin(pos[1]) * RIGHT + np.cos(pos[0])
* UP
139: (20)                 vector_field = ArrowVectorField(func)
140: (20)                 self.add(vector_field)
141: (20)                 self.wait()
142: (20)                 func = VectorField.scale_func(func, 0.5)
143: (20)
self.play(vector_field.animate.become(ArrowVectorField(func)))
144: (20)                 self.wait()
145: (8)             Returns
146: (8)             -----
147: (8)             `Callable[[np.ndarray], np.ndarray]`  

148: (12)         The scaled vector field function.
149: (8)         """
150: (8)         return lambda p: func(p * scalar)
def fit_to_coordinate_system(self, coordinate_system: CoordinateSystem):
151: (4)         """Scale the vector field to fit a coordinate system.
152: (8)         This method is useful when the vector field is defined in a coordinate
153: (8)         system
154: (8)         different from the one used to display the vector field.
155: (8)         This method can only be used once because it transforms the origin of
each vector.
156: (8)         Parameters

```

```

157: (8)           -----
158: (8)           coordinate_system
159: (12)          The coordinate system to fit the vector field to.
160: (8)
161: (8)          self.apply_function(lambda pos:
coordinate_system.coords_to_point(*pos))
162: (4)          def nudge(
163: (8)            self,
164: (8)            mob: Mobject,
165: (8)            dt: float = 1,
166: (8)            substeps: int = 1,
167: (8)            pointwise: bool = False,
168: (4)        ) -> VectorField:
169: (8)          """Nudge a :class:`~.Mobject` along the vector field.
170: (8)          Parameters
171: (8)          -----
172: (8)            mob
173: (12)          The mobject to move along the vector field
174: (8)            dt
175: (12)          A scalar to the amount the mobject is moved along the vector
field.
176: (12)          The actual distance is based on the magnitude of the vector field.
177: (8)            substeps
178: (12)          The amount of steps the whole nudge is divided into. Higher values
179: (12)          give more accurate approximations.
180: (8)            pointwise
181: (12)          Whether to move the mobject along the vector field. If `False` the
182: (12)          vector field takes effect on the center of the given
183: (12)          :class:`~.Mobject`. If `True` the vector field takes effect on the
184: (12)          points of the individual points of the :class:`~.Mobject`,
185: (12)          potentially distorting it.
186: (8)          Returns
187: (8)          -----
188: (8)          VectorField
189: (12)          This vector field.
190: (8)          Examples
191: (8)          -----
192: (8)          .. manim:: Nudging
193: (12)          class Nudging(Scene):
194: (16)          def construct(self):
195: (20)          func = lambda pos: np.sin(pos[1] / 2) * RIGHT +
np.cos(pos[0] / 2) * UP
196: (20)          vector_field = ArrowVectorField(
197: (24)            func, x_range=[-7, 7, 1], y_range=[-4, 4, 1],
length_func=lambda x: x / 2
198: (20)          )
199: (20)          self.add(vector_field)
200: (20)          circle = Circle(radius=2).shift(LEFT)
201: (20)          self.add(circle.copy().set_color(GRAY))
202: (20)          dot = Dot().move_to(circle)
203: (20)          vector_field.nudge(circle, -2, 60, True)
204: (20)          vector_field.nudge(dot, -2, 60)
205: (20)
circle.add_updater(vector_field.get_nudge_updater(pointwise=True))
206: (20)          dot.add_updater(vector_field.get_nudge_updater())
207: (20)          self.add(circle, dot)
208: (20)          self.wait(6)
209: (8)          """
210: (8)          def runge_kutta(self, p: Sequence[float], step_size: float) -> float:
211: (12)          """Returns the change in position of a point along a vector field.
212: (12)          Parameters
213: (12)          -----
214: (12)            p
215: (15)          The position of each point being moved along the vector field.
216: (12)            step_size
217: (15)          A scalar that is used to determine how much a point is shifted
in a single step.
218: (12)          Returns
219: (12)          -----

```

```

220: (12)           float
221: (15)           How much the point is shifted.
222: (12)
223: (12)
224: (12)
225: (12)
226: (12)
227: (12)
228: (8)           step_size = dt / substeps
229: (8)           for _ in range(substeps):
230: (12)           if pointwise:
231: (16)             mob.apply_function(lambda p: p + runge_kutta(self, p,
232: (12)               step_size))
233: (16)           else:
234: (8)             mob.shift(runge_kutta(self, mob.get_center(), step_size))
235: (4)           return self
236: (8)
237: (8)
238: (8)
239: (8)
240: (4)           def nudge_submobjects(
241: (8)             self,
242: (8)             dt: float = 1,
243: (8)             substeps: int = 1,
244: (8)             pointwise: bool = False,
245: (12)         ) -> VectorField:
246: (12)           """Apply a nudge along the vector field to all submobjects.
247: (8)           Parameters
248: (8)           -----
249: (8)           dt
250: (8)             A scalar to the amount the mobject is moved along the vector
251: (12)             field.
252: (8)           The actual distance is based on the magnitude of the vector field.
253: (8)           substeps
254: (8)             The amount of steps the whole nudge is divided into. Higher values
255: (12)             give more accurate approximations.
256: (8)           pointwise
257: (8)             Whether to move the mobject along the vector field. See
258: (12)             :meth:`nudge` for details.
259: (8)
260: (4)           Returns
261: (8)
262: (8)
263: (8)
264: (4)           def get_nudge_updater(
265: (8)             self,
266: (8)             speed: float = 1,
267: (8)             pointwise: bool = False,
268: (12)         ) -> Callable[[Mobject, float], Mobject]:
269: (8)             """Get an update function to move a :class:`~.Mobject` along the
270: (12)             vector field.
271: (8)             When used with :meth:`~.Mobject.add_updater`, the mobject will move
272: (12)             along the vector field, where its speed is determined by the magnitude of the vector field.
273: (8)             Parameters
274: (8)             -----
275: (8)             speed
276: (12)             At `speed=1` the distance a mobject moves per second is equal to
277: (8)             the magnitude of the vector field along its path. The speed value scales the speed of such a
278: (8)             mobject.
279: (8)             pointwise
272: (12)             Whether to move the mobject along the vector field. See
273: (8)
274: (8)
275: (8)
276: (12)
277: (8)
278: (8)
279: (4)             pointwise=pointwise)
279: (4)           def start_submobject_movement(

```

```

280: (8)             self,
281: (8)             speed: float = 1,
282: (8)             pointwise: bool = False,
283: (4)         ) -> VectorField:
284: (8)             """Start continuously moving all submobjects along the vector field.
285: (8)             Calling this method multiple times will result in removing the
previous updater created by this method.
286: (8)             Parameters
287: (8)             -----
288: (8)             speed
289: (12)                 The speed at which to move the submobjects. See
:meth:`get_nudge_updater` for details.
290: (8)             pointwise
291: (12)                 Whether to move the mobject along the vector field. See
:meth:`nudge` for details.
292: (8)             Returns
293: (8)             -----
294: (8)             VectorField
295: (12)                 This vector field.
296: (8)             """
297: (8)             self.stop_submobject_movement()
298: (8)             self.submob_movement_updater = lambda mob, dt: mob.nudge_submobjects(
299: (12)                 dt * speed,
300: (12)                 pointwise=pointwise,
301: (8)             )
302: (8)             self.add_updater(self.submob_movement_updater)
303: (8)             return self
304: (4)         def stop_submobject_movement(self) -> VectorField:
305: (8)             """Stops the continuous movement started using
:meth:`start_submobject_movement` .
306: (8)             Returns
307: (8)             -----
308: (8)             VectorField
309: (12)                 This vector field.
310: (8)             """
311: (8)             self.remove_updater(self.submob_movement_updater)
312: (8)             self.submob_movement_updater = None
313: (8)             return self
314: (4)         def get_colored_background_image(self, sampling_rate: int = 5) ->
Image.Image:
315: (8)             """Generate an image that displays the vector field.
316: (8)             The color at each position is calculated by passing the positng
through a
317: (8)             series of steps:
318: (8)             Calculate the vector field function at that position, map that vector
to a
319: (8)             single value using `self.color_scheme` and finally generate a color
from
320: (8)             that value using the color gradient.
321: (8)             Parameters
322: (8)             -----
323: (8)             sampling_rate
324: (12)                 The stepsize at which pixels get included in the image. Lower
values give
325: (12)                 more accurate results, but may take a long time to compute.
326: (8)             Returns
327: (8)             -----
328: (8)             Image.Image
329: (12)                 The vector field image.
330: (8)             """
331: (8)             if self.single_color:
332: (12)                 raise ValueError(
333: (16)                     "There is no point in generating an image if the vector field
uses a single color.",
334: (12)                 )
335: (8)                 ph = int(config["pixel_height"] / sampling_rate)
336: (8)                 pw = int(config["pixel_width"] / sampling_rate)
337: (8)                 fw = config["frame_width"]
338: (8)                 fh = config["frame_height"]

```

```

339: (8)             points_array = np.zeros((ph, pw, 3))
340: (8)             x_array = np.linspace(-fw / 2, fw / 2, pw)
341: (8)             y_array = np.linspace(fh / 2, -fh / 2, ph)
342: (8)             x_array = x_array.reshape((1, len(x_array)))
343: (8)             y_array = y_array.reshape((len(y_array), 1))
344: (8)             x_array = x_array.repeat(ph, axis=0)
345: (8)             y_array.repeat(pw, axis=1) # TODO why not y_array =
y_array.repeat(...)?
346: (8)             points_array[:, :, 0] = x_array
347: (8)             points_array[:, :, 1] = y_array
348: (8)             rgbs = np.apply_along_axis(self.pos_to_rgb, 2, points_array)
349: (8)             return Image.fromarray((rgbs * 255).astype("uint8"))
350: (4)             def get_vectorized_rgba_gradient_function(
351: (8)                 self,
352: (8)                 start: float,
353: (8)                 end: float,
354: (8)                 colors: Iterable[ParsableManimColor],
355: (4)             ):
356: (8)                 """
357: (8)                 Generates a gradient of rgbs as a numpy array
358: (8)                 Parameters
359: (8)                 -----
360: (8)                 start
361: (12)                     start value used for inverse interpolation at
:func:`~.inverse_interpolate`  

362: (8)                 end
363: (12)                     end value used for inverse interpolation at
:func:`~.inverse_interpolate`  

364: (8)                 colors
365: (12)                     list of colors to generate the gradient
366: (8)             Returns
367: (8)             -----
368: (12)                     function to generate the gradients as numpy arrays representing
rgba values
369: (8)             """
370: (8)             rgbs = np.array([color_to_rgb(c) for c in colors])
371: (8)             def func(values, opacity=1):
372: (12)                 alphas = inverse_interpolate(start, end, np.array(values))
373: (12)                 alphas = np.clip(alphas, 0, 1)
374: (12)                 scaled_alphas = alphas * (len(rgbs) - 1)
375: (12)                 indices = scaled_alphas.astype(int)
376: (12)                 next_indices = np.clip(indices + 1, 0, len(rgbs) - 1)
377: (12)                 inter_alphas = scaled_alphas % 1
378: (12)                 inter_alphas = inter_alphas.repeat(3).reshape((len(indices), 3))
379: (12)                 result = interpolate(rgbs[indices], rgbs[next_indices],
inter_alphas)
380: (12)                 result = np.concatenate(
381: (16)                     (result, np.full([len(result), 1], opacity)),
382: (16)                     axis=1,
383: (12)                 )
384: (12)                 return result
385: (8)             return func
386: (0)             class ArrowVectorField(VectorField):
387: (4)                 """A :class:`VectorField` represented by a set of change vectors.
388: (4)                 Vector fields are always based on a function defining the
:class:`~.Vector` at every position.
389: (4)                     The values of this functions is displayed as a grid of vectors.
390: (4)                     By default the color of each vector is determined by it's magnitude.
391: (4)                     Other color schemes can be used however.
392: (4)                     Parameters
393: (4)                     -----
394: (4)                     func
395: (8)                     The function defining the rate of change at every position of the
vector field.
396: (4)                     color
397: (8)                     The color of the vector field. If set, position-specific coloring is
disabled.
398: (4)                     color_scheme
399: (8)                     A function mapping a vector to a single value. This value gives the

```

position in the color gradient defined using `min\_color\_scheme\_value`, `max\_color\_scheme\_value` and `colors`.

400: (4) min\_color\_scheme\_value  
 401: (8) The value of the color\_scheme function to be mapped to the first color in `colors`. Lower values also result in the first color of the gradient.

402: (4) max\_color\_scheme\_value  
 403: (8) The value of the color\_scheme function to be mapped to the last color in `colors`. Higher values also result in the last color of the gradient.

404: (4) colors  
 405: (8) The colors defining the color gradient of the vector field.

406: (4) x\_range  
 407: (8) A sequence of x\_min, x\_max, delta\_x

408: (4) y\_range  
 409: (8) A sequence of y\_min, y\_max, delta\_y

410: (4) z\_range  
 411: (8) A sequence of z\_min, z\_max, delta\_z

412: (4) three\_dimensions  
 413: (8) Enables three\_dimensions. Default set to False, automatically turns True if z\_range is not None.

414: (4) length\_func  
 415: (8) The function determining the displayed size of the vectors. The actual size of the vector is passed, the returned value will be used as display vector. By default this is used to cap the displayed size of vectors to reduce the clutter.

416: (4) opacity  
 417: (8) The opacity of the arrows.

418: (4) vector\_config  
 419: (8) Additional arguments to be passed to the :class:`~.Vector` constructor

420: (4) kwargs  
 421: (8) Additional arguments to be passed to the :class:`~.VGroup` constructor

422: (4) Examples  
 423: (4) -----  
 424: (8) .. manim:: BasicUsage  
 425: (12) :save\_last\_frame:  
 426: (8) class BasicUsage(Scene):  
 427: (12) def construct(self):  
 428: (16) func = lambda pos: ((pos[0] \* UR + pos[1] \* LEFT) - pos) / 3  
 429: (16) self.add(ArrowVectorField(func))  
 430: (4) .. manim:: SizingAndSpacing  
 431: (8) class SizingAndSpacing(Scene):  
 432: (12) def construct(self):  
 433: (16) func = lambda pos: np.sin(pos[0] / 2) \* UR + np.cos(pos[1] / 2) \* LEFT  
 434: (16) vf = ArrowVectorField(func, x\_range=[-7, 7, 1])  
 435: (16) self.add(vf)  
 436: (16) self.wait()  
 437: (16) length\_func = lambda x: x / 3  
 438: (16) vf2 = ArrowVectorField(func, x\_range=[-7, 7, 1],  
 439: (16) length\_func=length\_func)  
 440: (16) self.play(vf.animate.become(vf2))  
 441: (16) self.wait()  
 442: (4) .. manim:: Coloring  
 443: (8) :save\_last\_frame:  
 444: (8) class Coloring(Scene):  
 445: (12) def construct(self):  
 446: (16) func = lambda pos: pos - LEFT \* 5  
 447: (16) colors = [RED, YELLOW, BLUE, DARK\_GRAY]  
 448: (16) min\_radius = Circle(radius=2, color=colors[0]).shift(LEFT \* 5)  
 449: (16) max\_radius = Circle(radius=10, color=colors[-1]).shift(LEFT \* 5)  
 450: (16) vf = ArrowVectorField(  
 451: (20) func, min\_color\_scheme\_value=2, max\_color\_scheme\_value=10,  
 452: (16) colors=colors  
 453: (16) )  
 454: (16) self.add(vf, min\_radius, max\_radius)  
 455: (4) """

```

457: (4)
458: (8)
459: (8)
460: (8)
461: (8)
462: (8)
463: (8)
464: (8)
465: (8)
466: (8)
467: (8)
468: (8)
469: (8)
470: (8)
471: (8)
472: (8)
473: (4)
474: (8)
475: (12)
476: (12)
477: (8)
478: (8)
479: (12)
480: (12)
481: (8)
482: (8)
483: (8)
484: (12)
485: (12)
486: (8)
487: (12)
488: (8)
489: (12)
490: (16)
491: (12)
492: (8)
493: (8)
494: (12)
495: (12)
496: (12)
497: (12)
498: (12)
499: (12)
500: (12)
501: (8)
502: (8)
503: (8)
504: (8)
505: (12)
506: (8)
507: (8)
508: (8)
509: (8)
510: (8)
511: (8)
512: (12)
513: (16)
514: (16)
515: (12)
516: (8)
517: (8)
518: (4)
519: (8)
520: (8)
521: (8)
522: (8)
523: (8)

def __init__(self,
             func: Callable[[np.ndarray], np.ndarray],
             color: ParsableManimColor | None = None,
             color_scheme: Callable[[np.ndarray], float] | None = None,
             min_color_scheme_value: float = 0,
             max_color_scheme_value: float = 2,
             colors: Sequence[ParsableManimColor] = DEFAULT_SCALAR_FIELD_COLORS,
             x_range: Sequence[float] = None,
             y_range: Sequence[float] = None,
             z_range: Sequence[float] = None,
             three_dimensions: bool = False, # Automatically True if z_range is
set
469: (8)
sigmoid(norm),
470: (8)
471: (8)
472: (8)
473: (4)
474: (8)
475: (12)
476: (12)
477: (8)
478: (8)
479: (12)
480: (12)
481: (8)
482: (8)
483: (8)
484: (12)
485: (12)
486: (8)
487: (12)
488: (8)
489: (12)
490: (16)
491: (12)
492: (8)
493: (8)
494: (12)
495: (12)
496: (12)
497: (12)
498: (12)
499: (12)
500: (12)
501: (8)
502: (8)
503: (8)
504: (8)
505: (12)
506: (8)
507: (8)
508: (8)
509: (8)
510: (8)
511: (8)
512: (12)
513: (16)
514: (16)
515: (12)
516: (8)
517: (8)
518: (4)
519: (8)
520: (8)
521: (8)
522: (8)
523: (8)

        self.x_range = x_range or [
            floor(-config["frame_width"] / 2),
            ceil(config["frame_width"] / 2),
        ]
        self.y_range = y_range or [
            floor(-config["frame_height"] / 2),
            ceil(config["frame_height"] / 2),
        ]
        self.ranges = [self.x_range, self.y_range]
        if three_dimensions or z_range:
            self.z_range = z_range or self.y_range.copy()
            self.ranges += [self.z_range]
        else:
            self.ranges += [[0, 0]]
        for i in range(len(self.ranges)):
            if len(self.ranges[i]) == 2:
                self.ranges[i] += [0.5]
            self.ranges[i][1] += self.ranges[i][2]
        self.x_range, self.y_range, self.z_range = self.ranges
super().__init__(
    func,
    color,
    color_scheme,
    min_color_scheme_value,
    max_color_scheme_value,
    colors,
    **kwargs,
)
self.length_func = length_func
self.opacity = opacity
if vector_config is None:
    vector_config = {}
self.vector_config = vector_config
self.func = func
x_range = np.arange(*self.x_range)
y_range = np.arange(*self.y_range)
z_range = np.arange(*self.z_range)
self.add(
    *[
        self.get_vector(x * RIGHT + y * UP + z * OUT)
        for x, y, z in it.product(x_range, y_range, z_range)
    ]
)
self.set_opacity(self.opacity)

def get_vector(self, point: np.ndarray):
    """Creates a vector in the vector field.
    The created vector is based on the function of the vector field and is
rooted in the given point. Color and length fit the specifications of
this vector field.
Parameters
"""

```

```

524: (8)           -----
525: (8)           point
526: (12)         The root point of the vector.
527: (8)
528: (8)           """
529: (8)           output = np.array(self.func(point))
530: (8)           norm = np.linalg.norm(output)
531: (12)         if norm != 0:
532: (8)             output *= self.length_func(norm) / norm
533: (8)           vect = Vector(output, **self.vector_config)
534: (8)           vect.shift(point)
535: (12)         if self.single_color:
536: (8)             vect.set_color(self.color)
537: (8)         else:
538: (12)             vect.set_color(self.pos_to_color(point))
539: (0)          return vect
540: (4)          """StreamLines represent the flow of a :class:`VectorField` using the
trace of moving agents.
541: (4)          Vector fields are always based on a function defining the vector at every
position.
542: (4)          The values of this functions is displayed by moving many agents along the
vector field
543: (4)          and showing their trace.
544: (4)          Parameters
545: (4)          -----
546: (4)          func
547: (8)          The function defining the rate of change at every position of the
vector field.
548: (4)          color
549: (8)          The color of the vector field. If set, position-specific coloring is
disabled.
550: (4)          color_scheme
551: (8)          A function mapping a vector to a single value. This value gives the
position in the color gradient defined using `min_color_scheme_value`, `max_color_scheme_value`
and `colors`.
552: (4)          min_color_scheme_value
553: (8)          The value of the color_scheme function to be mapped to the first color
in `colors`. Lower values also result in the first color of the gradient.
554: (4)          max_color_scheme_value
555: (8)          The value of the color_scheme function to be mapped to the last color
in `colors`. Higher values also result in the last color of the gradient.
556: (4)          colors
557: (8)          The colors defining the color gradient of the vector field.
558: (4)          x_range
559: (8)          A sequence of x_min, x_max, delta_x
560: (4)          y_range
561: (8)          A sequence of y_min, y_max, delta_y
562: (4)          z_range
563: (8)          A sequence of z_min, z_max, delta_z
564: (4)          three_dimensions
565: (8)          Enables three_dimensions. Default set to False, automatically turns
True if
566: (8)          z_range is not None.
567: (4)          noise_factor
568: (8)          The amount by which the starting position of each agent is altered
along each axis. Defaults to :code:`delta_y / 2` if not defined.
569: (4)          n_repeats
570: (8)          The number of agents generated at each starting point.
571: (4)          dt
572: (8)          The factor by which the distance an agent moves per step is stretched.
Lower values result in a better approximation of the trajectories in the vector field.
573: (4)          virtual_time
574: (8)          The time the agents get to move in the vector field. Higher values
therefore result in longer stream lines. However, this whole time gets simulated upon creation.
575: (4)          max_anchors_per_line
576: (8)          The maximum number of anchors per line. Lines with more anchors get
reduced in complexity, not in length.
577: (4)          padding
578: (8)          The distance agents can move out of the generation area before being

```

```

terminated.

579: (4)           stroke_width
580: (8)             The stroke width of the stream lines.
581: (4)           opacity
582: (8)             The opacity of the stream lines.
583: (4)           Examples
584: (4)           -----
585: (4)           .. manim:: BasicUsage
586: (8)             :save_last_frame:
587: (8)               class BasicUsage(Scene):
588: (12)                 def construct(self):
589: (16)                   func = lambda pos: ((pos[0] * UR + pos[1] * LEFT) - pos) / 3
590: (16)                   self.add(StreamLines(func))
591: (4)           .. manim:: SpawningAndFlowingArea
592: (8)             :save_last_frame:
593: (8)               class SpawningAndFlowingArea(Scene):
594: (12)                 def construct(self):
595: (16)                   func = lambda pos: np.sin(pos[0]) * UR + np.cos(pos[1]) * LEFT
596: (16)
597: (20)           + pos / 5
598: (16)
599: (16)           padding=1
600: (16)
601: (16)           * 2.5)]
602: (16)
603: (20)
604: (16)
605: (4)
606: (4)           """
607: (8)           def __init__(
608: (8)             self,
609: (8)             func: Callable[[np.ndarray], np.ndarray],
610: (8)             color: ParsableManimColor | None = None,
611: (8)             color_scheme: Callable[[np.ndarray], float] | None = None,
612: (8)             min_color_scheme_value: float = 0,
613: (8)             max_color_scheme_value: float = 2,
614: (8)             colors: Sequence[ParsableManimColor] = DEFAULT_SCALAR_FIELD_COLORS,
615: (8)             x_range: Sequence[float] = None,
616: (8)             y_range: Sequence[float] = None,
617: (8)             z_range: Sequence[float] = None,
618: (8)             three_dimensions: bool = False,
619: (8)             noise_factor: float | None = None,
620: (8)             n_repeats=1,
621: (8)             dt=0.05,
622: (8)             virtual_time=3,
623: (8)             max_anchors_per_line=100,
624: (8)             padding=3,
625: (8)             stroke_width=1,
626: (8)             opacity=1,
627: (4)             **kwargs,
628: (8)           ):
629: (12)             self.x_range = x_range or [
630: (12)               floor(-config["frame_width"] / 2),
631: (8)               ceil(config["frame_width"] / 2),
632: (8)             ]
633: (12)             self.y_range = y_range or [
634: (12)               floor(-config["frame_height"] / 2),
635: (8)               ceil(config["frame_height"] / 2),
636: (8)             ]
637: (8)             self.ranges = [self.x_range, self.y_range]
638: (12)             if three_dimensions or z_range:
639: (12)               self.z_range = z_range or self.y_range.copy()
640: (8)               self.ranges += [self.z_range]
641: (12)             else:
642: (8)               self.ranges += [[0, 0]]
643: (12)             for i in range(len(self.ranges)):
644: (8)               if len(self.ranges[i]) == 2:

```

```

644: (16)           self.ranges[i] += [0.5]
645: (12)           self.ranges[i][1] += self.ranges[i][2]
646: (8)            self.x_range, self.y_range, self.z_range = self.ranges
647: (8)            super().__init__(
648: (12)              func,
649: (12)              color,
650: (12)              color_scheme,
651: (12)              min_color_scheme_value,
652: (12)              max_color_scheme_value,
653: (12)              colors,
654: (12)              **kwargs,
655: (8)
656: (8)
657: (12)          self.noise_factor = (
658: (8)            noise_factor if noise_factor is not None else self.y_range[2] / 2
659: (8)
660: (8)          self.n_repeats = n_repeats
661: (8)          self.virtual_time = virtual_time
662: (8)          self.max_anchors_per_line = max_anchors_per_line
663: (8)          self.padding = padding
664: (8)          self.stroke_width = stroke_width
665: (8)          half_noise = self.noise_factor / 2
666: (8)          np.random.seed(0)
667: (8)          start_points = np.array(
668: (12)            [
669: (16)              (x - half_noise) * RIGHT
670: (16)              + (y - half_noise) * UP
671: (16)              + (z - half_noise) * OUT
672: (16)              + self.noise_factor * np.random.random(3)
673: (16)              for n in range(self.n_repeats)
674: (16)              for x in np.arange(*self.x_range)
675: (16)              for y in np.arange(*self.y_range)
676: (12)              for z in np.arange(*self.z_range)
677: (8)
678: (8)
679: (12)      def outside_box(p):
680: (16)          return (
681: (16)              p[0] < self.x_range[0] - self.padding
682: (16)              or p[0] > self.x_range[1] + self.padding - self.x_range[2]
683: (16)              or p[1] < self.y_range[0] - self.padding
684: (16)              or p[1] > self.y_range[1] + self.padding - self.y_range[2]
685: (16)              or p[2] < self.z_range[0] - self.padding
686: (12)              or p[2] > self.z_range[1] + self.padding - self.z_range[2]
687: (8)
688: (8)
689: (12)      max_steps = ceil(virtual_time / dt) + 1
690: (12)      if not self.single_color:
691: (16)          self.background_img = self.get_colored_background_image()
692: (20)          if config["renderer"] == RendererType.OPENGL:
693: (20)              self.values_to_rgbs =
694: (20)              self.get_vectorized_rgba_gradient_function(
695: (16)                  min_color_scheme_value,
696: (8)                  max_color_scheme_value,
697: (12)                  colors,
698: (12)
699: (16)                  )
700: (16)                  for point in start_points:
701: (16)                      points = [point]
702: (20)                      for _ in range(max_steps):
703: (16)                          last_point = points[-1]
704: (12)                          new_point = last_point + dt * func(last_point)
705: (12)                          if outside_box(new_point):
706: (16)                              break
707: (12)                          points.append(new_point)
708: (12)                      step = max_steps
709: (12)                      if not step:
710: (12)                          continue
711: (12)                      line = get_vectorized_mobject_class()()

```

```

712: (16)             line.set_stroke(
713: (20)                 color=self.color, width=self.stroke_width, opacity=opacity
714: (16)
715: (12)         )
716: (16)     else:
717: (20)         if config.renderer == RendererType.OPENGL:
718: (20)             line.set_stroke(width=self.stroke_width / 4.0)
719: (24)             norms = np.array(
720: (20)                 [np.linalg.norm(self.func(point)) for point in
721: (20)                   line.points],
722: (24)
723: (24)             )
724: (20)             line.set_rgba_array_direct(
725: (16)                 self.values_to_rgbs(norms, opacity),
726: (20)                 name="stroke_rgba",
727: (24)
728: (28)             )
729: (24)         else:
730: (20)             if np.any(self.z_range != np.array([0, 0.5, 0.5])):
731: (24)                 line.set_stroke(
732: (20)                     [self.pos_to_color(p) for p in
733: (12)                         ]
734: (8)                     )
735: (4)             else:
736: (8)                 line.color_using_background_image(self.background_img)
737: (20)             line.set_stroke(width=self.stroke_width, opacity=opacity)
738: (12)         self.add(line)
739: (8)     self.stream_lines = [*self.subobjects]
740: (4) def create(
741: (8)     self,
742: (8)     lag_ratio: float | None = None,
743: (8)     run_time: Callable[[float], float] | None = None,
744: (8)     **kwargs,
745: (4) ) -> AnimationGroup:
746: (8)     """The creation animation of the stream lines.
747: (8)     The stream lines appear in random order.
748: (8)     Parameters
749: (8)     -----
750: (8)     lag_ratio
751: (12)         The lag ratio of the animation.
752: (12)         If undefined, it will be selected so that the total animation
753: (12)         length is 1.5 times the run time of each stream line creation.
754: (8)     run_time
755: (12)         The run time of every single stream line creation. The runtime of
756: (12)         the whole animation might be longer due to the `lag_ratio`.
757: (12)         If undefined, the virtual time of the stream lines is used as run
758: (12)         time.
759: (16)         Returns
760: (16)         -----
761: (16)         :class:`~.AnimationGroup`
762: (16)         The creation animation of the stream lines.
763: (16) Examples
764: (16) -----
765: (16) .. manim:: StreamLineCreation
766: (16)     class StreamLineCreation(Scene):
767: (16)         def construct(self):
768: (20)             func = lambda pos: (pos[0] * UR + pos[1] * LEFT) - pos
769: (20)             stream_lines = StreamLines(
770: (20)                 func,
771: (20)                 color=YELLOW,
772: (20)                 x_range=[-7, 7, 1],
773: (20)                 y_range=[-4, 4, 1],
774: (20)                 stroke_width=3,
775: (20)                 virtual_time=1, # use shorter lines
776: (20)                 max_anchors_per_line=5, # better performance with
777: (20)                 fewer anchors
778: (20)             )
779: (20)             self.play(stream_lines.create()) # uses virtual_time as
780: (20)             run_time
781: (20)             self.wait()
782: (8)         """
783: (8)         if run_time is None:

```

```

774: (12)                      run_time = self.virtual_time
775: (8)                       if lag_ratio is None:
776: (12)                         lag_ratio = run_time / 2 / len(self.submobjects)
777: (8)                         animations = [
778: (12)                           Create(line, run_time=run_time, **kwargs) for line in
self.stream_lines
779: (8)                           ]
780: (8)                         random.shuffle(animations)
781: (8)                         return AnimationGroup(*animations, lag_ratio=lag_ratio)
782: (4)                     def start_animation(
783: (8)                       self,
784: (8)                         warm_up: bool = True,
785: (8)                         flow_speed: float = 1,
786: (8)                         time_width: float = 0.3,
787: (8)                         rate_func: Callable[[float], float] = linear,
788: (8)                         line_animation_class: type[ShowPassingFlash] = ShowPassingFlash,
789: (8)                         **kwargs,
790: (4)                     ) -> None:
791: (8)                         """Animates the stream lines using an updater.
792: (8)                         The stream lines will continuously flow
Parameters
-----
793: (8)                         warm_up
794: (8)                           If `True` the animation is initialized line by line. Otherwise it
795: (8) starts with all lines shown.
796: (12)                         flow_speed
797: (8)                           At `flow_speed=1` the distance the flow moves per second is equal
798: (12) to the magnitude of the vector field along its path. The speed value scales the speed of this
flow.
799: (8)                         time_width
800: (12)                           The proportion of the stream line shown while being animated
801: (8)                         rate_func
802: (12)                           The rate function of each stream line flashing
803: (8)                         line_animation_class
804: (12)                           The animation class being used
805: (8)                         Examples
-----
806: (8)
807: (8)                         .. manim:: ContinuousMotion
808: (12)                         class ContinuousMotion(Scene):
809: (16)                           def construct(self):
810: (20)                             func = lambda pos: np.sin(pos[0] / 2) * UR + np.cos(pos[1]
/ 2) * LEFT
811: (20)                             max_anchors_per_line=30)
812: (20)
813: (20)                             stream_lines = StreamLines(func, stroke_width=3,
814: (20)                             self.add(stream_lines)
815: (8)                             stream_lines.start_animation(warm_up=False,
816: (8)                             self.wait(stream_lines.virtual_time /
817: (12)                             """
818: (12)                             for line in self.stream_lines:
819: (16)                               run_time = line.duration / flow_speed
820: (16)                               line.anim = line_animation_class(
821: (16)                                 line,
822: (16)                                 run_time=run_time,
823: (16)                                 rate_func=rate_func,
824: (12)                                 time_width=time_width,
825: (12)                                 **kwargs,
826: (12)                               )
827: (12)                               line.anim.begin()
828: (16)                               line.time = random.random() * self.virtual_time
829: (12)                               if warm_up:
830: (8)                                 line.time *= -1
831: (12)                               self.add(line.anim.mobject)
832: (16)                         def updater(mob, dt):
833: (16)                           for line in mob.stream_lines:
834: (20)                             line.time += dt * flow_speed
835: (16)                             if line.time >= self.virtual_time:
836: (16)                               line.time -= self.virtual_time

```

```

835: (16)                         line.anim.interpolate(np.clip(line.time / line.anim.run_time,
836: (8)                           0, 1))
837: (8)                         self.add_updater(updater)
838: (8)                         self.flow_animation = updater
839: (8)                         self.flow_speed = flow_speed
840: (4)                         self.time_width = time_width
841: (8)                         def end_animation(self) -> AnimationGroup:
842: (8)                           """End the stream line animation smoothly.
843: (8)                           Returns an animation resulting in fully displayed stream lines without
844: (8)                           a noticeable cut.
845: (8)                           Returns
846: (12)                          -----
847: (8)                           :class:`~.AnimationGroup`  

848: (8)                           The animation fading out the running stream animation.
849: (8)                           Raises
850: (12)                          -----
851: (8)                           ValueError
852: (8)                           if no stream line animation is running
853: (8)                           Examples
854: (12)                          -----
855: (16)                           .. manim:: EndAnimation
856: (20)                           class EndAnimation(Scene):
857: (20)                             def construct(self):
858: (24)                               func = lambda pos: np.sin(pos[0] / 2) * UR + np.cos(pos[1]
/ 2) * LEFT
859: (20)                               virtual_time=1, color=BLUE
860: (20)                               )
861: (20)                               self.add(stream_lines)
862: (20)                               stream_lines.start_animation(warm_up=False,
863: (20)                               flow_speed=1.5, time_width=0.5)
864: (8)                               self.wait(1)
865: (8)                               self.play(stream_lines.end_animation())
866: (12)                               """
867: (8)                               if self.flow_animation is None:
868: (12)                                 raise ValueError("You have to start the animation before fading it
out.")
869: (16)                               def hide_and_wait(mob, alpha):
870: (12)                                 if alpha == 0:
871: (16)                                   mob.set_stroke(opacity=0)
872: (8)                                 elif alpha == 1:
873: (12)                                   mob.set_stroke(opacity=1)
874: (12)                               def finish_updater_cycle(line, alpha):
875: (12)                                 line.time += dt * self.flow_speed
876: (16)                                 line.anim.interpolate(min(line.time / line.anim.run_time, 1))
877: (16)                                 if alpha == 1:
878: (8)                                   self.remove(line.anim.mobject)
879: (8)                                   line.anim.finish()
880: (8)                                 max_run_time = self.virtual_time / self.flow_speed
881: (8)                                 creation_rate_func = ease_out_sine
882: (12)                                 creation_staring_speed = creation_rate_func(0.001) * 1000
883: (8)                                 creation_run_time = (
884: (8)                                   max_run_time / (1 + self.time_width) * creation_staring_speed
885: (8)                                 )
886: (8)                                 dt = 1 / config["frame_rate"]
887: (8)                                 animations = []
888: (8)                                 self.remove_updater(self.flow_animation)
889: (12)                                 self.flow_animation = None
890: (16)                                 for line in self.stream_lines:
891: (16)                                   create = Create(
892: (16)                                     line,
893: (12)                                     run_time=creation_run_time,
894: (12)                                     rate_func=creation_rate_func,
895: (16)                                   )
896: (20)                                   if line.time <= 0:
897: (24)                                     animations.append(
Succession(
UpdateFromAlphaFunc(

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
898: (28)
899: (28)
900: (28)
901: (24)
902: (24)
903: (20)
904: (16)
905: (16)
906: (16)
907: (12)
908: (16)
909: (16)
910: (20)
911: (24)
912: (28)
913: (28)
914: (28)
915: (24)
916: (24)
917: (20)
918: (16)
919: (8)      return AnimationGroup(*animations)
```

-----  
File 104 - value\_tracker.py:

```
1: (0)      """Simple mobjects that can be used for storing (and updating) a value."""
2: (0)      from __future__ import annotations
3: (0)      __all__ = ["ValueTracker", "ComplexValueTracker"]
4: (0)      import numpy as np
5: (0)      from manim.mobject.mobject import Mobject
6: (0)      from manim.mobject.opengl.opengl_compatibility import ConvertToOpenGL
7: (0)      from manim.utils.paths import straight_path
8: (0)      class ValueTracker(Mobject, metaclass=ConvertToOpenGL):
9: (4)          """A mobject that can be used for tracking (real-valued) parameters.
10: (4)          Useful for animating parameter changes.
11: (4)          Not meant to be displayed. Instead the position encodes some
12: (4)          number, often one which another animation or continual_animation
13: (4)          uses for its update function, and by treating it as a mobject it can
14: (4)          still be animated and manipulated just like anything else.
15: (4)          This value changes continuously when animated using the :attr:`animate` syntax.
16: (4)      Examples
17: (4)      -----
18: (4)      .. manim:: ValueTrackerExample
19: (8)          class ValueTrackerExample(Scene):
20: (12)              def construct(self):
21: (16)                  number_line = NumberLine()
22: (16)                  pointer = Vector(DOWN)
23: (16)                  label = MathTex("x").add_updater(lambda m: m.next_to(pointer,
UP))
24: (16)
25: (16)
26: (20)
27: (32)
28: (32)
29: (28)
30: (16)
31: (16)
32: (16)
33: (16)
34: (16)
35: (16)
36: (16)
37: (16)
38: (16)
39: (16)
40: (16)                  tracker = ValueTracker(0)
pointer.add_updater(
    lambda m: m.next_to(
        number_line.n2p(tracker.get_value()),
        UP
    )
)
self.add(number_line, pointer, label)
tracker += 1.5
self.wait(1)
tracker -= 4
self.wait(0.5)
self.play(tracker.animate.set_value(5))
self.wait(0.5)
self.play(tracker.animate.set_value(3))
self.play(tracker.animate.increment_value(-2))
self.wait(0.5)
```

```

41: (4)          .. note::
42: (8)          You can also link ValueTrackers to updaters. In this case, you have to
make sure that the
43: (8)          ValueTracker is added to the scene by ``add``
44: (4)          .. manim:: ValueTrackerExample
45: (8)          class ValueTrackerExample(Scene):
46: (12)          def construct(self):
47: (16)          tracker = ValueTracker(0)
48: (16)          label = Dot(radius=3).add_updater(lambda x :
x.set_x(tracker.get_value()))
49: (16)          self.add(label)
50: (16)          self.add(tracker)
51: (16)          tracker.add_updater(lambda mobject, dt:
mobject.increment_value(dt))
52: (16)          self.wait(2)
53: (4)          """
54: (4)          def __init__(self, value=0, **kwargs):
55: (8)          super().__init__(**kwargs)
56: (8)          self.set(points=np.zeros((1, 3)))
57: (8)          self.set_value(value)
58: (4)          def get_value(self) -> float:
59: (8)          """Get the current value of this ValueTracker."""
60: (8)          return self.points[0, 0]
61: (4)          def set_value(self, value: float):
62: (8)          """Sets a new scalar value to the ValueTracker"""
63: (8)          self.points[0, 0] = value
64: (8)          return self
65: (4)          def increment_value(self, d_value: float):
66: (8)          """Increments (adds) a scalar value to the ValueTracker"""
67: (8)          self.set_value(self.get_value() + d_value)
68: (8)          return self
69: (4)          def __bool__(self):
70: (8)          """Return whether the value of this value tracker evaluates as
true."""
71: (8)          return bool(self.get_value())
72: (4)          def __iadd__(self, d_value: float):
73: (8)          """adds ``+=`` syntax to increment the value of the ValueTracker"""
74: (8)          self.increment_value(d_value)
75: (8)          return self
76: (4)          def __ifloordiv__(self, d_value: float):
77: (8)          """Set the value of this value tracker to the floor division of the
current value by ``d_value``."""
78: (8)          self.set_value(self.get_value() // d_value)
79: (8)          return self
80: (4)          def __imod__(self, d_value: float):
81: (8)          """Set the value of this value tracker to the current value modulo
``d_value``."""
82: (8)          self.set_value(self.get_value() % d_value)
83: (8)          return self
84: (4)          def __imul__(self, d_value: float):
85: (8)          """Set the value of this value tracker to the product of the current
value and ``d_value``."""
86: (8)          self.set_value(self.get_value() * d_value)
87: (8)          return self
88: (4)          def __ipow__(self, d_value: float):
89: (8)          """Set the value of this value tracker to the current value raised to
the power of ``d_value``."""
90: (8)          self.set_value(self.get_value() ** d_value)
91: (8)          return self
92: (4)          def __isub__(self, d_value: float):
93: (8)          """adds ``-=`` syntax to decrement the value of the ValueTracker"""
94: (8)          self.increment_value(-d_value)
95: (8)          return self
96: (4)          def __itruediv__(self, d_value: float):
97: (8)          """Sets the value of this value tracker to the current value divided
by ``d_value``."""
98: (8)          self.set_value(self.get_value() / d_value)
99: (8)          return self
100: (4)         def interpolate(self, mobject1, mobject2, alpha,

```

```

path_func=straight_path()):
    """
    Turns self into an interpolation between mobject1
    and mobject2.
    """
    self.set(points=path_func(mobject1.points, mobject2.points, alpha))
    return self
class ComplexValueTracker(ValueTracker):
    """Tracks a complex-valued parameter.
    When the value is set through :attr:`animate`, the value will take a
straight path from the
    source point to the destination point.
Examples
-----
.. manim:: ComplexValueTrackerExample
    class ComplexValueTrackerExample(Scene):
        def construct(self):
            tracker = ComplexValueTracker(-2+1j)
            dot = Dot().add_updater(
                lambda x: x.move_to(tracker.points)
            )
            self.add(NumberPlane(), dot)
            self.play(tracker.animate.set_value(3+2j))
            self.play(tracker.animate.set_value(tracker.get_value() * 1j))
            self.play(tracker.animate.set_value(tracker.get_value() - 2j))
            self.play(tracker.animate.set_value(tracker.get_value() / (-2
+ 3j)))
        """
        def get_value(self):
            """Get the current value of this value tracker as a complex number.
            The value is internally stored as a points array [a, b, 0]. This can
be accessed directly
            to represent the value geometrically, see the usage example."""
            return complex(*self.points[0, :2])
        def set_value(self, z):
            """Sets a new complex value to the ComplexValueTracker"""
            z = complex(z)
            self.points[0, :2] = (z.real, z.imag)
            return self
-----
```

## File 105 - plugins\_flags.py:

```

1: (0)      """Plugin Managing Utility"""
2: (0)      from __future__ import annotations
3: (0)      import sys
4: (0)      from typing import Any
5: (0)      if sys.version_info < (3, 10):
6: (4)          from importlib_metadata import entry_points
7: (0)      else:
8: (4)          from importlib.metadata import entry_points
9: (0)      from manim import console
10: (0)      __all__ = ["list_plugins"]
11: (0)      def get_plugins() -> dict[str, Any]:
12: (4)          plugins: dict[str, Any] = {
13: (8)              entry_point.name: entry_point.load()
14: (8)              for entry_point in entry_points(group="manim.plugins")
15: (4)          }
16: (4)          return plugins
17: (0)      def list_plugins() -> None:
18: (4)          console.print("[green bold]Plugins:[/green bold]", justify="left")
19: (4)          plugins = get_plugins()
20: (4)          for plugin in plugins:
21: (8)              console.print(f" • {plugin}")
-----
```

## File 106 - cairo\_renderer.py:

```

1: (0)         from __future__ import annotations
2: (0)         import typing
3: (0)         import numpy as np
4: (0)         from manim.utils.hashing import get_hash_from_play_call
5: (0)         from .. import config, logger
6: (0)         from ..camera.camera import Camera
7: (0)         from ..mobject.mobject import Mobject
8: (0)         from ..scene.scene_file_writer import SceneFileWriter
9: (0)         from ..utils.exceptions import EndSceneEarlyException
10: (0)        from ..utils.iterables import list_update
11: (0)        if typing.TYPE_CHECKING:
12: (4)            import types
13: (4)            from typing import Any, Iterable
14: (4)            from manim.animation.animation import Animation
15: (4)            from manim.scene.scene import Scene
16: (0)        __all__ = ["CairoRenderer"]
17: (0)        class CairoRenderer:
18: (4)            """A renderer using Cairo.
19: (4)            num_plays : Number of play() functions in the scene.
20: (4)            time: time elapsed since initialisation of scene.
21: (4)            """
22: (4)            def __init__(
23: (8)                self,
24: (8)                file_writer_class=SceneFileWriter,
25: (8)                camera_class=None,
26: (8)                skip_animations=False,
27: (8)                **kwargs,
28: (4)            ):
29: (8)                self._file_writer_class = file_writer_class
30: (8)                camera_cls = camera_class if camera_class is not None else Camera
31: (8)                self.camera = camera_cls()
32: (8)                self._original_skipping_status = skip_animations
33: (8)                self.skip_animations = skip_animations
34: (8)                self.animations_hashes = []
35: (8)                self.num_plays = 0
36: (8)                self.time = 0
37: (8)                self.static_image = None
38: (4)            def init_scene(self, scene):
39: (8)                self.file_writer: Any = self._file_writer_class(
40: (12)                    self,
41: (12)                    scene.__class__.__name__,
42: (8)                )
43: (4)            def play(
44: (8)                self,
45: (8)                scene: Scene,
46: (8)                *args: Animation | Iterable[Animation] |
types.GeneratorType[Animation],
47: (8)                **kwargs,
48: (4)            ):
49: (8)                self.skip_animations = self._original_skipping_status
50: (8)                self.update_skipping_status()
51: (8)                scene.compile_animation_data(*args, **kwargs)
52: (8)                if self.skip_animations:
53: (12)                    logger.debug(f"Skipping animation {self.num_plays}")
54: (12)                    hash_current_animation = None
55: (12)                    self.time += scene.duration
56: (8)                else:
57: (12)                    if config["disable_caching"]:
58: (16)                        logger.info("Caching disabled.")
59: (16)                        hash_current_animation = f"uncached_{self.num_plays:05}"
60: (12)                    else:
61: (16)                        hash_current_animation = get_hash_from_play_call(
62: (20)                            scene,
63: (20)                            self.camera,
64: (20)                            scene.animations,
65: (20)                            scene.mobjects,
66: (16)                        )
67: (16)                        if self.file_writer.is_already_cached(hash_current_animation):

```

```

68: (20)                               logger.info(
69: (24)                                 f"Animation {self.num_plays} : Using cached data (hash
: %(hash_current_animation)s)",
70: (24)                                 {"hash_current_animation": hash_current_animation},
71: (20)                                 )
72: (20)                                 self.skip_animations = True
73: (20)                                 self.time += scene.duration
74: (8)                                  self.file_writer.add_partial_movie_file(hash_current_animation)
75: (8)                                  self.animations_hashes.append(hash_current_animation)
76: (8)                                  logger.debug(
77: (12)                                    "List of the first few animation hashes of the scene: %(h)s",
78: (12)                                    {"h": str(self.animations_hashes[:5])}),
79: (8)                                 )
80: (8)                                  self.file_writer.begin_animation(not self.skip_animations)
81: (8)                                  scene.begin_animations()
82: (8)                                  self.save_static_frame_data(scene, scene.static_mobjects)
83: (8)                                  if scene.is_current_animation_frozen_frame():
84: (12)                                    self.update_frame(scene, mobjects=scene.moving_mobjects)
85: (12)                                    self.freeze_current_frame(scene.duration)
86: (8)                                  else:
87: (12)                                    scene.play_internal()
88: (8)                                  self.file_writer.end_animation(not self.skip_animations)
89: (8)                                  self.num_plays += 1
90: (4) def update_frame( # TODO Description in Docstring
91: (8)     self,
92: (8)     scene,
93: (8)     mobjects: typing.Iterable[Mobject] | None = None,
94: (8)     include_submobjects: bool = True,
95: (8)     ignore_skipping: bool = True,
96: (8)     **kwargs,
97: (4) ):
98: (8)     """Update the frame.
99: (8)     Parameters
100: (8)     -----
101: (8)     scene
102: (8)     mobjects
103: (12)       list of mobjects
104: (8)     include_submobjects
105: (8)     ignore_skipping
106: (8)     **kwargs
107: (8)     """
108: (8)     if self.skip_animations and not ignore_skipping:
109: (12)       return
110: (8)     if not mobjects:
111: (12)       mobjects = list_update(
112: (16)         scene.mobjects,
113: (16)         scene.foreground_mobjects,
114: (12)       )
115: (8)     if self.static_image is not None:
116: (12)       self.camera.set_frame_to_background(self.static_image)
117: (8)     else:
118: (12)       self.camera.reset()
119: (8)     kwargs["include_submobjects"] = include_submobjects
120: (8)     self.camera.capture_mobjects(mobjects, **kwargs)
121: (4) def render(self, scene, time, moving_mobjects):
122: (8)     self.update_frame(scene, moving_mobjects)
123: (8)     self.add_frame(self.get_frame())
124: (4) def get_frame(self):
125: (8)     """
126: (8)       Gets the current frame as NumPy array.
127: (8)       Returns
128: (8)       -----
129: (8)       np.array
130: (12)         NumPy array of pixel values of each pixel in screen.
131: (12)         The shape of the array is height x width x 3
132: (8)         """
133: (8)         return np.array(self.camera.pixel_array)
134: (4) def add_frame(self, frame: np.ndarray, num_frames: int = 1):
135: (8)         """

```

```

136: (8)           Adds a frame to the video_file_stream
137: (8)           Parameters
138: (8)
139: (8)
140: (12)          -----
141: (8)           frame
142: (12)          The frame to add, as a pixel array.
143: (8)           num_frames
144: (8)          The number of times to add frame.
145: (8)
146: (12)          """
147: (8)           if self.skip_animations:
148: (8)             return
149: (12)           self.time += num_frames * dt
150: (4)           for _ in range(num_frames):
151: (8)             self.file_writer.write_frame(frame)
152: (8)           def freeze_current_frame(self, duration: float):
153: (8)             """Adds a static frame to the movie for a given duration. The static
154: (8)             frame is the current frame.
155: (12)           Parameters
156: (8)
157: (8)             duration
158: (8)               [description]
159: (12)             """
160: (12)           dt = 1 / self.camera.frame_rate
161: (8)           self.add_frame(
162: (8)             self.get_frame(),
163: (12)             num_frames=int(duration / dt),
164: (8)           )
165: (4)           def show_frame(self):
166: (8)
167: (8)             """Opens the current frame in the Default Image Viewer
168: (8)             of your system.
169: (8)             """
170: (8)             self.update_frame(ignore_skipping=True)
171: (8)             self.camera.get_image().show()
172: (4)           def save_static_frame_data(
173: (8)             self,
174: (8)             scene: Scene,
175: (8)             static_mobjects: typing.Iterable[Mobject],
176: (8)           ) -> typing.Iterable[Mobject] | None:
177: (8)             """Compute and save the static frame, that will be reused at each
178: (8)             frame
179: (12)             to avoid unnecessarily computing static mobjects.
180: (8)             Parameters
181: (12)             -----
182: (8)             scene
183: (8)               The scene played.
184: (8)             static_mobjects
185: (12)               Static mobjects of the scene. If None, self.static_image is set to
186: (8)             None
187: (8)             Returns
188: (8)             -----
189: (12)             typing.Iterable[Mobject]
190: (8)               The static image computed.
191: (8)             """
192: (8)             self.static_image = None
193: (8)             if not static_mobjects:
194: (8)               return None
195: (8)             self.update_frame(scene, mobjects=static_mobjects)
196: (8)             self.static_image = self.get_frame()
197: (8)             return self.static_image
198: (4)           def update_skipping_status(self):
199: (8)
200: (8)             """This method is used internally to check if the current
201: (8)             animation needs to be skipped or not. It also checks if
202: (8)             the number of animations that were played correspond to
203: (8)             the number of animations that need to be played, and
204: (8)             raises an EndSceneEarlyException if they don't correspond.
205: (8)             """
206: (8)             if self.file_writer.sections[-1].skip_animations:

```

```

202: (12)           self.skip_animations = True
203: (8)            if config["save_last_frame"]:
204: (12)              self.skip_animations = True
205: (8)            if (
206: (12)                config["from_animation_number"]
207: (12)                and self.num_plays < config["from_animation_number"]
208: (8)            ):
209: (12)              self.skip_animations = True
210: (8)            if (
211: (12)                config["upto_animation_number"]
212: (12)                and self.num_plays > config["upto_animation_number"]
213: (8)            ):
214: (12)              self.skip_animations = True
215: (12)              raise EndSceneEarlyException()
216: (4)            def scene_finished(self, scene):
217: (8)              if self.num_plays:
218: (12)                  self.file_writer.finish()
219: (8)              elif config.write_to_movie:
220: (12)                  config.save_last_frame = True
221: (12)                  config.write_to_movie = False
222: (8)              else:
223: (12)                  self.static_image = None
224: (12)                  self.update_frame(scene)
225: (8)              if config["save_last_frame"]:
226: (12)                  self.static_image = None
227: (12)                  self.update_frame(scene)
228: (12)                  self.file_writer.save_final_image(self.camera.get_image())

```

---

## File 107 - shader\_wrapper.py:

```

1: (0)          from __future__ import annotations
2: (0)          import copy
3: (0)          import re
4: (0)          from pathlib import Path
5: (0)          import moderngl
6: (0)          import numpy as np
7: (0)          from .. import logger
8: (0)          __all__ = ["ShaderWrapper"]
9: (0)          def get_shader_dir():
10: (4)             return Path(__file__).parent / "shaders"
11: (0)          def find_file(file_name: Path, directories: list[Path]) -> Path:
12: (4)              if file_name.exists():
13: (8)                  return file_name
14: (4)              possible_paths = (directory / file_name for directory in directories)
15: (4)              for path in possible_paths:
16: (8)                  if path.exists():
17: (12)                      return path
18: (8)                  else:
19: (12)                      logger.debug(f"{path} does not exist.")
20: (4)              raise OSError(f"{file_name} not Found")
21: (0)          class ShaderWrapper:
22: (4)              def __init__(
23: (8)                  self,
24: (8)                  vert_data=None,
25: (8)                  vert_indices=None,
26: (8)                  shader_folder=None,
27: (8)                  uniforms=None, # A dictionary mapping names of uniform variables
28: (8)                  texture_paths=None, # A dictionary mapping names to filepaths for
29: (8)                  textures.
30: (8)                  depth_test=False,
31: (4)                  render_primitive=moderngl.TRIANGLE_STRIP,
32: (8)              ):
33: (8)                  self.vert_data = vert_data
34: (8)                  self.vert_indices = vert_indices
35: (8)                  self.vert_attributes = vert_data.dtype.names
36: (8)                  self.shader_folder = Path(shader_folder or "")
36: (8)                  self.uniforms = uniforms or {}

```

```

37: (8)                     self.texture_paths = texture_paths or {}
38: (8)                     self.depth_test = depth_test
39: (8)                     self.render_primitive = str(render_primitive)
40: (8)                     self.init_program_code()
41: (8)                     self.refresh_id()
42: (4) def copy(self):
43: (8)     result = copy.copy(self)
44: (8)     result.vert_data = np.array(self.vert_data)
45: (8)     if result.vert_indices is not None:
46: (12)         result.vert_indices = np.array(self.vert_indices)
47: (8)     if self.uniforms:
48: (12)         result.uniforms = dict(self.uniforms)
49: (8)     if self.texture_paths:
50: (12)         result.texture_paths = dict(self.texture_paths)
51: (8)     return result
52: (4) def is_valid(self):
53: (8)     return all(
54: (12)         [
55: (16)             self.vert_data is not None,
56: (16)             self.program_code["vertex_shader"] is not None,
57: (16)             self.program_code["fragment_shader"] is not None,
58: (12)         ],
59: (8)     )
60: (4) def get_id(self):
61: (8)     return self.id
62: (4) def get_program_id(self):
63: (8)     return self.program_id
64: (4) def create_id(self):
65: (8)     return "|".join(
66: (12)         map(
67: (16)             str,
68: (16)             [
69: (20)                 self.program_id,
70: (20)                 self.uniforms,
71: (20)                 self.texture_paths,
72: (20)                 self.depth_test,
73: (20)                 self.render_primitive,
74: (16)             ],
75: (12)         ),
76: (8)     )
77: (4) def refresh_id(self):
78: (8)     self.program_id = self.create_program_id()
79: (8)     self.id = self.create_id()
80: (4) def create_program_id(self):
81: (8)     return hash(
82: (12)         "".join(
83: (16)             self.program_code[f"{name}_shader"] or ""
84: (16)             for name in ("vertex", "geometry", "fragment")
85: (12)         ),
86: (8)     )
87: (4) def init_program_code(self):
88: (8)     def get_code(name: str) -> str | None:
89: (12)         return get_shader_code_from_file(
90: (16)             self.shader_folder / f"{name}.glsl",
91: (12)         )
92: (8)         self.program_code = {
93: (12)             "vertex_shader": get_code("vert"),
94: (12)             "geometry_shader": get_code("geom"),
95: (12)             "fragment_shader": get_code("frag"),
96: (8)         }
97: (4) def get_program_code(self):
98: (8)     return self.program_code
99: (4) def replace_code(self, old, new):
100: (8)     code_map = self.program_code
101: (8)     for name, _code in code_map.items():
102: (12)         if code_map[name] is None:
103: (16)             continue
104: (12)         code_map[name] = re.sub(old, new, code_map[name])
105: (8)     self.refresh_id()

```

```

106: (4)             def combine_with(self, *shader_wrappers):
107: (8)                 if len(shader_wrappers) == 0:
108: (12)                     return
109: (8)                 if self.vert_indices is not None:
110: (12)                     num_verts = len(self.vert_data)
111: (12)                     indices_list = [self.vert_indices]
112: (12)                     data_list = [self.vert_data]
113: (12)                     for sw in shader_wrappers:
114: (16)                         indices_list.append(sw.vert_indices + num_verts)
115: (16)                         data_list.append(sw.vert_data)
116: (16)                         num_verts += len(sw.vert_data)
117: (12)                     self.vert_indices = np.hstack(indices_list)
118: (12)                     self.vert_data = np.hstack(data_list)
119: (8)                 else:
120: (12)                     self.vert_data = np.hstack(
121: (16)                         [self.vert_data, *(sw.vert_data for sw in shader_wrappers)],
122: (12)                     )
123: (8)             return self
124: (0)         filename_to_code_map: dict = {}
125: (0)         def get_shader_code_from_file(filename: Path) -> str | None:
126: (4)             if filename in filename_to_code_map:
127: (8)                 return filename_to_code_map[filename]
128: (4)             try:
129: (8)                 filepath = find_file(
130: (12)                     filename,
131: (12)                     directories=[get_shader_dir(), Path("/")],
132: (8)                 )
133: (4)             except OSError:
134: (8)                 return None
135: (4)             result = filepath.read_text()
136: (4)             insertions = re.findall(
137: (8)                 r"^\#include ../include/.*\.\glsl$",
138: (8)                 result,
139: (8)                 flags=re.MULTILINE,
140: (4)             )
141: (4)             for line in insertions:
142: (8)                 inserted_code = get_shader_code_from_file(
143: (12)                     Path() / "include" / line.replace("#include ../include/", ""),
144: (8)                 )
145: (8)                 if inserted_code is None:
146: (12)                     return None
147: (8)                 result = result.replace(line, inserted_code)
148: (4)             filename_to_code_map[filename] = result
149: (4)             return result
150: (0)         def get_colormap_code(rgb_list):
151: (4)             data = ",".join("vec3({}, {}, {})".format(*rgb) for rgb in rgb_list)
152: (4)             return f"vec3[{len(rgb_list)}]({data})"

```

-----  
File 108 - opengl\_renderer.py:

```

1: (0)             from __future__ import annotations
2: (0)             import itertools as it
3: (0)             import sys
4: (0)             import time
5: (0)             from typing import Any
6: (0)             if sys.version_info < (3, 8):
7: (4)                 from backports.cached_property import cached_property
8: (0)             else:
9: (4)                 from functools import cached_property
10: (0)             import moderngl
11: (0)             import numpy as np
12: (0)             from PIL import Image
13: (0)             from manim import config, logger
14: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject, OpenGLPoint
15: (0)             from manim.mobject.opengl.opengl_vectorized_mobject import OpenGLVMOBJ
16: (0)             from manim.utils.caching import handle_caching_play
17: (0)             from manim.utils.color import color_to_rgba

```

```

18: (0)         from manim.utils.exceptions import EndSceneEarlyException
19: (0)         from ..constants import *
20: (0)         from ..scene.scene_file_writer import SceneFileWriter
21: (0)         from ..utils import opengl
22: (0)         from ..utils.config_ops import _Data
23: (0)         from ..utils.simple_functions import clip
24: (0)         from ..utils.space_ops import (
25: (4)             angle_of_vector,
26: (4)             quaternion_from_angle_axis,
27: (4)             quaternion_mult,
28: (4)             rotation_matrix_transpose,
29: (4)             rotation_matrix_transpose_from_quaternion,
30: (0)
31: (0)     )
32: (0)     from .shader import Mesh, Shader
33: (4)     from .vectorized_mobject_rendering import (
34: (4)         render_opengl_vectorized_mobject_fill,
35: (4)         render_opengl_vectorized_mobject_stroke,
36: (0)
37: (0) class OpenGLCamera(OpenGLMobject):
38: (4)     euler_angles = _Data()
39: (4)     def __init__(
40: (8)         self,
41: (8)         frame_shape=None,
42: (8)         center_point=None,
43: (8)         euler_angles=[0, 0, 0],
44: (8)         focal_distance=2,
45: (8)         light_source_position=[-10, 10, 10],
46: (8)         orthographic=False,
47: (8)         minimum_polar_angle=-PI / 2,
48: (8)         maximum_polar_angle=PI / 2,
49: (8)         model_matrix=None,
50: (8)         **kwargs,
51: (4)     ):
52: (8)         self.use_z_index = True
53: (8)         self.frame_rate = 60
54: (8)         self.orthographic = orthographic
55: (8)         self.minimum_polar_angle = minimum_polar_angle
56: (8)         self.maximum_polar_angle = maximum_polar_angle
57: (8)         if self.orthographic:
58: (12)             self.projection_matrix = opengl.orthographic_projection_matrix()
59: (12)             self.unformatted_projection_matrix =
opengl.orthographic_projection_matrix(
60: (16)                 format=False,
61: (12)             )
62: (8)             else:
63: (12)                 self.projection_matrix = opengl.perspective_projection_matrix()
64: (12)                 self.unformatted_projection_matrix =
opengl.perspective_projection_matrix(
65: (16)                 format=False,
66: (12)             )
67: (8)             if frame_shape is None:
68: (12)                 self.frame_shape = (config["frame_width"], config["frame_height"])
69: (8)             else:
70: (12)                 self.frame_shape = frame_shape
71: (8)             if center_point is None:
72: (12)                 self.center_point = ORIGIN
73: (8)             else:
74: (12)                 self.center_point = center_point
75: (8)             if model_matrix is None:
76: (12)                 model_matrix = opengl.translation_matrix(0, 0, 11)
77: (8)             self.focal_distance = focal_distance
78: (8)             if light_source_position is None:
79: (12)                 self.light_source_position = [-10, 10, 10]
80: (8)             else:
81: (12)                 self.light_source_position = light_source_position
82: (8)             self.light_source = OpenGLPoint(self.light_source_position)
83: (8)             self.default_model_matrix = model_matrix
84: (8)             super().__init__(model_matrix=model_matrix, should_render=False,

```

```

**kwargs)
85: (8)         if euler_angles is None:
86: (12)           euler_angles = [0, 0, 0]
87: (8)           euler_angles = np.array(euler_angles, dtype=float)
88: (8)           self.euler_angles = euler_angles
89: (8)           self.refresh_rotation_matrix()
90: (4)       def get_position(self):
91: (8)           return self.model_matrix[:, 3][:3]
92: (4)       def set_position(self, position):
93: (8)           self.model_matrix[:, 3][:3] = position
94: (8)           return self
95: (4)   @cached_property
96: (4)   def formatted_view_matrix(self):
97: (8)       return opengl.matrix_to_shader_input(np.linalg.inv(self.model_matrix))
98: (4)   @cached_property
99: (4)   def unformatted_view_matrix(self):
100: (8)      return np.linalg.inv(self.model_matrix)
101: (4)   def init_points(self):
102: (8)       self.set_points([ORIGIN, LEFT, RIGHT, DOWN, UP])
103: (8)       self.set_width(self.frame_shape[0], stretch=True)
104: (8)       self.set_height(self.frame_shape[1], stretch=True)
105: (8)       self.move_to(self.center_point)
106: (4)   def to_default_state(self):
107: (8)       self.center()
108: (8)       self.set_height(config["frame_height"])
109: (8)       self.set_width(config["frame_width"])
110: (8)       self.set_euler_angles(0, 0, 0)
111: (8)       self.model_matrix = self.default_model_matrix
112: (8)       return self
113: (4)   def refresh_rotation_matrix(self):
114: (8)       theta, phi, gamma = self.euler_angles
115: (8)       quat = quaternion_mult(
116: (12)           quaternion_from_angle_axis(theta, OUT, axis_normalized=True),
117: (12)           quaternion_from_angle_axis(phi, RIGHT, axis_normalized=True),
118: (12)           quaternion_from_angle_axis(gamma, OUT, axis_normalized=True),
119: (8)       )
120: (8)       self.inverse_rotation_matrix =
rotation_matrix_transpose_from_quaternion(quat)
121: (4)   def rotate(self, angle, axis=OUT, **kwargs):
122: (8)       curr_rot_T = self.inverse_rotation_matrix
123: (8)       added_rot_T = rotation_matrix_transpose(angle, axis)
124: (8)       new_rot_T = np.dot(curr_rot_T, added_rot_T)
125: (8)       Fz = new_rot_T[2]
126: (8)       phi = np.arccos(Fz[2])
127: (8)       theta = angle_of_vector(Fz[:2]) + PI / 2
128: (8)       partial_rot_T = np.dot(
129: (12)           rotation_matrix_transpose(phi, RIGHT),
130: (12)           rotation_matrix_transpose(theta, OUT),
131: (8)       )
132: (8)       gamma = angle_of_vector(np.dot(partial_rot_T, new_rot_T.T)[:, 0])
133: (8)       self.set_euler_angles(theta, phi, gamma)
134: (8)       return self
135: (4)   def set_euler_angles(self, theta=None, phi=None, gamma=None):
136: (8)       if theta is not None:
137: (12)           self.euler_angles[0] = theta
138: (8)       if phi is not None:
139: (12)           self.euler_angles[1] = phi
140: (8)       if gamma is not None:
141: (12)           self.euler_angles[2] = gamma
142: (8)       self.refresh_rotation_matrix()
143: (8)       return self
144: (4)   def set_theta(self, theta):
145: (8)       return self.set_euler_angles(theta=theta)
146: (4)   def set_phi(self, phi):
147: (8)       return self.set_euler_angles(phi=phi)
148: (4)   def set_gamma(self, gamma):
149: (8)       return self.set_euler_angles(gamma=gamma)
150: (4)   def increment_theta(self, dtheta):
151: (8)       self.euler_angles[0] += dtheta

```

```

152: (8)                     self.refresh_rotation_matrix()
153: (8)                     return self
154: (4)             def increment_phi(self, dphi):
155: (8)                 phi = self.euler_angles[1]
156: (8)                 new_phi = clip(phi + dphi, -PI / 2, PI / 2)
157: (8)                 self.euler_angles[1] = new_phi
158: (8)                 self.refresh_rotation_matrix()
159: (8)                 return self
160: (4)             def increment_gamma(self, dgamma):
161: (8)                 self.euler_angles[2] += dgamma
162: (8)                 self.refresh_rotation_matrix()
163: (8)                 return self
164: (4)             def get_shape(self):
165: (8)                 return (self.get_width(), self.get_height())
166: (4)             def get_center(self):
167: (8)                 return self.points[0]
168: (4)             def get_width(self):
169: (8)                 points = self.points
170: (8)                 return points[2, 0] - points[1, 0]
171: (4)             def get_height(self):
172: (8)                 points = self.points
173: (8)                 return points[4, 1] - points[3, 1]
174: (4)             def get_focal_distance(self):
175: (8)                 return self.focal_distance * self.get_height()
176: (4)             def interpolate(self, *args, **kwargs):
177: (8)                 super().interpolate(*args, **kwargs)
178: (8)                 self.refresh_rotation_matrix()
179: (0)         class OpenGLRenderer:
180: (4)             def __init__(self, file_writer_class=SceneFileWriter,
skip_animations=False):
181: (8)                 self.anti_alias_width = 1.5
182: (8)                 self._file_writer_class = file_writer_class
183: (8)                 self._original_skipping_status = skip_animations
184: (8)                 self.skip_animations = skip_animations
185: (8)                 self.animation_start_time = 0
186: (8)                 self.animation_elapsed_time = 0
187: (8)                 self.time = 0
188: (8)                 self.animations_hashes = []
189: (8)                 self.num_plays = 0
190: (8)                 self.camera = OpenGLCamera()
191: (8)                 self.pressed_keys = set()
192: (8)                 self.path_to_texture_id = {}
193: (8)                 self.background_color = config["background_color"]
194: (4)             def init_scene(self, scene):
195: (8)                 self.partial_movie_files = []
196: (8)                 self.file_writer: Any = self._file_writer_class(
197: (12)                     self,
198: (12)                     scene.__class__.__name__,
199: (8)                 )
200: (8)                 self.scene = scene
201: (8)                 self.background_color = config["background_color"]
202: (8)                 if not hasattr(self, "window"):
203: (12)                     if self.should_create_window():
204: (16)                         from .opengl_renderer_window import Window
205: (16)                         self.window = Window(self)
206: (16)                         self.context = self.window.ctx
207: (16)                         self.frame_buffer_object = self.context.detect_framebuffer()
208: (12)                     else:
209: (16)                         self.window = None
210: (16)                         try:
211: (20)                             self.context = moderngl.create_context(standalone=True)
212: (16)                         except Exception:
213: (20)                             self.context = moderngl.create_context(
214: (24)                                 standalone=True,
215: (24)                                 backend="egl",
216: (20)   )
217: (16)                                 self.frame_buffer_object =
218: (16)                                     self.frame_buffer_object.use()

```

```

219: (12)                     self.context.enable(moderngl.BLEND)
220: (12)                     self.context.wireframe = config["enable_wireframe"]
221: (12)                     self.context.blend_func =
222: (16)                         moderngl.SRC_ALPHA,
223: (16)                         moderngl.ONE_MINUS_SRC_ALPHA,
224: (16)                         moderngl.ONE,
225: (16)                         moderngl.ONE,
226: (12)                     )
227: (4) def should_create_window(self):
228: (8)     if config["force_window"]:
229: (12)         logger.warning(
230: (16)             "'--force_window' is enabled, this is intended for debugging
purposes"
231: (16)             "and may impact performance if used when outputting files",
232: (12)         )
233: (12)         return True
234: (8)     return (
235: (12)         config["preview"]
236: (12)         and not config["save_last_frame"]
237: (12)         and not config["format"]
238: (12)         and not config["write_to_movie"]
239: (12)         and not config["dry_run"]
240: (8)     )
241: (4) def get_pixel_shape(self):
242: (8)     if hasattr(self, "frame_buffer_object"):
243: (12)         return self.frame_buffer_object.viewport[2:4]
244: (8)     else:
245: (12)         return None
246: (4) def refresh_perspective_uniforms(self, camera):
247: (8)     pw, ph = self.get_pixel_shape()
248: (8)     fw, fh = camera.get_shape()
249: (8)     anti_alias_width = self.anti_alias_width / (ph / fh)
250: (8)     rotation = camera.inverse_rotation_matrix
251: (8)     light_pos = camera.light_source.get_location()
252: (8)     light_pos = np.dot(rotation, light_pos)
253: (8)     self.perspective_uniforms = {
254: (12)         "frame_shape": camera.get_shape(),
255: (12)         "anti_alias_width": anti_alias_width,
256: (12)         "camera_center": tuple(camera.get_center()),
257: (12)         "camera_rotation": tuple(np.array(rotation).T.flatten()),
258: (12)         "light_source_position": tuple(light_pos),
259: (12)         "focal_distance": camera.get_focal_distance(),
260: (8)     }
261: (4) def render_mobject(self, mobject):
262: (8)     if isinstance(mobject, OpenGLMobject):
263: (12)         if config["use_projection_fill_shaders"]:
264: (16)             render_opengl_vectorized_mobject_fill(self, mobject)
265: (12)         if config["use_projection_stroke_shaders"]:
266: (16)             render_opengl_vectorized_mobject_stroke(self, mobject)
267: (8)     shader_wrapper_list = mobject.get_shader_wrapper_list()
268: (8)     for shader_wrapper in shader_wrapper_list:
269: (12)         shader = Shader(self.context, shader_wrapper.shader_folder)
270: (12)         for name, path in shader_wrapper.texture_paths.items():
271: (16)             tid = self.get_texture_id(path)
272: (16)             shader.shader_program[name].value = tid
273: (12)         for name, value in it.chain(
274: (16)             shader_wrapper.uniforms.items(),
275: (16)             self.perspective_uniforms.items(),
276: (12)         ):
277: (16)             try:
278: (20)                 shader.set_uniform(name, value)
279: (16)             except KeyError:
280: (20)                 pass
281: (12)         try:
282: (16)             shader.set_uniform(
283: (20)                 "u_view_matrix", self.scene.camera.formatted_view_matrix
284: (16)             )
285: (16)             shader.set_uniform(
286: (20)                 "u_projection_matrix",

```

```

287: (20)                     self.scene.camera.projection_matrix,
288: (16)                 )
289: (12)             except KeyError:
290: (16)                 pass
291: (12)             if shader_wrapper.depth_test:
292: (16)                 self.context.enable(moderngl.DEPTH_TEST)
293: (12)             else:
294: (16)                 self.context.disable(moderngl.DEPTH_TEST)
295: (12)             mesh = Mesh(
296: (16)                 shader,
297: (16)                 shader_wrapper.vert_data,
298: (16)                 indices=shader_wrapper.vert_indices,
299: (16)                 use_depth_test=shader_wrapper.depth_test,
300: (16)                 primitive=mobject.render_primitive,
301: (12)             )
302: (12)             mesh.set_uniforms(self)
303: (12)             mesh.render()
304: (4)             def get_texture_id(self, path):
305: (8)                 if repr(path) not in self.path_to_texture_id:
306: (12)                     tid = len(self.path_to_texture_id)
307: (12)                     texture = self.context.texture(
308: (16)                         size=path.size,
309: (16)                         components=len(path.getbands()),
310: (16)                         data=path.tobytes(),
311: (12)                     )
312: (12)                     texture.repeat_x = False
313: (12)                     texture.repeat_y = False
314: (12)                     texture.filter = (moderngl.NEAREST, moderngl.NEAREST)
315: (12)                     texture.swizzle = "RRR1" if path.mode == "L" else "RGBA"
316: (12)                     texture.use(location=tid)
317: (12)                     self.path_to_texture_id[repr(path)] = tid
318: (8)             return self.path_to_texture_id[repr(path)]
319: (4)             def update_skipping_status(self):
320: (8)                 """
321: (8)                 This method is used internally to check if the current
322: (8)                 animation needs to be skipped or not. It also checks if
323: (8)                 the number of animations that were played correspond to
324: (8)                 the number of animations that need to be played, and
325: (8)                 raises an EndSceneEarlyException if they don't correspond.
326: (8)                 """
327: (8)                 if self.file_writer.sections[-1].skip_animations:
328: (12)                     self.skip_animations = True
329: (8)                 if (
330: (12)                     config["from_animation_number"]
331: (12)                     and self.num_plays < config["from_animation_number"]
332: (8)                 ):
333: (12)                     self.skip_animations = True
334: (8)                 if (
335: (12)                     config["upto_animation_number"]
336: (12)                     and self.num_plays > config["upto_animation_number"]
337: (8)                 ):
338: (12)                     self.skip_animations = True
339: (12)                     raise EndSceneEarlyException()
340: (4)             @handle_caching_play
341: (4)             def play(self, scene, *args, **kwargs):
342: (8)                 self.animation_start_time = time.time()
343: (8)                 self.file_writer.begin_animation(not self.skip_animations)
344: (8)                 scene.compile_animation_data(*args, **kwargs)
345: (8)                 scene.begin_animations()
346: (8)                 if scene.is_current_animation_frozen_frame():
347: (12)                     self.update_frame(scene)
348: (12)                     if not self.skip_animations:
349: (16)                         for _ in range(int(config.frame_rate * scene.duration)):
350: (20)                             self.file_writer.write_frame(self)
351: (12)                     if self.window is not None:
352: (16)                         self.window.swap_buffers()
353: (16)                         while time.time() - self.animation_start_time <
354: (20)   scene.duration:
355: (20)   pass

```

```

355: (12)                         self.animation_elapsed_time = scene.duration
356: (8)
357: (12)                         else:
358: (8)                             scene.play_internal()
359: (8)                             self.file_writer.end_animation(not self.skip_animations)
360: (8)                             self.time += scene.duration
361: (4)                             self.num_plays += 1
362: (8)                         def clear_screen(self):
363: (8)                             self.frame_buffer_object.clear(*self.background_color)
364: (4)                             self.window.swap_buffers()
365: (8)                         def render(self, scene, frame_offset, moving_mobjects):
366: (8)                             self.update_frame(scene)
367: (12)                             if self.skip_animations:
368: (8)                                 return
369: (8)                             self.file_writer.write_frame(self)
370: (12)                             if self.window is not None:
371: (12)                                 self.window.swap_buffers()
372: (16)                                 while self.animation_elapsed_time < frame_offset:
373: (16)                                     self.update_frame(scene)
374: (16)                                     self.window.swap_buffers()
375: (4)                         def update_frame(self, scene):
376: (8)                             self.frame_buffer_object.clear(*self.background_color)
377: (8)                             self.refresh_perspective_uniforms(scene.camera)
378: (8)                             for mobject in scene.mobjects:
379: (12)                                 if not mobject.should_render:
380: (16)                                     continue
381: (12)                                 self.render_mobject(mobject)
382: (8)                             for obj in scene.meshes:
383: (12)                                 for mesh in obj.get_meshes():
384: (16)                                     mesh.set_uniforms(self)
385: (16)                                     mesh.render()
386: (8)                             self.animation_elapsed_time = time.time() - self.animation_start_time
387: (4)                         def scene_finished(self, scene):
388: (8)                             if self.num_plays > 0:
389: (12)                                 self.file_writer.finish()
390: (8)                             elif self.num_plays == 0 and config.write_to_movie:
391: (12)                                 config.write_to_movie = False
392: (8)                             if self.should_save_last_frame():
393: (12)                                 config.save_last_frame = True
394: (12)                                 self.update_frame(scene)
395: (12)                                 self.file_writer.save_final_image(self.get_image())
395: (4)                         def should_save_last_frame(self):
396: (8)                             if config["save_last_frame"]:
397: (12)                                 return True
398: (8)                             if self.scene.interactive_mode:
399: (12)                                 return False
400: (8)                             return self.num_plays == 0
401: (4)                         def get_image(self) -> Image.Image:
402: (8)                             """
403: (8)                             to image represents
404: (8)                             currently bound frame
405: (8)                             specifically
406: (8)                             means there is no
407: (8)                             means the first
408: (8)
409: (8)
410: (8)
411: (12)                             Returns
412: (8)                             -----
413: (8)                             PIL.Image
414: (8)                             The PIL image of the array.
415: (12)                             """
416: (12)                             raw_buffer_data = self.get_raw_frame_buffer_object_data()
417: (12)                             image = Image.frombytes(
418: (12)                                 "RGBA",
418: (12)                                 self.get_pixel_shape(),
418: (12)                                 raw_buffer_data,
418: (12)                                 "raw",

```

```

419: (12)                 "RGBA",
420: (12)                 0,
421: (12)                 -1,
422: (8)                  )
423: (8)                  return image
424: (4)                  def save_static_frame_data(self, scene, static_mobjects):
425: (8)                      pass
426: (4)                  def get_frame_buffer_object(self, context, samples=0):
427: (8)                      pixel_width = config["pixel_width"]
428: (8)                      pixel_height = config["pixel_height"]
429: (8)                      num_channels = 4
430: (8)                      return context.framebuffer(
431: (12)                          color_attachments=context.texture(
432: (16)                              (pixel_width, pixel_height),
433: (16)                              components=num_channels,
434: (16)                              samples=samples,
435: (12)                          ),
436: (12)                          depth_attachment=context.depth_renderbuffer(
437: (16)                              (pixel_width, pixel_height),
438: (16)                              samples=samples,
439: (12)                          ),
440: (8)                  )
441: (4)                  def get_raw_frame_buffer_object_data(self, dtype="f1"):
442: (8)                      num_channels = 4
443: (8)                      ret = self.frame_buffer_object.read(
444: (12)                          viewport=self.frame_buffer_object.viewport,
445: (12)                          components=num_channels,
446: (12)                          dtype=dtype,
447: (8)                      )
448: (8)                      return ret
449: (4)                  def get_frame(self):
450: (8)                      raw = self.get_raw_frame_buffer_object_data(dtype="f1")
451: (8)                      pixel_shape = self.get_pixel_shape()
452: (8)                      result_dimensions = (pixel_shape[1], pixel_shape[0], 4)
453: (8)                      np_buf = np.frombuffer(raw, dtype="uint8").reshape(result_dimensions)
454: (8)                      np_buf = np.flipud(np_buf)
455: (8)                      return np_buf
456: (4)                  def pixel_coords_to_space_coords(self, px, py, relative=False,
top_left=False):
457: (8)                      pixel_shape = self.get_pixel_shape()
458: (8)                      if pixel_shape is None:
459: (12)                          return np.array([0, 0, 0])
460: (8)                      pw, ph = pixel_shape
461: (8)                      fw, fh = config["frame_width"], config["frame_height"]
462: (8)                      fc = self.camera.get_center()
463: (8)                      if relative:
464: (12)                          return 2 * np.array([px / pw, py / ph, 0])
465: (8)                      else:
466: (12)                          scale = fh / ph
467: (12)                          return fc + scale * np.array(
468: (16)                              [(px - pw / 2), (-1 if top_left else 1) * (py - ph / 2), 0]
469: (12)                          )
470: (4)                  @property
471: (4)                  def background_color(self):
472: (8)                      return self._background_color
473: (4)                  @background_color.setter
474: (4)                  def background_color(self, value):
475: (8)                      self._background_color = color_to_rgba(value, 1.0)

-----

```

## File 109 - opengl\_renderer\_window.py:

```

1: (0)                  from __future__ import annotations
2: (0)                  import moderngl_window as mglw
3: (0)                  from moderngl_window.context.pyglet.window import Window as PygletWindow
4: (0)                  from moderngl_window.timers.clock import Timer
5: (0)                  from screeninfo import get_monitors
6: (0)                  from .. import __version__, config

```

```

7: (0)             __all__ = ["Window"]
8: (0)             class Window(PygletWindow):
9: (4)                 fullscreen = False
10: (4)                resizable = True
11: (4)                gl_version = (3, 3)
12: (4)                vsync = True
13: (4)                cursor = True
14: (4)                def __init__(self, renderer, size=config.window_size, **kwargs):
15: (8)                    monitors = get_monitors()
16: (8)                    mon_index = config.window_monitor
17: (8)                    monitor = monitors[min(mon_index, len(monitors) - 1)]
18: (8)                    if size == "default":
19: (12)                        window_width = monitor.width
20: (12)                        if not config.fullscreen:
21: (16)                            window_width //= 2
22: (12)                            window_height = int(
23: (16)                                window_width * config.frame_height // config.frame_width,
24: (12)                            )
25: (12)                            size = (window_width, window_height)
26: (8)                    else:
27: (12)                        size = tuple(size)
28: (8)                    super().__init__(size=size)
29: (8)                    self.title = f"Manim Community {__version__}"
30: (8)                    self.size = size
31: (8)                    self.renderer = renderer
32: (8)                    mglw.activate_context(window=self)
33: (8)                    self.timer = Timer()
34: (8)                    self.config = mglw.WindowConfig(ctx=self.ctx, wnd=self,
35: (8)                        timer=self.timer)
36: (8)                    self.timer.start()
37: (8)                    self.swap_buffers()
38: (8)                    initial_position = self.find_initial_position(size, monitor)
39: (4)                    self.position = initial_position
40: (8)                    def on_mouse_motion(self, x, y, dx, dy):
41: (8)                        super().on_mouse_motion(x, y, dx, dy)
42: (8)                        point = self.renderer.pixel_coords_to_space_coords(x, y)
43: (8)                        d_point = self.renderer.pixel_coords_to_space_coords(dx, dy,
44: (8)                            relative=True)
45: (4)                        self.renderer.scene.on_mouse_motion(point, d_point)
46: (8)                    def on_mouse_scroll(self, x, y, x_offset: float, y_offset: float):
47: (8)                        super().on_mouse_scroll(x, y, x_offset, y_offset)
48: (8)                        point = self.renderer.pixel_coords_to_space_coords(x, y)
49: (12)                        offset = self.renderer.pixel_coords_to_space_coords(
50: (12)                            x_offset,
51: (8)                            y_offset,
52: (8)                            relative=True,
53: (8)                        )
54: (8)                        self.renderer.scene.on_mouse_scroll(point, offset)
55: (4)                    def on_key_press(self, symbol, modifiers):
56: (8)                        self.renderer.pressed_keys.add(symbol)
57: (8)                        super().on_key_press(symbol, modifiers)
58: (8)                        self.renderer.scene.on_key_press(symbol, modifiers)
59: (12)                    def on_key_release(self, symbol, modifiers):
60: (8)                        if symbol in self.renderer.pressed_keys:
61: (8)                            self.renderer.pressed_keys.remove(symbol)
62: (8)                        super().on_key_release(symbol, modifiers)
63: (8)                        self.renderer.scene.on_key_release(symbol, modifiers)
64: (8)                    def on_mouse_drag(self, x, y, dx, dy, buttons, modifiers):
65: (8)                        super().on_mouse_drag(x, y, dx, dy, buttons, modifiers)
66: (8)                        point = self.renderer.pixel_coords_to_space_coords(x, y)
67: (8)                        d_point = self.renderer.pixel_coords_to_space_coords(dx, dy,
68: (8)                            relative=True)
69: (8)                        self.renderer.scene.on_mouse_drag(point, d_point, buttons, modifiers)
70: (8)                    def find_initial_position(self, size, monitor):
71: (12)                        custom_position = config.window_position
72: (8)                        window_width, window_height = size
73: (8)                        if len(custom_position) == 1:
74: (12)                            raise ValueError(
75: (16)                                "window_position must specify both Y and X positions (Y/X ->

```

```

UR). Also accepts LEFT/RIGHT/ORIGIN/UP/DOWN.",

73: (12)          )
74: (8)          if custom_position in ["LEFT", "RIGHT"]:
75: (12)          custom_position = "0" + custom_position[0]
76: (8)          elif custom_position in ["UP", "DOWN"]:
77: (12)          custom_position = custom_position[0] + "0"
78: (8)          elif custom_position == "ORIGIN":
79: (12)          custom_position = "0" * 2
80: (8)          elif "," in custom_position:
81: (12)              return tuple(map(int, custom_position.split(",")))
82: (8)          char_to_n = {"L": 0, "U": 0, "O": 1, "R": 2, "D": 2}
83: (8)          width_diff = monitor.width - window_width
84: (8)          height_diff = monitor.height - window_height
85: (8)          return (
86: (12)              monitor.x + char_to_n[custom_position[1]] * width_diff // 2,
87: (12)              -monitor.y + char_to_n[custom_position[0]] * height_diff // 2,
88: (8)          )
89: (4)      def on_mouse_press(self, x, y, button, modifiers):
90: (8)          super().on_mouse_press(x, y, button, modifiers)
91: (8)          point = self.renderer.pixel_coords_to_space_coords(x, y)
92: (8)          mouse_button_map = {
93: (12)              1: "LEFT",
94: (12)              2: "MOUSE",
95: (12)              4: "RIGHT",
96: (8)          }
97: (8)          self.renderer.scene.on_mouse_press(point, mouse_button_map[button],
modifiers)

-----

```

## File 110 - scene.py:

```

1: (0)      """Basic canvas for animations."""
2: (0)      from __future__ import annotations
3: (0)      from manim.utils.parameter_parsing import flatten_iterable_parameters
4: (0)      __all__ = ["Scene"]
5: (0)      import copy
6: (0)      import datetime
7: (0)      import inspect
8: (0)      import platform
9: (0)      import random
10: (0)      import threading
11: (0)      import time
12: (0)      import types
13: (0)      from queue import Queue
14: (0)      import srt
15: (0)      from manim.scene.section import DefaultSectionType
16: (0)      try:
17: (4)          import dearpygui.dearpygui as dpg
18: (4)          dearpygui_imported = True
19: (0)      except ImportError:
20: (4)          dearpygui_imported = False
21: (0)      from typing import TYPE_CHECKING
22: (0)      import numpy as np
23: (0)      from tqdm import tqdm
24: (0)      from watchdog.events import FileSystemEventHandler
25: (0)      from watchdog.observers import Observer
26: (0)      from manim.mobject.mobject import Mobject
27: (0)      from manim.mobject.opengl.opengl_mobject import OpenGLPoint
28: (0)      from .. import config, logger
29: (0)      from ..animation.animation import Animation, Wait, prepare_animation
30: (0)      from ..camera.camera import Camera
31: (0)      from ..constants import *
32: (0)      from ..gui.gui import configure_pygui
33: (0)      from ..renderer.cairo_renderer import CairoRenderer
34: (0)      from ..renderer.opengl_renderer import OpenGLRenderer
35: (0)      from ..renderer.shader import Object3D
36: (0)      from ..utils import opengl, space_ops
37: (0)      from ..utils.exceptions import EndSceneEarlyException, RerunSceneException

```

```

38: (0)             from ..utils.family import extract_mobject_family_members
39: (0)             from ..utils.family_ops import
restructure_list_to_exclude_certain_family_members
40: (0)             from ..utils.file_ops import open_media_file
41: (0)             from ..utils.iterables import list_difference_update, list_update
42: (0)             if TYPE_CHECKING:
43: (4)                 from typing import Callable, Iterable
44: (0)             class RerunSceneHandler(FileSystemEventHandler):
45: (4)                 """A class to handle rerunning a Scene after the input file is
modified."""
46: (4)             def __init__(self, queue):
47: (8)                 super().__init__()
48: (8)                 self.queue = queue
49: (4)             def on_modified(self, event):
50: (8)                 self.queue.put(("rerun_file", [], {}))
class Scene:
    """A Scene is the canvas of your animation.
The primary role of :class:`Scene` is to provide the user with tools to
mobjects and animations. Generally speaking, a manim script consists of a
that derives from :class:`Scene` whose :meth:`Scene.construct` method is
by the user's code.
Mobjects are displayed on screen by calling :meth:`Scene.add` and removed
screen by calling :meth:`Scene.remove`. All mobjects currently on screen
in :attr:`Scene.mobjects`. Animations are played by calling
A :class:`Scene` is rendered internally by calling :meth:`Scene.render`.
turn calls :meth:`Scene.setup`, :meth:`Scene.construct`, and
:meth:`Scene.tear_down`, in that order.
It is not recommended to override the ``__init__`` method in user Scenes.
that should be ran before a Scene is rendered, use :meth:`Scene.setup`
Examples
-----
Override the :meth:`Scene.construct` method with your code.
.. code-block:: python
    class MyScene(Scene):
        def construct(self):
            self.play(Write(Text("Hello World!")))
"""
def __init__(
    self,
    renderer=None,
    camera_class=Camera,
    always_update_mobjects=False,
    random_seed=None,
    skip_animations=False,
):
    self.camera_class = camera_class
    self.always_update_mobjects = always_update_mobjects
    self.random_seed = random_seed
    self.skip_animations = skip_animations
    self.animations = None
    self.stop_condition = None
    self.moving_mobjects = []
    self.static_mobjects = []
    self.time_progression = None
    self.duration = None
    self.last_t = None
    self.queue = Queue()
    self.skip_animation_preview = False
    self.meshes = []
    self.camera_target = ORIGIN

```

```

96: (8)             self.widgets = []
97: (8)             self.dearpygui_imported = dearpygui_imported
98: (8)             self.updaters = []
99: (8)             self.point_lights = []
100: (8)            self.ambient_light = None
101: (8)            self.key_to_function_map = {}
102: (8)            self.mouse_press_callbacks = []
103: (8)            self.interactive_mode = False
104: (8)            if config.renderer == RendererType.OPENGL:
105: (12)                self.mouse_point = OpenGLPoint()
106: (12)                self.mouse_drag_point = OpenGLPoint()
107: (12)                if renderer is None:
108: (16)                    renderer = OpenGLRenderer()
109: (8)            if renderer is None:
110: (12)                self.renderer = CairoRenderer(
111: (16)                    camera_class=self.camera_class,
112: (16)                    skip_animations=self.skip_animations,
113: (12)                )
114: (8)            else:
115: (12)                self.renderer = renderer
116: (8)            self.renderer.init_scene(self)
117: (8)            self.mobjects = []
118: (8)            self.foreground_mobjects = []
119: (8)            if self.random_seed is not None:
120: (12)                random.seed(self.random_seed)
121: (12)                np.random.seed(self.random_seed)
122: (4)            @property
123: (4)            def camera(self):
124: (8)                return self.renderer.camera
125: (4)            def __deepcopy__(self, clone_from_id):
126: (8)                cls = self.__class__
127: (8)                result = cls.__new__(cls)
128: (8)                clone_from_id[id(self)] = result
129: (8)                for k, v in self.__dict__.items():
130: (12)                    if k in ["renderer", "time_progression"]:
131: (16)                        continue
132: (12)                    if k == "camera_class":
133: (16)                        setattr(result, k, v)
134: (12)                    setattr(result, k, copy.deepcopy(v, clone_from_id))
135: (8)                result.mobject_updater_lists = []
136: (8)                for mobject in self.mobjects:
137: (12)                    cloned_updaters = []
138: (12)                    for updater in mobject.updaters:
139: (16)                        free_variable_map = inspect.getclosurevars(updater).nonlocals
140: (16)                        cloned_co_freevars = []
141: (16)                        cloned_closure = []
142: (16)                        for free_variable_name in updater.__code__.co_freevars:
143: (20)                            free_variable_value =
free_variable_map[free_variable_name]
144: (20)                            if id(free_variable_value) not in clone_from_id:
145: (24)                                raise Exception(
146: (28)                                    f"{free_variable_name} is referenced from an
updater "
147: (28)                                    "but is not an attribute of the Scene, which isn't
"
148: (28)                                    "allowed.",
149: (24)                                )
150: (20)                                cloned_co_freevars.append(free_variable_name)
151: (20)                                cloned_closure.append(
152: (24)
types.CellType(clone_from_id[id(free_variable_value)]),
153: (20)                                )
154: (16)                                cloned_updater = types.FunctionType(
155: (20)
updater.__code__.replace(co_freevars=tuple(cloned_co_freevars)),
156: (20)                                updater.__globals__,
157: (20)                                updater.__name__,
158: (20)                                updater.__defaults__,
159: (20)                                tuple(cloned_closure),

```

```

160: (16) )
161: (16)     cloned_updaters.append(cloned_updater)
162: (12)     mobject_clone = clone_from_id[id(mobject)]
163: (12)     mobject_clone.updaters = cloned_updaters
164: (12)     if len(cloned_updaters) > 0:
165: (16)         result.mobject_updater_lists.append((mobject_clone,
cloned_updaters))
166: (8)     return result
167: (4) def render(self, preview: bool = False):
168: (8) """
169: (8)     Renders this Scene.
170: (8)     Parameters
171: (8)     -----
172: (8)     preview
173: (12)         If true, opens scene in a file viewer.
174: (8)
175: (8) self.setup()
176: (8) try:
177: (12)     self.construct()
178: (8) except EndSceneEarlyException:
179: (12)     pass
180: (8) except RerunSceneException as e:
181: (12)     self.remove(*self.mobjects)
182: (12)     self.renderer.clear_screen()
183: (12)     self.renderer.num_plays = 0
184: (12)     return True
185: (8) self.tear_down()
186: (8) self.renderer.scene_finished(self)
187: (8) if (
188: (12)     self.renderer.num_plays
189: (12)     or config["format"] == "png"
190: (12)     or config["save_last_frame"]
191: (8) ):
192: (12)     logger.info(
193: (16)     f"Rendered {str(self)}\nPlayed {self.renderer.num_plays}
animations",
194: (12)
195: (8)
196: (12)
197: (8)
198: (12)
199: (4) def setup(self):
200: (8) """
201: (8)     This is meant to be implemented by any scenes which
202: (8)     are commonly subclassed, and have some common setup
203: (8)     involved before the construct method is called.
204: (8)
205: (8)
206: (4) def tear_down(self):
207: (8) """
208: (8)     This is meant to be implemented by any scenes which
209: (8)     are commonly subclassed, and have some common method
210: (8)     to be invoked before the scene ends.
211: (8)
212: (8)
213: (4) def construct(self):
214: (8) """
215: (8)     Add content to the Scene.
216: (8)     From within :meth:`Scene.construct`, display mobjects on screen by
217: (8)     :meth:`Scene.add` and remove them from screen by calling
Play
218: (8)     All mobjects currently on screen are kept in :attr:`Scene.mobjects`.
219: (8)     animations by calling :meth:`Scene.play`.
Notes
220: (8)
221: (8)     Initialization code should go in :meth:`Scene.setup`. Termination
code should
222: (8)     go in :meth:`Scene.tear_down`.

```

```

223: (8)             Examples
224: (8)
225: (8)             -----
226: (8)             A typical manim script includes a class derived from :class:`Scene` with an
227: (8)             overridden :meth:`Scene.construct` method:
228: (12)            .. code-block:: python
229: (16)            class MyScene(Scene):
230: (20)            def construct(self):
231: (24)                self.play(Write(Text("Hello World!")))
232: (8)             See Also
233: (8)             -----
234: (8)             :meth:`Scene.setup`
235: (8)             :meth:`Scene.render`
236: (8)             :meth:`Scene.tear_down`
237: (8)             """
238: (4)             pass # To be implemented in subclasses
239: (8)             def next_section(
240: (8)                 self,
241: (8)                 name: str = "unnamed",
242: (8)                 type: str = DefaultSectionType.NORMAL,
243: (8)                 skip_animations: bool = False,
244: (4)             ) -> None:
245: (8)                 """Create separation here; the last section gets finished and a new
246: (8)                 section.
247: (8)                 Refer to :doc:`the documentation</tutorials/output_and_config>` on how
248: (8)                 to use sections.
249: (4)             self.renderer.file_writer.next_section(name, type, skip_animations)
250: (8)             def __str__(self):
251: (4)                 return self.__class__.__name__
252: (8)             def get_attrs(self, *keys: str):
253: (8)                 """
254: (8)                 Gets attributes of a scene given the attribute's identifier/name.
255: (8)                 Parameters
256: (8)                 -----
257: (12)                 *keys
258: (8)                 Name(s) of the argument(s) to return the attribute of.
259: (8)                 Returns
260: (8)                 -----
261: (12)                 list
262: (8)                 List of attributes of the passed identifiers.
263: (8)                 """
264: (4)             return [getattr(self, key) for key in keys]
265: (8)             def update_mobjects(self, dt: float):
266: (8)                 """
267: (8)                 Begins updating all mobjects in the Scene.
268: (8)                 Parameters
269: (8)                 -----
270: (12)                 dt
271: (8)                 Change in time between updates. Defaults (mostly) to
272: (8)                 """
273: (12)                 for mobject in self.mobjects:
274: (8)                     mobject.update(dt)
275: (8)             def update_meshes(self, dt):
276: (12)                 for obj in self.meshes:
277: (16)                     for mesh in obj.get_family():
278: (8)                         mesh.update(dt)
279: (8)             def update_self(self, dt: float):
280: (8)                 """
281: (8)                 Run all scene updater functions.
282: (8)                 Among all types of update functions (mobject updaters, mesh updaters, scene updaters), scene update functions are called last.
283: (8)                 Parameters
284: (8)                 -----
285: (12)                 dt
286: (8)                 Scene time since last update.

```

See Also

```

287: (8)             -----
288: (8)             :meth:`.Scene.add_updater`  

289: (8)             :meth:`.Scene.remove_updater`  

290: (8)             """  

291: (8)             for func in self.updaters:  

292: (12)             func(dt)  

293: (4) def should_update_mobjects(self) -> bool:  

294: (8)             """  

295: (8)             Returns True if the mobjects of this scene should be updated.  

296: (8)             In particular, this checks whether  

297: (8)             - the :attr:`always_update_mobjects` attribute of :class:`.Scene`  

298: (10)             is set to ``True``,  

299: (8)             - the :class:`.Scene` itself has time-based updaters attached,  

300: (8)             - any mobject in this :class:`.Scene` has time-based updaters  

attached.  

301: (8)             This is only called when a single Wait animation is played.  

302: (8)             """  

303: (8)             wait_animation = self.animations[0]  

304: (8)             if wait_animation.is_static_wait is None:  

305: (12)             should_update = (  

306: (16)                 self.always_update_mobjects  

307: (16)                 or self.updaters  

308: (16)                 or wait_animation.stop_condition is not None  

309: (16)                 or any(  

310: (20)                     mob.has_time_based_updater()  

311: (20)                     for mob in self.get_mobject_family_members()  

312: (16)                 )  

313: (12)             )  

314: (12)             wait_animation.is_static_wait = not should_update  

315: (8)             return not wait_animation.is_static_wait  

316: (4) def get_top_level_mobjects(self):  

317: (8)             """  

318: (8)             Returns all mobjects which are not submobjects.  

319: (8)             Returns  

320: (8)             -----  

321: (8)             list  

322: (12)             List of top level mobjects.  

323: (8)             """  

324: (8)             families = [m.get_family() for m in self.mobjects]  

325: (8)             def is_top_level(mobject):  

326: (12)                 num_families = sum((mob in family) for family in families)  

327: (12)                 return num_families == 1  

328: (8)             return list(filter(is_top_level, self.mobjects))  

329: (4)             def get_mobject_family_members(self):  

330: (8)             """  

331: (8)             Returns list of family-members of all mobjects in scene.  

332: (8)             If a Circle() and a VGroup(Rectangle(),Triangle()) were added,  

333: (8)             it returns not only the Circle(), Rectangle() and Triangle(), but  

334: (8)             also the VGroup() object.  

335: (8)             Returns  

336: (8)             -----  

337: (8)             list  

338: (12)             List of mobject family members.  

339: (8)             """  

340: (8)             if config.renderer == RendererType.OPENGL:  

341: (12)                 family_members = []  

342: (12)                 for mob in self.mobjects:  

343: (16)                     family_members.extend(mob.get_family())  

344: (12)                 return family_members  

345: (8)             elif config.renderer == RendererType.CAIRO:  

346: (12)                 return extract_mobject_family_members(  

347: (16)                     self.mobjects,  

348: (16)                     use_z_index=self.renderer.camera.use_z_index,  

349: (12)                 )  

350: (4)             def add(self, *mobjects: Mobject):  

351: (8)             """  

352: (8)             Mobjects will be displayed, from background to  

353: (8)             foreground in the order with which they are added.  

354: (8)             Parameters

```

```

355: (8)           -----
356: (8)           *mobjects
357: (12)          Mobjects to add.
358: (8)          Returns
359: (8)          -----
360: (8)          Scene
361: (12)          The same scene after adding the Mobjects in.
362: (8)          """
363: (8)          if config.renderer == RendererType.OPENGL:
364: (12)             new_mobjects = []
365: (12)             new_meshes = []
366: (12)             for mobject_or_mesh in mobjects:
367: (16)               if isinstance(mobject_or_mesh, Object3D):
368: (20)                 new_meshes.append(mobject_or_mesh)
369: (16)               else:
370: (20)                 new_mobjects.append(mobject_or_mesh)
371: (12)             self.remove(*new_mobjects)
372: (12)             self.mobjects += new_mobjects
373: (12)             self.remove(*new_meshes)
374: (12)             self.meshes += new_meshes
375: (8)           elif config.renderer == RendererType.CAIRO:
376: (12)             mobjects = [*mobjects, *self.foreground_mobjects]
377: (12)             self.restructure_mobjects(to_remove=mobjects)
378: (12)             self.mobjects += mobjects
379: (12)             if self.moving_mobjects:
380: (16)               self.restructure_mobjects(
381: (20)                 to_remove=mobjects,
382: (20)                 mobject_list_name="moving_mobjects",
383: (16)               )
384: (16)               self.moving_mobjects += mobjects
385: (8)           return self
386: (4)           def add_mobjects_from_animations(self, animations):
387: (8)             curr_mobjects = self.get_mobject_family_members()
388: (8)             for animation in animations:
389: (12)               if animation.is_introducer():
390: (16)                 continue
391: (12)               mob = animation.mobject
392: (12)               if mob is not None and mob not in curr_mobjects:
393: (16)                 self.add(mob)
394: (16)                 curr_mobjects += mob.get_family()
395: (4)           def remove(self, *mobjects: Mobject):
396: (8)           """
397: (8)           Removes mobjects in the passed list of mobjects
398: (8)           from the scene and the foreground, by removing them
399: (8)           from "mobjects" and "foreground_mobjects"
400: (8)           Parameters
401: (8)           -----
402: (8)           *mobjects
403: (12)           The mobjects to remove.
404: (8)           """
405: (8)           if config.renderer == RendererType.OPENGL:
406: (12)             mobjects_to_remove = []
407: (12)             meshes_to_remove = set()
408: (12)             for mobject_or_mesh in mobjects:
409: (16)               if isinstance(mobject_or_mesh, Object3D):
410: (20)                 meshes_to_remove.add(mobject_or_mesh)
411: (16)               else:
412: (20)                 mobjects_to_remove.append(mobject_or_mesh)
413: (12)             self.mobjects =
restructure_list_to_exclude_certain_family_members(
414: (16)               self.mobjects,
415: (16)               mobjects_to_remove,
416: (12)             )
417: (12)             self.meshes = list(
418: (16)               filter(lambda mesh: mesh not in set(meshes_to_remove),
self.meshes),
419: (12)             )
420: (12)             return self
421: (8)           elif config.renderer == RendererType.CAIRO:

```

```

422: (12)             for list_name in "mobjects", "foreground_mobjects":
423: (16)                 self.restructure_mobjects(mobjects, list_name, False)
424: (12)             return self
425: (4)             def replace(self, old_mobject: Mobject, new_mobject: Mobject) -> None:
426: (8)                 """Replace one mobject in the scene with another, preserving draw
order.
427: (8)                 If ``old_mobject`` is a submobject of some other Mobject (e.g. a
428: (8) :class:`.Group`), the new_mobject will replace it inside the group,
429: (8) without otherwise changing the parent mobject.
430: (8)             Parameters
431: (8)             -----
432: (8)             old_mobject
433: (12)                 The mobject to be replaced. Must be present in the scene.
434: (8)             new_mobject
435: (12)                 A mobject which must not already be in the scene.
436: (8)
437: (8)             if old_mobject is None or new_mobject is None:
438: (12)                 raise ValueError("Specified mobjects cannot be None")
439: (8)
440: (12)             def replace_in_list(
441: (8)                 mobj_list: list[Mobject], old_m: Mobject, new_m: Mobject
442: (12)             ) -> bool:
443: (16)                 for i in range(0, len(mobj_list)):
444: (20)                     if mobj_list[i] == old_m:
445: (20)                         mobj_list[i] = new_m
446: (12)                         return True
447: (16)                     for mob in mobj_list: # noqa: SIM110
448: (20)                         if replace_in_list(mob.submobjects, old_m, new_m):
449: (20)                             return True
450: (12)                     return False
451: (12)             replaced = replace_in_list(
452: (8)                 self.mobjects, old_mobject, new_mobject
453: (8)             ) or replace_in_list(self.foreground_mobjects, old_mobject,
new_mobject)
454: (8)             if not replaced:
455: (12)                 raise ValueError(f"Could not find {old_mobject} in scene")
456: (8)
457: (8)             def add_updater(self, func: Callable[[float], None]) -> None:
458: (8)                 """Add an update function to the scene.
459: (8)                 The scene updater functions are run every frame,
and they are the last type of updaters to run.
460: (8)                 .. WARNING::
461: (12)                     When using the Cairo renderer, scene updaters that
462: (12)                     modify mobjects are not detected in the same way
463: (12)                     that mobject updaters are. To be more concrete,
464: (12)                     a mobject only modified via a scene updater will
465: (12)                     not necessarily be added to the list of *moving
466: (12)                     mobjects* and thus might not be updated every frame.
467: (8)                     TL;DR: Use mobject updaters to update mobjects.
468: (8)
469: (8)
470: (12)             Parameters
471: (12)             -----
472: (12)             func
473: (8)                 The updater function. It takes a float, which is the
474: (8)                 time difference since the last update (usually equal
475: (8)                 to the frame rate).
476: (8)
477: (8)
478: (8)             See also
479: (4)             -----
480: (8)             :meth:`.Scene.remove_updater`
481: (8)             :meth:`.Scene.update_self`
482: (8)
483: (8)
484: (12)             self.updaters.append(func)
485: (8)
486: (8)
487: (8)             def remove_updater(self, func: Callable[[float], None]) -> None:
488: (8)                 """Remove an update function from the scene.

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
489: (8)             """
490: (8)             self.updaters = [f for f in self.updaters if f is not func]
491: (4)             def restructure_mobjects(
492: (8)                 self,
493: (8)                 to_remove: Mobject,
494: (8)                 mobject_list_name: str = "mobjects",
495: (8)                 extract_families: bool = True,
496: (4)             ):
497: (8)                 """
498: (8)             tl:wr
499: (12)             If your scene has a Group(), and you removed a mobject from the
Group,
500: (12)             this dissolves the group and puts the rest of the mobjects
501: (12)             in self.mobjects or self.foreground_mobjects.
502: (8)             In cases where the scene contains a group, e.g. Group(m1, m2, m3), but
one
503: (8)             mobjects
504: (8)             will be edited to contain other submobjects, but not m1, e.g. it will
now
505: (8)             insert m2 and m3 to where the group once was.
506: (8)             Parameters
507: (8)             -----
508: (8)             to_remove
509: (12)             The Mobject to remove.
510: (8)             mobject_list_name
511: (12)             The list of mobjects ("mobjects", "foreground_mobjects" etc) to
remove from.
512: (8)             extract_families
513: (12)             Whether the mobject's families should be recursively extracted.
514: (8)             Returns
515: (8)             -----
516: (8)             Scene
517: (12)             The Scene mobject with restructured Mobjects.
518: (8)             """
519: (8)             if extract_families:
520: (12)                 to_remove = extract_mobject_family_members(
521: (16)                     to_remove,
522: (16)                     use_z_index=self.renderer.camera.use_z_index,
523: (12)                 )
524: (8)                 _list = getattr(self, mobject_list_name)
525: (8)                 new_list = self.get_restructured_mobject_list(_list, to_remove)
526: (8)                 setattr(self, mobject_list_name, new_list)
527: (8)                 return self
528: (4)             def get_restructured_mobject_list(self, mobjects: list, to_remove: list):
529: (8)             """
530: (8)             Given a list of mobjects and a list of mobjects to be removed, this
531: (8)             filters out the removable mobjects from the list of mobjects.
532: (8)             Parameters
533: (8)             -----
534: (8)             mobjects
535: (12)             The Mobjects to check.
536: (8)             to_remove
537: (12)             The list of mobjects to remove.
538: (8)             Returns
539: (8)             -----
540: (8)             list
541: (12)             The list of mobjects with the mobjects to remove removed.
542: (8)             """
543: (8)             new_mobjects = []
544: (8)             def add_safe_mobjects_from_list(list_to_examine, set_to_remove):
545: (12)                 for mob in list_to_examine:
546: (16)                     if mob in set_to_remove:
547: (20)                         continue
548: (16)                     intersect = set_to_remove.intersection(mob.get_family())
549: (16)                     if intersect:
550: (20)                         add_safe_mobjects_from_list(mob.submobjects, intersect)
551: (16)                     else:

```

```

552: (20)                                     new_mobjects.append(mob)
553: (8)                                      add_safe_mobjects_from_list(mobjects, set(to_remove))
554: (8)                                      return new_mobjects
555: (4)                                     def add_foreground_mobjects(self, *mobjects: Mobject):
556: (8)   """
557: (8)   Adds mobjects to the foreground, and internally to the list
558: (8)   foreground_mobjects, and mobjects.
559: (8)   Parameters
560: (8)   -----
561: (8)   *mobjects
562: (12)   The Mobjects to add to the foreground.
563: (8)   Returns
564: (8)   -----
565: (8)   Scene
566: (12)   The Scene, with the foreground mobjects added.
567: (8)   """
568: (8)   self.foreground_mobjects = list_update(self.foreground_mobjects,
mobjects)
569: (8)   self.add(*mobjects)
570: (8)   return self
571: (4)                                     def add_foreground_mobject(self, mobject: Mobject):
572: (8)   """
573: (8)   Adds a single mobject to the foreground, and internally to the list
574: (8)   foreground_mobjects, and mobjects.
575: (8)   Parameters
576: (8)   -----
577: (8)   mobject
578: (12)   The Mobject to add to the foreground.
579: (8)   Returns
580: (8)   -----
581: (8)   Scene
582: (12)   The Scene, with the foreground mobject added.
583: (8)   """
584: (8)   return self.add_foreground_mobjects(mobject)
585: (4)                                     def remove_foreground_mobjects(self, *to_remove: Mobject):
586: (8)   """
587: (8)   Removes mobjects from the foreground, and internally from the list
588: (8)   foreground_mobjects.
589: (8)   Parameters
590: (8)   -----
591: (8)   *to_remove
592: (12)   The mobject(s) to remove from the foreground.
593: (8)   Returns
594: (8)   -----
595: (8)   Scene
596: (12)   The Scene, with the foreground mobjects removed.
597: (8)   """
598: (8)   self.restructure_mobjects(to_remove, "foreground_mobjects")
599: (8)   return self
600: (4)                                     def remove_foreground_mobject(self, mobject: Mobject):
601: (8)   """
602: (8)   Removes a single mobject from the foreground, and internally from the
list
603: (8)   foreground_mobjects.
604: (8)   Parameters
605: (8)   -----
606: (8)   mobject
607: (12)   The mobject to remove from the foreground.
608: (8)   Returns
609: (8)   -----
610: (8)   Scene
611: (12)   The Scene, with the foreground mobject removed.
612: (8)   """
613: (8)   return self.remove_foreground_mobjects(mobject)
614: (4)                                     def bring_to_front(self, *mobjects: Mobject):
615: (8)   """
616: (8)   Adds the passed mobjects to the scene again,
617: (8)   pushing them to the front of the scene.
618: (8)   Parameters

```

```

619: (8)           -----
620: (8)           *mobjects
621: (12)          The mobject(s) to bring to the front of the scene.
622: (8)           Returns
623: (8)           -----
624: (8)           Scene
625: (12)          The Scene, with the mobjects brought to the front
626: (12)          of the scene.
627: (8)           """
628: (8)           self.add(*mobjects)
629: (8)           return self
630: (4)            def bring_to_back(self, *mobjects: Mobject):
631: (8)           """
632: (8)           Removes the mobject from the scene and
633: (8)           adds them to the back of the scene.
634: (8)           Parameters
635: (8)           -----
636: (8)           *mobjects
637: (12)          The mobject(s) to push to the back of the scene.
638: (8)           Returns
639: (8)           -----
640: (8)           Scene
641: (12)          The Scene, with the mobjects pushed to the back
642: (12)          of the scene.
643: (8)           """
644: (8)           self.remove(*mobjects)
645: (8)           self.mobjects = list(mobjects) + self.mobjects
646: (8)           return self
647: (4)            def clear(self):
648: (8)           """
649: (8)           Removes all mobjects present in self.mobjects
650: (8)           and self.foreground_mobjects from the scene.
651: (8)           Returns
652: (8)           -----
653: (8)           Scene
654: (12)          The Scene, with all of its mobjects in
655: (12)          self.mobjects and self.foreground_mobjects
656: (12)          removed.
657: (8)           """
658: (8)           self.mobjects = []
659: (8)           self.foreground_mobjects = []
660: (8)           return self
661: (4)            def get_moving_mobjects(self, *animations: Animation):
662: (8)           """
663: (8)           Gets all moving mobjects in the passed animation(s).
664: (8)           Parameters
665: (8)           -----
666: (8)           *animations
667: (12)          The animations to check for moving mobjects.
668: (8)           Returns
669: (8)           -----
670: (8)           list
671: (12)          The list of mobjects that could be moving in
672: (12)          the Animation(s)
673: (8)           """
674: (8)           animation_mobjects = [anim.mobject for anim in animations]
675: (8)           mobjects = self.get_mobject_family_members()
676: (8)           for i, mob in enumerate(mobjects):
677: (12)             update_possibilities = [
678: (16)               mob in animation_mobjects,
679: (16)               len(mob.get_family_updaters()) > 0,
680: (16)               mob in self.foreground_mobjects,
681: (12)             ]
682: (12)             if any(update_possibilities):
683: (16)               return mobjects[i:]
684: (8)           return []
685: (4)            def get_moving_and_static_mobjects(self, animations):
686: (8)           all_mobjects = list_update(self.mobjects, self.foreground_mobjects)
687: (8)           all_mobject_families = extract_mobject_family_members(

```

```

688: (12)                         all_mobjects,
689: (12)                         use_z_index=self.renderer.camera.use_z_index,
690: (12)                         only_those_with_points=True,
691: (8)                          )
692: (8)                          moving_mobjects = self.get_moving_mobjects(*animations)
693: (8)                          all_moving_mobject_families = extract_mobject_family_members(
694: (12)                            moving_mobjects,
695: (12)                            use_z_index=self.renderer.camera.use_z_index,
696: (8)                          )
697: (8)                          static_mobjects = list_difference_update(
698: (12)                            all_mobject_families,
699: (12)                            all_moving_mobject_families,
700: (8)                          )
701: (8)                          return all_moving_mobject_families, static_mobjects
702: (4)  def compile_animations(
703: (8)    self,
704: (8)    *args: Animation | Iterable[Animation] |
types.GeneratorType[Animation],
705: (8)    **kwargs,
706: (4)  ):
707: (8)  """
708: (8)  Creates _MethodAnimations from any _AnimationBuilders and updates
animation
709: (8)  kwargs with kwargs passed to play().
710: (8)  Parameters
711: (8)  -----
712: (8)  *args
713: (12)    Animations to be played.
714: (8)  **kwargs
715: (12)    Configuration for the call to play().
716: (8)  Returns
717: (8)  -----
718: (8)  Tuple[:class:`Animation`]
719: (12)    Animations to be played.
720: (8)  """
721: (8)  animations = []
722: (8)  arg_anims = flatten_iterable_parameters(args)
723: (8)  for arg in arg_anims:
724: (12)    try:
725: (16)      animations.append(prepare_animation(arg))
726: (12)    except TypeError:
727: (16)      if inspect.ismethod(arg):
728: (20)        raise TypeError(
729: (24)          "Passing Mobject methods to Scene.play is no longer"
730: (24)          " supported. Use Mobject.animate instead.",
731: (20)        )
732: (16)      else:
733: (20)        raise TypeError(
734: (24)          f"Unexpected argument {arg} passed to Scene.play().",
735: (20)        )
736: (8)      for animation in animations:
737: (12)        for k, v in kwargs.items():
738: (16)          setattr(animation, k, v)
739: (8)      return animations
740: (4)  def _get_animation_time_progression(
741: (8)    self, animations: list[Animation], duration: float
742: (4)  ):
743: (8)  """
744: (8)    You will hardly use this when making your own animations.
745: (8)    This method is for Manim's internal use.
746: (8)    Uses :func:`~.get_time_progression` to obtain a
747: (8)    CommandLine ProgressBar whose ``fill_time`` is
748: (8)    dependent on the qualities of the passed Animation,
749: (8)    Parameters
750: (8)    -----
751: (8)    animations
752: (12)      The list of animations to get
753: (12)      the time progression for.
754: (8)    duration

```

```

755: (12)                               duration of wait time
756: (8)                                Returns
757: (8)                                -----
758: (8)                                time_progression
759: (12)                                The CommandLine Progress Bar.
760: (8)                                """
761: (8)                                if len(animations) == 1 and isinstance(animations[0], Wait):
762: (12)                                    stop_condition = animations[0].stop_condition
763: (12)                                    if stop_condition is not None:
764: (16)  time_progression = self.get_time_progression(
765: (20)  duration,
766: (20)  f"Waiting for {stop_condition.__name__}",
767: (20)  n_iterations=-1, # So it doesn't show % progress
768: (20)  override_skip_animations=True,
769: (16)
770: (12)                                )
771: (16)                                else:
772: (20)                                    time_progression = self.get_time_progression(
773: (20)  duration,
774: (16)  f"Waiting {self.renderer.num_plays}",
775: (8)
776: (12)                                )
777: (16)                                else:
778: (16)                                    time_progression = self.get_time_progression(
779: (20)  duration,
780: (24)  "".join(
781: (24)  [
782: (24)  f"Animation {self.renderer.num_plays}: ",
783: (20)  str(animations[0]),
784: (16)  (" , etc." if len(animations) > 1 else ""),
785: (12)
786: (8)                                ],
787: (4)                                )
788: (8)                                return time_progression
789: (8)                                def get_time_progression(
790: (8)                                    self,
791: (8)                                    run_time: float,
792: (8)                                    description,
793: (4)                                    n_iterations: int | None = None,
794: (8)                                    override_skip_animations: bool = False,
795: (4):
796: (8)                                """
797: (8)                                You will hardly use this when making your own animations.
798: (8)                                This method is for Manim's internal use.
799: (8)                                Returns a CommandLine ProgressBar whose ``fill_time``
800: (8)                                is dependent on the ``run_time`` of an animation,
801: (8)                                the iterations to perform in that animation
802: (8)                                and a bool saying whether or not to consider
803: (8)                                the skipped animations.
804: (8)                                Parameters
805: (12)                                -----
806: (8)                                run_time
807: (12)                                    The ``run_time`` of the animation.
808: (8)                                n_iterations
809: (12)                                    The number of iterations in the animation.
810: (8)                                override_skip_animations
811: (8)                                    Whether or not to show skipped animations in the progress bar.
812: (8)                                Returns
813: (12)                                -----
814: (8)                                time_progression
815: (8)                                    The CommandLine Progress Bar.
816: (12)
817: (8)
818: (12)
819: (12)
820: (8)
821: (12)
822: (12)
823: (12)

```

```

824: (12)             leave=config["progress_bar"] == "leave",
825: (12)             ascii=True if platform.system() == "Windows" else None,
826: (12)             disable=config["progress_bar"] == "none",
827: (8)
828: (8)
829: (4)         def get_run_time(self, animations: list[Animation]):
830: (8)             """
831: (8)                 Gets the total run time for a list of animations.
832: (8)             Parameters
833: (8)             -----
834: (8)             animations
835: (12)                 A list of the animations whose total
836: (12)                 ``run_time`` is to be calculated.
837: (8)             Returns
838: (8)             -----
839: (8)             float
840: (12)                 The total ``run_time`` of all of the animations in the list.
841: (8)
842: (8)             if len(animations) == 1 and isinstance(animations[0], Wait):
843: (12)                 return animations[0].duration
844: (8)             else:
845: (12)                 return np.max([animation.run_time for animation in animations])
846: (4)         def play(
847: (8)             self,
848: (8)             *args: Animation | Iterable[Animation] |
types.GeneratorType[Animation],
849: (8)                 subcaption=None,
850: (8)                 subcaption_duration=None,
851: (8)                 subcaption_offset=0,
852: (8)                 **kwargs,
853: (4):
854: (8)             r"""Plays an animation in this scene.
855: (8)             Parameters
856: (8)             -----
857: (8)             args
858: (12)                 Animations to be played.
859: (8)             subcaption
860: (12)                 The content of the external subcaption that should
861: (12)                 be added during the animation.
862: (8)             subcaption_duration
863: (12)                 The duration for which the specified subcaption is
864: (12)                 added. If ``None`` (the default), the run time of the
865: (12)                 animation is taken.
866: (8)             subcaption_offset
867: (12)                 An offset (in seconds) for the start time of the
868: (12)                 added subcaption.
869: (8)             kwargs
870: (12)                 All other keywords are passed to the renderer.
871: (8)
872: (8)             if (
873: (12)                 self.interactive_mode
874: (12)                 and config.renderer == RendererType.OPENGL
875: (12)                 and threading.current_thread().name != "MainThread"
876: (8):
877: (12)
878: (16)
879: (20)
880: (20)
881: (20)
882: (16)
883: (12)
884: (12)
885: (16)
886: (20)
887: (20)
888: (20)
889: (16)
890: (12)
891: (12)

```

```

892: (8)                     start_time = self.renderer.time
893: (8)                     self.renderer.play(self, *args, **kwargs)
894: (8)                     run_time = self.renderer.time - start_time
895: (8)                     if subcaption:
896: (12)                         if subcaption_duration is None:
897: (16)                             subcaption_duration = run_time
898: (12)                         self.add_subcaption(
899: (16)                             content=subcaption,
900: (16)                             duration=subcaption_duration,
901: (16)                             offset=-run_time + subcaption_offset,
902: (12)                         )
903: (4)                     def wait(
904: (8)                         self,
905: (8)                         duration: float = DEFAULT_WAIT_TIME,
906: (8)                         stop_condition: Callable[[], bool] | None = None,
907: (8)                         frozen_frame: bool | None = None,
908: (4)                     ):
909: (8)                         """Plays a "no operation" animation.
910: (8)                         Parameters
911: (8)                         -----
912: (8)                         duration
913: (12)                             The run time of the animation.
914: (8)                         stop_condition
915: (12)                             A function without positional arguments that is evaluated every
time
916: (12)                             a frame is rendered. The animation only stops when the return
value
917: (12)                             of the function is truthy, or when the time specified in
``duration``
918: (12)                             passes.
919: (8)                         frozen_frame
920: (12)                             If True, updater functions are not evaluated, and the animation
a frozen frame. If False, updater functions are called and frames
are rendered as usual. If None (the default), the scene tries to
determine whether or not the frame is frozen on its own.
921: (8)                         See also
922: (8)                         -----
923: (8)                         :class:`.Wait`, :meth:`.should_mobjects_update`  

924: (8)                         """
925: (8)                     self.play(
926: (8)                         Wait(
927: (8)                             run_time=duration,
928: (8)                             stop_condition=stop_condition,
929: (12)                             frozen_frame=frozen_frame,
930: (16)                         )
931: (16)                     )
932: (16)                     )
933: (12)                 )
934: (8)             )
935: (4)             def pause(self, duration: float = DEFAULT_WAIT_TIME):
936: (8)                 """Pauses the scene (i.e., displays a frozen frame).
937: (8)                 This is an alias for :meth:`.wait` with ``frozen_frame``
set to ``True``.
938: (8)                 Parameters
939: (8)                 -----
940: (8)                 duration
941: (8)                     The duration of the pause.
942: (12)                 See also
943: (8)                 -----
944: (8)                 :meth:`.wait`, :class:`.Wait`  

945: (8)                 """
946: (8)                 self.wait(duration=duration, frozen_frame=True)
947: (8)             def wait_until(self, stop_condition: Callable[[], bool], max_time: float =
60):
948: (4)                 """Wait until a condition is satisfied, up to a given maximum
duration.
949: (8)                 Parameters
950: (8)                 -----
951: (8)                 stop_condition
952: (8)                     A function with no arguments that determines whether or not the
scene should keep waiting.
953: (12)
954: (12)

```

```

955: (8)             max_time
956: (12)            The maximum wait time in seconds.
957: (8)
958: (8)             """
959: (4)              self.wait(max_time, stop_condition=stop_condition)
960: (8)            self,
961: (8)            *animations: Animation | Iterable[Animation] |
types.GeneratorType[Animation],
962: (8)            **play_kwarg,
963: (4):
964: (8)            """Given a list of animations, compile the corresponding
965: (8)            static and moving mobjects, and gather the animation durations.
966: (8)            This also begins the animations.
967: (8)            Parameters
968: (8)            -----
969: (8)            animations
970: (12)           Animation or mobject with mobject method and params
971: (8)
972: (12)           play_kwarg
973: (12)             Named parameters affecting what was passed in ``animations``,
974: (8)               e.g. ``run_time``, ``lag_ratio`` and so on.
975: (8)            Returns
976: (8)            -----
977: (8)            self, None
978: (8)             None if there is nothing to play, or self otherwise.
979: (8)            """
980: (12)           if len(animations) == 0:
981: (8)             raise ValueError("Called Scene.play with no animations")
982: (8)           self.animations = self.compile_animations(*animations, **play_kwarg)
983: (8)           self.add_mobjects_from_animations(self.animations)
984: (8)           self.last_t = 0
985: (8)           self.stop_condition = None
986: (8)           self.moving_mobjects = []
987: (8)           self.static_mobjects = []
988: (12)           if len(self.animations) == 1 and isinstance(self.animations[0], Wait):
989: (16)             if self.should_update_mobjects():
990: (16)               self.update_mobjects(dt=0) # Any problems with this?
991: (12)             self.stop_condition = self.animations[0].stop_condition
992: (16)
993: (16)
994: (16)
995: (8)             else:
996: (8)               self.duration = self.animations[0].duration
997: (4)               self.animations[0].is_static_wait = True
998: (8)             return None
999: (8)           self.duration = self.get_run_time(self.animations)
1000: (12)
1001: (12)
1002: (8)
1003: (12)
1004: (16)
1005: (16)
1006: (12)
1007: (4)
1008: (8)
(generally a Wait)."""
1009: (8)
1010: (12)
1011: (12)
1012: (12)
1013: (8)
1014: (4)
1015: (8)
1016: (8)
1017: (8)
1018: (8)
1019: (8)
1020: (8)
1021: (8)

      def begin_animations(self) -> None:
        """Start the animations of the scene."""
        for animation in self.animations:
          animation._setup_scene(self)
          animation.begin()
        if config.renderer == RendererType.CAIRO:
          (
            self.moving_mobjects,
            self.static_mobjects,
          ) = self.get_moving_and_static_mobjects(self.animations)
      def is_current_animation_frozen_frame(self) -> bool:
        """Returns whether the current animation produces a static frame
        (generally a Wait)."""
        return (
          isinstance(self.animations[0], Wait)
          and len(self.animations) == 1
          and self.animations[0].is_static_wait
        )
      def play_internal(self, skip_rendering: bool = False):
        """
        This method is used to prep the animations for rendering,
        apply the arguments and parameters required to them,
        render them, and write them to the video file.
        Parameters
        -----
        skip_rendering

```

```

1022: (12)             Whether the rendering should be skipped, by default False
1023: (8)
1024: (8)
1025: (8)
1026: (12)
1027: (12)
1028: (8)
1029: (8)
1030: (12)
1031: (12)
1032: (16)
1033: (12)
1034: (16)
1035: (16)
1036: (8)
1037: (12)
1038: (12)
1039: (8)
1040: (12)
1041: (8)
1042: (8)
1043: (4)
1044: (8)
1045: (12)
1046: (8)
1047: (12)
1048: (16)
1049: (12)           enabled",
1050: (12)
1051: (8)
1052: (12)
1053: (12)           enabled")
1054: (8)
1055: (12)
1056: (16)
1057: (16)
1058: (12)
1059: (12)
1060: (8)
1061: (12)
1062: (12)
1063: (8)
1064: (12)
1065: (12)
1066: (8)
1067: (4)
1068: (8)
1069: (8)
1070: (8)
1071: (8)
1072: (12)
1073: (8)
1074: (8)
1075: (12)
1076: (12)
1077: (16)
1078: (20)
1079: (12)
1080: (12)
1081: (12)
1082: (16)
1083: (16)
1084: (12)
1085: (12)
1086: (12)

        """
        self.duration = self.get_run_time(self.animations)
        self.time_progression = self._get_animation_time_progression(
            self.animations,
            self.duration,
        )
        for t in self.time_progression:
            self.update_to_time(t)
            if not skip_rendering and not self.skip_animation_preview:
                self.renderer.render(self, t, self.moving_mobjects)
            if self.stop_condition is not None and self.stop_condition():
                self.time_progression.close()
                break
        for animation in self.animations:
            animation.finish()
            animation.clean_up_from_scene(self)
        if not self.renderer.skip_animations:
            self.update_mobjects(0)
        self.renderer.static_image = None
        self.time_progression.close()
    def check_interactive_embed_is_valid(self):
        if config["force_window"]:
            return True
        if self.skip_animation_preview:
            logger.warning(
                "Disabling interactive embed as 'skip_animation_preview' is
            )
            return False
        elif config["write_to_movie"]:
            logger.warning("Disabling interactive embed as 'write_to_movie' is
                           return False
        elif config["format"]:
            logger.warning(
                "Disabling interactive embed as '--format' is set as "
                + config["format"],
            )
            return False
        elif not self.renderer.window:
            logger.warning("Disabling interactive embed as no window was
                           return False
        elif config.dry_run:
            logger.warning("Disabling interactive embed as dry_run is
                           return False
        return True
    def interactive_embed(self):
        """
        Like embed(), but allows for screen interaction.
        """
        if not self.check_interactive_embed_is_valid():
            return
        self.interactive_mode = True
    def ipython(shell, namespace):
        import manim.opengl
        def load_module_into_namespace(module, namespace):
            for name in dir(module):
                namespace[name] = getattr(module, name)
        load_module_into_namespace(manim, namespace)
        load_module_into_namespace(manim.opengl, namespace)
    def embedded_rerun(*args, **kwargs):
        self.queue.put(("rerun_keyboard", args, kwargs))
        shell.exiter()
    namespace["rerun"] = embedded_rerun
    shell(local_ns=namespace)
    self.queue.put(("exit_keyboard", [], {}))

```

```

1087: (8)             def get_embedded_method(method_name):
1088: (12)             return lambda *args, **kwargs: self.queue.put((method_name, args,
1089: (8)               kwargs))
1090: (8)               local_namespace = inspect.currentframe().f_back.f_locals
1091: (12)             for method in ("play", "wait", "add", "remove"):
1092: (12)                 embedded_method = get_embedded_method(method)
1093: (8)                   local_namespace[method] = embedded_method
1094: (8)             from sqlite3 import connect
1095: (8)             from IPython.core.getipython import get_ipython
1096: (8)             from IPython.terminal.embed import InteractiveShellEmbed
1097: (8)             from traits.config import Config
1098: (8)             cfg = Config()
1099: (8)             cfg.TerminalInteractiveShell.confirm_exit = False
1100: (12)             if get_ipython() is None:
1101: (8)                 shell = InteractiveShellEmbed.instance(config=cfg)
1102: (12)             else:
1103: (8)                 shell = InteractiveShellEmbed(config=cfg)
1104: (8)             hist = get_ipython().history_manager
1105: (8)             hist.db = connect(hist.hist_file, check_same_thread=False)
1106: (12)             keyboard_thread = threading.Thread(
1107: (12)                 target=ipython,
1108: (8)                   args=(shell, local_namespace),
1109: (8)             )
1110: (12)             if not shell.pt_app:
1111: (8)                 keyboard_thread.daemon = True
1112: (8)             keyboard_thread.start()
1113: (12)             if self.dearpygui_imported and config["enable_gui"]:
1114: (16)                 if not dpg.is_dearpygui_running():
1115: (20)                     gui_thread = threading.Thread(
1116: (20)                         target=configure_pygui,
1117: (20)                           args=(self.renderer, self.widgets),
1118: (16)                             kwargs={"update": False},
1119: (16)                         )
1120: (12)                     gui_thread.start()
1121: (16)                 else:
1122: (8)                     configure_pygui(self.renderer, self.widgets, update=True)
1123: (8)                     self.camera.model_matrix = self.camera.default_model_matrix
1124: (4)                     self.interact(shell, keyboard_thread)
1125: (8)             def interact(self, shell, keyboard_thread):
1126: (8)                 event_handler = RerunSceneHandler(self.queue)
1127: (8)                 file_observer = Observer()
1128: recursive=True)
1129: (8)                 file_observer.schedule(event_handler, config["input_file"],
1130: (8)                     )
1131: (8)                 file_observer.start()
1132: (8)                 self.quit_interaction = False
1133: (8)                 keyboard_thread_needs_join = shell.pt_app is not None
1134: (12)                 assert self.queue.qsize() == 0
1135: (16)                 last_time = time.time()
1136: (16)                 while not (self.renderer.window.is_closing or self.quit_interaction):
1137: (20)                     if not self.queue.empty():
1138: (24)                         tup = self.queue.get_nowait()
1139: (28)                         if tup[0].startswith("rerun"):
1140: (24)                             if not tup[0].endswith("keyboard"):
1141: (24)                                 if shell.pt_app:
1142: (20)                                     shell.pt_app.app.exit(exception=EOFError)
1143: (20)                                     file_observer.unschedule_all()
1144: (20)                                     raise RerunSceneException
1145: (24)                                     keyboard_thread.join()
1146: (28)                                     kwargs = tup[2]
1147: (24)                                     if "from_animation_number" in kwargs:
1148: (20)   config["from_animation_number"] = kwargs[
1149: (20)   "from_animation_number"
1150: (20)   ]
1151: (16)   keyboard_thread.join()
1152: (20)   file_observer.unschedule_all()
1153: (24)   raise RerunSceneException
1154: (20)                                     elif tup[0].startswith("exit"):
1155: (20)   if not tup[0].endswith("keyboard") and shell.pt_app:
1156: (24)   shell.pt_app.app.exit(exception=EOFError)

```

```

1154: (20)                         keyboard_thread.join()
1155: (20)                         while self.queue.qsize() > 0:
1156: (24)                           self.queue.get()
1157: (20)                           keyboard_thread_needs_join = False
1158: (20)                           break
1159: (16)                           else:
1160: (20)                             method, args, kwargs = tup
1161: (20)                             getattr(self, method)(*args, **kwargs)
1162: (12)                           else:
1163: (16)                             self.renderer.animation_start_time = 0
1164: (16)                             dt = time.time() - last_time
1165: (16)                             last_time = time.time()
1166: (16)                             self.renderer.render(self, dt, self.moving_mobjects)
1167: (16)                             self.update_mobjects(dt)
1168: (16)                             self.update_meshes(dt)
1169: (16)                             self.update_self(dt)
1170: (8)                            if shell is not None and keyboard_thread_needs_join:
1171: (12)                              shell.pt_app.app.exit(exception=EOFError)
1172: (12)                              keyboard_thread.join()
1173: (12)                              while self.queue.qsize() > 0:
1174: (16)                                self.queue.get()
1175: (8)                                file_observer.stop()
1176: (8)                                file_observer.join()
1177: (8)                                if self.dearpygui_imported and config["enable_gui"]:
1178: (12)                                  dpg.stop_dearpygui()
1179: (8)                                if self.renderer.window.is_closing:
1180: (12)                                  self.renderer.window.destroy()
1181: (4) def embed(self):
1182: (8)   if not config["preview"]:
1183: (12)     logger.warning("Called embed() while no preview window is
available.")
1184: (12)
1185: (8)
1186: (12)
1187: (12)
1188: (8)
1189: (8)
1190: (8)
1191: (8)
1192: (8)
1193: (12)
1194: (12)
self.moving_mobjects),
1195: (8)
1196: (8)
1197: (8)
1198: (12)
1199: (12)
1200: (12)
1201: (12)
1202: (12)
1203: (8)
1204: (12)
1205: (8)
1206: (8)
1207: (4)
1208: (8)
1209: (8)
1210: (8)
1211: (12)
1212: (12)
1213: (12)
1214: (8)
1215: (8)
1216: (8)
1217: (4)
1218: (8)
1219: (4)
1220: (8)
                               local_ns = inspect.currentframe().f_back.f_locals
for method in (
    "play",
    "wait",
    "add",
    "remove",
    "interact",
):
    local_ns[method] = getattr(self, method)
shell(local_ns=local_ns, stack_depth=2)
raise Exception("Exiting scene.")
def update_to_time(self, t):
    dt = t - self.last_t
    self.last_t = t
    for animation in self.animations:
        animation.update_mobjects(dt)
        alpha = t / animation.run_time
        animation.interpolate(alpha)
    self.update_mobjects(dt)
    self.update_meshes(dt)
    self.update_self(dt)
def add_subcaption(
    self, content: str, duration: float = 1, offset: float = 0
) -> None:
    r"""Adds an entry in the corresponding subcaption file

```

```

1221: (8)                                at the current time stamp.
1222: (8)                                The current time stamp is obtained from ``Scene.renderer.time``.
1223: (8)                                Parameters
1224: (8)                                -----
1225: (8)                                content
1226: (12)                               The subcaption content.
1227: (8)                                duration
1228: (12)                               The duration (in seconds) for which the subcaption is shown.
1229: (8)                                offset
1230: (12)                               This offset (in seconds) is added to the starting time stamp
1231: (12)                               of the subcaption.
1232: (8)                                Examples
1233: (8)                                -----
1234: (8)                                This example illustrates both possibilities for adding
1235: (8)                                subcaptions to Manimations::
1236: (12)                               class SubcaptionExample(Scene):
1237: (16)                                 def construct(self):
1238: (20)                                   square = Square()
1239: (20)                                   circle = Circle()
1240: (20)                                   self.add_subcaption("Hello square!", duration=1)
1241: (20)                                   self.play(Create(square))
1242: (20)                                   self.play(
1243: (24)                                     Transform(square, circle),
1244: (24)                                     subcaption="The square transforms."
1245: (20)                                   )
1246: (8)                                 """
1247: (8)                                 subtitle = srt.Subtitle(
1248: (12)                                   index=len(self.renderer.file_writer.subcaptions),
1249: (12)                                   content=content,
1250: (12)                                   start=datetime.timedelta(seconds=float(self.renderer.time +
offset)),
1251: (12)                                   end=datetime.timedelta(
1252: (16)                                     seconds=float(self.renderer.time + offset + duration)
1253: (12)                                   ),
1254: (8)                                 )
1255: (8)                                 self.renderer.file_writer.subcaptions.append(subtitle)
1256: (4)                                def add_sound(
1257: (8)                                   self,
1258: (8)                                   sound_file: str,
1259: (8)                                   time_offset: float = 0,
1260: (8)                                   gain: float | None = None,
1261: (8)                                   **kwargs,
1262: (4)                                 ):
1263: (8)                                 """
1264: (8)                                 This method is used to add a sound to the animation.
1265: (8)                                Parameters
1266: (8)                                -----
1267: (8)                                sound_file
1268: (12)                               The path to the sound file.
1269: (8)                                time_offset
1270: (12)                               The offset in the sound file after which
1271: (12)                               the sound can be played.
1272: (8)                                gain
1273: (12)                               Amplification of the sound.
1274: (8)                                Examples
1275: (8)                                -----
1276: (8)                                .. manim:: SoundExample
1277: (12)                                 :no_autoplay:
1278: (12)                                 class SoundExample(Scene):
1279: (16)                                   def construct(self):
1280: (20)                                     dot = Dot().set_color(GREEN)
1281: (20)                                     self.add_sound("click.wav")
1282: (20)                                     self.add(dot)
1283: (20)                                     self.wait()
1284: (20)                                     self.add_sound("click.wav")
1285: (20)                                     dot.set_color(BLUE)
1286: (20)                                     self.wait()
1287: (20)                                     self.add_sound("click.wav")
1288: (20)                                     dot.set_color(RED)

```

```

1289: (20)                     self.wait()
1290: (8)             Download the resource for the previous example `here
<https://github.com/ManimCommunity/manim/blob/main/docs/source/_static/click.wav`_ .
1291: (8)
1292: (8)             if self.renderer.skip_animations:
1293: (12)                 return
1294: (8)                 time = self.renderer.time + time_offset
1295: (8)                 self.renderer.file_writer.add_sound(sound_file, time, gain, **kwargs)
1296: (4)             def on_mouse_motion(self, point, d_point):
1297: (8)                 self.mouse_point.move_to(point)
1298: (8)                 if SHIFT_VALUE in self.renderer.pressed_keys:
1299: (12)                     shift = -d_point
1300: (12)                     shift[0] *= self.camera.get_width() / 2
1301: (12)                     shift[1] *= self.camera.get_height() / 2
1302: (12)                     transform = self.camera.inverse_rotation_matrix
1303: (12)                     shift = np.dot(np.transpose(transform), shift)
1304: (12)                     self.camera.shift(shift)
1305: (4)             def on_mouse_scroll(self, point, offset):
1306: (8)                 if not config.use_projection_stroke_shaders:
1307: (12)                     factor = 1 + np.arctan(-2.1 * offset[1])
1308: (12)                     self.camera.scale(factor, about_point=self.camera_target)
1309: (8)             self.mouse_scroll_orbit_controls(point, offset)
1310: (4)             def on_key_press(self, symbol, modifiers):
1311: (8)                 try:
1312: (12)                     char = chr(symbol)
1313: (8)                 except OverflowError:
1314: (12)                     logger.warning("The value of the pressed key is too large.")
1315: (12)                     return
1316: (8)                 if char == "r":
1317: (12)                     self.camera.to_default_state()
1318: (12)                     self.camera_target = np.array([0, 0, 0], dtype=np.float32)
1319: (8)
1320: (12)                     self.quit_interaction = True
1321: (8)
1322: (12)                 else:
1323: (16)                     if char in self.key_to_function_map:
1324: (4)                         self.key_to_function_map[char]()
1325: (8)             def on_key_release(self, symbol, modifiers):
1326: (4)                 pass
1327: (8)             def on_mouse_drag(self, point, d_point, buttons, modifiers):
1328: (8)                 self.mouse_drag_point.move_to(point)
1329: (12)                 if buttons == 1:
1330: (12)                     self.camera.increment_theta(-d_point[0])
1331: (8)                     self.camera.increment_phi(d_point[1])
1332: (12)                 elif buttons == 4:
1333: (12)                     camera_x_axis = self.camera.model_matrix[:3, 0]
1334: (12)                     horizontal_shift_vector = -d_point[0] * camera_x_axis
1335: (12)                     vertical_shift_vector = -d_point[1] * np.cross(OUT, camera_x_axis)
vertical_shift_vector
1336: (12)                     total_shift_vector = horizontal_shift_vector +
1337: (8)                     self.camera.shift(1.1 * total_shift_vector)
1338: (4)                     self.mouse_drag_orbit_controls(point, d_point, buttons, modifiers)
1339: (8)             def mouse_scroll_orbit_controls(self, point, offset):
1340: (8)                 camera_to_target = self.camera_target - self.camera.get_position()
1341: (8)                 camera_to_target *= np.sign(offset[1])
1342: (8)                 shift_vector = 0.01 * camera_to_target
1343: (12)                 self.camera.model_matrix =
1344: (8)                     opengl.translation_matrix(*shift_vector) @
self.camera.model_matrix
1345: (4)             def mouse_drag_orbit_controls(self, point, d_point, buttons, modifiers):
1346: (8)                 if buttons == 1:
1347: (12)                     self.camera.model_matrix =
1348: (16)                     opengl.rotation_matrix(z=-d_point[0])
1349: (16)                     @ opengl.translation_matrix(*-self.camera_target)
1350: (16)                     @ self.camera.model_matrix
1351: (12)
1352: (12)                     )
1353: (12)                     camera_position = self.camera.get_position()
1354: (12)                     camera_y_axis = self.camera.model_matrix[:3, 1]
axis_of_rotation = space_ops.normalize(

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1355: (16) np.cross(camera_y_axis, camera_position),
1356: (12) )
1357: (12) rotation_matrix = space_ops.rotation_matrix(
1358: (16)     d_point[1],
1359: (16)     axis_of_rotation,
1360: (16)     homogeneous=True,
1361: (12) )
1362: (12) maximum_polar_angle = self.camera.maximum_polar_angle
1363: (12) minimum_polar_angle = self.camera.minimum_polar_angle
1364: (12) potential_camera_model_matrix = rotation_matrix @
self.camera.model_matrix
1365: (12) potential_camera_location = potential_camera_model_matrix[:3, 3]
1366: (12) potential_camera_y_axis = potential_camera_model_matrix[:3, 1]
1367: (12) sign = (
1368: (16)     np.sign(potential_camera_y_axis[2])
1369: (16)     if potential_camera_y_axis[2] != 0
1370: (16)     else 1
1371: (12) )
1372: (12) potential_polar_angle = sign * np.arccos(
1373: (16)     potential_camera_location[2]
1374: (16)     / np.linalg.norm(potential_camera_location),
1375: (12) )
1376: (12) if minimum_polar_angle <= potential_polar_angle <=
maximum_polar_angle:
1377: (16)     self.camera.model_matrix = potential_camera_model_matrix
1378: (12) else:
1379: (16)     sign = np.sign(camera_y_axis[2]) if camera_y_axis[2] != 0 else
1
1380: (16)     current_polar_angle = sign * np.arccos(
1381: (20)         camera_position[2] / np.linalg.norm(camera_position),
1382: (16)     )
1383: (16)     if potential_polar_angle > maximum_polar_angle:
1384: (20)         polar_angle_delta = maximum_polar_angle -
current_polar_angle
1385: (16)
1386: (20)     else:
1387: (16)         polar_angle_delta = minimum_polar_angle -
current_polar_angle
1388: (20)
1389: (20)
1390: (20)
1391: (16)
1392: (16)     self.camera.model_matrix = rotation_matrix @
self.camera.model_matrix
1393: (12)     self.camera.model_matrix = (
1394: (16)         opengl.translation_matrix(*self.camera_target)
1395: (16)         @ self.camera.model_matrix
1396: (12)     )
1397: (8) elif buttons == 4:
1398: (12)     camera_x_axis = self.camera.model_matrix[:3, 0]
1399: (12)     horizontal_shift_vector = -d_point[0] * camera_x_axis
1400: (12)     vertical_shift_vector = -d_point[1] * np.cross(OUT, camera_x_axis)
1401: (12)     total_shift_vector = horizontal_shift_vector +
vertical_shift_vector
1402: (12)     self.camera.model_matrix = (
1403: (16)         opengl.translation_matrix(*total_shift_vector)
1404: (16)         @ self.camera.model_matrix
1405: (12)     )
1406: (12)     self.camera_target += total_shift_vector
1407: (4) def set_key_function(self, char, func):
1408: (8)     self.key_to_function_map[char] = func
1409: (4) def on_mouse_press(self, point, button, modifiers):
1410: (8)     for func in self.mouse_press_callbacks:
1411: (12)         func()

```

-----  
File 111 - \_\_init\_\_.py:

1: (0)

File 112 - moving\_camera\_scene.py:

```

1: (0)          """A scene whose camera can be moved around.
2: (0)          .. SEEALSO::
3: (4)            :mod:`.moving_camera`
4: (0)          Examples
5: (0)
6: (0)          .. manim:: ChangingCameraWidthAndRestore
7: (4)            class ChangingCameraWidthAndRestore(MovingCameraScene):
8: (8)              def construct(self):
9: (12)                text = Text("Hello World").set_color(BLUE)
10: (12)               self.add(text)
11: (12)               self.camera.frame.save_state()
12: (12)               self.play(self.camera.frame.animate.set(width=text.width * 1.2))
13: (12)               self.wait(0.3)
14: (12)               self.play(Restore(self.camera.frame))
15: (0)          .. manim:: MovingCameraCenter
16: (4)            class MovingCameraCenter(MovingCameraScene):
17: (8)              def construct(self):
18: (12)                s = Square(color=RED, fill_opacity=0.5).move_to(2 * LEFT)
19: (12)                t = Triangle(color=GREEN, fill_opacity=0.5).move_to(2 * RIGHT)
20: (12)                self.wait(0.3)
21: (12)                self.add(s, t)
22: (12)                self.play(self.camera.frame.animate.move_to(s))
23: (12)                self.wait(0.3)
24: (12)                self.play(self.camera.frame.animate.move_to(t))
25: (0)          .. manim:: MovingAndZoomingCamera
26: (4)            class MovingAndZoomingCamera(MovingCameraScene):
27: (8)              def construct(self):
28: (12)                s = Square(color=BLUE, fill_opacity=0.5).move_to(2 * LEFT)
29: (12)                t = Triangle(color=YELLOW, fill_opacity=0.5).move_to(2 * RIGHT)
30: (12)                self.add(s, t)
31: (12)
32: (12)                self.play(self.camera.frame.animate.move_to(s).set(width=s.width*2))
33: (12)                self.wait(0.3)
34: (12)                self.play(self.camera.frame.animate.move_to(t).set(width=t.width*2))
35: (12)                  self.play(self.camera.frame.animate.move_to(ORIGIN).set(width=14))
36: (0)          .. manim:: MovingCameraOnGraph
37: (4)            class MovingCameraOnGraph(MovingCameraScene):
38: (8)              def construct(self):
39: (12)                self.camera.frame.save_state()
40: (12)                ax = Axes(x_range=[-1, 10], y_range=[-1, 10])
41: (12)                graph = ax.plot(lambda x: np.sin(x), color=WHITE, x_range=[0, 3 *
PI])
42: (12)                dot_1 = Dot(ax.i2gp(graph.t_min, graph))
43: (12)                dot_2 = Dot(ax.i2gp(graph.t_max, graph))
44: (12)                self.add(ax, graph, dot_1, dot_2)
45: (12)                self.play(self.camera.frame.animate.scale(0.5).move_to(dot_1))
46: (12)                self.play(self.camera.frame.animate.move_to(dot_2))
47: (12)                self.play(Restore(self.camera.frame))
48: (12)                self.wait()
49: (0)          """
50: (0)            from __future__ import annotations
51: (0)            __all__ = ["MovingCameraScene"]
52: (0)            from manim.animation.animation import Animation
53: (0)            from ..camera.moving_camera import MovingCamera
54: (0)            from ..scene.scene import Scene
55: (0)            from ..utils.family import extract_mobject_family_members
56: (0)            from ..utils.iterables import list_update
57: (4)              class MovingCameraScene(Scene):
58: (4)                """
59: (4)                  This is a Scene, with special configurations and properties that
60: (4)                  make it suitable for cases where the camera must be moved around.
61: (4)                  .. SEEALSO::

```

```

61: (8)             :class:`.MovingCamera`  

62: (4)  

63: (4)         """  

64: (8)         def __init__(self, camera_class=MovingCamera, **kwargs):  

65: (4)             super().__init__(camera_class=camera_class, **kwargs)  

66: (4)         def get_moving_mobjects(self, *animations):  

67: (8)             """  

68: (8)                 This method returns a list of all of the Mobjects in the Scene that  

69: (8)                 are moving, that are also in the animations passed.  

70: (8)             Parameters  

71: (8)             -----  

72: (12)             *animations  

73: (8)                 The Animations whose mobjects will be checked.  

74: (8)             """  

75: (8)             moving_mobjects = super().get_moving_mobjects(*animations)  

76: (8)             all_moving_mobjects = extract_mobject_family_members(moving_mobjects)  

77: (8)             movement_indicators =  

self.renderer.camera.get_mobjects_indicating_movement()  

78: (12)             for movement_indicator in movement_indicators:  

79: (16)                 if movement_indicator in all_moving_mobjects:  

80: (16)                     return list_update(self.mobjects, moving_mobjects)  

80: (8)             return moving_mobjects
-----
```

## File 113 - vectorized\_mobject\_rendering.py:

```

1: (0)             from __future__ import annotations  

2: (0)             import collections  

3: (0)             import numpy as np  

4: (0)             from ..utils import opengl  

5: (0)             from ..utils.space_ops import cross2d, earclip_triangulation  

6: (0)             from .shader import Shader  

7: (0)             __all__ = [  

8: (4)                 "render_opengl_vectorized_mobject_fill",  

9: (4)                 "render_opengl_vectorized_mobject_stroke",  

10: (0)             ]  

11: (0)             def build_matrix_lists(mob):  

12: (4)                 root_hierarchical_matrix = mob.hierarchical_model_matrix()  

13: (4)                 matrix_to_mobject_list = collections.defaultdict(list)  

14: (4)                 if mob.has_points():  

15: (8)  

matrix_to_mobject_list[tuple(root_hierarchical_matrix.ravel())].append(mob)  

16: (4)                 mob_to_hierarchical_matrix = {mob: root_hierarchical_matrix}  

17: (4)                 dfs = [mob]  

18: (4)                 while dfs:  

19: (8)                     parent = dfs.pop()  

20: (8)                     for child in parent.subobjects:  

21: (12)                         child_hierarchical_matrix = (  

22: (16)                             mob_to_hierarchical_matrix[parent] @ child.model_matrix  

23: (12)                         )  

24: (12)                         mob_to_hierarchical_matrix[child] = child_hierarchical_matrix  

25: (12)                         if child.has_points():  

26: (16)  

matrix_to_mobject_list[tuple(child_hierarchical_matrix.ravel())].append(  

27: (20)   child,  

28: (16)   )  

29: (12)   dfs.append(child)  

30: (4)   return matrix_to_mobject_list  

31: (0)             def render_opengl_vectorized_mobject_fill(renderer, mobject):  

32: (4)                 matrix_to_mobject_list = build_matrix_lists(mobject)  

33: (4)                 for matrix_tuple, mobject_list in matrix_to_mobject_list.items():  

34: (8)                     model_matrix = np.array(matrix_tuple).reshape((4, 4))  

35: (8)                     render_mobject_fills_with_matrix(renderer, model_matrix, mobject_list)  

36: (0)             def render_mobject_fills_with_matrix(renderer, model_matrix, mobjects):  

37: (4)                 total_size = 0  

38: (4)                 for submob in mobjects:  

39: (8)                     total_size += triangulate_mobject(submob).shape[0]  

40: (4)                     attributes = np.empty(  

41: (8)                                     total_size,
```

```

42: (8)           dtype=[
43: (12)             ("in_vert", np.float32, (3,)),
44: (12)             ("in_color", np.float32, (4,)),
45: (12)             ("texture_coords", np.float32, (2,)),
46: (12)             ("texture_mode", np.int32),
47: (8)         ],
48: (4)     )
49: (4)     write_offset = 0
50: (4)     for submob in mobjects:
51: (8)         if not submob.has_points():
52: (12)             continue
53: (8)         mobject_triangulation = triangulate_mobject(submob)
54: (8)         end_offset = write_offset + mobject_triangulation.shape[0]
55: (8)         attributes[write_offset:end_offset] = mobject_triangulation
56: (8)         attributes["in_color"][write_offset:end_offset] = np.repeat(
57: (12)             submob.fill_rgba,
58: (12)             mobject_triangulation.shape[0],
59: (12)             axis=0,
60: (8)         )
61: (8)         write_offset = end_offset
62: (4)     fill_shader = Shader(renderer.context, name="vectorized_mobject_fill")
63: (4)     fill_shader.set_uniform(
64: (8)         "u_model_view_matrix",
65: (8)         opengl.matrix_to_shader_input(
66: (12)             renderer.camera.unformatted_view_matrix @ model_matrix,
67: (8)         ),
68: (4)     )
69: (4)     fill_shader.set_uniform(
70: (8)         "u_projection_matrix",
71: (8)         renderer.scene.camera.projection_matrix,
72: (4)     )
73: (4)     vbo = renderer.context.buffer(attributes.tobytes())
74: (4)     vao = renderer.context.simple_vertex_array(
75: (8)         fill_shader.shader_program,
76: (8)         vbo,
77: (8)         *attributes.dtype.names,
78: (4)     )
79: (4)     vao.render()
80: (4)     vao.release()
81: (4)     vbo.release()
82: (0)     def triangulate_mobject(mob):
83: (4)         if not mob.needs_new_triangulation:
84: (8)             return mob.triangulation
85: (4)         points = mob.points
86: (4)         b0s = points[0::3]
87: (4)         b1s = points[1::3]
88: (4)         b2s = points[2::3]
89: (4)         v01s = b1s - b0s
90: (4)         v12s = b2s - b1s
91: (4)         crosses = cross2d(v01s, v12s)
92: (4)         convexities = np.sign(crosses)
93: (4)         if mob.orientation == 1:
94: (8)             concave_parts = convexities > 0
95: (8)             convex_parts = convexities <= 0
96: (4)         else:
97: (8)             concave_parts = convexities < 0
98: (8)             convex_parts = convexities >= 0
99: (4)         atol = mob.tolerance_for_point_equality
100: (4)        end_of_loop = np.zeros(len(b0s), dtype=bool)
101: (4)        end_of_loop[:-1] = (np.abs(b2s[:-1] - b0s[1:]) > atol).any(1)
102: (4)        end_of_loop[-1] = True
103: (4)        indices = np.arange(len(points), dtype=int)
104: (4)        inner_vert_indices = np.hstack(
105: (8)            [
106: (12)                indices[0::3],
107: (12)                indices[1::3][concave_parts],
108: (12)                indices[2::3][end_of_loop],
109: (8)            ],
110: (4)        )

```

```

111: (4)             inner_vert_indices.sort()
112: (4)             rings = np.arange(1, len(inner_vert_indices) + 1)[inner_vert_indices % 3
== 2]
113: (4)             inner_verts = points[inner_vert_indices]
114: (4)             inner_tri_indices = inner_vert_indices[earclip_triangulation(inner_verts,
rings)]
115: (4)             bezier_triangle_indices = np.reshape(indices, (-1, 3))
116: (4)             concave_triangle_indices =
np.reshape(bezier_triangle_indices[concave_parts], (-1))
117: (4)             convex_triangle_indices =
np.reshape(bezier_triangle_indices[convex_parts], (-1))
118: (4)             points = points[
119: (8)                 np.hstack(
120: (12)                   [
121: (16)                     concave_triangle_indices,
122: (16)                     convex_triangle_indices,
123: (16)                     inner_tri_indices,
124: (12)                   ],
125: (8)                 )
126: (4)             ]
127: (4)             texture_coords = np.tile(
128: (8)               [
129: (12)                 [0.0, 0.0],
130: (12)                 [0.5, 0.0],
131: (12)                 [1.0, 1.0],
132: (8)               ],
133: (8)               (points.shape[0] // 3, 1),
134: (4)             )
135: (4)             texture_mode = np.hstack(
136: (8)               (
137: (12)                 np.ones(concave_triangle_indices.shape[0]),
138: (12)                 -1 * np.ones(convex_triangle_indices.shape[0]),
139: (12)                 np.zeros(inner_tri_indices.shape[0]),
140: (8)               ),
141: (4)             )
142: (4)             attributes = np.zeros(
143: (8)               points.shape[0],
144: (8)               dtype=[
145: (12)                 ("in_vert", np.float32, (3,)),
146: (12)                 ("in_color", np.float32, (4,)),
147: (12)                 ("texture_coords", np.float32, (2,)),
148: (12)                 ("texture_mode", np.int32),
149: (8)               ],
150: (4)             )
151: (4)             attributes["in_vert"] = points
152: (4)             attributes["texture_coords"] = texture_coords
153: (4)             attributes["texture_mode"] = texture_mode
154: (4)             mob.triangulation = attributes
155: (4)             mob.needs_new_triangulation = False
156: (4)             return attributes
157: (0)             def render_opengl_vectorized_mobject_stroke(renderer, mobject):
158: (4)               matrix_to_mobject_list = build_matrix_lists(mobject)
159: (4)               for matrix_tuple, mobject_list in matrix_to_mobject_list.items():
160: (8)                 model_matrix = np.array(matrix_tuple).reshape((4, 4))
161: (8)                 render_mobject_strokes_with_matrix(renderer, model_matrix,
mobject_list)
162: (0)             def render_mobject_strokes_with_matrix(renderer, model_matrix, mobjects):
163: (4)               total_size = 0
164: (4)               for submob in mobjects:
165: (8)                 total_size += submob.points.shape[0]
166: (4)                 points = np.empty((total_size, 3))
167: (4)                 colors = np.empty((total_size, 4))
168: (4)                 widths = np.empty(total_size)
169: (4)                 write_offset = 0
170: (4)                 for submob in mobjects:
171: (8)                   if not submob.has_points():
172: (12)                     continue
173: (8)                   end_offset = write_offset + submob.points.shape[0]
174: (8)                   points[write_offset:end_offset] = submob.points

```

```

175: (8)             if submob.stroke_rgba.shape[0] ==
points[write_offset:end_offset].shape[0]:
176: (12)                 colors[write_offset:end_offset] = submob.stroke_rgba
177: (8)             else:
178: (12)                 colors[write_offset:end_offset] = np.repeat(
179: (16)                     submob.stroke_rgba,
180: (16)                     submob.points.shape[0],
181: (16)                     axis=0,
182: (12)                 )
183: (8)                 widths[write_offset:end_offset] = np.repeat(
184: (12)                     submob.stroke_width,
185: (12)                     submob.points.shape[0],
186: (8)                 )
187: (8)             write_offset = end_offset
188: (4)             stroke_data = np.zeros(
189: (8)                 len(points),
190: (8)                 dtype=[
191: (12)                     ("current_curve", np.float32, (3, 3)),
192: (12)                     ("tile_coordinate", np.float32, (2,)),
193: (12)                     ("in_color", np.float32, (4,)),
194: (12)                     ("in_width", np.float32),
195: (8)                 ],
196: (4)             )
197: (4)             stroke_data["in_color"] = colors
198: (4)             stroke_data["in_width"] = widths
199: (4)             curves = np.reshape(points, (-1, 3, 3))
200: (4)             stroke_data["current_curve"] = np.repeat(curves, 3, axis=0)
201: (4)             stroke_data = np.tile(stroke_data, 2)
202: (4)             stroke_data["tile_coordinate"] = np.vstack(
203: (8)                 (
204: (12)                     np.tile(
205: (16)                         [
206: (20)                             [0.0, 0.0],
207: (20)                             [0.0, 1.0],
208: (20)                             [1.0, 1.0],
209: (16)                         ],
210: (16)                         (len(points) // 3, 1),
211: (12)                     ),
212: (12)                     np.tile(
213: (16)                         [
214: (20)                             [0.0, 0.0],
215: (20)                             [1.0, 0.0],
216: (20)                             [1.0, 1.0],
217: (16)                         ],
218: (16)                         (len(points) // 3, 1),
219: (12)                     ),
220: (8)                 ),
221: (4)             )
222: (4)             shader = Shader(renderer.context, "vectorized_mobject_stroke")
223: (4)             shader.set_uniform(
224: (8)                 "u_model_view_matrix",
225: (8)                 opengl.matrix_to_shader_input(
226: (12)                     renderer.camera.unformatted_view_matrix @ model_matrix,
227: (8)                 ),
228: (4)             )
229: (4)             shader.set_uniform("u_projection_matrix",
renderer.scene.camera.projection_matrix)
230: (4)             shader.set_uniform("manim_unit_normal", tuple(-
mobjects[0].unit_normal[0]))
231: (4)             vbo = renderer.context.buffer(stroke_data.tobytes())
232: (4)             vao = renderer.context.simple_vertex_array(
233: (8)                 shader.shader_program, vbo, *stroke_data.dtype.names
234: (4)             )
235: (4)             renderer.frame_buffer_object.use()
236: (4)             vao.render()
237: (4)             vao.release()
238: (4)             vbo.release()

```

-----

## File 114 - X11.py:

```

1: (0)          """X11 Colors
2: (0)          These color and their names (taken from
3: (0)          https://www.w3schools.com/colors/colors\_x11.asp) were developed at the
4: (0)          Massachusetts Intitute of Technology (MIT) during
5: (0)          the development of color based computer display system.
6: (0)          To use the colors from this list, access them directly from the module (which
7: (0)          is exposed to Manim's global name space):
8: (0)          .. code:: pycon
9: (4)              >>> from manim import X11
10: (4)             >>> X11.BEIGE
11: (4)             ManimColor('#F5F5DC')
12: (0)             List of Color Constants
13: (0)             -----
14: (0)             .. automanimcolormodule:: manim.utils.color.X11
15: (0)
16: (0)             from .core import ManimColor
17: (0)             ALICEBLUE = ManimColor("#F0F8FF")
18: (0)             ANTIQUEWHITE = ManimColor("#FAEBD7")
19: (0)             ANTIQUEWHITE1 = ManimColor("#FFEFD8")
20: (0)             ANTIQUEWHITE2 = ManimColor("#EEDFCC")
21: (0)             ANTIQUEWHITE3 = ManimColor("#CDC0B0")
22: (0)             ANTIQUEWHITE4 = ManimColor("#8B8378")
23: (0)             AQUAMARINE1 = ManimColor("#7FFFAD")
24: (0)             AQUAMARINE2 = ManimColor("#76EEC6")
25: (0)             AQUAMARINE4 = ManimColor("#458B74")
26: (0)             AZURE1 = ManimColor("#F0FFFF")
27: (0)             AZURE2 = ManimColor("#E0EEEE")
28: (0)             AZURE3 = ManimColor("#C1CDCE")
29: (0)             AZURE4 = ManimColor("#838B8B")
30: (0)             BEIGE = ManimColor("#F5F5DC")
31: (0)             BISQUE1 = ManimColor("#FFE4C4")
32: (0)             BISQUE2 = ManimColor("#EED5B7")
33: (0)             BISQUE3 = ManimColor("#CDB79E")
34: (0)             BISQUE4 = ManimColor("#8B7D6B")
35: (0)             BLACK = ManimColor("#000000")
36: (0)             BLANCHEDALMOND = ManimColor("#FFEBCD")
37: (0)             BLUE1 = ManimColor("#0000FF")
38: (0)             BLUE2 = ManimColor("#0000EE")
39: (0)             BLUE4 = ManimColor("#00008B")
40: (0)             BLUEVIOLET = ManimColor("#8A2BE2")
41: (0)             BROWN = ManimColor("#A52A2A")
42: (0)             BROWN1 = ManimColor("#FF4040")
43: (0)             BROWN2 = ManimColor("#EE3B3B")
44: (0)             BROWN3 = ManimColor("#CD3333")
45: (0)             BROWN4 = ManimColor("#8B2323")
46: (0)             BURLYWOOD = ManimColor("#DEB887")
47: (0)             BURLYWOOD1 = ManimColor("#FFD39B")
48: (0)             BURLYWOOD2 = ManimColor("#EBC591")
49: (0)             BURLYWOOD3 = ManimColor("#CDAA7D")
50: (0)             BURLYWOOD4 = ManimColor("#8B7355")
51: (0)             CADETBLUE = ManimColor("#5F9EA0")
52: (0)             CADETBLUE1 = ManimColor("#98F5FF")
53: (0)             CADETBLUE2 = ManimColor("#8EE5EE")
54: (0)             CADETBLUE3 = ManimColor("#7AC5CD")
55: (0)             CADETBLUE4 = ManimColor("#53868B")
56: (0)             CHARTREUSE1 = ManimColor("#7FFF00")
57: (0)             CHARTREUSE2 = ManimColor("#76EE00")
58: (0)             CHARTREUSE3 = ManimColor("#66CD00")
59: (0)             CHARTREUSE4 = ManimColor("#458B00")
60: (0)             CHOCOLATE = ManimColor("#D2691E")
61: (0)             CHOCOLATE1 = ManimColor("#FF7F24")
62: (0)             CHOCOLATE2 = ManimColor("#EE7621")
63: (0)             CHOCOLATE3 = ManimColor("#CD661D")
64: (0)             CORAL = ManimColor("#FF7F50")
65: (0)             CORAL1 = ManimColor("#FF7256")
66: (0)             CORAL2 = ManimColor("#EE6A50")

```

```

67: (0) CORAL3 = ManimColor("#CD5B45")
68: (0) CORAL4 = ManimColor("#8B3E2F")
69: (0) CORNFLOWERBLUE = ManimColor("#6495ED")
70: (0) CORNSILK1 = ManimColor("#FFF8DC")
71: (0) CORNSILK2 = ManimColor("#EEE8CD")
72: (0) CORNSILK3 = ManimColor("#CDC8B1")
73: (0) CORNSILK4 = ManimColor("#8B8878")
74: (0) CYAN1 = ManimColor("#00FFFF")
75: (0) CYAN2 = ManimColor("#00EEEE")
76: (0) CYAN3 = ManimColor("#00CDCD")
77: (0) CYAN4 = ManimColor("#008B8B")
78: (0) DARKGOLDENROD = ManimColor("#B8860B")
79: (0) DARKGOLDENROD1 = ManimColor("#FFB90F")
80: (0) DARKGOLDENROD2 = ManimColor("#EEAD0E")
81: (0) DARKGOLDENROD3 = ManimColor("#CD950C")
82: (0) DARKGOLDENROD4 = ManimColor("#8B6508")
83: (0) DARKGREEN = ManimColor("#006400")
84: (0) DARKKHAKI = ManimColor("#BDB76B")
85: (0) DARKOLIVEGREEN = ManimColor("#556B2F")
86: (0) DARKOLIVEGREEN1 = ManimColor("#CAFF70")
87: (0) DARKOLIVEGREEN2 = ManimColor("#BCEE68")
88: (0) DARKOLIVEGREEN3 = ManimColor("#A2CD5A")
89: (0) DARKOLIVEGREEN4 = ManimColor("#6E8B3D")
90: (0) DARKORANGE = ManimColor("#FF8C00")
91: (0) DARKORANGE1 = ManimColor("#FF7F00")
92: (0) DARKORANGE2 = ManimColor("#EE7600")
93: (0) DARKORANGE3 = ManimColor("#CD6600")
94: (0) DARKORANGE4 = ManimColor("#8B4500")
95: (0) DARKORCHID = ManimColor("#9932CC")
96: (0) DARKORCHID1 = ManimColor("#BF3EFF")
97: (0) DARKORCHID2 = ManimColor("#B23AEE")
98: (0) DARKORCHID3 = ManimColor("#9A32CD")
99: (0) DARKORCHID4 = ManimColor("#68228B")
100: (0) DARKSALMON = ManimColor("#E9967A")
101: (0) DARKSEAGREEN = ManimColor("#8FBC8F")
102: (0) DARKSEAGREEN1 = ManimColor("#C1FFC1")
103: (0) DARKSEAGREEN2 = ManimColor("#B4EEB4")
104: (0) DARKSEAGREEN3 = ManimColor("#9BCD9B")
105: (0) DARKSEAGREEN4 = ManimColor("#698B69")
106: (0) DARKSLATEBLUE = ManimColor("#483D8B")
107: (0) DARKSLATEGRAY = ManimColor("#2F4F4F")
108: (0) DARKSLATEGRAY1 = ManimColor("#97FFFF")
109: (0) DARKSLATEGRAY2 = ManimColor("#8DEEEE")
110: (0) DARKSLATEGRAY3 = ManimColor("#79CDCD")
111: (0) DARKSLATEGRAY4 = ManimColor("#528B8B")
112: (0) DARKTURQUOISE = ManimColor("#00CED1")
113: (0) DARKVIOLET = ManimColor("#9400D3")
114: (0) DEEPPINK1 = ManimColor("#FF1493")
115: (0) DEEPPINK2 = ManimColor("#EE1289")
116: (0) DEEPPINK3 = ManimColor("#CD1076")
117: (0) DEEPPINK4 = ManimColor("#8B0A50")
118: (0) DEEPSKYBLUE1 = ManimColor("#00BFFF")
119: (0) DEEPSKYBLUE2 = ManimColor("#00B2EE")
120: (0) DEEPSKYBLUE3 = ManimColor("#009ACD")
121: (0) DEEPSKYBLUE4 = ManimColor("#00688B")
122: (0) DIMGRAY = ManimColor("#696969")
123: (0) DODGERBLUE1 = ManimColor("#1E90FF")
124: (0) DODGERBLUE2 = ManimColor("#1C86EE")
125: (0) DODGERBLUE3 = ManimColor("#1874CD")
126: (0) DODGERBLUE4 = ManimColor("#104E8B")
127: (0) FIREBRICK = ManimColor("#B22222")
128: (0) FIREBRICK1 = ManimColor("#FF3030")
129: (0) FIREBRICK2 = ManimColor("#EE2C2C")
130: (0) FIREBRICK3 = ManimColor("#CD2626")
131: (0) FIREBRICK4 = ManimColor("#8B1A1A")
132: (0) FLORALWHITE = ManimColor("#FFF0F0")
133: (0) FORESTGREEN = ManimColor("#228B22")
134: (0) GAINSBORO = ManimColor("#DCDCDC")
135: (0) GHOSTWHITE = ManimColor("#F8F8FF")

```

```
136: (0) GOLD1 = ManimColor("#FFD700")
137: (0) GOLD2 = ManimColor("#EEC900")
138: (0) GOLD3 = ManimColor("#CDAD00")
139: (0) GOLD4 = ManimColor("#8B7500")
140: (0) GOLDENROD = ManimColor("#DAA520")
141: (0) GOLDENROD1 = ManimColor("#FFC125")
142: (0) GOLDENROD2 = ManimColor("#EEB422")
143: (0) GOLDENROD3 = ManimColor("#CD9B1D")
144: (0) GOLDENROD4 = ManimColor("#8B6914")
145: (0) GRAY = ManimColor("#BEBEBE")
146: (0) GRAY1 = ManimColor("#030303")
147: (0) GRAY2 = ManimColor("#050505")
148: (0) GRAY3 = ManimColor("#080808")
149: (0) GRAY4 = ManimColor("#0A0A0A")
150: (0) GRAY5 = ManimColor("#0D0D0D")
151: (0) GRAY6 = ManimColor("#0F0F0F")
152: (0) GRAY7 = ManimColor("#121212")
153: (0) GRAY8 = ManimColor("#141414")
154: (0) GRAY9 = ManimColor("#171717")
155: (0) GRAY10 = ManimColor("#1A1A1A")
156: (0) GRAY11 = ManimColor("#1C1C1C")
157: (0) GRAY12 = ManimColor("#1F1F1F")
158: (0) GRAY13 = ManimColor("#212121")
159: (0) GRAY14 = ManimColor("#242424")
160: (0) GRAY15 = ManimColor("#262626")
161: (0) GRAY16 = ManimColor("#292929")
162: (0) GRAY17 = ManimColor("#2B2B2B")
163: (0) GRAY18 = ManimColor("#2E2E2E")
164: (0) GRAY19 = ManimColor("#303030")
165: (0) GRAY20 = ManimColor("#333333")
166: (0) GRAY21 = ManimColor("#363636")
167: (0) GRAY22 = ManimColor("#383838")
168: (0) GRAY23 = ManimColor("#3B3B3B")
169: (0) GRAY24 = ManimColor("#3D3D3D")
170: (0) GRAY25 = ManimColor("#404040")
171: (0) GRAY26 = ManimColor("#424242")
172: (0) GRAY27 = ManimColor("#454545")
173: (0) GRAY28 = ManimColor("#474747")
174: (0) GRAY29 = ManimColor("#4A4A4A")
175: (0) GRAY30 = ManimColor("#4D4D4D")
176: (0) GRAY31 = ManimColor("#4F4F4F")
177: (0) GRAY32 = ManimColor("#525252")
178: (0) GRAY33 = ManimColor("#545454")
179: (0) GRAY34 = ManimColor("#575757")
180: (0) GRAY35 = ManimColor("#595959")
181: (0) GRAY36 = ManimColor("#5C5C5C")
182: (0) GRAY37 = ManimColor("#5E5E5E")
183: (0) GRAY38 = ManimColor("#616161")
184: (0) GRAY39 = ManimColor("#636363")
185: (0) GRAY40 = ManimColor("#666666")
186: (0) GRAY41 = ManimColor("#696969")
187: (0) GRAY42 = ManimColor("#6B6B6B")
188: (0) GRAY43 = ManimColor("#6E6E6E")
189: (0) GRAY44 = ManimColor("#707070")
190: (0) GRAY45 = ManimColor("#737373")
191: (0) GRAY46 = ManimColor("#757575")
192: (0) GRAY47 = ManimColor("#787878")
193: (0) GRAY48 = ManimColor("#7A7A7A")
194: (0) GRAY49 = ManimColor("#7D7D7D")
195: (0) GRAY50 = ManimColor("#7F7F7F")
196: (0) GRAY51 = ManimColor("#828282")
197: (0) GRAY52 = ManimColor("#858585")
198: (0) GRAY53 = ManimColor("#878787")
199: (0) GRAY54 = ManimColor("#8A8A8A")
200: (0) GRAY55 = ManimColor("#8C8C8C")
201: (0) GRAY56 = ManimColor("#8F8F8F")
202: (0) GRAY57 = ManimColor("#919191")
203: (0) GRAY58 = ManimColor("#949494")
204: (0) GRAY59 = ManimColor("#969696")
```

```
205: (0)           GRAY60 = ManimColor("#999999")
206: (0)           GRAY61 = ManimColor("#9C9C9C")
207: (0)           GRAY62 = ManimColor("#9E9E9E")
208: (0)           GRAY63 = ManimColor("#A1A1A1")
209: (0)           GRAY64 = ManimColor("#A3A3A3")
210: (0)           GRAY65 = ManimColor("#A6A6A6")
211: (0)           GRAY66 = ManimColor("#A8A8A8")
212: (0)           GRAY67 = ManimColor("#ABABAB")
213: (0)           GRAY68 = ManimColor("#ADADAD")
214: (0)           GRAY69 = ManimColor("#B0B0B0")
215: (0)           GRAY70 = ManimColor("#B3B3B3")
216: (0)           GRAY71 = ManimColor("#B5B5B5")
217: (0)           GRAY72 = ManimColor("#B8B8B8")
218: (0)           GRAY73 = ManimColor("#BABABA")
219: (0)           GRAY74 = ManimColor("#BDBDBD")
220: (0)           GRAY75 = ManimColor("#BFBFBF")
221: (0)           GRAY76 = ManimColor("#C2C2C2")
222: (0)           GRAY77 = ManimColor("#C4C4C4")
223: (0)           GRAY78 = ManimColor("#C7C7C7")
224: (0)           GRAY79 = ManimColor("#C9C9C9")
225: (0)           GRAY80 = ManimColor("#CCCCCC")
226: (0)           GRAY81 = ManimColor("#CFCFCF")
227: (0)           GRAY82 = ManimColor("#D1D1D1")
228: (0)           GRAY83 = ManimColor("#D4D4D4")
229: (0)           GRAY84 = ManimColor("#D6D6D6")
230: (0)           GRAY85 = ManimColor("#D9D9D9")
231: (0)           GRAY86 = ManimColor("#DBDBDB")
232: (0)           GRAY87 = ManimColor("#DEDEDE")
233: (0)           GRAY88 = ManimColor("#E0E0E0")
234: (0)           GRAY89 = ManimColor("#E3E3E3")
235: (0)           GRAY90 = ManimColor("#E5E5E5")
236: (0)           GRAY91 = ManimColor("#E8E8E8")
237: (0)           GRAY92 = ManimColor("#EBEBEB")
238: (0)           GRAY93 = ManimColor("#EDEDED")
239: (0)           GRAY94 = ManimColor("#F0F0F0")
240: (0)           GRAY95 = ManimColor("#F2F2F2")
241: (0)           GRAY97 = ManimColor("#F7F7F7")
242: (0)           GRAY98 = ManimColor("#FAFAFA")
243: (0)           GRAY99 = ManimColor("#FCFCFC")
244: (0)           GREEN1 = ManimColor("#00FF00")
245: (0)           GREEN2 = ManimColor("#00EE00")
246: (0)           GREEN3 = ManimColor("#00CD00")
247: (0)           GREEN4 = ManimColor("#008B00")
248: (0)           GREENYELLOW = ManimColor("#ADFF2F")
249: (0)           HONEYDEW1 = ManimColor("#F0FFF0")
250: (0)           HONEYDEW2 = ManimColor("#E0EEE0")
251: (0)           HONEYDEW3 = ManimColor("#C1CDC1")
252: (0)           HONEYDEW4 = ManimColor("#838B83")
253: (0)           HOTPINK = ManimColor("#FF69B4")
254: (0)           HOTPINK1 = ManimColor("#FF6EB4")
255: (0)           HOTPINK2 = ManimColor("#EE6AA7")
256: (0)           HOTPINK3 = ManimColor("#CD6090")
257: (0)           HOTPINK4 = ManimColor("#8B3A62")
258: (0)           INDIANRED = ManimColor("#CD5C5C")
259: (0)           INDIANRED1 = ManimColor("#FF6A6A")
260: (0)           INDIANRED2 = ManimColor("#EE6363")
261: (0)           INDIANRED3 = ManimColor("#CD5555")
262: (0)           INDIANRED4 = ManimColor("#8B3A3A")
263: (0)           IVORY1 = ManimColor("#FFFFF0")
264: (0)           IVORY2 = ManimColor("#EEEEEE")
265: (0)           IVORY3 = ManimColor("#CDCDC1")
266: (0)           IVORY4 = ManimColor("#8B8B83")
267: (0)           KHAKI = ManimColor("#F0E68C")
268: (0)           KHAKI1 = ManimColor("#FFF68F")
269: (0)           KHAKI2 = ManimColor("#EEE685")
270: (0)           KHAKI3 = ManimColor("#CDC673")
271: (0)           KHAKI4 = ManimColor("#8B864E")
272: (0)           LAVENDER = ManimColor("#E6E6FA")
273: (0)           LAVENDERBLUSH1 = ManimColor("#FFF0F5")
```

```

274: (0)          LAVENDERBLUSH2 = ManimColor("#EEE0E5")
275: (0)          LAVENDERBLUSH3 = ManimColor("#CDC1C5")
276: (0)          LAVENDERBLUSH4 = ManimColor("#8B8386")
277: (0)          LAWNGREEN = ManimColor("#7CFC00")
278: (0)          LEMONCHIFFON1 = ManimColor("#FFFACD")
279: (0)          LEMONCHIFFON2 = ManimColor("#EEE9BF")
280: (0)          LEMONCHIFFON3 = ManimColor("#CDC9A5")
281: (0)          LEMONCHIFFON4 = ManimColor("#8B8970")
282: (0)          LIGHT = ManimColor("#EEDD82")
283: (0)          LIGHTBLUE = ManimColor("#ADD8E6")
284: (0)          LIGHTBLUE1 = ManimColor("#BFEFFF")
285: (0)          LIGHTBLUE2 = ManimColor("#B2DFEE")
286: (0)          LIGHTBLUE3 = ManimColor("#9AC0CD")
287: (0)          LIGHTBLUE4 = ManimColor("#68838B")
288: (0)          LIGHTCORAL = ManimColor("#F08080")
289: (0)          LIGHTCYAN1 = ManimColor("#E0FFFF")
290: (0)          LIGHTCYAN2 = ManimColor("#D1EEEE")
291: (0)          LIGHTCYAN3 = ManimColor("#B4CDCE")
292: (0)          LIGHTCYAN4 = ManimColor("#7A8B8B")
293: (0)          LIGHTGOLDENROD1 = ManimColor("#FFEC8B")
294: (0)          LIGHTGOLDENROD2 = ManimColor("#EEDC82")
295: (0)          LIGHTGOLDENROD3 = ManimColor("#CDBE70")
296: (0)          LIGHTGOLDENROD4 = ManimColor("#8B814C")
297: (0)          LIGHTGOLDENRODYELLOW = ManimColor("#FAFAD2")
298: (0)          LIGHTGRAY = ManimColor("#D3D3D3")
299: (0)          LIGHTPINK = ManimColor("#FFB6C1")
300: (0)          LIGHTPINK1 = ManimColor("#FFAEB9")
301: (0)          LIGHTPINK2 = ManimColor("#EEA2AD")
302: (0)          LIGHTPINK3 = ManimColor("#CD8C95")
303: (0)          LIGHTPINK4 = ManimColor("#8B5F65")
304: (0)          LIGHTSALMON1 = ManimColor("#FFA07A")
305: (0)          LIGHTSALMON2 = ManimColor("#EE9572")
306: (0)          LIGHTSALMON3 = ManimColor("#CD8162")
307: (0)          LIGHTSALMON4 = ManimColor("#8B5742")
308: (0)          LIGHTSEAGREEN = ManimColor("#20B2AA")
309: (0)          LIGHTSKYBLUE = ManimColor("#87CEFA")
310: (0)          LIGHTSKYBLUE1 = ManimColor("#B0E2FF")
311: (0)          LIGHTSKYBLUE2 = ManimColor("#A4D3EE")
312: (0)          LIGHTSKYBLUE3 = ManimColor("#8DB6CD")
313: (0)          LIGHTSKYBLUE4 = ManimColor("#607B8B")
314: (0)          LIGHTSLATEBLUE = ManimColor("#8470FF")
315: (0)          LIGHTSLATEGRAY = ManimColor("#778899")
316: (0)          LIGHTSTEELBLUE = ManimColor("#B0C4DE")
317: (0)          LIGHTSTEELBLUE1 = ManimColor("#CAE1FF")
318: (0)          LIGHTSTEELBLUE2 = ManimColor("#BCD2EE")
319: (0)          LIGHTSTEELBLUE3 = ManimColor("#A2B5CD")
320: (0)          LIGHTSTEELBLUE4 = ManimColor("#6E7B8B")
321: (0)          LIGHTYELLOW1 = ManimColor("#FFFFE0")
322: (0)          LIGHTYELLOW2 = ManimColor("#EEEED1")
323: (0)          LIGHTYELLOW3 = ManimColor("#CDCDB4")
324: (0)          LIGHTYELLOW4 = ManimColor("#8B8B7A")
325: (0)          LIMEGREEN = ManimColor("#32CD32")
326: (0)          LINEN = ManimColor("#FAF0E6")
327: (0)          MAGENTA = ManimColor("#FF00FF")
328: (0)          MAGENTA2 = ManimColor("#EE00EE")
329: (0)          MAGENTA3 = ManimColor("#CD00CD")
330: (0)          MAGENTA4 = ManimColor("#8B008B")
331: (0)          MAROON = ManimColor("#B03060")
332: (0)          MAROON1 = ManimColor("#FF34B3")
333: (0)          MAROON2 = ManimColor("#EE30A7")
334: (0)          MAROON3 = ManimColor("#CD2990")
335: (0)          MAROON4 = ManimColor("#8B1C62")
336: (0)          MEDIUM = ManimColor("#66CDAA")
337: (0)          MEDIUMAQUAMARINE = ManimColor("#66CDAA")
338: (0)          MEDIUMBLUE = ManimColor("#0000CD")
339: (0)          MEDIUMMORCHID = ManimColor("#BA55D3")
340: (0)          MEDIUMMORCHID1 = ManimColor("#E066FF")
341: (0)          MEDIUMMORCHID2 = ManimColor("#D15FEE")
342: (0)          MEDIUMMORCHID3 = ManimColor("#B452CD")

```

```

343: (0) MEDIUMORCHID4 = ManimColor("#7A378B")
344: (0) MEDIUMPURPLE = ManimColor("#9370DB")
345: (0) MEDIUMPURPLE1 = ManimColor("#AB82FF")
346: (0) MEDIUMPURPLE2 = ManimColor("#9F79EE")
347: (0) MEDIUMPURPLE3 = ManimColor("#8968CD")
348: (0) MEDIUMPURPLE4 = ManimColor("#5D478B")
349: (0) MEDIUMSEAGREEN = ManimColor("#3CB371")
350: (0) MEDIUMSLATEBLUE = ManimColor("#7B68EE")
351: (0) MEDIUMSPRINGGREEN = ManimColor("#00FA9A")
352: (0) MEDIUMTURQUOISE = ManimColor("#48D1CC")
353: (0) MEDIUMVIOLETRED = ManimColor("#C71585")
354: (0) MIDNIGHTBLUE = ManimColor("#191970")
355: (0) MINTCREAM = ManimColor("#F5FFFA")
356: (0) MISTYROSE1 = ManimColor("#FFE4E1")
357: (0) MISTYROSE2 = ManimColor("#EED5D2")
358: (0) MISTYROSE3 = ManimColor("#CDB7B5")
359: (0) MISTYROSE4 = ManimColor("#8B7D7B")
360: (0) MOCCASIN = ManimColor("#FFE4B5")
361: (0) NAVAJOWHITE1 = ManimColor("#FFDEAD")
362: (0) NAVAJOWHITE2 = ManimColor("#EECFA1")
363: (0) NAVAJOWHITE3 = ManimColor("#CDB38B")
364: (0) NAVAJOWHITE4 = ManimColor("#8B795E")
365: (0) NAVYBLUE = ManimColor("#000080")
366: (0) OLDLACE = ManimColor("#FDF5E6")
367: (0) OLIVEDRAB = ManimColor("#6B8E23")
368: (0) OLIVEDRAB1 = ManimColor("#C0FF3E")
369: (0) OLIVEDRAB2 = ManimColor("#B3EE3A")
370: (0) OLIVEDRAB4 = ManimColor("#698B22")
371: (0) ORANGE1 = ManimColor("#FFA500")
372: (0) ORANGE2 = ManimColor("#EE9A00")
373: (0) ORANGE3 = ManimColor("#CD8500")
374: (0) ORANGE4 = ManimColor("#8B5A00")
375: (0) ORANGERED1 = ManimColor("#FF4500")
376: (0) ORANGERED2 = ManimColor("#EE4000")
377: (0) ORANGERED3 = ManimColor("#CD3700")
378: (0) ORANGERED4 = ManimColor("#8B2500")
379: (0) ORCHID = ManimColor("#DA70D6")
380: (0) ORCHID1 = ManimColor("#FF83FA")
381: (0) ORCHID2 = ManimColor("#EE7AE9")
382: (0) ORCHID3 = ManimColor("#CD69C9")
383: (0) ORCHID4 = ManimColor("#8B4789")
384: (0) PALE = ManimColor("#DB7093")
385: (0) PALEGOLDENROD = ManimColor("#EEE8AA")
386: (0) PALEGREEN = ManimColor("#98FB98")
387: (0) PALEGREEN1 = ManimColor("#9AFF9A")
388: (0) PALEGREEN2 = ManimColor("#90EE90")
389: (0) PALEGREEN3 = ManimColor("#7CCD7C")
390: (0) PALEGREEN4 = ManimColor("#548B54")
391: (0) PALETURQUOISE = ManimColor("#AFEEEE")
392: (0) PALETURQUOISE1 = ManimColor("#BBFFFF")
393: (0) PALETURQUOISE2 = ManimColor("#AEEEEEE")
394: (0) PALETURQUOISE3 = ManimColor("#96CDCD")
395: (0) PALETURQUOISE4 = ManimColor("#668B8B")
396: (0) PALEVIOLETRED = ManimColor("#DB7093")
397: (0) PALEVIOLETRED1 = ManimColor("#FF82AB")
398: (0) PALEVIOLETRED2 = ManimColor("#EE799F")
399: (0) PALEVIOLETRED3 = ManimColor("#CD6889")
400: (0) PALEVIOLETRED4 = ManimColor("#8B475D")
401: (0) PAPAYAWHIP = ManimColor("#FFEF5")
402: (0) PEACHPUFF1 = ManimColor("#FFDAB9")
403: (0) PEACHPUFF2 = ManimColor("#EECBAD")
404: (0) PEACHPUFF3 = ManimColor("#CDAF95")
405: (0) PEACHPUFF4 = ManimColor("#8B7765")
406: (0) PINK = ManimColor("#FFC0CB")
407: (0) PINK1 = ManimColor("#FFB5C5")
408: (0) PINK2 = ManimColor("#EEA9B8")
409: (0) PINK3 = ManimColor("#CD919E")
410: (0) PINK4 = ManimColor("#8B636C")
411: (0) PLUM = ManimColor("#DDA0DD")

```

```
412: (0) PLUM1 = ManimColor("#FFBBFF")
413: (0) PLUM2 = ManimColor("#EEAEEE")
414: (0) PLUM3 = ManimColor("#CD96CD")
415: (0) PLUM4 = ManimColor("#8B668B")
416: (0) POWDERBLUE = ManimColor("#B0E0E6")
417: (0) PURPLE = ManimColor("#A020F0")
418: (0) PURPLE1 = ManimColor("#9B30FF")
419: (0) PURPLE2 = ManimColor("#912CEE")
420: (0) PURPLE3 = ManimColor("#7D26CD")
421: (0) PURPLE4 = ManimColor("#551A8B")
422: (0) RED1 = ManimColor("FF0000")
423: (0) RED2 = ManimColor("EE0000")
424: (0) RED3 = ManimColor("CD0000")
425: (0) RED4 = ManimColor("8B0000")
426: (0) ROSYBROWN = ManimColor("#BC8F8F")
427: (0) ROSYBROWN1 = ManimColor("#FFC1C1")
428: (0) ROSYBROWN2 = ManimColor("#EEB4B4")
429: (0) ROSYBROWN3 = ManimColor("#CD9B9B")
430: (0) ROSYBROWN4 = ManimColor("#8B6969")
431: (0) ROYALBLUE = ManimColor("#4169E1")
432: (0) ROYALBLUE1 = ManimColor("#4876FF")
433: (0) ROYALBLUE2 = ManimColor("#436EEE")
434: (0) ROYALBLUE3 = ManimColor("#3A5FCD")
435: (0) ROYALBLUE4 = ManimColor("#27408B")
436: (0) SADDLEBROWN = ManimColor("#8B4513")
437: (0) SALMON = ManimColor("#FA8072")
438: (0) SALMON1 = ManimColor("#FF8C69")
439: (0) SALMON2 = ManimColor("#EE8262")
440: (0) SALMON3 = ManimColor("#CD7054")
441: (0) SALMON4 = ManimColor("#8B4C39")
442: (0) SANDYBROWN = ManimColor("#F4A460")
443: (0) SEAGREEN1 = ManimColor("#54FF9F")
444: (0) SEAGREEN2 = ManimColor("#4EEE94")
445: (0) SEAGREEN3 = ManimColor("#43CD80")
446: (0) SEAGREEN4 = ManimColor("#2E8B57")
447: (0) SEASHELL1 = ManimColor("#FFF5EE")
448: (0) SEASHELL2 = ManimColor("#EEE5DE")
449: (0) SEASHELL3 = ManimColor("#CDC5BF")
450: (0) SEASHELL4 = ManimColor("#8B8682")
451: (0) SIENNA = ManimColor("#A0522D")
452: (0) SIENNA1 = ManimColor("#FF8247")
453: (0) SIENNA2 = ManimColor("#EE7942")
454: (0) SIENNA3 = ManimColor("#CD6839")
455: (0) SIENNA4 = ManimColor("#8B4726")
456: (0) SKYBLUE = ManimColor("#87CEEB")
457: (0) SKYBLUE1 = ManimColor("#87CEFF")
458: (0) SKYBLUE2 = ManimColor("#7EC0EE")
459: (0) SKYBLUE3 = ManimColor("#6CA6CD")
460: (0) SKYBLUE4 = ManimColor("#4A708B")
461: (0) SLATEBLUE = ManimColor("#6A5ACD")
462: (0) SLATEBLUE1 = ManimColor("#836FFF")
463: (0) SLATEBLUE2 = ManimColor("#7A67EE")
464: (0) SLATEBLUE3 = ManimColor("#6959CD")
465: (0) SLATEBLUE4 = ManimColor("#473C8B")
466: (0) SLATEGRAY = ManimColor("#708090")
467: (0) SLATEGRAY1 = ManimColor("#C6E2FF")
468: (0) SLATEGRAY2 = ManimColor("#B9D3EE")
469: (0) SLATEGRAY3 = ManimColor("#9FB6CD")
470: (0) SLATEGRAY4 = ManimColor("#6C7B8B")
471: (0) SNOW1 = ManimColor("#FFFAFA")
472: (0) SNOW2 = ManimColor("#EEE9E9")
473: (0) SNOW3 = ManimColor("#CDC9C9")
474: (0) SNOW4 = ManimColor("#8B8989")
475: (0) SPRINGGREEN1 = ManimColor("#00FF7F")
476: (0) SPRINGGREEN2 = ManimColor("#00EE76")
477: (0) SPRINGGREEN3 = ManimColor("#00CD66")
478: (0) SPRINGGREEN4 = ManimColor("#008B45")
479: (0) STEELBLUE = ManimColor("#4682B4")
480: (0) STEELBLUE1 = ManimColor("#63B8FF")
```

```

481: (0) STEELBLUE2 = ManimColor("#5CACCE")
482: (0) STEELBLUE3 = ManimColor("#4F94CD")
483: (0) STEELBLUE4 = ManimColor("#36648B")
484: (0) TAN = ManimColor("#D2B48C")
485: (0) TAN1 = ManimColor("#FFA54F")
486: (0) TAN2 = ManimColor("#EE9A49")
487: (0) TAN3 = ManimColor("#CD853F")
488: (0) TAN4 = ManimColor("#8B5A2B")
489: (0) THISTLE = ManimColor("#D8BFD8")
490: (0) THISTLE1 = ManimColor("#FFE1FF")
491: (0) THISTLE2 = ManimColor("#EED2EE")
492: (0) THISTLE3 = ManimColor("#CDB5CD")
493: (0) THISTLE4 = ManimColor("#8B7B8B")
494: (0) TOMATO1 = ManimColor("#FF6347")
495: (0) TOMATO2 = ManimColor("#EE5C42")
496: (0) TOMATO3 = ManimColor("#CD4F39")
497: (0) TOMATO4 = ManimColor("#8B3626")
498: (0) TURQUOISE = ManimColor("#40E0D0")
499: (0) TURQUOISE1 = ManimColor("#00F5FF")
500: (0) TURQUOISE2 = ManimColor("#00E5EE")
501: (0) TURQUOISE3 = ManimColor("#00C5CD")
502: (0) TURQUOISE4 = ManimColor("#00868B")
503: (0) VIOLET = ManimColor("#EE82EE")
504: (0) VIOLETRED = ManimColor("#D02090")
505: (0) VIOLETRED1 = ManimColor("#FF3E96")
506: (0) VIOLETRED2 = ManimColor("#EE3A8C")
507: (0) VIOLETRED3 = ManimColor("#CD3278")
508: (0) VIOLETRED4 = ManimColor("#8B2252")
509: (0) WHEAT = ManimColor("#F5DEB3")
510: (0) WHEAT1 = ManimColor("#FFE7BA")
511: (0) WHEAT2 = ManimColor("#EED8AE")
512: (0) WHEAT3 = ManimColor("#CDBA96")
513: (0) WHEAT4 = ManimColor("#8B7E66")
514: (0) WHITE = ManimColor("#FFFFFF")
515: (0) WHITESMOKE = ManimColor("#F5F5F5")
516: (0) YELLOW1 = ManimColor("#FFFF00")
517: (0) YELLOW2 = ManimColor("#EEEE00")
518: (0) YELLOW3 = ManimColor("#CDCD00")
519: (0) YELLOW4 = ManimColor("#8B8B00")
520: (0) YELLOWGREEN = ManimColor("#9ACD32")

```

---

## File 115 - core.py:

```

1: (0) """Manim's (internal) color data structure and some utilities for
2: (0) color conversion.
3: (0) This module contains the implementation of :class:`.ManimColor`,
4: (0) the data structure internally used to represent colors.
5: (0) The preferred way of using these colors is by importing their constants from
manim:
6: (0) .. code-block:: pycon
7: (4)     >>> from manim import RED, GREEN, BLUE
8: (4)     >>> print(RED)
9: (0) Note this way uses the name of the colors in UPPERCASE.
10: (0) .. note::
11: (4)     The colors of type "C" have an alias equal to the colordname without a
letter,
12: (4)     e.g. GREEN = GREEN_C
13: (0)
14: (0) from __future__ import annotations
15: (0) import colorsys
16: (0) import random
17: (0) import re
18: (0) from typing import Any, Sequence, TypeVar, Union, overload
19: (0) import numpy as np
20: (0) import numpy.typing as npt
21: (0) from typing_extensions import Self, TypeAlias
22: (0) from manim.typing import (

```

```

23: (4)                 HSV_Array_Float,
24: (4)                 HSV_Tuple_Float,
25: (4)                 ManimColorDType,
26: (4)                 ManimColorInternal,
27: (4)                 RGB_Array_Float,
28: (4)                 RGB_Array_Int,
29: (4)                 RGB_Tuple_Float,
30: (4)                 RGB_Tuple_Int,
31: (4)                 RGBA_Array_Float,
32: (4)                 RGBA_Array_Int,
33: (4)                 RGBA_Tuple_Float,
34: (4)                 RGBA_Tuple_Int,
35: (0)
36: (0)             )
37: (0)         from ...utils.space_ops import normalize
38: (0)         re_hex = re.compile("((?<=#)|(?:0x))[A-F0-9]{6,8}", re.IGNORECASE)
39: (0)     class ManimColor:
40: (4)         """Internal representation of a color.
41: (4)         The ManimColor class is the main class for the representation of a color.
42: (4)         It's internal representation is a 4 element array of floats corresponding
43: (4)         to a [r,g,b,a] value where r,g,b,a can be between 0 to 1.
44: (4)         This is done in order to reduce the amount of color inconsistencies by
45: (4)         constantly
46: (4)             casting between integers and floats which introduces errors.
47: (4)             The class can accept any value of type :class:`ParsableManimColor` i.e.
48: (4)             ManimColor, int, str, RGB_Tuple_Int, RGB_Tuple_Float, RGBA_Tuple_Int,
49: (4)             RGBA_Tuple_Float, RGB_Array_Int,
50: (4)                 RGB_Array_Float, RGBA_Array_Int, RGBA_Array_Float
51: (4)             ManimColor itself only accepts singular values and will directly interpret
52: (4)             them into a single color if possible
53: (4)             Be careful when passing strings to ManimColor it can create a big overhead
54: (4)             for the color processing.
55: (4)             If you want to parse a list of colors use the function :meth:`parse` in
56: (4)             :class:`ManimColor` which assumes that
57: (4)                 you are going to pass a list of color so arrays will not be interpreted as
58: (4)                 a single color.
59: (4)             .. warning::
60: (4)                 If you pass an array of numbers to :meth:`parse` it will interpret the
61: (4)                 r,g,b,a numbers in that array as colors
62: (4)                 so instead of the expect singular color you get and array with 4
63: (4)                 colors.
64: (4)             For conversion behaviors see the ``_internal`` functions for further
65: (4)             documentation
66: (4)             You can create a ``ManimColor`` instance via its classmethods. See the
67: (4)             respective methods for more info.
68: (4)             .. code-block:: python
69: (4)                 mycolor = ManimColor.from_rgb((0, 1, 0.4, 0.5))
70: (4)                 myothercolor = ManimColor.from_rgb((153, 255, 255))
71: (4)             You can also convert between different color spaces:
72: (4)             .. code-block:: python
73: (4)                 mycolor_hex = mycolor.to_hex()
74: (4)                 myoriginalcolor = ManimColor.from_hex(mycolor_hex).to_hsv()
75: (4)             Parameters
76: (4)             -----
77: (4)             value
78: (4)                 Some representation of a color (e.g., a string or
79: (4)                 a suitable tuple). The default ``None`` is ``BLACK``.
80: (4)             alpha
81: (4)                 The opacity of the color. By default, colors are
82: (4)                 fully opaque (value 1.0).
83: (4)             """
84: (4)         def __init__(
85: (4)             self,
86: (4)             value: ParsableManimColor | None,
87: (4)             alpha: float = 1.0,
88: (4)         ) -> None:
89: (4)             if value is None:
90: (4)                 self._internal_value = np.array((0, 0, 0, alpha),
91: (4)             elif isinstance(value, ManimColor):

```

```

81: (12)                         self._internal_value = value._internal_value
82: (8)                          elif isinstance(value, int):
83: (12)                            self._internal_value = ManimColor._internal_from_integer(value,
alpha)
84: (8)                          elif isinstance(value, str):
85: (12)                            result = re_hex.search(value)
86: (12)                            if result is not None:
87: (16)                              self._internal_value = ManimColor._internal_from_hex_string(
88: (20)                                result.group(), alpha
89: (16)                            )
90: (12)                          else:
91: (16)                            self._internal_value = ManimColor._internal_from_string(value)
92: (8)                          elif isinstance(value, (list, tuple, np.ndarray)):
93: (12)                            length = len(value)
94: (12)                            if all(isinstance(x, float) for x in value):
95: (16)                              if length == 3:
96: (20)                                self._internal_value =
ManimColor._internal_from_rgb(value, alpha) # type: ignore
97: (16)                              elif length == 4:
98: (20)                                self._internal_value =
ManimColor._internal_from_rgba(value) # type: ignore
99: (16)                            else:
100: (20)                              raise ValueError(
101: (24)                                f"ManimColor only accepts lists/tuples/arrays of
length 3 or 4, not {length}"
102: (20)                            )
103: (12)                          else:
104: (16)                            if length == 3:
105: (20)                              self._internal_value = ManimColor._internal_from_int_rgb(
106: (24)                                value, alpha # type: ignore
107: (20)                            )
108: (16)                            elif length == 4:
109: (20)                              self._internal_value =
ManimColor._internal_from_int_rgba(value) # type: ignore
110: (16)                            else:
111: (20)                              raise ValueError(
112: (24)                                f"ManimColor only accepts lists/tuples/arrays of
length 3 or 4, not {length}"
113: (20)                            )
114: (8)                          elif hasattr(value, "get_hex") and callable(value.get_hex):
115: (12)                            result = re_hex.search(value.get_hex())
116: (12)                            if result is None:
117: (16)                              raise ValueError(f"Failed to parse a color from {value}")
118: (12)                            self._internal_value = ManimColor._internal_from_hex_string(
119: (16)                              result.group(), alpha
120: (12)                            )
121: (8)                          else:
122: (12)                            raise TypeError(
123: (16)                              "ManimColor only accepts int, str, list[int, int, int], "
124: (16)                              "list[int, int, int, int], list[float, float, float], "
125: (16)                              f"list[float, float, float, float], not {type(value)}"
126: (12)                            )
127: (4) @property
128: (4) def _internal_value(self) -> ManimColorInternal:
129: (8)     """Returns the internal value of the current Manim color [r,g,b,a]
float array
130: (8)     Returns
131: (8)     -----
132: (8)     ManimColorInternal
133: (12)       internal color representation
134: (8)     """
135: (8)     return self.__value
136: (4) @_internal_value.setter
137: (4) def _internal_value(self, value: ManimColorInternal) -> None:
138: (8)     """Overwrites the internal color value of the ManimColor object
Parameters
139: (8)     -----
140: (8)     value : ManimColorInternal
141: (8)       The value which will overwrite the current color
142: (12)

```

```

143: (8)             Raises
144: (8)
145: (8)
146: (12)            -----
147: (8)             TypeError
148: (8)             Raises a TypeError if an invalid array is passed
149: (12)            """
150: (8)             if not isinstance(value, np.ndarray):
151: (12)                 raise TypeError("value must be a numpy array")
152: (8)             if value.shape[0] != 4:
153: (12)                 raise TypeError("Array must have 4 values exactly")
154: (8)             self._value: ManimColorInternal = value
155: (4)             @staticmethod
156: (4)             def _internal_from_integer(value: int, alpha: float) ->
157: (12)             ManimColorInternal:
158: (16)                 return np.asarray(
159: (16)                     (
160: (16)                         ((value >> 16) & 0xFF) / 255,
161: (12)                         ((value >> 8) & 0xFF) / 255,
162: (12)                         ((value >> 0) & 0xFF) / 255,
163: (8)                         alpha,
164: (4)                     ),
165: (4)                     dtype=ManimColorDType,
166: (8)             @staticmethod
167: (8)             def _internal_from_hex_string(hex: str, alpha: float) ->
168: (12)             ManimColorInternal:
169: (12)                 """Internal function for converting a hex string into the internal
170: (8)                 representation of a ManimColor.
171: (8)                 .. warning::
172: (12)                     This does not accept any prefixes like # or similar in front of
173: (8)                     the hex string.
174: (12)                     This is just intended for the raw hex part
175: (8)                     *For internal use only*
176: (8)             Parameters
177: (8)             -----
178: (8)             hex : str
179: (8)             hex string to be parsed
180: (12)             alpha : float
181: (12)             alpha value used for the color
182: (8)             Returns
183: (8)             -----
184: (8)             ManimColorInternal
185: (8)             Internal color representation
186: (12)            """
187: (16)             if len(hex) == 6:
188: (16)                 hex += "00"
189: (16)             tmp = int(hex, 16)
190: (16)             return np.asarray(
191: (12)                 (
192: (12)                     ((tmp >> 24) & 0xFF) / 255,
193: (8)                     ((tmp >> 16) & 0xFF) / 255,
194: (4)                     ((tmp >> 8) & 0xFF) / 255,
195: (4)                     alpha,
196: (8)                 ),
197: (4)                 dtype=ManimColorDType,
198: (8)             @staticmethod
199: (4)             def _internal_from_int_rgb(
200: (8)                 rgb: RGB_Tuple_Int, alpha: float = 1.0
201: (8)             ) -> ManimColorInternal:
202: (8)                 """Internal function for converting a rgb tuple of integers into the
203: (12)                 internal representation of a ManimColor.
204: (8)                 *For internal use only*
205: (12)             Parameters
206: (8)             -----
207: (8)             rgb : RGB_Tuple_Int
208: (8)             integer rgb tuple to be parsed
209: (8)             alpha : float, optional
210: (12)             optional alpha value, by default 1.0
211: (8)             Returns

```

```

207: (8)           -----
208: (8)           ManimColorInternal
209: (12)          Internal color representation
210: (8)          """
211: (8)          value: np.ndarray = np.asarray(rgb, dtype=ManimColorDType).copy() /
212: (8)          value.resize(4, refcheck=False)
213: (8)          value[3] = alpha
214: (8)          return value
215: (4)          @staticmethod
216: (4)          def _internal_from_rgb(
217: (8)              rgb: RGB_Tuple_Float, alpha: float = 1.0
218: (4)          ) -> ManimColorInternal:
219: (8)              """Internal function for converting a rgb tuple of floats into the
internal representation of a ManimColor.
220: (8)          *For internal use only*
221: (8)          Parameters
222: (8)          -----
223: (8)          rgb : RGB_Tuple_Float
224: (12)         float rgb tuple to be parsed
225: (8)          alpha : float, optional
226: (12)         optional alpha value, by default 1.0
227: (8)          Returns
228: (8)          -----
229: (8)          ManimColorInternal
230: (12)         Internal color representation
231: (8)          """
232: (8)          value: np.ndarray = np.asarray(rgb, dtype=ManimColorDType).copy()
233: (8)          value.resize(4, refcheck=False)
234: (8)          value[3] = alpha
235: (8)          return value
236: (4)          @staticmethod
237: (4)          def _internal_from_int_rgba(rgb: RGBA_Tuple_Int) -> ManimColorInternal:
238: (8)              """Internal function for converting a rgba tuple of integers into the
internal representation of a ManimColor.
239: (8)          *For internal use only*
240: (8)          Parameters
241: (8)          -----
242: (8)          rgba : RGBA_Tuple_Int
243: (12)         int rgba tuple to be parsed
244: (8)          Returns
245: (8)          -----
246: (8)          ManimColorInternal
247: (12)         Internal color representation
248: (8)          """
249: (8)          return np.asarray(rgb, dtype=ManimColorDType) / 255
250: (4)          @staticmethod
251: (4)          def _internal_from_rgba(rgb: RGBA_Tuple_Float) -> ManimColorInternal:
252: (8)              """Internal function for converting a rgba tuple of floats into the
internal representation of a ManimColor.
253: (8)          *For internal use only*
254: (8)          Parameters
255: (8)          -----
256: (8)          rgba : RGBA_Tuple_Float
257: (12)         int rgba tuple to be parsed
258: (8)          Returns
259: (8)          -----
260: (8)          ManimColorInternal
261: (12)         Internal color representation
262: (8)          """
263: (8)          return np.asarray(rgb, dtype=ManimColorDType)
264: (4)          @staticmethod
265: (4)          def _internal_from_string(name: str) -> ManimColorInternal:
266: (8)              """Internal function for converting a string into the internal
representation of a ManimColor.
267: (8)          This is not used for hex strings, please refer to
:meth:`_internal_from_hex` for this functionality.
268: (8)          *For internal use only*
269: (8)          Parameters

```

```

270: (8)           -----
271: (8)           name : str
272: (12)          The color name to be parsed into a color. Refer to the different
color Modules in the documentation Page to
273: (12)          find the corresponding Color names.
274: (8)          Returns
275: (8)           -----
276: (8)           ManimColorInternal
277: (12)          Internal color representation
278: (8)          Raises
279: (8)           -----
280: (8)          ValueError
281: (12)          Raises a ValueError if the color name is not present with manim
282: (8)           """
283: (8)           from . import _all_color_dict
284: (8)           upper_name = name.upper()
285: (8)           if upper_name in _all_color_dict:
286: (12)             return _all_color_dict[upper_name]._internal_value
287: (8)           else:
288: (12)             raise ValueError(f"Color {name} not found")
289: (4)           def to_integer(self) -> int:
290: (8)             """Converts the current ManimColor into an integer
291: (8)             Returns
292: (8)             -----
293: (8)             int
294: (12)               integer representation of the color
295: (8)             .. warning::
296: (12)               This will return only the rgb part of the color
297: (8)             """
298: (8)             return int.from_bytes(
299: (12)               (self._internal_value[:3] * 255).astype(int).tobytes(), "big"
300: (8)             )
301: (4)           def to_rgb(self) -> RGB_Array_Float:
302: (8)             """Converts the current ManimColor into a rgb array of floats
303: (8)             Returns
304: (8)             -----
305: (8)             RGB_Array_Float
306: (12)               rgb array with 3 elements of type float
307: (8)             """
308: (8)             return self._internal_value[:3]
309: (4)           def to_int_rgb(self) -> RGB_Array_Int:
310: (8)             """Converts the current ManimColor into a rgb array of int
311: (8)             Returns
312: (8)             -----
313: (8)             RGB_Array_Int
314: (12)               rgb array with 3 elements of type int
315: (8)             """
316: (8)             return (self._internal_value[:3] * 255).astype(int)
317: (4)           def to_rgba(self) -> RGBA_Array_Float:
318: (8)             """Converts the current ManimColor into a rgba array of floats
319: (8)             Returns
320: (8)             -----
321: (8)             RGBA_Array_Float
322: (12)               rgba array with 4 elements of type float
323: (8)             """
324: (8)             return self._internal_value
325: (4)           def to_int_rgba(self) -> RGBA_Array_Int:
326: (8)             """Converts the current ManimColor into a rgba array of int
327: (8)             Returns
328: (8)             -----
329: (8)             RGBA_Array_Int
330: (12)               rgba array with 4 elements of type int
331: (8)             """
332: (8)             return (self._internal_value * 255).astype(int)
333: (4)           def to_rgba_with_alpha(self, alpha: float) -> RGBA_Array_Float:
334: (8)             """Converts the current ManimColor into a rgba array of float as
:meth:`to_rgba` but you can change the alpha
335: (8)               value.
336: (8)               Parameters

```

```

337: (8)           -----
338: (8)           alpha : float
339: (12)          alpha value to be used in the return value
340: (8)          Returns
341: (8)          -----
342: (8)           RGBA_Array_Float
343: (12)          rgba array with 4 elements of type float
344: (8)          """
345: (8)          return np.fromiter((*self._internal_value[:3], alpha),
dtype=ManimColorDType)
346: (4)          def to_int_rgba_with_alpha(self, alpha: float) -> RGBA_Array_Int:
347: (8)          """Converts the current ManimColor into a rgba array of integers as
:meth:`to_int_rgba` but you can change the alpha
348: (8)          value.
349: (8)          Parameters
350: (8)          -----
351: (8)          alpha : float
352: (12)          alpha value to be used for the return value. (Will automatically
be scaled from 0-1 to 0-255 so just pass 0-1)
353: (8)          Returns
354: (8)          -----
355: (8)           RGBA_Array_Int
356: (12)          rgba array with 4 elements of type int
357: (8)          """
358: (8)          tmp = self._internal_value * 255
359: (8)          tmp[3] = alpha * 255
360: (8)          return tmp.astype(int)
361: (4)          def to_hex(self, with_alpha: bool = False) -> str:
362: (8)          """Converts the manim color to a hexadecimal representation of the
color
363: (8)          Parameters
364: (8)          -----
365: (8)          with_alpha : bool, optional
366: (12)          Changes the result from 6 to 8 values where the last 2 nibbles
represent the alpha value of 0-255,
367: (12)          by default False
368: (8)          Returns
369: (8)          -----
370: (8)          str
371: (12)          A hex string starting with a # with either 6 or 8 nibbles
depending on your input, by default 6 i.e #XXXXXX
372: (8)          """
373: (8)          tmp = (
374: (12)          f"#{{int(self._internal_value[0] * 255):02X}}"
375: (12)          f"#{{int(self._internal_value[1] * 255):02X}}"
376: (12)          f"#{{int(self._internal_value[2] * 255):02X}}"
377: (8)          )
378: (8)          if with_alpha:
379: (12)          tmp += f"#{{int(self._internal_value[3] * 255):02X}}"
380: (8)          return tmp
381: (4)          def to_hsv(self) -> HSV_Array_Float:
382: (8)          """Converts the Manim Color to HSV array.
383: (8)          .. note::
384: (11)          Be careful this returns an array in the form `[h, s, v]` where the
elements are floats.
385: (11)          This might be confusing because rgb can also be an array of floats
so you might want to annotate the usage
386: (11)          of this function in your code by typing the variables with
:class:`HSV_Array_Float` in order to differentiate
387: (11)          between rgb arrays and hsv arrays
388: (8)          Returns
389: (8)          -----
390: (8)           HSV_Array_Float
391: (12)          A hsv array containing 3 elements of type float ranging from 0 to
1
392: (8)          """
393: (8)          return colorsys.rgb_to_hsv(*self.to_rgb())
394: (4)          def invert(self, with_alpha=False) -> ManimColor:
395: (8)          """Returns an linearly inverted version of the color (no inplace

```

```

changes)
396: (8)          Parameters
397: (8)          -----
398: (8)          with_alpha : bool, optional
399: (12)          if true the alpha value will be inverted too, by default False
400: (12)          .. note::
401: (16)          This can result in unintended behavior where objects are not
displayed because their alpha
402: (16)          value is suddenly 0 or very low. Please keep that in mind when
setting this to true
403: (8)          Returns
404: (8)          -----
405: (8)          ManimColor
406: (12)          The linearly inverted ManimColor
407: (8)
408: (8)          """
409: (4)          return ManimColor(1.0 - self._internal_value, with_alpha)
410: (8)          def interpolate(self, other: ManimColor, alpha: float) -> ManimColor:
411: (8)          """Interpolates between the current and the given ManimColor and
412: (8)          returns the interpolated color
413: (8)          Parameters
414: (8)          -----
415: (8)          other : ManimColor
416: (12)          The other ManimColor to be used for interpolation
417: (12)          alpha : float
418: (12)          A point on the line in rgba colorspace connecting the two colors
419: (8)          i.e. the interpolation point
420: (8)          0 corresponds to the current ManimColor and 1 corresponds to the
421: (8)          other ManimColor
422: (8)          Returns
423: (8)          -----
424: (12)          ManimColor
425: (8)          The interpolated ManimColor
426: (8)
427: (4)          """
428: (8)          return ManimColor(
429: (12)          self._internal_value * (1 - alpha) + other._internal_value * alpha
430: (8)          )
431: (4)          @classmethod
432: (4)          def from_rgb(
433: (8)          cls,
434: (8)          rgb: RGB_Array_Float | RGB_Tuple_Float | RGB_Array_Int |
435: (8)          alpha: float = 1.0,
436: (8)          ) -> Self:
437: (8)          """Creates a ManimColor from an RGB Array. Automagically decides which
438: (8)          type it is int/float
439: (8)          .. warning::
440: (8)          Please make sure that your elements are not floats if you want
441: (8)          integers. A 5.0 will result in the input
442: (8)          being interpreted as if it was a float rgb array with the value
443: (8)          5.0 and not the integer 5
444: (8)          Parameters
445: (8)          -----
446: (8)          rgb : RGB_Array_Float | RGB_Tuple_Float | RGB_Array_Int |
447: (8)          Any 3 Element Iterable
448: (8)          alpha : float, optional
449: (8)          alpha value to be used in the color, by default 1.0
450: (8)          Returns
451: (8)          -----
452: (8)          ManimColor
453: (8)          Returns the ManimColor object
454: (8)
455: (8)          """
456: (8)          return cls(rgb, alpha)
457: (4)          @classmethod
458: (4)          def from_rgba(
459: (8)          cls, rgba: RGBA_Array_Float | RGBA_Tuple_Float | RGBA_Array_Int |
460: (8)          alpha: float = 1.0,
461: (8)          ) -> Self:
462: (8)          """Creates a ManimColor from an RGBA Array. Automagically decides

```

```

which type it is int/float
453: (8)          .. warning::
454: (12)          Please make sure that your elements are not floats if you want
integers. A 5.0 will result in the input
455: (12)          being interpreted as if it was a float rgb array with the value
5.0 and not the integer 5
456: (8)          Parameters
457: (8)          -----
458: (8)          rgba : RGBA_Array_Float | RGBA_Tuple_Float | RGBA_Array_Int |
RGBATuple_Int
459: (12)          Any 4 Element Iterable
460: (8)          Returns
461: (8)          -----
462: (8)          ManimColor
463: (12)          Returns the ManimColor object
464: (8)          """
465: (8)          return cls(rgba)
466: (4)          @classmethod
467: (4)          def from_hex(cls, hex: str, alpha: float = 1.0) -> Self:
468: (8)          """Creates a Manim Color from a hex string, prefixes allowed # and 0x
469: (8)          Parameters
470: (8)          -----
471: (8)          hex : str
472: (12)          The hex string to be converted (currently only supports 6 nibbles)
473: (8)          alpha : float, optional
474: (12)          alpha value to be used for the hex string, by default 1.0
475: (8)          Returns
476: (8)          -----
477: (8)          ManimColor
478: (12)          The ManimColor represented by the hex string
479: (8)          """
480: (8)          return cls(hex, alpha)
481: (4)          @classmethod
482: (4)          def from_hsv(
483: (8)          cls, hsv: HSV_Array_Float | HSV_Tuple_Float, alpha: float = 1.0
484: (4)          ) -> Self:
485: (8)          """Creates a ManimColor from an HSV Array
486: (8)          Parameters
487: (8)          -----
488: (8)          hsv : HSV_Array_Float | HSV_Tuple_Float
489: (12)          Any 3 Element Iterable containing floats from 0-1
490: (8)          alpha : float, optional
491: (12)          the alpha value to be used, by default 1.0
492: (8)          Returns
493: (8)          -----
494: (8)          ManimColor
495: (12)          The ManimColor with the corresponding RGB values to the HSV
496: (8)          """
497: (8)          rgb = colorsys.hsv_to_rgb(*hsv)
498: (8)          return cls(rgb, alpha)
499: (4)          @overload
500: (4)          @classmethod
501: (4)          def parse(
502: (8)          cls,
503: (8)          color: ParsableManimColor | None,
504: (8)          alpha: float = ...,
505: (4)          ) -> Self: ...
506: (4)          @overload
507: (4)          @classmethod
508: (4)          def parse(
509: (8)          cls,
510: (8)          color: Sequence[ParsableManimColor],
511: (8)          alpha: float = ...,
512: (4)          ) -> list[Self]: ...
513: (4)          @classmethod
514: (4)          def parse(
515: (8)          cls,
516: (8)          color: ParsableManimColor | list[ParsableManimColor] | None,
517: (8)          alpha: float = 1.0,

```

```

518: (4)             ) -> Self | list[Self]:
519: (8)
520: (8)             """
521: (8)             Handles the parsing of a list of colors or a single color.
522: (8)             Parameters
523: (8)             -----
524: (8)             color
524: (12)            The color or list of colors to parse. Note that this function can
not accept rgba tuples. It will assume that you mean list[ManimColor] and will return a list of
ManimColors.
525: (8)             alpha
526: (12)            The alpha value to use if a single color is passed. or if a list
of colors is passed to set the value of all colors.
527: (8)             Returns
528: (8)             -----
529: (8)             ManimColor
530: (12)            Either a list of colors or a singular color depending on the input
531: (8)             """
532: (8)             if isinstance(color, (list, tuple)):
533: (12)                return [cls(c, alpha) for c in color] # type: ignore
534: (8)                return cls(color, alpha) # type: ignore
535: (4)             @staticmethod
536: (4)             def gradient(colors: list[ManimColor], length: int):
537: (8)               """This is not implemented by now refer to :func:`color_gradient` for
a working implementation for now"""
538: (8)               raise NotImplementedError
539: (4)             def __repr__(self) -> str:
540: (8)               return f"{self.__class__.__name__}({self.to_hex()})"
541: (4)             def __str__(self) -> str:
542: (8)               return f"{self.to_hex()}"
543: (4)             def __eq__(self, other: object) -> bool:
544: (8)               if not isinstance(other, ManimColor):
545: (12)                 raise TypeError(
546: (16)                   f"Cannot compare {self.__class__.__name__} with
{other.__class__.__name__}"
547: (12)                 )
548: (8)               return np.allclose(self._internal_value, other._internal_value)
549: (4)             def __add__(self, other: ManimColor) -> ManimColor:
550: (8)               return ManimColor(self._internal_value + other._internal_value)
551: (4)             def __sub__(self, other: ManimColor) -> ManimColor:
552: (8)               return ManimColor(self._internal_value - other._internal_value)
553: (4)             def __mul__(self, other: ManimColor) -> ManimColor:
554: (8)               return ManimColor(self._internal_value * other._internal_value)
555: (4)             def __truediv__(self, other: ManimColor) -> ManimColor:
556: (8)               return ManimColor(self._internal_value / other._internal_value)
557: (4)             def __floordiv__(self, other: ManimColor) -> ManimColor:
558: (8)               return ManimColor(self._internal_value // other._internal_value)
559: (4)             def __mod__(self, other: ManimColor) -> ManimColor:
560: (8)               return ManimColor(self._internal_value % other._internal_value)
561: (4)             def __pow__(self, other: ManimColor) -> ManimColor:
562: (8)               return ManimColor(self._internal_value**other._internal_value)
563: (4)             def __and__(self, other: ManimColor) -> ManimColor:
564: (8)               return ManimColor(self.to_integer() & other.to_integer())
565: (4)             def __or__(self, other: ManimColor) -> ManimColor:
566: (8)               return ManimColor(self.to_integer() | other.to_integer())
567: (4)             def __xor__(self, other: ManimColor) -> ManimColor:
568: (8)               return ManimColor(self.to_integer() ^ other.to_integer())
569: (0)             ParsableManimColor: TypeAlias = Union[
570: (4)               ManimColor,
571: (4)               int,
572: (4)               str,
573: (4)               RGB_Tuple_Int,
574: (4)               RGB_Tuple_Float,
575: (4)               RGBA_Tuple_Int,
576: (4)               RGBA_Tuple_Float,
577: (4)               RGB_Array_Int,
578: (4)               RGB_Array_Float,
579: (4)               RGBA_Array_Int,
580: (4)               RGBA_Array_Float,
581: (0)             ]

```

```

582: (0)         """`ParsableManimColor` represents all the types which can be parsed
583: (0)         to a color in Manim.
584: (0)
585: (0)         ManimColorT = TypeVar("ManimColorT", bound=ManimColor)
586: (0)         def color_to_rgb(color: ParsableManimColor) -> RGB_Array_Float:
587: (4)             """Helper function for use in functional style programming.
588: (4)             Refer to :meth:`to_rgb` in :class:`ManimColor`.
589: (4)             Parameters
590: (4)             -----
591: (4)             color : ParsableManimColor
592: (8)                 A color
593: (4)             Returns
594: (4)             -----
595: (4)             RGB_Array_Float
596: (8)                 the corresponding rgb array
597: (4)
598: (4)             return ManimColor(color).to_rgb()
599: (0)         def color_to_rgba(color: ParsableManimColor, alpha: float = 1) ->
600: (4)             """Helper function for use in functional style programming refer to
:meth:`to_rgba_with_alpha` in :class:`ManimColor`
601: (4)             Parameters
602: (4)             -----
603: (4)             color : ParsableManimColor
604: (8)                 A color
605: (4)             alpha : float, optional
606: (8)                 alpha value to be used in the color, by default 1
607: (4)             Returns
608: (4)             -----
609: (4)             RGBA_Array_Float
610: (8)                 the corresponding rgba array
611: (4)
612: (4)             return ManimColor(color).to_rgba_with_alpha(alpha)
613: (0)         def color_to_int_rgb(color: ParsableManimColor) -> RGB_Array_Int:
614: (4)             """Helper function for use in functional style programming refer to
:meth:`to_int_rgb` in :class:`ManimColor`
615: (4)             Parameters
616: (4)             -----
617: (4)             color : ParsableManimColor
618: (8)                 A color
619: (4)             Returns
620: (4)             -----
621: (4)             RGB_Array_Int
622: (8)                 the corresponding int rgb array
623: (4)
624: (4)             return ManimColor(color).to_int_rgb()
625: (0)         def color_to_int_rgba(color: ParsableManimColor, alpha: float = 1.0) ->
626: (4)             """Helper function for use in functional style programming refer to
:meth:`to_int_rgba_with_alpha` in :class:`ManimColor`
627: (4)             Parameters
628: (4)             -----
629: (4)             color : ParsableManimColor
630: (8)                 A color
631: (4)             alpha : float, optional
632: (8)                 alpha value to be used in the color, by default 1.0
633: (4)             Returns
634: (4)             -----
635: (4)             RGBA_Array_Int
636: (8)                 the corresponding int rgba array
637: (4)
638: (4)             return ManimColor(color).to_int_rgba_with_alpha(alpha)
639: (0)         def rgb_to_color(
640: (4)             rgb: RGB_Array_Float | RGB_Tuple_Float | RGB_Array_Int | RGB_Tuple_Int,
641: (0)         ) -> ManimColor:
642: (4)             """Helper function for use in functional style programming refer to
:meth:`from_rgb` in :class:`ManimColor`
643: (4)             Parameters
644: (4)             -----

```

```

645: (4)             rgb : RGB_Array_Float | RGB_Tuple_Float
646: (8)             A 3 element iterable
647: (4)             Returns
648: (4)
649: (4)             ManimColor
650: (8)             A ManimColor with the corresponding value
651: (4)
652: (4)             """
653: (0)             return ManimColor.from_rgb(rgb)
def rgba_to_color(
654: (4)             rgba: RGBA_Array_Float | RGBA_Tuple_Float | RGBA_Array_Int |
655: (0)             ) -> ManimColor:
656: (4)             """Helper function for use in functional style programming refer to
:meth:`from_rgba` in :class:`ManimColor`
657: (4)             Parameters
658: (4)
659: (4)             rgba : RGBA_Array_Float | RGBA_Tuple_Float
660: (8)             A 4 element iterable
661: (4)             Returns
662: (4)
663: (4)             ManimColor
664: (8)             A ManimColor with the corresponding value
665: (4)
666: (4)             """
667: (0)             return ManimColor.from_rgba(rgba)
def rgb_to_hex(
668: (4)             rgb: RGB_Array_Float | RGB_Tuple_Float | RGB_Array_Int | RGB_Tuple_Int,
669: (0)             ) -> str:
670: (4)             """Helper function for use in functional style programming refer to
:meth:`from_rgb` in :class:`ManimColor`
671: (4)             Parameters
672: (4)
673: (4)             rgb : RGB_Array_Float | RGB_Tuple_Float
674: (8)             A 3 element iterable
675: (4)             Returns
676: (4)
677: (4)             str
678: (8)             A hex representation of the color, refer to :meth:`to_hex` in
:class:`ManimColor`
679: (4)
680: (4)             """
681: (0)             return ManimColor.from_rgb(rgb).to_hex()
def hex_to_rgb(hex_code: str) -> RGB_Array_Float:
682: (4)             """Helper function for use in functional style programming refer to
:meth:`to_hex` in :class:`ManimColor`
683: (4)             Parameters
684: (4)
685: (4)             hex_code : str
686: (8)             A hex string representing a color
687: (4)             Returns
688: (4)
689: (4)             RGB_Array_Float
690: (8)             RGB array representing the color
691: (4)
692: (4)             """
693: (0)             return ManimColor(hex_code).to_rgb()
def invert_color(color: ManimColorT) -> ManimColorT:
694: (4)             """Helper function for use in functional style programming refer to
:meth:`invert` in :class:`ManimColor`
695: (4)             Parameters
696: (4)
697: (4)             color : ManimColor
698: (8)             A ManimColor
699: (4)             Returns
700: (4)
701: (4)             ManimColor
702: (8)             The linearly inverted ManimColor
703: (4)
704: (4)             """
705: (0)             return color.invert()
def interpolate_arrays(
706: (4)             arr1: npt.NDArray[Any], arr2: npt.NDArray[Any], alpha: float
707: (0)             ) -> np.ndarray:

```

```

708: (4)             """Helper function used in Manim to fade between two objects smoothly
709: (4)             Parameters
710: (4)             -----
711: (4)             arr1 : npt.NDArray[Any]
712: (8)                 The first array of colors
713: (4)             arr2 : npt.NDArray[Any]
714: (8)                 The second array of colors
715: (4)             alpha : float
716: (8)                 The alpha value corresponding to the interpolation point between the
two inputs
717: (4)             Returns
718: (4)             -----
719: (4)             np.ndarray
720: (8)                 The interpolated value of the to arrays
721: (4)             """
722: (4)             return (1 - alpha) * arr1 + alpha * arr2
723: (0)             def color_gradient(
724: (4)                 reference_colors: Sequence[ParsableManimColor],
725: (4)                 length_of_output: int,
726: (0)             ) -> list[ManimColor] | ManimColor:
727: (4)                 """Creates a list of colors interpolated between the input array of colors
with a specific number of colors
728: (4)                 Parameters
729: (4)                 -----
730: (4)                 reference_colors : Sequence[ParsableManimColor]
731: (8)                     The colors to be interpolated between or spread apart
732: (4)                 length_of_output : int
733: (8)                     The number of colors that the output should have, ideally more than
the input
734: (4)             Returns
735: (4)             -----
736: (4)             list[ManimColor] | ManimColor
737: (8)                 A list of ManimColor's which has the interpolated colors
738: (4)             """
739: (4)             if length_of_output == 0:
740: (8)                 return ManimColor(reference_colors[0])
741: (4)             if len(reference_colors) == 1:
742: (8)                 return [ManimColor(reference_colors[0])] * length_of_output
743: (4)             rgbs = [color_to_rgb(color) for color in reference_colors]
744: (4)             alphas = np.linspace(0, (len(rgbs) - 1), length_of_output)
745: (4)             floors = alphas.astype("int")
746: (4)             alphas_mod1 = alphas % 1
747: (4)             alphas_mod1[-1] = 1
748: (4)             floors[-1] = len(rgbs) - 2
749: (4)             return [
750: (8)                 rgb_to_color((rgbs[i] * (1 - alpha)) + (rgbs[i + 1] * alpha))
751: (8)                 for i, alpha in zip(floors, alphas_mod1)
752: (4)             ]
753: (0)             def interpolate_color(
754: (4)                 color1: ManimColorT, color2: ManimColor, alpha: float
755: (0)             ) -> ManimColorT:
756: (4)                 """Standalone function to interpolate two ManimColors and get the result
refer to :meth:`interpolate` in :class:`ManimColor`
757: (4)                 Parameters
758: (4)                 -----
759: (4)                 color1 : ManimColor
760: (8)                     First ManimColor
761: (4)                 color2 : ManimColor
762: (8)                     Second ManimColor
763: (4)                 alpha : float
764: (8)                     The alpha value determining the point of interpolation between the
colors
765: (4)             Returns
766: (4)             -----
767: (4)             ManimColor
768: (8)                 The interpolated ManimColor
769: (4)             """
770: (4)             return color1.interpolate(color2, alpha)
771: (0)             def average_color(*colors: ParsableManimColor) -> ManimColor:

```

```

772: (4)             """Determines the Average color of the given parameters
773: (4)             Returns
774: (4)             -----
775: (4)             ManimColor
776: (8)             The average color of the input
777: (4)
778: (4)             """
779: (4)             rgbs = np.array([color_to_rgb(color) for color in colors])
780: (4)             mean_rgb = np.apply_along_axis(np.mean, 0, rgbs)
781: (0)             return rgb_to_color(mean_rgb)
782: (4)         def random_bright_color() -> ManimColor:
783: (4)             """Returns you a random bright color
784: (4)             .. warning::
785: (8)                 This operation is very expensive please keep in mind the performance
loss.
786: (4)             Returns
787: (4)             -----
788: (4)             ManimColor
789: (8)             A bright ManimColor
790: (4)
791: (4)             """
792: (4)             curr_rgb = color_to_rgb(random_color())
793: (0)             new_rgb = interpolate_arrays(curr_rgb, np.ones(len(curr_rgb)), 0.5)
794: (4)             return ManimColor(new_rgb)
795: (4)
796: (8)         def random_color() -> ManimColor:
797: (4)             """Return you a random ManimColor
798: (4)             .. warning::
799: (8)                 This operation is very expensive please keep in mind the performance
loss.
800: (4)             Returns
801: (4)             -----
802: (4)             ManimColor
803: (8)             _description_
804: (0)
805: (4)         def get_shaded_rgb(
806: (4)             rgb: npt.NDArray[Any],
807: (4)             point: npt.NDArray[Any],
808: (4)             unit_normal_vect: npt.NDArray[Any],
809: (0)             light_source: npt.NDArray[Any],
810: (4)         ) -> RGBA_Array_Float:
811: (4)             to_sun = normalize(light_source - point)
812: (4)             factor = 0.5 * np.dot(unit_normal_vect, to_sun) ** 3
813: (8)             if factor < 0:
814: (4)                 factor *= 0.5
815: (4)             result = rgb + factor
816: (0)             return result
817: (4)         __all__ = [
818: (4)             "ManimColor",
819: (4)             "ManimColorDType",
820: (4)             "ParsableManimColor",
821: (4)             "color_to_rgb",
822: (4)             "color_to_rgba",
823: (4)             "color_to_int_rgb",
824: (4)             "color_to_int_rgba",
825: (4)             "rgb_to_color",
826: (4)             "rgba_to_color",
827: (4)             "rgb_to_hex",
828: (4)             "hex_to_rgb",
829: (4)             "invert_color",
830: (4)             "interpolate_arrays",
831: (4)             "color_gradient",
832: (4)             "interpolate_color",
833: (4)             "average_color",
834: (4)             "random_bright_color",
835: (4)             "random_color",
836: (0)             "get_shaded_rgb",
]
-----
```

## File 116 - XKCD.py:

```

1: (0)          """Colors from the XKCD Color Name Survey
2: (0)          XKCD is a popular `web comic <https://xkcd.com/353/>`__ created by Randall
Munroe.
3: (0)          His ``Color Name Survey <http://blog.xkcd.com/2010/05/03/color-survey-
4: (0)          200000 participants) resulted in a list of nearly 1000 color names.
5: (0)          While the ``XKCD`` module is exposed to Manim's global name space, the colors
6: (0)          in it are not. This means that in order to use the colors, access them via the
module name:
7: (0)          .. code:: pycon
8: (4)          >>> from manim import XKCD
9: (4)          >>> XKCD.MANGO
10: (4)          ManimColor('#FFA62B')
11: (0)          List of Color Constants
12: (0)
13: (0)          These hex values are non official approximate values intended to simulate the
colors in HTML,
14: (0)          taken from https://www.w3schools.com/colors/colors_xkcd.asp.
15: (0)          ... automanimcolormodule:: manim.utils.color.XKCD
16: (0)
17: (0)          """
18: (0)          from .core import ManimColor
19: (0)          ACIDGREEN = ManimColor("#8FFE09")
20: (0)          ADOBE = ManimColor("#BD6C48")
21: (0)          ALGAE = ManimColor("#54AC68")
22: (0)          ALGAEGREEN = ManimColor("#21C36F")
23: (0)          ALMOSTBLACK = ManimColor("#070D0D")
24: (0)          AMBER = ManimColor("#FEB308")
25: (0)          AMETHYST = ManimColor("#9B5FC0")
26: (0)          APPLE = ManimColor("#6ECB3C")
27: (0)          APPLEGREEN = ManimColor("#76CD26")
28: (0)          APRICOT = ManimColor("#FFB16D")
29: (0)          AQUA = ManimColor("#13EAC9")
30: (0)          AQUABLUE = ManimColor("#02D8E9")
31: (0)          AQUAGREEN = ManimColor("#12E193")
32: (0)          AQUAMARINE = ManimColor("#2EE8BB")
33: (0)          ARMYGREEN = ManimColor("#4B5D16")
34: (0)          ASPARAGUS = ManimColor("#77AB56")
35: (0)          AUBERGINE = ManimColor("#3D0734")
36: (0)          AUBURN = ManimColor("#9A3001")
37: (0)          AVOCADO = ManimColor("#90B134")
38: (0)          AVOCADOGREEN = ManimColor("#87A922")
39: (0)          AZUL = ManimColor("#1D5DEC")
40: (0)          AZURE = ManimColor("#069AF3")
41: (0)          BABYBLUE = ManimColor("#A2CFE")
42: (0)          BABYGREEN = ManimColor("#8cff9e")
43: (0)          BABYPINK = ManimColor("#FFB7CE")
44: (0)          BABYPOO = ManimColor("#AB9004")
45: (0)          BABYPOOP = ManimColor("#937C00")
46: (0)          BABYPOOPGREEN = ManimColor("#8F9805")
47: (0)          BABYPUKEGREEN = ManimColor("#B6C406")
48: (0)          BABYPURPLE = ManimColor("#CA9BF7")
49: (0)          BABYSHITBROWN = ManimColor("#AD900D")
50: (0)          BABYSHITGREEN = ManimColor("#889717")
51: (0)          BANANA = ManimColor("#FFFF7E")
52: (0)          BANANAYELLOW = ManimColor("#FAFE4B")
53: (0)          BARBIEPINK = ManimColor("#FE46A5")
54: (0)          BARFGREEN = ManimColor("#94AC02")
55: (0)          BARNEY = ManimColor("#AC1DB8")
56: (0)          BARNEYPURPLE = ManimColor("#A00498")
57: (0)          BATTLESHIPGREY = ManimColor("#6B7C85")
58: (0)          BEIGE = ManimColor("#E6DAA6")
59: (0)          BERRY = ManimColor("#990F4B")
60: (0)          BILE = ManimColor("#B5C306")
61: (0)          BLACK = ManimColor("#000000")
62: (0)          BLAND = ManimColor("#AFA88B")

```

```

62: (0)           BLOOD = ManimColor("#770001")
63: (0)           BLOODORANGE = ManimColor("#FE4B03")
64: (0)           BLOODRED = ManimColor("#980002")
65: (0)           BLUE = ManimColor("#0343DF")
66: (0)           BLUEBERRY = ManimColor("#464196")
67: (0)           BLUEBLUE = ManimColor("#2242C7")
68: (0)           BLUEGREEN = ManimColor("#0F9B8E")
69: (0)           BLUEGREY = ManimColor("#85A3B2")
70: (0)           BLUEPURPLE = ManimColor("#5A06EF")
71: (0)           BLUEVIOLET = ManimColor("#5D06E9")
72: (0)           BLUEWITHAHINTOFPURPLE = ManimColor("#533CC6")
73: (0)           BLUEYGREEN = ManimColor("#2BB179")
74: (0)           BLUEYGREY = ManimColor("#89A0B0")
75: (0)           BLUEYPURPLE = ManimColor("#6241C7")
76: (0)           BLUISH = ManimColor("#2976BB")
77: (0)           BLUISHGREEN = ManimColor("#10A674")
78: (0)           BLUISHGREY = ManimColor("#748B97")
79: (0)           BLUISHPURPLE = ManimColor("#703BE7")
80: (0)           BLURPLE = ManimColor("#5539CC")
81: (0)           BLUSH = ManimColor("#F29E8E")
82: (0)           BLUSHPINK = ManimColor("#FE828C")
83: (0)           BOOGER = ManimColor("#9BB53C")
84: (0)           BOOGERGREEN = ManimColor("#96B403")
85: (0)           BORDEAUX = ManimColor("#7B002C")
86: (0)           BORINGGREEN = ManimColor("#63B365")
87: (0)           BOTTLEGREEN = ManimColor("#044A05")
88: (0)           BRICK = ManimColor("#A03623")
89: (0)           BRICKORANGE = ManimColor("#C14A09")
90: (0)           BRICKRED = ManimColor("#8F1402")
91: (0)           BRIGHTAQUA = ManimColor("#0BF9EA")
92: (0)           BRIGHTBLUE = ManimColor("#0165FC")
93: (0)           BRIGHTCYAN = ManimColor("#41FDFF")
94: (0)           BRIGHTGREEN = ManimColor("#01FF07")
95: (0)           BRIGHTLAVENDER = ManimColor("#C760FF")
96: (0)           BRIGHTLIGHTBLUE = ManimColor("#26F7FD")
97: (0)           BRIGHTLIGHTGREEN = ManimColor("#2DFE54")
98: (0)           BRIGHTLILAC = ManimColor("#C95EFB")
99: (0)           BRIGHTLIME = ManimColor("#87FD05")
100: (0)          BRIGHTLIMEGREEN = ManimColor("#65FE08")
101: (0)          BRIGHTMAGENTA = ManimColor("#FF08E8")
102: (0)          BRIGHTOLIVE = ManimColor("#9CBB04")
103: (0)          BRIGHTORANGE = ManimColor("#FF5B00")
104: (0)          BRIGHTPINK = ManimColor("#FE01B1")
105: (0)          BRIGHTPURPLE = ManimColor("#BE03FD")
106: (0)          BRIGHTRED = ManimColor("#FF000D")
107: (0)          BRIGHTSEAGREEN = ManimColor("#05FFA6")
108: (0)          BRIGHTSKYBLUE = ManimColor("#02CCFE")
109: (0)          BRIGHTTEAL = ManimColor("#01F9C6")
110: (0)          BRIGHTTURQUOISE = ManimColor("#0FFFEC")
111: (0)          BRIGHTVIOLET = ManimColor("#AD0AFD")
112: (0)          BRIGHTYELLOW = ManimColor("#FFFFD01")
113: (0)          BRIGHTYELLOWGREEN = ManimColor("#9DFF00")
114: (0)          BRITISHRACINGGREEN = ManimColor("#05480D")
115: (0)          BRONZE = ManimColor("#A87900")
116: (0)          BROWN = ManimColor("#653700")
117: (0)          BROWNGREEN = ManimColor("#706C11")
118: (0)          BROWNGREY = ManimColor("#8D8468")
119: (0)          BROWNISH = ManimColor("#9C6D57")
120: (0)          BROWNISHGREEN = ManimColor("#6A6E09")
121: (0)          BROWNISHGREY = ManimColor("#86775F")
122: (0)          BROWNISHORANGE = ManimColor("#CB7723")
123: (0)          BROWNISHPINK = ManimColor("#C27E79")
124: (0)          BROWNISHPURPLE = ManimColor("#76424E")
125: (0)          BROWNISHRED = ManimColor("#9E3623")
126: (0)          BROWNISHYELLOW = ManimColor("#C9B003")
127: (0)          BROWNORANGE = ManimColor("#B96902")
128: (0)          BROWNRED = ManimColor("#922B05")
129: (0)          BROWNYELLOW = ManimColor("#B29705")
130: (0)          BROWNYGREEN = ManimColor("#6F6C0A")

```

```

131: (0) BROWNYORANGE = ManimColor("#CA6B02")
132: (0) BRUISE = ManimColor("#7E4071")
133: (0) BUBBLEGUM = ManimColor("#FF6CB5")
134: (0) BUBBLEGUMPINK = ManimColor("#FF69AF")
135: (0) BUFF = ManimColor("#FEF69E")
136: (0) BURGUNDY = ManimColor("#610023")
137: (0) BURNTORANGE = ManimColor("#C04E01")
138: (0) BURNTRED = ManimColor("#9F2305")
139: (0) BURNTSIENA = ManimColor("#B75203")
140: (0) BURNTSIENNA = ManimColor("#B04E0F")
141: (0) BURNTNUMBER = ManimColor("#A0450E")
142: (0) BURNTYELLOW = ManimColor("#D5AB09")
143: (0) BURPLE = ManimColor("#6832E3")
144: (0) BUTTER = ManimColor("#FFFF81")
145: (0) BUTTERSCOTCH = ManimColor("#FDB147")
146: (0) BUTTERYELLOW = ManimColor("#FFFD74")
147: (0) CADETBLUE = ManimColor("#4E7496")
148: (0) CAMEL = ManimColor("#C69F59")
149: (0) CAMO = ManimColor("#7F8F4E")
150: (0) CAMOGREEN = ManimColor("#526525")
151: (0) CAMOUFLAGEGREEN = ManimColor("#4B6113")
152: (0) CANARY = ManimColor("#FDFF63")
153: (0) CANARYYELLOW = ManimColor("#FFFE40")
154: (0) CANDYPINK = ManimColor("#FF63E9")
155: (0) CARAMEL = ManimColor("#AF6F09")
156: (0) CARMINE = ManimColor("#9D0216")
157: (0) CARNATION = ManimColor("#FD798F")
158: (0) CARNATIONPINK = ManimColor("#FF7FA7")
159: (0) CAROLINABLUE = ManimColor("#8AB8FE")
160: (0) CELADON = ManimColor("#BEFDB7")
161: (0) CELERY = ManimColor("#C1FD95")
162: (0) CEMENT = ManimColor("#A5A391")
163: (0) CERISE = ManimColor("#DE0C62")
164: (0) CERULEAN = ManimColor("#0485D1")
165: (0) CERULEANBLUE = ManimColor("#056EEE")
166: (0) CHARCOAL = ManimColor("#343837")
167: (0) CHARCOALGREY = ManimColor("#3C4142")
168: (0) CHARTREUSE = ManimColor("#C1F80A")
169: (0) CHERRY = ManimColor("#CF0234")
170: (0) CHERRYRED = ManimColor("#F7022A")
171: (0) CHESTNUT = ManimColor("#742802")
172: (0) CHOCOLATE = ManimColor("#3D1C02")
173: (0) CHOCOLATEBROWN = ManimColor("#411900")
174: (0) CINNAMON = ManimColor("#AC4F06")
175: (0) CLARET = ManimColor("#680018")
176: (0) CLAY = ManimColor("#B66A50")
177: (0) CLAYBROWN = ManimColor("#B2713D")
178: (0) CLEARBLUE = ManimColor("#247AFD")
179: (0) COBALT = ManimColor("#1E488F")
180: (0) COBALTBBLUE = ManimColor("#030AA7")
181: (0) COCOA = ManimColor("#875F42")
182: (0) COFFEE = ManimColor("#A6814C")
183: (0) COOLBLUE = ManimColor("#4984B8")
184: (0) COOLGREEN = ManimColor("#33B864")
185: (0) COOLGREY = ManimColor("#95A3A6")
186: (0) COPPER = ManimColor("#B66325")
187: (0) CORAL = ManimColor("#FC5A50")
188: (0) CORALPINK = ManimColor("#FF6163")
189: (0) CORNFLOWER = ManimColor("#6A79F7")
190: (0) CORNFLOWERBLUE = ManimColor("#5170D7")
191: (0) CRANBERRY = ManimColor("#9E003A")
192: (0) CREAM = ManimColor("#FFFFC2")
193: (0) CREME = ManimColor("#FFFFB6")
194: (0) CRIMSON = ManimColor("#8C000F")
195: (0) CUSTARD = ManimColor("#FFFD78")
196: (0) CYAN = ManimColor("#00FFFF")
197: (0) DANDELION = ManimColor("#FEDF08")
198: (0) DARK = ManimColor("#1B2431")
199: (0) DARKAQUA = ManimColor("#05696B")

```

```

200: (0)           DARKAQUAMARINE = ManimColor("#017371")
201: (0)           DARKBEIGE = ManimColor("#AC9362")
202: (0)           DARKBLUE = ManimColor("#030764")
203: (0)           DARKBLUEGREEN = ManimColor("#005249")
204: (0)           DARKBLUEGREY = ManimColor("#1F3B4D")
205: (0)           DARKBROWN = ManimColor("#341C02")
206: (0)           DARKCORAL = ManimColor("#CF524E")
207: (0)           DARKCREAM = ManimColor("#FFF39A")
208: (0)           DARKCYAN = ManimColor("#0A888A")
209: (0)           DARKFORESTGREEN = ManimColor("#002D04")
210: (0)           DARKFUCHSIA = ManimColor("#9D0759")
211: (0)           DARKGOLD = ManimColor("#B59410")
212: (0)           DARKGRASSGREEN = ManimColor("#388004")
213: (0)           DARKGREEN = ManimColor("#054907")
214: (0)           DARKGREENBLUE = ManimColor("#1F6357")
215: (0)           DARKGREY = ManimColor("#363737")
216: (0)           DARKGREYBLUE = ManimColor("#29465B")
217: (0)           DARKHOTPINK = ManimColor("#D90166")
218: (0)           DARKINDIGO = ManimColor("#1F0954")
219: (0)           DARKISHBLUE = ManimColor("#014182")
220: (0)           DARKISHGREEN = ManimColor("#287C37")
221: (0)           DARKISHPINK = ManimColor("#DA467D")
222: (0)           DARKISHPURPLE = ManimColor("#751973")
223: (0)           DARKISHRED = ManimColor("#A90308")
224: (0)           DARKKHAKI = ManimColor("#9B8F55")
225: (0)           DARKLAVENDER = ManimColor("#856798")
226: (0)           DARKLILAC = ManimColor("#9C6DA5")
227: (0)           DARKLIME = ManimColor("#84B701")
228: (0)           DARKLIMEGREEN = ManimColor("#7EBD01")
229: (0)           DARKMAGENTA = ManimColor("#960056")
230: (0)           DARKMAROON = ManimColor("#3C0008")
231: (0)           DARKMAUVE = ManimColor("#874C62")
232: (0)           DARKMINT = ManimColor("#48C072")
233: (0)           DARKMINTGREEN = ManimColor("#20C073")
234: (0)           DARKMUSTARD = ManimColor("#A88905")
235: (0)           DARKNAVY = ManimColor("#000435")
236: (0)           DARKNAVYBLUE = ManimColor("#00022E")
237: (0)           DARKOLIVE = ManimColor("#373E02")
238: (0)           DARKOLIVEGREEN = ManimColor("#3C4D03")
239: (0)           DARKORANGE = ManimColor("#C65102")
240: (0)           DARKPASTELGREEN = ManimColor("#56AE57")
241: (0)           DARKPEACH = ManimColor("#DE7E5D")
242: (0)           DARKPERIWINKLE = ManimColor("#665FD1")
243: (0)           DARKPINK = ManimColor("#CB416B")
244: (0)           DARKPLUM = ManimColor("#3F012C")
245: (0)           DARKPURPLE = ManimColor("#35063E")
246: (0)           DARKRED = ManimColor("#840000")
247: (0)           DARKROSE = ManimColor("#B5485D")
248: (0)           DARKROYALBLUE = ManimColor("#02066F")
249: (0)           DARKSAGE = ManimColor("#598556")
250: (0)           DARKSALMON = ManimColor("#C85A53")
251: (0)           DARKSAND = ManimColor("#A88F59")
252: (0)           DARKSEAFOAM = ManimColor("#1FB57A")
253: (0)           DARKSEAFOAMGREEN = ManimColor("#3EAFF6")
254: (0)           DARKSEAGREEN = ManimColor("#11875D")
255: (0)           DARKSKYBLUE = ManimColor("#448EE4")
256: (0)           DARKSLATEBLUE = ManimColor("#214761")
257: (0)           DARKTAN = ManimColor("#AF884A")
258: (0)           DARKTAUPE = ManimColor("#7F684E")
259: (0)           DARKTEAL = ManimColor("#014D4E")
260: (0)           DARKTURQUOISE = ManimColor("#045C5A")
261: (0)           DARKVIOLET = ManimColor("#34013F")
262: (0)           DARKYELLOW = ManimColor("#D5B60A")
263: (0)           DARKYELLOWGREEN = ManimColor("#728F02")
264: (0)           DEEP AQUA = ManimColor("#08787F")
265: (0)           DEEP BLUE = ManimColor("#040273")
266: (0)           DEEP BROWN = ManimColor("#410200")
267: (0)           DEEP GREEN = ManimColor("#02590F")
268: (0)           DEEP LAVENDER = ManimColor("#8D5EB7")

```

```

269: (0) DEEPLILAC = ManimColor("#966EBD")
270: (0) DEEPMAGENTA = ManimColor("#A0025C")
271: (0) DEEPORANGE = ManimColor("#DC4D01")
272: (0) DEEPPINK = ManimColor("#CB0162")
273: (0) DEEPPURPLE = ManimColor("#36013F")
274: (0) DEEPRED = ManimColor("#9A0200")
275: (0) DEEPROSE = ManimColor("#C74767")
276: (0) DEEPSEABLUE = ManimColor("#015482")
277: (0) DEEPSKYBLUE = ManimColor("#0D75F8")
278: (0) DEEpteal = ManimColor("#00555A")
279: (0) DEEPTURQUOISE = ManimColor("#017374")
280: (0) DEEPVIOLET = ManimColor("#490648")
281: (0) DENIM = ManimColor("#3B638C")
282: (0) DENIMBLUE = ManimColor("#3B5B92")
283: (0) DESERT = ManimColor("#CCAD60")
284: (0) DIARRHEA = ManimColor("#9F8303")
285: (0) DIRT = ManimColor("#8A6E45")
286: (0) DIRTBROWN = ManimColor("#836539")
287: (0) DIRTYBLUE = ManimColor("#3F829D")
288: (0) DIRTYGREEN = ManimColor("#667E2C")
289: (0) DIRTYORANGE = ManimColor("#C87606")
290: (0) DIRTPINK = ManimColor("#CA7B80")
291: (0) DIRTPURPLE = ManimColor("#734A65")
292: (0) DIRTYYELLOW = ManimColor("#CDC50A")
293: (0) DODGERBLUE = ManimColor("#3E82FC")
294: (0) DRAB = ManimColor("#828344")
295: (0) DRABGREEN = ManimColor("#749551")
296: (0) DRIEDBLOOD = ManimColor("#4B0101")
297: (0) DUCKEGGBLUE = ManimColor("#C3FBF4")
298: (0) DULLBLUE = ManimColor("#49759C")
299: (0) DULLBROWN = ManimColor("#876E4B")
300: (0) DULLGREEN = ManimColor("#74A662")
301: (0) DULLORANGE = ManimColor("#D8863B")
302: (0) DULLPINK = ManimColor("#D5869D")
303: (0) DULLPURPLE = ManimColor("#84597E")
304: (0) DULLRED = ManimColor("#BB3F3F")
305: (0) DULLTEAL = ManimColor("#5F9E8F")
306: (0) DULLYELLOW = ManimColor("#EEDC5B")
307: (0) DUSK = ManimColor("#4E5481")
308: (0) DUSKBLUE = ManimColor("#26538D")
309: (0) DUSKYBLUE = ManimColor("#475F94")
310: (0) DUSKPINK = ManimColor("#CC7A8B")
311: (0) DUSKYPURPLE = ManimColor("#895B7B")
312: (0) DUSKYROSE = ManimColor("#BA6873")
313: (0) DUST = ManimColor("#B2996E")
314: (0) DUSTYBLUE = ManimColor("#5A86AD")
315: (0) DUSTYGREEN = ManimColor("#76A973")
316: (0) DUSTYLAVENDER = ManimColor("#AC86A8")
317: (0) DUSTYORANGE = ManimColor("#F0833A")
318: (0) DUSTYPINK = ManimColor("#D58A94")
319: (0) DUSTYPURPLE = ManimColor("#825F87")
320: (0) DUSTYRED = ManimColor("#B9484E")
321: (0) DUSTYROSE = ManimColor("#C0737A")
322: (0) DUSTYTEAL = ManimColor("#4C9085")
323: (0) EARTH = ManimColor("#A2653E")
324: (0) EASTERGREEN = ManimColor("#8CFD7E")
325: (0) EASTERPURPLE = ManimColor("#C071FE")
326: (0) ECRU = ManimColor("#FEFFCA")
327: (0) EGGPLANT = ManimColor("#380835")
328: (0) EGGPLANTPURPLE = ManimColor("#430541")
329: (0) EGGSHELL = ManimColor("#FFFCC4")
330: (0) EGGSHELLBLUE = ManimColor("#C4FFF7")
331: (0) ELECTRICBLUE = ManimColor("#0652FF")
332: (0) ELECTRICGREEN = ManimColor("#21FC0D")
333: (0) ELECTRICCLIME = ManimColor("#A8FF04")
334: (0) ELECTRICPINK = ManimColor("#FF0490")
335: (0) ELECTRICPURPLE = ManimColor("#AA23FF")
336: (0) EMERALD = ManimColor("#01A049")
337: (0) EMERALDGREEN = ManimColor("#028F1E")

```

```

338: (0) EVERGREEN = ManimColor("#05472A")
339: (0) FADEDBLUE = ManimColor("#658CBB")
340: (0) FADEDGREEN = ManimColor("#7BB274")
341: (0) FADEDORANGE = ManimColor("#F0944D")
342: (0) FADEDPINK = ManimColor("#DE9DAC")
343: (0) FADEDPURPLE = ManimColor("#916E99")
344: (0) FADEDRED = ManimColor("#D3494E")
345: (0) FADEDYELLOW = ManimColor("#FEFF7F")
346: (0) FAWN = ManimColor("#CFAF7B")
347: (0) FERN = ManimColor("#63A950")
348: (0) FERNGREEN = ManimColor("#548D44")
349: (0) FIREENGINEERED = ManimColor("#FE0002")
350: (0) FLATBLUE = ManimColor("#3C73A8")
351: (0) FLATGREEN = ManimColor("#699D4C")
352: (0) FLUORESCENTGREEN = ManimColor("#08FF08")
353: (0) FLUROGREEN = ManimColor("#0AFF02")
354: (0) FOAMGREEN = ManimColor("#90FDA9")
355: (0) FOREST = ManimColor("#0B5509")
356: (0) FORESTGREEN = ManimColor("#06470C")
357: (0) FORRESTGREEN = ManimColor("#154406")
358: (0) FRENCHBLUE = ManimColor("#436BAD")
359: (0) FRESHGREEN = ManimColor("#69D84F")
360: (0) FROGGREEN = ManimColor("#58BC08")
361: (0) FUCHSIA = ManimColor("#ED0DD9")
362: (0) GOLD = ManimColor("#DBB40C")
363: (0) GOLDEN = ManimColor("#F5BF03")
364: (0) GOLDENBROWN = ManimColor("#B27A01")
365: (0) GOLDENROD = ManimColor("#F9BC08")
366: (0) GOLDENYELLOW = ManimColor("#FEC615")
367: (0) GRAPE = ManimColor("#6C3461")
368: (0) GRAPEFRUIT = ManimColor("#FD5956")
369: (0) GRAPEPURPLE = ManimColor("#5D1451")
370: (0) GRASS = ManimColor("#5CAC2D")
371: (0) GRASSGREEN = ManimColor("#3F9B0B")
372: (0) GRASSYGREEN = ManimColor("#419C03")
373: (0) GREEN = ManimColor("#15B01A")
374: (0) GREENAPPLE = ManimColor("#5EDC1F")
375: (0) GREENBLUE = ManimColor("#01C08D")
376: (0) GREENBROWN = ManimColor("#544E03")
377: (0) GREENGREY = ManimColor("#77926F")
378: (0) GREENISH = ManimColor("#40A368")
379: (0) GREENISHBEIGE = ManimColor("#C9D179")
380: (0) GREENISHBLUE = ManimColor("#0B8B87")
381: (0) GREENISHBROWN = ManimColor("#696112")
382: (0) GREENISHCYAN = ManimColor("#2AFEB7")
383: (0) GREENISHGREY = ManimColor("#96AE8D")
384: (0) GREENISHTAN = ManimColor("#BCCB7A")
385: (0) GREENISHTEAL = ManimColor("#32BF84")
386: (0) GREENISHTURQUOISE = ManimColor("#00FBBA")
387: (0) GREENISHYELLOW = ManimColor("#CDFD02")
388: (0) GREENTEAL = ManimColor("#0CB577")
389: (0) GREENYBLUE = ManimColor("#42B395")
390: (0) GREENYBROWN = ManimColor("#696006")
391: (0) GREENYELLOW = ManimColor("#B5CE08")
392: (0) GREENYGREY = ManimColor("#7EA07A")
393: (0) GREENYYELLOW = ManimColor("#C6F808")
394: (0) GREY = ManimColor("#929591")
395: (0) GREYBLUE = ManimColor("#647D8E")
396: (0) GREYBROWN = ManimColor("#7F7053")
397: (0) GREYGREEN = ManimColor("#86A17D")
398: (0) GREYISH = ManimColor("#A8A495")
399: (0) GREYISHBLUE = ManimColor("#5E819D")
400: (0) GREYISHBROWN = ManimColor("#7A6A4F")
401: (0) GREYISHGREEN = ManimColor("#82A67D")
402: (0) GREYISHPINK = ManimColor("#C88D94")
403: (0) GREYISHPURPLE = ManimColor("#887191")
404: (0) GREYISHTEAL = ManimColor("#719F91")
405: (0) GREYPINK = ManimColor("#C3909B")
406: (0) GREYPURPLE = ManimColor("#826D8C")

```

```
407: (0)           GREYTEAL = ManimColor("#5E9B8A")
408: (0)           GROSSGREEN = ManimColor("#A0BF16")
409: (0)           GUNMETAL = ManimColor("#536267")
410: (0)           HAZEL = ManimColor("#8E7618")
411: (0)           HEATHER = ManimColor("#A484AC")
412: (0)           HELIOTROPE = ManimColor("#D94FF5")
413: (0)           HIGHLIGHTERGREEN = ManimColor("#1BFC06")
414: (0)           HOSPITALGREEN = ManimColor("#9BE5AA")
415: (0)           HOTGREEN = ManimColor("#25FF29")
416: (0)           HOTMAGENTA = ManimColor("#F504C9")
417: (0)           HOTPINK = ManimColor("#FF028D")
418: (0)           HOTPURPLE = ManimColor("#CB00F5")
419: (0)           HUNTERGREEN = ManimColor("#0B4008")
420: (0)           ICE = ManimColor("#D6FFFA")
421: (0)           ICEBLUE = ManimColor("#D7FFFE")
422: (0)           ICKYGREEN = ManimColor("#8FAE22")
423: (0)           INDIANRED = ManimColor("#850E04")
424: (0)           INDIGO = ManimColor("#380282")
425: (0)           INDIGOBLUE = ManimColor("#3A18B1")
426: (0)           IRIS = ManimColor("#6258C4")
427: (0)           IRISHGREEN = ManimColor("#019529")
428: (0)           IVORY = ManimColor("#FFFFCB")
429: (0)           JADE = ManimColor("#1FA774")
430: (0)           JADEGREEN = ManimColor("#2BAF6A")
431: (0)           JUNGLEGREEN = ManimColor("#048243")
432: (0)           KELLEYGREEN = ManimColor("#009337")
433: (0)           KELLYGREEN = ManimColor("#02AB2E")
434: (0)           KERMITGREEN = ManimColor("#5CB200")
435: (0)           KEYLIME = ManimColor("#AEFF6E")
436: (0)           KHAKI = ManimColor("#AAA662")
437: (0)           KHAKIGREEN = ManimColor("#728639")
438: (0)           KIWI = ManimColor("#9CEF43")
439: (0)           KIWIGREEN = ManimColor("#8EE53F")
440: (0)           LAVENDER = ManimColor("#C79FFF")
441: (0)           LAVENDERBLUE = ManimColor("#8B88F8")
442: (0)           LAVENDERPINK = ManimColor("#DD85D7")
443: (0)           LAWNGREEN = ManimColor("#4DA409")
444: (0)           LEAF = ManimColor("#71AA34")
445: (0)           LEAFGREEN = ManimColor("#5CA904")
446: (0)           LEAFYGREEN = ManimColor("#51B73B")
447: (0)           LEATHER = ManimColor("#AC7434")
448: (0)           LEMON = ManimColor("#FDFF52")
449: (0)           LEMONGREEN = ManimColor("#ADF802")
450: (0)           LEMONLIME = ManimColor("#BFFE28")
451: (0)           LEMONYELLOW = ManimColor("#FDFF38")
452: (0)           LICHEN = ManimColor("#8FB67B")
453: (0)           LIGHTAQUA = ManimColor("#8CFFDB")
454: (0)           LIGHTAQUAMARINE = ManimColor("#7BFDC7")
455: (0)           LIGHTBEIGE = ManimColor("#FFEB6")
456: (0)           LIGHTBLUE = ManimColor("#7BC8F6")
457: (0)           LIGHTBLUEGREEN = ManimColor("#7EFBB3")
458: (0)           LIGHTBLUEGREY = ManimColor("#B7C9E2")
459: (0)           LIGHTBLUISHGREEN = ManimColor("#76FDA8")
460: (0)           LIGHTBRIGHTGREEN = ManimColor("#53FE5C")
461: (0)           LIGHTBROWN = ManimColor("#AD8150")
462: (0)           LIGHTBURGUNDY = ManimColor("#A8415B")
463: (0)           LIGHTCYAN = ManimColor("#ACFFFC")
464: (0)           LIGHTEGGPLANT = ManimColor("#894585")
465: (0)           LIGHTERGREEN = ManimColor("#75FD63")
466: (0)           LIGHTERPURPLE = ManimColor("#A55AF4")
467: (0)           LIGHTFORESTGREEN = ManimColor("#4F9153")
468: (0)           LIGHTGOLD = ManimColor("#FDDC5C")
469: (0)           LIGHTGRASSGREEN = ManimColor("#9AF764")
470: (0)           LIGHTGREEN = ManimColor("#76FF7B")
471: (0)           LIGHTGREENBLUE = ManimColor("#56FCA2")
472: (0)           LIGHTGREENISHBLUE = ManimColor("#63F7B4")
473: (0)           LIGHTGREY = ManimColor("#D8DCD6")
474: (0)           LIGHTGREYBLUE = ManimColor("#9DBCD4")
475: (0)           LIGHTGREYGREEN = ManimColor("#B7E1A1")
```

```

476: (0)           LIGHTINDIGO = ManimColor("#6D5ACF")
477: (0)           LIGHTISHBLUE = ManimColor("#3D7AFD")
478: (0)           LIGHTISHGREEN = ManimColor("#61E160")
479: (0)           LIGHTISHPURPLE = ManimColor("#A552E6")
480: (0)           LIGHTISHRED = ManimColor("#FE2F4A")
481: (0)           LIGHTKHAKI = ManimColor("#E6F2A2")
482: (0)           LIGHTLAVENDAR = ManimColor("#EFC0FE")
483: (0)           LIGHTLAVENDER = ManimColor("#DFC5FE")
484: (0)           LIGHTLIGHTBLUE = ManimColor("#CAFFFB")
485: (0)           LIGHTLIGHTGREEN = ManimColor("#C8FFB0")
486: (0)           LIGHTLILAC = ManimColor("#EDC8FF")
487: (0)           LIGHTLIME = ManimColor("#AEFD6C")
488: (0)           LIGHTLIMEGREEN = ManimColor("#B9FF66")
489: (0)           LIGHTMAGENTA = ManimColor("#FA5FF7")
490: (0)           LIGHTMAROON = ManimColor("#A24857")
491: (0)           LIGHTMAUVE = ManimColor("#C292A1")
492: (0)           LIGHTMINT = ManimColor("#B6FFBB")
493: (0)           LIGHTMINTGREEN = ManimColor("#A6FBB2")
494: (0)           LIGHTMOSSGREEN = ManimColor("#A6C875")
495: (0)           LIGHTMUSTARD = ManimColor("#F7D560")
496: (0)           LIGHTNAVY = ManimColor("#155084")
497: (0)           LIGHTNAVYBLUE = ManimColor("#2E5A88")
498: (0)           LIGHTNEONGREEN = ManimColor("#4EFD54")
499: (0)           LIGHTOLIVE = ManimColor("#ACBF69")
500: (0)           LIGHTOLIVEGREEN = ManimColor("#A4BE5C")
501: (0)           LIGHTORANGE = ManimColor("#FDAA48")
502: (0)           LIGHTPASTELGREEN = ManimColor("#B2FBA5")
503: (0)           LIGHTPEACH = ManimColor("#FFD8B1")
504: (0)           LIGHTPEAGREEN = ManimColor("#C4FE82")
505: (0)           LIGHTPERIWINKLE = ManimColor("#C1C6FC")
506: (0)           LIGHTPINK = ManimColor("#FFD1DF")
507: (0)           LIGHTPLUM = ManimColor("#9D5783")
508: (0)           LIGHTPURPLE = ManimColor("#BF77F6")
509: (0)           LIGHTRED = ManimColor("#FF474C")
510: (0)           LIGHTROSE = ManimColor("#FFC5CB")
511: (0)           LIGHTROYALBLUE = ManimColor("#3A2EFE")
512: (0)           LIGHTSAGE = ManimColor("#BCECAC")
513: (0)           LIGHTSALMON = ManimColor("#FEA993")
514: (0)           LIGHTSEAFoAM = ManimColor("#A0FEBF")
515: (0)           LIGHTSEAFoAMGREEN = ManimColor("#a7ffb5")
516: (0)           LIGHTSEAGREEN = ManimColor("#98F6B0")
517: (0)           LIGHTSKYBLUE = ManimColor("#C6FCFF")
518: (0)           LIGHTTAN = ManimColor("#FBEEAC")
519: (0)           LIGHTTEAL = ManimColor("#90E4C1")
520: (0)           LIGHTTURQUOISE = ManimColor("#7EF4CC")
521: (0)           LIGHTURPLE = ManimColor("#B36FF6")
522: (0)           LIGHTVIOLET = ManimColor("#D6B4FC")
523: (0)           LIGHTYELLOW = ManimColor("#FFE7A")
524: (0)           LIGHTYELLOWGREEN = ManimColor("#CCFD7F")
525: (0)           LIGHTYELLOWISHGREEN = ManimColor("#C2FF89")
526: (0)           LILAC = ManimColor("#CEA2FD")
527: (0)           LILIAC = ManimColor("#C48EFD")
528: (0)           LIME = ManimColor("#AAFF32")
529: (0)           LIMEGREEN = ManimColor("#89FE05")
530: (0)           LIMEYELLOW = ManimColor("#D0FE1D")
531: (0)           LIPSTICK = ManimColor("#D5174E")
532: (0)           LIPSTICKRED = ManimColor("#C0022F")
533: (0)           MACARONIANDCHEESE = ManimColor("#EFB435")
534: (0)           MAGENTA = ManimColor("#C20078")
535: (0)           MAHOGANY = ManimColor("#4A0100")
536: (0)           MAIZE = ManimColor("#F4D054")
537: (0)           MANGO = ManimColor("#FFA62B")
538: (0)           MANILLA = ManimColor("#FFFA86")
539: (0)           MARIGOLD = ManimColor("#FCC006")
540: (0)           MARINE = ManimColor("#042E60")
541: (0)           MARINEBLUE = ManimColor("#01386A")
542: (0)           MAROON = ManimColor("#650021")
543: (0)           MAUVE = ManimColor("#AE7181")
544: (0)           MEDIUMBLUE = ManimColor("#2C6FBB")

```

```

545: (0) MEDIUMBROWN = ManimColor("#7F5112")
546: (0) MEDIUMGREEN = ManimColor("#39AD48")
547: (0) MEDIUMGREY = ManimColor("#7D7F7C")
548: (0) MEDIUMPINK = ManimColor("#F36196")
549: (0) MEDIUMPURPLE = ManimColor("#9E43A2")
550: (0) MELON = ManimColor("#FF7855")
551: (0) MERLOT = ManimColor("#730039")
552: (0) METALLICBLUE = ManimColor("#4F738E")
553: (0) MIDBLUE = ManimColor("#276AB3")
554: (0) MIDGREEN = ManimColor("#50A747")
555: (0) MIDNIGHT = ManimColor("#03012D")
556: (0) MIDNIGHTBLUE = ManimColor("#020035")
557: (0) MIDNIGHTPURPLE = ManimColor("#280137")
558: (0) MILITARYGREEN = ManimColor("#667C3E")
559: (0) MILKCHOCOLATE = ManimColor("#7F4E1E")
560: (0) MINT = ManimColor("#9FFEB0")
561: (0) MINTGREEN = ManimColor("#8FFF9F")
562: (0) MINTYGREEN = ManimColor("#0BF77D")
563: (0) MOCHA = ManimColor("#9D7651")
564: (0) MOSS = ManimColor("#769958")
565: (0) MOSSGREEN = ManimColor("#658B38")
566: (0) MOSSYGREEN = ManimColor("#638B27")
567: (0) MUD = ManimColor("#735C12")
568: (0) MUDBROWN = ManimColor("#60460F")
569: (0) MUDDYBROWN = ManimColor("#886806")
570: (0) MUDDYGREEN = ManimColor("#657432")
571: (0) MUDDYYELLOW = ManimColor("#BFAC05")
572: (0) MUDGREEN = ManimColor("#606602")
573: (0) MULBERRY = ManimColor("#920A4E")
574: (0) MURKYGREEN = ManimColor("#6C7A0E")
575: (0) MUSHROOM = ManimColor("#BA9E88")
576: (0) MUSTARD = ManimColor("#CEB301")
577: (0) MUSTARDBROWN = ManimColor("#AC7E04")
578: (0) MUSTARDGREEN = ManimColor("#A8B504")
579: (0) MUSTARDYELLOW = ManimColor("#D2BD0A")
580: (0) MUTEDBLUE = ManimColor("#3B719F")
581: (0) MUTEDGREEN = ManimColor("#5FA052")
582: (0) MUTEDPINK = ManimColor("#D1768F")
583: (0) MUTEDPURPLE = ManimColor("#805B87")
584: (0) NASTYGREEN = ManimColor("#70B23F")
585: (0) NAVY = ManimColor("#01153E")
586: (0) NAVYBLUE = ManimColor("#001146")
587: (0) NAVYGREEN = ManimColor("#35530A")
588: (0) NEONBLUE = ManimColor("#04D9FF")
589: (0) NEONGREEN = ManimColor("#0cff0c")
590: (0) NEONPINK = ManimColor("#FE019A")
591: (0) NEONPURPLE = ManimColor("#BC13FE")
592: (0) NEONRED = ManimColor("#FF073A")
593: (0) NEONYELLOW = ManimColor("#CFFF04")
594: (0) NICEBLUE = ManimColor("#107AB0")
595: (0) NIGHTBLUE = ManimColor("#040348")
596: (0) OCEAN = ManimColor("#017B92")
597: (0) OCEANBLUE = ManimColor("#03719C")
598: (0) OCEANGREEN = ManimColor("#3D9973")
599: (0) OCHER = ManimColor("#BF9B0C")
600: (0) OCHRE = ManimColor("#BF9005")
601: (0) OCRE = ManimColor("#C69C04")
602: (0) OFFBLUE = ManimColor("#5684AE")
603: (0) OFFGREEN = ManimColor("#6BA353")
604: (0) OFFWHITE = ManimColor("#FFFFE4")
605: (0) OFFYELLOW = ManimColor("#F1F33F")
606: (0) OLDPINK = ManimColor("#C77986")
607: (0) OLDROSE = ManimColor("#C87F89")
608: (0) OLIVE = ManimColor("#6E750E")
609: (0) OLIVEBROWN = ManimColor("#645403")
610: (0) OLIVEDRAB = ManimColor("#6F7632")
611: (0) OLIVEGREEN = ManimColor("#677A04")
612: (0) OLIVEYELLOW = ManimColor("#C2B709")
613: (0) ORANGE = ManimColor("#F97306")

```

```

614: (0) ORANGEBROWN = ManimColor("#BE6400")
615: (0) ORANGEISH = ManimColor("#FD8D49")
616: (0) ORANGEPINK = ManimColor("#FF6F52")
617: (0) ORANGERED = ManimColor("#FE420F")
618: (0) ORANGEYBROWN = ManimColor("#B16002")
619: (0) ORANGEYELLOW = ManimColor("#FFAD01")
620: (0) ORANGEYRED = ManimColor("#FA4224")
621: (0) ORANGEYYELLOW = ManimColor("#FDB915")
622: (0) ORANGISH = ManimColor("#FC824A")
623: (0) ORANGISHBROWN = ManimColor("#B25F03")
624: (0) ORANGISHRED = ManimColor("#F43605")
625: (0) ORCHID = ManimColor("#C875C4")
626: (0) PALE = ManimColor("#FFF9D0")
627: (0) PALEAQUA = ManimColor("#B8FFEB")
628: (0) PALEBLUE = ManimColor("#D0FEFE")
629: (0) PALEBROWN = ManimColor("#B191E6")
630: (0) PALEYAN = ManimColor("#B7FFFA")
631: (0) PALEGOLD = ManimColor("#FDDE6C")
632: (0) PALEGREEN = ManimColor("#C7FDB5")
633: (0) PALEGREY = ManimColor("#FDFDFE")
634: (0) PALELAVENDER = ManimColor("#EECFEE")
635: (0) PALELIGHTGREEN = ManimColor("#B1FC99")
636: (0) PALELILAC = ManimColor("#E4CBFF")
637: (0) PALELIME = ManimColor("#BEFD73")
638: (0) PALELIMEGREEN = ManimColor("#B1FF65")
639: (0) PALEMAGENTA = ManimColor("#D767AD")
640: (0) PALEMAUVE = ManimColor("#FED0FC")
641: (0) PALEOLIVE = ManimColor("#B9CC81")
642: (0) PALEOLIVEGREEN = ManimColor("#B1D27B")
643: (0) PALEORANGE = ManimColor("#FFA756")
644: (0) PALEPEACH = ManimColor("#FFE5AD")
645: (0) PALEPINK = ManimColor("#FFCFDC")
646: (0) PALEPURPLE = ManimColor("#B790D4")
647: (0) PALERED = ManimColor("#D9544D")
648: (0) PALEROSE = ManimColor("#FDC1C5")
649: (0) PALESALMON = ManimColor("#FFB19A")
650: (0) PALESKYBLUE = ManimColor("#BDF6FE")
651: (0) PALETEAL = ManimColor("#82CBB2")
652: (0) PALETURQUOISE = ManimColor("#A5FB05")
653: (0) PALEVIOLET = ManimColor("#CEAEFA")
654: (0) PALEYELLOW = ManimColor("#FFFF84")
655: (0) PARCHMENT = ManimColor("#FEFC00")
656: (0) PASTELBLUE = ManimColor("#A2BFFE")
657: (0) PASTELGREEN = ManimColor("#B0FF9D")
658: (0) PASTELORANGE = ManimColor("#FF964F")
659: (0) PASTELPINK = ManimColor("#FFBACD")
660: (0) PASTELPURPLE = ManimColor("#CAA0FF")
661: (0) PASTELRED = ManimColor("#DB5856")
662: (0) PASTELYELLOW = ManimColor("#FFFE71")
663: (0) PEA = ManimColor("#A4BF20")
664: (0) PEACH = ManimColor("#FFB07C")
665: (0) PEACHYPINK = ManimColor("#FF9A8A")
666: (0) PEACOCKBLUE = ManimColor("#016795")
667: (0) PEAGREEN = ManimColor("#8EAB12")
668: (0) PEAR = ManimColor("#CBF85F")
669: (0) PEASOUP = ManimColor("#929901")
670: (0) PEASOUPGREEN = ManimColor("#94A617")
671: (0) PERIWINKLE = ManimColor("#8E82FE")
672: (0) PERIWINKLEBLUE = ManimColor("#8F99FB")
673: (0) PERRYWINKLE = ManimColor("#8F8CE7")
674: (0) PETROL = ManimColor("#005F6A")
675: (0) PIGPINK = ManimColor("#E78EA5")
676: (0) PINE = ManimColor("#2B5D34")
677: (0) PINEGREEN = ManimColor("#0A481E")
678: (0) PINK = ManimColor("#FF81C0")
679: (0) PINKISH = ManimColor("#D46A7E")
680: (0) PINKISHBROWN = ManimColor("#B17261")
681: (0) PINKISHGREY = ManimColor("#C8ACA9")
682: (0) PINKISHORANGE = ManimColor("#FF724C")

```

```

683: (0) PINKISHPURPLE = ManimColor("#D648D7")
684: (0) PINKISHRED = ManimColor("#F10C45")
685: (0) PINKISHTAN = ManimColor("#D99B82")
686: (0) PINKPURPLE = ManimColor("#EF1DE7")
687: (0) PINKRED = ManimColor("#F5054F")
688: (0) PINKY = ManimColor("#FC86AA")
689: (0) PINKYPURPLE = ManimColor("#C94CBE")
690: (0) PINKYRED = ManimColor("#FC2647")
691: (0) PISSYYELLOW = ManimColor("#DDD618")
692: (0) PISTACHIO = ManimColor("#C0FA8B")
693: (0) PLUM = ManimColor("#580F41")
694: (0) PLUMPURPLE = ManimColor("#4E0550")
695: (0) POISONGREEN = ManimColor("#40FD14")
696: (0) POO = ManimColor("#8F7303")
697: (0) POOBROWN = ManimColor("#885F01")
698: (0) POOP = ManimColor("#7F5E00")
699: (0) POOPBROWN = ManimColor("#7A5901")
700: (0) POOPGREEN = ManimColor("#6F7C00")
701: (0) POWDERBLUE = ManimColor("#B1D1FC")
702: (0) POWDERPINK = ManimColor("#FFB2D0")
703: (0) PRIMARYBLUE = ManimColor("#0804F9")
704: (0) PRUSSIANBLUE = ManimColor("#004577")
705: (0) PUCE = ManimColor("#A57E52")
706: (0) PUKE = ManimColor("#A5A502")
707: (0) PUKEBROWN = ManimColor("#947706")
708: (0) PUKEGREEN = ManimColor("#9AAE07")
709: (0) PUKEYELLOW = ManimColor("#C2BE0E")
710: (0) PUMPKIN = ManimColor("#E17701")
711: (0) PUMPKINORANGE = ManimColor("#FB7D07")
712: (0) PUREBLUE = ManimColor("#0203E2")
713: (0) PURPLE = ManimColor("#7E1E9C")
714: (0) PURPLEBLUE = ManimColor("#5D21D0")
715: (0) PURPLEBROWN = ManimColor("#673A3F")
716: (0) PURPLEGREY = ManimColor("#866F85")
717: (0) PURPLEISH = ManimColor("#98568D")
718: (0) PURPLEISHBLUE = ManimColor("#6140EF")
719: (0) PURPLEISHPINK = ManimColor("#DF4EC8")
720: (0) PURPLEPINK = ManimColor("#D725DE")
721: (0) PURPLERED = ManimColor("#990147")
722: (0) PURPLEY = ManimColor("#8756E4")
723: (0) PURPLEYBLUE = ManimColor("#5F34E7")
724: (0) PURPLEYGREY = ManimColor("#947E94")
725: (0) PURPLEYPINK = ManimColor("#C83CB9")
726: (0) PURPLISH = ManimColor("#94568C")
727: (0) PURPLISHBLUE = ManimColor("#601EF9")
728: (0) PURPLISHBROWN = ManimColor("#6B4247")
729: (0) PURPLISHGREY = ManimColor("#7A687F")
730: (0) PURPLISHPINK = ManimColor("#CE5DAE")
731: (0) PURPLISHRED = ManimColor("#B0054B")
732: (0) PURPLY = ManimColor("#983FB2")
733: (0) PURPLYBLUE = ManimColor("#661AEE")
734: (0) PURLYPINK = ManimColor("#F075E6")
735: (0) PUTTY = ManimColor("#BEAE8A")
736: (0) RACINGGREEN = ManimColor("#014600")
737: (0) RADIOACTIVEGREEN = ManimColor("#2CFA1F")
738: (0) RASPBERRY = ManimColor("#B00149")
739: (0) RAWSIENNA = ManimColor("#9A6200")
740: (0) RAWNUMBER = ManimColor("#A75E09")
741: (0) REALLYLIGHTBLUE = ManimColor("#D4FFFF")
742: (0) RED = ManimColor("#E50000")
743: (0) REDBROWN = ManimColor("#8B2E16")
744: (0) REDDISH = ManimColor("#C44240")
745: (0) REDDISHBROWN = ManimColor("#7F2B0A")
746: (0) REDDISHGREY = ManimColor("#997570")
747: (0) REDDISHORANGE = ManimColor("#F8481C")
748: (0) REDDISHPINK = ManimColor("#FE2C54")
749: (0) REDDISHPURPLE = ManimColor("#910951")
750: (0) REDDYBROWN = ManimColor("#6E1005")
751: (0) REDORANGE = ManimColor("#FD3C06")

```

```

752: (0)           REDPINK = ManimColor("#FA2A55")
753: (0)           REDPURPLE = ManimColor("#820747")
754: (0)           REDVIOLET = ManimColor("#9E0168")
755: (0)           REDWINE = ManimColor("#8C0034")
756: (0)           RICHBLUE = ManimColor("#021BF9")
757: (0)           RICHPURPLE = ManimColor("#720058")
758: (0)           ROBINEGGBLUE = ManimColor("#8AF1FE")
759: (0)           ROBINSEGG = ManimColor("#6DEDFD")
760: (0)           ROBINSEGGBLUE = ManimColor("#98EFF9")
761: (0)           ROSA = ManimColor("#FE86A4")
762: (0)           ROSE = ManimColor("#CF6275")
763: (0)           ROSEPINK = ManimColor("#F7879A")
764: (0)           ROSERED = ManimColor("#BE013C")
765: (0)           ROSYPINK = ManimColor("#F6688E")
766: (0)           ROGUE = ManimColor("#AB1239")
767: (0)           ROYAL = ManimColor("#0C1793")
768: (0)           ROYALBLUE = ManimColor("#0504AA")
769: (0)           ROYALPURPLE = ManimColor("#4B006E")
770: (0)           RUBY = ManimColor("#CA0147")
771: (0)           RUSSET = ManimColor("#A13905")
772: (0)           RUST = ManimColor("#A83C09")
773: (0)           RUSTBROWN = ManimColor("#8B3103")
774: (0)           RUSTORANGE = ManimColor("#C45508")
775: (0)           RUSTRED = ManimColor("#AA2704")
776: (0)           RUSTYORANGE = ManimColor("#CD5909")
777: (0)           RUSTYRED = ManimColor("#AF2F0D")
778: (0)           SAFFRON = ManimColor("#FEB209")
779: (0)           SAGE = ManimColor("#87AE73")
780: (0)           SAGEGREEN = ManimColor("#88B378")
781: (0)           SALMON = ManimColor("#FF796C")
782: (0)           SALMONPINK = ManimColor("#FE7B7C")
783: (0)           SAND = ManimColor("#E2CA76")
784: (0)           SANDBROWN = ManimColor("#CBA560")
785: (0)           SANDSTONE = ManimColor("#C9AE74")
786: (0)           SANDY = ManimColor("#F1DA7A")
787: (0)           SANDYBROWN = ManimColor("#C4A661")
788: (0)           SANDYELLOW = ManimColor("#FCE166")
789: (0)           SANDYYELLOW = ManimColor("#FDEE73")
790: (0)           SAPGREEN = ManimColor("#5C8B15")
791: (0)           SAPPHIRE = ManimColor("#2138AB")
792: (0)           SCARLET = ManimColor("#BE0119")
793: (0)           SEA = ManimColor("#3C9992")
794: (0)           SEABLUE = ManimColor("#047495")
795: (0)           SEAFOAM = ManimColor("#80F9AD")
796: (0)           SEAFOAMBLUE = ManimColor("#78D1B6")
797: (0)           SEAFOAMGREEN = ManimColor("#7AF9AB")
798: (0)           SEAGREEN = ManimColor("#53FCA1")
799: (0)           SEAWEED = ManimColor("#18D17B")
800: (0)           SEAWEEDGREEN = ManimColor("#35AD6B")
801: (0)           SEPIA = ManimColor("#985E2B")
802: (0)           SHAMROCK = ManimColor("#01B44C")
803: (0)           SHAMROCKGREEN = ManimColor("#02C14D")
804: (0)           SHIT = ManimColor("#7F5F00")
805: (0)           SHITBROWN = ManimColor("#7B5804")
806: (0)           SHITGREEN = ManimColor("#758000")
807: (0)           SHOCKINGPINK = ManimColor("#FE02A2")
808: (0)           SICKGREEN = ManimColor("#9DB92C")
809: (0)           SICKLYGREEN = ManimColor("#94B21C")
810: (0)           SICKLYYELLOW = ManimColor("#D0E429")
811: (0)           SIENNA = ManimColor("#A9561E")
812: (0)           SILVER = ManimColor("#C5C9C7")
813: (0)           SKY = ManimColor("#82CAF0")
814: (0)           SKYBLUE = ManimColor("#75BBFD")
815: (0)           SLATE = ManimColor("#516572")
816: (0)           SLATEBLUE = ManimColor("#5B7C99")
817: (0)           SLATEGREEN = ManimColor("#658D6D")
818: (0)           SLATEGREY = ManimColor("#59656D")
819: (0)           SLIMEGREEN = ManimColor("#99CC04")
820: (0)           SNOT = ManimColor("#ACBB0D")

```

```

821: (0)           SNOTGREEN = ManimColor("#9DC100")
822: (0)           SOFTBLUE = ManimColor("#6488EA")
823: (0)           SOFTGREEN = ManimColor("#6FC276")
824: (0)           SOFTPINK = ManimColor("#FDB0C0")
825: (0)           SOFTPURPLE = ManimColor("#A66FB5")
826: (0)           SPEARMINT = ManimColor("#1EF876")
827: (0)           SPRINGGREEN = ManimColor("#A9F971")
828: (0)           SPRUCE = ManimColor("#0A5F38")
829: (0)           SQUASH = ManimColor("#F2AB15")
830: (0)           STEEL = ManimColor("#738595")
831: (0)           STEELBLUE = ManimColor("#5A7D9A")
832: (0)           STEELGREY = ManimColor("#6F828A")
833: (0)           STONE = ManimColor("#ADA587")
834: (0)           STORMYBLUE = ManimColor("#507B9C")
835: (0)           STRAW = ManimColor("#FCF679")
836: (0)           STRAWBERRY = ManimColor("#FB2943")
837: (0)           STRONGBLUE = ManimColor("#0C06F7")
838: (0)           STRONGPINK = ManimColor("#FF0789")
839: (0)           SUNFLOWER = ManimColor("#FFC512")
840: (0)           SUNFLOWERYELLOW = ManimColor("#FFDA03")
841: (0)           SUNNYYELLOW = ManimColor("#FFF917")
842: (0)           SUNSHINEYELLOW = ManimColor("#FFFD37")
843: (0)           SUNYELLOW = ManimColor("#FFDF22")
844: (0)           SWAMP = ManimColor("#698339")
845: (0)           SWAMPGREEN = ManimColor("#748500")
846: (0)           TAN = ManimColor("#D1B26F")
847: (0)           TANBROWN = ManimColor("#AB7E4C")
848: (0)           TANGERINE = ManimColor("#FF9408")
849: (0)           TANGREEN = ManimColor("#A9BE70")
850: (0)           TAUPE = ManimColor("#B9A281")
851: (0)           TEA = ManimColor("#65AB7C")
852: (0)           TEAGREEN = ManimColor("#BDF8A3")
853: (0)           TEAL = ManimColor("#029386")
854: (0)           TEALBLUE = ManimColor("#01889F")
855: (0)           TEALGREEN = ManimColor("#25A36F")
856: (0)           TEALISH = ManimColor("#24BCA8")
857: (0)           TEALISHGREEN = ManimColor("#0CDC73")
858: (0)           TERRACOTA = ManimColor("#CB6843")
859: (0)           TERRACOTTA = ManimColor("#C9643B")
860: (0)           TIFFANYBLUE = ManimColor("#7BF2DA")
861: (0)           TOMATO = ManimColor("#EF4026")
862: (0)           TOMORED = ManimColor("#EC2D01")
863: (0)           TOPAZ = ManimColor("#13BBAF")
864: (0)           TOUPE = ManimColor("#C7AC7D")
865: (0)           TOXICGREEN = ManimColor("#61DE2A")
866: (0)           TREEGREEN = ManimColor("#2A7E19")
867: (0)           TRUEBLUE = ManimColor("#010FCC")
868: (0)           TRUEGREEN = ManimColor("#089404")
869: (0)           TURQUOISE = ManimColor("#06C2AC")
870: (0)           TURQUOISEBLUE = ManimColor("#06B1C4")
871: (0)           TURQUOISEGREEN = ManimColor("#04F489")
872: (0)           TURTLEGREEN = ManimColor("#75B84F")
873: (0)           TWILIGHT = ManimColor("#4E518B")
874: (0)           TWILIGHTBLUE = ManimColor("#0A437A")
875: (0)           UGLYBLUE = ManimColor("#31668A")
876: (0)           UGLYBROWN = ManimColor("#7D7103")
877: (0)           UGLYGREEN = ManimColor("#7A9703")
878: (0)           UGLYPINK = ManimColor("#CD7584")
879: (0)           UGLYPURPLE = ManimColor("#A442A0")
880: (0)           UGLYYELLOW = ManimColor("#D0C101")
881: (0)           ULTRAMARINE = ManimColor("#2000B1")
882: (0)           ULTRAMARINEBLUE = ManimColor("#1805DB")
883: (0)           UMBER = ManimColor("#B26400")
884: (0)           VELVET = ManimColor("#750851")
885: (0)           VERMILION = ManimColor("#F4320C")
886: (0)           VERYDARKBLUE = ManimColor("#000133")
887: (0)           VERYDARKBROWN = ManimColor("#1D0200")
888: (0)           VERYDARKGREEN = ManimColor("#062E03")
889: (0)           VERYDARKPURPLE = ManimColor("#2A0134")

```

```

890: (0)          VERYLIGHTBLUE = ManimColor("#D5FFFF")
891: (0)          VERYLIGHTBROWN = ManimColor("#D3B683")
892: (0)          VERYLIGHTGREEN = ManimColor("#D1FFBD")
893: (0)          VERYLIGHTPINK = ManimColor("#FFF4F2")
894: (0)          VERYLIGHTPURPLE = ManimColor("#F6CEFC")
895: (0)          VERYPALEBLUE = ManimColor("#D6FFE")
896: (0)          VERYPALEGREEN = ManimColor("#CFFDBC")
897: (0)          VIBRANTBLUE = ManimColor("#0339F8")
898: (0)          VIBRANTGREEN = ManimColor("#0ADD08")
899: (0)          VIBRANTPURPLE = ManimColor("#AD03DE")
900: (0)          VIOLET = ManimColor("#9A0EEA")
901: (0)          VIOLETBLUE = ManimColor("#510AC9")
902: (0)          VIOLETPINK = ManimColor("#FB5FFC")
903: (0)          VIOLETRED = ManimColor("#A50055")
904: (0)          VIRIDIAN = ManimColor("#1E9167")
905: (0)          VIVIDBLUE = ManimColor("#152EFF")
906: (0)          VIVIDGREEN = ManimColor("#2FEF10")
907: (0)          VIVIDPURPLE = ManimColor("#9900FA")
908: (0)          VOMIT = ManimColor("#A2A415")
909: (0)          VOMITGREEN = ManimColor("#89A203")
910: (0)          VOMITYELLOW = ManimColor("#C7C10C")
911: (0)          WARMBLUE = ManimColor("#4B57DB")
912: (0)          WARMBROWN = ManimColor("#964E02")
913: (0)          WARMGREY = ManimColor("#978A84")
914: (0)          WARMPINK = ManimColor("#FB5581")
915: (0)          WARMPURPLE = ManimColor("#952E8F")
916: (0)          WASHEDOUTGREEN = ManimColor("#BCF5A6")
917: (0)          WATERBLUE = ManimColor("#0E87CC")
918: (0)          WATERMELON = ManimColor("#FD4659")
919: (0)          WEIRDGREEN = ManimColor("#3AE57F")
920: (0)          WHEAT = ManimColor("#FBDD7E")
921: (0)          WHITE = ManimColor("#FFFFFF")
922: (0)          WINDOWSBLUE = ManimColor("#3778BF")
923: (0)          WINE = ManimColor("#80013F")
924: (0)          WINERED = ManimColor("#7B0323")
925: (0)          WINTERGREEN = ManimColor("#20F986")
926: (0)          WISTERIA = ManimColor("#A87DC2")
927: (0)          YELLOW = ManimColor("#FFFF14")
928: (0)          YELLOWBROWN = ManimColor("#B79400")
929: (0)          YELLOWGREEN = ManimColor("#BBF90F")
930: (0)          YELLOWWISH = ManimColor("#FAEE66")
931: (0)          YELLOWISHBROWN = ManimColor("#9B7A01")
932: (0)          YELLOWISHGREEN = ManimColor("#B0DD16")
933: (0)          YELLOWISHORANGE = ManimColor("#FFAB0F")
934: (0)          YELLOWISHTAN = ManimColor("#FCFC81")
935: (0)          YELLOWOCHRE = ManimColor("#CB9D06")
936: (0)          YELLOWORANGE = ManimColor("#FCB001")
937: (0)          YELLOWTAN = ManimColor("#FFE36E")
938: (0)          YELLOWYBROWN = ManimColor("#AE8B0C")
939: (0)          YELLOWYGREEN = ManimColor("#BFF128")

```

---

File 117 - debug.py:

```

1: (0)          """Debugging utilities."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = ["print_family", "index_labels"]
4: (0)          from manim.mobject.mobject import Mobject
5: (0)          from manim.mobject.text.numbers import Integer
6: (0)          from ..mobject.types.vectorized_mobject import VGroup
7: (0)          from .color import BLACK
8: (0)          def print_family(mobject, n_tabs=0):
9: (4)          """For debugging purposes"""
10: (4)          print("\t" * n_tabs, mobject, id(mobject))
11: (4)          for submob in mobject.submobjects:
12: (8)              print_family(submob, n_tabs + 1)
13: (0)          def index_labels(
14: (4)              mobject: Mobject,

```

```

15: (4)             label_height: float = 0.15,
16: (4)             background_stroke_width=5,
17: (4)             background_stroke_color=BLACK,
18: (4)             **kwargs,
19: (0)
20: (4)         ):
21: (4)             """Returns a :class:`~.VGroup` of :class:`~.Integer` mobjects
22: (4)             that shows the index of each submobject.
23: (4)             Useful for working with parts of complicated mobjects.
24: (4)             Parameters
25: (4)             -----
26: (4)             mobject
27: (4)                 The mobject that will have its submobjects labelled.
28: (4)             label_height
29: (4)                 The height of the labels, by default 0.15.
30: (8)             background_stroke_width
31: (4)                 The stroke width of the outline of the labels, by default 5.
32: (8)             background_stroke_color
33: (4)                 The stroke color of the outline of labels.
34: (8)             kwargs
35: (8)                 Additional parameters to be passed into the :class:`~.Integer` mobjects used to construct the labels.
36: (4)             Examples
37: (4)             -----
38: (4)             .. manim:: IndexLabelsExample
39: (8)                 :save_last_frame:
40: (8)                 class IndexLabelsExample(Scene):
41: (12)                     def construct(self):
42: (16)                         text = MathTex(
43: (20)                             "\frac{d}{dx}f(x)g(x) =",
44: (20)                             "f(x)\frac{d}{dx}g(x)",
45: (20)                             "+",
46: (20)                             "g(x)\frac{d}{dx}f(x)",
47: (16)                         )
48: (16)                         indices = index_labels(text[0])
49: (16)                         text[0][1].set_color(PURPLE_B)
50: (16)                         text[0][8:12].set_color(DARK_BLUE)
51: (16)                         self.add(text, indices)
52: (4)
53: (4)             """
54: (4)             labels = VGroup()
55: (4)             for n, submob in enumerate(mobject):
56: (8)                 label = Integer(n, **kwargs)
57: (8)                 label.set_stroke(
58: (12)                     background_stroke_color, background_stroke_width, background=True
59: (8)                 )
60: (8)                 label.height = label_height
61: (8)                 label.move_to(submob)
62: (8)                 labels.add(label)
63: (4)
64: (4)             return labels

```

---

#### File 118 - BS381.py:

```

1: (0)             """British Color Standard
2: (0)             This module contains colors defined in one of the British Standards
3: (0)             for colors, BS381C. This standard specifies colors used in identification,
4: (0)             coding, and other special purposes. See https://www.britishstandardcolour.com/
5: (0)             for more information.
6: (0)             To use the colors from this list, access them directly from the module (which
7: (0)             is exposed to Manim's global name space):
8: (0)                 .. code:: pycon
9: (4)                     >>> from manim import BS381
10: (4)                     >>> BS381.OXFORD_BLUE
11: (4)                     ManimColor('#1F3057')
12: (0)             List of Color Constants
13: (0)             -----
14: (0)             These hex values (taken from
https://www.w3schools.com/colors/colors\_british.asp)
15: (0)             are non official approximate values intended to simulate the ones defined

```

```

16: (0)          in the standard:
17: (0)          .. automanimcolormodule:: manim.utils.color.BS381
18: (0)
19: (0)          """
20: (0)          from .core import ManimColor
21: (0)          BS381_101 = ManimColor("#94BFAC")
22: (0)          SKY_BLUE = ManimColor("#94BFAC")
23: (0)          BS381_102 = ManimColor("#5B9291")
24: (0)          TURQUOISE_BLUE = ManimColor("#5B9291")
25: (0)          BS381_103 = ManimColor("#3B6879")
26: (0)          PEACOCK_BLUE = ManimColor("#3B6879")
27: (0)          BS381_104 = ManimColor("#264D7E")
28: (0)          AZURE_BLUE = ManimColor("#264D7E")
29: (0)          BS381_105 = ManimColor("#1F3057")
30: (0)          OXFORD_BLUE = ManimColor("#1F3057")
31: (0)          BS381_106 = ManimColor("#2A283D")
32: (0)          ROYAL_BLUE = ManimColor("#2A283D")
33: (0)          BS381_107 = ManimColor("#3A73A9")
34: (0)          STRONG_BLUE = ManimColor("#3A73A9")
35: (0)          BS381_108 = ManimColor("#173679")
36: (0)          AIRCRAFT_BLUE = ManimColor("#173679")
37: (0)          BS381_109 = ManimColor("#1C5680")
38: (0)          MIDDLE_BLUE = ManimColor("#1C5680")
39: (0)          BS381_110 = ManimColor("#2C3E75")
40: (0)          ROUNDDEL_BLUE = ManimColor("#2C3E75")
41: (0)          BS381_111 = ManimColor("#8CC5BB")
42: (0)          PALE_BLUE = ManimColor("#8CC5BB")
43: (0)          BS381_112 = ManimColor("#78ADC2")
44: (0)          ARCTIC_BLUE = ManimColor("#78ADC2")
45: (0)          FIESTA_BLUE = ManimColor("#78ADC2")
46: (0)          BS381_113 = ManimColor("#3F687D")
47: (0)          DEEP_SAXE_BLUE = ManimColor("#3F687D")
48: (0)          BS381_114 = ManimColor("#1F4B61")
49: (0)          RAIL_BLUE = ManimColor("#1F4B61")
50: (0)          BS381_115 = ManimColor("#5F88C1")
51: (0)          COBALT_BLUE = ManimColor("#5F88C1")
52: (0)          BS381_166 = ManimColor("#2458AF")
53: (0)          FRENCH_BLUE = ManimColor("#2458AF")
54: (0)          BS381_169 = ManimColor("#135B75")
55: (0)          TRAFFIC_BLUE = ManimColor("#135B75")
56: (0)          BS381_172 = ManimColor("#A7C6EB")
57: (0)          PALE_ROUNDDEL_BLUE = ManimColor("#A7C6EB")
58: (0)          BS381_174 = ManimColor("#64A0AA")
59: (0)          ORIENT_BLUE = ManimColor("#64A0AA")
60: (0)          BS381_175 = ManimColor("#4F81C5")
61: (0)          LIGHT_FRENCH_BLUE = ManimColor("#4F81C5")
62: (0)          BS381_210 = ManimColor("#BBC9A5")
63: (0)          SKY = ManimColor("#BBC9A5")
64: (0)          BS381_216 = ManimColor("#BCD890")
65: (0)          EAU_DE_NIL = ManimColor("#BCD890")
66: (0)          BS381_217 = ManimColor("#96BF65")
67: (0)          SEA_GREEN = ManimColor("#96BF65")
68: (0)          BS381_218 = ManimColor("#698B47")
69: (0)          GRASS_GREEN = ManimColor("#698B47")
70: (0)          BS381_219 = ManimColor("#757639")
71: (0)          SAGE_GREEN = ManimColor("#757639")
72: (0)          BS381_220 = ManimColor("#4B5729")
73: (0)          OLIVE_GREEN = ManimColor("#4B5729")
74: (0)          BS381_221 = ManimColor("#507D3A")
75: (0)          BRILLIANT_GREEN = ManimColor("#507D3A")
76: (0)          BS381_222 = ManimColor("#6A7031")
77: (0)          LIGHT_BRONZE_GREEN = ManimColor("#6A7031")
78: (0)          BS381_223 = ManimColor("#49523A")
79: (0)          MIDDLE_BRONZE_GREEN = ManimColor("#49523A")
80: (0)          BS381_224 = ManimColor("#3E4630")
81: (0)          DEEP_BRONZE_GREEN = ManimColor("#3E4630")
82: (0)          BS381_225 = ManimColor("#406A28")
83: (0)          LIGHT_BRUNSWICK_GREEN = ManimColor("#406A28")
84: (0)          BS381_226 = ManimColor("#33533B")

```

```

85: (0) BS381_227 = ManimColor("#254432")
86: (0) DEEP_BRUNSWICK_GREEN = ManimColor("#254432")
87: (0) BS381_228 = ManimColor("#428B64")
88: (0) EMERALD_GREEN = ManimColor("#428B64")
89: (0) BS381_241 = ManimColor("#4F5241")
90: (0) DARK_GREEN = ManimColor("#4F5241")
91: (0) BS381_262 = ManimColor("#44945E")
92: (0) BOLD_GREEN = ManimColor("#44945E")
93: (0) BS381_267 = ManimColor("#476A4C")
94: (0) DEEP_CHROME_GREEN = ManimColor("#476A4C")
95: (0) TRAFFIC_GREEN = ManimColor("#476A4C")
96: (0) BS381_275 = ManimColor("#8FC693")
97: (0) OPALINE_GREEN = ManimColor("#8FC693")
98: (0) BS381_276 = ManimColor("#2E4C1E")
99: (0) LINCON_GREEN = ManimColor("#2E4C1E")
100: (0) BS381_277 = ManimColor("#364A20")
101: (0) CYPRESS_GREEN = ManimColor("#364A20")
102: (0) BS381_278 = ManimColor("#87965A")
103: (0) LIGHT OLIVE_GREEN = ManimColor("#87965A")
104: (0) BS381_279 = ManimColor("#3B3629")
105: (0) STEEL_FURNITURE_GREEN = ManimColor("#3B3629")
106: (0) BS381_280 = ManimColor("#68AB77")
107: (0) VERDIGRIS_GREEN = ManimColor("#68AB77")
108: (0) BS381_282 = ManimColor("#506B52")
109: (0) FOREST_GREEN = ManimColor("#506B52")
110: (0) BS381_283 = ManimColor("#7E8F6E")
111: (0) AIRCRAFT_GREY_GREEN = ManimColor("#7E8F6E")
112: (0) BS381_284 = ManimColor("#6B6F5A")
113: (0) SPRUCE_GREEN = ManimColor("#6B6F5A")
114: (0) BS381_285 = ManimColor("#5F5C4B")
115: (0) NATO_GREEN = ManimColor("#5F5C4B")
116: (0) BS381_298 = ManimColor("#4F5138")
117: (0) OLIVE_DRAB = ManimColor("#4F5138")
118: (0) BS381_309 = ManimColor("#FEEC04")
119: (0) CANARY_YELLOW = ManimColor("#FEEC04")
120: (0) BS381_310 = ManimColor("#FEF963")
121: (0) PRIMROSE = ManimColor("#FEF963")
122: (0) BS381_315 = ManimColor("#FEF96A")
123: (0) GRAPEFRUIT = ManimColor("#FEF96A")
124: (0) BS381_320 = ManimColor("#9E7339")
125: (0) LIGHT_BROWN = ManimColor("#9E7339")
126: (0) BS381_337 = ManimColor("#4C4A3C")
127: (0) VERY_DARK_DRAB = ManimColor("#4C4A3C")
128: (0) BS381_350 = ManimColor("#7B6B4F")
129: (0) DARK_EARTH = ManimColor("#7B6B4F")
130: (0) BS381_352 = ManimColor("#FCED96")
131: (0) PALE_CREAM = ManimColor("#FCED96")
132: (0) BS381_353 = ManimColor("#FDF07A")
133: (0) DEEP_CREAM = ManimColor("#FDF07A")
134: (0) BS381_354 = ManimColor("#E9BB43")
135: (0) PRIMROSE_2 = ManimColor("#E9BB43")
136: (0) BS381_355 = ManimColor("#FDD906")
137: (0) LEMON = ManimColor("#FDD906")
138: (0) BS381_356 = ManimColor("#FCC808")
139: (0) GOLDEN_YELLOW = ManimColor("#FCC808")
140: (0) BS381_358 = ManimColor("#F6C870")
141: (0) LIGHT_BUFF = ManimColor("#F6C870")
142: (0) BS381_359 = ManimColor("#DBAC50")
143: (0) MIDDLE_BUFF = ManimColor("#DBAC50")
144: (0) BS381_361 = ManimColor("#D4B97D")
145: (0) LIGHT_STONE = ManimColor("#D4B97D")
146: (0) BS381_362 = ManimColor("#AC7C42")
147: (0) MIDDLE_STONE = ManimColor("#AC7C42")
148: (0) BS381_363 = ManimColor("#FDE706")
149: (0) BOLD_YELLOW = ManimColor("#FDE706")
150: (0) BS381_364 = ManimColor("#CEC093")
151: (0) PORTLAND_STONE = ManimColor("#CEC093")
152: (0) BS381_365 = ManimColor("#F4F0BD")
153: (0) VELLUM = ManimColor("#F4F0BD")

```

```

154: (0) BS381_366 = ManimColor("#F5E7A1")
155: (0) LIGHT_BEIGE = ManimColor("#F5E7A1")
156: (0) BS381_367 = ManimColor("#FEF6BF")
157: (0) MANILLA = ManimColor("#fef6bf")
158: (0) BS381_368 = ManimColor("#DD7B00")
159: (0) TRAFFIC_YELLOW = ManimColor("#DD7B00")
160: (0) BS381_369 = ManimColor("#FEEBA8")
161: (0) BISCUIT = ManimColor("#feeba8")
162: (0) BS381_380 = ManimColor("#BBA38A")
163: (0) CAMOUFLAGE_DESERT_SAND = ManimColor("#BBA38A")
164: (0) BS381_384 = ManimColor("#EEDFA5")
165: (0) LIGHT_STRAW = ManimColor("#EEDFA5")
166: (0) BS381_385 = ManimColor("#E8C88F")
167: (0) LIGHT_BISCUIT = ManimColor("#E8C88F")
168: (0) BS381_386 = ManimColor("#E6C18D")
169: (0) CHAMPAGNE = ManimColor("#e6c18d")
170: (0) BS381_387 = ManimColor("#CFB48A")
171: (0) SUNRISE = ManimColor("#cfb48a")
172: (0) SUNSHINE = ManimColor("#cfb48a")
173: (0) BS381_388 = ManimColor("#E4CF93")
174: (0) BEIGE = ManimColor("#e4cf93")
175: (0) BS381_389 = ManimColor("#B2A788")
176: (0) CAMOUFLAGE_BEIGE = ManimColor("#B2A788")
177: (0) BS381_397 = ManimColor("#F3D163")
178: (0) JASMINE_YELLOW = ManimColor("#F3D163")
179: (0) BS381_411 = ManimColor("#74542F")
180: (0) MIDDLE_BROWN = ManimColor("#74542F")
181: (0) BS381_412 = ManimColor("#5C422E")
182: (0) DARK_BROWN = ManimColor("#5C422E")
183: (0) BS381_413 = ManimColor("#402D21")
184: (0) NUT_BROWN = ManimColor("#402D21")
185: (0) BS381_414 = ManimColor("#A86C29")
186: (0) GOLDEN_BROWN = ManimColor("#A86C29")
187: (0) BS381_415 = ManimColor("#61361E")
188: (0) IMPERIAL_BROWN = ManimColor("#61361E")
189: (0) BS381_420 = ManimColor("#A89177")
190: (0) DARK_CAMOUFLAGE_DESERT_SAND = ManimColor("#A89177")
191: (0) BS381_435 = ManimColor("#845B4D")
192: (0) CAMOUFLAGE_RED = ManimColor("#845B4D")
193: (0) BS381_436 = ManimColor("#564B47")
194: (0) DARK_CAMOUFLAGE_BROWN = ManimColor("#564B47")
195: (0) BS381_439 = ManimColor("#753B1E")
196: (0) ORANGE_BROWN = ManimColor("#753B1E")
197: (0) BS381_443 = ManimColor("#C98A71")
198: (0) SALMON = ManimColor("#c98a71")
199: (0) BS381_444 = ManimColor("#A65341")
200: (0) TERRACOTTA = ManimColor("#a65341")
201: (0) BS381_445 = ManimColor("#83422B")
202: (0) VENETIAN_RED = ManimColor("#83422B")
203: (0) BS381_446 = ManimColor("#774430")
204: (0) RED_OXIDE = ManimColor("#774430")
205: (0) BS381_447 = ManimColor("#F3B28B")
206: (0) SALMON_PINK = ManimColor("#F3B28B")
207: (0) BS381_448 = ManimColor("#67403A")
208: (0) DEEP_INDIAN_RED = ManimColor("#67403A")
209: (0) BS381_449 = ManimColor("#693B3F")
210: (0) LIGHT_PURPLE_BROWN = ManimColor("#693B3F")
211: (0) BS381_452 = ManimColor("#613339")
212: (0) DARK_CRIMSON = ManimColor("#613339")
213: (0) BS381_453 = ManimColor("#FBDED6")
214: (0) SHELL_PINK = ManimColor("#FBDED6")
215: (0) BS381_454 = ManimColor("#E8A1A2")
216: (0) PALE_ROUNDEL_RED = ManimColor("#E8A1A2")
217: (0) BS381_460 = ManimColor("#BD8F56")
218: (0) DEEP_BUFF = ManimColor("#BD8F56")
219: (0) BS381_473 = ManimColor("#793932")
220: (0) GULF_RED = ManimColor("#793932")
221: (0) BS381_489 = ManimColor("#8D5B41")
222: (0) LEAF_BROWN = ManimColor("#8D5B41")

```

```

223: (0) BS381_490 = ManimColor("#573320")
224: (0) BEECH_BROWN = ManimColor("#573320")
225: (0) BS381_499 = ManimColor("#59493E")
226: (0) SERVICE_BROWN = ManimColor("#59493E")
227: (0) BS381_536 = ManimColor("#BB3016")
228: (0) POPPY = ManimColor("#bb3016")
229: (0) BS381_537 = ManimColor("#DD3420")
230: (0) SIGNAL_RED = ManimColor("#DD3420")
231: (0) BS381_538 = ManimColor("#C41C22")
232: (0) POST_OFFICE_RED = ManimColor("#C41C22")
233: (0) CHERRY = ManimColor("#c41c22")
234: (0) BS381_539 = ManimColor("#D21E2B")
235: (0) CurrANT_RED = ManimColor("#D21E2B")
236: (0) BS381_540 = ManimColor("#8B1A32")
237: (0) CRIMSON = ManimColor("#8b1a32")
238: (0) BS381_541 = ManimColor("#471B21")
239: (0) MAROON = ManimColor("#471b21")
240: (0) BS381_542 = ManimColor("#982D57")
241: (0) RUBY = ManimColor("#982d57")
242: (0) BS381_557 = ManimColor("#EF841E")
243: (0) LIGHT_ORANGE = ManimColor("#EF841E")
244: (0) BS381_564 = ManimColor("#DD3524")
245: (0) BOLD_RED = ManimColor("#DD3524")
246: (0) BS381_568 = ManimColor("#FB9C06")
247: (0) APRICOT = ManimColor("#fb9c06")
248: (0) BS381_570 = ManimColor("#A83C19")
249: (0) TRAFFIC_RED = ManimColor("#A83C19")
250: (0) BS381_591 = ManimColor("#D04E09")
251: (0) DEEP_ORANGE = ManimColor("#D04E09")
252: (0) BS381_592 = ManimColor("#E45523")
253: (0) INTERNATIONAL_ORANGE = ManimColor("#E45523")
254: (0) BS381_593 = ManimColor("#F24816")
255: (0) RAIL_RED = ManimColor("#F24816")
256: (0) AZO_ORANGE = ManimColor("#F24816")
257: (0) BS381_626 = ManimColor("#A0A9AA")
258: (0) CAMOUFLAGE_GREY = ManimColor("#A0A9AA")
259: (0) BS381_627 = ManimColor("#BEC0B8")
260: (0) LIGHT_AIRCRAFT_GREY = ManimColor("#BEC0B8")
261: (0) BS381_628 = ManimColor("#9D9D7E")
262: (0) SILVER_GREY = ManimColor("#9D9D7E")
263: (0) BS381_629 = ManimColor("#7A838B")
264: (0) DARK_CAMOUFLAGE_GREY = ManimColor("#7A838B")
265: (0) BS381_630 = ManimColor("#A5AD98")
266: (0) FRENCH_GREY = ManimColor("#A5AD98")
267: (0) BS381_631 = ManimColor("#9AAA9F")
268: (0) LIGHT_GREY = ManimColor("#9AAA9F")
269: (0) BS381_632 = ManimColor("#6B7477")
270: (0) DARK ADMIRALTY_GREY = ManimColor("#6B7477")
271: (0) BS381_633 = ManimColor("#424C53")
272: (0) RAF_BLUE_GREY = ManimColor("#424C53")
273: (0) BS381_634 = ManimColor("#6F7264")
274: (0) SLATE = ManimColor("#6f7264")
275: (0) BS381_635 = ManimColor("#525B55")
276: (0) LEAD = ManimColor("#525b55")
277: (0) BS381_636 = ManimColor("#5F7682")
278: (0) PRU_BLUE = ManimColor("#5F7682")
279: (0) BS381_637 = ManimColor("#8E9B9C")
280: (0) MEDIUM_SEA_GREY = ManimColor("#8E9B9C")
281: (0) BS381_638 = ManimColor("#6C7377")
282: (0) DARK_SEA_GREY = ManimColor("#6C7377")
283: (0) BS381_639 = ManimColor("#667563")
284: (0) LIGHT_SLATE_GREY = ManimColor("#667563")
285: (0) BS381_640 = ManimColor("#566164")
286: (0) EXTRA_DARK_SEA_GREY = ManimColor("#566164")
287: (0) BS381_642 = ManimColor("#282B2F")
288: (0) NIGHT = ManimColor("#282b2f")
289: (0) BS381_671 = ManimColor("#4E5355")
290: (0) MIDDLE_GRAPHITE = ManimColor("#4E5355")
291: (0) BS381_676 = ManimColor("#A9B7B9")

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
292: (0)          LIGHT_WEATHERWORK_GREY = ManimColor("#A9B7B9")
293: (0)          BS381_677 = ManimColor("#676F76")
294: (0)          DARK_WEATHERWORK_GREY = ManimColor("#676F76")
295: (0)          BS381_692 = ManimColor("#7B93A3")
296: (0)          SMOKE_GREY = ManimColor("#7B93A3")
297: (0)          BS381_693 = ManimColor("#88918D")
298: (0)          AIRCRAFT_GREY = ManimColor("#88918D")
299: (0)          BS381_694 = ManimColor("#909A92")
300: (0)          DOVE_GREY = ManimColor("#909A92")
301: (0)          BS381_697 = ManimColor("#B6D3CC")
302: (0)          LIGHT ADMIRALTY_GREY = ManimColor("#B6D3CC")
303: (0)          BS381_796 = ManimColor("#6E4A75")
304: (0)          DARK_VIOLET = ManimColor("#6E4A75")
305: (0)          BS381_797 = ManimColor("#C9A8CE")
306: (0)          LIGHT_VIOLET = ManimColor("#C9A8CE")
```

---

File 119 - typing.py:

```
1: (0)      """Custom type definitions used in Manim.
2: (0)      .. admonition:: Note for developers
3: (4)          :class: important
4: (4)          Around the source code there are multiple strings which look like this:
5: (4)          .. code-block::
6: (8)              ...
7: (8)              [CATEGORY]
8: (8)              <category_name>
9: (8)              ''
10: (4)         All type aliases defined under those strings will be automatically
11: (4)         classified under that category.
12: (4)         If you need to define a new category, respect the format described above.
13: (0)
14: (0)     from __future__ import annotations
15: (0)     from os import PathLike
16: (0)     from typing import Callable, Union
17: (0)     import numpy as np
18: (0)     import numpy.typing as npt
19: (0)     from typing_extensions import TypeAlias
20: (0)     __all__ = [
21: (4)         "ManimFloat",
22: (4)         "ManimInt",
23: (4)         "ManimColorDType",
24: (4)         "RGB_Array_Float",
25: (4)         "RGB_Tuple_Float",
26: (4)         "RGB_Array_Int",
27: (4)         "RGB_Tuple_Int",
28: (4)         "RGBA_Array_Float",
29: (4)         "RGBA_Tuple_Float",
30: (4)         "RGBA_Array_Int",
31: (4)         "RGBA_Tuple_Int",
32: (4)         "HSV_Array_Float",
33: (4)         "HSV_Tuple_Float",
34: (4)         "ManimColorInternal",
35: (4)         "PointDType",
36: (4)         "InternalPoint2D",
37: (4)         "Point2D",
38: (4)         "InternalPoint2D_Array",
39: (4)         "Point2D_Array",
40: (4)         "InternalPoint3D",
41: (4)         "Point3D",
42: (4)         "InternalPoint3D_Array",
43: (4)         "Point3D_Array",
44: (4)         "Vector2D",
45: (4)         "Vector2D_Array",
46: (4)         "Vector3D",
47: (4)         "Vector3D_Array",
48: (4)         "VectorND",
49: (4)         "VectorND_Array",
```

```

50: (4)           "RowVector",
51: (4)           "ColVector",
52: (4)           "MatrixMN",
53: (4)           "Zeros",
54: (4)           "QuadraticBezierPoints",
55: (4)           "QuadraticBezierPoints_Array",
56: (4)           "QuadraticBezierPath",
57: (4)           "QuadraticSpline",
58: (4)           "CubicBezierPoints",
59: (4)           "CubicBezierPoints_Array",
60: (4)           "CubicBezierPath",
61: (4)           "CubicSpline",
62: (4)           "BezierPoints",
63: (4)           "BezierPoints_Array",
64: (4)           "BezierPath",
65: (4)           "Spline",
66: (4)           "FlatBezierPoints",
67: (4)           "FunctionOverride",
68: (4)           "PathFuncType",
69: (4)           "MappingFunction",
70: (4)           "Image",
71: (4)           "GrayscaleImage",
72: (4)           "RGBImage",
73: (4)           "RGBAImage",
74: (4)           "StrPath",
75: (4)           "StrOrBytesPath",
76: (0)
77: (0)
78: (0)           [CATEGORY]
79: (0)           Primitive data types
80: (0)
81: (0)           ManimFloat: TypeAlias = np.float64
82: (0)           """A double-precision floating-point value (64 bits, or 8 bytes),
83: (0)           according to the IEEE 754 standard.
84: (0)
85: (0)           ManimInt: TypeAlias = np.int64
86: (0)           r"""A long integer (64 bits, or 8 bytes).
87: (0)           It can take values between :math:`-2^{63}` and :math:`+2^{63} - 1` ,
88: (0)           which expressed in base 10 is a range between around
89: (0)           :math:`-9.223 \cdot 10^{18}` and :math:`+9.223 \cdot 10^{18}` .
90: (0)
91: (0)
92: (0)           [CATEGORY]
93: (0)           Color types
94: (0)
95: (0)           ManimColorDType: TypeAlias = ManimFloat
96: (0)           """Data type used in :class:`~.ManimColorInternal`: a
97: (0)           double-precision float between 0 and 1.
98: (0)
99: (0)           RGB_Array_Float: TypeAlias = npt.NDArray[ManimColorDType]
100: (0)          """`shape: (3,)``
101: (0)          A :class:`numpy.ndarray` of 3 floats between 0 and 1, representing a
102: (0)          color in RGB format.
103: (0)          Its components describe, in order, the intensity of Red, Green, and
104: (0)          Blue in the represented color.
105: (0)
106: (0)           RGB_Tuple_Float: TypeAlias = tuple[float, float, float]
107: (0)          """`shape: (3,)```
108: (0)          A tuple of 3 floats between 0 and 1, representing a color in RGB
109: (0)          format.
110: (0)          Its components describe, in order, the intensity of Red, Green, and
111: (0)          Blue in the represented color.
112: (0)
113: (0)           RGB_Array_Int: TypeAlias = npt.NDArray[ManimInt]
114: (0)          """`shape: (3,)```
115: (0)          A :class:`numpy.ndarray` of 3 integers between 0 and 255,
116: (0)          representing a color in RGB format.
117: (0)          Its components describe, in order, the intensity of Red, Green, and
118: (0)          Blue in the represented color.

```

```

119: (0)
120: (0)
121: (0)
122: (0)
123: (0)
124: (0)
125: (0)
126: (0)
127: (0)
128: (0)
129: (0)
130: (0)
131: (0)
132: (0)
133: (0)
134: (0)
135: (0)
136: (0)
137: (0)
138: (0)
139: (0)
140: (0)
141: (0)
142: (0)
143: (0)
144: (0)
145: (0)
146: (0)
147: (0)
148: (0)
149: (0)
150: (0)
151: (0)
152: (0)
153: (0)
154: (0)
155: (0)
156: (0)
157: (0)
158: (0)
159: (0)
160: (0)
161: (0)
162: (0)
163: (0)
164: (0)
165: (0)
166: (0)
167: (0)
168: (0)
169: (0)
170: (0)
171: (0)
172: (0)
173: (0)
174: (0)
175: (0)
176: (0)
177: (0)
178: (0)
179: (0)
180: (0)
181: (0)
182: (0)
183: (0)
184: (0)
185: (0)
186: (0)
187: (0)

        """
        RGB_Tuple_Int: TypeAlias = tuple[int, int, int]
        """``shape: (3,)``
        A tuple of 3 integers between 0 and 255, representing a color in RGB format.
        Its components describe, in order, the intensity of Red, Green, and Blue in the represented color.
        """
        """
        RGBA_Array_Float: TypeAlias = npt.NDArray[ManimColorDType]
        """``shape: (4,)``
        A :class:`numpy.ndarray` of 4 floats between 0 and 1, representing a color in RGBA format.
        Its components describe, in order, the intensity of Red, Green, Blue and Alpha (opacity) in the represented color.
        """
        """
        RGBA_Tuple_Float: TypeAlias = tuple[float, float, float, float]
        """``shape: (4,)``
        A tuple of 4 floats between 0 and 1, representing a color in RGBA format.
        Its components describe, in order, the intensity of Red, Green, Blue and Alpha (opacity) in the represented color.
        """
        """
        RGBA_Array_Int: TypeAlias = npt.NDArray[ManimInt]
        """``shape: (4,)``
        A :class:`numpy.ndarray` of 4 integers between 0 and 255, representing a color in RGBA format.
        Its components describe, in order, the intensity of Red, Green, Blue and Alpha (opacity) in the represented color.
        """
        """
        RGBA_Tuple_Int: TypeAlias = tuple[int, int, int, int]
        """``shape: (4,)``
        A tuple of 4 integers between 0 and 255, representing a color in RGBA format.
        Its components describe, in order, the intensity of Red, Green, Blue and Alpha (opacity) in the represented color.
        """
        """
        HSV_Array_Float: TypeAlias = RGB_Array_Float
        """``shape: (3,)``
        A :class:`numpy.ndarray` of 3 floats between 0 and 1, representing a color in HSV (or HSB) format.
        Its components describe, in order, the Hue, Saturation and Value (or Brightness) in the represented color.
        """
        """
        HSV_Tuple_Float: TypeAlias = RGB_Tuple_Float
        """``shape: (3,)``
        A tuple of 3 floats between 0 and 1, representing a color in HSV (or HSB) format.
        Its components describe, in order, the Hue, Saturation and Value (or Brightness) in the represented color.
        """
        """
        ManimColorInternal: TypeAlias = RGBA_Array_Float
        """``shape: (4,)``
        Internal color representation used by :class:`~.ManimColor`, following the RGBA format.
        It is a :class:`numpy.ndarray` consisting of 4 floats between 0 and 1, describing respectively the intensities of Red, Green, Blue and Alpha (opacity) in the represented color.
        """
        """
        [CATEGORY]
        Point types
        """
        """
        PointDType: TypeAlias = ManimFloat
        """"Default type for arrays representing points: a double-precision floating point value.
        """
        """
        InternalPoint2D: TypeAlias = npt.NDArray[PointDType]
        """``shape: (2,)``
        A 2-dimensional point: ``[float, float]``.

```

```

188: (0)          .. note::
189: (4)            This type alias is mostly made available for internal use, and
190: (4)            only includes the NumPy type.
191: (0)
192: (0)          Point2D: TypeAlias = Union[InternalPoint2D, tuple[float, float]]
193: (0)          """``shape: (2,)``
194: (0)            A 2-dimensional point: ``[float, float]``.
195: (0)            Normally, a function or method which expects a `Point2D` as a
196: (0)            parameter can handle being passed a `Point3D` instead.
197: (0)
198: (0)          InternalPoint2D_Array: TypeAlias = npt.NDArray[PointDType]
199: (0)          """``shape: (N, 2)``
200: (0)            An array of `InternalPoint2D` objects: ``[[float, float], ...]``.
201: (0)          .. note::
202: (4)            This type alias is mostly made available for internal use, and
203: (4)            only includes the NumPy type.
204: (0)
205: (0)          Point2D_Array: TypeAlias = Union[InternalPoint2D_Array, tuple[Point2D, ...]]
206: (0)          """``shape: (N, 2)``
207: (0)            An array of `Point2D` objects: ``[[float, float], ...]``.
208: (0)            Normally, a function or method which expects a `Point2D_Array` as a
209: (0)            parameter can handle being passed a `Point3D_Array` instead.
210: (0)            Please refer to the documentation of the function you are using for
211: (0)            further type information.
212: (0)
213: (0)          InternalPoint3D: TypeAlias = npt.NDArray[PointDType]
214: (0)          """``shape: (3,)``
215: (0)            A 3-dimensional point: ``[float, float, float]``.
216: (0)          .. note::
217: (4)            This type alias is mostly made available for internal use, and
218: (4)            only includes the NumPy type.
219: (0)
220: (0)          Point3D: TypeAlias = Union[InternalPoint3D, tuple[float, float, float]]
221: (0)          """``shape: (3,)``
222: (0)            A 3-dimensional point: ``[float, float, float]``.
223: (0)
224: (0)          InternalPoint3D_Array: TypeAlias = npt.NDArray[PointDType]
225: (0)          """``shape: (N, 3)``
226: (0)            An array of `Point3D` objects: ``[[float, float, float], ...]``.
227: (0)          .. note::
228: (4)            This type alias is mostly made available for internal use, and
229: (4)            only includes the NumPy type.
230: (0)
231: (0)          Point3D_Array: TypeAlias = Union[InternalPoint3D_Array, tuple[Point3D, ...]]
232: (0)          """``shape: (N, 3)``
233: (0)            An array of `Point3D` objects: ``[[float, float, float], ...]``.
234: (0)            Please refer to the documentation of the function you are using for
235: (0)            further type information.
236: (0)
237: (0)
238: (0)          [CATEGORY]
239: (0)          Vector types
240: (0)
241: (0)          Vector2D: TypeAlias = npt.NDArray[PointDType]
242: (0)          """``shape: (2,)``
243: (0)            A 2-dimensional vector: ``[float, float]``.
244: (0)            Normally, a function or method which expects a `Vector2D` as a
245: (0)            parameter can handle being passed a `Vector3D` instead.
246: (0)          .. caution::
247: (4)            Do not confuse with the :class:`~.Vector` or :class:`~.Arrow`  

248: (4)            VMobjects!
249: (0)
250: (0)          Vector2D_Array: TypeAlias = npt.NDArray[PointDType]
251: (0)          """``shape: (M, 2)``
252: (0)            An array of `Vector2D` objects: ``[[float, float], ...]``.
253: (0)            Normally, a function or method which expects a `Vector2D_Array` as a
254: (0)            parameter can handle being passed a `Vector3D_Array` instead.
255: (0)
256: (0)          Vector3D: TypeAlias = npt.NDArray[PointDType]

```

```

257: (0)         """``shape: (3,)``
258: (0)         A 3-dimensional vector: ``[float, float, float]``.
259: (0)         .. caution::
260: (4)             Do not confuse with the :class:`~.Vector` or :class:`~.Arrow3D`  

261: (4)             VMobjects!
262: (0)
263: (0)         Vector3D_Array: TypeAlias = npt.NDArray[PointDType]
264: (0)         """``shape: (M, 3)``
265: (0)         An array of `Vector3D` objects: ``[[float, float, float], ...]``.
266: (0)
267: (0)         VectorND: TypeAlias = npt.NDArray[PointDType]
268: (0)         """``shape (N,)``
269: (0)         An :math:`N`-dimensional vector: ``[float, ...]``.
270: (0)         .. caution::
271: (4)             Do not confuse with the :class:`~.Vector` VMobject! This type alias  

272: (4)             is named "VectorND" instead of "Vector" to avoid potential name  

273: (4)             collisions.
274: (0)
275: (0)         VectorND_Array: TypeAlias = npt.NDArray[PointDType]
276: (0)         """``shape (M, N)``
277: (0)         An array of `VectorND` objects: ``[[[float, ...], ...], ...]``.
278: (0)
279: (0)         RowVector: TypeAlias = npt.NDArray[PointDType]
280: (0)         """``shape: (1, N)``
281: (0)         A row vector: ``[[float, ...]]``.
282: (0)
283: (0)         ColVector: TypeAlias = npt.NDArray[PointDType]
284: (0)         """``shape: (N, 1)``
285: (0)         A column vector: ``[[float], [float], ...]``.
286: (0)
287: (0)
288: (0)         [CATEGORY]
289: (0)         Matrix types
290: (0)
291: (0)         MatrixMN: TypeAlias = npt.NDArray[PointDType]
292: (0)         """``shape: (M, N)``
293: (0)         A matrix: ``[[float, ...], [float, ...], ...]``.
294: (0)
295: (0)         Zeros: TypeAlias = MatrixMN
296: (0)         """``shape: (M, N)``
297: (0)         A `MatrixMN` filled with zeros, typically created with  

298: (0)         ``numpy.zeros((M, N))``.
299: (0)
300: (0)
301: (0)         [CATEGORY]
302: (0)         Bézier types
303: (0)
304: (0)         QuadraticBezierPoints: TypeAlias = Union[
305: (4)             npt.NDArray[PointDType], tuple[Point3D, Point3D, Point3D]
306: (0)         ]
307: (0)         """``shape: (3, 3)``
308: (0)         A `Point3D_Array` of 3 control points for a single quadratic Bézier  

309: (0)         curve:
310: (0)         ``[[float, float, float], [float, float, float], [float, float, float]]``.
311: (0)
312: (0)         QuadraticBezierPoints_Array: TypeAlias = Union[
313: (4)             npt.NDArray[PointDType], tuple[QuadraticBezierPoints, ...]
314: (0)         ]
315: (0)         """``shape: (N, 3, 3)``
316: (0)         An array of :math:`N` `QuadraticBezierPoints` objects:  

317: (0)         ``[[[float, float, float], [float, float, float], [float, float, float]],  

...].``.
318: (0)
319: (0)         QuadraticBezierPath: TypeAlias = Point3D_Array
320: (0)         """``shape: (3*N, 3)``
321: (0)         A `Point3D_Array` of :math:`3N` points, where each one of the  

322: (0)         :math:`N` consecutive blocks of 3 points represents a quadratic  

323: (0)         Bézier curve:
324: (0)         ``[[float, float, float], ...], ...]``.

```

```

325: (0)             Please refer to the documentation of the function you are using for
326: (0)             further type information.
327: (0)
328: (0)             QuadraticSpline: TypeAlias = QuadraticBezierPath
329: (0)             """``shape: (3*N, 3)``
330: (0)             A special case of `QuadraticBezierPath` where all the :math:`N` quadratic Bézier curves are connected, forming a quadratic spline:
331: (0)             ``[[float, float, float], ..., ...]``.
332: (0)
333: (0)             Please refer to the documentation of the function you are using for
334: (0)             further type information.
335: (0)
336: (0)             CubicBezierPoints: TypeAlias = Union[
337: (4)                 npt.NDArray[PointDType], tuple[Point3D, Point3D, Point3D, Point3D]
338: (0)             ]
339: (0)             """``shape: (4, 3)``
340: (0)             A `Point3D_Array` of 4 control points for a single cubic Bézier
341: (0)             curve:
342: (0)             ``[[float, float, float], [float, float, float], [float, float, float],
343: [float, float, float]]``.
344: (0)
345: (4)             CubicBezierPoints_Array: TypeAlias = Union[
346: (0)                 npt.NDArray[PointDType], tuple[CubicBezierPoints, ...]
347: (0)             ]
348: (0)             """``shape: (N, 4, 3)``
349: (0)             An array of :math:`N` `CubicBezierPoints` objects:
350: [float, float, float]], ...]``.
351: (0)
352: (0)             CubicBezierPath: TypeAlias = Point3D_Array
353: (0)             """``shape: (4*N, 3)``
354: (0)             A `Point3D_Array` of :math:`4N` points, where each one of the
355: (0)             :math:`N` consecutive blocks of 4 points represents a cubic Bézier
356: (0)             curve:
357: (0)             ``[[float, float, float], ..., ...]``.
358: (0)
359: (0)             Please refer to the documentation of the function you are using for
360: further type information.
361: (0)
362: (0)             CubicSpline: TypeAlias = CubicBezierPath
363: (0)             """``shape: (4*N, 3)``
364: (0)             A special case of `CubicBezierPath` where all the :math:`N` cubic
365: (0)             Bézier curves are connected, forming a quadratic spline:
366: (0)             ``[[float, float, float], ..., ...]``.
367: (0)
368: (0)             BezierPoints: TypeAlias = Point3D_Array
369: (0)             r"""``shape: (PPC, 3)``
370: (0)             A `Point3D_Array` of :math:`\text{PPC}` control points
371: (:math:\text{PPC}: Points Per Curve} = n + 1) for a single
372: (0)             :math:n`-th degree Bézier curve:
373: (0)             ``[[float, float, float], ...]``.
374: (0)
375: (0)             Please refer to the documentation of the function you are using for
376: further type information.
377: (0)
378: (0)             BezierPoints_Array: TypeAlias = Union[npt.NDArray[PointDType],
379: tuple[BezierPoints, ...]]
380: (0)             r"""``shape: (N, PPC, 3)``
381: (0)             An array of :math:`N` `BezierPoints` objects containing
382: (:math:\text{PPC}: Point3D` objects each
383: (:math:\text{PPC}: Points Per Curve} = n + 1):
384: (0)             ``[[[float, float, float], ...], ...]``.
385: (0)
386: (0)             BezierPath: TypeAlias = Point3D_Array
387: (0)             r"""``shape: (PPC*N, 3)``
388: (0)             A `Point3D_Array` of :math:\text{PPC} \cdot N` points, where each
389: (0)             one of the :math:N` consecutive blocks of :math:\text{PPC}` control
390: (0)             points (:math:\text{PPC}: Points Per Curve} = n + 1) represents a

```

```

391: (0)             Bézier curve of :math:`n`-th degree:  

392: (0)             ``[[float, float, float], ...], ...]``.  

393: (0)             Please refer to the documentation of the function you are using for  

394: (0)             further type information.  

395: (0)  

396: (0)             Spline: TypeAlias = BezierPath  

397: (0)             r"""\`shape: (PPC*N, 3)`  

398: (0)             A special case of `BezierPath` where all the :math:`N` Bézier curves  

399: (0)             consisting of :math:`\text{PPC}` `Point3D` objects  

400: (0)             (:math:`\text{Points Per Curve} = n + 1` ) are connected, forming  

401: (0)             an :math:`n`-th degree spline:  

402: (0)             ``[[float, float, float], ...], ...]``.  

403: (0)             Please refer to the documentation of the function you are using for  

404: (0)             further type information.  

405: (0)  

406: (0)             FlatBezierPoints: TypeAlias = Union[npt.NDArray[PointDType], tuple[float,  

...]]  

407: (0)             """`shape: (3*PPC*N,)`  

408: (0)             A flattened array of Bézier control points:  

409: (0)             ``[float, ...]``.  

410: (0)  

411: (0)  

412: (0)             [CATEGORY]  

413: (0)             Function types  

414: (0)  

415: (0)             FunctionOverride: TypeAlias = Callable  

416: (0)             """Function type returning an :class:`~.Animation` for the specified  

417: (0)             :class:`~.Mobject`.  

418: (0)  

419: (0)             PathFuncType: TypeAlias = Callable[[Point3D, Point3D, float], Point3D]  

420: (0)             """Function mapping two `Point3D` objects and an alpha value to a new  

421: (0)             `Point3D`.  

422: (0)  

423: (0)             MappingFunction: TypeAlias = Callable[[Point3D], Point3D]  

424: (0)             """A function mapping a `Point3D` to another `Point3D`.""""  

425: (0)  

426: (0)             [CATEGORY]  

427: (0)             Image types  

428: (0)  

429: (0)             Image: TypeAlias = npt.NDArray[ManimInt]  

430: (0)             """`shape: (height, width) | (height, width, 3) | (height, width, 4)`  

431: (0)             A rasterized image with a height of ``height`` pixels and a width of  

432: (0)             ``width`` pixels.  

433: (0)             Every value in the array is an integer from 0 to 255.  

434: (0)             Every pixel is represented either by a single integer indicating its  

435: (0)             lightness (for greyscale images), an `RGB_Array_Int` or an  

436: (0)             `RGBA_Array_Int`.  

437: (0)  

438: (0)             GrayscaleImage: TypeAlias = Image  

439: (0)             """`shape: (height, width)`  

440: (0)             A 100% opaque grayscale `Image`, where every pixel value is a  

441: (0)             `ManimInt` indicating its lightness (black -> gray -> white).  

442: (0)  

443: (0)             RGBImage: TypeAlias = Image  

444: (0)             """`shape: (height, width, 3)`  

445: (0)             A 100% opaque `Image` in color, where every pixel value is an  

446: (0)             `RGB_Array_Int` object.  

447: (0)  

448: (0)             RGBAImage: TypeAlias = Image  

449: (0)             """`shape: (height, width, 4)`  

450: (0)             An `Image` in color where pixels can be transparent. Every pixel  

451: (0)             value is an `RGBA_Array_Int` object.  

452: (0)  

453: (0)  

454: (0)             [CATEGORY]  

455: (0)             Path types  

456: (0)  

457: (0)             StrPath: TypeAlias = Union[str, PathLike[str]]  

458: (0)             """A string or :class:`os.PathLike` representing a path to a

```

```

459: (0)             directory or file.
460: (0)
461: (0)             StrOrBytesPath: TypeAlias = Union[str, bytes, PathLike[str], PathLike[bytes]]
462: (0)             """A string, bytes or :class:`os.PathLike` object representing a path
463: (0)             to a directory or file.
464: (0)             """
-----
```

File 120 - bezier.py:

```

1: (0)             """Utility functions related to Bézier curves."""
2: (0)             from __future__ import annotations
3: (0)             from manim.typing import (
4: (4)                 BezierPoints,
5: (4)                 ColVector,
6: (4)                 MatrixMN,
7: (4)                 Point3D,
8: (4)                 Point3D_Array,
9: (4)                 PointDType,
10: (4)                QuadraticBezierPoints,
11: (4)                QuadraticBezierPoints_Array,
12: (0)
13: (0)            )
14: (4)            __all__ = [
15: (4)                "bezier",
16: (4)                "partial_bezier_points",
17: (4)                "partial_quadratic_bezier_points",
18: (4)                "interpolate",
19: (4)                "integer_interpolate",
20: (4)                "mid",
21: (4)                "inverse_interpolate",
22: (4)                "match_interpolate",
23: (4)                "get_smooth_handle_points",
24: (4)                "get_smooth_cubic_bezier_handle_points",
25: (4)                "diag_to_matrix",
26: (4)                "is_closed",
27: (4)                "proportions_along_bezier_curve_for_point",
28: (4)                "point_lies_on_bezier",
29: (0)
30: (0)            from functools import reduce
31: (0)            from typing import Any, Callable, Sequence, overload
32: (0)            import numpy as np
33: (0)            import numpy.typing as npt
34: (0)            from scipy import linalg
35: (0)            from ..utils.simple_functions import choose
36: (0)            from ..utils.space_ops import cross2d, find_intersection
37: (0)            def bezier(
38: (4)                points: Sequence[Point3D] | Point3D_Array,
39: (0)            ) -> Callable[[float], Point3D]:
40: (4)                """Classic implementation of a bezier curve.
41: (4)                Parameters
42: (4)                -----
43: (8)                points
44: (4)                    points defining the desired bezier curve.
45: (4)                Returns
46: (4)                -----
47: (8)                    function describing the bezier curve.
48: (0)                    You can pass a t value between 0 and 1 to get the corresponding point
49: (0)                    on the curve.
50: (4)
51: (8)                    """
52: (12)                   n = len(points) - 1
53: (12)                   if n == 3:
54: (12)                       return lambda t: np.asarray(
55: (12)                           (1 - t) ** 3 * points[0]
56: (12)                           + 3 * t * (1 - t) ** 2 * points[1]
57: (12)                           + 3 * (1 - t) * t**2 * points[2]
58: (12)                           + t**3 * points[3],
59: (12)                           dtype=PointDType,
60: (8)                   )
```

```

58: (4)             if n == 2:
59: (8)                 return lambda t: np.asarray(
60: (12)                     (1 - t) ** 2 * points[0] + 2 * t * (1 - t) * points[1] + t**2 *
61: (12)                         dtype=PointDType,
62: (8)                 )
63: (4)             return lambda t: np.asarray(
64: (8)                 np.asarray(
65: (12)                     [
66: (16)                         (((1 - t) ** (n - k)) * (t**k) * choose(n, k) * point)
67: (16)                         for k, point in enumerate(points)
68: (12)                     ],
69: (12)                         dtype=PointDType,
70: (8)                     ).sum(axis=0)
71: (4)             )
72: (0)         def partial_bezier_points(points: BezierPoints, a: float, b: float) ->
BezierPoints:
73: (4)             """Given an array of points which define bezier curve, and two numbers
0<=a<b<=1, return an array of the same size,
74: (4)                 which describes the portion of the original bezier curve on the interval
[a, b].
75: (4)             This algorithm is pretty nifty, and pretty dense.
76: (4)             Parameters
77: (4)             -----
78: (4)             points
79: (8)                 set of points defining the bezier curve.
80: (4)             a
81: (8)                 lower bound of the desired partial bezier curve.
82: (4)             b
83: (8)                 upper bound of the desired partial bezier curve.
84: (4)             Returns
85: (4)             -----
86: (4)             np.ndarray
87: (8)                 Set of points defining the partial bezier curve.
88: (4)             """
89: (4)             _len = len(points)
90: (4)             if a == 1:
91: (8)                 return np.asarray([points[-1]] * _len, dtype=PointDType)
92: (4)             a_to_1 = np.asarray(
93: (8)                 [bezier(points[i:])(a) for i in range(_len)],
94: (8)                 dtype=PointDType,
95: (4)             )
96: (4)             end_prop = (b - a) / (1.0 - a)
97: (4)             return np.asarray(
98: (8)                 [bezier(a_to_1[: i + 1])(end_prop) for i in range(_len)],
99: (8)                 dtype=PointDType,
100: (4)             )
101: (0)         def partial_quadratic_bezier_points(
102: (4)             points: QuadraticBezierPoints, a: float, b: float
103: (0)         ) -> QuadraticBezierPoints:
104: (4)             if a == 1:
105: (8)                 return np.asarray(3 * [points[-1]])
106: (4)             def curve(t: float) -> Point3D:
107: (8)                 return np.asarray(
108: (12)                     points[0] * (1 - t) * (1 - t)
109: (12)                     + 2 * points[1] * t * (1 - t)
110: (12)                     + points[2] * t * t
111: (8)                 )
112: (4)             h0 = curve(a) if a > 0 else points[0]
113: (4)             h2 = curve(b) if b < 1 else points[2]
114: (4)             h1_prime = (1 - a) * points[1] + a * points[2]
115: (4)             end_prop = (b - a) / (1.0 - a)
116: (4)             h1 = (1 - end_prop) * h0 + end_prop * h1_prime
117: (4)             return np.asarray((h0, h1, h2))
118: (0)         def split_quadratic_bezier(points: QuadraticBezierPoints, t: float) ->
BezierPoints:
119: (4)             """Split a quadratic Bézier curve at argument ``t`` into two quadratic
curves.
120: (4)             Parameters

```

```

121: (4)           -----
122: (4)           points
123: (8)             The control points of the bezier curve
124: (8)             has shape ``[a1, h1, b1]``
125: (4)           t
126: (8)             The ``t``-value at which to split the Bézier curve
127: (4)           Returns
128: (4)           -----
129: (8)             The two Bézier curves as a list of tuples,
130: (8)             has the shape ``[a1, h1, b1], [a2, h2, b2]``
131: (4)           """
132: (4)           a1, h1, a2 = points
133: (4)           s1 = interpolate(a1, h1, t)
134: (4)           s2 = interpolate(h1, a2, t)
135: (4)           p = interpolate(s1, s2, t)
136: (4)           return np.array((a1, s1, p, p, s2, a2))
137: (0)           def subdivide_quadratic_bezier(points: QuadraticBezierPoints, n: int) ->
138: (4)             """Subdivide a quadratic Bézier curve into ``n`` subcurves which have the
139: (4)             same shape.
140: (4)             The points at which the curve is split are located at the
141: (4)             arguments :math:`t = i/n` for :math:`i = 1, \dots, n-1`.
142: (4)             Parameters
143: (4)             -----
144: (4)             points
145: (4)               The control points of the Bézier curve in form ``[a1, h1, b1]``
146: (4)             n
147: (4)               The number of curves to subdivide the Bézier curve into
148: (4)             Returns
149: (4)             -----
150: (4)             The new points for the Bézier curve in the form ``[a1, h1, b1, a2, h2,
151: (4)             b2, ...]``
152: (4)             ..
153: (4)             .. image:: /_static/bezier_subdivision_example.png
154: (4)             """
155: (4)             beziers = np.empty((n, 3, 3))
156: (4)             current = points
157: (4)             for j in range(0, n):
158: (4)               i = n - j
159: (4)               tmp = split_quadratic_bezier(current, 1 / i)
160: (0)             def quadratic_bezier_remap(
161: (4)               triplets: QuadraticBezierPoints_Array, new_number_of_curves: int
162: (0)             ) -> QuadraticBezierPoints_Array:
163: (4)               """Remaps the number of curves to a higher amount by splitting bezier
164: (4)               curves
165: (4)               Parameters
166: (4)               -----
167: (4)               triplets
168: (4)                 The triplets of the quadratic bezier curves to be remapped shape(n, 3,
169: (4)                 3)
170: (4)                 new_number_of_curves
171: (4)                   The number of curves that the output will contain. This needs to be
172: (4)                   higher than the current number.
173: (4)                   Returns
174: (4)                   -----
175: (4)                   The new triplets for the quadratic bezier curves.
176: (4)                   """
177: (4)                   difference = new_number_of_curves - len(triplets)
178: (4)                   if difference <= 0:
179: (4)                     return triplets
180: (4)                   new_triplets = np.zeros((new_number_of_curves, 3, 3))
181: (4)                   idx = 0
182: (4)                   for triplet in triplets:
183: (4)                     if difference > 0:
184: (4)                       tmp_noc = int(np.ceil(difference / len(triplets))) + 1
185: (4)                       tmp = subdivide_quadratic_bezier(triplet, tmp_noc).reshape(-1, 3,
3)

```

```

183: (12)             for i in range(tmp_noc):
184: (16)                 new_triplets[idx + i] = tmp[i]
185: (12)                 difference -= tmp_noc - 1
186: (12)                 idx += tmp_noc
187: (8)             else:
188: (12)                 new_triplets[idx] = triplet
189: (12)                 idx += 1
190: (4)             return new_triplets
191: (4)
192: (4)             """
193: (4)             This is an alternate version of the function just for documentation
194: (4)             purposes
195: (4)             -----
196: (4)             difference = new_number_of_curves - len(triplets)
197: (4)             if difference <= 0:
198: (8)                 return triplets
199: (4)             new_triplets = []
200: (4)             for triplet in triplets:
201: (8)                 if difference > 0:
202: (12)                     tmp_noc = int(np.ceil(difference / len(triplets))) + 1
203: (16)                     tmp = subdivide_quadratic_bezier(triplet, tmp_noc).reshape(-1, 3,
204: (12)                         3)
205: (8)
206: (12)                     for i in range(tmp_noc):
207: (16)                         new_triplets.append(tmp[i])
208: (12)                     difference -= tmp_noc - 1
209: (8)                 else:
210: (12)                     new_triplets.append(triplet)
211: (4)             return new_triplets
212: (4)
213: (4)
214: (4)             @overload
215: (0)             def interpolate(start: float, end: float, alpha: float) -> float: ...
216: (0)             @overload
217: (0)             def interpolate(start: Point3D, end: Point3D, alpha: float) -> Point3D: ...
218: (0)             def interpolate(
219: (4)                 start: int | float | Point3D, end: int | float | Point3D, alpha: float |
220: (4)                     ) -> float | Point3D:
221: (8)                 return (1 - alpha) * start + alpha * end
222: (4)             def integer_interpolate(
223: (8)                 start: float,
224: (8)                 end: float,
225: (8)                 alpha: float,
226: (4)             ) -> tuple[int, float]:
227: (4)
228: (4)                 """
229: (8)                 This is a variant of interpolate that returns an integer and the residual
230: (4)                 Parameters
231: (4)                 -----
232: (4)                 start
233: (8)                     The start of the range
234: (4)                 end
235: (8)                     The end of the range
236: (4)                 alpha
237: (8)                     a float between 0 and 1.
238: (4)             Returns
239: (4)                 -----
240: (4)                 tuple[int, float]
241: (8)                     This returns an integer between start and end (inclusive) representing
242: (8)                     appropriate interpolation between them, along with a
243: (8)                     "residue" representing a new proportion between the
244: (8)                     returned integer and the next one of the
245: (8)                     list.
246: (4)             Example
247: (4)             -----
248: (4)             .. code-block:: pycon
249: (8)                 >>> integer, residue = integer_interpolate(start=0, end=10,
250: (8)                               alpha=0.46)
251: (8)                 >>> np.allclose((integer, residue), (4, 0.6))
252: (8)                 True
253: (4)
254: (4)                 """
255: (4)                 if alpha >= 1:

```

```

248: (8)             return (int(end - 1), 1.0)
249: (4)             if alpha <= 0:
250: (8)                 return (int(start), 0)
251: (4)             value = int(interpolate(start, end, alpha))
252: (4)             residue = ((end - start) * alpha) % 1
253: (4)             return (value, residue)
254: (0)             @overload
255: (0)             def mid(start: float, end: float) -> float: ...
256: (0)             @overload
257: (0)             def mid(start: Point3D, end: Point3D) -> Point3D: ...
258: (0)             def mid(start: float | Point3D, end: float | Point3D) -> float | Point3D:
259: (4)                 """Returns the midpoint between two values.
260: (4)             Parameters
261: (4)                 -----
262: (4)                 start
263: (8)                     The first value
264: (4)                 end
265: (8)                     The second value
266: (4)             Returns
267: (4)                 -----
268: (8)                 The midpoint between the two values
269: (4)                 """
270: (4)             return (start + end) / 2.0
271: (0)             @overload
272: (0)             def inverse_interpolate(start: float, end: float, value: float) -> float: ...
273: (0)             @overload
274: (0)             def inverse_interpolate(start: float, end: float, value: Point3D) -> Point3D:
...
275: (0)             @overload
276: (0)             def inverse_interpolate(start: Point3D, end: Point3D, value: Point3D) ->
Point3D: ...
277: (0)             def inverse_interpolate(
278: (4)                 start: float | Point3D, end: float | Point3D, value: float | Point3D
279: (0)             ) -> float | Point3D:
280: (4)                 """Perform inverse interpolation to determine the alpha
281: (4)                 values that would produce the specified ``value``
282: (4)                 given the ``start`` and ``end`` values or points.
283: (4)             Parameters
284: (4)                 -----
285: (4)                 start
286: (8)                     The start value or point of the interpolation.
287: (4)                 end
288: (8)                     The end value or point of the interpolation.
289: (4)                 value
290: (8)                     The value or point for which the alpha value
291: (8)                     should be determined.
292: (4)             Returns
293: (4)                 -----
294: (8)                 The alpha values producing the given input
295: (8)                 when interpolating between ``start`` and ``end``.
296: (4)             Example
297: (4)                 -----
298: (4)             .. code-block:: pycon
299: (8)                 >>> inverse_interpolate(start=2, end=6, value=4)
300: (8)                 0.5
301: (8)                 >>> start = np.array([1, 2, 1])
302: (8)                 >>> end = np.array([7, 8, 11])
303: (8)                 >>> value = np.array([4, 5, 5])
304: (8)                 >>> inverse_interpolate(start, end, value)
305: (8)                 array([0.5, 0.5, 0.4])
306: (4)                 """
307: (4)                 return np.true_divide(value - start, end - start)
308: (0)             @overload
309: (0)             def match_interpolate(
310: (4)                 new_start: float,
311: (4)                 new_end: float,
312: (4)                 old_start: float,
313: (4)                 old_end: float,
314: (4)                 old_value: float,

```

```

315: (0)             ) -> float: ...
316: (0)             @overload
317: (0)             def match_interpolate(
318: (4)                 new_start: float,
319: (4)                 new_end: float,
320: (4)                 old_start: float,
321: (4)                 old_end: float,
322: (4)                 old_value: Point3D,
323: (0)             ) -> Point3D: ...
324: (0)             def match_interpolate(
325: (4)                 new_start: float,
326: (4)                 new_end: float,
327: (4)                 old_start: float,
328: (4)                 old_end: float,
329: (4)                 old_value: float | Point3D,
330: (0)             ) -> float | Point3D:
331: (4)             """Interpolate a value from an old range to a new range.
332: (4)             Parameters
333: (4)             -----
334: (4)             new_start
335: (8)                 The start of the new range.
336: (4)             new_end
337: (8)                 The end of the new range.
338: (4)             old_start
339: (8)                 The start of the old range.
340: (4)             old_end
341: (8)                 The end of the old range.
342: (4)             old_value
343: (8)                 The value within the old range whose corresponding
344: (8)                 value in the new range (with the same alpha value)
345: (8)                 is desired.
346: (4)             Returns
347: (4)             -----
348: (8)             The interpolated value within the new range.
349: (4)             Examples
350: (4)             -----
351: (4)             >>> match_interpolate(0, 100, 10, 20, 15)
352: (4)             50.0
353: (4)             """
354: (4)             old_alpha = inverse_interpolate(old_start, old_end, old_value)
355: (4)             return interpolate(
356: (8)                 new_start,
357: (8)                 new_end,
358: (8)                 old_alpha, # type: ignore
359: (4)             )
360: (0)             def get_smooth_cubic_bezier_handle_points(
361: (4)                 points: Point3D_Array,
362: (0)             ) -> tuple[BezierPoints, BezierPoints]:
363: (4)                 points = np.asarray(points)
364: (4)                 num_handles = len(points) - 1
365: (4)                 dim = points.shape[1]
366: (4)                 if num_handles < 1:
367: (8)                     return np.zeros((0, dim)), np.zeros((0, dim))
368: (4)                 l, u = 2, 1
369: (4)                 diag: MatrixMN = np.zeros((l + u + 1, 2 * num_handles))
370: (4)                 diag[0, 1::2] = -1
371: (4)                 diag[0, 2::2] = 1
372: (4)                 diag[1, 0::2] = 2
373: (4)                 diag[1, 1::2] = 1
374: (4)                 diag[2, 1:-2:2] = -2
375: (4)                 diag[3, 0:-3:2] = 1
376: (4)                 diag[2, -2] = -1
377: (4)                 diag[1, -1] = 2
378: (4)                 b: Point3D_Array = np.zeros((2 * num_handles, dim))
379: (4)                 b[1::2] = 2 * points[1:]
380: (4)                 b[0] = points[0]
381: (4)                 b[-1] = points[-1]
382: (4)                 def solve_func(b: ColVector) -> ColVector | MatrixMN:
383: (8)                     return linalg.solve_banded((l, u), diag, b) # type: ignore

```

```

384: (4)             use_closed_solve_function = is_closed(points)
385: (4)             if use_closed_solve_function:
386: (8)                 matrix = diag_to_matrix((l, u), diag)
387: (8)                 matrix[-1, [0, 1, -2, -1]] = [2, -1, 1, -2]
388: (8)                 matrix[0, :] = np.zeros(matrix.shape[1])
389: (8)                 matrix[0, [0, -1]] = [1, 1]
390: (8)                 b[0] = 2 * points[0]
391: (8)                 b[-1] = np.zeros(dim)
392: (8)             def closed_curve_solve_func(b: ColVector) -> ColVector | MatrixMN:
393: (12)                 return linalg.solve(matrix, b) # type: ignore
394: (4)             handle_pairs = np.zeros((2 * num_handles, dim))
395: (4)             for i in range(dim):
396: (8)                 if use_closed_solve_function:
397: (12)                     handle_pairs[:, i] = closed_curve_solve_func(b[:, i])
398: (8)                 else:
399: (12)                     handle_pairs[:, i] = solve_func(b[:, i])
400: (4)             return handle_pairs[0::2], handle_pairs[1::2]
401: (0)         def get_smooth_handle_points(
402: (4)             points: BezierPoints,
403: (0)         ) -> tuple[BezierPoints, BezierPoints]:
404: (4)             """Given some anchors (points), compute handles so the resulting bezier
curve is smooth.
405: (4)             Parameters
406: (4)             -----
407: (4)             points
408: (8)                 Anchors.
409: (4)             Returns
410: (4)             -----
411: (4)             typing.Tuple[np.ndarray, np.ndarray]
412: (8)                 Computed handles.
413: (4)             """
414: (4)             points = np.asarray(points)
415: (4)             num_handles = len(points) - 1
416: (4)             dim = points.shape[1]
417: (4)             if num_handles < 1:
418: (8)                 return np.zeros((0, dim)), np.zeros((0, dim))
419: (4)             l, u = 2, 1
420: (4)             diag: MatrixMN = np.zeros((l + u + 1, 2 * num_handles))
421: (4)             diag[0, 1::2] = -1
422: (4)             diag[0, 2::2] = 1
423: (4)             diag[1, 0::2] = 2
424: (4)             diag[1, 1::2] = 1
425: (4)             diag[2, 1:-2:2] = -2
426: (4)             diag[3, 0:-3:2] = 1
427: (4)             diag[2, -2] = -1
428: (4)             diag[1, -1] = 2
429: (4)             b = np.zeros((2 * num_handles, dim))
430: (4)             b[1::2] = 2 * points[1:]
431: (4)             b[0] = points[0]
432: (4)             b[-1] = points[-1]
433: (4)             def solve_func(b: ColVector) -> ColVector | MatrixMN:
434: (8)                 return linalg.solve_banded((l, u), diag, b) # type: ignore
435: (4)             use_closed_solve_function = is_closed(points)
436: (4)             if use_closed_solve_function:
437: (8)                 matrix = diag_to_matrix((l, u), diag)
438: (8)                 matrix[-1, [0, 1, -2, -1]] = [2, -1, 1, -2]
439: (8)                 matrix[0, :] = np.zeros(matrix.shape[1])
440: (8)                 matrix[0, [0, -1]] = [1, 1]
441: (8)                 b[0] = 2 * points[0]
442: (8)                 b[-1] = np.zeros(dim)
443: (8)             def closed_curve_solve_func(b: ColVector) -> ColVector | MatrixMN:
444: (12)                 return linalg.solve(matrix, b) # type: ignore
445: (4)             handle_pairs = np.zeros((2 * num_handles, dim))
446: (4)             for i in range(dim):
447: (8)                 if use_closed_solve_function:
448: (12)                     handle_pairs[:, i] = closed_curve_solve_func(b[:, i])
449: (8)                 else:
450: (12)                     handle_pairs[:, i] = solve_func(b[:, i])
451: (4)             return handle_pairs[0::2], handle_pairs[1::2]

```

```

452: (0)         def diag_to_matrix(
453: (4)             l_and_u: tuple[int, int], diag: npt.NDArray[Any]
454: (0)         ) -> npt.NDArray[Any]:
455: (4)             """
456: (4)                 Converts array whose rows represent diagonal
457: (4)                 entries of a matrix into the matrix itself.
458: (4)                 See scipy.linalg.solve_banded
459: (4)             """
460: (4)             l, u = l_and_u
461: (4)             dim = diag.shape[1]
462: (4)             matrix = np.zeros((dim, dim))
463: (4)             for i in range(l + u + 1):
464: (8)                 np.fill_diagonal(
465: (12)                     matrix[max(0, i - u) :, max(0, u - i) :],
466: (12)                     diag[i, max(0, u - i) :],
467: (8)                 )
468: (4)             return matrix
469: (0)         def get_quadratic_approximation_of_cubic(
470: (4)             a0: Point3D, h0: Point3D, h1: Point3D, a1: Point3D
471: (0)         ) -> BezierPoints:
472: (4)             a0 = np.array(a0, ndmin=2)
473: (4)             h0 = np.array(h0, ndmin=2)
474: (4)             h1 = np.array(h1, ndmin=2)
475: (4)             a1 = np.array(a1, ndmin=2)
476: (4)             T0 = h0 - a0
477: (4)             T1 = a1 - h1
478: (4)             has_infl = np.ones(len(a0), dtype=bool)
479: (4)             p = h0 - a0
480: (4)             q = h1 - 2 * h0 + a0
481: (4)             r = a1 - 3 * h1 + 3 * h0 - a0
482: (4)             a = cross2d(q, r)
483: (4)             b = cross2d(p, r)
484: (4)             c = cross2d(p, q)
485: (4)             disc = b * b - 4 * a * c
486: (4)             has_infl &= disc > 0
487: (4)             sqrt_disc = np.sqrt(np.abs(disc))
488: (4)             settings = np.seterr(all="ignore")
489: (4)             ti_bounds = []
490: (4)             for sgn in [-1, +1]:
491: (8)                 ti = (-b + sgn * sqrt_disc) / (2 * a)
492: (8)                 ti[a == 0] = (-c / b)[a == 0]
493: (8)                 ti[(a == 0) & (b == 0)] = 0
494: (8)                 ti_bounds.append(ti)
495: (4)             ti_min, ti_max = ti_bounds
496: (4)             np.seterr(**settings)
497: (4)             ti_min_in_range = has_infl & (0 < ti_min) & (ti_min < 1)
498: (4)             ti_max_in_range = has_infl & (0 < ti_max) & (ti_max < 1)
499: (4)             t_mid = 0.5 * np.ones(len(a0))
500: (4)             t_mid[ti_min_in_range] = ti_min[ti_min_in_range]
501: (4)             t_mid[ti_max_in_range] = ti_max[ti_max_in_range]
502: (4)             m, n = a0.shape
503: (4)             t_mid = t_mid.repeat(n).reshape((m, n))
504: (4)             mid = bezier([a0, h0, h1, a1])(t_mid) # type: ignore
505: (4)             Tm = bezier([h0 - a0, h1 - h0, a1 - h1])(t_mid) # type: ignore
506: (4)             i0 = find_intersection(a0, T0, mid, Tm)
507: (4)             i1 = find_intersection(a1, T1, mid, Tm)
508: (4)             m, n = np.shape(a0)
509: (4)             result = np.zeros((6 * m, n))
510: (4)             result[0::6] = a0
511: (4)             result[1::6] = i0
512: (4)             result[2::6] = mid
513: (4)             result[3::6] = mid
514: (4)             result[4::6] = i1
515: (4)             result[5::6] = a1
516: (4)             return result
517: (0)         def is_closed(points: Point3D_Array) -> bool:
518: (4)             return np.allclose(points[0], points[-1]) # type: ignore
519: (0)         def proportions_along_bezier_curve_for_point(
520: (4)             point: Point3D,

```

```

521: (4)             control_points: BezierPoints,
522: (4)             round_to: float = 1e-6,
523: (0)         ) -> npt.NDArray[Any]:
524: (4)             """Obtains the proportion along the bezier curve corresponding to a given
525: (4)             given the bezier curve's control points.
526: (4)             The bezier polynomial is constructed using the coordinates of the given
527: (4)             as well as the bezier curve's control points. On solving the polynomial
528: (4)             if there are roots common to every dimension, those roots give the
529: (4)             curve the point is at. If there are no real roots, the point does not lie
530: (4)             on the curve.
531: (4)
532: (4)
533: (8)             Parameters
534: (8)             -----
535: (4)             point
536: (8)                 The Cartesian Coordinates of the point whose parameter
537: (8)                 should be obtained.
538: (8)             control_points
539: (4)                 The Cartesian Coordinates of the ordered control
540: (8)                 points of the bezier curve on which the point may
541: (8)                 or may not lie.
542: (4)             round_to
543: (4)                 A float whose number of decimal places all values
544: (8)                 such as coordinates of points will be rounded.
545: (12)             Returns
546: (12)
547: (12)
548: (12)
549: (12)
550: (12)
551: (4)             -----
552: (4)             :class:`ValueError`
553: (4)                 When ``point`` and the control points have different shapes.
554: (8)
555: (4)
556: (4)             """
557: (8)
558: (12)             if not all(np.shape(point) == np.shape(c_p) for c_p in control_points):
559: (8)                 raise ValueError(
560: (4)                     f"Point {point} and Control Points {control_points} have different
561: (4)                     "
562: (4)
563: (4)
564: (8)             control_points = np.array(control_points)
565: (8)             n = len(control_points) - 1
566: (8)             roots = []
567: (12)             for dim, coord in enumerate(point):
568: (12)                 control_coords = control_points[:, dim]
569: (12)                 terms = []
570: (12)                 for term_power in range(n, -1, -1):
571: (16)                     outercoeff = choose(n, term_power)
572: (16)                     term = []
573: (16)                     sign = 1
574: (20)                     for subterm_num in range(term_power, -1, -1):
575: (16)                         innercoeff = choose(term_power, subterm_num) * sign
576: (16)                         subterm = innercoeff * control_coords[subterm_num]
577: (12)                         if term_power == 0:
578: (8)                             subterm -= coord
579: (12)                         term.append(subterm)
580: (8)                         sign *= -1
581: (8)                     terms.append(outercoeff * sum(np.array(term)))
582: (8)             if all(term == 0 for term in terms):
583: (8)                 continue
584: (8)             bezier_poly = np.polynomial.Polynomial(terms[::-1])
585: (8)             polyroots = bezier_poly.roots() # type: ignore
586: (8)             if len(polyroots) > 0:

```

```

583: (12)                                polynom_roots = np.around(polynomial_roots, int(np.log10(1 /
round_to)))
584: (8)                                roots.append(polynomial_roots)
585: (4)                                roots = [[root for root in rootlist if root.imag == 0] for rootlist in
roots]
586: (4)                                roots = reduce(np.intersect1d, roots) # type: ignore
587: (4)                                result = np.asarray([r.real for r in roots if 0 <= r.real <= 1])
588: (4)                                return result
589: (0)                                def point_lies_on_bezier(
590: (4)                                    point: Point3D,
591: (4)                                    control_points: BezierPoints,
592: (4)                                    round_to: float = 1e-6,
593: (0)                                ) -> bool:
594: (4)                                    """Checks if a given point lies on the bezier curves with the given
control points.
595: (4)                                    This is done by solving the bezier polynomial with the point as the
constant term; if
596: (4)                                    any real roots exist, the point lies on the bezier curve.
597: (4)                                    Parameters
598: (4)                                    -----
599: (4)                                    point
600: (8)                                    The Cartesian Coordinates of the point to check.
601: (4)                                    control_points
602: (8)                                    The Cartesian Coordinates of the ordered control
603: (8)                                    points of the bezier curve on which the point may
604: (8)                                    or may not lie.
605: (4)                                    round_to
606: (8)                                    A float whose number of decimal places all values
607: (8)                                    such as coordinates of points will be rounded.
608: (4)                                    Returns
609: (4)                                    -----
610: (4)                                    bool
611: (8)                                    Whether the point lies on the curve.
612: (4)                                    """
613: (4)                                roots = proportions_along_bezier_curve_for_point(point, control_points,
round_to)
614: (4)                                return len(roots) > 0

```

---

## File 121 - AS2700.py:

```

1: (0)                                """Australian Color Standard
2: (0)                                In 1985 the Australian Independent Color Standard AS 2700 was created. In
3: (0)                                this standard, all colors can be identified via a category code (one of
4: (0)                                B -- Blue, G -- Green, N -- Neutrals (grey), P -- Purple, R -- Red, T --
Blue/Green,
5: (0)                                X -- Yellow/Red, Y -- Yellow) and a number. The colors also have (natural)
names.
6: (0)                                To use the colors from this list, access them directly from the module (which
7: (0)                                is exposed to Manim's global name space):
8: (0)                                .. code:: pycon
9: (4)                                    >>> from manim import AS2700
10: (4)                                   >>> AS2700.B23_BRIGHT_BLUE
11: (4)                                   ManimColor('#174F90')
12: (0)                                   List of Color Constants
13: (0)                                   -----
14: (0)                                   These hex values (taken from
https://www.w3schools.com/colors/colors_australia.asp)
15: (0)                                   are non official approximate values intended to simulate AS 2700 colors:
16: (0)                                   .. automanimcolormodule:: manim.utils.color.AS2700
17: (0)                                   """
18: (0)                                   from .core import ManimColor
19: (0)                                   B11_RICH_BLUE = ManimColor("#2B3770")
20: (0)                                   B12_ROYAL_BLUE = ManimColor("#2C3563")
21: (0)                                   B13_NAVY_BLUE = ManimColor("#28304D")
22: (0)                                   B14_SAPPHIRE = ManimColor("#28426B")
23: (0)                                   B15_MID_BLUE = ManimColor("#144B6F")
24: (0)                                   B21_ULTRAMARINE = ManimColor("#2C5098")

```

```

25: (0)          B22_HOMEBUSH_BLUE = ManimColor("#215097")
26: (0)          B23_BRIGHT_BLUE = ManimColor("#174F90")
27: (0)          B24_HARBOUR_BLUE = ManimColor("#1C6293")
28: (0)          B25_AQUA = ManimColor("#5097AC")
29: (0)          B32_POWDER_BLUE = ManimColor("#B7C8DB")
30: (0)          B33_MIST_BLUE = ManimColor("#E0E6E2")
31: (0)          B34_PARADISE_BLUE = ManimColor("#3499BA")
32: (0)          B35_PALE_BLUE = ManimColor("#CDE4E2")
33: (0)          B41_BLUEBELL = ManimColor("#5B94D1")
34: (0)          B42_PURPLE_BLUE = ManimColor("#5E7899")
35: (0)          B43_GREY_BLUE = ManimColor("#627C8D")
36: (0)          B44_LIGHT_GREY_BLUE = ManimColor("#C0C0C1")
37: (0)          B45_SKY_BLUE = ManimColor("#7DB7C7")
38: (0)          B51_PERIWINKLE = ManimColor("#3871AC")
39: (0)          B53_DARK_GREY_BLUE = ManimColor("#4F6572")
40: (0)          B55_STORM_BLUE = ManimColor("#3F7C94")
41: (0)          B61_CORAL_SEA = ManimColor("#2B3873")
42: (0)          B62_MIDNIGHT_BLUE = ManimColor("#292A34")
43: (0)          B64_CHARCOAL = ManimColor("#363E45")
44: (0)          G11_BOTTLE_GREEN = ManimColor("#253A32")
45: (0)          G12 HOLLY = ManimColor("#21432D")
46: (0)          G13_EMERALD = ManimColor("#195F35")
47: (0)          G14_MOSS_GREEN = ManimColor("#33572D")
48: (0)          G15_RAINFOREST_GREEN = ManimColor("#3D492D")
49: (0)          G16_TRAFFIC_GREEN = ManimColor("#305442")
50: (0)          G17_MINT_GREEN = ManimColor("#006B45")
51: (0)          G21_JADE = ManimColor("#127453")
52: (0)          G22_SERPENTINE = ManimColor("#78A681")
53: (0)          G23_SHAMROCK = ManimColor("#336634")
54: (0)          G24_FERN_TREE = ManimColor("#477036")
55: (0)          G25_OLIVE = ManimColor("#595B2A")
56: (0)          G26_APPLE_GREEN = ManimColor("#4E9843")
57: (0)          G27_HOMEBUSH_GREEN = ManimColor("#017F4D")
58: (0)          G31_VERTIGRIS = ManimColor("#468A65")
59: (0)          G32_OPALINE = ManimColor("#AFCBB8")
60: (0)          G33_LETTUCE = ManimColor("#7B9954")
61: (0)          G34_AVOCADO = ManimColor("#757C4C")
62: (0)          G35_LIME_GREEN = ManimColor("#89922E")
63: (0)          G36_KIKUYU = ManimColor("#95B43B")
64: (0)          G37_BEANSTALK = ManimColor("#45A56A")
65: (0)          G41_LAWN_GREEN = ManimColor("#0D875D")
66: (0)          G42_GLACIER = ManimColor("#D5E1D2")
67: (0)          G43_SURF_GREEN = ManimColor("#C8C8A7")
68: (0)          G44_PALM_GREEN = ManimColor("#99B179")
69: (0)          G45_CHARTREUSE = ManimColor("#C7C98D")
70: (0)          G46_CITRONELLA = ManimColor("#BFC83E")
71: (0)          G47_CRYSTAL_GREEN = ManimColor("#ADCCA8")
72: (0)          G51_SPRUCE = ManimColor("#05674F")
73: (0)          G52_EUCALYPTUS = ManimColor("#66755B")
74: (0)          G53_BANKSIA = ManimColor("#929479")
75: (0)          G54_MIST_GREEN = ManimColor("#7A836D")
76: (0)          G55_LICHEN = ManimColor("#A7A98C")
77: (0)          G56_SAGE_GREEN = ManimColor("#677249")
78: (0)          G61_DARK_GREEN = ManimColor("#283533")
79: (0)          G62_RIVERGUM = ManimColor("#617061")
80: (0)          G63_DEEP_BRONZE_GREEN = ManimColor("#333334")
81: (0)          G64_SLATE = ManimColor("#5E6153")
82: (0)          G65_TI_TREE = ManimColor("#5D5F4E")
83: (0)          G66_ENVIRONMENT_GREEN = ManimColor("#484C3F")
84: (0)          G67_ZUCCHINI = ManimColor("#2E443A")
85: (0)          N11_PEARL_GREY = ManimColor("#D8D3C7")
86: (0)          N12_PASTEL_GREY = ManimColor("#CCCCCC")
87: (0)          N14_WHITE = ManimColor("#FFFFFF")
88: (0)          N15_HOMEBUSH_GREY = ManimColor("#A29B93")
89: (0)          N22_CLOUD_GREY = ManimColor("#C4C1B9")
90: (0)          N23_NEUTRAL_GREY = ManimColor("#CCCCCC")
91: (0)          N24_SILVER_GREY = ManimColor("#BDC7C5")
92: (0)          N25_BIRCH_GREY = ManimColor("#ABA498")
93: (0)          N32_GREEN_GREY = ManimColor("#8E9282")

```

```

94: (0)           N33_LIGHTBOX_GREY = ManimColor("#ACADAD")
95: (0)           N35_LIGHT_GREY = ManimColor("#A6A7A1")
96: (0)           N41_OYSTER = ManimColor("#998F78")
97: (0)           N42_STORM_GREY = ManimColor("#858F88")
98: (0)           N43_PIPELINE_GREY = ManimColor("#999999")
99: (0)           N44_BRIDGE_GREY = ManimColor("#767779")
100: (0)          N45_KOALA_GREY = ManimColor("#928F88")
101: (0)          N52_MID_GREY = ManimColor("#727A77")
102: (0)          N53_BLUE_GREY = ManimColor("#7C8588")
103: (0)          N54_BASALT = ManimColor("#585C63")
104: (0)          N55_LEAD_GREY = ManimColor("#5E5C58")
105: (0)          N61_BLACK = ManimColor("#2A2A2C")
106: (0)          N63_PEWTER = ManimColor("#596064")
107: (0)          N64_DARK_GREY = ManimColor("#4B5259")
108: (0)          N65_GRAPHITE_GREY = ManimColor("#45474A")
109: (0)          P11_MAGENTA = ManimColor("#7B2B48")
110: (0)          P12_PURPLE = ManimColor("#85467B")
111: (0)          P13_VIOLET = ManimColor("#5D3A61")
112: (0)          P14_BLUEBERRY = ManimColor("#4C4176")
113: (0)          P21_SUNSET_PINK = ManimColor("#E3BBBD")
114: (0)          P22_CYCLAMEN = ManimColor("#83597D")
115: (0)          P23_LILAC = ManimColor("#A69FB1")
116: (0)          P24_JACKARANDA = ManimColor("#795F91")
117: (0)          P31_DUSTY_PINK = ManimColor("#DBBEBC")
118: (0)          P33_RIBBON_PINK = ManimColor("#D1BCC9")
119: (0)          P41_ERICA_PINK = ManimColor("#C55A83")
120: (0)          P42_MULBERRY = ManimColor("#A06574")
121: (0)          P43_WISTERIA = ManimColor("#756D91")
122: (0)          P52_PLUM = ManimColor("#6E3D4B")
123: (0)          R11_INTERNATIONAL_ORANGE = ManimColor("#CE482A")
124: (0)          R12_SCARLET = ManimColor("#CD392A")
125: (0)          R13_SIGNAL_RED = ManimColor("#BA312B")
126: (0)          R14_WARATAH = ManimColor("#AA2429")
127: (0)          R15_CRIMSON = ManimColor("#9E2429")
128: (0)          R21_TANGERINE = ManimColor("#E96957")
129: (0)          R22_HOMEBUSH_RED = ManimColor("#D83A2D")
130: (0)          R23_LOLLIPOP = ManimColor("#CC5058")
131: (0)          R24_STRAWBERRY = ManimColor("#B4292A")
132: (0)          R25_ROSE_PINK = ManimColor("#E8919C")
133: (0)          R32_APPLE_BLOSSOM = ManimColor("#F2E1D8")
134: (0)          R33_GHOST_GUM = ManimColor("#E8DAD4")
135: (0)          R34_MUSHROOM = ManimColor("#D7C0B6")
136: (0)          R35_DEEP_ROSE = ManimColor("#CD6D71")
137: (0)          R41_SHELL_PINK = ManimColor("#F9D9BB")
138: (0)          R42_SALMON_PINK = ManimColor("#D99679")
139: (0)          R43_RED_DUST = ManimColor("#D0674F")
140: (0)          R44_POSSUM = ManimColor("#A18881")
141: (0)          R45_RUBY = ManimColor("#8F3E5C")
142: (0)          R51_BURNT_PINK = ManimColor("#E19B8E")
143: (0)          R52_TERRACOTTA = ManimColor("#A04C36")
144: (0)          R53_RED_GUM = ManimColor("#8D4338")
145: (0)          R54_RASPBERRY = ManimColor("#852F31")
146: (0)          R55_CLARET = ManimColor("#67292D")
147: (0)          R62_VENETIAN_RED = ManimColor("#77372B")
148: (0)          R63_RED_OXIDE = ManimColor("#663334")
149: (0)          R64_DEEP_INDIAN_RED = ManimColor("#542E2B")
150: (0)          R65_MAROON = ManimColor("#3F2B3C")
151: (0)          T11_TROPICAL_BLUE = ManimColor("#006698")
152: (0)          T12_DIAMANTIA = ManimColor("#006C74")
153: (0)          T14_MALACHITE = ManimColor("#105154")
154: (0)          T15_TURQUOISE = ManimColor("#098587")
155: (0)          T22_ORIENTAL_BLUE = ManimColor("#358792")
156: (0)          T24_BLUE_JADE = ManimColor("#427F7E")
157: (0)          T32_HUON_GREEN = ManimColor("#72B3B1")
158: (0)          T33_SMOKE_BLUE = ManimColor("#9EB6B2")
159: (0)          T35_GREEN_ICE = ManimColor("#78AEA2")
160: (0)          T44_BLUE_GUM = ManimColor("#6A8A88")
161: (0)          T45_COOTAMUNDRA = ManimColor("#759E91")
162: (0)          T51_MOUNTAIN_BLUE = ManimColor("#295668")

```

```

163: (0)          T53_PEACOCK_BLUE = ManimColor("#245764")
164: (0)          T63_TEAL = ManimColor("#183F4E")
165: (0)          X11_BUTTERSCOTCH = ManimColor("#D38F43")
166: (0)          X12_PUMPKIN = ManimColor("#DD7E1A")
167: (0)          X13_MARIGOLD = ManimColor("#ED7F15")
168: (0)          X14_MANDARIN = ManimColor("#E45427")
169: (0)          X15_ORANGE = ManimColor("#E36C2B")
170: (0)          X21_PALE_OCHRE = ManimColor("#DAA45F")
171: (0)          X22_SAFFRON = ManimColor("#F6AA51")
172: (0)          X23_APRICOT = ManimColor("#FEB56D")
173: (0)          X24_ROCKMELON = ManimColor("#F6894B")
174: (0)          X31_RAFFIA = ManimColor("#EBC695")
175: (0)          X32_MAGNOLIA = ManimColor("#F1DEBE")
176: (0)          X33_WARM_WHITE = ManimColor("#F3E7D4")
177: (0)          X34_DRIFTWOOD = ManimColor("#D5C4AE")
178: (0)          X41_BUFF = ManimColor("#C28A44")
179: (0)          X42_BISCUIT = ManimColor("#DEBA92")
180: (0)          X43_BEIGE = ManimColor("#C9AA8C")
181: (0)          X45_CINNAMON = ManimColor("#AC826D")
182: (0)          X51_TAN = ManimColor("#8F5F32")
183: (0)          X52_COFFEE = ManimColor("#AD7948")
184: (0)          X53_GOLDEN_TAN = ManimColor("#925629")
185: (0)          X54_BROWN = ManimColor("#68452C")
186: (0)          X55_NUT_BROWN = ManimColor("#764832")
187: (0)          X61_WOMBAT = ManimColor("#6E5D52")
188: (0)          X62_DARK_EARTH = ManimColor("#6E5D52")
189: (0)          X63_IRONBARK = ManimColor("#443B36")
190: (0)          X64_CHOCOLATE = ManimColor("#4A3B31")
191: (0)          X65_DARK_BROWN = ManimColor("#4F372D")
192: (0)          Y11_CANARY = ManimColor("#E7BD11")
193: (0)          Y12_WATTLE = ManimColor("#E8AF01")
194: (0)          Y13_VIVID_YELLOW = ManimColor("#FCAE01")
195: (0)          Y14_GOLDEN_YELLOW = ManimColor("#F5A601")
196: (0)          Y15_SUNFLOWER = ManimColor("#FFA709")
197: (0)          Y16_INCA_GOLD = ManimColor("#DF8C19")
198: (0)          Y21_PRIMROSE = ManimColor("#F5CF5B")
199: (0)          Y22_CUSTARD = ManimColor("#EFD25C")
200: (0)          Y23_BUTTERCUP = ManimColor("#E0CD41")
201: (0)          Y24_STRAW = ManimColor("#E3C882")
202: (0)          Y25_DEEP_CREAM = ManimColor("#F3C968")
203: (0)          Y26_HOMEBUSH_GOLD = ManimColor("#FCC51A")
204: (0)          Y31_LILY_GREEN = ManimColor("#E3E3CD")
205: (0)          Y32_FLUMMERY = ManimColor("#E6DF9E")
206: (0)          Y33_PALE_PRIMROSE = ManimColor("#F5F3CE")
207: (0)          Y34_CREAM = ManimColor("#FFE3BE")
208: (0)          Y35_OFF_WHITE = ManimColor("#F1E9D5")
209: (0)          Y41 OLIVE_YELLOW = ManimColor("#8E7426")
210: (0)          Y42_MUSTARD = ManimColor("#C4A32E")
211: (0)          Y43_PARCHMENT = ManimColor("#D4C9A3")
212: (0)          Y44_SAND = ManimColor("#DCC18B")
213: (0)          Y45_MANILLA = ManimColor("#E5D0A7")
214: (0)          Y51_BRONZE_OLIVE = ManimColor("#695D3E")
215: (0)          Y52_CHAMOIS = ManimColor("#BEA873")
216: (0)          Y53 SANDSTONE = ManimColor("#D5BF8E")
217: (0)          Y54_OATMEAL = ManimColor("#CAAE82")
218: (0)          Y55_DEEP_STONE = ManimColor("#BC9969")
219: (0)          Y56_MERINO = ManimColor("#C9B79E")
220: (0)          Y61_BLACK_OLIVE = ManimColor("#47473B")
221: (0)          Y62_SUGAR_CANE = ManimColor("#BCA55C")
222: (0)          Y63_KHAKI = ManimColor("#826843")
223: (0)          Y65_MUSHROOM = ManimColor("#A39281")
224: (0)          Y66_MUDSTONE = ManimColor("#574E45")

```

---

File 122 - section.py:

```

1: (0)          """building blocks of segmented video API"""
2: (0)          from __future__ import annotations

```

```

3: (0)             from enum import Enum
4: (0)             from pathlib import Path
5: (0)             from typing import Any
6: (0)             from manim import get_video_metadata
7: (0)             __all__ = ["Section", "DefaultSectionType"]
8: (0)             class DefaultSectionType(str, Enum):
9: (4)                 """The type of a section can be used for third party applications.
10: (4)                 A presentation system could for example use the types to create loops.
11: (4)                 Examples
12: (4)                 -----
13: (4)                 This class can be reimplemented for more types::
14: (8)                 class PresentationSectionType(str, Enum):
15: (12)                     NORMAL = "presentation.normal"
16: (12)                     SKIP = "presentation.skip"
17: (12)                     LOOP = "presentation.loop"
18: (12)                     COMPLETE_LOOP = "presentation.complete_loop"
19: (4)                     """
20: (4)                     NORMAL = "default.normal"
21: (0)             class Section:
22: (4)                 """A :class:`Scene` can be segmented into multiple Sections.
23: (4)                 Refer to :doc:`the documentation</tutorials/output_and_config>` for more
info.
24: (4)                 It consists of multiple animations.
25: (4)                 Attributes
26: (4)                 -----
27: (4)                 type
28: (8)                     Can be used by a third party applications to classify different types
of sections.
29: (4)
30: (8)                     video
31: (8)                         Path to video file with animations belonging to section relative to
sections directory.
32: (4)
33: (8)                     name
34: (4)                         Human readable, non-unique name for this section.
35: (8)                     skip_animations
36: (4)                         Skip rendering the animations in this section when ``True``.
37: (8)                     partial_movie_files
38: (4)                         Animations belonging to this section.
39: (4)                     See Also
40: (4)                     -----
41: (4)                     :class:`.DefaultSectionType`
42: (4)                     :meth:`.CairoRenderer.update_skipping_status`
43: (4)                     :meth:`.OpenGLRenderer.update_skipping_status`
44: (4)                     """
45: (8)                     def __init__(self, type: str, video: str | None, name: str,
skip_animations: bool):
46: (8)                         self.type = type
47: (8)                         self.video: str | None = video
48: (8)                         self.name = name
49: (8)                         self.skip_animations = skip_animations
50: (4)                         self.partial_movie_files: list[str | None] = []
51: (8)                     def is_empty(self) -> bool:
52: (8)                         """Check whether this section is empty.
53: (8)                         Note that animations represented by ``None`` are also counted.
54: (8)                         """
55: (4)                         return len(self.partial_movie_files) == 0
56: (4)                     def get_clean_partial_movie_files(self) -> list[str]:
57: (8)                         """Return all partial movie files that are not ``None``."""
58: (8)                         return [el for el in self.partial_movie_files if el is not None]
59: (4)                     def get_dict(self, sections_dir: Path) -> dict[str, Any]:
60: (8)                         """Get dictionary representation with metadata of output video.
61: (8)                         The output from this function is used from every section to build the
sections index file.
62: (8)                         The output video must have been created in the ``sections_dir`` before
executing this method.
63: (8)                         This is the main part of the Segmented Video API.
64: (8)                         """
65: (12)                         if self.video is None:
66: (8)                             raise ValueError(

```

```

66: (16)             f"Section '{self.name}' cannot be exported as dict, it does
not have a video path assigned to it"
67: (12)         )
68: (8)             video_metadata = get_video_metadata(sections_dir / self.video)
69: (8)             return dict(
70: (12)             {
71: (16)                 "name": self.name,
72: (16)                 "type": self.type,
73: (16)                 "video": self.video,
74: (12)             },
75: (12)             **video_metadata,
76: (8)         )
77: (4)     def __repr__(self):
78: (8)         return f"<Section '{self.name}' stored in '{self.video}'>"
```

-----

File 123 - caching.py:

```

1: (0)             from __future__ import annotations
2: (0)             from typing import Callable
3: (0)             from .. import config, logger
4: (0)             from ..utils.hashing import get_hash_from_play_call
5: (0)             __all__ = ["handle_caching_play"]
6: (0)     def handle_caching_play(func: Callable[..., None]):
7: (4)             """Decorator that returns a wrapped version of func that will compute
8: (4)             the hash of the play invocation.
9: (4)             The returned function will act according to the computed hash: either skip
10: (4)             the animation because it's already cached, or let the invoked function
11: (4)             play normally.
12: (4)             Parameters
13: (4)             -----
14: (4)             func
15: (8)                 The play like function that has to be written to the video file
stream.
16: (8)             Take the same parameters as `scene.play`.
17: (4)             """
18: (4)     def wrapper(self, scene, *args, **kwargs):
19: (8)         self.skip_animations = self._original_skipping_status
20: (8)         self.update_skipping_status()
21: (8)         animations = scene.compile_animations(*args, **kwargs)
22: (8)         scene.add_mobjects_from_animations(animations)
23: (8)         if self.skip_animations:
24: (12)             logger.debug(f"Skipping animation {self.num_plays}")
25: (12)             func(self, scene, *args, **kwargs)
26: (12)             self.animations_hashes.append(None)
27: (12)             self.file_writer.add_partial_movie_file(None)
28: (12)             return
29: (8)         if not config["disable_caching"]:
30: (12)             mobjects_on_scene = scene.mobjects
31: (12)             hash_play = get_hash_from_play_call(
32: (16)                 self,
33: (16)                 self.camera,
34: (16)                 animations,
35: (16)                 mobjects_on_scene,
36: (12)             )
37: (12)             if self.file_writer.is_already_cached(hash_play):
38: (16)                 logger.info(
39: (20)                     f"Animation {self.num_plays} : Using cached data (hash : %
(hash_play)s)",
40: (20)
41: (16)
42: (16)
43: (8)
44: (12)
45: (8)
46: (8)
47: (8)
48: (12)
49: (16)
50: (16)
51: (16)
52: (16)
53: (16)
54: (16)
55: (16)
56: (16)
57: (16)
58: (16)
59: (16)
60: (16)
61: (16)
62: (16)
63: (16)
64: (16)
65: (16)
66: (16)
67: (16)
68: (16)
69: (16)
70: (16)
71: (16)
72: (16)
73: (16)
74: (16)
75: (16)
76: (16)
77: (16)
78: (16)
79: (16)
80: (16)
81: (16)
82: (16)
83: (16)
84: (16)
85: (16)
86: (16)
87: (16)
88: (16)
89: (16)
90: (16)
91: (16)
92: (16)
93: (16)
94: (16)
95: (16)
96: (16)
97: (16)
98: (16)
99: (16)
100: (16)
101: (16)
102: (16)
103: (16)
104: (16)
105: (16)
106: (16)
107: (16)
108: (16)
109: (16)
110: (16)
111: (16)
112: (16)
113: (16)
114: (16)
115: (16)
116: (16)
117: (16)
118: (16)
119: (16)
120: (16)
121: (16)
122: (16)
123: (16)
124: (16)
125: (16)
126: (16)
127: (16)
128: (16)
129: (16)
130: (16)
131: (16)
132: (16)
133: (16)
134: (16)
135: (16)
136: (16)
137: (16)
138: (16)
139: (16)
140: (16)
141: (16)
142: (16)
143: (16)
144: (16)
145: (16)
146: (16)
147: (16)
148: (16)
149: (16)
150: (16)
151: (16)
152: (16)
153: (16)
154: (16)
155: (16)
156: (16)
157: (16)
158: (16)
159: (16)
160: (16)
161: (16)
162: (16)
163: (16)
164: (16)
165: (16)
166: (16)
167: (16)
168: (16)
169: (16)
170: (16)
171: (16)
172: (16)
173: (16)
174: (16)
175: (16)
176: (16)
177: (16)
178: (16)
179: (16)
180: (16)
181: (16)
182: (16)
183: (16)
184: (16)
185: (16)
186: (16)
187: (16)
188: (16)
189: (16)
190: (16)
191: (16)
192: (16)
193: (16)
194: (16)
195: (16)
196: (16)
197: (16)
198: (16)
199: (16)
200: (16)
201: (16)
202: (16)
203: (16)
204: (16)
205: (16)
206: (16)
207: (16)
208: (16)
209: (16)
210: (16)
211: (16)
212: (16)
213: (16)
214: (16)
215: (16)
216: (16)
217: (16)
218: (16)
219: (16)
220: (16)
221: (16)
222: (16)
223: (16)
224: (16)
225: (16)
226: (16)
227: (16)
228: (16)
229: (16)
230: (16)
231: (16)
232: (16)
233: (16)
234: (16)
235: (16)
236: (16)
237: (16)
238: (16)
239: (16)
240: (16)
241: (16)
242: (16)
243: (16)
244: (16)
245: (16)
246: (16)
247: (16)
248: (16)
249: (16)
250: (16)
251: (16)
252: (16)
253: (16)
254: (16)
255: (16)
256: (16)
257: (16)
258: (16)
259: (16)
260: (16)
261: (16)
262: (16)
263: (16)
264: (16)
265: (16)
266: (16)
267: (16)
268: (16)
269: (16)
270: (16)
271: (16)
272: (16)
273: (16)
274: (16)
275: (16)
276: (16)
277: (16)
278: (16)
279: (16)
280: (16)
281: (16)
282: (16)
283: (16)
284: (16)
285: (16)
286: (16)
287: (16)
288: (16)
289: (16)
290: (16)
291: (16)
292: (16)
293: (16)
294: (16)
295: (16)
296: (16)
297: (16)
298: (16)
299: (16)
300: (16)
301: (16)
302: (16)
303: (16)
304: (16)
305: (16)
306: (16)
307: (16)
308: (16)
309: (16)
310: (16)
311: (16)
312: (16)
313: (16)
314: (16)
315: (16)
316: (16)
317: (16)
318: (16)
319: (16)
320: (16)
321: (16)
322: (16)
323: (16)
324: (16)
325: (16)
326: (16)
327: (16)
328: (16)
329: (16)
330: (16)
331: (16)
332: (16)
333: (16)
334: (16)
335: (16)
336: (16)
337: (16)
338: (16)
339: (16)
340: (16)
341: (16)
342: (16)
343: (16)
344: (16)
345: (16)
346: (16)
347: (16)
348: (16)
349: (16)
350: (16)
351: (16)
352: (16)
353: (16)
354: (16)
355: (16)
356: (16)
357: (16)
358: (16)
359: (16)
360: (16)
361: (16)
362: (16)
363: (16)
364: (16)
365: (16)
366: (16)
367: (16)
368: (16)
369: (16)
370: (16)
371: (16)
372: (16)
373: (16)
374: (16)
375: (16)
376: (16)
377: (16)
378: (16)
379: (16)
380: (16)
381: (16)
382: (16)
383: (16)
384: (16)
385: (16)
386: (16)
387: (16)
388: (16)
389: (16)
390: (16)
391: (16)
392: (16)
393: (16)
394: (16)
395: (16)
396: (16)
397: (16)
398: (16)
399: (16)
400: (16)
401: (16)
402: (16)
403: (16)
404: (16)
405: (16)
406: (16)
407: (16)
408: (16)
409: (16)
410: (16)
411: (16)
412: (16)
413: (16)
414: (16)
415: (16)
416: (16)
417: (16)
418: (16)
419: (16)
420: (16)
421: (16)
422: (16)
423: (16)
424: (16)
425: (16)
426: (16)
427: (16)
428: (16)
429: (16)
430: (16)
431: (16)
432: (16)
433: (16)
434: (16)
435: (16)
436: (16)
437: (16)
438: (16)
439: (16)
440: (16)
441: (16)
442: (16)
443: (16)
444: (16)
445: (16)
446: (16)
447: (16)
448: (16)
449: (16)
450: (16)
451: (16)
452: (16)
453: (16)
454: (16)
455: (16)
456: (16)
457: (16)
458: (16)
459: (16)
460: (16)
461: (16)
462: (16)
463: (16)
464: (16)
465: (16)
466: (16)
467: (16)
468: (16)
469: (16)
470: (16)
471: (16)
472: (16)
473: (16)
474: (16)
475: (16)
476: (16)
477: (16)
478: (16)
479: (16)
480: (16)
481: (16)
482: (16)
483: (16)
484: (16)
485: (16)
486: (16)
487: (16)
488: (16)
489: (16)
490: (16)
491: (16)
492: (16)
493: (16)
494: (16)
495: (16)
496: (16)
497: (16)
498: (16)
499: (16)
500: (16)
501: (16)
502: (16)
503: (16)
504: (16)
505: (16)
506: (16)
507: (16)
508: (16)
509: (16)
510: (16)
511: (16)
512: (16)
513: (16)
514: (16)
515: (16)
516: (16)
517: (16)
518: (16)
519: (16)
520: (16)
521: (16)
522: (16)
523: (16)
524: (16)
525: (16)
526: (16)
527: (16)
528: (16)
529: (16)
530: (16)
531: (16)
532: (16)
533: (16)
534: (16)
535: (16)
536: (16)
537: (16)
538: (16)
539: (16)
540: (16)
541: (16)
542: (16)
543: (16)
544: (16)
545: (16)
546: (16)
547: (16)
548: (16)
549: (16)
550: (16)
551: (16)
552: (16)
553: (16)
554: (16)
555: (16)
556: (16)
557: (16)
558: (16)
559: (16)
560: (16)
561: (16)
562: (16)
563: (16)
564: (16)
565: (16)
566: (16)
567: (16)
568: (16)
569: (16)
570: (16)
571: (16)
572: (16)
573: (16)
574: (16)
575: (16)
576: (16)
577: (16)
578: (16)
579: (16)
580: (16)
581: (16)
582: (16)
583: (16)
584: (16)
585: (16)
586: (16)
587: (16)
588: (16)
589: (16)
590: (16)
591: (16)
592: (16)
593: (16)
594: (16)
595: (16)
596: (16)
597: (16)
598: (16)
599: (16)
600: (16)
601: (16)
602: (16)
603: (16)
604: (16)
605: (16)
606: (16)
607: (16)
608: (16)
609: (16)
610: (16)
611: (16)
612: (16)
613: (16)
614: (16)
615: (16)
616: (16)
617: (16)
618: (16)
619: (16)
620: (16)
621: (16)
622: (16)
623: (16)
624: (16)
625: (16)
626: (16)
627: (16)
628: (16)
629: (16)
630: (16)
631: (16)
632: (16)
633: (16)
634: (16)
635: (16)
636: (16)
637: (16)
638: (16)
639: (16)
640: (16)
641: (16)
642: (16)
643: (16)
644: (16)
645: (16)
646: (16)
647: (16)
648: (16)
649: (16)
650: (16)
651: (16)
652: (16)
653: (16)
654: (16)
655: (16)
656: (16)
657: (16)
658: (16)
659: (16)
660: (16)
661: (16)
662: (16)
663: (16)
664: (16)
665: (16)
666: (16)
667: (16)
668: (16)
669: (16)
670: (16)
671: (16)
672: (16)
673: (16)
674: (16)
675: (16)
676: (16)
677: (16)
678: (16)
679: (16)
680: (16)
681: (16)
682: (16)
683: (16)
684: (16)
685: (16)
686: (16)
687: (16)
688: (16)
689: (16)
690: (16)
691: (16)
692: (16)
693: (16)
694: (16)
695: (16)
696: (16)
697: (16)
698: (16)
699: (16)
700: (16)
701: (16)
702: (16)
703: (16)
704: (16)
705: (16)
706: (16)
707: (16)
708: (16)
709: (16)
710: (16)
711: (16)
712: (16)
713: (16)
714: (16)
715: (16)
716: (16)
717: (16)
718: (16)
719: (16)
720: (16)
721: (16)
722: (16)
723: (16)
724: (16)
725: (16)
726: (16)
727: (16)
728: (16)
729: (16)
730: (16)
731: (16)
732: (16)
733: (16)
734: (16)
735: (16)
736: (16)
737: (16)
738: (16)
739: (16)
740: (16)
741: (16)
742: (16)
743: (16)
744: (16)
745: (16)
746: (16)
747: (16)
748: (16)
749: (16)
750: (16)
751: (16)
752: (16)
753: (16)
754: (16)
755: (16)
756: (16)
757: (16)
758: (16)
759: (16)
760: (16)
761: (16)
762: (16)
763: (16)
764: (16)
765: (16)
766: (16)
767: (16)
768: (16)
769: (16)
770: (16)
771: (16)
772: (16)
773: (16)
774: (16)
775: (16)
776: (16)
777: (16)
778: (16)
779: (16)
780: (16)
781: (16)
782: (16)
783: (16)
784: (16)
785: (16)
786: (16)
787: (16)
788: (16)
789: (16)
790: (16)
791: (16)
792: (16)
793: (16)
794: (16)
795: (16)
796: (16)
797: (16)
798: (16)
799: (16)
800: (16)
801: (16)
802: (16)
803: (16)
804: (16)
805: (16)
806: (16)
807: (16)
808: (16)
809: (16)
810: (16)
811: (16)
812: (16)
813: (16)
814: (16)
815: (16)
816: (16)
817: (16)
818: (16)
819: (16)
820: (16)
821: (16)
822: (16)
823: (16)
824: (16)
825: (16)
826: (16)
827: (16)
828: (16)
829: (16)
830: (16)
831: (16)
832: (16)
833: (16)
834: (16)
835: (16)
836: (16)
837: (16)
838: (16)
839: (16)
840: (16)
841: (16)
842: (16)
843: (16)
844: (16)
845: (16)
846: (16)
847: (16)
848: (16)
849: (16)
850: (16)
851: (16)
852: (16)
853: (16)
854: (16)
855: (16)
856: (16)
857: (16)
858: (16)
859: (16)
860: (16)
861: (16)
862: (16)
863: (16)
864: (16)
865: (16)
866: (16)
867: (16)
868: (16)
869: (16)
870: (16)
871: (16)
872: (16)
873: (16)
874: (16)
875: (16)
876: (16)
877: (16)
878: (16)
879: (16)
880: (16)
881: (16)
882: (16)
883: (16)
884: (16)
885: (16)
886: (16)
887: (16)
888: (16)
889: (16)
890: (16)
891: (16)
892: (16)
893: (16)
894: (16)
895: (16)
896: (16)
897: (16)
898: (16)
899: (16)
900: (16)
901: (16)
902: (16)
903: (16)
904: (16)
905: (16)
906: (16)
907: (16)
908: (16)
909: (16)
910: (16)
911: (16)
912: (16)
913: (16)
914: (16)
915: (16)
916: (16)
917: (16)
918: (16)
919: (16)
920: (16)
921: (16)
922: (16)
923: (16)
924: (16)
925: (16)
926: (16)
927: (16)
928: (16)
929: (16)
930: (16)
931: (16)
932: (16)
933: (16)
934: (16)
935: (16)
936: (16)
937: (16)
938: (16)
939: (16)
940: (16)
941: (16)
942: (16)
943: (16)
944: (16)
945: (16)
946: (16)
947: (16)
948: (16)
949: (16)
950: (16)
951: (16)
952: (16)
953: (16)
954: (16)
955: (16)
956: (16)
957: (16)
958: (16)
959: (16)
960: (16)
961: (16)
962: (16)
963: (16)
964: (16)
965: (16)
966: (16)
967: (16)
968: (16)
969: (16)
970: (16)
971: (16)
972: (16)
973: (16)
974: (16)
975: (16)
976: (16)
977: (16)
978: (16)
979: (16)
980: (16)
981: (16)
982: (16)
983: (16)
984: (16)
985: (16)
986: (16)
987: (16)
988: (16)
989: (16)
990: (16)
991: (16)
992: (16)
993: (16)
994: (16)
995: (16)
996: (16)
997: (16)
998: (16)
999: (16)
1000: (16)
```

```

49: (12)                 {"h": str(self.animations_hashes[:5])},
50: (8)             )
51: (8)         func(self, scene, *args, **kwargs)
52: (4)     return wrapper

```

-----

## File 124 - commands.py:

```

1: (0)         from __future__ import annotations
2: (0)         import json
3: (0)         import os
4: (0)         from pathlib import Path
5: (0)         from subprocess import run
6: (0)         from typing import Generator
7: (0)     __all__ = [
8: (4)         "capture",
9: (4)         "get_video_metadata",
10: (4)         "get_dir_layout",
11: (0)     ]
12: (0)     def capture(command, cwd=None, command_input=None):
13: (4)         p = run(
14: (8)             command,
15: (8)             cwd=cwd,
16: (8)             input=command_input,
17: (8)             capture_output=True,
18: (8)             text=True,
19: (8)             encoding="utf-8",
20: (4)         )
21: (4)         out, err = p.stdout, p.stderr
22: (4)         return out, err, p.returncode
23: (0)     def get_video_metadata(path_to_video: str | os.PathLike) -> dict[str]:
24: (4)         command = [
25: (8)             "ffprobe",
26: (8)             "-v",
27: (8)             "error",
28: (8)             "-select_streams",
29: (8)             "v:0",
30: (8)             "-show_entries",
31: (8)             "stream=width,height,nb_frames,duration,avg_frame_rate,codec_name",
32: (8)             "-print_format",
33: (8)             "json",
34: (8)             str(path_to_video),
35: (4)         ]
36: (4)         config, err, exitcode = capture(command)
37: (4)         assert exitcode == 0, f"FFprobe error: {err}"
38: (4)         return json.loads(config)["streams"][0]
39: (0)     def get_dir_layout(dirpath: Path) -> Generator[str, None, None]:
40: (4)         """Get list of paths relative to dirpath of all files in dir and subdirs
recursively."""
41: (4)         for p in dirpath.iterdir():
42: (8)             if p.is_dir():
43: (12)                 yield from get_dir_layout(p)
44: (12)                 continue
45: (8)             yield str(p.relative_to(dirpath))

```

-----

## File 125 - \_\_init\_\_.py:

```
1: (0)
```

-----

## File 126 - \_\_init\_\_.py:

```

1: (0)         """Utilities for working with colors and predefined color constants.
2: (0)         Color data structure
3: (0)         -----

```

```

4: (0)          .. autosummary::
5: (3)          :toctree: ../reference
6: (3)          core
7: (0)          Predefined colors
-----
8: (0)          There are several predefined colors available in Manim:
9: (0)          - The colors listed in :mod:`.color.manim_colors` are loaded into
10: (0)         Manim's global name space.
11: (2)         - The colors in :mod:`.color.AS2700`, :mod:`.color.BS381`, :mod:`.color.X11`,
12: (0)         and :mod:`.color.XKCD` need to be accessed via their module (which are
13: (2)
available
14: (2)           in Manim's global name space), or imported separately. For example:
15: (2)           .. code:: pycon
16: (5)             >>> from manim import XKCD
17: (5)             >>> XKCD.AVOCADO
18: (5)             ManimColor('#90B134')
19: (2)           Or, alternatively:
20: (2)           .. code:: pycon
21: (5)             >>> from manim.utils.color.XKCD import AVOCADO
22: (5)             >>> AVOCADO
23: (5)             ManimColor('#90B134')
24: (0)           The following modules contain the predefined color constants:
25: (0)           .. autosummary::
26: (3)             :toctree: ../reference
27: (3)             manim_colors
28: (3)             AS2700
29: (3)             BS381
30: (3)             XKCD
31: (3)             X11
32: (0)
33: (0)           """
34: (0)           from typing import Dict, List
35: (0)           from . import AS2700, BS381, X11, XKCD
36: (0)           from .core import *
37: (0)           from .manim_colors import *
38: (4)           _all_color_dict: Dict[str, ManimColor] = {
39: (4)             k: v for k, v in globals().items() if isinstance(v, ManimColor)
}

```

#### File 127 - \_\_init\_\_.py:

```

1: (0)          """Utilities for building the Manim documentation.
2: (0)          For more information about the Manim documentation building, see:
3: (0)          - :doc:`/contributing/development`, specifically the ``Documentation``
4: (4)          bullet point under :ref:`polishing-changes-and-submitting-a-pull-request`  

5: (0)          - :doc:`/contributing/docs`
6: (0)          .. autosummary::
7: (3)            :toctree: ../reference
8: (3)            autoaliasattr_directive
9: (3)            autocolor_directive
10: (3)           manim_directive
11: (3)           module_parsing
12: (0)

```

#### File 128 - config\_ops.py:

```

1: (0)          """Utilities that might be useful for configuration dictionaries."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = [
4: (4)            "merge_dicts_recursively",
5: (4)            "update_dict_recursively",
6: (4)            "DictAsObject",
7: (0)            ]
8: (0)          import itertools as it
9: (0)          import numpy as np
10: (0)         def merge_dicts_recursively(*dicts):

```

```

11: (4)          """
12: (4)          Creates a dict whose keyset is the union of all the
13: (4)          input dictionaries. The value for each key is based
14: (4)          on the first dict in the list with that key.
15: (4)          dicts later in the list have higher priority
16: (4)          When values are dictionaries, it is applied recursively
17: (4)          """
18: (4)          result = {}
19: (4)          all_items = it.chain(*(d.items() for d in dicts))
20: (4)          for key, value in all_items:
21: (8)              if key in result and isinstance(result[key], dict) and
isinstance(value, dict):
22: (12)                  result[key] = merge_dicts_recursively(result[key], value)
23: (8)              else:
24: (12)                  result[key] = value
25: (4)          return result
26: (0)          def update_dict_recursively(current_dict, *others):
27: (4)              updated_dict = merge_dicts_recursively(current_dict, *others)
28: (4)              current_dict.update(updated_dict)
29: (0)          class DictAsObject:
30: (4)              def __init__(self, dictin):
31: (8)                  self.__dict__ = dictin
32: (0)          class _Data:
33: (4)              """Descriptor that allows _Data variables to be grouped and accessed from
self.data["attr"] via self.attr.
34: (4)                  self.data attributes must be arrays.
35: (4)                  """
36: (4)                  def __set_name__(self, obj, name):
37: (8)                      self.name = name
38: (4)                  def __get__(self, obj, owner):
39: (8)                      return obj.data[self.name]
40: (4)                  def __set__(self, obj, array: np.ndarray):
41: (8)                      obj.data[self.name] = array
42: (0)          class _Uniforms:
43: (4)              """Descriptor that allows _Uniforms variables to be grouped from
self.uniforms["attr"] via self.attr.
44: (4)                  self.uniforms attributes must be floats.
45: (4)                  """
46: (4)                  def __set_name__(self, obj, name):
47: (8)                      self.name = name
48: (4)                  def __get__(self, obj, owner):
49: (8)                      return obj.__dict__["uniforms"][self.name]
50: (4)                  def __set__(self, obj, num: float):
51: (8)                      obj.__dict__["uniforms"][self.name] = num

```

---

## File 129 - deprecation.py:

```

1: (0)          """Decorators for deprecating classes, functions and function parameters."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = ["deprecated", "deprecated_params"]
4: (0)          import inspect
5: (0)          import re
6: (0)          from typing import Any, Callable, Iterable
7: (0)          from decorator import decorate, decorator
8: (0)          from .. import logger
9: (0)          def _get_callable_info(callable: Callable) -> tuple[str, str]:
10: (4)              """Returns type and name of a callable.
11: (4)              Parameters
12: (4)              -----
13: (4)              callable
14: (8)                  The callable
15: (4)              Returns
16: (4)              -----
17: (4)              Tuple[str, str]
18: (8)                  The type and name of the callable. Type can be one of "class",
"method" (for
19: (8)                  functions defined in classes) or "function"). For methods, name is

```

```

Class.method.
20: (4)         """
21: (4)             what = type(callable).__name__
22: (4)             name = callable.__qualname__
23: (4)             if what == "function" and "." in name:
24: (8)                 what = "method"
25: (4)             elif what != "function":
26: (8)                 what = "class"
27: (4)             return (what, name)
28: (0)         def _deprecation_text_component(
29: (4)             since: str | None,
30: (4)             until: str | None,
31: (4)             message: str,
32: (0)         ) -> str:
33: (4)             """Generates a text component used in deprecation messages.
34: (4)             Parameters
35: (4)             -----
36: (4)             since
37: (8)                 The version or date since deprecation
38: (4)             until
39: (8)                 The version or date until removal of the deprecated callable
40: (4)             message
41: (8)                 The reason for why the callable has been deprecated
42: (4)             Returns
43: (4)             -----
44: (4)             str
45: (8)                 The deprecation message text component.
46: (4)             """
47: (4)             since = f"since {since}" if since else ""
48: (4)             until = (
49: (8)                 f"is expected to be removed after {until}"
50: (8)                 if until
51: (8)                 else "may be removed in a later version"
52: (4)             )
53: (4)             msg = " " + message if message else ""
54: (4)             return f"deprecated {since}and {until}.{{msg}}"
55: (0)         def deprecated(
56: (4)             func: Callable = None,
57: (4)             since: str | None = None,
58: (4)             until: str | None = None,
59: (4)             replacement: str | None = None,
60: (4)             message: str | None = "",
61: (0)         ) -> Callable:
62: (4)             """Decorator to mark a callable as deprecated.
63: (4)             The decorated callable will cause a warning when used. The docstring of
64: (4)             deprecated callable is adjusted to indicate that this callable is
65: (4)             Parameters
66: (4)             -----
67: (4)             func
68: (8)                 The function to be decorated. Should not be set by the user.
69: (4)             since
70: (8)                 The version or date since deprecation.
71: (4)             until
72: (8)                 The version or date until removal of the deprecated callable.
73: (4)             replacement
74: (8)                 The identifier of the callable replacing the deprecated one.
75: (4)             message
76: (8)                 The reason for why the callable has been deprecated.
77: (4)             Returns
78: (4)             -----
79: (4)             Callable
80: (8)                 The decorated callable.
81: (4)             Examples
82: (4)             -----
83: (4)             Basic usage::
84: (8)                 from manim.utils.deprecation import deprecated
85: (8)                 @deprecated

```

```

86: (8)             def foo(**kwargs):
87: (12)            pass
88: (8)            @deprecated
89: (8)            class Bar:
90: (12)              def __init__(self):
91: (16)                pass
92: (12)                @deprecated
93: (12)                def baz(self):
94: (16)                  pass
95: (8)                  foo()
96: (8)                  a = Bar()
97: (8)                  a.baz()
98: (4) You can specify additional information for a more precise warning:::
99: (8)             from manim.utils.deprecation import deprecated
100: (8)            @deprecated(
101: (12)              since="v0.2",
102: (12)              until="v0.4",
103: (12)              replacement="bar",
104: (12)              message="It is cooler."
105: (8)
106: (8)            )
107: (12)            def foo():
108: (8)              pass
109: (8)              foo()
110: (4) You may also use dates instead of versions:::
111: (8)             from manim.utils.deprecation import deprecated
112: (8)             @deprecated(since="05/01/2021", until="06/01/2021")
113: (12)             def foo():
114: (8)               pass
115: (8)               foo()
116: (4)
117: (8)               """
118: (4)               if func is None:
119: (8)                 return lambda func: deprecated(func, since, until, replacement,
120: (8)                   message)
121: (8)                   what, name = _get_callable_info(func)
122: (8)                   def warning_msg(for_docs: bool = False) -> str:
123: (8)                     """Generate the deprecation warning message.
124: (12)                     Parameters
125: (8)                     -----
126: (8)                     for_docs
127: (8)                       Whether or not to format the message for use in documentation.
128: (12)                     Returns
129: (8)                     -----
130: (8)                     str
131: (8)                       The deprecation message.
132: (12)
133: (12)
134: (16)                     msg = message
135: (16)                     if replacement is not None:
136: (12)                       repl = replacement
137: (8)                       if for_docs:
138: (12)                         mapper = {"class": "class", "method": "meth", "function":
139: (16)                           "func"}
140: (12)                           repl = f":{mapper[what]}:{~.{replacement}}"
141: (8)                           msg = f"Use {repl} instead.{'' + message if message else ''}"
142: (8)                           deprecated = _deprecation_text_component(since, until, msg)
143: (8)                           return f"The {what} {name} has been {deprecated}"
144: (12)                     def deprecate_docs(func: Callable):
145: (8)                       """Adjust docstring to indicate the deprecation.
146: (8)                       Parameters
147: (8)                       -----
148: (8)                       func
149: (12)                         The callable whose docstring to adjust.
150: (8)                         """
151: (8)                         warning = warning_msg(True)
152: (8)                         doc_string = func.__doc__ or ""
153: (8)                         func.__doc__ = f"{doc_string}\n\n.. attention:: Deprecated\n{n
154: (8) {warning}}
155: (4)                     def deprecate(func: Callable, *args, **kwargs):
156: (8)                       """The actual decorator used to extend the callables behavior.
157: (8)                       Logs a warning message.

```

```

152: (8)                               Parameters
153: (8)-----func
154: (8)                               The callable to decorate.
155: (12)-----args
156: (8)                               The arguments passed to the given callable.
157: (12)-----kwargs
158: (8)                               The keyword arguments passed to the given callable.
159: (12)-----Returns
160: (8)-----Any
161: (8)                               The return value of the given callable when being passed the given
162: (8)                               arguments.
163: (12)-----"""
164: (12)                               logger.warning(warning_msg())
165: (8)-----return func(*args, **kwargs)
166: (8)-----if type(func).__name__ != "function":
167: (8)                               deprecate_docs(func)
168: (4)-----func.__init__ = decorate(func.__init__, deprecate)
169: (8)-----return func
170: (8)-----func = decorate(func, deprecate)
171: (8)-----deprecate_docs(func)
172: (4)-----return func
173: (4)-----def deprecated_params(
174: (4)                               params: str | Iterable[str] | None = None,
175: (0)                               since: str | None = None,
176: (4)                               until: str | None = None,
177: (4)                               message: str | None = "",
178: (4)                               redirections: None | (
179: (4)                                   Iterable[tuple[str, str] | Callable[..., dict[str, Any]]]
180: (4)                               ) = None,
181: (8)-----) -> Callable:
182: (4)-----"""Decorator to mark parameters of a callable as deprecated.
183: (0)-----It can also be used to automatically redirect deprecated parameter values
184: (4)-----replacements.
185: (4)-----Parameters
186: (4)-----params
187: (4)-----The parameters to be deprecated. Can consist of:
188: (4)-----* An iterable of strings, with each element representing a parameter
189: (4)-----* A single string, with parameter names separated by commas or spaces.
190: (8)-----since
191: (8)-----The version or date since deprecation.
192: (8)-----until
193: (4)-----The version or date until removal of the deprecated callable.
194: (8)-----message
195: (4)-----The reason for why the callable has been deprecated.
196: (8)-----redirections
197: (4)-----A list of parameter redirections. Each redirection can be one of the
198: (4)-----* A tuple of two strings. The first string defines the name of the
199: (4)-----parameter; the second string defines the name of the parameter to
200: (8)-----when attempting to use the first string.
201: (8)-----* A function performing the mapping operation. The parameter names of
202: (10)-----function determine which parameters are used as input. The function
203: (10)-----return a dictionary which contains the redirected arguments.
204: (8)-----Redirected parameters are also implicitly deprecated.
205: (10)-----Returns
206: (8)-----Callable
207: (8)-----The decorated callable.
208: (4)-----Raises
209: (4)-----"""
210: (4)-----"""
211: (8)-----"""
212: (4)-----"""
213: (4)-----"""

```

```

214: (4)                         ValueError
215: (8)                         If no parameters are defined (neither explicitly nor implicitly).
216: (4)                         ValueError
217: (8)                         If defined parameters are invalid python identifiers.
218: (4)                         Examples
219: (4)                         -----
220: (4)                         Basic usage::
221: (8)                         from manim.utils.deprecation import deprecated_params
222: (8)                         @deprecated_params(params="a, b, c")
223: (8)                         def foo(**kwargs):
224: (12)                           pass
225: (8)                           foo(x=2, y=3, z=4)
226: (8)                           foo(a=2, b=3, z=4)
227: (4)                         You can also specify additional information for a more precise warning::
228: (8)                         from manim.utils.deprecation import deprecated_params
229: (8)                         @deprecated_params(
230: (12)                           params="a, b, c",
231: (12)                           since="v0.2",
232: (12)                           until="v0.4",
233: (12)                           message="The letters x, y, z are cooler."
234: (8)                         )
235: (8)                         def foo(**kwargs):
236: (12)                           pass
237: (8)                           foo(a=2)
238: (4)                         Basic parameter redirection::
239: (8)                         from manim.utils.deprecation import deprecated_params
240: (8)                         @deprecated_params(redirections=[
241: (12)                           ("old_param", "new_param"),
242: (12)                           lambda old_param2: {"new_param2": old_param2}
243: (8)                         ])
244: (8)                         def foo(**kwargs):
245: (12)                           return kwargs
246: (8)                           foo(x=1, old_param=2)
247: (4)                         Redirecting using a calculated value::
248: (8)                         from manim.utils.deprecation import deprecated_params
249: (8)                         @deprecated_params(redirections=[
250: (12)                           lambda runtime_in_ms: {"run_time": runtime_in_ms / 1000}
251: (8)                         ])
252: (8)                         def foo(**kwargs):
253: (12)                           return kwargs
254: (8)                           foo(runtime_in_ms=500)
255: (4)                         Redirecting multiple parameter values to one::
256: (8)                         from manim.utils.deprecation import deprecated_params
257: (8)                         @deprecated_params(redirections=[
258: (12)                           lambda buff_x=1, buff_y=1: {"buff": (buff_x, buff_y)}
259: (8)                         ])
260: (8)                         def foo(**kwargs):
261: (12)                           return kwargs
262: (8)                           foo(buff_x=2)
263: (4)                         Redirect one parameter to multiple::
264: (8)                         from manim.utils.deprecation import deprecated_params
265: (8)                         @deprecated_params(redirections=[
266: (12)                           lambda buff=1: {"buff_x": buff[0], "buff_y": buff[1]} if
isinstance(buff, tuple)
267: (20)                               else {"buff_x": buff,      "buff_y": buff}
268: (8)                         ])
269: (8)                         def foo(**kwargs):
270: (12)                           return kwargs
271: (8)                           foo(buff=0)
272: (8)                           foo(buff=(1,2))
273: (4)                         """
274: (4)                         if callable(params):
275: (8)                           raise ValueError("deprecate_parameters requires arguments to be
specified.")
276: (4)                         if params is None:
277: (8)                           params = []
278: (4)                           params = re.split(r"[,\s]+", params) if isinstance(params, str) else
list(params)
279: (4)                         if redirections is None:

```

```

280: (8)             redirections = []
281: (4)             for redirector in redirections:
282: (8)                 if isinstance(redirector, tuple):
283: (12)                     params.append(redirector[0])
284: (8)                 else:
285: (12)                     params.extend(list(inspect.signature(redirector).parameters))
286: (4)             params = list(dict.fromkeys(params))
287: (4)             identifier = re.compile(r"^\d\W*\Z", re.UNICODE)
288: (4)             if not all(re.match(identifier, param) for param in params):
289: (8)                 raise ValueError("Given parameter values are invalid.")
290: (4)             redirections = list(redirections)
291: (4)             def warning_msg(func: Callable, used: list[str]):
292: (8)                 """Generate the deprecation warning message.
293: (8)                 Parameters
294: (8)                 -----
295: (8)                 func
296: (12)                     The callable with deprecated parameters.
297: (8)                 used
298: (12)                     The list of deprecated parameters used in a call.
299: (8)
300: (8)
301: (8)
302: (12)             Returns
303: (8)             -----
304: (8)             str
305: (8)                 The deprecation message.
306: (8)
307: (8)             what, name = _get_callable_info(func)
308: (8)             plural = len(used) > 1
309: (8)             parameter_s = "s" if plural else ""
310: (8)             used_ = ", ".join(used[:-1]) + " and " + used[-1] if plural else
311: (8)             used[0]
312: (8)             has_have_been = "have been" if plural else "has been"
313: (8)             deprecated = _deprecation_text_component(since, until, message)
314: (8)             return f"The parameter{parameter_s} {used_} of {what} {name}"
315: (8)
316: (12)             {has_have_been} {deprecated}
317: (8)
318: (12)             def redirect_params(kwags: dict, used: list[str]):
319: (8)                 """Adjust the keyword arguments as defined by the redirections.
320: (8)                 Parameters
321: (8)                 -----
322: (8)                 kwags
323: (8)                     The keyword argument dictionary to be updated.
324: (8)                 used
325: (8)                     The list of deprecated parameters used in a call.
326: (8)
327: (12)             """
328: (16)             for redirector in redirections:
329: (16)                 if isinstance(redirector, tuple):
330: (20)                     old_param, new_param = redirector
331: (16)                     if old_param in used:
332: (16)                         kwags[new_param] = kwags.pop(old_param)
333: (16)                     else:
334: (16)                         redirector_params =
335: (16)                         list(inspect.signature(redirector).parameters)
336: (16)                         redirector_args = {}
337: (16)                         for redirector_param in redirector_params:
338: (20)                             if redirector_param in used:
339: (20)                                 redirector_args[redirector_param] =
340: (24)                                 kwargs.pop(redirector_param)
341: (16)                                 if len(redirector_args) > 0:
342: (20)                                     kwargs.update(redirector(**redirector_args))
343: (4)             def deprecate_params(func, *args, **kwargs):
344: (8)                 """The actual decorator function used to extend the callables
behavior.
345: (8)                 Logs a warning message when a deprecated parameter is used and
346: (8)                 redirects it if
347: (8)                     specified.
348: (8)                 Parameters
349: (8)                 -----
350: (8)                 func
351: (12)                     The callable to decorate.
352: (8)                 args
353: (12)                     The arguments passed to the given callable.

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
343: (8)                               kwargs
344: (12)                             The keyword arguments passed to the given callable.
345: (8)                               Returns
346: (8)                               -----
347: (8)                               Any
348: (12)                             The return value of the given callable when being passed the given
349: (12)                             arguments.
350: (8)                               """
351: (8)                               used = []
352: (8)                               for param in params:
353: (12)                                 if param in kwargs:
354: (16)                                   used.append(param)
355: (8)                               if len(used) > 0:
356: (12)                                 logger.warning(warning_msg(func, used))
357: (12)                                 redirect_params(kwargs, used)
358: (8)                               return func(*args, **kwargs)
359: (4)                               return decorator(deprecate_params)
```

---

## File 130 - zoomed\_scene.py:

```
1: (0)                               """A scene supporting zooming in on a specified section.
2: (0)                               Examples
3: (0)                               -----
4: (0)                               .. manim:: UseZoomedScene
5: (4)                               class UseZoomedScene(ZoomedScene):
6: (8)                               def construct(self):
7: (12)                                 dot = Dot().set_color(GREEN)
8: (12)                                 self.add(dot)
9: (12)                                 self.wait(1)
10: (12)                                self.activate_zooming/animate=False)
11: (12)                                self.wait(1)
12: (12)                                self.play(dot.animate.shift(LEFT))
13: (0)                               .. manim:: ChangingZoomScale
14: (4)                               class ChangingZoomScale(ZoomedScene):
15: (8)                               def __init__(self, **kwargs):
16: (12)                                 ZoomedScene.__init__(
17: (16)                                   self,
18: (16)                                   zoom_factor=0.3,
19: (16)                                   zoomed_display_height=1,
20: (16)                                   zoomed_display_width=3,
21: (16)                                   image_frame_stroke_width=20,
22: (16)                                   zoomed_camera_config={
23: (20)                                     "default_frame_stroke_width": 3,
24: (16)                                   },
25: (16)                                   **kwargs
26: (12)                               )
27: (8)                               def construct(self):
28: (12)                                 dot = Dot().set_color(GREEN)
29: (12)                                 sq = Circle(fill_opacity=1, radius=0.2).next_to(dot, RIGHT)
30: (12)                                 self.add(dot, sq)
31: (12)                                 self.wait(1)
32: (12)                                self.activate_zooming/animate=False)
33: (12)                                self.wait(1)
34: (12)                                self.play(dot.animate.shift(LEFT * 0.3))
35: (12)                                self.play(self.zoomed_camera.frame.animate.scale(4))
36: (12)                                self.play(self.zoomed_camera.frame.animate.shift(0.5 * DOWN))
37: (0)                               """
38: (0)                               from __future__ import annotations
39: (0)                               __all__ = ["ZoomedScene"]
40: (0)                               from ..animation.transform import ApplyMethod
41: (0)                               from ..camera.moving_camera import MovingCamera
42: (0)                               from ..camera.multi_camera import MultiCamera
43: (0)                               from ..constants import *
44: (0)                               from ..mobject.types.image_mobject import ImageMobjectFromCamera
45: (0)                               from ..scene.moving_camera_scene import MovingCameraScene
46: (0)                               class ZoomedScene(MovingCameraScene):
47: (4)                               """
```

```

48: (4)             This is a Scene with special configurations made for when
49: (4)             a particular part of the scene must be zoomed in on and displayed
50: (4)             separately.
51: (4)
52: (4)             def __init__(
53: (8)                 self,
54: (8)                 camera_class=MultiCamera,
55: (8)                 zoomed_display_height=3,
56: (8)                 zoomed_display_width=3,
57: (8)                 zoomed_display_center=None,
58: (8)                 zoomed_display_corner=UP + RIGHT,
59: (8)                 zoomed_display_corner_buff=DEFAULT_MOBJECT_TO_EDGE_BUFFER,
60: (8)                 zoomed_camera_config={
61: (12)                     "default_frame_stroke_width": 2,
62: (12)                     "background_opacity": 1,
63: (8)                 },
64: (8)                 zoomed_camera_image_mobject_config={},
65: (8)                 zoomed_camera_frame_starting_position=ORIGIN,
66: (8)                 zoom_factor=0.15,
67: (8)                 image_frame_stroke_width=3,
68: (8)                 zoom_activated=False,
69: (8)                 **kwargs,
70: (4):
71: (8)             self.zoomed_display_height = zoomed_display_height
72: (8)             self.zoomed_display_width = zoomed_display_width
73: (8)             self.zoomed_display_center = zoomed_display_center
74: (8)             self.zoomed_display_corner = zoomed_display_corner
75: (8)             self.zoomed_display_corner_buff = zoomed_display_corner_buff
76: (8)             self.zoomed_camera_config = zoomed_camera_config
77: (8)             self.zoomed_camera_image_mobject_config =
zoomed_camera_image_mobject_config
78: (8)             self.zoomed_camera_frame_starting_position = (
79: (12)                 zoomed_camera_frame_starting_position
80: (8)             )
81: (8)             self.zoom_factor = zoom_factor
82: (8)             self.image_frame_stroke_width = image_frame_stroke_width
83: (8)             self.zoom_activated = zoom_activated
84: (8)             super().__init__(camera_class=camera_class, **kwargs)
85: (4)             def setup(self):
86: (8)                 """
87: (8)                 This method is used internally by Manim to
88: (8)                 setup the scene for proper use.
89: (8)                 """
90: (8)                 super().setup()
91: (8)                 zoomed_camera = MovingCamera(**self.zoomed_camera_config)
92: (8)                 zoomed_display = ImageMobjectFromCamera(
93: (12)                     zoomed_camera, **self.zoomed_camera_image_mobject_config
94: (8)                 )
95: (8)                 zoomed_display.add_display_frame()
96: (8)                 for mob in zoomed_camera.frame, zoomed_display:
97: (12)                     mob.stretch_to_fit_height(self.zoomed_display_height)
98: (12)                     mob.stretch_to_fit_width(self.zoomed_display_width)
99: (8)                 zoomed_camera.frame.scale(self.zoom_factor)
100: (8)
zoomed_camera.frame.move_to(self.zoomed_camera_frame_starting_position)
101: (8)                 if self.zoomed_display_center is not None:
102: (12)                     zoomed_display.move_to(self.zoomed_display_center)
103: (8)
104: (12)                 else:
105: (16)                     zoomed_display.to_corner(
106: (16)                         self.zoomed_display_corner,
107: (12)                         buff=self.zoomed_display_corner_buff,
108: (8)                     )
109: (8)                     self.zoomed_camera = zoomed_camera
110: (4)                     self.zoomed_display = zoomed_display
def activate_zooming(self, animate: bool = False):
    """
    This method is used to activate the zooming for
    the zoomed_camera.
    Parameters

```

```

115: (8)           -----
116: (8)           animate
117: (12)          Whether or not to animate the activation
118: (12)          of the zoomed camera.
119: (8)           """
120: (8)           self.zoom_activated = True
121: (8)
self.renderer.camera.add_image_mobject_from_camera(self.zoomed_display)
122: (8)           if animate:
123: (12)          self.play(self.get_zoom_in_animation())
124: (12)          self.play(self.get_zoomed_display_pop_out_animation())
125: (8)           self.add_foreground_mobjects(
126: (12)             self.zoomed_camera.frame,
127: (12)             self.zoomed_display,
128: (8)           )
def get_zoom_in_animation(self, run_time: float = 2, **kwargs):
130: (8)           """
131: (8)           Returns the animation of camera zooming in.
132: (8)           Parameters
133: (8)           -----
134: (8)           run_time
135: (12)          The run_time of the animation of the camera zooming in.
136: (8)           **kwargs
137: (12)          Any valid keyword arguments of ApplyMethod()
138: (8)           Returns
139: (8)           -----
140: (8)           ApplyMethod
141: (12)          The animation of the camera zooming in.
142: (8)           """
143: (8)           frame = self.zoomed_camera.frame
144: (8)           full_frame_height = self.camera.frame_height
145: (8)           full_frame_width = self.camera.frame_width
146: (8)           frame.save_state()
147: (8)           frame.stretch_to_fit_width(full_frame_width)
148: (8)           frame.stretch_to_fit_height(full_frame_height)
149: (8)           frame.center()
150: (8)           frame.set_stroke(width=0)
151: (8)           return ApplyMethod(frame.restore, run_time=run_time, **kwargs)
152: (4)           def get_zoomed_display_pop_out_animation(self, **kwargs):
153: (8)           """
154: (8)           This is the animation of the popping out of the
155: (8)           mini-display that shows the content of the zoomed
156: (8)           camera.
157: (8)           Returns
158: (8)           -----
159: (8)           ApplyMethod
160: (12)          The Animation of the Zoomed Display popping out.
161: (8)           """
162: (8)           display = self.zoomed_display
163: (8)           display.save_state()
164: (8)           display.replace(self.zoomed_camera.frame, stretch=True)
165: (8)           return ApplyMethod(display.restore)
166: (4)           def get_zoom_factor(self):
167: (8)           """
168: (8)           Returns the Zoom factor of the Zoomed camera.
169: (8)           Defined as the ratio between the height of the
170: (8)           zoomed camera and the height of the zoomed mini
171: (8)           display.
172: (8)           Returns
173: (8)           -----
174: (8)           float
175: (12)          The zoom factor.
176: (8)           """
177: (8)           return self.zoomed_camera.frame.height / self.zoomed_display.height
-----
```

File 131 - manim\_colors.py:

```

1: (0)          """Colors included in the global name space.
2: (0)          These colors form Manim's default color space.
3: (0)          .. manim:: ColorsOverview
4: (4)          :save_last_frame:
5: (4)          :hide_source:
6: (4)          import manim.utils.color.manim_colors as Colors
7: (4)          class ColorsOverview(Scene):
8: (8)          def construct(self):
9: (12)          def color_group(color):
10: (16)          group = VGroup(
11: (20)          *[[
12: (24)                  Line(ORIGIN, RIGHT * 1.5, stroke_width=35,
color=getattr(Colors, name.upper())))
13: (24)                  for name in subnames(color)
14: (20)              ]
15: (16)          ).arrange_submobjects(buff=0.4, direction=DOWN)
16: (16)          name = Text(color).scale(0.6).next_to(group, UP, buff=0.3)
17: (16)          if any(decender in color for decender in "gjpqy"):
18: (20)          name.shift(DOWN * 0.08)
19: (16)          group.add(name)
20: (16)          return group
21: (12)          def subnames(name):
22: (16)          return [name + "_" + char for char in "abcde"]
23: (12)          color_groups = VGroup(
24: (16)          *[[
25: (20)                  color_group(color)
26: (20)                  for color in [
27: (24)                      "blue",
28: (24)                      "teal",
29: (24)                      "green",
30: (24)                      "yellow",
31: (24)                      "gold",
32: (24)                      "red",
33: (24)                      "maroon",
34: (24)                      "purple",
35: (20)                  ]
36: (16)              ]
37: (12)          ).arrange_submobjects(buff=0.2, aligned_edge=DOWN)
38: (12)          for line, char in zip(color_groups[0], "abcde"):
39: (16)              color_groups.add(Text(char).scale(0.6).next_to(line, LEFT,
buff=0.2))
40: (12)          def named_lines_group(length, colors, names, text_colors,
align_to_block):
41: (16)              lines = VGroup(
42: (20)              *[[
43: (24)                  Line(
44: (28)                      ORIGIN,
45: (28)                      RIGHT * length,
46: (28)                      stroke_width=55,
47: (28)                      color=getattr(Colors, color.upper())),
48: (24)                  )
49: (24)                  for color in colors
50: (20)              ]
51: (16)          ).arrange_submobjects(buff=0.6, direction=DOWN)
52: (16)          for line, name, color in zip(lines, names, text_colors):
53: (20)              line.add(Text(name, color=color).scale(0.6).move_to(line))
54: (16)              lines.next_to(color_groups, DOWN, buff=0.5).align_to(
55: (20)                  color_groups[align_to_block], LEFT
56: (16)              )
57: (16)              return lines
58: (12)          other_colors = (
59: (16)              "pink",
60: (16)              "light_pink",
61: (16)              "orange",
62: (16)              "light_brown",
63: (16)              "dark_brown",
64: (16)              "gray_brown",
65: (12)          )
66: (12)          other_lines = named_lines_group(

```

```

67: (16)                      3.2,
68: (16)                      other_colors,
69: (16)                      other_colors,
70: (16)                      [BLACK] * 4 + [WHITE] * 2,
71: (16)                      0,
72: (12) )
73: (12)     gray_lines = named_lines_group(
74: (16)         6.6,
75: (16)         ["white"] + subnames("gray") + ["black"],
76: (16)         [
77: (20)             "white",
78: (20)             "lighter_gray / gray_a",
79: (20)             "light_gray / gray_b",
80: (20)             "gray / gray_c",
81: (20)             "dark_gray / gray_d",
82: (20)             "darker_gray / gray_e",
83: (20)             "black",
84: (16)         ],
85: (16)         [BLACK] * 3 + [WHITE] * 4,
86: (16)         2,
87: (12)     )
88: (12)     pure_colors = (
89: (16)         "pure_red",
90: (16)         "pure_green",
91: (16)         "pure_blue",
92: (12)     )
93: (12)     pure_lines = named_lines_group(
94: (16)         3.2,
95: (16)         pure_colors,
96: (16)         pure_colors,
97: (16)         [BLACK, BLACK, WHITE],
98: (16)         6,
99: (12)     )
100: (12)    self.add(color_groups, other_lines, gray_lines, pure_lines)
101: (12)    VGroup(*self.mobjects).move_to(ORIGIN)
102: (0) ... automanimcolormodule:: manim.utils.color.manim_colors
103: (0)
104: (0) """
105: (0)     from typing import List
106: (0)     from .core import ManimColor
107: (0)     WHITE = ManimColor("#FFFFFF")
108: (0)     GRAY_A = ManimColor("#DDDDDD")
109: (0)     GREY_A = ManimColor("#DDDDDD")
110: (0)     GRAY_B = ManimColor("#BBBBBB")
111: (0)     GREY_B = ManimColor("#BBBBBB")
112: (0)     GRAY_C = ManimColor("#888888")
113: (0)     GREY_C = ManimColor("#888888")
114: (0)     GRAY_D = ManimColor("#444444")
115: (0)     GREY_D = ManimColor("#444444")
116: (0)     GRAY_E = ManimColor("#222222")
117: (0)     GREY_E = ManimColor("#222222")
118: (0)     BLACK = ManimColor("#000000")
119: (0)     LIGHTER_GRAY = ManimColor("#DDDDDD")
120: (0)     LIGHTER_GREY = ManimColor("#DDDDDD")
121: (0)     LIGHT_GRAY = ManimColor("#BBBBBB")
122: (0)     LIGHT_GREY = ManimColor("#BBBBBB")
123: (0)     GRAY = ManimColor("#888888")
124: (0)     GREY = ManimColor("#888888")
125: (0)     DARK_GRAY = ManimColor("#444444")
126: (0)     DARK_GREY = ManimColor("#444444")
127: (0)     DARKER_GRAY = ManimColor("#222222")
128: (0)     DARKER_GREY = ManimColor("#222222")
129: (0)     BLUE_A = ManimColor("#C7E9F1")
130: (0)     BLUE_B = ManimColor("#9CDCEB")
131: (0)     BLUE_C = ManimColor("#58C4DD")
132: (0)     BLUE_D = ManimColor("#29ABCA")
133: (0)     BLUE_E = ManimColor("#236B8E")
134: (0)     PURE_BLUE = ManimColor("#0000FF")
135: (0)     BLUE = ManimColor("#58C4DD")
136: (0)     DARK_BLUE = ManimColor("#236B8E")

```

```

136: (0)           TEAL_A = ManimColor("#ACEAD7")
137: (0)           TEAL_B = ManimColor("#76DDC0")
138: (0)           TEAL_C = ManimColor("#5CD0B3")
139: (0)           TEAL_D = ManimColor("#55C1A7")
140: (0)           TEAL_E = ManimColor("#49A88F")
141: (0)           TEAL = ManimColor("#5CD0B3")
142: (0)           GREEN_A = ManimColor("#C9E2AE")
143: (0)           GREEN_B = ManimColor("#A6CF8C")
144: (0)           GREEN_C = ManimColor("#83C167")
145: (0)           GREEN_D = ManimColor("#77B05D")
146: (0)           GREEN_E = ManimColor("#699C52")
147: (0)           PURE_GREEN = ManimColor("#00FF00")
148: (0)           GREEN = ManimColor("#83C167")
149: (0)           YELLOW_A = ManimColor("#FFF1B6")
150: (0)           YELLOW_B = ManimColor("#FFEA94")
151: (0)           YELLOW_C = ManimColor("#FFFF00")
152: (0)           YELLOW_D = ManimColor("#F4D345")
153: (0)           YELLOW_E = ManimColor("#E8C11C")
154: (0)           YELLOW = ManimColor("#FFFF00")
155: (0)           GOLD_A = ManimColor("#F7C797")
156: (0)           GOLD_B = ManimColor("#F9B775")
157: (0)           GOLD_C = ManimColor("#F0AC5F")
158: (0)           GOLD_D = ManimColor("#E1A158")
159: (0)           GOLD_E = ManimColor("#C78D46")
160: (0)           GOLD = ManimColor("#F0AC5F")
161: (0)           RED_A = ManimColor("#F7A1A3")
162: (0)           RED_B = ManimColor("#FF8080")
163: (0)           RED_C = ManimColor("#FC6255")
164: (0)           RED_D = ManimColor("#E65A4C")
165: (0)           RED_E = ManimColor("#CF5044")
166: (0)           PURE_RED = ManimColor("#FF0000")
167: (0)           RED = ManimColor("#FC6255")
168: (0)           MAROON_A = ManimColor("#ECABC1")
169: (0)           MAROON_B = ManimColor("#EC92AB")
170: (0)           MAROON_C = ManimColor("#C55F73")
171: (0)           MAROON_D = ManimColor("#A24D61")
172: (0)           MAROON_E = ManimColor("#94424F")
173: (0)           MAROON = ManimColor("#C55F73")
174: (0)           PURPLE_A = ManimColor("#CAA3E8")
175: (0)           PURPLE_B = ManimColor("#B189C6")
176: (0)           PURPLE_C = ManimColor("#9A72AC")
177: (0)           PURPLE_D = ManimColor("#715582")
178: (0)           PURPLE_E = ManimColor("#644172")
179: (0)           PURPLE = ManimColor("#9A72AC")
180: (0)           PINK = ManimColor("#D147BD")
181: (0)           LIGHT_PINK = ManimColor("#DC75CD")
182: (0)           ORANGE = ManimColor("#FF862F")
183: (0)           LIGHT_BROWN = ManimColor("#CD853F")
184: (0)           DARK_BROWN = ManimColor("#8B4513")
185: (0)           GRAY_BROWN = ManimColor("#736357")
186: (0)           GREY_BROWN = ManimColor("#736357")
187: (0)           LOGO_WHITE = ManimColor("#ECE7E2")
188: (0)           LOGO_GREEN = ManimColor("#87C2A5")
189: (0)           LOGO_BLUE = ManimColor("#525893")
190: (0)           LOGO_RED = ManimColor("#E07A5F")
191: (0)           LOGO_BLACK = ManimColor("#343434")
192: (0)           _all_manim_colors: List[ManimColor] = [
193: (4)             x for x in globals().values() if isinstance(x, ManimColor)
194: (0)           ]

```

-----

File 132 - three\_d\_scene.py:

```

1: (0)           """A scene suitable for rendering three-dimensional objects and animations."""
2: (0)           from __future__ import annotations
3: (0)           __all__ = ["ThreeDScene", "SpecialThreeDScene"]
4: (0)           import warnings
5: (0)           from typing import Iterable, Sequence

```

```

6: (0)             import numpy as np
7: (0)             from manim.mobject.geometry.line import Line
8: (0)             from manim.mobject.graphing.coordinate_systems import ThreeDAxes
9: (0)             from manim.mobject.opengl.opengl_mobject import OpenGLMobject
10: (0)            from manim.mobject.three_d.three_dimensions import Sphere
11: (0)            from manim.mobject.value_tracker import ValueTracker
12: (0)            from .. import config
13: (0)            from ..animation.animation import Animation
14: (0)            from ..animation.transform import Transform
15: (0)            from ..camera.three_d_camera import ThreeDCamera
16: (0)            from ..constants import DEGREES, RendererType
17: (0)            from ..mobject.mobject import Mobject
18: (0)            from ..mobject.types.vectorized_mobject import VectorizedPoint, VGroup
19: (0)            from ..renderer.opengl_renderer import OpenGLCamera
20: (0)            from ..scene.scene import Scene
21: (0)            from ..utils.config_ops import merge_dicts_recursively
22: (0)        class ThreeDScene(Scene):
23: (4)            """
24: (4)                This is a Scene, with special configurations and properties that
25: (4)                make it suitable for Three Dimensional Scenes.
26: (4)            """
27: (4)        def __init__(
28: (8)            self,
29: (8)            camera_class=ThreeDCamera,
30: (8)            ambient_camera_rotation=None,
31: (8)            default_angled_camera_orientation_kwargs=None,
32: (8)            **kwargs,
33: (4)        ):
34: (8)            self.ambient_camera_rotation = ambient_camera_rotation
35: (8)            if default_angled_camera_orientation_kwargs is None:
36: (12)                default_angled_camera_orientation_kwargs = {
37: (16)                    "phi": 70 * DEGREES,
38: (16)                    "theta": -135 * DEGREES,
39: (12)                }
40: (8)            self.default_angled_camera_orientation_kwargs = (
41: (12)                default_angled_camera_orientation_kwargs
42: (8)            )
43: (8)            super().__init__(camera_class=camera_class, **kwargs)
44: (4)        def set_camera_orientation(
45: (8)            self,
46: (8)            phi: float | None = None,
47: (8)            theta: float | None = None,
48: (8)            gamma: float | None = None,
49: (8)            zoom: float | None = None,
50: (8)            focal_distance: float | None = None,
51: (8)            frame_center: Mobject | Sequence[float] | None = None,
52: (8)            **kwargs,
53: (4)        ):
54: (8)            """
55: (8)                This method sets the orientation of the camera in the scene.
56: (8)                Parameters
57: (8)                -----
58: (8)                phi
59: (12)                    The polar angle i.e the angle between Z_AXIS and Camera through
ORIGIN in radians.
60: (8)
61: (12)                    theta
62: (8)                        The azimuthal angle i.e the angle that spins the camera around the
Z_AXIS.
63: (12)                    focal_distance
64: (8)                        The focal_distance of the Camera.
65: (12)                    gamma
66: (8)                        The rotation of the camera about the vector from the ORIGIN to the
67: (12)                    zoom
68: (8)                        The zoom factor of the scene.
69: (12)                    frame_center
70: (8)                        The new center of the camera frame in cartesian coordinates.
71: (8)            if phi is not None:

```

```

72: (12)                     self.renderer.camera.set_phi(phi)
73: (8)                      if theta is not None:
74: (12)                        self.renderer.camera.set_theta(theta)
75: (8)                      if focal_distance is not None:
76: (12)                        self.renderer.camera.set_focal_distance(focal_distance)
77: (8)                      if gamma is not None:
78: (12)                        self.renderer.camera.set_gamma(gamma)
79: (8)                      if zoom is not None:
80: (12)                        self.renderer.camera.set_zoom(zoom)
81: (8)                      if frame_center is not None:
82: (12)                        self.renderer.camera._frame_center.move_to(frame_center)
83: (4)  def begin_ambient_camera_rotation(self, rate: float = 0.02, about: str =
  "theta"):
84: (8)    """
85: (8)      This method begins an ambient rotation of the camera about the Z_AXIS,
86: (8)      in the anticlockwise direction
87: (8)      Parameters
88: (8)      -----
89: (8)      rate
90: (12)        The rate at which the camera should rotate about the Z_AXIS.
91: (12)        Negative rate means clockwise rotation.
92: (8)      about
93: (12)        one of 3 options: ["theta", "phi", "gamma"]. defaults to theta.
  """
94: (8)
95: (8)      about: str = about.lower()
96: (8)    try:
97: (12)      if config.renderer == RendererType.CAIRO:
98: (16)        trackers = {
99: (20)          "theta": self.camera.theta_tracker,
100: (20)          "phi": self.camera.phi_tracker,
101: (20)          "gamma": self.camera.gamma_tracker,
102: (16)
103: (16)
104: (16)
105: (16)
106: (12)
107: (16)
108: (16)
109: (20)
110: (20)
111: (20)
112: (16)
113: (16)
114: (16)
115: (8)
116: (12)
117: (4)    raise ValueError("Invalid ambient rotation angle.")
118: (8)  def stop_ambient_camera_rotation(self, about="theta"):
  """
119: (8)      This method stops all ambient camera rotation.
  """
120: (8)
121: (8)
122: (8)
123: (12)    try:
124: (16)      if config.renderer == RendererType.CAIRO:
125: (20)        trackers = {
126: (20)          "theta": self.camera.theta_tracker,
127: (20)          "phi": self.camera.phi_tracker,
128: (20)          "gamma": self.camera.gamma_tracker,
129: (16)
130: (16)
131: (16)
132: (12)
133: (16)
134: (8)
135: (12)
136: (4)    except Exception:
137: (8)      raise ValueError("Invalid ambient rotation angle.")
138: (8)
139: (8)  def begin_3dillusion_camera_rotation(
  self,
  rate: float = 1,
  origin_phi: float | None = None,

```

```

140: (8)                 origin_theta: float | None = None,
141: (4)                 ):
142: (8)                 """
143: (8)                 This method creates a 3D camera rotation illusion around
144: (8)                 the current camera orientation.
145: (8)                 Parameters
146: (8)                 -----
147: (8)                 rate
148: (12)                 The rate at which the camera rotation illusion should operate.
149: (8)                 origin_phi
150: (12)                 The polar angle the camera should move around. Defaults
151: (12)                 to the current phi angle.
152: (8)                 origin_theta
153: (12)                 The azimuthal angle the camera should move around. Defaults
154: (12)                 to the current theta angle.
155: (8)                 """
156: (8)                 if origin_theta is None:
157: (12)                     origin_theta = self.renderer.camera.theta_tracker.get_value()
158: (8)                 if origin_phi is None:
159: (12)                     origin_phi = self.renderer.camera.phi_tracker.get_value()
160: (8)                 val_tracker_theta = ValueTracker(0)
161: (8)                 def update_theta(m, dt):
162: (12)                     val_tracker_theta.increment_value(dt * rate)
163: (12)                     val_for_left_right = 0.2 * np.sin(val_tracker_theta.get_value())
164: (12)                     return m.set_value(origin_theta + val_for_left_right)
165: (8)                 self.renderer.camera.theta_tracker.add_updater(update_theta)
166: (8)                 self.add(self.renderer.camera.theta_tracker)
167: (8)                 val_tracker_phi = ValueTracker(0)
168: (8)                 def update_phi(m, dt):
169: (12)                     val_tracker_phi.increment_value(dt * rate)
170: (12)                     val_for_up_down = 0.1 * np.cos(val_tracker_phi.get_value()) - 0.1
171: (12)                     return m.set_value(origin_phi + val_for_up_down)
172: (8)                 self.renderer.camera.phi_tracker.add_updater(update_phi)
173: (8)                 self.add(self.renderer.camera.phi_tracker)
174: (4)                 def stop_3dillusion_camera_rotation(self):
175: (8)                 """
176: (8)                 This method stops all illusion camera rotations.
177: (8)                 """
178: (8)                 self.renderer.camera.theta_tracker.clear_updaters()
179: (8)                 self.remove(self.renderer.camera.theta_tracker)
180: (8)                 self.renderer.camera.phi_tracker.clear_updaters()
181: (8)                 self.remove(self.renderer.camera.phi_tracker)
182: (4)                 def move_camera(
183: (8)                     self,
184: (8)                     phi: float | None = None,
185: (8)                     theta: float | None = None,
186: (8)                     gamma: float | None = None,
187: (8)                     zoom: float | None = None,
188: (8)                     focal_distance: float | None = None,
189: (8)                     frame_center: Mobject | Sequence[float] | None = None,
190: (8)                     added_anims: Iterable[Animation] = [],
191: (8)                     **kwargs,
192: (4)                 ):
193: (8)                 """
194: (8)                 This method animates the movement of the camera
195: (8)                 to the given spherical coordinates.
196: (8)                 Parameters
197: (8)                 -----
198: (8)                 phi
199: (12)                 The polar angle i.e the angle between Z_AXIS and Camera through
ORIGIN in radians.
200: (8)                 theta
201: (12)                 The azimuthal angle i.e the angle that spins the camera around the
Z_AXIS.
202: (8)                 focal_distance
203: (12)                 The radial focal_distance between ORIGIN and Camera.
204: (8)                 gamma
205: (12)                 The rotation of the camera about the vector from the ORIGIN to the
Camera.

```

```

206: (8)           zoom
207: (12)          The zoom factor of the camera.
208: (8)
209: (12)          frame_center
210: (8)          The new center of the camera frame in cartesian coordinates.
211: (12)          added_anims
212: (8)          Any other animations to be played at the same time.
213: (8)
214: (8)          """
215: (12)          anims = []
216: (12)          if config.renderer == RendererType.CAIRO:
217: (16)            self.camera: ThreeDCamera
218: (16)            value_tracker_pairs = [
219: (16)              (phi, self.camera.phi_tracker),
220: (16)              (theta, self.camera.theta_tracker),
221: (16)              (focal_distance, self.camera.focal_distance_tracker),
222: (16)              (gamma, self.camera.gamma_tracker),
223: (16)              (zoom, self.camera.zoom_tracker),
224: (12)            ]
225: (20)            for value, tracker in value_tracker_pairs:
226: (12)              if value is not None:
227: (16)                anims.append(tracker.animate.set_value(value))
228: (12)            if frame_center is not None:
229: (12)              anims.append(self.camera._frame_center.animate.move_to(frame_center))
230: (12)            elif config.renderer == RendererType.OPENGL:
231: (12)              cam: OpenGLCamera = self.camera
232: (16)              cam2 = cam.copy()
233: (16)              methods = {
234: (16)                "theta": cam2.set_theta,
235: (16)                "phi": cam2.set_phi,
236: (16)                "gamma": cam2.set_gamma,
237: (16)                "zoom": cam2.scale,
238: (12)                "frame_center": cam2.move_to,
239: (12)              }
240: (20)              if frame_center is not None:
241: (16)                if isinstance(frame_center, OpenGLObject):
242: (12)                  frame_center = frame_center.get_center()
243: (12)                frame_center = list(frame_center)
244: (16)                zoom_value = None
245: (12)                if zoom is not None:
246: (16)                  zoom_value = config.frame_height / (zoom * cam.height)
247: (16)                for value, method in [
248: (16)                  [theta, "theta"],
249: (16)                  [phi, "phi"],
250: (16)                  [gamma, "gamma"],
251: (12)                  [zoom_value, "zoom"],
252: (16)                  [frame_center, "frame_center"],
253: (20)                ]:
254: (12)                  if value is not None:
255: (16)                    methods[method](value)
256: (20)                if focal_distance is not None:
257: (20)                  warnings.warn(
258: (16)                    "focal distance of OpenGLCamera can not be adjusted.",
259: (12)                    stacklevel=2,
260: (8)                  )
261: (8)                  anims += [Transform(cam, cam2)]
262: (12)                self.play(*anims + added_anims, **kwargs)
263: (4)                if frame_center is not None and config.renderer == RendererType.CAIRO:
264: (8)                  self.remove(self.camera._frame_center)
265: (8)              def get_moving_mobjects(self, *animations: Animation):
266: (8)                """
267: (8)                  This method returns a list of all of the Mobjects in the Scene that
268: (8)                  are moving, that are also in the animations passed.
269: (8)                  Parameters
270: (12)                  -----
271: (8)                  *animations
272: (8)                  The animations whose mobjects will be checked.
273: (8)                """
274: (8)                moving_mobjects = super().get_moving_mobjects(*animations)
275: (8)                camera_mobjects = self.renderer.camera.get_value_trackers() + [

```

```

274: (12)                     self.renderer.camera._frame_center,
275: (8)                 ]
276: (8)             if any(cm in moving_mobjects for cm in camera_mobjects):
277: (12)                 return self.mobjects
278: (8)             return moving_mobjects
279: (4)         def add_fixed_orientation_mobjects(self, *mobjects: Mobject, **kwargs):
280: (8)             """
281: (8)             This method is used to prevent the rotation and tilting
282: (8)             of mobjects as the camera moves around. The mobject can
283: (8)             still move in the x,y,z directions, but will always be
284: (8)             at the angle (relative to the camera) that it was at
285: (8)             when it was passed through this method.)
286: (8)             Parameters
287: (8)             -----
288: (8)             *mobjects
289: (12)                 The Mobject(s) whose orientation must be fixed.
290: (8)             **kwargs
291: (12)                 Some valid kwargs are
292: (16)                     use_static_center_func : bool
293: (16)                     center_func : function
294: (8)                 """
295: (8)             if config.renderer == RendererType.CAIRO:
296: (12)                 self.add(*mobjects)
297: (12)                 self.renderer.camera.add_fixed_orientation_mobjects(*mobjects,
298: (**kwargs)
299: (8)
300: (12)
301: (16)
302: (16)
303: (4)         def add_fixed_in_frame_mobjects(self, *mobjects: Mobject):
304: (8)             """
305: (8)             This method is used to prevent the rotation and movement
306: (8)             of mobjects as the camera moves around. The mobject is
307: (8)             essentially overlaid, and is not impacted by the camera's
308: (8)             movement in any way.
309: (8)             Parameters
310: (8)             -----
311: (8)             *mobjects
312: (12)                 The Mobjects whose orientation must be fixed.
313: (8)
314: (8)             if config.renderer == RendererType.CAIRO:
315: (12)                 self.add(*mobjects)
316: (12)                 self.camera: ThreeDCamera
317: (12)                 self.camera.add_fixed_in_frame_mobjects(*mobjects)
318: (8)             elif config.renderer == RendererType.OPENGL:
319: (12)                 for mob in mobjects:
320: (16)                     mob: OpenGLMobject
321: (16)                     mob.fix_in_frame()
322: (16)                     self.add(mob)
323: (4)         def remove_fixed_orientation_mobjects(self, *mobjects: Mobject):
324: (8)             """
325: (8)             This method "unfixes" the orientation of the mobjects
326: (8)             passed, meaning they will no longer be at the same angle
327: (8)             relative to the camera. This only makes sense if the
328: (8)             mobject was passed through add_fixed_orientation_mobjects first.
329: (8)             Parameters
330: (8)             -----
331: (8)             *mobjects
332: (12)                 The Mobjects whose orientation must be unfixed.
333: (8)
334: (8)             if config.renderer == RendererType.CAIRO:
335: (12)                 self.renderer.camera.remove_fixed_orientation_mobjects(*mobjects)
336: (8)
337: (12)             elif config.renderer == RendererType.OPENGL:
338: (16)                 for mob in mobjects:
339: (16)                     mob: OpenGLMobject
340: (16)                     mob.unfix_orientation()
341: (4)                     self.remove(mob)
342: (4)         def remove_fixed_in_frame_mobjects(self, *mobjects: Mobject):

```

```

342: (8)             """
343: (9)             This method undoes what add_fixed_in_frame_mobjects does.
344: (9)             It allows the mobject to be affected by the movement of
345: (9)             the camera.
346: (8)             Parameters
347: (8)             -----
348: (8)             *mobjects
349: (12)            The Mobjects whose position and orientation must be unfixed.
350: (8)
351: (8)
352: (12)
353: (8)
354: (12)
355: (16)
356: (16)
357: (16)
358: (4)             if config.renderer == RendererType.CAIRO:
359: (8)                 self.renderer.camera.remove_fixed_in_frame_mobjects(*mobjects)
360: (8)             elif config.renderer == RendererType.OPENGL:
361: (8)                 for mob in mobjects:
362: (8)                     mob: OpenGLMobject
363: (8)                     mob.unfix_from_frame()
364: (8)                     self.remove(mob)
365: (12)            def set_to_default_angled_camera_orientation(self, **kwargs):
366: (12)                """
367: (8)                    This method sets the default_angled_camera_orientation to the
368: (8)                    keyword arguments passed, and sets the camera to that orientation.
369: (12)                    Parameters
370: (8)                    -----
371: (8)                    **kwargs
372: (8)                    Some recognised kwargs are phi, theta, focal_distance, gamma,
373: (0)                      which have the same meaning as the parameters in
374: (4)                      """
375: (4)                      config = dict(
376: (4)                          self.default_camera_orientation_kwargs,
377: (4)                      ) # Where doe this come from?
378: (4)                      config.update(kwargs)
379: (4)                      self.set_camera_orientation(**config)
380: (4)
381: (4)
382: (4)            class SpecialThreeDScene(ThreeDScene):
383: (8)                """
384: (8)                    An extension of :class:`ThreeDScene` with more settings.
385: (8)                    It has some extra configuration for axes, spheres,
386: (8)                    and an override for low quality rendering. Further key differences
387: (8)                    are:
388: (8)                    * The camera shades applicable 3DMobjects by default,
389: (8)                      except if rendering in low quality.
390: (8)                    * Some default params for Spheres and Axes have been added.
391: (8)
392: (8)
393: (12)            def __init__(
394: (8)                self,
395: (8)                cut_axes_at_radius=True,
396: (8)                camera_config={"should_apply_shading": True, "exponential_projection":
397: (8)                    True},
398: (8)
399: (8)
400: (8)                three_d_axes_config={
401: (12)                  "num_axis_pieces": 1,
402: (12)                  "axis_config": {
403: (16)                      "unit_size": 2,
404: (16)                      "tick_frequency": 1,
405: (16)                      "numbers_with_elongated_ticks": [0, 1, 2],
406: (16)                      "stroke_width": 2,
407: (12)                  },
408: (12)                  sphere_config={"radius": 2, "resolution": (24, 48)},
409: (12)                  default_angled_camera_position={
410: (16)                      "phi": 70 * DEGREES,
411: (16)                      "theta": -110 * DEGREES,
412: (12)                  },
413: (12)                  low_quality_config={
414: (16)                      "camera_config": {"should_apply_shading": False},
415: (16)                      "three_d_axes_config": {"num_axis_pieces": 1},
416: (16)                      "sphere_config": {"resolution": (12, 24)},
417: (12)                  },
418: (12)                  **kwargs,
419: (4)                ):
420: (8)                    self.cut_axes_at_radius = cut_axes_at_radius
421: (8)                    self.camera_config = camera_config

```

```

409: (8)                     self.three_d_axes_config = three_d_axes_config
410: (8)                     self.sphere_config = sphere_config
411: (8)                     self.default_angled_camera_position = default_angled_camera_position
412: (8)                     self.low_quality_config = low_quality_config
413: (8)                     if self.renderer.camera_config["pixel_width"] ==
config["pixel_width"]:
414: (12)                         _config = {}
415: (8)                     else:
416: (12)                         _config = self.low_quality_config
417: (8)                     _config = merge_dicts_recursively(_config, kwargs)
418: (8)                     super().__init__(**_config)
419: (4)                     def get_axes(self):
420: (8)                         """Return a set of 3D axes.
421: (8)                         Returns
422: (8)                         -----
423: (8)                         :class:`.ThreeDAxes`
424: (12)                         A set of 3D axes.
425: (8)
426: (8)                     axes = ThreeDAxes(**self.three_d_axes_config)
427: (8)                     for axis in axes:
428: (12)                         if self.cut_axes_at_radius:
429: (16)                             p0 = axis.get_start()
430: (16)                             p1 = axis.number_to_point(-1)
431: (16)                             p2 = axis.number_to_point(1)
432: (16)                             p3 = axis.get_end()
433: (16)                             new_pieces = VGroup(Line(p0, p1), Line(p1, p2), Line(p2, p3))
434: (16)                             for piece in new_pieces:
435: (20)                                 piece.shade_in_3d = True
436: (16)                                 new_pieces.match_style(axis.pieces)
437: (16)                                 axis.pieces.submobjects = new_pieces.submobjects
438: (12)                             for tick in axis.tick_marks:
439: (16)                                 tick.add(VectorizedPoint(1.5 * tick.get_center())))
440: (8)                         return axes
441: (4)                     def get_sphere(self, **kwargs):
442: (8)
443: (8)                         """"
444: (8)                         Returns a sphere with the passed keyword arguments as properties.
445: (8)                         Parameters
446: (8)                         -----
447: (12)                         **kwargs
448: (8)                             Any valid parameter of :class:`~.Sphere` or :class:`~.Surface`.
449: (8)                         Returns
450: (8)                         -----
451: (12)                         :class:`~.Sphere`
452: (8)                             The sphere object.
453: (8)
454: (8)                         config = merge_dicts_recursively(self.sphere_config, kwargs)
455: (4)                         return Sphere(**config)
456: (8)                     def get_default_camera_position(self):
457: (8)
458: (8)                         """"
459: (8)                         Returns the default_angled_camera position.
460: (8)                         Returns
461: (12)                         dict
462: (8)                             Dictionary of phi, theta, focal_distance, and gamma.
463: (8)
464: (4)                         return self.default_angled_camera_position
465: (8)                     def set_camera_to_default_position(self):
466: (8)
467: (8)                         Sets the camera to its default position.
468: (8)                         """
469: (8)                         self.set_camera_orientation(**self.default_angled_camera_position)

```

#### File 133 - scene\_file\_writer.py:

```

1: (0)                     """The interface between scenes and ffmpeg."""
2: (0)                     from __future__ import annotations
3: (0)                     __all__ = ["SceneFileWriter"]

```

```

4: (0)          import json
5: (0)          import os
6: (0)          import shutil
7: (0)          import subprocess
8: (0)          from pathlib import Path
9: (0)          from typing import TYPE_CHECKING, Any
10: (0)         import numpy as np
11: (0)         import srt
12: (0)         from PIL import Image
13: (0)         from pydub import AudioSegment
14: (0)         from manim import __version__
15: (0)         from .. import config, logger
16: (0)         from ..config.logger_utils import set_file_logger
17: (0)         from ..constants import RendererType
18: (0)         from ..utils.file_ops import (
19: (4)             add_extension_if_not_present,
20: (4)             add_version_before_extension,
21: (4)             ensure_executable,
22: (4)             guarantee_existence,
23: (4)             is_gif_format,
24: (4)             is_png_format,
25: (4)             is_webm_format,
26: (4)             modify_atime,
27: (4)             write_to_movie,
28: (0)
29: (0)         from ..utils.sounds import get_full_sound_file_path
30: (0)         from .section import DefaultSectionType, Section
31: (0)         if TYPE_CHECKING:
32: (4)             from manim.renderer.opengl_renderer import OpenGLRenderer
33: (0)         class SceneFileWriter:
34: (4)             """
35: (4)             SceneFileWriter is the object that actually writes the animations
36: (4)             played, into video files, using FFMPEG.
37: (4)             This is mostly for Manim's internal use. You will rarely, if ever,
38: (4)             have to use the methods for this class, unless tinkering with the very
39: (4)             fabric of Manim's reality.
40: (4)             Attributes
41: (4)             -----
42: (8)                 sections : list of :class:`.Section`
43: (12)                   used to segment scene
44: (8)                 sections_output_dir : :class:`pathlib.Path`
45: (12)                   where are section videos stored
46: (8)                 output_name : str
47: (12)                   name of movie without extension and basis for section video names
48: (4)             Some useful attributes are:
49: (8)                 "write_to_movie" (bool=False)
50: (12)                   Whether or not to write the animations into a video file.
51: (8)                 "movie_file_extension" (str=".mp4")
52: (12)                   The file-type extension of the outputted video.
53: (8)                 "partial_movie_files"
54: (12)                   List of all the partial-movie files.
55: (4)             """
56: (4)             force_output_as_scene_name = False
57: (4)             def __init__(self, renderer, scene_name, **kwargs):
58: (8)                 self.renderer = renderer
59: (8)                 self.init_output_directories(scene_name)
60: (8)                 self.init_audio()
61: (8)                 self.frame_count = 0
62: (8)                 self.partial_movie_files: list[str] = []
63: (8)                 self.subcaptions: list[srt.Subtitle] = []
64: (8)                 self.sections: list[Section] = []
65: (8)                 self.next_section(
66: (12)                     name="autocreated", type=DefaultSectionType.NORMAL,
skip_animations=False
67: (8)
68: (8)
69: (12)
70: (16)
video output.\n"

```

```

71: (16)                                     "For installing ffmpeg please consult
https://docs.manim.community/en/stable/installation.html\n"
72: (16)                                     "Make sure to either add ffmpeg to the PATH environment
variable\n"
73: (16)                                     "or set path to the ffmpeg executable under the ffmpeg header
in Manim's configuration."
74: (12) )
75: (4)     def init_output_directories(self, scene_name):
76: (8)         """Initialise output directories.
77: (8)         Notes
78: (8)         -----
79: (8)         The directories are read from ``config``, for example
80: (8)         ``config['media_dir']``. If the target directories don't already
81: (8)         exist, they will be created.
82: (8)         """
83: (8)         if config["dry_run"]: # in dry-run mode there is no output
84: (12)             return
85: (8)         if config["input_file"]:
86: (12)             module_name = config.get_dir("input_file").stem
87: (8)         else:
88: (12)             module_name = ""
89: (8)         if SceneFileWriter.force_output_as_scene_name:
90: (12)             self.output_name = Path(scene_name)
91: (8)         elif config["output_file"] and not config["write_all"]:
92: (12)             self.output_name = config.get_dir("output_file")
93: (8)         else:
94: (12)             self.output_name = Path(scene_name)
95: (8)         if config["media_dir"]:
96: (12)             image_dir = guarantee_existence(
97: (16)                 config.get_dir(
98: (20)                     "images_dir", module_name=module_name,
scene_name=scene_name
99: (16)
100: (12)
101: (12)
102: (16)
103: (12)
104: (8)
105: (12)
106: (16)
107: (20)
scene_name=scene_name
108: (16)
109: (12)
110: (12)
111: (16)
112: (12)
113: (12)
114: (12)
115: (16)
116: (20)
117: (24)
scene_name=scene_name
118: (20)
119: (16)
120: (12)
121: (16)
122: (20)
123: (16)
124: (16)
125: (20)
126: (24)
127: (20)
128: (16)
129: (12)
130: (16)
131: (20)
132: (20)
133: (20)

```

```

134: (16)                               ),
135: (12)                               )
136: (12)                         if config["log_to_file"]:
137: (16)                           log_dir = guarantee_existence(config.get_dir("log_dir"))
138: (16)                           set_file_logger(
139: (20)                             scene_name=scene_name, module_name=module_name,
log_dir=log_dir
140: (16)                               )
141: (4)                           def finish_last_section(self) -> None:
142: (8)                             """Delete current section if it is empty."""
143: (8)                             if len(self.sections) and self.sections[-1].is_empty():
144: (12)                               self.sections.pop()
145: (4)                           def next_section(self, name: str, type: str, skip_animations: bool) ->
None:
146: (8)                             """Create segmentation cut here."""
147: (8)                             self.finish_last_section()
148: (8)                             section_video: str | None = None
149: (8)                           if (
150: (12)                             not config.dry_run
151: (12)                             and write_to_movie()
152: (12)                             and config.save_sections
153: (12)                             and not skip_animations
154: (8)                           ):
155: (12)                             section_video = f"
{self.output_name}_{len(self.sections):04}_{name}{config.movie_file_extension}"
156: (8)                           self.sections.append(
157: (12)                             Section(
158: (16)                               type,
159: (16)                               section_video,
160: (16)                               name,
161: (16)                               skip_animations,
162: (12)                             ),
163: (8)                           )
164: (4)                           def add_partial_movie_file(self, hash_animation: str):
165: (8)                             """Adds a new partial movie file path to `scene.partial_movie_files` and current section from a hash.
166: (8)                               This method will compute the path from the hash. In addition to that it adds the new animation to the current section.
167: (8)                               Parameters
168: (8)                               -----
169: (8)                               hash_animation
170: (12)                                 Hash of the animation.
171: (8)                               """
172: (8)                           if not hasattr(self, "partial_movie_directory") or not
write_to_movie():
173: (12)                               return
174: (8)                           if hash_animation is None:
175: (12)                             self.partial_movie_files.append(None)
176: (12)                             self.sections[-1].partial_movie_files.append(None)
177: (8)                           else:
178: (12)                             new_partial_movie_file = str(
179: (16)                               self.partial_movie_directory
180: (16)                               / f"{hash_animation}{config['movie_file_extension']}"
181: (12)                             )
182: (12)                             self.partial_movie_files.append(new_partial_movie_file)
183: (12)
self.sections[-1].partial_movie_files.append(new_partial_movie_file)
184: (4)                           def get_resolution_directory(self):
185: (8)                             """Get the name of the resolution directory directly containing the video file.
186: (8)                               This method gets the name of the directory that immediately contains
the
188: (8)                               video file. This name is ``<height_in_pixels_of_video>p<frame_rate>``.
189: (8)                               For example, if you are rendering an 854x480 px animation at 15fps, the name of the directory that immediately contains the video, file will be ``480p15``.
190: (8)
191: (8)
192: (8)                               The file structure should look something like::
193: (12)                               MEDIA_DIR
194: (16)                               |-- Tex

```

```

195: (16)                                |--texts
196: (16)                                |--videos
197: (16)                                |--<name_of_fileContaining_scene>
198: (20)                                |--<height_in_pixels_of_video>p<frame_rate>
199: (24)                                |--<scene_name>.mp4
200: (8)                                Returns
201: (8)                                -----
202: (8)                                :class:`str`
203: (12)                               The name of the directory.
204: (8)
205: (8)                                pixel_height = config["pixel_height"]
206: (8)                                frame_rate = config["frame_rate"]
207: (8)                                return f"{pixel_height}p{frame_rate}"
208: (4)                                def init_audio(self):
209: (8)                                """
210: (8)                                Preps the writer for adding audio to the movie.
211: (8)
212: (8)                                self.includes_sound = False
213: (4)                                def create_audio_segment(self):
214: (8)
215: (8)                                Creates an empty, silent, Audio Segment.
216: (8)
217: (8)                                self.audio_segment = AudioSegment.silent()
218: (4)                                def add_audio_segment(
219: (8)                                self,
220: (8)                                new_segment: AudioSegment,
221: (8)                                time: float | None = None,
222: (8)                                gain_to_background: float | None = None,
223: (4)                                ):
224: (8)
225: (8)                                This method adds an audio segment from an
226: (8)                                AudioSegment type object and suitable parameters.
227: (8)                                Parameters
228: (8)
229: (8)                                new_segment
230: (12)                               The audio segment to add
231: (8)
232: (12)                               time
233: (12)                               the timestamp at which the
234: (8)                               sound should be added.
235: (12)                               gain_to_background
236: (8)                               The gain of the segment from the background.
237: (8)
238: (12)                               if not self.includes_sound:
239: (12)                                 self.includes_sound = True
240: (8)                                 self.create_audio_segment()
241: (8)                                 segment = self.audio_segment
242: (8)                                 curr_end = segment.duration_seconds
243: (12)                               if time is None:
244: (8)                                 time = curr_end
245: (12)                               if time < 0:
246: (8)                                 raise ValueError("Adding sound at timestamp < 0")
247: (8)                                 new_end = time + new_segment.duration_seconds
248: (8)                                 diff = new_end - curr_end
249: (12)                               if diff > 0:
250: (16)                                 segment = segment.append(
251: (16)                                   AudioSegment.silent(int(np.ceil(diff * 1000))),
252: (12)                                   crossfade=0,
253: (8)
254: (12)                                   self.audio_segment = segment.overlay(
255: (12)                                     new_segment,
256: (12)                                     position=int(1000 * time),
257: (8)                                     gain_during_overlay=gain_to_background,
258: (4)                                   )
259: (8)
260: (8)                                def add_sound(
261: (8)                                self,
262: (8)                                sound_file: str,
263: (8)                                time: float | None = None,
264: (8)                                gain: float | None = None,
265: (8)                                **kwargs,

```

```

264: (4)                               ):
265: (8)                               """
266: (8)                               This method adds an audio segment from a sound file.
267: (8)                               Parameters
268: (8)                               -----
269: (8)                               sound_file
270: (12)                             The path to the sound file.
271: (8)                               time
272: (12)                             The timestamp at which the audio should be added.
273: (8)                               gain
274: (12)                             The gain of the given audio segment.
275: (8)                               **kwargs
276: (12)                             This method uses add_audio_segment, so any keyword arguments
277: (12)                             used there can be referenced here.
278: (8)                               """
279: (8)                               file_path = get_full_sound_file_path(sound_file)
280: (8)                               new_segment = AudioSegment.from_file(file_path)
281: (8)                               if gain:
282: (12)                             new_segment = new_segment.apply_gain(gain)
283: (8)                               self.add_audio_segment(new_segment, time, **kwargs)
284: (4)                               def begin_animation(self, allow_write: bool = False, file_path=None):
285: (8)                               """
286: (8)                               Used internally by manim to stream the animation to FFMPEG for
287: (8)                               displaying or writing to a file.
288: (8)                               Parameters
289: (8)                               -----
290: (8)                               allow_write
291: (12)                             Whether or not to write to a video file.
292: (8)                               """
293: (8)                               if write_to_movie() and allow_write:
294: (12)                             self.open_movie_pipe(file_path=file_path)
295: (4)                               def end_animation(self, allow_write: bool = False):
296: (8)                               """
297: (8)                               Internally used by Manim to stop streaming to
298: (8)                               FFMPEG gracefully.
299: (8)                               Parameters
300: (8)                               -----
301: (8)                               allow_write
302: (12)                             Whether or not to write to a video file.
303: (8)                               """
304: (8)                               if write_to_movie() and allow_write:
305: (12)                             self.close_movie_pipe()
306: (4)                               def write_frame(self, frame_or_renderer: np.ndarray | OpenGLRenderer):
307: (8)                               """
308: (8)                               Used internally by Manim to write a frame to
309: (8)                               the FFMPEG input buffer.
310: (8)                               Parameters
311: (8)                               -----
312: (8)                               frame_or_renderer
313: (12)                             Pixel array of the frame.
314: (8)                               """
315: (8)                               if config.renderer == RendererType.OPENGL:
316: (12)                             self.write_opengl_frame(frame_or_renderer)
317: (8)                               elif config.renderer == RendererType.CAIRO:
318: (12)                             frame = frame_or_renderer
319: (12)                             if write_to_movie():
320: (16)                               self.writing_process.stdin.write(frame.tobytes())
321: (12)                             if is_png_format() and not config["dry_run"]:
322: (16)                               self.output_image_from_array(frame)
323: (4)                               def write_opengl_frame(self, renderer: OpenGLRenderer):
324: (8)                               if write_to_movie():
325: (12)                             self.writing_process.stdin.write(
326: (16)                               renderer.get_raw_frame_buffer_object_data(),
327: (12) )
328: (8)                               elif is_png_format() and not config["dry_run"]:
329: (12)                               target_dir = self.image_file_path.parent /
330: (12)                               extension = self.image_file_path.suffix
331: (12)                               self.output_image(

```

```

332: (16)                         renderer.get_image(),
333: (16)                         target_dir,
334: (16)                         extension,
335: (16)                         config["zero_pad"],
336: (12)
337: (4)      def output_image_from_array(self, frame_data):
338: (8)          target_dir = self.image_file_path.parent / self.image_file_path.stem
339: (8)          extension = self.image_file_path.suffix
340: (8)          self.output_image(
341: (12)              Image.fromarray(frame_data),
342: (12)              target_dir,
343: (12)              extension,
344: (12)              config["zero_pad"],
345: (8)
346: (4)      def output_image(self, image: Image.Image, target_dir, ext, zero_pad:
347: (8)          bool):
348: (12)              if zero_pad:
349: {ext}"}
349: (8)
350: (12)
351: (8)
352: (4)
353: (8)
354: (8)      def save_final_image(self, image: np.ndarray):
355: (8)          """
356: (8)              The name is a misnomer. This method saves the image
357: (8)              passed to it as an in the default image directory.
358: (8)              Parameters
359: (12)                  -----
360: (8)                  image
361: (8)                      The pixel array of the image to save.
362: (12)
363: (8)
364: (12)
365: (8)      add_version_before_extension(self.image_file_path)
366: (8)          image.save(self.image_file_path)
367: (4)      def finish(self):
368: (8)
369: (8)          """
370: (8)          Finishes writing to the FFMPEG buffer or writing images
371: (8)          to output directory.
372: (8)          Combines the partial movie files into the
373: (8)          whole scene.
374: (8)          If save_last_frame is True, saves the last
375: (8)          frame in the default image directory.
376: (8)
377: (12)
378: (16)
379: (12)
380: (12)
381: (16)
382: (12)
383: (16)
384: (12)
385: (16)
386: (8)
387: (12)
self.image_file_path.stem
388: (12)          target_dir = self.image_file_path.parent /
389: (8)
390: (12)
391: (4)      def open_movie_pipe(self, file_path=None):
392: (8)
393: (8)          """
394: (8)              Used internally by Manim to initialise
395: (8)              FFMPEG and begin writing to FFMPEG's input
            buffer.

```

```

396: (8)             """
397: (8)         if file_path is None:
398: (12)             file_path = self.partial_movie_files[self.renderer.num_plays]
399: (8)         self.partial_movie_file_path = file_path
400: (8)         fps = config["frame_rate"]
401: (8)         if fps == int(fps): # fps is integer
402: (12)             fps = int(fps)
403: (8)         if config.renderer == RendererType.OPENGL:
404: (12)             width, height = self.renderer.get_pixel_shape()
405: (8)         else:
406: (12)             height = config["pixel_height"]
407: (12)             width = config["pixel_width"]
408: (8)         command = [
409: (12)             config.ffmpeg_executable,
410: (12)             "-y", # overwrite output file if it exists
411: (12)             "-f",
412: (12)             "rawvideo",
413: (12)             "-s",
414: (12)             "%dx%d" % (width, height), # size of one frame
415: (12)             "-pix_fmt",
416: (12)             "rgba",
417: (12)             "-r",
418: (12)             str(fps), # frames per second
419: (12)             "-i",
420: (12)             "-", # The input comes from a pipe
421: (12)             "-an", # Tells FFMPEG not to expect any audio
422: (12)             "-loglevel",
423: (12)             config["ffmpeg_loglevel"].lower(),
424: (12)             "-metadata",
425: (12)             f"comment=Rendered with Manim Community v{__version__}",
426: (8)         ]
427: (8)         if config.renderer == RendererType.OPENGL:
428: (12)             command += ["-vf", "vflip"]
429: (8)         if is_webm_format():
430: (12)             command += ["-vcodec", "libvpx-vp9", "-auto-alt-ref", "0"]
431: (8)         elif config["transparent"]:
432: (12)             command += ["-vcodec", "qtrle"]
433: (8)         else:
434: (12)             command += ["-vcodec", "libx264", "-pix_fmt", "yuv420p"]
435: (8)         command += [file_path]
436: (8)         self.writing_process = subprocess.Popen(command,
stdin=subprocess.PIPE)
437: (4)             def close_movie_pipe(self):
438: (8)                 """
439: (8)                     Used internally by Manim to gracefully stop writing to FFMPEG's input
buffer
440: (8)                     """
441: (8)                     self.writing_process.stdin.close()
442: (8)                     self.writing_process.wait()
443: (8)                     logger.info(
444: (12)                         f"Animation {self.renderer.num_plays} : Partial movie file written
in %(path)s",
445: (12)                         {"path": f"'{self.partial_movie_file_path}'"},
446: (8)                     )
447: (4)             def is_already_cached(self, hash_invocation: str):
448: (8)                 """Will check if a file named with `hash_invocation` exists.
Parameters
-----
hash_invocation
The hash corresponding to an invocation to either `scene.play` or
`scene.wait`.
451: (8)             Returns
452: (12)             -----
453: (8)             :class:`bool`
454: (8)                 Whether the file exists.
455: (8)             """
456: (12)             if not hasattr(self, "partial_movie_directory") or not
457: (8)                 write_to_movie():
458: (8)                     return False
459: (12)

```

```

460: (8)             path = (
461: (12)             self.partial_movie_directory
462: (12)             / f"{hash_invocation}{config['movie_file_extension']}"
463: (8)         )
464: (8)         return path.exists()
465: (4)     def combine_files(
466: (8)             self,
467: (8)             input_files: list[str],
468: (8)             output_file: Path,
469: (8)             create_gif=False,
470: (8)             includes_sound=False,
471: (4)         ):
472: (8)             file_list = self.partial_movie_directory /
"partial_movie_file_list.txt"
473: (8)             logger.debug(
474: (12)                 f"Partial movie files to combine ({len(input_files)} files): %"
(p)s",
475: (12)
476: (8)
477: (8)
478: (12)             {"p": input_files[:5]},
479: (12)
480: (16)             )
481: (16)
482: (8)
483: (12)             commands = [
484: (12)                 config.ffmpeg_executable,
485: (12)                 "-y", # overwrite output file if it exists
486: (12)                 "-f",
487: (12)                 "concat",
488: (12)                 "-safe",
489: (12)                 "0",
490: (12)                 "-i",
491: (12)                 str(file_list),
492: (12)                 "-loglevel",
493: (12)                 config.ffmpeg_loglevel.lower(),
494: (12)                 "-metadata",
495: (12)                 f"comment=Rendered with Manim Community v{__version__}",
496: (8)                 "-nostdin",
497: (8)             ]
498: (12)             if create_gif:
499: (16)                 commands += [
500: (16)                     "-vf",
501: (12)                     f"fps={np.clip(config['frame_rate'], 1, 50)},split[s0][s1];"
[s0]palettegen=stats_mode=dither=[p];[s1]
[p]paletteuse=dither=bayer:bayer_scale=5:diff_mode=rectangle",
502: (12)                 ]
503: (8)             else:
504: (12)                 commands += ["-c", "copy"]
505: (8)             if not includes_sound:
506: (12)                 commands += ["-an"]
507: (8)             commands += [str(output_file)]
508: (8)             combine_process = subprocess.Popen(commands)
509: (4)             combine_process.wait()
510: (8)         def combine_to_movie(self):
511: (8)             """Used internally by Manim to combine the separate
512: (8)             partial movie files that make up a Scene into a single
513: (8)             video file for that Scene.
514: (8)
515: (8)             """
516: (8)             partial_movie_files = [el for el in self.partial_movie_files if el is
not None]
517: (8)             movie_file_path = self.movie_file_path
518: (12)             if is_gif_format():
519: (8)                 movie_file_path = self.gif_file_path
520: (12)             logger.info("Combining to Movie file.")
521: (12)             self.combine_files(
522: (12)                 partial_movie_files,
523: (12)                 movie_file_path,
522: (12)                 is_gif_format(),
523: (12)                 self.includes_sound,

```

```

524: (8) )
525: (8)     if self.includes_sound:
526: (12)         sound_file_path = movie_file_path.with_suffix(".wav")
527: (12)         self.add_audio_segment(AudioSegment.silent(0))
528: (12)         self.audio_segment.export(
529: (16)             sound_file_path,
530: (16)             bitrate="312k",
531: (12)         )
532: (12)         temp_file_path = movie_file_path.with_name(
533: (16)             f"{movie_file_path.stem}_temp{movie_file_path.suffix}"
534: (12)         )
535: (12)         commands = [
536: (16)             config.ffmpeg_executable,
537: (16)             "-i",
538: (16)             str(movie_file_path),
539: (16)             "-i",
540: (16)             str(sound_file_path),
541: (16)             "-y", # overwrite output file if it exists
542: (16)             "-c:v",
543: (16)             "copy",
544: (16)             "-c:a",
545: (16)             "aac",
546: (16)             "-b:a",
547: (16)             "320k",
548: (16)             "-map",
549: (16)             "0:v:0",
550: (16)             "-map",
551: (16)             "1:a:0",
552: (16)             "-loglevel",
553: (16)             config.ffmpeg_loglevel.lower(),
554: (16)             "-metadata",
555: (16)             f"comment=Rendered with Manim Community v{__version__}",
556: (16)             str(temp_file_path),
557: (12)         ]
558: (12)         subprocess.call(commands)
559: (12)         shutil.move(str(temp_file_path), str(movie_file_path))
560: (12)         sound_file_path.unlink()
561: (8)         self.print_file_ready_message(str(movie_file_path))
562: (8)         if write_to_movie():
563: (12)             for file_path in partial_movie_files:
564: (16)                 modify_atime(file_path)
565: (4)         def combine_to_section_videos(self) -> None:
566: (8)             """Concatenate partial movie files for each section."""
567: (8)             self.finish_last_section()
568: (8)             sections_index: list[dict[str, Any]] = []
569: (8)             for section in self.sections:
570: (12)                 if section.video is not None:
571: (16)                     logger.info(f"Combining partial files for section
572: (16)                         {section.name}'")
573: (20)                     self.combine_files(
574: (20)                         section.get_clean_partial_movie_files(),
575: (16)                         self.sections_output_dir / section.video,
576: (16)                     )
577: (8)             sections_index.append(section.get_dict(self.sections_output_dir))
578: (8)             with (self.sections_output_dir / f"{self.output_name}.json").open("w")
579: (4)             json.dump(sections_index, file, indent=4)
580: (8)         def clean_cache(self):
581: (8)             """Will clean the cache by removing the oldest partial_movie_files."""
582: (12)             cached_partial_movies = [
583: (12)                 (self.partial_movie_directory / file_name)
584: (12)                 for file_name in self.partial_movie_directory.iterdir()
585: (8)                 if file_name != "partial_movie_file_list.txt"
586: (8)             ]
587: (12)             if len(cached_partial_movies) > config["max_files_cached"]:
588: (16)                 number_files_to_delete = (
589: (12)                     len(cached_partial_movies) - config["max_files_cached"]
589: (12)                 )

```

```

590: (12)             oldest_files_to_delete = sorted(
591: (16)                 cached_partial_movies,
592: (16)                 key=lambda path: path.stat().st_atime,
593: (12)             )[number_files_to_delete]
594: (12)             for file_to_delete in oldest_files_to_delete:
595: (16)                 file_to_delete.unlink()
596: (12)             logger.info(
597: (16)                 f"The partial movie directory is full (>
{config['max_files_cached']} files). Therefore, manim has removed the {number_files_to_delete} oldest file(s)."
598: (16)             " You can change this behaviour by changing max_files_cached
in config.",
599: (12)         )
600: (4)             def flush_cache_directory(self):
601: (8)                 """Delete all the cached partial movie files"""
602: (8)                 cached_partial_movies = [
603: (12)                     self.partial_movie_directory / file_name
604: (12)                     for file_name in self.partial_movie_directory.iterdir()
605: (12)                     if file_name != "partial_movie_file_list.txt"
606: (8)
607: (8)
608: (12)
609: (8)
610: (12)
611: (12)             {"par_dir": self.partial_movie_directory},
612: (8)
613: (4)             def write_subcaption_file(self):
614: (8)                 """Writes the subcaption file."""
615: (8)
616: (12)
617: (8)
618: (8)
encoding="utf-8")
619: (8)
620: (4)             def print_file_ready_message(self, file_path):
621: (8)                 """Prints the "File Ready" message to STDOUT."""
622: (8)
623: (8)
f'''{file_path}''')
-----
```

## File 134 - vector\_space\_scene.py:

```

1: (0)             """A scene suitable for vector spaces."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = ["VectorScene", "LinearTransformationScene"]
4: (0)             from typing import Callable
5: (0)             import numpy as np
6: (0)             from manim.mobject.geometry.arc import Dot
7: (0)             from manim.mobject.geometry.line import Arrow, Line, Vector
8: (0)             from manim.mobject.geometry.polygram import Rectangle
9: (0)             from manim.mobject.graphing.coordinate_systems import Axes, NumberPlane
10: (0)            from manim.mobject.opengl.opengl_mobject import OpenGLMobject
11: (0)            from manim.mobject.text.tex_mobject import MathTex, Tex
12: (0)            from manim.utils.config_ops import update_dict_recursively
13: (0)            from .. import config
14: (0)            from ..animation.animation import Animation
15: (0)            from ..animation.creation import Create, Write
16: (0)            from ..animation.fading import FadeOut
17: (0)            from ..animation.growing import GrowArrow
18: (0)            from ..animation.transform import ApplyFunction, ApplyPointwiseFunction,
Transform
19: (0)            from ..constants import *
20: (0)            from ..mobject.matrix import Matrix
21: (0)            from ..mobject.mobject import Mobject
22: (0)            from ..mobject.types.vectorized_mobject import VGroup, VMobject
23: (0)            from ..scene.scene import Scene
```

```

24: (0)         from ..utils.color import (
25: (4)             BLACK,
26: (4)             BLUE_D,
27: (4)             GREEN_C,
28: (4)             GREY,
29: (4)             RED_C,
30: (4)             WHITE,
31: (4)             YELLOW,
32: (4)             ManimColor,
33: (4)             ParsableManimColor,
34: (0)
35: (0)         )
36: (0)         from ..utils.rate_functions import rush_from, rush_into
37: (0)         from ..utils.space_ops import angle_of_vector
38: (0)             X_COLOR = GREEN_C
39: (0)             Y_COLOR = RED_C
40: (0)             Z_COLOR = BLUE_D
41: (0)         class VectorScene(Scene):
42: (4)             def __init__(self, basis_vector_stroke_width=6, **kwargs):
43: (8)                 super().__init__(**kwargs)
44: (8)                 self.basis_vector_stroke_width = basis_vector_stroke_width
45: (4)             def add_plane(self, animate: bool = False, **kwargs):
46: (8)                 """
47: (8)                     Adds a NumberPlane object to the background.
48: (8)                     Parameters
49: (8)                     -----
50: (8)                     animate
51: (12)                         Whether or not to animate the addition of the plane via Create.
52: (8)                     **kwargs
53: (12)                         Any valid keyword arguments accepted by NumberPlane.
54: (8)                     Returns
55: (8)                     -----
56: (8)                     NumberPlane
57: (12)                         The NumberPlane object.
58: (8)
59: (8)                     plane = NumberPlane(**kwargs)
60: (12)                     if animate:
61: (8)                         self.play(Create(plane, lag_ratio=0.5))
62: (8)                     self.add(plane)
63: (8)                     return plane
64: (4)             def add_axes(self, animate: bool = False, color: bool = WHITE, **kwargs):
65: (8)
66: (8)                 """
67: (8)                     Adds a pair of Axes to the Scene.
68: (8)                     Parameters
69: (8)                     -----
70: (8)                     animate
71: (12)                         Whether or not to animate the addition of the axes through Create.
72: (8)                     color
73: (12)                         The color of the axes. Defaults to WHITE.
74: (8)
75: (12)                     axes = Axes(color=color, axis_config={"unit_size": 1})
76: (8)                     if animate:
77: (12)                         self.play(Create(axes))
78: (8)                     self.add(axes)
79: (8)                     return axes
80: (4)             def lock_in_faded_grid(self, dimness: float = 0.7, axes_dimness: float =
81: (8)                         0.5):
82: (8)
83: (8)                     """
84: (8)                     This method freezes the NumberPlane and Axes that were already
85: (12)                         in the background, and adds new, manipulatable ones to the foreground.
86: (8)                     Parameters
87: (8)                     -----
88: (8)                     dimness
89: (12)                         The required dimness of the NumberPlane
90: (8)                     axes_dimness
91: (12)                         The required dimness of the Axes.
92: (8)
93: (8)                     plane = self.add_plane()
94: (8)                     axes = plane.get_axes()
95: (8)                     plane.fade(dimness)

```

```

92: (8)             axes.set_color(WHITE)
93: (8)             axes.fade(axes_dimness)
94: (8)             self.add(axes)
95: (8)             self.renderer.update_frame()
96: (8)             self.renderer.camera = Camera(self.renderer.get_frame())
97: (8)             self.clear()
98: (4)             def get_vector(self, numerical_vector: np.ndarray | list | tuple,
**kwargs):
99: (8)                 """
100: (8)                 Returns an arrow on the Plane given an input numerical vector.
101: (8)                 Parameters
102: (8)                 -----
103: (8)                 numerical_vector
104: (12)                 The Vector to plot.
105: (8)                 **kwargs
106: (12)                 Any valid keyword argument of Arrow.
107: (8)                 Returns
108: (8)                 -----
109: (8)                 Arrow
110: (12)                 The Arrow representing the Vector.
111: (8)                 """
112: (8)             return Arrow(
113: (12)                 self.plane.coords_to_point(0, 0),
114: (12)                 self.plane.coords_to_point(*numerical_vector[:2]),
115: (12)                 buff=0,
116: (12)                 **kwargs,
117: (8)             )
118: (4)             def add_vector(
119: (8)                 self,
120: (8)                 vector: Arrow | list | tuple | np.ndarray,
121: (8)                 color: str = YELLOW,
122: (8)                 animate: bool = True,
123: (8)                 **kwargs,
124: (4)             ):
125: (8)                 """
126: (8)                 Returns the Vector after adding it to the Plane.
127: (8)                 Parameters
128: (8)                 -----
129: (8)                 vector
130: (12)                 It can be a pre-made graphical vector, or the
131: (12)                 coordinates of one.
132: (8)                 color
133: (12)                 The string of the hex color of the vector.
134: (12)                 This is only taken into consideration if
135: (12)                 'vector' is not an Arrow. Defaults to YELLOW.
136: (8)                 animate
137: (12)                 Whether or not to animate the addition of the vector
138: (12)                 by using GrowArrow
139: (8)                 **kwargs
140: (12)                 Any valid keyword argument of Arrow.
141: (12)                 These are only considered if vector is not
142: (12)                 an Arrow.
143: (8)                 Returns
144: (8)                 -----
145: (8)                 Arrow
146: (12)                 The arrow representing the vector.
147: (8)                 """
148: (8)                 if not isinstance(vector, Arrow):
149: (12)                     vector = Vector(vector, color=color, **kwargs)
150: (8)                 if animate:
151: (12)                     self.play(GrowArrow(vector))
152: (8)                     self.add(vector)
153: (8)                     return vector
154: (4)             def write_vector_coordinates(self, vector: Arrow, **kwargs):
155: (8)                 """
156: (8)                 Returns a column matrix indicating the vector coordinates,
157: (8)                 after writing them to the screen.
158: (8)                 Parameters
159: (8)                 -----

```

```

160: (8)           vector
161: (12)          The arrow representing the vector.
162: (8)
163: (12)          **kwargs
164: (8)          Any valid keyword arguments of :meth:`~.Vector.coordinate_label`:
165: (8)          Returns
166: (8)          -----
167: (12)          :class:`~.Matrix`
168: (8)          The column matrix representing the vector.
169: (8)          """
170: (8)          coords = vector.coordinate_label(**kwargs)
171: (8)          self.play(Write(coords))
172: (8)          return coords
173: (4)          def get_basis_vectors(self, i_hat_color: str = X_COLOR, j_hat_color: str =
Y_COLOR):
174: (8)          """
175: (8)          Returns a VGroup of the Basis Vectors (1,0) and (0,1)
176: (8)          Parameters
177: (8)          -----
178: (8)          i_hat_color
179: (12)          The hex colour to use for the basis vector in the x direction
180: (8)          j_hat_color
181: (12)          The hex colour to use for the basis vector in the y direction
182: (8)          Returns
183: (8)          -----
184: (12)          VGroup
185: (8)          VGroup of the Vector Mobjects representing the basis vectors.
186: (8)          """
187: (12)          return VGroup(
188: (16)          *(
189: (16)              Vector(vect, color=color,
stroke_width=self.basis_vector_stroke_width)
190: (16)              for vect, color in [([1, 0], i_hat_color), ([0, 1],
j_hat_color)]
191: (12)          )
192: (4)          def get_basis_vector_labels(self, **kwargs):
193: (8)          """
194: (8)          Returns naming labels for the basis vectors.
195: (8)          Parameters
196: (8)          -----
197: (8)          **kwargs
198: (12)          Any valid keyword arguments of get_vector_label:
199: (16)          vector,
200: (16)          label (str, MathTex)
201: (16)          at_tip (bool=False),
202: (16)          direction (str="left"),
203: (16)          rotate (bool),
204: (16)          color (str),
205: (16)          label_scale_factor=VECTOR_LABEL_SCALE_FACTOR (int, float),
206: (8)          """
207: (8)          i_hat, j_hat = self.get_basis_vectors()
208: (8)          return VGroup(
209: (12)          *((
210: (16)              self.get_vector_label(
211: (20)                  vect, label, color=color, label_scale_factor=1, **kwargs
212: (16)              )
213: (16)              for vect, label, color in [
214: (20)                  (i_hat, "\u2102{\u210d}", X_COLOR),
215: (20)                  (j_hat, "\u2102{\u210e}", Y_COLOR),
216: (16)              ]
217: (12)          )
218: (8)          )
219: (4)          def get_vector_label(
220: (8)          self,
221: (8)          vector: Vector,
222: (8)          label,
223: (8)          at_tip: bool = False,
224: (8)          direction: str = "left",
225: (8)          rotate: bool = False,

```

```

226: (8)             color: str | None = None,
227: (8)             label_scale_factor: float = LARGE_BUFF - 0.2,
228: (4)             ):
229: (8)             """
230: (8)             Returns naming labels for the passed vector.
231: (8)             Parameters
232: (8)             -----
233: (8)             vector
234: (12)            Vector Object for which to get the label.
235: (8)             at_tip
236: (12)            Whether or not to place the label at the tip of the vector.
237: (8)             direction
238: (12)            If the label should be on the "left" or right of the vector.
239: (8)             rotate
240: (12)            Whether or not to rotate it to align it with the vector.
241: (8)             color
242: (12)            The color to give the label.
243: (8)             label_scale_factor
244: (12)            How much to scale the label by.
245: (8)             Returns
246: (8)             -----
247: (8)             MathTex
248: (12)            The MathTex of the label.
249: (8)             """
250: (8)
251: (12)
252: (16)
253: (12)
254: (12)
255: (16)
256: (12)
257: (8)
258: (8)
259: (8)
260: (12)
261: (12)
262: (12)
263: (8)
264: (12)
265: (12)
266: (16)
267: (12)
268: (16)
269: (12)
270: (16)
271: (12)
272: (12)
273: (8)
274: (4)
275: (8)
**kwargs
276: (4)
277: (8)
278: (8)
279: (8)
280: (8)
281: (8)
282: (8)
283: (12)
284: (8)
285: (12)
286: (8)
287: (12)
288: (8)
289: (12)
290: (8)
291: (8)
292: (8)
293: (12)

def label_vector(
    self, vector: Vector, label: MathTex | str, animate: bool = True,
    **kwargs
):
    """
    Shortcut method for creating, and animating the addition of
    a label for the vector.
    Parameters
    -----
    vector
        The vector for which the label must be added.
    label
        The MathTex/string of the label.
    animate
        Whether or not to animate the labelling w/ Write
    **kwargs
        Any valid keyword argument of get_vector_label
    Returns
    -----
    :class:`~.MathTex`
        The MathTex of the label.
    """

```

```

294: (8)             """
295: (8)         label = self.get_vector_label(vector, label, **kwargs)
296: (8)         if animate:
297: (12)             self.play(Write(label, run_time=1))
298: (8)         self.add(label)
299: (8)         return label
300: (4)     def position_x_coordinate(
301: (8)         self,
302: (8)         x_coord,
303: (8)         x_line,
304: (8)         vector,
305: (4)     ): # TODO Write DocStrings for this.
306: (8)         x_coord.next_to(x_line, -np.sign(vector[1]) * UP)
307: (8)         x_coord.set_color(X_COLOR)
308: (8)         return x_coord
309: (4)     def position_y_coordinate(
310: (8)         self,
311: (8)         y_coord,
312: (8)         y_line,
313: (8)         vector,
314: (4)     ): # TODO Write DocStrings for this.
315: (8)         y_coord.next_to(y_line, np.sign(vector[0]) * RIGHT)
316: (8)         y_coord.set_color(Y_COLOR)
317: (8)         return y_coord
318: (4)     def coords_to_vector(
319: (8)         self,
320: (8)         vector: np.ndarray | list | tuple,
321: (8)         coords_start: np.ndarray | list | tuple = 2 * RIGHT + 2 * UP,
322: (8)         clean_up: bool = True,
323: (4)     ):
324: (8)         """
325: (8)         This method writes the vector as a column matrix (henceforth called
326: (8)         the label),
327: (8)         takes the values in it one by one, and form the corresponding
328: (8)         lines that make up the x and y components of the vector. Then, an
329: (8)         Vector() based vector is created between the lines on the Screen.
330: (8)         Parameters
331: (8)         -----
332: (12)             vector
333: (8)                 The vector to show.
334: (12)             coords_start
335: (12)                 The starting point of the location of
336: (12)                 the label of the vector that shows it
337: (12)                 numerically.
338: (8)                 Defaults to 2 * RIGHT + 2 * UP or (2,2)
339: (12)             clean_up
340: (12)                 Whether or not to remove whatever
341: (8)                 this method did after it's done.
342: (8)         """
343: (8)         starting_mobjects = list(self.mobjects)
344: (8)         array = Matrix(vector)
345: (8)         array.shift(coords_start)
346: (8)         arrow = Vector(vector)
347: (8)         x_line = Line(ORIGIN, vector[0] * RIGHT)
348: (8)         y_line = Line(x_line.get_end(), arrow.get_end())
349: (8)         x_line.set_color(X_COLOR)
350: (8)         y_line.set_color(Y_COLOR)
351: (8)         x_coord, y_coord = array.get_mob_matrix().flatten()
352: (8)         self.play(Write(array, run_time=1))
353: (8)         self.wait()
354: (12)         self.play(
355: (16)             ApplyFunction(
356: (16)                 lambda x: self.position_x_coordinate(x, x_line, vector),
357: (12)                 x_coord,
358: (8)             ),
359: (8)         )
360: (8)         self.play(Create(x_line))
361: (12)         animations = [
362: (8)             ApplyFunction(

```

```

362: (16)           lambda y: self.position_y_coordinate(y, y_line, vector),
363: (16)           y_coord,
364: (12)           ),
365: (12)           FadeOut(array.get_brackets()),
366: (8)           ]
367: (8)           self.play(*animations)
368: (8)           y_coord, _ = (anim.mobject for anim in animations)
369: (8)           self.play(Create(y_line))
370: (8)           self.play(Create(arrow))
371: (8)           self.wait()
372: (8)           if clean_up:
373: (12)             self.clear()
374: (12)             self.add(*starting_mobjects)
375: (4) def vector_to_coords(
376: (8)     self,
377: (8)     vector: np.ndarray | list | tuple,
378: (8)     integer_labels: bool = True,
379: (8)     clean_up: bool = True,
380: (4)   ):
381: (8)   """
382: (8)   This method displays vector as a Vector() based vector, and then shows
383: (8)   the corresponding lines that make up the x and y components of the
vector.
384: (8)   Then, a column matrix (henceforth called the label) is created near
the
385: (8)   head of the Vector.
386: (8)   Parameters
387: (8)   -----
388: (8)   vector
389: (12)     The vector to show.
390: (8)   integer_labels
391: (12)     Whether or not to round the value displayed.
392: (12)     in the vector's label to the nearest integer
393: (8)   clean_up
394: (12)     Whether or not to remove whatever
395: (12)     this method did after it's done.
396: (8)   """
397: (8)   starting_mobjects = list(self.mobjects)
398: (8)   show_creation = False
399: (8)   if isinstance(vector, Arrow):
400: (12)     arrow = vector
401: (12)     vector = arrow.get_end()[:2]
402: (8)   else:
403: (12)     arrow = Vector(vector)
404: (12)     show_creation = True
405: (8)   array = arrow.coordinate_label(integer_labels=integer_labels)
406: (8)   x_line = Line(ORIGIN, vector[0] * RIGHT)
407: (8)   y_line = Line(x_line.get_end(), arrow.get_end())
408: (8)   x_line.set_color(X_COLOR)
409: (8)   y_line.set_color(Y_COLOR)
410: (8)   x_coord, y_coord = array.get_entries()
411: (8)   x_coord_start = self.position_x_coordinate(x_coord.copy(), x_line,
vector)
412: (8)   y_coord_start = self.position_y_coordinate(y_coord.copy(), y_line,
vector)
413: (8)   brackets = array.get_brackets()
414: (8)   if show_creation:
415: (12)     self.play(Create(arrow))
416: (8)     self.play(Create(x_line), Write(x_coord_start), run_time=1)
417: (8)     self.play(Create(y_line), Write(y_coord_start), run_time=1)
418: (8)     self.wait()
419: (8)     self.play(
420: (12)       Transform(x_coord_start, x_coord, lag_ratio=0),
421: (12)       Transform(y_coord_start, y_coord, lag_ratio=0),
422: (12)       Write(brackets, run_time=1),
423: (8)     )
424: (8)     self.wait()
425: (8)     self.remove(x_coord_start, y_coord_start, brackets)
426: (8)     self.add(array)

```

```

427: (8)             if clean_up:
428: (12)            self.clear()
429: (12)            self.add(*starting_mobjects)
430: (8)             return array, x_line, y_line
431: (4)              def show_ghost_movement(self, vector: Arrow | list | tuple | np.ndarray):
432: (8)                """
433: (8)                  This method plays an animation that partially shows the entire plane
434: (8)                  in the direction of a particular vector. This is useful when you wish
435: (8)                  to
436: (8)                  convey the idea of mentally moving the entire plane in a direction,
437: (8)                  without
438: (8)                  actually moving the plane.
439: (8)                  Parameters
440: (12)                 -----
441: (8)                 vector
442: (8)                     The vector which indicates the direction of movement.
443: (12)                 """
444: (8)                 if isinstance(vector, Arrow):
445: (12)                   vector = vector.get_end() - vector.get_start()
446: (8)                 elif len(vector) == 2:
447: (12)                   vector = np.append(np.array(vector), 0.0)
448: (8)                 x_max = int(config["frame_x_radius"] + abs(vector[0]))
449: (12)                 y_max = int(config["frame_y_radius"] + abs(vector[1]))
450: (16)                 dots = VMobject(
451: (16)                   *
452: (16)                     Dot(x * RIGHT + y * UP)
453: (12)                     for x in range(-x_max, x_max)
454: (8)                       for y in range(-y_max, y_max)
455: (8)                 )
456: (8)                 dots.set_fill(BLACK, opacity=0)
457: (8)                 dots_halfway = dots.copy().shift(vector / 2).set_fill(WHITE, 1)
458: (8)                 dots_end = dots.copy().shift(vector)
459: (8)                 self.play(Transform(dots, dots_halfway, rate_func=rush_into))
460: (8)                 self.play(Transform(dots, dots_end, rate_func=rush_from))
461: (0)                  self.remove(dots)
462: (4)              class LinearTransformationScene(VectorScene):
463: (4)                """
464: (4)                  This scene contains special methods that make it
465: (4)                  especially suitable for showing linear transformations.
466: (4)                  Parameters
467: (4)                  -----
468: (8)                    include_background_plane
469: (4)                      Whether or not to include the background plane in the scene.
470: (8)                    include_foreground_plane
471: (4)                      Whether or not to include the foreground plane in the scene.
472: (8)                    background_plane_kwargs
473: (4)                      Parameters to be passed to :class:`NumberPlane` to adjust the
474: (8)                    foreground_plane_kwargs
475: (4)                      Parameters to be passed to :class:`NumberPlane` to adjust the
476: (8)                    show_coordinates
477: (4)                      Whether or not to include the coordinates for the background plane.
478: (8)                    show_basis_vectors
479: (4)                      Whether to show the basis x_axis -> ``i_hat`` and y_axis -> ``j_hat``
480: (8)                    vectors.
481: (4)                    basis_vector_stroke_width
482: (8)                      The ``stroke_width`` of the basis vectors.
483: (4)                    i_hat_color
484: (8)                      The color of the ``i_hat`` vector.
485: (4)                    j_hat_color
486: (8)                      The color of the ``j_hat`` vector.
487: (4)                    leave_ghost_vectors
488: (4)                      Indicates the previous position of the basis vectors following a
489: (4)              Examples
490: (4)              -----

```

```

489: (4)          .. manim:: LinearTransformationSceneExample
490: (8)          class LinearTransformationSceneExample(LinearTransformationScene):
491: (12)          def __init__(self, **kwargs):
492: (16)              LinearTransformationScene.__init__(
493: (20)                  self,
494: (20)                  show_coordinates=True,
495: (20)                  leave_ghost_vectors=True,
496: (20)                  **kwargs
497: (16)          )
498: (12)          def construct(self):
499: (16)              matrix = [[1, 1], [0, 1]]
500: (16)              self.apply_matrix(matrix)
501: (16)              self.wait()
502: (4)
503: (4)          """
504: (8)          def __init__(
505: (8)              self,
506: (8)              include_background_plane: bool = True,
507: (8)              include_foreground_plane: bool = True,
508: (8)              background_plane_kwargs: dict | None = None,
509: (8)              foreground_plane_kwargs: dict | None = None,
510: (8)              show_coordinates: bool = False,
511: (8)              show_basis_vectors: bool = True,
512: (8)              basis_vector_stroke_width: float = 6,
513: (8)              i_hat_color: ParsableManimColor = X_COLOR,
514: (8)              j_hat_color: ParsableManimColor = Y_COLOR,
515: (8)              leave_ghost_vectors: bool = False,
516: (8)              **kwargs,
517: (4):
518: (8)                  super().__init__(**kwargs)
519: (8)                  self.include_background_plane = include_background_plane
520: (8)                  self.include_foreground_plane = include_foreground_plane
521: (8)                  self.show_coordinates = show_coordinates
522: (8)                  self.show_basis_vectors = show_basis_vectors
523: (8)                  self.basis_vector_stroke_width = basis_vector_stroke_width
524: (8)                  self.i_hat_color = ManimColor(i_hat_color)
525: (8)                  self.j_hat_color = ManimColor(j_hat_color)
526: (8)                  self.leave_ghost_vectors = leave_ghost_vectors
527: (12)                  self.background_plane_kwargs = {
528: (12)                      "color": GREY,
529: (16)                      "axis_config": {
530: (12)                          "color": GREY,
531: (12)                      },
532: (16)                      "background_line_style": {
533: (16)                          "stroke_color": GREY,
534: (12)                          "stroke_width": 1,
535: (8)                      },
536: (8)                  self.ghost_vectors = VGroup()
537: (8)                  self.foreground_plane_kwargs = {
538: (12)                      "x_range": np.array([-config["frame_width"],
config["frame_width"], 1.0]),
539: (12)                      "y_range": np.array([-config["frame_width"],
config["frame_width"], 1.0]),
540: (12)                      "faded_line_ratio": 1,
541: (8)                  }
542: (8)                  self.update_default_configs(
543: (12)                      (self.foreground_plane_kwargs, self.background_plane_kwargs),
544: (12)                      (foreground_plane_kwargs, background_plane_kwargs),
545: (8)                  )
546: (4)          @staticmethod
547: (4)          def update_default_configs(default_configs, passed_configs):
548: (8)              for default_config, passed_config in zip(default_configs,
passed_configs):
549: (12)                  if passed_config is not None:
550: (16)                      update_dict_recursively(default_config, passed_config)
551: (4)          def setup(self):
552: (8)              if hasattr(self, "has_already_setup"):
553: (12)                  return
554: (8)              self.has_already_setup = True

```

```

555: (8)             self.background_mobjects = []
556: (8)             self.foreground_mobjects = []
557: (8)             self.transformable_mobjects = []
558: (8)             self.moving_vectors = []
559: (8)             self.transformable_labels = []
560: (8)             self.moving_mobjects = []
561: (8)             self.background_plane = NumberPlane(**self.background_plane_kwargs)
562: (8)             if self.show_coordinates:
563: (12)                 self.background_plane.add_coordinates()
564: (8)             if self.include_background_plane:
565: (12)                 self.add_background_mobject(self.background_plane)
566: (8)             if self.include_foreground_plane:
567: (12)                 self.plane = NumberPlane(**self.foreground_plane_kwargs)
568: (12)                 self.add_transformable_mobject(self.plane)
569: (8)             if self.show_basis_vectors:
570: (12)                 self.basis_vectors = self.get_basis_vectors(
571: (16)                     i_hat_color=self.i_hat_color,
572: (16)                     j_hat_color=self.j_hat_color,
573: (12)                 )
574: (12)                 self.moving_vectors += list(self.basis_vectors)
575: (12)                 self.i_hat, self.j_hat = self.basis_vectors
576: (12)                 self.add(self.basis_vectors)
577: (4)             def add_special_mobjects(self, mob_list: list, *mobs_to_add: Mobject):
578: (8)                 """
579: (8)                     Adds mobjects to a separate list that can be tracked,
580: (8)                     if these mobjects have some extra importance.
581: (8)                     Parameters
582: (8)                     -----
583: (8)                     mob_list
584: (12)                         The special list to which you want to add
585: (12)                         these mobjects.
586: (8)                     *mobs_to_add
587: (12)                         The mobjects to add.
588: (8)                     """
589: (8)                     for mobject in mobs_to_add:
590: (12)                         if mobject not in mob_list:
591: (16)                             mob_list.append(mobject)
592: (16)                             self.add(mobject)
593: (4)             def add_background_mobject(self, *mobjects: Mobject):
594: (8)                 """
595: (8)                     Adds the mobjects to the special list
596: (8)                     self.background_mobjects.
597: (8)                     Parameters
598: (8)                     -----
599: (8)                     *mobjects
600: (12)                         The mobjects to add to the list.
601: (8)                     """
602: (8)                     self.add_special_mobjects(self.background_mobjects, *mobjects)
603: (4)             def add_foreground_mobject(self, *mobjects: Mobject):
604: (8)                 """
605: (8)                     Adds the mobjects to the special list
606: (8)                     self.foreground_mobjects.
607: (8)                     Parameters
608: (8)                     -----
609: (8)                     *mobjects
610: (12)                         The mobjects to add to the list
611: (8)                     """
612: (8)                     self.add_special_mobjects(self.foreground_mobjects, *mobjects)
613: (4)             def add_transformable_mobject(self, *mobjects: Mobject):
614: (8)                 """
615: (8)                     Adds the mobjects to the special list
616: (8)                     self.transformable_mobjects.
617: (8)                     Parameters
618: (8)                     -----
619: (8)                     *mobjects
620: (12)                         The mobjects to add to the list.
621: (8)                     """
622: (8)                     self.add_special_mobjects(self.transformable_mobjects, *mobjects)
623: (4)             def add_moving_mobject(

```

```

624: (8)           self, mobject: Mobject, target_mobject: Mobject | None = None
625: (4)
626: (8)
627: (8)
628: (8)
629: (8)
630: (8)
631: (8)
632: (8)
633: (8)
634: (8)
635: (12)
636: (8)
637: (12)
638: (8)
639: (8)
640: (8)
641: (4)
642: (8)
643: (8)
``VGroup`` of
644: (8)
645: (8)
646: (8)
647: (4)
648: (8)
3
649: (4)
650: (8)
651: (8)
652: (8)
653: (8)
654: (8)
655: (12)
656: (8)
657: (12)
658: (8)
659: (12)
660: (8)
661: (8)
662: (8)
663: (8)
664: (8)
665: (12)
666: (12)
667: (12)
668: (12)
669: (12)
670: (12)
671: (12)
672: (8)
673: (8)
674: (8)
def get_ghost_vectors(self) -> VGroup:
    """
    Returns all ghost vectors ever added to ``self``. Each element is a
    two ghost vectors.
    """
    return self.ghost_vectors
def get_unit_square(
    self, color: str = YELLOW, opacity: float = 0.3, stroke_width: float =
):
    """
    Returns a unit square for the current NumberPlane.
    Parameters
    -----
    color
        The string of the hex color code of the color wanted.
    opacity
        The opacity of the square
    stroke_width
        The stroke_width in pixels of the border of the square
    Returns
    -----
    Square
    """
    square = self.square = Rectangle(
        color=color,
        width=self.plane.get_x_unit_size(),
        height=self.plane.get_y_unit_size(),
        stroke_color=color,
        stroke_width=stroke_width,
        fill_color=color,
        fill_opacity=opacity,
    )
    square.move_to(self.plane.coords_to_point(0, 0), DL)
    return square
def add_unit_square(self, animate: bool = False, **kwargs):
    """
    Adds a unit square to the scene via
    self.get_unit_square.
    Parameters
    -----
    animate
        Whether or not to animate the addition
        with DrawBorderThenFill.
    **kwargs
        Any valid keyword arguments of
        self.get_unit_square()
    Returns
    -----
    Square
        The unit square.

```

```

691: (8)             """
692: (8)             square = self.get_unit_square(**kwargs)
693: (8)             if animate:
694: (12)                 self.play(
695: (16)                     DrawBorderThenFill(square),
696: (16)                     Animation(Group(*self.moving_vectors)),
697: (12)                 )
698: (8)             self.add_transformable_mobject(square)
699: (8)             self.bring_to_front(*self.moving_vectors)
700: (8)             self.square = square
701: (8)             return self
702: (4)             def add_vector(
703: (8)                 self, vector: Arrow | list | tuple | np.ndarray, color: str = YELLOW,
**kwargs
704: (4)             ):
705: (8)                 """
706: (8)                 Adds a vector to the scene, and puts it in the special
707: (8)                 list self.moving_vectors.
708: (8)                 Parameters
709: (8)                 -----
710: (8)                 vector
711: (12)                     It can be a pre-made graphical vector, or the
712: (12)                     coordinates of one.
713: (8)                 color
714: (12)                     The string of the hex color of the vector.
715: (12)                     This is only taken into consideration if
716: (12)                     'vector' is not an Arrow. Defaults to YELLOW.
717: (8)                 **kwargs
718: (12)                     Any valid keyword argument of VectorScene.add_vector.
719: (8)                 Returns
720: (8)                 -----
721: (8)                 Arrow
722: (12)                     The arrow representing the vector.
723: (8)                 """
724: (8)                 vector = super().add_vector(vector, color=color, **kwargs)
725: (8)                 self.moving_vectors.append(vector)
726: (8)                 return vector
727: (4)             def write_vector_coordinates(self, vector: Arrow, **kwargs):
728: (8)                 """
729: (8)                 Returns a column matrix indicating the vector coordinates,
730: (8)                 after writing them to the screen, and adding them to the
731: (8)                 special list self.foreground_mobjects
732: (8)                 Parameters
733: (8)                 -----
734: (8)                 vector
735: (12)                     The arrow representing the vector.
736: (8)                 **kwargs
737: (12)                     Any valid keyword arguments of
VectorScene.write_vector_coordinates
738: (8)                 Returns
739: (8)                 -----
740: (8)                 Matrix
741: (12)                     The column matrix representing the vector.
742: (8)                 """
743: (8)                 coords = super().write_vector_coordinates(vector, **kwargs)
744: (8)                 self.add_foreground_mobject(coords)
745: (8)                 return coords
746: (4)             def add_transformable_label(
747: (8)                 self,
748: (8)                 vector: Vector,
749: (8)                 label: MathTex | str,
750: (8)                 transformation_name: str | MathTex = "L",
751: (8)                 new_label: str | MathTex | None = None,
752: (8)                 **kwargs,
753: (4)             ):
754: (8)                 """
755: (8)                 Method for creating, and animating the addition of
756: (8)                 a transformable label for the vector.
757: (8)                 Parameters

```

```

758: (8)           -----
759: (8)           vector
760: (12)          The vector for which the label must be added.
761: (8)           label
762: (12)          The MathTex/string of the label.
763: (8)           transformation_name
764: (12)          The name to give the transformation as a label.
765: (8)           new_label
766: (12)          What the label should display after a Linear Transformation
767: (8)           **kwargs
768: (12)          Any valid keyword argument of get_vector_label
769: (8)           Returns
770: (8)           -----
771: (8)           :class:`~.MathTex`
772: (12)          The MathTex of the label.
773: (8)           """
774: (8)           label_mob = self.label_vector(vector, label, **kwargs)
775: (8)           if new_label:
776: (12)             label_mob.target_text = new_label
777: (8)           else:
778: (12)             label_mob.target_text = "{}({})".format(
779: (16)               transformation_name,
780: (16)               label_mob.get_tex_string(),
781: (12)             )
782: (8)           label_mob.vector = vector
783: (8)           label_mob.kwargs = kwargs
784: (8)           if "animate" in label_mob.kwargs:
785: (12)             label_mob.kwargs.pop("animate")
786: (8)             self.transformable_labels.append(label_mob)
787: (8)           return label_mob
788: (4)           def add_title(
789: (8)             self,
790: (8)             title: str | MathTex | Tex,
791: (8)             scale_factor: float = 1.5,
792: (8)             animate: bool = False,
793: (4)           ):
794: (8)             """
795: (8)             Adds a title, after scaling it, adding a background rectangle,
796: (8)             moving it to the top and adding it to foreground_mobjects adding
797: (8)             it as a local variable of self. Returns the Scene.
798: (8)             Parameters
799: (8)             -----
800: (8)             title
801: (12)               What the title should be.
802: (8)             scale_factor
803: (12)               How much the title should be scaled by.
804: (8)             animate
805: (12)               Whether or not to animate the addition.
806: (8)             Returns
807: (8)             -----
808: (8)             LinearTransformationScene
809: (12)               The scene with the title added to it.
810: (8)             """
811: (8)             if not isinstance(title, (Mobject, OpenGLMobject)):
812: (12)               title = Tex(title).scale(scale_factor)
813: (8)             title.to_edge(UP)
814: (8)             title.add_background_rectangle()
815: (8)             if animate:
816: (12)               self.play(Write(title))
817: (8)             self.add_foreground_mobject(title)
818: (8)             self.title = title
819: (8)             return self
820: (4)           def get_matrix_transformation(self, matrix: np.ndarray | list | tuple):
821: (8)             """
822: (8)             Returns a function corresponding to the linear
823: (8)             transformation represented by the matrix passed.
824: (8)             Parameters
825: (8)             -----
826: (8)             matrix

```

```

827: (12)           The matrix.
828: (8)
829: (8)           """
830: (4)           return self.get_transposed_matrix_transformation(np.array(matrix).T)
831: (8)           def get_transposed_matrix_transformation(
832: (4)           self, transposed_matrix: np.ndarray | list | tuple
833: (8)           ):
834: (8)           """
835: (8)           Returns a function corresponding to the linear
836: (8)           transformation represented by the transposed
837: (8)           matrix passed.
838: (8)           Parameters
839: (8)           -----
840: (12)           transposed_matrix
841: (8)           The matrix.
842: (8)           """
843: (8)           transposed_matrix = np.array(transposed_matrix)
844: (12)           if transposed_matrix.shape == (2, 2):
845: (12)               new_matrix = np.identity(3)
846: (12)               new_matrix[:2, :2] = transposed_matrix
847: (8)               transposed_matrix = new_matrix
848: (12)           elif transposed_matrix.shape != (3, 3):
849: (8)               raise ValueError("Matrix has bad dimensions")
850: (4)           return lambda point: np.dot(point, transposed_matrix)
851: (8)           def get_piece_movement(self, pieces: list | tuple | np.ndarray):
852: (8)           """
853: (8)           This method returns an animation that moves an arbitrary
854: (8)           mobject in "pieces" to its corresponding .target value.
855: (8)           If self.leave_ghost_vectors is True, ghosts of the original
856: (8)           positions/mobjects are left on screen
857: (8)           Parameters
858: (8)           -----
859: (12)           pieces
860: (8)           The pieces for which the movement must be shown.
861: (8)           Returns
862: (8)           -----
863: (12)           Animation
864: (8)           The animation of the movement.
865: (8)           """
866: (8)           v_pieces = [piece for piece in pieces if isinstance(piece, VMobject)]
867: (8)           start = VGroup(*v_pieces)
868: (8)           target = VGroup(*(mob.target for mob in v_pieces))
869: (12)           if self.leave_ghost_vectors and start.submobjects:
870: (12)               self.ghost_vectors.add(start.copy().fade(0.7))
871: (8)               self.add(self.ghost_vectors[-1])
872: (4)           return Transform(start, target, lag_ratio=0)
873: (8)           def get_moving_mobject_movement(self, func: Callable[[np.ndarray],
874: (8)           np.ndarray]):
875: (8)           """
876: (8)           This method returns an animation that moves a mobject
877: (8)           in "self.moving_mobjects" to its corresponding .target value.
878: (8)           func is a function that determines where the .target goes.
879: (8)           Parameters
880: (12)           func
881: (12)           The function that determines where the .target of
882: (8)           the moving mobject goes.
883: (8)           Returns
884: (8)           -----
885: (12)           Animation
886: (8)           The animation of the movement.
887: (8)           """
888: (12)           for m in self.moving_mobjects:
889: (16)               if m.target is None:
890: (12)                   m.target = m.copy()
891: (12)                   target_point = func(m.get_center())
892: (8)                   m.target.move_to(target_point)
893: (4)           return self.get_piece_movement(self.moving_mobjects)
894: (8)           def get_vector_movement(self, func: Callable[[np.ndarray], np.ndarray]):
895: (8)           """

```

```

895: (8)             This method returns an animation that moves a mobject
896: (8)             in "self.moving_vectors" to its corresponding .target value.
897: (8)             func is a function that determines where the .target goes.
898: (8)             Parameters
899: (8)             -----
900: (8)             func
901: (12)            The function that determines where the .target of
902: (12)            the moving mobject goes.
903: (8)
904: (8)
905: (8)             Returns
906: (8)             -----
907: (8)             Animation
908: (8)             The animation of the movement.
909: (12)             """
910: (12)             for v in self.moving_vectors:
911: (12)                 v.target = Vector(func(v.get_end()), color=v.get_color())
912: (16)                 norm = np.linalg.norm(v.target.get_end())
913: (8)                 if norm < 0.1:
914: (4)                     v.target.get_tip().scale(norm)
915: (8)             return self.get_piece_movement(self.moving_vectors)
def get_transformable_label_movement(self):
916: (8)             """
917: (8)             This method returns an animation that moves all labels
918: (8)             in "self.transformable_labels" to its corresponding .target .
919: (8)             Returns
920: (8)             -----
921: (12)             Animation
922: (8)             The animation of the movement.
923: (8)             """
924: (12)             for label in self.transformable_labels:
925: (16)                 label.target = self.get_vector_label(
926: (12)                     label.vector.target, label.target_text, **label.kwargs
927: (8)                 )
928: (4)             return self.get_piece_movement(self.transformable_labels)
def apply_matrix(self, matrix: np.ndarray | list | tuple, **kwargs):
929: (8)             """
930: (8)             Applies the transformation represented by the
931: (8)             given matrix to the number plane, and each vector/similar
932: (8)             mobject on it.
933: (8)
934: (8)
935: (8)             Parameters
936: (12)             -----
937: (8)             matrix
938: (12)                 The matrix.
939: (8)             **kwargs
940: (8)                 Any valid keyword argument of self.apply_transposed_matrix()
941: (4)             """
942: (8)             self.apply_transposed_matrix(np.array(matrix).T, **kwargs)
def apply_inverse(self, matrix: np.ndarray | list | tuple, **kwargs):
943: (8)             """
944: (8)             This method applies the linear transformation
945: (8)             represented by the inverse of the passed matrix
946: (8)             to the number plane, and each vector/similar mobject on it.
947: (8)
948: (8)             Parameters
949: (12)             -----
950: (8)             matrix
951: (12)                 The matrix whose inverse is to be applied.
952: (8)             **kwargs
953: (8)                 Any valid keyword argument of self.apply_matrix()
954: (4)             """
955: (8)             self.apply_matrix(np.linalg.inv(matrix), **kwargs)
def apply_transposed_matrix(
956: (4)             self, transposed_matrix: np.ndarray | list | tuple, **kwargs
957: (8)             ):
958: (8)             """
959: (8)             Applies the transformation represented by the
960: (8)             given transposed matrix to the number plane,
961: (8)             and each vector/similar mobject on it.
962: (8)
963: (8)             Parameters
964: (8)             -----
transposed_matrix

```

```

964: (12)           The matrix.
965: (8)            **kwargs
966: (12)           Any valid keyword argument of self.apply_function()
967: (8)
968: (8)           """
969: (8)           func = self.get_transposed_matrix_transformation(transposed_matrix)
970: (12)           if "path_arc" not in kwargs:
971: (16)               net_rotation = np.mean(
972: (12)                   [angle_of_vector(func(RIGHT)), angle_of_vector(func(UP)) -
973: (12)                       )
974: (8)               kwargs["path_arc"] = net_rotation
975: (4)           self.apply_function(func, **kwargs)
976: (8)       def apply_inverse_transpose(self, t_matrix: np.ndarray | list | tuple,
977: (8)           """
978: (8)           Applies the inverse of the transformation represented
979: (8)           by the given transposed matrix to the number plane and each
980: (8)           vector/similar mobject on it.
981: (8)       Parameters
982: (8)           -----
983: (12)           t_matrix
984: (8)               The matrix.
985: (12)           **kwargs
986: (8)               Any valid keyword argument of self.apply_transposed_matrix()
987: (8)           """
988: (8)           t_inv = np.linalg.inv(np.array(t_matrix).T).T
989: (4)           self.apply_transposed_matrix(t_inv, **kwargs)
990: (8)       def apply_nonlinear_transformation(
991: (4)           self, function: Callable[[np.ndarray], np.ndarray], **kwargs
992: (8)           ):
993: (8)           """
994: (8)           Applies the non-linear transformation represented
995: (8)           by the given function to the number plane and each
996: (8)           vector/similar mobject on it.
997: (8)       Parameters
998: (8)           -----
999: (12)           function
1000: (8)               The function.
1001: (12)           **kwargs
1002: (8)               Any valid keyword argument of self.apply_function()
1003: (8)           """
1004: (8)           self.plane.prepare_for_nonlinear_transform()
1005: (4)           self.apply_function(function, **kwargs)
1006: (8)       def apply_function(
1007: (8)           self,
1008: (8)           function: Callable[[np.ndarray], np.ndarray],
1009: (8)           added_anims: list = [],
1010: (4)           **kwargs,
1011: (8)           ):
1012: (8)           """
1013: (8)           Applies the given function to each of the mobjects in
1014: (8)           self.transformable_mobjects, and plays the animation showing
1015: (8)           this.
1016: (8)       Parameters
1017: (8)           -----
1018: (12)           function
1019: (12)               The function that affects each point
1020: (8)               of each mobject in self.transformable_mobjects.
1021: (12)           added_anims
1022: (12)               Any other animations that need to be played
1023: (8)               simultaneously with this.
1024: (12)           **kwargs
1025: (8)               Any valid keyword argument of a self.play() call.
1026: (8)           """
1027: (12)           if "run_time" not in kwargs:
1028: (8)               kwargs["run_time"] = 3
1029: (12)           anims = (
1030: (16)               [
                           ApplyPointwiseFunction(function, t_mob)

```

12/20/24, 4:24 AM

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
1031: (16)                     for t_mob in self.transformable_mobjects
1032: (12)                 ]
1033: (12)             + [
1034: (16)                 self.get_vector_movement(function),
1035: (16)                 self.get_transformable_label_movement(),
1036: (16)                 self.get_moving_mobject_movement(function),
1037: (12)             ]
1038: (12)             + [Animation(f_mob) for f_mob in self.foreground_mobjects]
1039: (12)             + added_anims
1040: (8)         )
1041: (8)     self.play(*anims, **kwargs)
```

---

File 135 - autocolor\_directive.py:

```
1: (0)         """A directive for documenting colors in Manim."""
2: (0)         from __future__ import annotations
3: (0)         import inspect
4: (0)         from typing import TYPE_CHECKING
5: (0)         from docutils import nodes
6: (0)         from docutils.parsers.rst import Directive
7: (0)         from manim import ManimColor
8: (0)         if TYPE_CHECKING:
9: (4)             from sphinx.application import Sphinx
10: (0)             __all__ = ["ManimColorModuleDocumenter"]
11: (0)         def setup(app: Sphinx) -> None:
12: (4)             app.add_directive("automanimcolormodule", ManimColorModuleDocumenter)
13: (0)         class ManimColorModuleDocumenter(Directive):
14: (4)             objtype = "automanimcolormodule"
15: (4)             required_arguments = 1
16: (4)             has_content = True
17: (4)             def add_directive_header(self, sig: str) -> None:
18: (8)                 super().add_directive_header(sig)
19: (4)             def run(self) -> list[nodes.Element]:
20: (8)                 module_name = self.arguments[0]
21: (8)                 try:
22: (12)                     import importlib
23: (12)                     module = importlib.import_module(module_name)
24: (8)                 except ImportError:
25: (12)                     return [
26: (16)                         nodes.error(
27: (20)                             None,
28: (20)                             nodes.paragraph(text="Failed to import module '%s'" %
29: (16)                               module_name),
30: (12)                         )
31: (8)                     ]
32: (8)                     num_color_cols = 2
33: (8)                     table = nodes.table(align="center")
34: (8)                     tgroup = nodes.tgroup(cols=num_color_cols * 2)
35: (8)                     table += tgroup
36: (12)                     for _ in range(num_color_cols * 2):
37: (8)                         tgroup += nodes.colspec(colwidth=1)
38: (8)                     thead = nodes.thead()
39: (8)                     row = nodes.row()
40: (12)                     for _ in range(num_color_cols):
41: (12)                         col1 = nodes.paragraph(text="Color Name")
42: (12)                         col2 = nodes.paragraph(text="RGB Hex Code")
43: (12)                         row += nodes.entry("", col1)
44: (8)                         row += nodes.entry("", col2)
45: (8)                     thead += row
46: (8)                     tgroup += thead
47: (8)                     color_elements = []
48: (12)                     for member_name, member_obj in inspect.getmembers(module):
49: (16)                         if isinstance(member_obj, ManimColor):
50: (16)                             r, g, b = member_obj.to_rgb()
51: (16)                             luminance = 0.2126 * r + 0.7152 * g + 0.0722 * b
52: (20)                             if luminance > 0.5:
53: (16)                                 font_color = "black"
```

```

53: (16)                     else:
54: (20)                         font_color = "white"
55: (16)                     color_elements.append((member_name, member_obj.to_hex()),
font_color))
56: (8)                     tbody = nodes.tbody()
57: (8)                     for base_i in range(0, len(color_elements), num_color_cols):
58: (12)                         row = nodes.row()
59: (12)                         for member_name, hex_code, font_color in color_elements[
60: (16)                             base_i : base_i + num_color_cols
61: (12)                         ]:
62: (16)                             col1 = nodes.literal(text=member_name)
63: (16)                             col2 = nodes.raw(
64: (20)                                 "",
65: (20)                                 f'<div style="background-color:{hex_code};padding: 0.25rem
0;border-radius:8px;margin: 0.5rem 0.2rem"><code style="color:{font_color};">{hex_code}</code>
</div>',
66: (20)                             format="html",
67: (16)                         )
68: (16)                         row += nodes.entry("", col1)
69: (16)                         row += nodes.entry("", col2)
70: (12)                     tbody += row
71: (8)                     tgroup += tbody
72: (8)                     return [table]

```

---

## File 136 - autoaliasattr\_directive.py:

```

1: (0)             """A directive for documenting type aliases and other module-level
attributes."""
2: (0)             from __future__ import annotations
3: (0)             from typing import TYPE_CHECKING
4: (0)             from docutils import nodes
5: (0)             from docutils.parsers.rst import Directive
6: (0)             from docutils.statemachine import ViewList
7: (0)             from manim.utils.docbuild.module_parsing import parse_module_attributes
8: (0)             if TYPE_CHECKING:
9: (4)                 from sphinx.application import Sphinx
10: (4)                 from typing_extensions import TypeAlias
11: (0)             __all__ = ["AliasAttrDocumenter"]
12: (0)             ALIAS_DOCS_DICT, DATA_DICT = parse_module_attributes()
13: (0)             ALIAS_LIST = [
14: (4)                 alias_name
15: (4)                 for module_dict in ALIAS_DOCS_DICT.values()
16: (4)                 for category_dict in module_dict.values()
17: (4)                 for alias_name in category_dict.keys()
18: (0)             ]
19: (0)             def smart_replace(base: str, alias: str, substitution: str) -> str:
20: (4)                 """Auxiliary function for substituting type aliases into a base
string, when there are overlaps between the aliases themselves.
Parameters
-----
base
        The string in which the type aliases will be located and
        replaced.
alias
        The substring to be substituted.
substitution
        The string which will replace every occurrence of ``alias``.
Returns
-----
str
        The new string after the alias substitution.
"""
26: (8)             occurrences = []
27: (4)             len_alias = len(alias)
28: (8)             len_base = len(base)
29: (4)             condition = lambda char: (not char.isalnum()) and char != "_"
30: (8)             start = 0
31: (4)
32: (4)
33: (4)
34: (8)
35: (4)
36: (4)
37: (4)
38: (4)
39: (4)
40: (4)

```

```

41: (4)             i = 0
42: (4)             while True:
43: (8)                 i = base.find(alias, start)
44: (8)                 if i == -1:
45: (12)                     break
46: (8)                 if (i == 0 or condition(base[i - 1])) and (
47: (12)                     i + len_alias == len_base or condition(base[i + len_alias]))
48: (8)                 ):
49: (12)                     occurrences.append(i)
50: (8)                 start = i + len_alias
51: (4)             for o in occurrences[::-1]:
52: (8)                 base = base[:o] + substitution + base[o + len_alias :]
53: (4)             return base
54: (0)         def setup(app: Sphinx) -> None:
55: (4)             app.add_directive("autoaliasattr", AliasAttrDocumenter)
56: (0)         class AliasAttrDocumenter(Directive):
57: (4)             """Directive which replaces Sphinx's Autosummary for module-level
58: (4)             attributes: instead, it manually crafts a new "Type Aliases"
59: (4)             section, where all the module-level attributes which are explicitly
60: (4)             annotated as :class:`TypeAlias` are considered as such, for their
61: (4)             use all around the Manim docs.
62: (4)             These type aliases are separated from the "regular" module-level
63: (4)             attributes, which get their traditional "Module Attributes"
64: (4)             section autogenerated with Sphinx's Autosummary under "Type
65: (4)             Aliases".
66: (4)             See ``docs/source/_templates/autosummary/module.rst`` to watch
67: (4)             this directive in action.
68: (4)             See :func:`~.parse_module_attributes` for more information on how
69: (4)             the modules are parsed to obtain the :class:`TypeAlias` information
70: (4)             and separate it from the other attributes.
71: (4)             """
72: (4)             objtype = "autoaliasattr"
73: (4)             required_arguments = 1
74: (4)             has_content = True
75: (4)             def run(self) -> list[nodes.Element]:
76: (8)                 module_name = self.arguments[0]
77: (8)                 module_alias_dict = ALIAS_DOCS_DICT.get(module_name[6:], None)
78: (8)                 module_attrs_list = DATA_DICT.get(module_name[6:], None)
79: (8)                 content = nodes.container()
80: (8)                 if module_alias_dict is not None:
81: (12)                     module_alias_section = nodes.section(ids=[f"{module_name}.alias"])
82: (12)                     content += module_alias_section
83: (12)                     module_alias_section += nodes.rubric(text="Type Aliases")
84: (12)                     for category_name, category_dict in module_alias_dict.items():
85: (16)                         category_section = nodes.section(
86: (20)                             ids=[category_name.lower().replace(" ", "_")]
87: (16)                         )
88: (16)                         module_alias_section += category_section
89: (16)                         if category_name:
90: (20)                             category_section += nodes.title(text=category_name)
91: (16)                             category_alias_container = nodes.container()
92: (16)                             category_section += category_alias_container
93: (16)                             for alias_name, alias_info in category_dict.items():
94: (20)                                 alias_def = alias_info["definition"]
95: (20)                                 for A in ALIAS_LIST:
96: (24)                                     alias_def = smart_replace(alias_def, A, f":class:`~.
{A}`")
97: (20)                                     unparsed = ViewList(
98: (24)   [
99: (28)   f".. class:: {alias_name}",
100: (28)   "",
101: (28)   "    .. parsed-literal::",
102: (28)   "",
103: (28)   f"        {alias_def}",
104: (28)   "",
105: (24)   ]
106: (20)   )
107: (20)   if "doc" in alias_info:
108: (24)   alias_doc = alias_info["doc"]

```

```

109: (24)                         for A in ALIAS_LIST:
110: (28)                           alias_doc = alias_doc.replace(f`{A}`",
f":class:`~.{A}`")
111: (24)                           doc_lines = alias_doc.split("\n")
112: (24)                           unparsed.extend(ViewList([f"    {line}" for line in
doc_lines]))
113: (20)                           alias_container = nodes.container()
114: (20)                           self.state.nested_parse(unparsed, 0, alias_container)
115: (20)                           category_alias_container += alias_container
116: (8)                            if module_attrs_list is not None:
117: (12)                              module_attrs_section = nodes.section(ids=[f"{module_name}.data"])
118: (12)                              content += module_attrs_section
119: (12)                              module_attrs_section += nodes.rubric(text="Module Attributes")
120: (12)                              unparsed = ViewList(
121: (16)                                [
122: (20)                                  "... autosummary::",
123: (20)                                  *(f"    {attr}" for attr in module_attrs_list),
124: (16)                                ]
125: (12)                            )
126: (12)                            data_container = nodes.container()
127: (12)                            self.state.nested_parse(unparsed, 0, data_container)
128: (12)                            module_attrs_section += data_container
129: (8)                            return [content]

```

---

File 137 - paths.py:

```

1: (0)         """Functions determining transformation paths between sets of points."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = [
4: (4)           "straight_path",
5: (4)           "path_along_arc",
6: (4)           "clockwise_path",
7: (4)           "counterclockwise_path",
8: (0)         ]
9: (0)         from typing import TYPE_CHECKING
10: (0)        import numpy as np
11: (0)        from ..constants import OUT
12: (0)        from ..utils.bezier import interpolate
13: (0)        from ..utils.space_ops import rotation_matrix
14: (0)        if TYPE_CHECKING:
15: (4)          from manim.typing import PathFuncType, Vector3D
16: (0)        STRAIGHT_PATH_THRESHOLD = 0.01
17: (0)        def straight_path():
18: (4)          """Simplest path function. Each point in a set goes in a straight path
toward its destination.
19: (4)          Examples
20: (4)          -----
21: (4)          .. manim :: StraightPathExample
22: (8)          class StraightPathExample(Scene):
23: (12)            def construct(self):
24: (16)              colors = [RED, GREEN, BLUE]
25: (16)              starting_points = VGroup(
26: (20)                *[
27: (24)                  Dot(LEFT + pos, color=color)
28: (24)                  for pos, color in zip([UP, DOWN, LEFT], colors)
29: (20)                ]
30: (16)            )
31: (16)            finish_points = VGroup(
32: (20)              *[
33: (24)                  Dot(RIGHT + pos, color=color)
34: (24)                  for pos, color in zip([ORIGIN, UP, DOWN], colors)
35: (20)                ]
36: (16)            )
37: (16)            self.add(starting_points)
38: (16)            self.add(finish_points)
39: (16)            for dot in starting_points:
40: (20)              self.add(TracedPath(dot.get_center,

```

12/20/24, 4:24 AM manims installed to implement with qhenomenology SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...

```

stroke_color=dot.get_color()))
41: (16)                         self.wait()
42: (16)                         self.play(
43: (20)                           Transform(
44: (24)                             starting_points,
45: (24)                             finish_points,
46: (24)                             path_func=utils.paths.straight_path(),
47: (24)                             run_time=2,
48: (20)                           )
49: (16)
50: (16)                         self.wait()
51: (4) """
52: (4)     return interpolate
53: (0) def path_along_circles(
54: (4)     arc_angle: float, circles_centers: np.ndarray, axis: Vector3D = OUT
55: (0) ) -> PathFuncType:
56: (4) """
57: (4)     """This function transforms each point by moving it roughly along a
58: (4)     circle, each with its own specified center.
59: (4)     The path may be seen as each point smoothly changing its orbit from its
60: (4)     starting position to its destination.
61: (4)             Parameters
62: (4)             -----
63: (4)             arc_angle
64: (8)                 The angle each point traverses around the quasicircle.
65: (4)             circles_centers
66: (8)                 The centers of each point's quasicircle to rotate around.
67: (4)             axis
68: (8)                 The axis of rotation.
69: (4)             Examples
70: (4)             -----
71: (8) .. manim :: PathAlongCirclesExample
72: (8)     class PathAlongCirclesExample(Scene):
73: (12)         def construct(self):
74: (16)             colors = [RED, GREEN, BLUE]
75: (16)             starting_points = VGroup(
76: (20)               *[[
77: (24)                   Dot(LEFT + pos, color=color)
78: (24)                   for pos, color in zip([UP, DOWN, LEFT], colors)
79: (20)               ]]
80: (16)
81: (16)
82: (20)
83: (16)
84: (16)
85: (16)
86: (16)
87: (20)
88: (16)
89: (16)
90: (16)
91: (16)
92: (20)
93: (24)
94: (24)
95: (24)
96: (28)
97: (24)
98: (24)
99: (20)
100: (16)
101: (16)
102: (4)
103: (4)
104: (8)
105: (4)
stroke_color=dot.get_color()))
106: (16)                         self.wait()
107: (16)                         self.play(
108: (20)                           Transform(
109: (24)                             starting_points,
110: (24)                             finish_points,
111: (24)                             path_func=utils.paths.path_along_circles(
112: (28)                               2 * PI, circle_center.get_center()
113: (24)                             ),
114: (24)                             run_time=3,
115: (20)                           )
116: (16)
117: (16)                         self.wait()
118: (4) """
119: (4)     if np.linalg.norm(axis) == 0:
120: (8)         axis = OUT
121: (4)     unit_axis = axis / np.linalg.norm(axis)

```

```

106: (4)             def path(start_points: np.ndarray, end_points: np.ndarray, alpha: float):
107: (8)                 detransformed_end_points = circles_centers + np.dot(
108: (12)                     end_points - circles_centers, rotation_matrix(-arc_angle,
109: (8)                         )
110: (8)                     rot_matrix = rotation_matrix(alpha * arc_angle, unit_axis)
111: (8)                     return circles_centers + np.dot(
112: (12)                         interpolate(start_points, detransformed_end_points, alpha)
113: (12)                         - circles_centers,
114: (12)                         rot_matrix.T,
115: (8)                     )
116: (4)             return path
117: (0)             def path_along_arc(arc_angle: float, axis: Vector3D = OUT) -> PathFuncType:
118: (4)                 """This function transforms each point by moving it along a circular arc.
119: (4)                 Parameters
120: (4)                 -----
121: (4)                 arc_angle
122: (8)                     The angle each point traverses around a circular arc.
123: (4)                 axis
124: (8)                     The axis of rotation.
125: (4)             Examples
126: (4)             -----
127: (4)             .. manim :: PathAlongArcExample
128: (8)             class PathAlongArcExample(Scene):
129: (12)                 def construct(self):
130: (16)                     colors = [RED, GREEN, BLUE]
131: (16)                     starting_points = VGroup(
132: (20)                         *[[
133: (24)                             Dot(LEFT + pos, color=color)
134: (24)                             for pos, color in zip([UP, DOWN, LEFT], colors)
135: (20)                         ]]
136: (16)
137: (16)
138: (20)
139: (24)
140: (24)
141: (20)
142: (16)
143: (16)
144: (16)
145: (16)
146: (20)
stroke_color=dot.get_color())))
147: (16)
148: (16)
149: (20)
150: (24)
151: (24)
152: (24)
153: (24)
154: (20)
155: (16)
156: (16)
157: (4)
158: (4)
159: (8)
160: (4)
161: (8)
162: (4)
163: (4)
164: (8)
165: (8)
166: (8)
167: (12)
2)
168: (8)
169: (8)
170: (4)
171: (0)
             """
if abs(arc_angle) < STRAIGHT_PATH_THRESHOLD:
    return straight_path()
if np.linalg.norm(axis) == 0:
    axis = OUT
unit_axis = axis / np.linalg.norm(axis)
def path(start_points: np.ndarray, end_points: np.ndarray, alpha: float):
    vекты = end_points - start_points
    centers = start_points + 0.5 * vекты
    if arc_angle != np.pi:
        centers += np.cross(unit_axis, vекты / 2.0) / np.tan(arc_angle /
rot_matrix = rotation_matrix(alpha * arc_angle, unit_axis)
return centers + np.dot(start_points - centers, rot_matrix.T)
return path
def clockwise_path() -> PathFuncType:

```

```

172: (4)             """This function transforms each point by moving clockwise around a half
circle.
173: (4)             Examples
174: (4)
175: (4)             .. manim :: ClockwisePathExample
176: (8)             class ClockwisePathExample(Scene):
177: (12)             def construct(self):
178: (16)                 colors = [RED, GREEN, BLUE]
179: (16)                 starting_points = VGroup(
180: (20)                     *[[
181: (24)                         Dot(LEFT + pos, color=color)
182: (24)                         for pos, color in zip([UP, DOWN, LEFT], colors)
183: (20)                     ]]
184: (16)
185: (16)                 finish_points = VGroup(
186: (20)                     *[[
187: (24)                         Dot(RIGHT + pos, color=color)
188: (24)                         for pos, color in zip([ORIGIN, UP, DOWN], colors)
189: (20)                     ]]
190: (16)
191: (16)                 self.add(starting_points)
192: (16)                 self.add(finish_points)
193: (16)                 for dot in starting_points:
194: (20)                     self.add(TracedPath(dot.get_center,
stroke_color=dot.get_color())))
195: (16)
196: (16)
197: (20)             self.wait()
198: (24)
199: (24)
200: (24)
201: (24)
202: (20)
203: (16)
204: (16)
205: (4)             """
206: (4)             return path_along_arc(-np.pi)
207: (0)             def counterclockwise_path() -> PathFuncType:
208: (4)                 """This function transforms each point by moving counterclockwise around a
half circle.
209: (4)             Examples
210: (4)
211: (4)             .. manim :: Counter-clockwisePathExample
212: (8)             class Counter-clockwisePathExample(Scene):
213: (12)             def construct(self):
214: (16)                 colors = [RED, GREEN, BLUE]
215: (16)                 starting_points = VGroup(
216: (20)                     *[[
217: (24)                         Dot(LEFT + pos, color=color)
218: (24)                         for pos, color in zip([UP, DOWN, LEFT], colors)
219: (20)                     ]]
220: (16)
221: (16)                 finish_points = VGroup(
222: (20)                     *[[
223: (24)                         Dot(RIGHT + pos, color=color)
224: (24)                         for pos, color in zip([ORIGIN, UP, DOWN], colors)
225: (20)                     ]]
226: (16)
227: (16)
228: (16)
229: (16)
230: (20)
stroke_color=dot.get_color())))
231: (16)
232: (16)
233: (20)
234: (24)
235: (24)
236: (24)

```

```

237: (24)                         run_time=2,
238: (20)                     )
239: (16)                 )
240: (16)             self.wait()
241: (4)             """
242: (4)         return path_along_arc(np.pi)
243: (0)     def spiral_path(angle: float, axis: Vector3D = OUT) -> PathFuncType:
244: (4)         """This function transforms each point by moving along a spiral to its
destination.

245: (4)             Parameters
246: (4)             -----
247: (4)             angle
248: (8)                 The angle each point traverses around a spiral.
249: (4)             axis
250: (8)                 The axis of rotation.
251: (4)             Examples
252: (4)             -----
253: (4)             .. manim :: SpiralPathExample
254: (8)             class SpiralPathExample(Scene):
255: (12)             def construct(self):
256: (16)                 colors = [RED, GREEN, BLUE]
257: (16)                 starting_points = VGroup(
258: (20)                     *[
259: (24)                         Dot(LEFT + pos, color=color)
260: (24)                         for pos, color in zip([UP, DOWN, LEFT], colors)
261: (20)                     ]
262: (16)
263: (16)                 finish_points = VGroup(
264: (20)                     *[
265: (24)                         Dot(RIGHT + pos, color=color)
266: (24)                         for pos, color in zip([ORIGIN, UP, DOWN], colors)
267: (20)                     ]
268: (16)
269: (16)
270: (16)
271: (16)
272: (20)             for dot in starting_points:
273: (16)                 self.add(TracedPath(dot.get_center,
stroke_color=dot.get_color())))
274: (16)             self.wait()
275: (20)             self.play(
276: (24)                 Transform(
277: (24)                     starting_points,
278: (24)                     finish_points,
279: (24)                     path_func=utils.paths.spiral_path(2 * TAU),
run_time=5,
280: (20)                 )
281: (16)
282: (16)             self.wait()
283: (4)             """
284: (4)             if abs(angle) < STRAIGHT_PATH_THRESHOLD:
285: (8)                 return straight_path()
286: (4)             if np.linalg.norm(axis) == 0:
287: (8)                 axis = OUT
288: (4)             unit_axis = axis / np.linalg.norm(axis)
289: (4)             def path(start_points: np.ndarray, end_points: np.ndarray, alpha: float):
290: (8)                 rot_matrix = rotation_matrix((alpha - 1) * angle, unit_axis)
291: (8)                 return start_points + alpha * np.dot(end_points - start_points,
rot_matrix.T)
292: (4)             return path

```

-----  
File 138 - family.py:

```

1: (0)             from __future__ import annotations
2: (0)             import itertools as it
3: (0)             from typing import Iterable
4: (0)             from ..mobject.mobject import Mobject
5: (0)             from ..utils.iterables import remove_list_redundancies

```

```

6: (0)             __all__ = ["extract_mobject_family_members"]
7: (0)             def extract_mobject_family_members(
8: (4)                 mobjects: Iterable[Mobject],
9: (4)                 use_z_index=False,
10: (4)                only_those_with_points: bool = False,
11: (0)            ):
12: (4)                """Returns a list of the types of mobjects and their family members
13: (4)                A "family" in this context refers to a mobject, its submobjects, and their
14: (4)                subsubobjects, recursively.
15: (4)                Parameters
16: (4)                    -----
17: (4)                mobjects
18: (8)                    The Mobjects currently in the Scene
19: (4)                only_those_with_points
20: (8)                    Whether or not to only do this for
21: (8)                    those mobjects that have points. By default False
22: (4)                Returns
23: (4)                    -----
24: (4)                list
25: (8)                    list of the mobjects and family members.
26: (4)                """
27: (4)                if only_those_with_points:
28: (8)                    method = Mobject.family_members_with_points
29: (4)                else:
30: (8)                    method = Mobject.get_family
31: (4)                extracted_mobjects = remove_list_redundancies(
32: (8)                    list(it.chain(*(method(m) for m in mobjects))),
33: (4)                )
34: (4)                if use_z_index:
35: (8)                    return sorted(extracted_mobjects, key=lambda m: m.z_index)
36: (4)                return extracted_mobjects

```

---

#### File 139 - images.py:

```

1: (0)             """Image manipulation utilities."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "get_full_raster_image_path",
5: (4)                 "drag_pixels",
6: (4)                 "invert_image",
7: (4)                 "change_to_rgba_array",
8: (0)             ]
9: (0)             from pathlib import Path
10: (0)            import numpy as np
11: (0)            from PIL import Image
12: (0)            from .. import config
13: (0)            from ..utils.file_ops import seek_full_path_from_defaults
14: (0)            def get_full_raster_image_path(image_file_name: str) -> Path:
15: (4)                return seek_full_path_from_defaults(
16: (8)                    image_file_name,
17: (8)                    default_dir=config.get_dir("assets_dir"),
18: (8)                    extensions=[".jpg", ".jpeg", ".png", ".gif", ".ico"],
19: (4)                )
20: (0)            def get_full_vector_image_path(image_file_name: str) -> Path:
21: (4)                return seek_full_path_from_defaults(
22: (8)                    image_file_name,
23: (8)                    default_dir=config.get_dir("assets_dir"),
24: (8)                    extensions=[".svg"],
25: (4)                )
26: (0)            def drag_pixels(frames: list[np.array]) -> list[np.array]:
27: (4)                curr = frames[0]
28: (4)                new_frames = []
29: (4)                for frame in frames:
30: (8)                    curr += (curr == 0) * np.array(frame)
31: (8)                    new_frames.append(np.array(curr))
32: (4)                return new_frames

```

```

33: (0)         def invert_image(image: np.array) -> Image:
34: (4)             arr = np.array(image)
35: (4)             arr = (255 * np.ones(arr.shape)).astype(arr.dtype) - arr
36: (4)             return Image.fromarray(arr)
37: (0)         def change_to_rgba_array(image, dtype="uint8"):
38: (4)             """Converts an RGB array into RGBA with the alpha value opacity maxed."""
39: (4)             pa = image
40: (4)             if len(pa.shape) == 2:
41: (8)                 pa = pa.reshape(list(pa.shape) + [1])
42: (4)             if pa.shape[2] == 1:
43: (8)                 pa = pa.repeat(3, axis=2)
44: (4)             if pa.shape[2] == 3:
45: (8)                 alphas = 255 * np.ones(
46: (12)                     list(pa.shape[:2]) + [1],
47: (12)                     dtype=dtype,
48: (8)                 )
49: (8)                 pa = np.append(pa, alphas, axis=2)
50: (4)             return pa
-----
```

## File 140 - opengl.py:

```

1: (0)         from __future__ import annotations
2: (0)         import numpy as np
3: (0)         import numpy.linalg as linalg
4: (0)         from .. import config
5: (0)         depth = 20
6: (0)         __all__ = [
7: (4)             "matrix_to_shader_input",
8: (4)             "orthographic_projection_matrix",
9: (4)             "perspective_projection_matrix",
10: (4)             "translation_matrix",
11: (4)             "x_rotation_matrix",
12: (4)             "y_rotation_matrix",
13: (4)             "z_rotation_matrix",
14: (4)             "rotate_in_place_matrix",
15: (4)             "rotation_matrix",
16: (4)             "scale_matrix",
17: (4)             "view_matrix",
18: (0)         ]
19: (0)         def matrix_to_shader_input(matrix):
20: (4)             return tuple(matrix.T.ravel())
21: (0)         def orthographic_projection_matrix(
22: (4)             width=None,
23: (4)             height=None,
24: (4)             near=1,
25: (4)             far=depth + 1,
26: (4)             format=True,
27: (0)         ):
28: (4)             if width is None:
29: (8)                 width = config["frame_width"]
30: (4)             if height is None:
31: (8)                 height = config["frame_height"]
32: (4)             projection_matrix = np.array(
33: (8)                 [
34: (12)                     [2 / width, 0, 0, 0],
35: (12)                     [0, 2 / height, 0, 0],
36: (12)                     [0, 0, -2 / (far - near), -(far + near) / (far - near)],
37: (12)                     [0, 0, 0, 1],
38: (8)                 ],
39: (4)             )
40: (4)             if format:
41: (8)                 return matrix_to_shader_input(projection_matrix)
42: (4)             else:
43: (8)                 return projection_matrix
44: (0)         def perspective_projection_matrix(width=None, height=None, near=2, far=50,
format=True):
45: (4)             if width is None:
```

```

46: (8)             width = config["frame_width"] / 6
47: (4)             if height is None:
48: (8)                 height = config["frame_height"] / 6
49: (4)             projection_matrix = np.array(
50: (8)                 [
51: (12)                     [2 * near / width, 0, 0, 0],
52: (12)                     [0, 2 * near / height, 0, 0],
53: (12)                     [0, 0, (far + near) / (near - far), (2 * far * near) / (near -
far)],
54: (12)                     [0, 0, -1, 0],
55: (8)                 ],
56: (4)             )
57: (4)             if format:
58: (8)                 return matrix_to_shader_input(projection_matrix)
59: (4)             else:
60: (8)                 return projection_matrix
61: (0)             def translation_matrix(x=0, y=0, z=0):
62: (4)                 return np.array(
63: (8)                 [
64: (12)                     [1, 0, 0, x],
65: (12)                     [0, 1, 0, y],
66: (12)                     [0, 0, 1, z],
67: (12)                     [0, 0, 0, 1],
68: (8)                 ],
69: (4)             )
70: (0)             def x_rotation_matrix(x=0):
71: (4)                 return np.array(
72: (8)                 [
73: (12)                     [1, 0, 0, 0],
74: (12)                     [0, np.cos(x), -np.sin(x), 0],
75: (12)                     [0, np.sin(x), np.cos(x), 0],
76: (12)                     [0, 0, 0, 1],
77: (8)                 ],
78: (4)             )
79: (0)             def y_rotation_matrix(y=0):
80: (4)                 return np.array(
81: (8)                 [
82: (12)                     [np.cos(y), 0, np.sin(y), 0],
83: (12)                     [0, 1, 0, 0],
84: (12)                     [-np.sin(y), 0, np.cos(y), 0],
85: (12)                     [0, 0, 0, 1],
86: (8)                 ],
87: (4)             )
88: (0)             def z_rotation_matrix(z=0):
89: (4)                 return np.array(
90: (8)                 [
91: (12)                     [np.cos(z), -np.sin(z), 0, 0],
92: (12)                     [np.sin(z), np.cos(z), 0, 0],
93: (12)                     [0, 0, 1, 0],
94: (12)                     [0, 0, 0, 1],
95: (8)                 ],
96: (4)             )
97: (0)             def rotate_in_place_matrix(initial_position, x=0, y=0, z=0):
98: (4)                 return np.matmul(
99: (8)                     translation_matrix(*-initial_position),
100: (8)                     np.matmul(
101: (12)                         rotation_matrix(x, y, z),
102: (12)                         translation_matrix(*initial_position),
103: (8)                     ),
104: (4)                 )
105: (0)             def rotation_matrix(x=0, y=0, z=0):
106: (4)                 return np.matmul(
107: (8)                     np.matmul(x_rotation_matrix(x), y_rotation_matrix(y)),
108: (8)                     z_rotation_matrix(z),
109: (4)                 )
110: (0)             def scale_matrix(scale_factor=1):
111: (4)                 return np.array(
112: (8)                 [
113: (12)                     [scale_factor, 0, 0, 0],

```

```

114: (12)                     [0, scale_factor, 0, 0],
115: (12)                     [0, 0, scale_factor, 0],
116: (12)                     [0, 0, 0, 1],
117: (8)                      ],
118: (4)
119: (0)          def view_matrix(
120: (4)            translation=None,
121: (4)            x_rotation=0,
122: (4)            y_rotation=0,
123: (4)            z_rotation=0,
124: (0)          ):
125: (4)            if translation is None:
126: (8)              translation = np.array([0, 0, depth / 2 + 1])
127: (4)            model_matrix = np.matmul(
128: (8)              np.matmul(
129: (12)                translation_matrix(*translation),
130: (12)                rotation_matrix(x=x_rotation, y=y_rotation, z=z_rotation),
131: (8)              ),
132: (8)              scale_matrix(),
133: (4)            )
134: (4)          return tuple(linalg.inv(model_matrix).T.ravel())

```

-----

File 141 - sounds.py:

```

1: (0)          """Sound-related utility functions."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = [
4: (4)            "get_full_sound_file_path",
5: (0)
6: (0)          from .. import config
7: (0)          from ..utils.file_ops import seek_full_path_from_defaults
8: (0)          def get_full_sound_file_path(sound_file_name):
9: (4)            return seek_full_path_from_defaults(
10: (8)              sound_file_name,
11: (8)              default_dir=config.get_dir("assets_dir"),
12: (8)              extensions=[".wav", ".mp3"],
13: (4)            )

```

-----

File 142 - hashing.py:

```

1: (0)          """Utilities for scene caching."""
2: (0)          from __future__ import annotations
3: (0)          import collections
4: (0)          import copy
5: (0)          import inspect
6: (0)          import json
7: (0)          import typing
8: (0)          import zlib
9: (0)          from time import perf_counter
10: (0)         from types import FunctionType, MappingProxyType, MethodType, ModuleType
11: (0)         from typing import Any
12: (0)         import numpy as np
13: (0)         from manim.animation.animation import Animation
14: (0)         from manim.camera.camera import Camera
15: (0)         from manim.mobject.mobject import Mobject
16: (0)         from .. import config, logger
17: (0)         if typing.TYPE_CHECKING:
18: (4)           from manim.scene.scene import Scene
19: (0)           __all__ = ["KEYS_TO_FILTER_OUT", "get_hash_from_play_call", "get_json"]
20: (0)           KEYS_TO_FILTER_OUT = {
21: (4)             "original_id",
22: (4)             "background",
23: (4)             "pixel_array",
24: (4)             "pixel_array_to_cairo_context",
25: (0)           }

```

```

26: (0)         class _Memoizer:
27: (4)             """Implements the memoization logic to optimize the hashing procedure and
prevent
28: (4)             the circular references within iterable processed.
29: (4)             Keeps a record of all the processed objects, and handle the logic to
return a place
30: (4)             holder instead of the original object if the object has already been
31: (4)             by the hashing logic (i.e, recursively checked, converted to JSON, etc...).
32: (4)             This class uses two signatures functions to keep a track of processed
objects :
33: (4)             hash or id. Whenever possible, hash is used to ensure a broader object
34: (4)             content-equality detection.
35: (4)             """
36: (4)             _already_processed = set()
37: (4)             _ALREADY_PROCESSED_PLACEHOLDER = "AP"
38: (4)             THRESHOLD_WARNING = 170_000
39: (4)             @classmethod
40: (4)             def reset_already_processed(cls):
41: (8)                 cls._already_processed.clear()
42: (4)             @classmethod
43: (4)             def check_already_processed_decorator(cls: _Memoizer, is_method: bool =
False):
44: (8)                 """Decorator to handle the arguments that goes through the decorated
function.
45: (8)                 or lets
46: (8)                 the decorated function call go ahead.
47: (8)                 Parameters
48: (8)                 -----
49: (8)                 is_method
50: (12)                     Whether the function passed is a method, by default False.
51: (8)
52: (8)
53: (12)             def layer(func):
54: (16)                 if is_method:
55: (20)                     return lambda self, obj: cls._handle_already_processed(
56: (20)                         obj,
57: (16)                         default_function=lambda obj: func(self, obj),
58: (12)                     )
59: (8)                     return lambda obj: cls._handle_already_processed(obj,
default_function=func)
60: (4)             return layer
61: (4)             @classmethod
62: (8)             def check_already_processed(cls, obj: Any) -> Any:
63: (8)                 """Checks if obj has been already processed. Returns itself if it has
not been,
64: (8)                 or the value of _ALREADY_PROCESSED_PLACEHOLDER if it has.
65: (8)                 Marks the object as processed in the second case.
66: (8)                 Parameters
67: (8)                 -----
68: (8)                 obj
69: (8)                     The object to check.
70: (8)                 Returns
71: (8)                 -----
72: (8)                 Any
73: (8)                     Either the object itself or the placeholder.
74: (8)                     """
75: (4)                     return cls._handle_already_processed(obj, lambda x: x)
76: (4)             @classmethod
77: (8)             def mark_as_processed(cls, obj: Any) -> None:
78: (8)                 """Marks an object as processed.
79: (8)                 Parameters
80: (8)                 -----
81: (8)                 obj
82: (8)                     The object to mark as processed.
83: (8)                     """
84: (8)                     cls._handle_already_processed(obj, lambda x: x)
85: (4)                     return cls._return(obj, id, lambda x: x, memoizing=False)

```

```

86: (4)             def _handle_already_processed(
87: (8)                 cls,
88: (8)                 obj,
89: (8)                 default_function: typing.Callable[[Any], Any],
90: (4)             ):
91: (8)                 if isinstance(
92: (12)                     obj,
93: (12)                     (
94: (16)                         int,
95: (16)                         float,
96: (16)                         str,
97: (16)                         complex,
98: (12)                     ),
99: (8)                 ) and obj not in [None, cls.ALREADY_PROCESSED_PLACEHOLDER]:
100: (12)                     return obj
101: (8)             if isinstance(obj, collections.abc.Hashable):
102: (12)                 try:
103: (16)                     return cls._return(obj, hash, default_function)
104: (12)                 except TypeError:
105: (16)                     pass
106: (8)                 return cls._return(obj, id, default_function)
107: (4)             @classmethod
108: (4)             def _return(
109: (8)                 cls,
110: (8)                 obj: typing.Any,
111: (8)                 obj_to_membership_sign: typing.Callable[[Any], int],
112: (8)                 default_func,
113: (8)                 memoizing=True,
114: (4)             ) -> str | Any:
115: (8)                 obj_membership_sign = obj_to_membership_sign(obj)
116: (8)                 if obj_membership_sign in cls._already_processed:
117: (12)                     return cls.ALREADY_PROCESSED_PLACEHOLDER
118: (8)                 if memoizing:
119: (12)                     if (
120: (16)                         not config.disable_caching_warning
121: (16)                         and len(cls._already_processed) == cls.THRESHOLD_WARNING
122: (12)                     ):
123: (16)                         logger.warning(
124: (20)                             "It looks like the scene contains a lot of sub-mobjects.
Caching "
125: (20)                             "is sometimes not suited to handle such large scenes, you
might "
126: (20)                             "consider disabling caching with --disable_caching to
potentially "
127: (20)                             "speed up the rendering process.",
128: (16)
129: (16)
130: (20)
 disable_caching_warning "
131: (20)
132: (16)
133: (12)
134: (8)
135: (0)
136: (4)
137: (8)
138: (8)
139: (8)
 'code', for
140: (8)
 nonlocals
141: (8)
142: (8)
143: (8)
144: (8)
 type is
145: (8)
146: (8)
147: (8)
                         This method is used to serialize objects to JSON format.
                         If obj is a function, then it will return a dict with two keys :
                         the code source, and 'nonlocals' for all nonlocalsvalues. (including
                         functions, that will be serialized as this is recursive.)
                         if obj is a np.ndarray, it converts it into a list.
                         if obj is an object with __dict__ attribute, it returns its __dict__.
                         Else, will let the JSONEncoder do the stuff, and throw an error if the
                         not suitable for JSONEncoder.
                         Parameters
                         -----

```

```

148: (8)           obj
149: (12)          Arbitrary object to convert
150: (8)           Returns
151: (8)           -----
152: (8)           Any
153: (12)          Python object that JSON encoder will recognize
154: (8)           """
155: (8)           if not (isinstance(obj, ModuleType)) and isinstance(
156: (12)             obj,
157: (12)             (MethodType, FunctionType),
158: (8)           ):
159: (12)             cvars = inspect.getclosurevars(obj)
160: (12)             cvardict = {**copy.copy(cvars.globals),
**copy.copy(cvars.nonlocals)}
161: (12)
162: (16)            for i in list(cvardict):
163: (20)              if isinstance(cvardict[i], ModuleType):
164: (12)                del cvardict[i]
165: (16)            try:
166: (12)              code = inspect.getsource(obj)
167: (16)            except (OSError, TypeError):
168: (12)              code = ""
169: (12)            return self._cleaned_iterable({"code": code, "nonlocals": cvardict})
170: (8)
171: (12)           elif isinstance(obj, np.ndarray):
172: (16)             if obj.size > 1000:
173: (12)               obj = np.resize(obj, (100, 100))
174: (16)               return f"TRUNCATED ARRAY: {repr(obj)}"
175: (12)             return repr(obj)
176: (12)           elif hasattr(obj, "__dict__"):
177: (16)             temp = getattr(obj, "__dict__")
178: (12)             if isinstance(temp, MappingProxyType):
179: (16)               return "MappingProxy"
180: (12)             return self._cleaned_iterable(temp)
181: (8)           elif isinstance(obj, np.uint8):
182: (12)             return int(obj)
183: (8)           return str(type(obj))
184: (4)           def _cleaned_iterable(self, iterable: typing.Iterable[Any]):
185: (8)             """Check for circular reference at each iterable that will go through
the JSONEncoder, as well as key of the wrong format.
186: (8)             If a key with a bad format is found (i.e not a int, string, or float),
it gets replaced by its hash using the same process implemented here.
187: (8)             If a circular reference is found within the iterable, it will be
replaced by the string "already processed".
188: (8)           Parameters
189: (12)           -----
190: (8)           iterable
191: (12)             The iterable to check.
192: (12)           """
193: (8)           def _key_to_hash(key):
194: (12)             return zlib.crc32(json.dumps(key, cls=_CustomEncoder).encode())
195: (12)           def _iter_check_list(lst):
196: (16)             processed_list = [None] * len(lst)
197: (16)             for i, el in enumerate(lst):
198: (20)               el = _Memoizer.check_already_processed(el)
199: (16)               if isinstance(el, (list, tuple)):
200: (20)                 new_value = _iter_check_list(el)
201: (16)               elif isinstance(el, dict):
202: (20)                 new_value = _iter_check_dict(el)
203: (16)               else:
204: (20)                 new_value = el
205: (12)             processed_list[i] = new_value
206: (12)             return processed_list
207: (12)           def _iter_check_dict(dct):
208: (16)             processed_dict = {}
209: (16)             for k, v in dct.items():
210: (20)               v = _Memoizer.check_already_processed(v)
211: (16)               if k in KEYS_TO_FILTER_OUT:
212: (20)                 continue
213: (16)               if not isinstance(k, (str, int, float, bool)) and k is not

```

None:

```

212: (20)                 k_new = _key_to_hash(k)
213: (16)               else:
214: (20)                 k_new = k
215: (16)               if isinstance(v, dict):
216: (20)                 new_value = _iter_check_dict(v)
217: (16)               elif isinstance(v, (list, tuple)):
218: (20)                 new_value = _iter_check_list(v)
219: (16)               else:
220: (20)                 new_value = v
221: (16)               processed_dict[k_new] = new_value
222: (12)             return processed_dict
223: (8)               if isinstance(iterable, (list, tuple)):
224: (12)                 return _iter_check_list(iterable)
225: (8)               elif isinstance(iterable, dict):
226: (12)                 return _iter_check_dict(iterable)
227: (4)               def encode(self, obj: Any):
228: (8)                 """Overriding of :meth:`JSONEncoder.encode`, to make our own process.
229: (8)               Parameters
230: (8)               -----
231: (8)               obj
232: (12)                 The object to encode in JSON.
233: (8)               Returns
234: (8)               -----
235: (8)               :class:`str`
236: (11)                 The object encoder with the standard json process.
237: (8)               """
238: (8)               _Memoizer.mark_as_processed(obj)
239: (8)               if isinstance(obj, (dict, list, tuple)):
240: (12)                 return super().encode(self._cleaned_iterable(obj))
241: (8)               return super().encode(obj)
242: (0)             def get_json(obj: dict):
243: (4)               """Recursively serialize `object` to JSON using the :class:`CustomEncoder`  

244: (4)             class.
245: (4)               Parameters
246: (4)               -----
247: (8)               obj
248: (4)                 The dict to flatten
249: (4)               Returns
250: (4)               -----
251: (8)               :class:`str`
252: (4)                 The flattened object
253: (4)               """
254: (0)               return json.dumps(obj, cls=_CustomEncoder)
255: (4)             def get_hash_from_play_call(
256: (4)               scene_object: Scene,
257: (4)               camera_object: Camera,
258: (4)               animations_list: typing.Iterable[Animation],
259: (0)               current_mobjects_list: typing.Iterable[Mobject],
260: (4)             ) -> str:
261: (4)               """Take the list of animations and a list of mobjects and output their
262: (4)               hashes. This is meant to be used for `scene.play` function.
263: (4)               Parameters
264: (4)               -----
265: (4)               scene_object
266: (8)                 The scene object.
267: (4)               camera_object
268: (8)                 The camera object used in the scene.
269: (4)               animations_list
270: (8)                 The list of animations.
271: (4)               current_mobjects_list
272: (8)                 The list of mobjects.
273: (4)               Returns
274: (4)               -----
275: (8)               :class:`str`
276: (4)                 A string concatenation of the respective hashes of `camera_object`,
`animations_list` and `current_mobjects_list`, separated by `_`.
277: (4)               """
278: (4)               logger.debug("Hashing ...")

```

```

277: (4)             t_start = perf_counter()
278: (4)             _Memoizer.mark_as_processed(scene_object)
279: (4)             camera_json = get_json(camera_object)
280: (4)             animations_list_json = [get_json(x) for x in sorted(animations_list,
key=str)]
281: (4)             current_mobjects_list_json = [get_json(x) for x in current_mobjects_list]
282: (4)             hash_camera, hash_animations, hash_current_mobjects = (
283: (8)                 zlib.crc32(repr(json_val).encode())
284: (8)                 for json_val in [camera_json, animations_list_json,
current_mobjects_list_json]
285: (4)             )
286: (4)             hash_complete = f"{hash_camera}_{hash_animations}_{hash_current_mobjects}"
287: (4)             t_end = perf_counter()
288: (4)             logger.debug("Hashing done in %(time)s s.", {"time": str(t_end - t_start)
[:8]})
289: (4)             _Memoizer.reset_already_processed()
290: (4)             logger.debug("Hash generated : %h", {"h": hash_complete})
291: (4)             return hash_complete
-----
```

## File 143 - file\_ops.py:

```

1: (0)         """Utility functions for interacting with the file system."""
2: (0)         from __future__ import annotations
3: (0)         __all__ = [
4: (4)             "add_extension_if_not_present",
5: (4)             "guarantee_existence",
6: (4)             "guarantee_empty_existence",
7: (4)             "seek_full_path_from_defaults",
8: (4)             "modify_atime",
9: (4)             "open_file",
10: (4)            "is_mp4_format",
11: (4)            "is_gif_format",
12: (4)            "is_png_format",
13: (4)            "is_webm_format",
14: (4)            "is_mov_format",
15: (4)            "write_to_movie",
16: (4)            "ensure_executable",
17: (0)        ]
18: (0)        import os
19: (0)        import platform
20: (0)        import shutil
21: (0)        import subprocess as sp
22: (0)        import time
23: (0)        from pathlib import Path
24: (0)        from shutil import copyfile
25: (0)        from typing import TYPE_CHECKING
26: (0)        if TYPE_CHECKING:
27: (4)            from ..scene.scene_file_writer import SceneFileWriter
28: (0)        from manim import __version__, config, logger
29: (0)        from .. import config, console
30: (0)        def is_mp4_format() -> bool:
31: (4)            """
32: (4)                Determines if output format is .mp4
33: (4)                Returns
34: (4)                -----
35: (4)                class:`bool`
36: (8)                ``True`` if format is set as mp4
37: (4)                """
38: (4)                return config["format"] == "mp4"
39: (0)        def is_gif_format() -> bool:
40: (4)            """
41: (4)                Determines if output format is .gif
42: (4)                Returns
43: (4)                -----
44: (4)                class:`bool`
45: (8)                ``True`` if format is set as gif
46: (4)                """
-----
```

```

47: (4)             return config["format"] == "gif"
48: (0)         def is_webm_format() -> bool:
49: (4)             """
50: (4)                 Determines if output format is .webm
51: (4)                 Returns
52: (4)                 -----
53: (4)                 class:`bool`
54: (8)                     ``True`` if format is set as webm
55: (4)                 """
56: (4)             return config["format"] == "webm"
57: (0)         def is_mov_format() -> bool:
58: (4)             """
59: (4)                 Determines if output format is .mov
60: (4)                 Returns
61: (4)                 -----
62: (4)                 class:`bool`
63: (8)                     ``True`` if format is set as mov
64: (4)                 """
65: (4)             return config["format"] == "mov"
66: (0)         def is_png_format() -> bool:
67: (4)             """
68: (4)                 Determines if output format is .png
69: (4)                 Returns
70: (4)                 -----
71: (4)                 class:`bool`
72: (8)                     ``True`` if format is set as png
73: (4)                 """
74: (4)             return config["format"] == "png"
75: (0)         def write_to_movie() -> bool:
76: (4)             """
77: (4)                 Determines from config if the output is a video format such as mp4 or gif,
if the --format is set as 'png'
78: (4)                     then it will take precedence even if the write_to_movie flag is set
79: (4)                 Returns
80: (4)                 -----
81: (4)                 class:`bool`
82: (8)                     ``True`` if the output should be written in a movie format
83: (4)                 """
84: (4)             if is_png_format():
85: (8)                 return False
86: (4)             return (
87: (8)                 config["write_to_movie"]
88: (8)                 or is_mp4_format()
89: (8)                 or is_gif_format()
90: (8)                 or is_webm_format()
91: (8)                 or is_mov_format()
92: (4)             )
93: (0)         def ensure_executable(path_to_exe: Path) -> bool:
94: (4)             if path_to_exe.parent == Path("."):
95: (8)                 executable = shutil.which(path_to_exe.stem)
96: (8)                 if executable is None:
97: (12)                     return False
98: (4)             else:
99: (8)                 executable = path_to_exe
100: (4)             return os.access(executable, os.X_OK)
101: (0)         def add_extension_if_not_present(file_name: Path, extension: str) -> Path:
102: (4)             if file_name.suffix != extension:
103: (8)                 return file_name.with_suffix(file_name.suffix + extension)
104: (4)             else:
105: (8)                 return file_name
106: (0)         def add_version_before_extension(file_name: Path) -> Path:
107: (4)             return file_name.with_name(
108: (8)                 f"{file_name.stem}_ManimCE_v{__version__}{file_name.suffix}"
109: (4)             )
110: (0)         def guarantee_existence(path: Path) -> Path:
111: (4)             if not path.exists():
112: (8)                 path.mkdir(parents=True)
113: (4)             return path.resolve(strict=True)
114: (0)         def guarantee_empty_existence(path: Path) -> Path:

```

```

115: (4)             if path.exists():
116: (8)                 shutil.rmtree(str(path))
117: (4)             path.mkdir(parents=True)
118: (4)             return path.resolve(strict=True)
119: (0)         def seek_full_path_from_defaults(
120: (4)             file_name: str, default_dir: Path, extensions: list[str]
121: (0)         ) -> Path:
122: (4)             possible_paths = [Path(file_name).expanduser()]
123: (4)             possible_paths += [
124: (8)                 Path(default_dir) / f"{file_name}{extension}" for extension in ["",
*extensions]
125: (4)             ]
126: (4)             for path in possible_paths:
127: (8)                 if path.exists():
128: (12)                     return path
129: (4)             error = (
130: (8)                 f"From: {Path.cwd()}, could not find {file_name} at either "
131: (8)                 f"of these locations: {list(map(str, possible_paths))}"
132: (4)             )
133: (4)             raise OSError(error)
134: (0)         def modify_atime(file_path: str) -> None:
135: (4)             """Will manually change the accessed time (called `atime`) of the file, as
on a lot of OS the accessed time refresh is disabled by default.
136: (4)             Parameters
137: (4)             -----
138: (4)             file_path
139: (8)                 The path of the file.
140: (4)             """
141: (4)             os.utime(file_path, times=(time.time(), Path(file_path).stat().st_mtime))
142: (0)         def open_file(file_path, in_browser=False):
143: (4)             current_os = platform.system()
144: (4)             if current_os == "Windows":
145: (8)                 os.startfile(file_path if not in_browser else file_path.parent)
146: (4)             else:
147: (8)                 if current_os == "Linux":
148: (12)                     commands = ["xdg-open"]
149: (12)                     file_path = file_path if not in_browser else file_path.parent
150: (8)                 elif current_os.startswith("CYGWIN"):
151: (12)                     commands = ["cygstart"]
152: (12)                     file_path = file_path if not in_browser else file_path.parent
153: (8)                 elif current_os == "Darwin":
154: (12)                     if is_gif_format():
155: (16)                         commands = ["ffplay", "-loglevel",
config["ffmpeg_loglevel"].lower()]
156: (12)                     else:
157: (16)                         commands = ["open"] if not in_browser else ["open", "-R"]
158: (8)                 else:
159: (12)                     raise OSError("Unable to identify your operating system...")
160: (8)             if config.preview_command:
161: (12)                 commands = [config.preview_command]
162: (8)                 commands.append(file_path)
163: (8)                 sp.run(commands)
164: (0)         def open_media_file(file_writer: SceneFileWriter) -> None:
165: (4)             file_paths = []
166: (4)             if config["save_last_frame"]:
167: (8)                 file_paths.append(file_writer.image_file_path)
168: (4)             if write_to_movie() and not is_gif_format():
169: (8)                 file_paths.append(file_writer.movie_file_path)
170: (4)             if write_to_movie() and is_gif_format():
171: (8)                 file_paths.append(file_writer.gif_file_path)
172: (4)             for file_path in file_paths:
173: (8)                 if config["show_in_file_browser"]:
174: (12)                     open_file(file_path, True)
175: (8)                 if config["preview"]:
176: (12)                     open_file(file_path, False)
177: (8)                     logger.info(f"Previewed File at: '{file_path}'")
178: (0)         def get_template_names() -> list[str]:
179: (4)             """Returns template names from the templates directory.
180: (4)             Returns

```

```

181: (4)             -----
182: (8)             :class:`list`  

183: (4)  

184: (4)             template_path = Path.resolve(Path(__file__).parent.parent / "templates")  

185: (4)             return [template_name.stem for template_name in  

template_path.glob("*.mtp")]
186: (0)             def get_template_path() -> Path:  

187: (4)             """Returns the Path of templates directory.  

188: (4)             Returns
189: (4)  

190: (8)             -----
191: (4)             :class:`Path`  

192: (4)  

193: (0)             def add_import_statement(file: Path):  

194: (4)             """Prepends an import statement in a file  

195: (4)             Parameters
196: (4)  

197: (8)             file
198: (4)  

199: (4)             """
200: (8)             with file.open("r+") as f:
201: (8)                 import_line = "from manim import *"
202: (8)                 content = f.read()
203: (8)                 f.seek(0)
204: (8)                 f.write(import_line + "\n" + content)
205: (0)             def copy_template_files(
206: (4)                 project_dir: Path = Path("."), template_name: str = "Default"
207: (0)             ) -> None:
208: (4)             """Copies template files from templates dir to project_dir.  

209: (4)             Parameters
210: (4)  

211: (8)             project_dir
212: (8)                 Path to project directory.
213: (12)             template_name
214: (4)                 Name of template.
215: (4)  

216: (8)             template_cfg_path = Path.resolve(
217: (4)                 Path(__file__).parent.parent / "templates/template.cfg",
218: (4)             )
219: (8)             template_scene_path = Path.resolve(
220: (4)                 Path(__file__).parent.parent / f"templates/{template_name}.mtp",
221: (4)             )
222: (8)             if not template_cfg_path.exists():
223: (4)                 raise FileNotFoundError(f"{template_cfg_path} : file does not exist")
224: (8)             if not template_scene_path.exists():
225: (4)                 raise FileNotFoundError(f"{template_scene_path} : file does not
exist")
226: (4)             copyfile(template_cfg_path, Path.resolve(project_dir / "manim.cfg"))
227: (4)             console.print("\n\t[green]copied[/green] [blue]manim.cfg[/blue]\n")
228: (4)             copyfile(template_scene_path, Path.resolve(project_dir / "main.py"))
229: (4)             console.print("\n\t[green]copied[/green] [blue]main.py[/blue]\n")
230: (4)             add_import_statement(Path.resolve(project_dir / "main.py"))

-----

```

## File 144 - iterables.py:

```

1: (0)             """Operations on iterables."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "adjacent_n_tuples",
5: (4)                 "adjacent_pairs",
6: (4)                 "all_elements_are_instances",
7: (4)                 "concatenate_lists",
8: (4)                 "list_difference_update",
9: (4)                 "list_update",
10: (4)                "listify",
11: (4)                "make_even",
12: (4)                "make_even_by_cycling",
13: (4)                "remove_list_redundancies",

```

```

14: (4)             "remove_nones",
15: (4)             "stretch_array_to_length",
16: (4)             "tuplify",
17: (0)
18: (0)         ]
19: (0)         import itertools as it
20: (0)         from typing import Any, Callable, Collection, Generator, Iterable, Reversible,
Sequence
21: (0)         import numpy as np
22: (0)         def adjacent_n_tuples(objects: Sequence, n: int) -> zip:
23: (4)             """Returns the Sequence objects cyclically split into n length tuples.
See Also
-----
25: (4)             adjacent_pairs : alias with n=2
26: (4)         Examples
27: (4)         -----
28: (4)             Normal usage::
29: (8)                 list(adjacent_n_tuples([1, 2, 3, 4], 2))
30: (8)                 list(adjacent_n_tuples([1, 2, 3, 4], 3))
31: (4)             """
32: (4)             return zip(*([*objects[k:], *objects[:k]] for k in range(n)))
33: (0)         def adjacent_pairs(objects: Sequence) -> zip:
34: (4)             """Alias for ``adjacent_n_tuples(objects, 2)``.
See Also
-----
37: (4)             adjacent_n_tuples
38: (4)         Examples
39: (4)         -----
40: (4)             Normal usage::
41: (8)                 list(adjacent_pairs([1, 2, 3, 4]))
42: (4)             """
43: (4)             return adjacent_n_tuples(objects, 2)
44: (0)         def all_elements_are_instances(iterable: Iterable, Class) -> bool:
45: (4)             """Returns ``True`` if all elements of iterable are instances of Class.
False otherwise.
"""
46: (4)             return all(isinstance(e, Class) for e in iterable)
47: (4)
48: (4)
49: (0)         def batch_by_property(
50: (4)             items: Sequence, property_func: Callable
51: (0)         ) -> list[tuple[list, Any]]:
52: (4)             """Takes in a Sequence, and returns a list of tuples, (batch, prop)
such that all items in a batch have the same output when
put into the Callable property_func, and such that chaining all these
batches together would give the original Sequence (i.e. order is
preserved).
See Also
-----
59: (4)             Normal usage::
60: (8)                 batch_by_property([(1, 2), (3, 4), (5, 6, 7), (8, 9)], len)
61: (4)             """
62: (4)             batch_prop_pairs = []
63: (4)             curr_batch = []
64: (4)             curr_prop = None
65: (4)             for item in items:
66: (8)                 prop = property_func(item)
67: (8)                 if prop != curr_prop:
68: (12)                     if len(curr_batch) > 0:
69: (16)                         batch_prop_pairs.append((curr_batch, curr_prop))
70: (12)                     curr_prop = prop
71: (12)                     curr_batch = [item]
72: (8)                 else:
73: (12)                     curr_batch.append(item)
74: (4)                 if len(curr_batch) > 0:
75: (8)                     batch_prop_pairs.append((curr_batch, curr_prop))
76: (4)             return batch_prop_pairs
77: (0)         def concatenate_lists(*list_of_lists: Iterable) -> list:
78: (4)             """Combines the Iterables provided as arguments into one list.
See Also
-----
81: (4)             Normal usage::

```

```

82: (8)           concatenate_lists([1, 2], [3, 4], [5])
83: (4)
84: (4)           """
85: (0)           return [item for lst in list_of_lists for item in lst]
86: (4)           def list_difference_update(l1: Iterable, l2: Iterable) -> list:
87: (4)               """Returns a list containing all the elements of l1 not in l2.
88: (4)               Examples
89: (4)               -----
90: (8)               Normal usage::
91: (4)                   list_difference_update([1, 2, 3, 4], [2, 4])
92: (4)                   """
93: (0)           return [e for e in l1 if e not in l2]
94: (4)           def list_update(l1: Iterable, l2: Iterable) -> list:
95: (8)               """Used instead of ``set.update()`` to maintain order,
96: (4)                   making sure duplicates are removed from l1, not l2.
97: (8)                   Removes overlap of l1 and l2 and then concatenates l2 unchanged.
98: (4)               Examples
99: (4)               -----
100: (8)              Normal usage::
101: (4)                  list_update([1, 2, 3], [2, 4, 4])
102: (4)                  """
103: (0)              return [e for e in l1 if e not in l2] + list(l2)
104: (4)           def listify(obj) -> list:
105: (4)               """Converts obj to a list intelligently.
106: (4)               Examples
107: (4)               -----
108: (8)               Normal usage::
109: (8)                   listify('str')    # ['str']
110: (8)                   listify((1, 2))  # [1, 2]
111: (4)                   listify(len)     # [<built-in function len>]
112: (4)
113: (8)               if isinstance(obj, str):
114: (4)                   return [obj]
115: (4)               try:
116: (8)                   return list(obj)
117: (4)               except TypeError:
118: (8)                   return [obj]
119: (0)           def make_even(iterable_1: Iterable, iterable_2: Iterable) -> tuple[list,
list]:
120: (4)
121: (4)               """Extends the shorter of the two iterables with duplicate values until
its
122: (4)               length is equal to the longer iterable (favours earlier elements).
See Also
123: (4)               make_even_by_cycling : cycles elements instead of favouring earlier ones
124: (4)               Examples
125: (4)
126: (4)               Normal usage::
127: (8)                   make_even([1, 2], [3, 4, 5, 6])
128: (8)                   ([1, 1, 2, 2], [3, 4, 5, 6])
129: (8)                   make_even([1, 2], [3, 4, 5, 6, 7])
130: (4)
131: (4)               list_1, list_2 = list(iterable_1), list(iterable_2)
132: (4)               len_list_1 = len(list_1)
133: (4)               len_list_2 = len(list_2)
134: (4)               length = max(len_list_1, len_list_2)
135: (4)               return (
136: (8)                   [list_1[(n * len_list_1) // length] for n in range(length)],
137: (8)                   [list_2[(n * len_list_2) // length] for n in range(length)],
138: (4)               )
139: (0)           def make_even_by_cycling(
140: (4)               iterable_1: Collection, iterable_2: Collection
141: (0)           ) -> tuple[list, list]:
142: (4)
143: (8)               """Extends the shorter of the two iterables with duplicate values until
its
144: (4)               length is equal to the longer iterable (cycles over shorter iterable).
See Also
145: (4)
146: (4)               make_even : favours earlier elements instead of cycling them
147: (4)               Examples

```

```

148: (4)      -----
149: (4)      Normal usage::
150: (8)      make_even_by_cycling([1, 2], [3, 4, 5, 6])
151: (8)      ([1, 2, 1, 2], [3, 4, 5, 6])
152: (8)      make_even_by_cycling([1, 2], [3, 4, 5, 6, 7])
153: (4)      """
154: (4)      length = max(len(iterable_1), len(iterable_2))
155: (4)      cycle1 = it.cycle(iterable_1)
156: (4)      cycle2 = it.cycle(iterable_2)
157: (4)      return (
158: (8)          [next(cycle1) for _ in range(length)],
159: (8)          [next(cycle2) for _ in range(length)],
160: (4)      )
161: (0)      def remove_list_redundancies(lst: Reversible) -> list:
162: (4)      """Used instead of ``list(set(l))`` to maintain order.
163: (4)      Keeps the last occurrence of each element.
164: (4)      """
165: (4)      reversed_result = []
166: (4)      used = set()
167: (4)      for x in reversed(lst):
168: (8)          if x not in used:
169: (12)              reversed_result.append(x)
170: (12)              used.add(x)
171: (4)      reversed_result.reverse()
172: (4)      return reversed_result
173: (0)      def remove_nones(sequence: Iterable) -> list:
174: (4)      """Removes elements where bool(x) evaluates to False.
175: (4)      Examples
176: (4)      -----
177: (4)      Normal usage::
178: (8)          remove_nones(['m', '', 'l', 0, 42, False, True])
179: (4)      """
180: (4)      return [x for x in sequence if x]
181: (0)      def resize_array(nparray: np.ndarray, length: int) -> np.ndarray:
182: (4)      """Extends/truncates nparray so that `len(result) == length`.
183: (8)          The elements of nparray are cycled to achieve the desired length.
184: (4)      See Also
185: (4)      -----
186: (4)          resize_preserving_order : favours earlier elements instead of cycling them
187: (4)          make_even_by_cycling : similar cycling behaviour for balancing 2 iterables
188: (4)      Examples
189: (4)      -----
190: (4)      Normal usage::
191: (8)          >>> points = np.array([[1, 2], [3, 4]])
192: (8)          >>> resize_array(points, 1)
193: (8)          array([[1, 2]])
194: (8)          >>> resize_array(points, 3)
195: (8)          array([[1, 2],
196: (15)              [3, 4],
197: (15)              [1, 2]])
198: (8)          >>> resize_array(points, 2)
199: (8)          array([[1, 2],
200: (15)              [3, 4]])
201: (4)      """
202: (4)      if len(nparray) == length:
203: (8)          return nparray
204: (4)      return np.resize(nparray, (length, *nparray.shape[1:]))
205: (0)      def resize_preserving_order(nparray: np.ndarray, length: int) -> np.ndarray:
206: (4)      """Extends/truncates nparray so that `len(result) == length`.
207: (8)          The elements of nparray are duplicated to achieve the desired length
208: (8)          (favours earlier elements).
209: (8)          Constructs a zeroes array of length if nparray is empty.
210: (4)      See Also
211: (4)      -----
212: (4)          resize_array : cycles elements instead of favouring earlier ones
213: (4)          make_even : similar earlier-favouring behaviour for balancing 2 iterables
214: (4)      Examples
215: (4)      -----
216: (4)      Normal usage:::
```

```

217: (8)             resize_preserving_order(np.array([]), 5)
218: (8)             nparray = np.array([[1, 2],
219: (28)                         [3, 4]])
220: (8)             resize_preserving_order(nparray, 1)
221: (8)             resize_preserving_order(nparray, 3)
222: (4)             """
223: (4)             if len(nparray) == 0:
224: (8)                 return np.zeros((length, *nparray.shape[1:]))
225: (4)             if len(nparray) == length:
226: (8)                 return nparray
227: (4)             indices = np.arange(length) * len(nparray) // length
228: (4)             return nparray[indices]
229: (0)             def resize_with_interpolation(nparray: np.ndarray, length: int) -> np.ndarray:
230: (4)                 """Extends/truncates nparray so that `len(result)` == length``.
231: (8)                     New elements are interpolated to achieve the desired length.
232: (8)                     Note that if nparray's length changes, its dtype may too
233: (8)                         (e.g. int -> float: see Examples)
234: (4)             See Also
235: (4)             -----
236: (4)             resize_array : cycles elements instead of interpolating
237: (4)             resize_preserving_order : favours earlier elements instead of
interpolating
238: (4)             Examples
239: (4)             -----
240: (4)             Normal usage:::
241: (8)                 nparray = np.array([[1, 2],
242: (28)                         [3, 4]])
243: (8)                 resize_with_interpolation(nparray, 1)
244: (8)                 resize_with_interpolation(nparray, 4)
245: (8)                 nparray = np.array([[[1, 2],[3, 4]]])
246: (8)                 resize_with_interpolation(nparray, 3)
247: (8)                 nparray = np.array([[1, 2], [3, 4], [5, 6]])
248: (8)                 resize_with_interpolation(nparray, 4)
249: (8)                 nparray = np.array([[1, 2], [3, 4], [1, 2]])
250: (8)                 resize_with_interpolation(nparray, 4)
251: (4)             """
252: (4)             if len(nparray) == length:
253: (8)                 return nparray
254: (4)             cont_indices = np.linspace(0, len(nparray) - 1, length)
255: (4)             return np.array(
256: (8)                 [
257: (12)                     (1 - a) * nparray[lh] + a * nparray[rh]
258: (12)                     for ci in cont_indices
259: (12)                         for lh, rh, a in [(int(ci), int(np.ceil(ci)), ci % 1)]
260: (8)                 ],
261: (4)             )
262: (0)             def stretch_array_to_length(nparray: np.ndarray, length: int) -> np.ndarray:
263: (4)                 curr_len = len(nparray)
264: (4)                 if curr_len > length:
265: (8)                     raise Warning("Trying to stretch array to a length shorter than its
own")
266: (4)                 indices = np.arange(length) / float(length)
267: (4)                 indices *= curr_len
268: (4)                 return nparray[indices.astype(int)]
269: (0)             def tuplify(obj) -> tuple:
270: (4)                 """Converts obj to a tuple intelligently.
271: (4)                 Examples
272: (4)                 -----
273: (4)                 Normal usage:::
274: (8)                     tuplify('str')    # ('str',)
275: (8)                     tuplify([1, 2])  # (1, 2)
276: (8)                     tuplify(len)     # (<built-in function len>,)
277: (4)                 """
278: (4)                 if isinstance(obj, str):
279: (8)                     return (obj,)
280: (4)                 try:
281: (8)                     return tuple(obj)
282: (4)                 except TypeError:
283: (8)                     return (obj,)
```

```

284: (0)         def uniq_chain(*args: Iterable) -> Generator:
285: (4)             """Returns a generator that yields all unique elements of the Iterables
286: (8)                 provided via args in the order provided.
287: (4)             Examples
288: (4)             -----
289: (4)             Normal usage::
290: (8)                 uniq_chain([1, 2], [2, 3], [1, 4, 4])
291: (4)             """
292: (4)             unique_items = set()
293: (4)             for x in it.chain(*args):
294: (8)                 if x in unique_items:
295: (12)                     continue
296: (8)                 unique_items.add(x)
297: (8)                 yield x
298: (0)         def hash_obj(obj: object) -> int:
299: (4)             """Determines a hash, even of potentially mutable objects."""
300: (4)             if isinstance(obj, dict):
301: (8)                 return hash(tuple(sorted((hash_obj(k), hash_obj(v)) for k, v in
302: obj.items())))
303: (4)             if isinstance(obj, set):
304: (8)                 return hash(tuple(sorted(hash_obj(e) for e in obj)))
305: (4)             if isinstance(obj, (tuple, list)):
306: (8)                 return hash(tuple(hash_obj(e) for e in obj))
306: (4)             return hash(obj)

-----

```

## File 145 - space\_ops.py:

```

1: (0)             """Utility functions for two- and three-dimensional vectors."""
2: (0)             from __future__ import annotations
3: (0)             import itertools as it
4: (0)             from typing import TYPE_CHECKING, Sequence
5: (0)             import numpy as np
6: (0)             from mapbox_earcut import triangulate_float32 as earcut
7: (0)             from scipy.spatial.transform import Rotation
8: (0)             from manim.constants import DOWN, OUT, PI, RIGHT, TAU, UP, RendererType
9: (0)             from manim.utils.iterables import adjacent_pairs
10: (0)             if TYPE_CHECKING:
11: (4)                 import numpy.typing as npt
12: (4)                 from manim.typing import (
13: (8)                     ManimFloat,
14: (8)                     Point3D_Array,
15: (8)                     Vector2D,
16: (8)                     Vector2D_Array,
17: (8)                     Vector3D,
18: (4)                 )
19: (0)             __all__ = [
20: (4)                 "quaternion_mult",
21: (4)                 "quaternion_from_angle_axis",
22: (4)                 "angle_axis_from_quaternion",
23: (4)                 "quaternion_conjugate",
24: (4)                 "rotate_vector",
25: (4)                 "thick_diagonal",
26: (4)                 "rotation_matrix",
27: (4)                 "rotation_about_z",
28: (4)                 "z_to_vector",
29: (4)                 "angle_of_vector",
30: (4)                 "angle_between_vectors",
31: (4)                 "normalize",
32: (4)                 "get_unit_normal",
33: (4)                 "compass_directions",
34: (4)                 "regular_vertices",
35: (4)                 "complex_to_R3",
36: (4)                 "R3_to_complex",
37: (4)                 "complex_func_to_R3_func",
38: (4)                 "center_of_mass",
39: (4)                 "midpoint",
40: (4)                 "find_intersection",

```

```

41: (4)                 "line_intersection",
42: (4)                 "get_winding_number",
43: (4)                 "shoelace",
44: (4)                 "shoelace_direction",
45: (4)                 "cross2d",
46: (4)                 "earclip_triangulation",
47: (4)                 "cartesian_to_spherical",
48: (4)                 "spherical_to_cartesian",
49: (4)                 "perpendicular_bisector",
50: (0)
51: (0)             def norm_squared(v: float) -> float:
52: (4)                 return np.dot(v, v)
53: (0)             def cross(v1: Vector3D, v2: Vector3D) -> Vector3D:
54: (4)                 return np.array(
55: (8)                     [
56: (12)                         v1[1] * v2[2] - v1[2] * v2[1],
57: (12)                         v1[2] * v2[0] - v1[0] * v2[2],
58: (12)                         v1[0] * v2[1] - v1[1] * v2[0],
59: (8)                     ]
60: (4)                 )
61: (0)             def quaternion_mult(
62: (4)                 *quats: Sequence[float],
63: (0)             ) -> np.ndarray | list[float | np.ndarray]:
64: (4)                 """Gets the Hamilton product of the quaternions provided.
65: (4)                 For more information, check `this Wikipedia page
66: (4)                 <https://en.wikipedia.org/wiki/Quaternion>__.
67: (4)                 Returns
68: (4)                 -----
69: (4)                 Union[np.ndarray, List[Union[float, np.ndarray]]]
70: (8)                 Returns a list of product of two quaternions.
71: (4)                 """
72: (4)                 if len(quats) == 0:
73: (8)                     return [1, 0, 0, 0]
74: (4)                 result = quats[0]
75: (4)                 for next_quat in quats[1:]:
76: (8)                     w1, x1, y1, z1 = result
77: (8)                     w2, x2, y2, z2 = next_quat
78: (8)                     result = [
79: (12)                         w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2,
80: (12)                         w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2,
81: (12)                         w1 * y2 + y1 * w2 + z1 * x2 - x1 * z2,
82: (12)                         w1 * z2 + z1 * w2 + x1 * y2 - y1 * x2,
83: (8)                     ]
84: (4)                 return result
85: (0)             def quaternion_from_angle_axis(
86: (4)                 angle: float,
87: (4)                 axis: np.ndarray,
88: (4)                 axis_normalized: bool = False,
89: (0)             ) -> list[float]:
90: (4)                 """Gets a quaternion from an angle and an axis.
91: (4)                 For more information, check `this Wikipedia page
92: (4)
<https://en.wikipedia.org/wiki/Conversion\_between\_quaternions\_and\_Euler\_angles>__.
93: (4)                 Parameters
94: (4)                 -----
95: (4)                 angle
96: (8)                     The angle for the quaternion.
97: (4)                 axis
98: (8)                     The axis for the quaternion
99: (4)                 axis_normalized
100: (8)                     Checks whether the axis is normalized, by default False
101: (4)                 Returns
102: (4)                 -----
103: (4)                 list[float]
104: (8)                     Gives back a quaternion from the angle and axis
105: (4)                 """
106: (4)                 if not axis_normalized:
107: (8)                     axis = normalize(axis)
108: (4)                 return [np.cos(angle / 2), *(np.sin(angle / 2) * axis)]

```

```

109: (0)         def angle_axis_from_quaternion(quaternion: Sequence[float]) ->
Sequence[float]:
110: (4)             """Gets angle and axis from a quaternion.
111: (4)             Parameters
112: (4)             -----
113: (4)             quaternion
114: (8)                 The quaternion from which we get the angle and axis.
115: (4)             Returns
116: (4)             -----
117: (4)             Sequence[float]
118: (8)                 Gives the angle and axis
119: (4)             """
120: (4)             axis = normalize(quaternion[1:], fall_back=np.array([1, 0, 0]))
121: (4)             angle = 2 * np.arccos(quaternion[0])
122: (4)             if angle > TAU / 2:
123: (8)                 angle = TAU - angle
124: (4)             return angle, axis
125: (0)         def quaternion_conjugate(quaternion: Sequence[float]) -> np.ndarray:
126: (4)             """Used for finding the conjugate of the quaternion
127: (4)             Parameters
128: (4)             -----
129: (4)             quaternion
130: (8)                 The quaternion for which you want to find the conjugate for.
131: (4)             Returns
132: (4)             -----
133: (4)             np.ndarray
134: (8)                 The conjugate of the quaternion.
135: (4)             """
136: (4)             result = np.array(quaternion)
137: (4)             result[1:] *= -1
138: (4)             return result
139: (0)         def rotate_vector(
140: (4)             vector: np.ndarray, angle: float, axis: np.ndarray = OUT
141: (0)         ) -> np.ndarray:
142: (4)             """Function for rotating a vector.
143: (4)             Parameters
144: (4)             -----
145: (4)             vector
146: (8)                 The vector to be rotated.
147: (4)             angle
148: (8)                 The angle to be rotated by.
149: (4)             axis
150: (8)                 The axis to be rotated, by default OUT
151: (4)             Returns
152: (4)             -----
153: (4)             np.ndarray
154: (8)                 The rotated vector with provided angle and axis.
155: (4)             Raises
156: (4)             -----
157: (4)             ValueError
158: (8)                 If vector is not of dimension 2 or 3.
159: (4)             """
160: (4)             if len(vector) > 3:
161: (8)                 raise ValueError("Vector must have the correct dimensions.")
162: (4)             if len(vector) == 2:
163: (8)                 vector = np.append(vector, 0)
164: (4)             return rotation_matrix(angle, axis) @ vector
165: (0)         def thick_diagonal(dim: int, thickness=2) -> np.ndarray:
166: (4)             row_indices = np.arange(dim).repeat(dim).reshape((dim, dim))
167: (4)             col_indices = np.transpose(row_indices)
168: (4)             return (np.abs(row_indices - col_indices) < thickness).astype("uint8")
169: (0)         def rotation_matrix_transpose_from_quaternion(quat: np.ndarray) ->
list[np.ndarray]:
170: (4)             """Converts the quaternion, quat, to an equivalent rotation matrix
representation.
171: (4)             For more information, check `this page
<https://in.mathworks.com/help/driving/ref/quaternion.rotmat.html>`_.
172: (4)             Parameters
173: (4)             -----

```

```

manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
175: (4)             quat
176: (8)             The quaternion which is to be converted.
177: (4)             Returns
178: (4)             -----
179: (4)             List[np.ndarray]
180: (8)             Gives back the Rotation matrix representation, returned as a 3-by-3
181: (8)             matrix or 3-by-3-by-N multidimensional array.
182: (4)             """
183: (4)             quat_inv = quaternion_conjugate(quat)
184: (4)             return [
185: (8)                 quaternion_mult(quat, [0, *basis], quat_inv)[1:]
186: (8)                 for basis in [
187: (12)                     [1, 0, 0],
188: (12)                     [0, 1, 0],
189: (12)                     [0, 0, 1],
190: (8)                 ]
191: (4)             ]
192: (0)             def rotation_matrix_from_quaternion(quat: np.ndarray) -> np.ndarray:
193: (4)             return np.transpose(rotation_matrix_transpose_from_quaternion(quat))
194: (0)             def rotation_matrix_transpose(angle: float, axis: np.ndarray) -> np.ndarray:
195: (4)             if all(np.array(axis)[:2] == np.zeros(2)):
196: (8)                 return rotation_about_z(angle * np.sign(axis[2])).T
197: (4)             return rotation_matrix(angle, axis).T
198: (0)             def rotation_matrix(
199: (4)                 angle: float,
200: (4)                 axis: np.ndarray,
201: (4)                 homogeneous: bool = False,
202: (0)             ) -> np.ndarray:
203: (4)             """
204: (4)             Rotation in R^3 about a specified axis of rotation.
205: (4)             """
206: (4)             inhomogeneous_rotation_matrix = Rotation.from_rotvec(
207: (8)                 angle * normalize(np.array(axis)))
208: (4)             ).as_matrix()
209: (4)             if not homogeneous:
210: (8)                 return inhomogeneous_rotation_matrix
211: (4)             else:
212: (8)                 rotation_matrix = np.eye(4)
213: (8)                 rotation_matrix[:3, :3] = inhomogeneous_rotation_matrix
214: (8)                 return rotation_matrix
215: (0)             def rotation_about_z(angle: float) -> np.ndarray:
216: (4)             """Returns a rotation matrix for a given angle.
217: (4)             Parameters
218: (4)             -----
219: (4)             angle
220: (8)             Angle for the rotation matrix.
221: (4)             Returns
222: (4)             -----
223: (4)             np.ndarray
224: (8)             Gives back the rotated matrix.
225: (4)             """
226: (4)             c, s = np.cos(angle), np.sin(angle)
227: (4)             return np.array(
228: (8)                 [
229: (12)                     [c, -s, 0],
230: (12)                     [s, c, 0],
231: (12)                     [0, 0, 1],
232: (8)                 ]
233: (4)             )
234: (0)             def z_to_vector(vector: np.ndarray) -> np.ndarray:
235: (4)             """
236: (4)             Returns some matrix in SO(3) which takes the z-axis to the
237: (4)             (normalized) vector provided as an argument
238: (4)             """
239: (4)             axis_z = normalize(vector)
240: (4)             axis_y = normalize(cross(axis_z, RIGHT))
241: (4)             axis_x = cross(axis_y, axis_z)
242: (4)             if np.linalg.norm(axis_y) == 0:
243: (8)                 axis_x = normalize(cross(UP, axis_z))

```

```

244: (8)             axis_y = -cross(axis_x, axis_z)
245: (4)             return np.array([axis_x, axis_y, axis_z]).T
246: (0)             def angle_of_vector(vector: Sequence[float] | np.ndarray) -> float:
247: (4)                 """Returns polar coordinate theta when vector is projected on xy plane.
248: (4)                 Parameters
249: (4)                 -----
250: (4)                 vector
251: (8)                     The vector to find the angle for.
252: (4)                 Returns
253: (4)                 -----
254: (4)                 float
255: (8)                     The angle of the vector projected.
256: (4)                 """
257: (4)                 if isinstance(vector, np.ndarray) and len(vector.shape) > 1:
258: (8)                     if vector.shape[0] < 2:
259: (12)                         raise ValueError("Vector must have the correct dimensions. (2, n)")
260: (8)                     c_vec = np.empty(vector.shape[1], dtype=np.complex128)
261: (8)                     c_vec.real = vector[0]
262: (8)                     c_vec.imag = vector[1]
263: (8)                     return np.angle(c_vec)
264: (4)                 return np.angle(complex(*vector[:2]))
265: (0)             def angle_between_vectors(v1: np.ndarray, v2: np.ndarray) -> float:
266: (4)                 """Returns the angle between two vectors.
267: (4)                 This angle will always be between 0 and pi
268: (4)                 Parameters
269: (4)                 -----
270: (4)                 v1
271: (8)                     The first vector.
272: (4)                 v2
273: (8)                     The second vector.
274: (4)                 Returns
275: (4)                 -----
276: (4)                 float
277: (8)                     The angle between the vectors.
278: (4)                 """
279: (4)                 return 2 * np.arctan2(
280: (8)                     np.linalg.norm(normalize(v1) - normalize(v2)),
281: (8)                     np.linalg.norm(normalize(v1) + normalize(v2)),
282: (4)                 )
283: (0)             def normalize(vect: np.ndarray | tuple[float], fall_back=None) -> np.ndarray:
284: (4)                 norm = np.linalg.norm(vect)
285: (4)                 if norm > 0:
286: (8)                     return np.array(vect) / norm
287: (4)                 else:
288: (8)                     return fall_back or np.zeros(len(vect))
289: (0)             def normalize_along_axis(array: np.ndarray, axis: np.ndarray) -> np.ndarray:
290: (4)                 """Normalizes an array with the provided axis.
291: (4)                 Parameters
292: (4)                 -----
293: (4)                 array
294: (8)                     The array which has to be normalized.
295: (4)                 axis
296: (8)                     The axis to be normalized to.
297: (4)                 Returns
298: (4)                 -----
299: (4)                 np.ndarray
300: (8)                     Array which has been normalized according to the axis.
301: (4)                 """
302: (4)                 norms = np.sqrt((array * array).sum(axis))
303: (4)                 norms[norms == 0] = 1
304: (4)                 buffed_norms = np.repeat(norms, array.shape[axis]).reshape(array.shape)
305: (4)                 array /= buffed_norms
306: (4)                 return array
307: (0)             def get_unit_normal(v1: Vector3D, v2: Vector3D, tol: float = 1e-6) ->
Vector3D:
308: (4)                 """Gets the unit normal of the vectors.
309: (4)                 Parameters
310: (4)                 -----

```

```

311: (4)           v1
312: (8)             The first vector.
313: (4)           v2
314: (8)             The second vector
315: (4)           tol
316: (8)             [description], by default 1e-6
317: (4)           Returns
318: (4)           -----
319: (4)           np.ndarray
320: (8)             The normal of the two vectors.
321: (4)           """
322: (4)           div1, div2 = max(np.abs(v1)), max(np.abs(v2))
323: (4)           if div1 == 0.0:
324: (8)             if div2 == 0.0:
325: (12)               return DOWN
326: (8)             u = v2 / div2
327: (4)           elif div2 == 0.0:
328: (8)             u = v1 / div1
329: (4)           else:
330: (8)             u1, u2 = v1 / div1, v2 / div2
331: (8)             cp = cross(u1, u2)
332: (8)             cp_norm = np.sqrt(norm_squared(cp))
333: (8)             if cp_norm > tol:
334: (12)               return cp / cp_norm
335: (8)             u = u1
336: (4)           if abs(u[0]) < tol and abs(u[1]) < tol:
337: (8)             return DOWN
338: (4)           cp = np.array([-u[0] * u[2], -u[1] * u[2], u[0] * u[0] + u[1] * u[1]])
339: (4)           cp_norm = np.sqrt(norm_squared(cp))
340: (4)           return cp / cp_norm
341: (0)           def compass_directions(n: int = 4, start_vect: np.ndarray = RIGHT) ->
np.ndarray:
342: (4)             """Finds the cardinal directions using tau.
343: (4)             Parameters
344: (4)             -----
345: (4)             n
346: (8)               The amount to be rotated, by default 4
347: (4)             start_vect
348: (8)               The direction for the angle to start with, by default RIGHT
349: (4)             Returns
350: (4)             -----
351: (4)             np.ndarray
352: (8)               The angle which has been rotated.
353: (4)             """
354: (4)             angle = TAU / n
355: (4)             return np.array([rotate_vector(start_vect, k * angle) for k in range(n)])
356: (0)           def regular_vertices(
357: (4)             n: int, *, radius: float = 1, start_angle: float | None = None
358: (0)           ) -> tuple[np.ndarray, float]:
359: (4)             """Generates regularly spaced vertices around a circle centered at the
origin.
360: (4)             Parameters
361: (4)             -----
362: (4)             n
363: (8)               The number of vertices
364: (4)             radius
365: (8)               The radius of the circle that the vertices are placed on.
366: (4)             start_angle
367: (8)               The angle the vertices start at.
368: (8)               If unspecified, for even ``n`` values, ``0`` will be used.
369: (8)               For odd ``n`` values, 90 degrees is used.
370: (4)             Returns
371: (4)             -----
372: (4)             vertices : :class:`numpy.ndarray`
373: (8)               The regularly spaced vertices.
374: (4)             start_angle : :class:`float`
375: (8)               The angle the vertices start at.
376: (4)             """
377: (4)             if start_angle is None:

```

```

378: (8)             if n % 2 == 0:
379: (12)             start_angle = 0
380: (8)         else:
381: (12)             start_angle = TAU / 4
382: (4)         start_vector = rotate_vector(RIGHT * radius, start_angle)
383: (4)         vertices = compass_directions(n, start_vector)
384: (4)         return vertices, start_angle
385: (0)     def complex_to_R3(complex_num: complex) -> np.ndarray:
386: (4)         return np.array((complex_num.real, complex_num.imag, 0))
387: (0)     def R3_to_complex(point: Sequence[float]) -> np.ndarray:
388: (4)         return complex(*point[:2])
389: (0)     def complex_func_to_R3_func(complex_func):
390: (4)         return lambda p: complex_to_R3(complex_func(R3_to_complex(p)))
391: (0)     def center_of_mass(points: Sequence[float]) -> np.ndarray:
392: (4)         """Gets the center of mass of the points in space.
393: (4)         Parameters
394: (4)         -----
395: (4)         points
396: (8)             The points to find the center of mass from.
397: (4)         Returns
398: (4)         -----
399: (4)         np.ndarray
400: (8)             The center of mass of the points.
401: (4)         """
402: (4)         return np.average(points, 0, np.ones(len(points)))
403: (0)     def midpoint(
404: (4)         point1: Sequence[float],
405: (4)         point2: Sequence[float],
406: (0)     ) -> float | np.ndarray:
407: (4)         """Gets the midpoint of two points.
408: (4)         Parameters
409: (4)         -----
410: (4)         point1
411: (8)             The first point.
412: (4)         point2
413: (8)             The second point.
414: (4)         Returns
415: (4)         -----
416: (4)         Union[float, np.ndarray]
417: (8)             The midpoint of the points
418: (4)         """
419: (4)         return center_of_mass([point1, point2])
420: (0)     def line_intersection(
421: (4)         line1: Sequence[np.ndarray], line2: Sequence[np.ndarray]
422: (0)     ) -> np.ndarray:
423: (4)         """Returns the intersection point of two lines, each defined by
424: (4)         a pair of distinct points lying on the line.
425: (4)         Parameters
426: (4)         -----
427: (4)         line1
428: (8)             A list of two points that determine the first line.
429: (4)         line2
430: (8)             A list of two points that determine the second line.
431: (4)         Returns
432: (4)         -----
433: (4)         np.ndarray
434: (8)             The intersection points of the two lines which are intersecting.
435: (4)         Raises
436: (4)         -----
437: (4)         ValueError
438: (8)             Error is produced if the two lines don't intersect with each other
439: (8)             or if the coordinates don't lie on the xy-plane.
440: (4)         """
441: (4)         if any(np.array([line1, line2])[:, :, 2].reshape(-1)):
442: (8)             raise ValueError("Coords must be in the xy-plane.")
443: (4)         padded =
444: (8)             np.pad(np.array(i)[:, :2], ((0, 0), (0, 1)), constant_values=1)
445: (8)             for i in (line1, line2)
446: (4)         )

```

```

447: (4)             line1, line2 = (cross(*i) for i in padded)
448: (4)             x, y, z = cross(line1, line2)
449: (4)             if z == 0:
450: (8)                 raise ValueError(
451: (12)                     "The lines are parallel, there is no unique intersection point."
452: (8)                 )
453: (4)             return np.array([x / z, y / z, 0])
454: (0)         def find_intersection(
455: (4)             p0s: Sequence[np.ndarray] | Point3D_Array,
456: (4)             v0s: Sequence[np.ndarray] | Point3D_Array,
457: (4)             p1s: Sequence[np.ndarray] | Point3D_Array,
458: (4)             v1s: Sequence[np.ndarray] | Point3D_Array,
459: (4)             threshold: float = 1e-5,
460: (0)         ) -> Sequence[np.ndarray]:
461: (4)             """
462: (4)             Return the intersection of a line passing through p0 in direction v0
463: (4)             with one passing through p1 in direction v1 (or array of intersections
464: (4)             from arrays of such points/directions).
465: (4)             For 3d values, it returns the point on the ray p0 + v0 * t closest to the
466: (4)             ray p1 + v1 * t
467: (4)             """
468: (4)             result = []
469: (4)             for p0, v0, p1, v1 in zip(*[p0s, v0s, p1s, v1s]):
470: (8)                 normal = cross(v1, cross(v0, v1))
471: (8)                 denom = max(np.dot(v0, normal), threshold)
472: (8)                 result += [p0 + np.dot(p1 - p0, normal) / denom * v0]
473: (4)             return result
474: (0)         def get_winding_number(points: Sequence[np.ndarray]) -> float:
475: (4)             """Determine the number of times a polygon winds around the origin.
476: (4)             The orientation is measured mathematically positively, i.e.,
477: (4)             counterclockwise.
478: (4)             Parameters
479: (4)             -----
480: (4)             points
481: (8)                 The vertices of the polygon being queried.
482: (4)             Examples
483: (4)             -----
484: (4)             >>> from manim import Square, get_winding_number
485: (4)             >>> polygon = Square()
486: (4)             >>> get_winding_number(polygon.get_vertices())
487: (4)             1.0
488: (4)             >>> polygon.shift(2*UP)
489: (4)             Square
490: (4)             >>> get_winding_number(polygon.get_vertices())
491: (4)             0.0
492: (4)             """
493: (4)             total_angle = 0
494: (4)             for p1, p2 in adjacent_pairs(points):
495: (8)                 d_angle = angle_of_vector(p2) - angle_of_vector(p1)
496: (8)                 d_angle = ((d_angle + PI) % TAU) - PI
497: (8)                 total_angle += d_angle
498: (4)             return total_angle / TAU
499: (0)         def shoelace(x_y: np.ndarray) -> float:
500: (4)             """2D implementation of the shoelace formula.
501: (4)             Returns
502: (4)             -----
503: (4)             :class:`float`
504: (8)                 Returns signed area.
505: (4)             """
506: (4)             x = x_y[:, 0]
507: (4)             y = x_y[:, 1]
508: (4)             return np.trapz(y, x)
509: (0)         def shoelace_direction(x_y: np.ndarray) -> str:
510: (4)             """
511: (4)             Uses the area determined by the shoelace method to determine whether
512: (4)             the input set of points is directed clockwise or counterclockwise.
513: (4)             Returns
514: (4)             -----
515: (4)             :class:`str`
```



```

583: (8)             i = min(i_range, key=lambda i: norm_squared(verts[i] - tmp_j_vert))
584: (8)             j = min(j_range, key=lambda j: norm_squared(verts[i] - verts[j]))
585: (8)             i = min(i_range, key=lambda i: norm_squared(verts[i] - verts[j]))
586: (8)             loop_connections[i] = j
587: (8)             loop_connections[j] = i
588: (8)             new_ring = next(
589: (12)                 (ring for ring in detached_rings if ring[0] <= j < ring[-1]), None
590: (8)
591: (8)             )
592: (12)             if new_ring is not None:
593: (12)                 detached_rings.remove(new_ring)
594: (8)                 attached_rings.append(new_ring)
595: (12)             else:
596: (4)                 raise Exception("Could not find a ring to attach")
597: (4)             after = []
598: (4)             end0 = 0
599: (8)             for end1 in ring_ends:
600: (8)                 after.extend(range(end0 + 1, end1))
601: (8)                 after.append(end0)
602: (4)                 end0 = end1
603: (4)             indices = []
604: (4)             i = 0
605: (8)             for _ in range(len(verts) + len(ring_ends) - 1):
606: (12)                 if i in loop_connections:
607: (12)                     j = loop_connections[i]
608: (12)                     indices.extend([i, j])
609: (8)                     i = after[j]
610: (12)                 else:
611: (12)                     indices.append(i)
612: (8)                     i = after[i]
613: (12)                     if i == 0:
614: (8)                         break
615: (4)             meta_indices = earcut(verts[indices, :2], [len(indices)])
616: (0)             return [indices[mi] for mi in meta_indices]
def cartesian_to_spherical(vec: Sequence[float]) -> np.ndarray:
    """Returns an array of numbers corresponding to each
    polar coordinate value (distance, phi, theta).
    Parameters
    -----
    vec
        A numpy array ``[x, y, z]``.
    """
    norm = np.linalg.norm(vec)
    if norm == 0:
        return 0, 0, 0
    r = norm
    phi = np.arccos(vec[2] / r)
    theta = np.arctan2(vec[1], vec[0])
    return np.array([r, theta, phi])
def spherical_to_cartesian(spherical: Sequence[float]) -> np.ndarray:
    """Returns a numpy array ``[x, y, z]`` based on the spherical
    coordinates given.
    Parameters
    -----
    spherical
        A list of three floats that correspond to the following:
        r - The distance between the point and the origin.
        theta - The azimuthal angle of the point to the positive x-axis.
        phi - The vertical angle of the point to the positive z-axis.
    """
    r, theta, phi = spherical
    return np.array([
        [
            r * np.cos(theta) * np.sin(phi),
            r * np.sin(theta) * np.sin(phi),
            r * np.cos(phi),
        ],
    ])
def perpendicular_bisector(
    line: Sequence[np.ndarray],

```

```

652: (4)             norm_vector=OUT,
653: (0)         ) -> Sequence[np.ndarray]:
654: (4)             """Returns a list of two points that correspond
655: (4)             to the ends of the perpendicular bisector of the
656: (4)             two points given.
657: (4)             Parameters
658: (4)             -----
659: (4)             line
660: (8)                 a list of two numpy array points (corresponding
661: (8)                 to the ends of a line).
662: (4)             norm_vector
663: (8)                 the vector perpendicular to both the line given
664: (8)                 and the perpendicular bisector.
665: (4)             Returns
666: (4)             -----
667: (4)             list
668: (8)                 A list of two numpy array points that correspond
669: (8)                 to the ends of the perpendicular bisector
670: (4)
671: (4)             """
672: (4)             p1 = line[0]
673: (4)             p2 = line[1]
674: (4)             direction = cross(p1 - p2, norm_vector)
675: (4)             m = midpoint(p1, p2)
676: (4)             return [m + direction, m - direction]

```

-----

## File 146 - exceptions.py:

```

1: (0)         from __future__ import annotations
2: (0)         __all__ = [
3: (4)             "EndSceneEarlyException",
4: (4)             "RerunSceneException",
5: (4)             "MultiAnimationOverrideException",
6: (0)
7: (0)         ]
8: (4)         class EndSceneEarlyException(Exception):
9: (0)             pass
10: (4)            class RerunSceneException(Exception):
11: (0)                pass
12: (4)                class MultiAnimationOverrideException(Exception):
13: (0)                    pass

```

-----

## File 147 - family\_ops.py:

```

1: (0)         from __future__ import annotations
2: (0)         import itertools as it
3: (0)         __all__ = [
4: (4)             "extract_mobject_family_members",
5: (4)             "restructure_list_to_exclude_certain_family_members",
6: (0)
7: (0)         ]
8: (0)         def extract_mobject_family_members(mobject_list,
only_those_with_points=False):
9: (4)             result = list(it.chain(*(mob.get_family() for mob in mobject_list)))
10: (4)             if only_those_with_points:
11: (8)                 result = [mob for mob in result if mob.has_points()]
12: (4)                 return result
13: (0)         def restructure_list_to_exclude_certain_family_members(mobject_list,
to_remove):
14: (4)             """
15: (4)                 Removes anything in to_remove from mobject_list, but in the event that one
16: (4)                 of
17: (4)                     the items to be removed is a member of the family of an item in
18: (4)                     the other family members are added back into the list.
19: (4)                     This is useful in cases where a scene contains a group, e.g. Group(m1, m2,
m3),
20: (4)                     but one of its submobjects is removed, e.g. scene.remove(m1), it's useful

```

```
12/20/24, 4:24 AM manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
19: (4)                 for the list of mobject_list to be edited to contain other submobjects,
but not m1.
20: (4)                 """
21: (4)                 new_list = []
22: (4)                 to_remove = extract_mobject_family_members(to_remove)
23: (4)                 def add_safe_mobjects_from_list(list_to_examine, set_to_remove):
24: (8)                     for mob in list_to_examine:
25: (12)                         if mob in set_to_remove:
26: (16)                             continue
27: (12)                         intersect = set_to_remove.intersection(mob.get_family())
28: (12)                         if intersect:
29: (16)                             add_safe_mobjects_from_list(mob.submobjects, intersect)
30: (12)                         else:
31: (16)                             new_list.append(mob)
32: (4)                 add_safe_mobjects_from_list(mobject_list, set(to_remove))
33: (4)                 return new_list
```

## File 148 - module ops.py:

```
1: (0)         from __future__ import annotations
2: (0)         import importlib.util
3: (0)         import inspect
4: (0)         import os
5: (0)         import re
6: (0)         import sys
7: (0)         import types
8: (0)         import warnings
9: (0)         from pathlib import Path
10: (0)        from .. import config, console, constants, logger
11: (0)        from ..scene.scene_file_writer import SceneFileWriter
12: (0)        __all__ = ["scene_classes_from_file"]
13: (0)        def get_module(file_name: Path):
14: (4)            if str(file_name) == "-":
15: (8)                module = types.ModuleType("input_scenes")
16: (8)                logger.info(
17: (12)                    "Enter the animation's code & end with an EOF (CTRL+D on
Linux/Unix, CTRL+Z on Windows):",
18: (8)                )
19: (8)                code = sys.stdin.read()
20: (8)                if not code.startswith("from manim import"):
21: (12)                    logger.warning(
22: (16)                        "Didn't find an import statement for Manim. Importing
automatically...",)
23: (12)                )
24: (12)                code = "from manim import *\n" + code
25: (8)                logger.info("Rendering animation from typed code...")
26: (8)                try:
27: (12)                    exec(code, module.__dict__)
28: (12)                    return module
29: (8)                except Exception as e:
30: (12)                    logger.error(f"Failed to render scene: {str(e)}")
31: (12)                    sys.exit(2)
32: (4)                else:
33: (8)                    if file_name.exists():
34: (12)                        ext = file_name.suffix
35: (12)                        if ext != ".py":
36: (16)                            raise ValueError(f"{file_name} is not a valid Manim python
script.")
37: (12)
38: (12)
39: (16)
40: (16)
41: (16)
42: (12)
43: (12)
44: (12)        file_name)
spec = importlib.util.spec_from_file_location(module_name,
module = importlib.util.module_from_spec(spec)
```

```

45: (12)                     sys.modules[module_name] = module
46: (12)                     sys.path.insert(0, str(file_name.parent.absolute()))
47: (12)                     spec.loader.exec_module(module)
48: (12)                     return module
49: (8)             else:
50: (12)                 raise FileNotFoundError(f"{file_name} not found")
51: (0)         def get_scene_classes_from_module(module):
52: (4)             from ..scene.scene import Scene
53: (4)             def is_child_scene(obj, module):
54: (8)                 return (
55: (12)                     inspect.isclass(obj)
56: (12)                     and issubclass(obj, Scene)
57: (12)                     and obj != Scene
58: (12)                     and obj.__module__.startswith(module.__name__))
59: (8)             )
60: (4)         return [
61: (8)             member[1]
62: (8)             for member in inspect.getmembers(module, lambda x: is_child_scene(x,
63: (4)             module))
64: (0)     ]
65: (4)     def get_scenes_to_render(scene_classes):
66: (8)         if not scene_classes:
67: (8)             logger.error(constants.NO_SCENE_MESSAGE)
68: (8)             return []
69: (8)         if config["write_all"]:
70: (8)             return scene_classes
71: (4)         result = []
72: (8)         for scene_name in config["scene_names"]:
73: (8)             found = False
74: (12)             for scene_class in scene_classes:
75: (16)                 if scene_class.__name__ == scene_name:
76: (16)                     result.append(scene_class)
77: (16)                     found = True
78: (8)                     break
79: (12)             if not found and (scene_name != ""):
80: (4)                 logger.error(constants.SCENE_NOT_FOUND_MESSAGE.format(scene_name))
81: (8)         if result:
82: (4)             return result
83: (8)         if len(scene_classes) == 1:
84: (8)             config["scene_names"] = [scene_classes[0].__name__]
85: (8)             return prompt_user_for_choice(scene_classes)
86: (0)     def prompt_user_for_choice(scene_classes):
87: (4)         num_to_class = {}
88: (4)         SceneFileWriter.force_output_as_scene_name = True
89: (4)         for count, scene_class in enumerate(scene_classes, 1):
90: (8)             name = scene_class.__name__
91: (8)             console.print(f"{count}: {name}", style="logging.level.info")
92: (8)             num_to_class[count] = scene_class
93: (4)         try:
94: (8)             user_input = console.input(
95: (12)                 f"[log.message] {constants.CHOOSE_NUMBER_MESSAGE} [/log.message]",
96: (8)             )
97: (8)             scene_classes = [
98: (12)                 num_to_class[int(num_str)]
99: (12)                 for num_str in re.split(r"\s*,\s*", user_input.strip())
100: (8)             ]
101: (8)             config["scene_names"] = [scene_class.__name__ for scene_class in
102: (8)             scene_classes]
103: (4)             return scene_classes
104: (8)         except KeyError:
105: (8)             logger.error(constants.INVALID_NUMBER_MESSAGE)
106: (8)             sys.exit(2)
107: (8)         except EOFError:
108: (8)             sys.exit(1)
109: (8)         except ValueError:
110: (8)             logger.error("No scenes were selected. Exiting.")
111: (0)             sys.exit(1)
111: (0)     def scene_classes_from_file(

```

```

112: (4)             file_path: Path, require_single_scene=False, full_list=False
113: (0)
114: (4)
115: (4)
116: (4)
117: (8)
118: (4)
119: (4)
120: (8)
121: (8)
122: (4)
      module = get_module(file_path)
      all_scene_classes = get_scene_classes_from_module(module)
      if full_list:
          return all_scene_classes
      scene_classes_to_render = get_scenes_to_render(all_scene_classes)
      if require_single_scene:
          assert len(scene_classes_to_render) == 1
          return scene_classes_to_render[0]
      return scene_classes_to_render
-----
```

## File 149 - ipython\_magic.py:

```

1: (0)             """Utilities for using Manim with IPython (in particular: Jupyter
notebooks)"""
2: (0)             from __future__ import annotations
3: (0)             import mimetypes
4: (0)             import os
5: (0)             import shutil
6: (0)             from datetime import datetime
7: (0)             from pathlib import Path
8: (0)             from typing import Any
9: (0)             from manim import Group, config, logger, tempconfig
10: (0)            from manim.__main__ import main
11: (0)            from manim.renderer.shader import shader_program_cache
12: (0)            from ..constants import RendererType
13: (0)            __all__ = ["ManimMagic"]
14: (0)
15: (4)            try:
16: (4)                from IPython import get_ipython
17: (4)                from IPython.core.interactiveshell import InteractiveShell
18: (4)                from IPython.core.magic import (
19: (8)                    Magics,
20: (8)                    line_cell_magic,
21: (8)                    magics_class,
22: (8)                    needs_local_scope,
23: (4)                )
24: (4)                from IPython.display import Image, Video, display
25: (0)            except ImportError:
26: (0)                pass
27: (0)            else:
28: (4)                @magics_class
29: (4)                class ManimMagic(Magics):
30: (12)                    def __init__(self, shell: InteractiveShell) -> None:
31: (12)                        super().__init__(shell)
32: (8)                        self.rendered_files = {}
33: (8)                    @needs_local_scope
34: (8)                    @line_cell_magic
35: (8)                    def manim(
36: (12)                        self,
37: (12)                        line: str,
38: (12)                        cell: str = None,
39: (8)                        local_ns: dict[str, Any] = None,
40: (12)                    ) -> None:
41: (12)                        r"""Render Manim scenes contained in IPython cells.
42: (12)                        Works as a line or cell magic.
43: (16)                        .. hint::
44: (16)                            This line and cell magic works best when used in a JupyterLab
45: (16)                            environment: while all of the functionality is available for
46: (16)                            classic Jupyter notebooks as well, it is possible that videos
47: (16)                            sometimes don't update on repeated execution of the same cell
48: (16)                            if the scene name stays the same.
49: (12)                            This problem does not occur when using JupyterLab.
50: (12)                            Please refer to `<https://jupyter.org/>`_ for more information
about JupyterLab
51: (12)                        and Jupyter notebooks.
51: (12)                        Usage in line mode::
```

```

52: (16)                         %manim [CLI options] MyAwesomeScene
53: (12)                         Usage in cell mode::
54: (16)                         %%manim [CLI options] MyAwesomeScene
55: (16)                         class MyAweseomeScene(Scene):
56: (20)                           def construct(self):
57: (24)                             ...
58: (12)                         Run ``%manim --help`` and ``%manim render --help`` for possible
command line interface options.
59: (12)                         .. note::
60: (16)                           The maximal width of the rendered videos that are displayed in
the notebook can be
61: (16)                           configured via the ``media_width`` configuration option. The
62: (16)                           default is set to ``25vw``, which is 25% of your current viewport width. To allow the
63: (16)                           as possible, set ``config.media_width = "100%"``.
64: (16)                           The ``media_embed`` option will embed the image/video output
65: (16)                           generally undesirable as it makes the notebooks very large,
66: (16)                           platforms (notably Google's CoLab, where it is automatically
67: (16)                           by ``config.embed = False``) and needed in cases when the
notebook (or converted HTML
68: (16)                           file) will be moved relative to the video locations. Use-cases
69: (16)                           documentation with Sphinx and JupyterBook. See also the
:mod:`manim directive for Sphinx
70: (16)                         <manim.utils.docbuild.manim_directive> .
71: (12)                         Examples
72: (12)                         -----
73: (12)                         First make sure to put ``import manim``, or even ``from manim
import *``
74: (12)                         in a cell and evaluate it. Then, a typical Jupyter notebook cell
for Manim
75: (12)                         could look as follows::
76: (16)                         %%manim -v WARNING --disable_caching -qm BannerExample
77: (16)                         config.media_width = "75%"
78: (16)                         config.media_embed = True
79: (16)                         class BannerExample(Scene):
80: (20)                           def construct(self):
81: (24)                             self.camera.background_color = "#ece6e2"
82: (24)                             banner_large =
ManimBanner(dark_theme=False).scale(0.7)
83: (24)                             self.play(banner_large.create())
84: (24)                             self.play(banner_large.expand())
85: (12)                         Evaluating this cell will render and display the ``BannerExample``
scene defined in the body of the cell.
86: (12)                         .. note::
87: (16)                           In case you want to hide the red box containing the output
progress bar, the ``progress_bar`` config
88: (16)                           option should be set to ``None``. This can also be done by
passing ``--progress_bar None`` as a
89: (16)                           CLI flag.
90: (12)                         """
91: (12)                         if cell:
92: (16)                           exec(cell, local_ns)
93: (12)                           args = line.split()
94: (12)                           if not len(args) or "-h" in args or "--help" in args or "--
version" in args:
95: (16)                             main(args, standalone_mode=False, prog_name="manim")
96: (16)                             return
97: (12)                           modified_args = self.add_additional_args(args)
98: (12)                           args = main(modified_args, standalone_mode=False,
prog_name="manim")
99: (12)                           with tempconfig(local_ns.get("config", {})):
100: (16)                             config.digest_args(args)
101: (16)                             renderer = None
102: (16)                             if config.renderer == RendererType.OPENGL:

```

```

103: (20)                     from manim.renderer.opengl_renderer import OpenGLRenderEngine
104: (20)                     renderer = OpenGLRenderEngine()
105: (16)                     try:
106: (20)                         SceneClass = local_ns[config["scene_names"][0]]
107: (20)                         scene = SceneClass(renderer=renderer)
108: (20)                         scene.render()
109: (16)                     finally:
110: (20)                         shader_program_cache.clear()
111: (20)                         if renderer is not None and renderer.window is not None:
112: (24)                             renderer.window.close()
113: (16)                     if config["output_file"] is None:
114: (20)                         logger.info("No output file produced")
115: (20)                         return
116: (16)                     local_path =
Path(config["output_file"]).relative_to(Path.cwd())
117: (16)                     tmpfile = (
118: (20)                         Path(config["media_dir"])
119: (20)                         / "jupyter"
120: (20)                         / f"({_generate_file_name()}{local_path.suffix})"
)
121: (16)                     if local_path in self.rendered_files:
122: (20)                         self.rendered_files[local_path].unlink()
123: (16)                     self.rendered_files[local_path] = tmpfile
124: (16)                     tmpfile.parent.mkdir(parents=True, exist_ok=True)
125: (16)                     shutil.copy(local_path, tmpfile)
126: (16)                     file_type = mimetypes.guess_type(config["output_file"])[0]
127: (16)                     embed = config["media_embed"]
128: (16)                     if embed is None:
129: (20)                         embed = "google.colab" in str(get_ipython())
130: (16)                     if file_type.startswith("image"):
131: (20)                         result = Image(filename=config["output_file"])
132: (16)                     else:
133: (20)                         result = Video(
134: (24)                             tmpfile,
135: (24)                             html_attributes=f'controls autoplay loop style="max-
width: {config["media_width"]};"',
136: (24)                             embed=embed,
)
137: (20)                     )
138: (16)                     display(result)
139: (16)             def add_additional_args(self, args: list[str]) -> list[str]:
140: (8)                 additional_args = ["--jupyter"]
141: (12)                 if "-t" in args and "--format" not in args:
142: (12)                     additional_args += ["--format", "webm"]
143: (16)                 return additional_args + args[:-1] + [""]
144: (12)             def _generate_file_name() -> str:
145: (0)                 return config["scene_names"][0] + "@" + datetime.now().strftime("%Y-%m-
%d@%H-%M-%S")
-----
```

## File 150 - rate\_functions.py:

```

1: (0)             """A selection of rate functions, i.e., *speed curves* for animations.
2: (0)             Please find a standard list at https://easings.net/. Here is a picture
3: (0)             for the non-standard ones
4: (0)             .. manim:: RateFuncExample
5: (4)                 :save_last_frame:
6: (4)                 class RateFuncExample(Scene):
7: (8)                     def construct(self):
8: (12)                         x = VGroup()
9: (12)                         for k, v in rate_functions.__dict__.items():
10: (16)                             if "function" in str(v):
11: (20)                                 if (
12: (24)                                     not k.startswith("__")
13: (24)                                     and not k.startswith("sqrt")
14: (24)                                     and not k.startswith("bezier")
15: (20)                                 ):
16: (24)                                     try:
17: (28)   rate_func = v
```

```

18: (28)                               plot = (
19: (32)                                 ParametricFunction(
20: (36)                                   lambda x: [x, rate_func(x), 0],
21: (36)                                   t_range=[0, 1, .01],
22: (36)                                   use_smoothing=False,
23: (36)                                   color=YELLOW,
24: (32)                                 )
25: (32)                                 .stretch_to_fit_width(1.5)
26: (32)                                 .stretch_to_fit_height(1)
27: (28)                               )
28: (28)                               plot_bg =
SurroundingRectangle(plot).set_color(WHITE)
29: (28)                               plot_title = (
30: (32)                                 Text(rate_func.__name__, weight=BOLD)
31: (32)                                 .scale(0.5)
32: (32)                                 .next_to(plot_bg, UP, buff=0.1)
33: (28)                               )
34: (28)                               x.add(VGroup(plot_bg, plot, plot_title))
35: (24)                               except: # because functions `not Quite There`,
`function squish_rate_func` are not working.
36: (28)                               pass
37: (12)                               x.arrange_in_grid(cols=8)
38: (12)                               x.height = config.frame_height
39: (12)                               x.width = config.frame_width
40: (12)                               x.move_to(ORIGIN).scale(0.95)
41: (12)                               self.add(x)
42: (0)                               There are primarily 3 kinds of standard easing functions:
43: (0)                               .. note:: The standard functions are not exported, so to use them you do
something like this:
44: (4)                               rate_func=rate_functions.ease_in_sine
45: (4)                               On the other hand, the non-standard functions, which are used more
commonly, are exported and can be used directly.
46: (0)                               .. manim:: RateFunctions1Example
47: (4)                               class RateFunctions1Example(Scene):
48: (8)                               def construct(self):
49: (12)                               line1 = Line(3*LEFT, 3*RIGHT).shift(UP).set_color(RED)
50: (12)                               line2 = Line(3*LEFT, 3*RIGHT).set_color(GREEN)
51: (12)                               line3 = Line(3*LEFT, 3*RIGHT).shift(DOWN).set_color(BLUE)
52: (12)                               dot1 = Dot().move_to(line1.get_left())
53: (12)                               dot2 = Dot().move_to(line2.get_left())
54: (12)                               dot3 = Dot().move_to(line3.get_left())
55: (12)                               label1 = Tex("Ease In").next_to(line1, RIGHT)
56: (12)                               label2 = Tex("Ease out").next_to(line2, RIGHT)
57: (12)                               label3 = Tex("Ease In Out").next_to(line3, RIGHT)
58: (12)                               self.play(
59: (16)                                 FadeIn(VGroup(line1, line2, line3)),
60: (16)                                 FadeIn(VGroup(dot1, dot2, dot3)),
61: (16)                                 Write(VGroup(label1, label2, label3)),
62: (12)                               )
63: (12)                               self.play(
64: (16)                                 MoveAlongPath(dot1, line1,
rate_func=rate_functions.ease_in_sine),
65: (16)                                 MoveAlongPath(dot2, line2,
rate_func=rate_functions.ease_out_sine),
66: (16)                                 MoveAlongPath(dot3, line3,
rate_func=rate_functions.ease_in_out_sine),
67: (16)                                 run_time=7
68: (12)                               )
69: (12)                               self.wait()
70: (0)                               """
71: (0)                               from __future__ import annotations
72: (0)                               __all__ = [
73: (4)                                 "linear",
74: (4)                                 "smooth",
75: (4)                                 "smoothstep",
76: (4)                                 "smootherstep",
77: (4)                                 "smoothererstep",
78: (4)                                 "rush_into",
79: (4)                                 "rush_from",

```

```

80: (4)             "slow_into",
81: (4)             "double_smooth",
82: (4)             "there_and_back",
83: (4)             "there_and_back_with_pause",
84: (4)             "running_start",
85: (4)             "not Quite there",
86: (4)             "wiggle",
87: (4)             "squish_rate_func",
88: (4)             "lingering",
89: (4)             "exponential_decay",
90: (0)
91: (0)
92: (0)         import typing
93: (0)         from functools import wraps
94: (0)         from math import sqrt
95: (0)         import numpy as np
96: (0)         from ..utils.bezier import bezier
97: (0)         from ..utils.simple_functions import sigmoid
def unit_interval(function):
    @wraps(function)
    def wrapper(t, *args, **kwargs):
        if 0 <= t <= 1:
            return function(t, *args, **kwargs)
        elif t < 0:
            return 0
        else:
            return 1
    return wrapper
def zero(function):
    @wraps(function)
    def wrapper(t, *args, **kwargs):
        if 0 <= t <= 1:
            return function(t, *args, **kwargs)
        else:
            return 0
    return wrapper
@unit_interval
def linear(t: float) -> float:
    return t
@unit_interval
def smooth(t: float, inflection: float = 10.0) -> float:
    error = sigmoid(-inflection / 2)
    return min(
        max((sigmoid(inflection * (t - 0.5)) - error) / (1 - 2 * error), 0),
        1,
    )
def smoothstep(t: float) -> float:
    """Implementation of the 1st order SmoothStep sigmoid function.
    The 1st derivative (speed) is zero at the endpoints.
    https://en.wikipedia.org/wiki/Smoothstep
    """
    return 0 if t <= 0 else 3 * t**2 - 2 * t**3 if t < 1 else 1
def smootherstep(t: float) -> float:
    """Implementation of the 2nd order SmoothStep sigmoid function.
    The 1st and 2nd derivatives (speed and acceleration) are zero at the
    endpoints.
    https://en.wikipedia.org/wiki/Smoothstep
    """
    return 0 if t <= 0 else 6 * t**5 - 15 * t**4 + 10 * t**3 if t < 1 else 1
def smootherstep(t: float) -> float:
    """Implementation of the 3rd order SmoothStep sigmoid function.
    The 1st, 2nd and 3rd derivatives (speed, acceleration and jerk) are zero
    at the endpoints.
    https://en.wikipedia.org/wiki/Smoothstep
    """
    alpha = 0
    if 0 < t < 1:
        alpha = 35 * t**4 - 84 * t**5 + 70 * t**6 - 20 * t**7
    elif t >= 1:
        alpha = 1

```

```

147: (4)             return alpha
148: (0)             @unit_interval
149: (0)             def rush_into(t: float, inflection: float = 10.0) -> float:
150: (4)                 return 2 * smooth(t / 2.0, inflection)
151: (0)             @unit_interval
152: (0)             def rush_from(t: float, inflection: float = 10.0) -> float:
153: (4)                 return 2 * smooth(t / 2.0 + 0.5, inflection) - 1
154: (0)             @unit_interval
155: (0)             def slow_into(t: float) -> float:
156: (4)                 return np.sqrt(1 - (1 - t) * (1 - t))
157: (0)             @unit_interval
158: (0)             def double_smooth(t: float) -> float:
159: (4)                 if t < 0.5:
160: (8)                     return 0.5 * smooth(2 * t)
161: (4)                 else:
162: (8)                     return 0.5 * (1 + smooth(2 * t - 1))
163: (0)             @zero
164: (0)             def there_and_back(t: float, inflection: float = 10.0) -> float:
165: (4)                 new_t = 2 * t if t < 0.5 else 2 * (1 - t)
166: (4)                 return smooth(new_t, inflection)
167: (0)             @zero
168: (0)             def there_and_back_with_pause(t: float, pause_ratio: float = 1.0 / 3) ->
float:
169: (4)                 a = 1.0 / pause_ratio
170: (4)                 if t < 0.5 - pause_ratio / 2:
171: (8)                     return smooth(a * t)
172: (4)                 elif t < 0.5 + pause_ratio / 2:
173: (8)                     return 1
174: (4)                 else:
175: (8)                     return smooth(a - a * t)
176: (0)             @unit_interval
177: (0)             def running_start(
178: (4)                 t: float,
179: (4)                 pull_factor: float = -0.5,
180: (0)             ) -> typing.Iterable: # what is func return type?
181: (4)                 return bezier([0, 0, pull_factor, pull_factor, 1, 1, 1])(t)
182: (0)             def not Quite There(
183: (4)                 func: typing.Callable[[float], float] = smooth,
184: (4)                 proportion: float = 0.7,
185: (0)             ) -> typing.Callable[[float], float]:
186: (4)                 def result(t):
187: (8)                     return proportion * func(t)
188: (4)                 return result
189: (0)             @zero
190: (0)             def wiggle(t: float, wiggles: float = 2) -> float:
191: (4)                 return there_and_back(t) * np.sin(wiggles * np.pi * t)
192: (0)             def squish_rate_func(
193: (4)                 func: typing.Callable[[float], float],
194: (4)                 a: float = 0.4,
195: (4)                 b: float = 0.6,
196: (0)             ) -> typing.Callable[[float], float]:
197: (4)                 def result(t):
198: (8)                     if a == b:
199: (12)                         return a
200: (8)                     if t < a:
201: (12)                         return func(0)
202: (8)                     elif t > b:
203: (12)                         return func(1)
204: (8)                     else:
205: (12)                         return func((t - a) / (b - a))
206: (4)                 return result
207: (0)             @unit_interval
208: (0)             def lingering(t: float) -> float:
209: (4)                 return squish_rate_func(lambda t: t, 0, 0.8)(t)
210: (0)             @unit_interval
211: (0)             def exponential_decay(t: float, half_life: float = 0.1) -> float:
212: (4)                 return 1 - np.exp(-t / half_life)
213: (0)             @unit_interval
214: (0)             def ease_in_sine(t: float) -> float:

```

```

215: (4)             return 1 - np.cos((t * np.pi) / 2)
216: (0)             @unit_interval
217: (0)             def ease_out_sine(t: float) -> float:
218: (4)                 return np.sin((t * np.pi) / 2)
219: (0)             @unit_interval
220: (0)             def ease_in_out_sine(t: float) -> float:
221: (4)                 return -(np.cos(np.pi * t) - 1) / 2
222: (0)             @unit_interval
223: (0)             def ease_in_quad(t: float) -> float:
224: (4)                 return t * t
225: (0)             @unit_interval
226: (0)             def ease_out_quad(t: float) -> float:
227: (4)                 return 1 - (1 - t) * (1 - t)
228: (0)             @unit_interval
229: (0)             def ease_in_out_quad(t: float) -> float:
230: (4)                 return 2 * t * t if t < 0.5 else 1 - pow(-2 * t + 2, 2) / 2
231: (0)             @unit_interval
232: (0)             def ease_in_cubic(t: float) -> float:
233: (4)                 return t * t * t
234: (0)             @unit_interval
235: (0)             def ease_out_cubic(t: float) -> float:
236: (4)                 return 1 - pow(1 - t, 3)
237: (0)             @unit_interval
238: (0)             def ease_in_out_cubic(t: float) -> float:
239: (4)                 return 4 * t * t * t if t < 0.5 else 1 - pow(-2 * t + 2, 3) / 2
240: (0)             @unit_interval
241: (0)             def ease_in_quart(t: float) -> float:
242: (4)                 return t * t * t * t
243: (0)             @unit_interval
244: (0)             def ease_out_quart(t: float) -> float:
245: (4)                 return 1 - pow(1 - t, 4)
246: (0)             @unit_interval
247: (0)             def ease_in_out_quart(t: float) -> float:
248: (4)                 return 8 * t * t * t * t if t < 0.5 else 1 - pow(-2 * t + 2, 4) / 2
249: (0)             @unit_interval
250: (0)             def ease_in_quint(t: float) -> float:
251: (4)                 return t * t * t * t * t
252: (0)             @unit_interval
253: (0)             def ease_out_quint(t: float) -> float:
254: (4)                 return 1 - pow(1 - t, 5)
255: (0)             @unit_interval
256: (0)             def ease_in_out_quint(t: float) -> float:
257: (4)                 return 16 * t * t * t * t * t if t < 0.5 else 1 - pow(-2 * t + 2, 5) / 2
258: (0)             @unit_interval
259: (0)             def ease_in_expo(t: float) -> float:
260: (4)                 return 0 if t == 0 else pow(2, 10 * t - 10)
261: (0)             @unit_interval
262: (0)             def ease_out_expo(t: float) -> float:
263: (4)                 return 1 if t == 1 else 1 - pow(2, -10 * t)
264: (0)             @unit_interval
265: (0)             def ease_in_out_expo(t: float) -> float:
266: (4)                 if t == 0:
267: (8)                     return 0
268: (4)                 elif t == 1:
269: (8)                     return 1
270: (4)                 elif t < 0.5:
271: (8)                     return pow(2, 20 * t - 10) / 2
272: (4)                 else:
273: (8)                     return (2 - pow(2, -20 * t + 10)) / 2
274: (0)             @unit_interval
275: (0)             def ease_in_circ(t: float) -> float:
276: (4)                 return 1 - sqrt(1 - pow(t, 2))
277: (0)             @unit_interval
278: (0)             def ease_out_circ(t: float) -> float:
279: (4)                 return sqrt(1 - pow(t - 1, 2))
280: (0)             @unit_interval
281: (0)             def ease_in_out_circ(t: float) -> float:
282: (4)                 return (
283: (8)                     (1 - sqrt(1 - pow(2 * t, 2))) / 2

```

```

284: (8)             if t < 0.5
285: (8)                 else (sqrt(1 - pow(-2 * t + 2, 2)) + 1) / 2
286: (4)
287: (0)             )
288: (0)         @unit_interval
289: (4)             def ease_in_back(t: float) -> float:
290: (4)                 c1 = 1.70158
291: (4)                 c3 = c1 + 1
292: (0)                 return c3 * t * t * t - c1 * t * t
293: (0)             @unit_interval
294: (4)             def ease_out_back(t: float) -> float:
295: (4)                 c1 = 1.70158
296: (4)                 c3 = c1 + 1
297: (0)                 return 1 + c3 * pow(t - 1, 3) + c1 * pow(t - 1, 2)
298: (0)             @unit_interval
299: (4)             def ease_in_out_back(t: float) -> float:
300: (4)                 c1 = 1.70158
301: (4)                 c2 = c1 * 1.525
302: (0)                 return (
303: (8)                     (pow(2 * t, 2) * ((c2 + 1) * 2 * t - c2)) / 2
304: (8)                     if t < 0.5
305: (4)                     else (pow(2 * t - 2, 2) * ((c2 + 1) * (t * 2 - 2) + c2) + 2) / 2
306: (0)
307: (0)             @unit_interval
308: (4)             def ease_in_elastic(t: float) -> float:
309: (4)                 c4 = (2 * np.pi) / 3
310: (4)                 if t == 0:
311: (8)                     return 0
312: (8)                 elif t == 1:
313: (8)                     return 1
314: (4)                 else:
315: (8)                     return -pow(2, 10 * t - 10) * np.sin((t * 10 - 10.75) * c4)
316: (0)             @unit_interval
317: (4)             def ease_out_elastic(t: float) -> float:
318: (4)                 c4 = (2 * np.pi) / 3
319: (4)                 if t == 0:
320: (8)                     return 0
321: (8)                 elif t == 1:
322: (8)                     return 1
323: (4)                 else:
324: (8)                     return pow(2, -10 * t) * np.sin((t * 10 - 0.75) * c4) + 1
325: (0)             @unit_interval
326: (4)             def ease_in_out_elastic(t: float) -> float:
327: (4)                 c5 = (2 * np.pi) / 4.5
328: (4)                 if t == 0:
329: (8)                     return 0
330: (8)                 elif t == 1:
331: (8)                     return 1
332: (4)                 elif t < 0.5:
333: (8)                     return -(pow(2, 20 * t - 10) * np.sin((20 * t - 11.125) * c5)) / 2
334: (4)                 else:
335: (8)                     return (pow(2, -20 * t + 10) * np.sin((20 * t - 11.125) * c5)) / 2 + 1
336: (0)             @unit_interval
337: (4)             def ease_in_bounce(t: float) -> float:
338: (4)                 return 1 - ease_out_bounce(1 - t)
339: (0)             @unit_interval
340: (4)             def ease_out_bounce(t: float) -> float:
341: (4)                 n1 = 7.5625
342: (4)                 d1 = 2.75
343: (4)                 if t < 1 / d1:
344: (8)                     return n1 * t * t
345: (4)                 elif t < 2 / d1:
346: (8)                     return n1 * (t - 1.5 / d1) * (t - 1.5 / d1) + 0.75
347: (4)                 elif t < 2.5 / d1:
348: (8)                     return n1 * (t - 2.25 / d1) * (t - 2.25 / d1) + 0.9375
349: (4)                 else:
350: (8)                     return n1 * (t - 2.625 / d1) * (t - 2.625 / d1) + 0.984375
351: (0)             @unit_interval
352: (4)             def ease_in_out_bounce(t: float) -> float:
353: (4)                 if t < 0.5:

```

```

353: (8)             return (1 - ease_out_bounce(1 - 2 * t)) / 2
354: (4)         else:
355: (8)             return (1 + ease_out_bounce(2 * t - 1)) / 2
-----
```

## File 151 - module\_parsing.py:

```

1: (0)             """Read and parse all the Manim modules and extract documentation from
them."""
2: (0)             from __future__ import annotations
3: (0)             import ast
4: (0)             from pathlib import Path
5: (0)             from typing_extensions import TypeAlias
6: (0)             __all__ = ["parse_module_attributes"]
7: (0)             AliasInfo: TypeAlias = dict[str, str]
8: (0)             """Dictionary with a `definition` key containing the definition of
9: (0)             a :class:`TypeAlias` as a string, and optionally a `doc` key containing
10: (0)            the documentation for that alias, if it exists.
11: (0)
12: (0)             AliasCategoryDict: TypeAlias = dict[str, AliasInfo]
13: (0)             """Dictionary which holds an `AliasInfo` for every alias name in a same
14: (0)            category.
15: (0)
16: (0)             ModuleLevelAliasDict: TypeAlias = dict[str, AliasCategoryDict]
17: (0)             """Dictionary containing every :class:`TypeAlias` defined in a module,
18: (0)            classified by category in different `AliasCategoryDict` objects.
19: (0)
20: (0)             AliasDocsDict: TypeAlias = dict[str, ModuleLevelAliasDict]
21: (0)             """Dictionary which, for every module in Manim, contains documentation
22: (0)            about their module-level attributes which are explicitly defined as
23: (0)            :class:`TypeAlias`, separating them from the rest of attributes.
24: (0)
25: (0)             DataDict: TypeAlias = dict[str, list[str]]
26: (0)             """Type for a dictionary which, for every module, contains a list with
27: (0)            the names of all their DOCUMENTED module-level attributes (identified
28: (0)            by Sphinx via the ``data`` role, hence the name) which are NOT
29: (0)            explicitly defined as :class:`TypeAlias`.
30: (0)
31: (0)             ALIAS_DOCS_DICT: AliasDocsDict = {}
32: (0)             DATA_DICT: DataDict = {}
33: (0)             MANIM_ROOT = Path(__file__).resolve().parent.parent.parent
34: (0)             def parse_module_attributes() -> tuple[AliasDocsDict, DataDict]:
35: (4)                 """Read all files, generate Abstract Syntax Trees from them, and
36: (4)                 extract useful information about the type aliases defined in the
37: (4)                 files: the category they belong to, their definition and their
38: (4)                 description, separating them from the "regular" module attributes.
39: (4)
40: (4)             Returns
41: (4)             -----
42: (8)             ALIAS_DOCS_DICT : `AliasDocsDict`
43: (8)                 A dictionary containing the information from all the type
44: (8)                 aliases in Manim. See `AliasDocsDict` for more information.
45: (8)             DATA_DICT : `DataDict`
46: (8)                 A dictionary containing the names of all DOCUMENTED
47: (8)                 module-level attributes which are not a :class:`TypeAlias`.
48: (4)
49: (4)             global ALIAS_DOCS_DICT
50: (4)             global DATA_DICT
51: (8)             if ALIAS_DOCS_DICT or DATA_DICT:
52: (8)                 return ALIAS_DOCS_DICT, DATA_DICT
53: (8)             for module_path in MANIM_ROOT.rglob("*.py"):
54: (8)                 module_name = module_path.resolve().relative_to(MANIM_ROOT)
55: (8)                 module_name = list(module_name.parts)
56: (8)                 module_name[-1] = module_name[-1].removesuffix(".py")
57: (8)                 module_name = ".".join(module_name)
58: (8)                 module_content = module_path.read_text(encoding="utf-8")
59: (8)                 module_dict: ModuleLevelAliasDict = {}
60: (8)                 category_dict: AliasCategoryDict | None = None
61: (8)                 alias_info: AliasInfo | None = None
```

```

61: (8)             data_list: list[str] = []
62: (8)             data_name: str | None = None
63: (8)             for node in ast.iter_child_nodes(ast.parse(module_content)):
64: (12)                 if (
65: (16)                     type(node) is ast.Expr
66: (16)                     and type(node.value) is ast.Constant
67: (16)                     and type(node.value.value) is str
68: (12)                 ):
69: (16)                     string = node.value.value.strip()
70: (16)                     section_str = "[CATEGORY]"
71: (16)                     if string.startswith(section_str):
72: (20)                         category_name = string[len(section_str) :].strip()
73: (20)                         module_dict[category_name] = {}
74: (20)                         category_dict = module_dict[category_name]
75: (20)                         alias_info = None
76: (16)                     elif alias_info:
77: (20)                         alias_info["doc"] = string
78: (16)                     elif data_name:
79: (20)                         data_list.append(data_name)
80: (16)                         continue
81: (12)                     if (
82: (16)                         type(node) is ast.If
83: (16)                         and (
84: (20)                             (
85: (24)                                 type(node.test) is ast.Name
86: (24)                                 and node.test.id == "TYPE_CHECKING"
87: (20)                             )
88: (20)                         or (
89: (24)                             type(node.test) is ast.Attribute
90: (24)                             and type(node.test.value) is ast.Name
91: (24)                             and node.test.value.id == "typing"
92: (24)                             and node.test.attr == "TYPE_CHECKING"
93: (20)                         )
94: (16)                     )
95: (12)                 ):
96: (16)                     inner_nodes = node.body
97: (12)                 else:
98: (16)                     inner_nodes = [node]
99: (12)                 for node in inner_nodes:
100: (16)                     if (
101: (20)                         type(node) is ast.AnnAssign
102: (20)                         and type(node.annotation) is ast.Name
103: (20)                         and node.annotation.id == "TypeAlias"
104: (20)                         and type(node.target) is ast.Name
105: (20)                         and node.value is not None
106: (16)                     ):
107: (20)                         alias_name = node.target.id
108: (20)                         def_node = node.value
109: (20)                         if (
110: (24)                             type(def_node) is ast.Subscript
111: (24)                             and type(def_node.value) is ast.Name
112: (24)                             and def_node.value.id == "Union"
113: (20)                         ):
114: (24)                             definition = " | ".join(
115: (28)                               ast.unparse(elem) for elem in def_node.slice.elts
116: (24)                             )
117: (20)                         else:
118: (24)                             definition = ast.unparse(def_node)
119: (20)                         definition = definition.replace("npt.", "")
120: (20)                         if category_dict is None:
121: (24)                             module_dict[""] = {}
122: (24)                             category_dict = module_dict[""]
123: (20)                             category_dict[alias_name] = {"definition": definition}
124: (20)                             alias_info = category_dict[alias_name]
125: (20)                             continue
126: (16)                         alias_info = None
127: (16)                         if type(node) is ast.AnnAssign:
128: (20)                             target = node.target
129: (16)                         elif type(node) is ast.Assign and len(node.targets) == 1:

```

```

130: (20)           target = node.targets[0]
131: (16)         else:
132: (20)             target = None
133: (16)             if type(target) is ast.Name:
134: (20)                 data_name = target.id
135: (16)             else:
136: (20)                 data_name = None
137: (8)             if len(module_dict) > 0:
138: (12)                 ALIAS_DOCS_DICT[module_name] = module_dict
139: (8)             if len(data_list) > 0:
140: (12)                 DATA_DICT[module_name] = data_list
141: (4)         return ALIAS_DOCS_DICT, DATA_DICT
-----
```

## File 152 - manim\_directive.py:

```

1: (0)           """
2: (0)           A directive for including Manim videos in a Sphinx document
3: (0)           =====
4: (0)           When rendering the HTML documentation, the ``.. manim::`` directive
5: (0)           implemented here allows to include rendered videos.
6: (0)           Its basic usage that allows processing **inline content** looks as follows::
7: (0)
8: (4)           .. manim:: MyScene
9: (8)               class MyScene(Scene):
10: (12)                   def construct(self):
11: (16)                       ...
12: (0)           It is required to pass the name of the class representing the scene to be rendered to the directive.
13: (0)
14: (0)           As a second application, the directive can also be used to render scenes that are defined within doctests, for example::
15: (0)               .. manim:: DirectiveDoctestExample
16: (4)                   :ref_classes: Dot
17: (8)                   >>> from manim import Create, Dot, RED, Scene
18: (8)                   >>> dot = Dot(color=RED)
19: (8)                   >>> dot.color
20: (8)                   ManimColor('#FC6255')
21: (8)                   >>> class DirectiveDoctestExample(Scene):
22: (8)                       ...     def construct(self):
23: (8)                           ...         self.play(Create(dot))
24: (8)
25: (0)           Options
26: (0)           -----
27: (0)           Options can be passed as follows::
28: (4)           .. manim:: <Class name>
29: (8)               :<option name>: <value>
30: (0)           The following configuration options are supported by the directive:
31: (0)
32: (4)               hide_source
33: (8)                   If this flag is present without argument, the source code is not displayed above the rendered video.
34: (8)
35: (4)               no_autoplay
36: (8)                   If this flag is present without argument, the video will not autoplay.
37: (8)
38: (4)               quality : {'low', 'medium', 'high', 'fourk'}
39: (8)                   Controls render quality of the video, in analogy to the corresponding command line flags.
40: (8)
41: (4)               save_as_gif
42: (8)                   If this flag is present without argument, the scene is rendered as a gif.
43: (8)
44: (4)               save_last_frame
45: (8)                   If this flag is present without argument, an image representing the last frame of the scene will be rendered and displayed, instead of a video.
46: (8)
47: (8)               ref_classes
48: (8)                   A list of classes, separated by spaces, that is rendered in a reference block after the source code.
49: (8)
50: (8)               ref_functions
51: (4)                   A list of functions, separated by spaces,
```

```

53: (8)                         that is rendered in a reference block after the source code.
54: (4)             ref_methods
55: (8)                 A list of methods, separated by spaces,
56: (8)                 that is rendered in a reference block after the source code.
57: (0)
58: (0)             """
59: (0)                 from __future__ import annotations
60: (0)                 import csv
61: (0)                 import itertools as it
62: (0)                 import os
63: (0)                 import re
64: (0)                 import shutil
65: (0)                 import sys
66: (0)                 import textwrap
67: (0)                 from pathlib import Path
68: (0)                 from timeit import timeit
69: (0)                 from typing import TYPE_CHECKING, Any
70: (0)                 import jinja2
71: (0)                 from docutils import nodes
72: (0)                 from docutils.parsers.rst import Directive, directives # type: ignore
73: (0)                 from docutils.statemachine import StringList
74: (0)                 from manim import QUALITIES
75: (0)                 from manim import __version__ as manim_version
76: (0)             if TYPE_CHECKING:
77: (4)                 from sphinx.application import Sphinx
78: (0)             __all__ = ["ManimDirective"]
79: (0)             classnamedict: dict[str, int] = {}
80: (0)             class SkipManimNode(nodes.Admonition, nodes.Element):
81: (4)                 """Auxiliary node class that is used when the ``skip-manim`` tag is
82: (4)                 present
83: (4)                     or ``.pot`` files are being built.
84: (4)                     Skips rendering the manim directive and outputs a placeholder instead.
85: (0)             def visit(self: SkipManimNode, node: nodes.Element, name: str = "") -> None:
86: (4)                 self.visit_admonition(node, name)
87: (4)                 if not isinstance(node[0], nodes.title):
88: (8)                     node.insert(0, nodes.title("skip-manim", "Example Placeholder"))
89: (0)             def depart(self: SkipManimNode, node: nodes.Element) -> None:
90: (4)                 self.depart_admonition(node)
91: (0)             def process_name_list(option_input: str, reference_type: str) -> list[str]:
92: (4)                 """Reformats a string of space separated class names
93: (4)                 as a list of strings containing valid Sphinx references.
94: (4)                 Tests
95: (4)                 -----
96: (4)                 :::
97: (8)                     >>> process_name_list("Tex TexTemplate", "class")
98: (8)                     [':class:`~.Tex`', ':class:`~.TexTemplate`"]
99: (8)                     >>> process_name_list("Scene.play Mobject.rotate", "func")
100: (8)                     [':func:`~.Scene.play`', ':func:`~.Mobject.rotate`"]
101: (0)                 """
102: (4)             return [f":{reference_type}:`~.{name}`" for name in option_input.split()]
103: (0)             class ManimDirective(Directive):
104: (4)                 """The manim directive, rendering videos while building
105: (4)                 the documentation.
106: (4)                 See the module docstring for documentation.
107: (4)                 """
108: (4)                 has_content = True
109: (4)                 required_arguments = 1
110: (4)                 optional_arguments = 0
111: (4)                 option_spec = {
112: (8)                     "hide_source": bool,
113: (8)                     "no_autoplay": bool,
114: (8)                     "quality": lambda arg: directives.choice(
115: (12)                         arg,
116: (12)                         ("low", "medium", "high", "fourk"),
117: (8)                     ),
118: (8)                     "save_as_gif": bool,
119: (8)                     "save_last_frame": bool,
120: (8)                     "ref_modules": lambda arg: process_name_list(arg, "mod"),

```

```

121: (8)             "ref_classes": lambda arg: process_name_list(arg, "class"),
122: (8)             "ref_functions": lambda arg: process_name_list(arg, "func"),
123: (8)             "ref_methods": lambda arg: process_name_list(arg, "meth"),
124: (4)         }
125: (4)         final_argument_whitespace = True
126: (4)         def run(self) -> list[nodes.Element]:
127: (8)             should_skip = (
128: (12)                 "skip-manim" in
self.state.document.settings.env.app.builder.tags.tags
129: (12)                     or self.state.document.settings.env.app.builder.name == "gettext"
130: (8)         )
131: (8)         if should_skip:
132: (12)             clsname = self.arguments[0]
133: (12)             node = SkipManimNode()
134: (12)             self.state.nested_parse(
135: (16)                 StringList(
136: (20)                     [
137: (24)                         f"Placeholder block for ``{clsname}``.",
138: (24)                         "",
139: (24)                         ".. code-block:: python",
140: (24)                         "",
141: (20)                     ]
142: (20)                     + [    " + line for line in self.content]
143: (20)                     +
144: (24)                         "",
145: (24)                         ".. raw:: html",
146: (24)                         "",
147: (24)                         f'      <pre data-manim-binder data-manim-classname="'
{clsname}">',
148: (20)                     ]
149: (20)                     + [    " + line for line in self.content]
150: (20)                     +
151: (16)                         [
152: (16)                             "
```

```

188: (8)             source_rel_name = source_file_name.relative_to(setup.confdir)
189: (8)             source_rel_dir = source_rel_name.parents[0]
190: (8)             dest_dir = Path(setup.app.builder.outdir, source_rel_dir).absolute()
191: (8)             if not dest_dir.exists():
192: (12)                 dest_dir.mkdir(parents=True, exist_ok=True)
193: (8)             source_block = [
194: (12)                 "... code-block:: python",
195: (12)                 "",
196: (12)                 "    from manim import *\n",
197: (12)                 *(("        " + line for line in self.content),
198: (12)                 ""),
199: (12)                 "... raw:: html",
200: (12)                 "",
201: (12)                 f'        <pre data-manim-binder data-manim-classname="{classname}">',
202: (12)                 *(("        " + line for line in self.content),
203: (12)                 "",
204: (12)                 "        </pre>",
205: (8)             ]
206: (8)             source_block = "\n".join(source_block)
207: (8)             config.media_dir = (Path(setup.confdir) / "media").absolute()
208: (8)             config.images_dir = "{media_dir}/images"
209: (8)             config.video_dir = "{media_dir}/videos/{quality}"
210: (8)             output_file = f"{classname}-{classnamesdict[classname]}"
211: (8)             config.assets_dir = Path("_static")
212: (8)             config.progress_bar = "none"
213: (8)             config.verbosity = "WARNING"
214: (8)             example_config = {
215: (12)                 "frame_rate": frame_rate,
216: (12)                 "no_autoplay": no_autoplay,
217: (12)                 "pixel_height": pixel_height,
218: (12)                 "pixel_width": pixel_width,
219: (12)                 "save_last_frame": save_last_frame,
220: (12)                 "write_to_movie": not save_last_frame,
221: (12)                 "output_file": output_file,
222: (8)             }
223: (8)             if save_last_frame:
224: (12)                 example_config["format"] = None
225: (8)             if save_as_gif:
226: (12)                 example_config["format"] = "gif"
227: (8)             user_code = self.content
228: (8)             if user_code[0].startswith("">>>> "): # check whether block comes from
doctest
229: (12)
230: (16)
"...""))
231: (12)
232: (8)
233: (12)
234: (12)
235: (12)
236: (8)
237: (8)
238: (12)
239: (16)
number=1)
240: (16)
241: (16)
242: (8)
243: (12)
from e
244: (8)
245: (12)
246: (12)
247: (12)
248: (8)
249: (8)
250: (12)
251: (12)
252: (12)
              with tempconfig(example_config):
                  run_time = timeit(lambda: exec("\n".join(code), globals()),
video_dir = config.get_dir("video_dir")
images_dir = config.get_dir("images_dir")
except Exception as e:
    raise RuntimeError(f"Error while rendering example {classname}")
from e
              _write_rendering_stats(
                  classname,
                  run_time,
                  self.state.document.settings.env.docname,
)
if not (save_as_gif or save_last_frame):
    filename = f"{output_file}.mp4"
    filesrc = video_dir / filename
    destfile = Path(dest_dir, filename)

```

```

253: (12)                     shutil.copyfile(filesrc, destfile)
254: (8)                      elif save_as_gif:
255: (12)                        filename = f"{output_file}.gif"
256: (12)                        filesrc = video_dir / filename
257: (8)                      elif save_last_frame:
258: (12)                        filename = f"{output_file}.png"
259: (12)                        filesrc = images_dir / filename
260: (8)                    else:
261: (12)                      raise ValueError("Invalid combination of render flags received.")
262: (8)        rendered_template = jinja2.Template(TEMPLATE).render(
263: (12)          clsname=clsname,
264: (12)          clsname_lowercase=clsname.lower(),
265: (12)          hide_source=hide_source,
266: (12)          filesrc_rel=Path(filesrc).relative_to(setup.confdir).as_posix(),
267: (12)          no_autoplay=no_autoplay,
268: (12)          output_file=output_file,
269: (12)          save_last_frame=save_last_frame,
270: (12)          save_as_gif=save_as_gif,
271: (12)          source_block=source_block,
272: (12)          ref_block=ref_block,
273: (8)      )
274: (8)      state_machine.insert_input(
275: (12)        rendered_template.split("\n"),
276: (12)        source=document.attributes["source"],
277: (8)      )
278: (8)    return []
279: (0)  rendering_times_file_path = Path("../rendering_times.csv")
280: (0)  def _write_rendering_stats(scene_name: str, run_time: str, file_name: str) ->
None:
281: (4)    with rendering_times_file_path.open("a") as file:
282: (8)      csv.writer(file).writerow(
283: (12)        [
284: (16)          re.sub(r"^(reference\/)|(manim\.)", "", file_name),
285: (16)          scene_name,
286: (16)          "%.3f" % run_time,
287: (12)        ],
288: (8)      )
289: (0)  def _log_rendering_times(*args: tuple[Any]) -> None:
290: (4)    if rendering_times_file_path.exists():
291: (8)      with rendering_times_file_path.open() as file:
292: (12)        data = list(csv.reader(file))
293: (8)        if len(data) == 0:
294: (12)          sys.exit()
295: (8)        print("\nRendering Summary\n-----\n")
296: (8)        data = [row for row in data if row]
297: (8)        max_file_length = max(len(row[0]) for row in data)
298: (8)        for key, group in it.groupby(data, key=lambda row: row[0]):
299: (12)          key = key.ljust(max_file_length + 1, ".")
300: (12)          group = list(group)
301: (12)          if len(group) == 1:
302: (16)            row = group[0]
303: (16)            print(f"{key}{row[2].rjust(7, '.')}s {row[1]}")
304: (16)            continue
305: (12)          time_sum = sum(float(row[2]) for row in group)
306: (12)          print(
307: (16)            f"{key}{f'{time_sum:.3f}'.rjust(7, '.')}s => {len(group)}"
308: (12)          )
309: (12)          for row in group:
310: (16)            print(f"{' ' * max_file_length} {row[2].rjust(7)}s {row[1]}")
311: (8)          print("")
312: (0)  def _delete_rendering_times(*args: tuple[Any]) -> None:
313: (4)    if rendering_times_file_path.exists():
314: (8)      rendering_times_file_path.unlink()
315: (0)  def setup(app: Sphinx) -> dict[str, Any]:
316: (4)    app.add_node(SkipManimNode, html=(visit, depart))
317: (4)    setup.app = app
318: (4)    setup.config = app.config
319: (4)    setup.confdir = app.confdir

```

```

320: (4)             app.add_directive("manim", ManimDirective)
321: (4)             app.connect("builder-initiated", _delete_rendering_times)
322: (4)             app.connect("build-finished", _log_rendering_times)
323: (4)             app.add_js_file("manim-binder.min.js")
324: (4)             app.add_js_file(
325: (8)                 None,
326: (8)                 body=textwrap.dedent(
327: (12)                     f"""
328: (16)                         window.initManimBinder({{branch: "v{{manim_version}}"}})
329: (12)                     """
330: (8)                 ).strip(),
331: (4)
332: (4)             )
333: (4)             metadata = {"parallel_read_safe": False, "parallel_write_safe": True}
334: (0)             return metadata
335: (0)             TEMPLATE = r"""
336: (0)             {% if not hide_source %}
337: (4)               .. raw:: html
338: (4)                 <div id="{{ classname_lowercase }}" class="admonition admonition-manim-
example">
339: (4)                   <p class="admonition-title">Example: {{ classname }} <a class="headerlink"
400: (#{{ classname_lowercase }})">¶</a></p>
340: (0)                   {% endif %}
341: (0)                   {% if not (save_as_gif or save_last_frame) %}
342: (4)                     .. raw:: html
343: (8)                       <video
344: (8)                           class="manim-video"
345: (8)                           controls
346: (8)                           loop
347: (8)                           {{ '' if no_autoplay else 'autoplay' }}
348: (8)                           src="./{{ output_file }}.mp4">
349: (4)                     </video>
350: (0)                     {% elif save_as_gif %}
351: (0)                       .. image:: /{{ filesrc_rel }}
352: (0)                           :align: center
353: (0)                     {% elif save_last_frame %}
354: (0)                       .. image:: /{{ filesrc_rel }}
355: (0)                           :align: center
356: (0)                     {% endif %}
357: (0)                     {% if not hide_source %}
358: (0)                       {{ source_block }}
359: (0)                     {{ ref_block }}
360: (4)                     .. raw:: html
361: (0)                       </div>
362: (0)                     {% endif %}
363: (0)             """

```

---

#### File 153 - simple\_functions.py:

```

1: (0)             """A collection of simple functions."""
2: (0)             from __future__ import annotations
3: (0)             __all__ = [
4: (4)                 "binary_search",
5: (4)                 "choose",
6: (4)                 "clip",
7: (4)                 "sigmoid",
8: (0)
9: (0)             import inspect
10: (0)            from functools import lru_cache
11: (0)            from types import MappingProxyType
12: (0)            from typing import Callable
13: (0)            import numpy as np
14: (0)            from scipy import special
15: (0)            def binary_search(
16: (4)                function: Callable[[int | float], int | float],
17: (4)                target: int | float,
18: (4)                lower_bound: int | float,
19: (4)                upper_bound: int | float,

```

```

20: (4)             tolerance: int | float = 1e-4,
21: (0)         ) -> int | float | None:
22: (4)             """Searches for a value in a range by repeatedly dividing the range in
half.
23: (4)             To be more precise, performs numerical binary search to determine the
24: (4)             input to ``function``, between the bounds given, that outputs ``target``
25: (4)             to within ``tolerance`` (default of 0.0001).
26: (4)             Returns ``None`` if no input can be found within the bounds.
27: (4)             Examples
28: (4)             -----
29: (4)             Consider the polynomial :math:`x^2 + 3x + 1` where we search for
30: (4)             a target value of :math:`11`. An exact solution is :math:`x = 2`.
31: (4)             :::
32: (8)                 >>> solution = binary_search(lambda x: x**2 + 3*x + 1, 11, 0, 5)
33: (8)                 >>> abs(solution - 2) < 1e-4
34: (8)                 True
35: (8)                 >>> solution = binary_search(lambda x: x**2 + 3*x + 1, 11, 0, 5,
tolerance=0.01)
36: (8)                 >>> abs(solution - 2) < 0.01
37: (8)                 True
38: (4)             Searching in the interval :math:`[0, 5]` for a target value of :math:`71`
39: (4)             does not yield a solution:::
40: (8)                 >>> binary_search(lambda x: x**2 + 3*x + 1, 71, 0, 5) is None
41: (8)                 True
42: (4)             """
43: (4)             lh = lower_bound
44: (4)             rh = upper_bound
45: (4)             mh = np.mean(np.array([lh, rh]))
46: (4)             while abs(rh - lh) > tolerance:
47: (8)                 mh = np.mean(np.array([lh, rh]))
48: (8)                 lx, mx, rx = (function(h) for h in (lh, mh, rh))
49: (8)                 if lx == target:
50: (12)                     return lh
51: (8)                 if rx == target:
52: (12)                     return rh
53: (8)                 if lx <= target <= rx:
54: (12)                     if mx > target:
55: (16)                         rh = mh
56: (12)                     else:
57: (16)                         lh = mh
58: (8)                     elif lx > target > rx:
59: (12)                         lh, rh = rh, lh
60: (8)                     else:
61: (12)                         return None
62: (4)             return mh
63: (0)             @lru_cache(maxsize=10)
64: (0)             def choose(n: int, k: int) -> int:
65: (4)                 """The binomial coefficient n choose k.
66: (4)                 :math:`\binom{n}{k}` describes the number of possible choices of
67: (4)                 :math:`k` elements from a set of :math:`n` elements.
68: (4)                 References
69: (4)                 -----
70: (4)                 - https://en.wikipedia.org/wiki/Combination
71: (4)
https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.comb.html
72: (4)             """
73: (4)             return special.comb(n, k, exact=True)
74: (0)             def clip(a, min_a, max_a):
75: (4)                 """Clips ``a`` to the interval [``min_a``, ``max_a``].
76: (4)                 Accepts any comparable objects (i.e. those that support <, >).
77: (4)                 Returns ``a`` if it is between ``min_a`` and ``max_a``.
78: (4)                 Otherwise, whichever of ``min_a`` and ``max_a`` is closest.
79: (4)                 Examples
79: (4)                 -----
80: (4)                 :::
81: (4)                 >>> clip(15, 11, 20)
82: (8)                 15
83: (8)                 >>> clip('a', 'h', 'k')
84: (8)                 'h'
85: (8)

```

```
manims_installed_to_implement_with_qhenomenology_SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOME...
86: (4)          """
87: (4)          if a < min_a:
88: (8)              return min_a
89: (4)          elif a > max_a:
90: (8)              return max_a
91: (4)          return a
92: (0)      def sigmoid(x: float) -> float:
93: (4)          """Returns the output of the logistic function.
94: (4)          The logistic function, a common example of a sigmoid function, is defined
95: (4)          as :math:`\frac{1}{1 + e^{-x}}`."
96: (4)          References
97: (4)          -----
98: (4)          - https://en.wikipedia.org/wiki/Sigmoid\_function
99: (4)          - https://en.wikipedia.org/wiki/Logistic\_function
100: (4)         """
101: (4)         return 1.0 / (1 + np.exp(-x))
```

-----  
File 154 - parameter\_parsing.py:

```
1: (0)          from __future__ import annotations
2: (0)          from types import GeneratorType
3: (0)          from typing import Iterable, TypeVar
4: (0)          T = TypeVar("T")
5: (0)      def flatten_iterable_parameters(
6: (4)          args: Iterable[T | Iterable[T] | GeneratorType],
7: (0)      ) -> list[T]:
8: (4)          """Flattens an iterable of parameters into a list of parameters.
9: (4)          Parameters
10: (4)          -----
11: (4)          args
12: (8)              The iterable of parameters to flatten.
13: (8)              [(generator), [], (), ...]
14: (4)          Returns
15: (4)          -----
16: (4)          :class:`list`
17: (8)              The flattened list of parameters.
18: (4)          """
19: (4)          flattened_parameters = []
20: (4)          for arg in args:
21: (8)              if isinstance(arg, (Iterable, GeneratorType)):
22: (12)                  flattened_parameters.extend(arg)
23: (8)              else:
24: (12)                  flattened_parameters.append(arg)
25: (4)          return flattened_parameters
```

-----  
File 155 - \_\_init\_\_.py:

```
1: (0)          """Utilities for Manim tests using `pytest <https://pytest.org>`_.
2: (0)          For more information about Manim testing, see:
3: (0)          - :doc:`/contributing/development`, specifically the ``Tests`` bullet
4: (4)              point under :ref:`polishing-changes-and-submitting-a-pull-request`
5: (0)          - :doc:`/contributing/testing`
6: (0)          .. autosummary::
7: (3)              :toctree: ../reference
8: (3)              frames_comparison
9: (3)              _frames_testers
10: (3)              _show_diff
11: (3)              _test_class_makers
12: (0)          """
```

-----  
File 156 - tex.py:

```
1: (0)          """Utilities for processing LaTeX templates."""
```

```

2: (0)         from __future__ import annotations
3: (0)         __all__ = [
4: (4)             "TexTemplate",
5: (0)         ]
6: (0)         import copy
7: (0)         import re
8: (0)         import warnings
9: (0)         from dataclasses import dataclass, field
10: (0)        from pathlib import Path
11: (0)        from typing import TYPE_CHECKING, Any
12: (0)        if TYPE_CHECKING:
13: (4)            from typing_extensions import Self
14: (4)            from manim.typing import StrPath
15: (0)        _DEFAULT_PREAMBLE = r"""\usepackage[english]{babel}
16: (0)        \usepackage{amsmath}
17: (0)        \usepackage{amssymb}"""
18: (0)        _BEGIN_DOCUMENT = r"\begin{document}"
19: (0)        _END_DOCUMENT = r"\end{document}"
20: (0)        @dataclass(eq=True)
21: (0)        class TexTemplate:
22: (4)            """TeX templates are used to create ``Tex`` and ``MathTex`` objects."""
23: (4)            body: str = field(default="", init=False)
24: (4)            """A custom body, can be set from a file."""
25: (4)            tex_compiler: str = "latex"
26: (4)            """The TeX compiler to be used, e.g. ``latex``, ``pdflatex`` or
27: (4)            ``lualatex``."""
28: (4)            output_format: str = ".dvi"
29: (4)            """The output format resulting from compilation, e.g. ``.dvi`` or
30: (4)            ``.pdf``."""
31: (4)            documentclass: str = r"\documentclass[preview]{standalone}"
32: (4)            """The command defining the documentclass, e.g. ``\documentclass[preview]
33: (4)            {standalone}``."""
34: (4)            preamble: str = _DEFAULT_PREAMBLE
35: (4)            """The document's preamble, i.e. the part between ``\documentclass`` and
36: (4)            ``\begin{document}``."""
37: (4)            placeholder_text: str = "YourTextHere"
38: (4)            """Text in the document that will be replaced by the expression to be
39: (8)            rendered."""
40: (8)            post_doc_commands: str = ""
41: (12)           """Text (definitions, commands) to be inserted at right after
42: (16)           ``\begin{document}``, e.g. ``\boldsymbol{}``."""
43: (16)           @property
44: (20)           def body(self) -> str:
45: (20)               """The entire TeX template."""
46: (20)               return self._body or "\n".join(
47: (20)                   filter(
48: (20)                       None,
49: (20)                       [
50: (16)                           self.documentclass,
51: (16)                           self.preamble,
52: (16)                           _BEGIN_DOCUMENT,
53: (16)                           self.post_doc_commands,
54: (16)                           self.placeholder_text,
55: (16)                           _END_DOCUMENT,
56: (16)                       ],
57: (12)                   )
58: (8)               )
59: (4)               @body.setter
60: (4)               def body(self, value: str) -> None:
61: (8)                   self._body = value
62: (4)               @classmethod
63: (4)               def from_file(cls, file: StrPath = "tex_template.tex", **kwargs: Any) ->
Self:
64: (8)                   """Create an instance by reading the content of a file.
65: (8)                   Using the ``add_to_preamble`` and ``add_to_document`` methods on this
66: (8)                   instance
67: (8)                   will have no effect, as the body is read from the file.
68: (8)                   """
69: (8)                   instance = cls(**kwargs)

```

```

63: (8)                     instance.body = Path(file).read_text(encoding="utf-8")
64: (8)                     return instance
65: (4)             def add_to_preamble(self, txt: str, prepend: bool = False) -> Self:
66: (8)                 r"""Adds text to the TeX template's preamble (e.g. definitions,
packages). Text can be inserted at the beginning or at the end of the preamble.
67: (8)             Parameters
68: (8)             -----
69: (8)             txt
70: (12)                 String containing the text to be added, e.g.
```\usepackage{hyperref}``.
71: (8)             prepend
72: (12)                 Whether the text should be added at the beginning of the preamble,
i.e. right after ```\documentclass```.
73: (12)                 Default is to add it at the end of the preamble, i.e. right before
```\begin{document}```.
74: (8)             """
75: (8)             if self._body:
76: (12)                 warnings.warn(
77: (16)                     "This TeX template was created with a fixed body, trying to
add text the preamble will have no effect.",
78: (16)                     UserWarning,
79: (16)                     stacklevel=2,
80: (12)                 )
81: (8)             if prepend:
82: (12)                 self.preamble = txt + "\n" + self.preamble
83: (8)
84: (12)             else:
85: (8)                 self.preamble += "\n" + txt
86: (8)             return self
87: (4)             def add_to_document(self, txt: str) -> Self:
88: (8)                 r"""Adds text to the TeX template just after \begin{document}, e.g.
```\boldmath```.
89: (8)             Parameters
90: (8)             -----
91: (8)             txt
92: (12)                 String containing the text to be added.
93: (8)             """
94: (12)             if self._body:
95: (16)                 warnings.warn(
96: (16)                     "This TeX template was created with a fixed body, trying to
add text the document will have no effect.",
97: (16)                     UserWarning,
98: (16)                     stacklevel=2,
99: (12)                 )
100: (8)                 self.post_doc_commands += txt
101: (8)             return self
102: (4)             def get_texcode_for_expression(self, expression: str) -> str:
103: (8)                 r"""Inserts expression verbatim into TeX template.
104: (8)             Parameters
105: (8)             -----
106: (12)             expression
107: (8)                 The string containing the expression to be typeset, e.g.
```\$\\sqrt{2}```
108: (8)             Returns
109: (8)             -----
110: (8)             :class:`str`
111: (12)                 LaTeX code based on current template, containing the given
```expression`` and ready for typesetting
112: (8)             """
113: (8)                 return self.body.replace(self.placeholder_text, expression)
114: (4)             def get_texcode_for_expression_in_env(
115: (8)                 self, expression: str, environment: str
116: (4)             ) -> str:
117: (8)                 r"""Inserts expression into TeX template wrapped in
```\begin{environment}``` and ```\end{environment}```.
118: (8)             Parameters
119: (8)             -----
120: (12)             expression
121: (8)                 The string containing the expression to be typeset, e.g.
```\$\\sqrt{2}```.

```

```

121: (8)           environment
122: (12)         The string containing the environment in which the expression
should be typeset, e.g. ``align``.
123: (8)           Returns
124: (8)           -----
125: (8)           :class:`str`
126: (12)         LaTeX code based on template, containing the given expression
inside its environment, ready for typesetting
127: (8)           """
128: (8)           begin, end = _texcode_for_environment(environment)
129: (8)           return self.body.replace(
130: (12)             self.placeholder_text, "\n".join([begin, expression, end]))
131: (8)       )
132: (4)   def copy(self) -> Self:
133: (8)       """Create a deep copy of the TeX template instance."""
134: (8)       return copy.deepcopy(self)
135: (0)   def _texcode_for_environment(environment: str) -> tuple[str, str]:
136: (4)       r"""Processes the tex_environment string to return the correct
``\begin{environment}{extra}{extra}`` and
``\end{environment}`` strings.
137: (4)   Parameters
138: (4)   -----
139: (4)   environment
140: (4)       The tex_environment as a string. Acceptable formats include:
141: (8)       ``{align*}``, ``align```, ``{tabular}[t]{cccl}``, ``tabular}{cccl```,
``\begin{tabular}[t]{cccl}``.
142: (4)       Returns
143: (4)       -----
144: (4)   Tuple[:class:`str`, :class:`str`]
145: (8)       A pair of strings representing the opening and closing of the tex
environment, e.g.
146: (8)           ``\begin{tabular}{cccl}`` and ``\end{tabular}``
147: (8)           """
148: (4)
149: (4)       environment.removeprefix(r"\begin").removeprefix("{")
150: (4)       begin = r"\begin{" + environment
151: (4)       if not begin.endswith("}", "]")):
152: (8)           begin += "}"
153: (4)       split_at_brace = re.split("}", environment, 1)
154: (4)       end = r"\end{" + split_at_brace[0] + "}"
155: (4)       return begin, end

```

-----

## File 157 - unit.py:

```

1: (0)       """Implement the Unit class."""
2: (0)       from __future__ import annotations
3: (0)       import numpy as np
4: (0)       from .. import config, constants
5: (0)       __all__ = ["Pixels", "Degrees", "Munits", "Percent"]
6: (0)       class _PixelUnits:
7: (4)           def __mul__(self, val):
8: (8)               return val * config.frame_width / config.pixel_width
9: (4)           def __rmul__(self, val):
10: (8)               return val * config.frame_width / config.pixel_width
11: (0)       class Percent:
12: (4)           def __init__(self, axis):
13: (8)               if np.array_equal(axis, constants.X_AXIS):
14: (12)                   self.length = config.frame_width
15: (8)               if np.array_equal(axis, constants.Y_AXIS):
16: (12)                   self.length = config.frame_height
17: (8)               if np.array_equal(axis, constants.Z_AXIS):
18: (12)                   raise NotImplementedError("length of Z axis is undefined")
19: (4)           def __mul__(self, val):
20: (8)               return val / 100 * self.length
21: (4)           def __rmul__(self, val):
22: (8)               return val / 100 * self.length
23: (0)       Pixels = _PixelUnits()
24: (0)       Degrees = constants.PI / 180

```

25: (0)

Munits = 1

-----  
File 158 - \_show\_diff.py:

```

1: (0)          from __future__ import annotations
2: (0)          import logging
3: (0)          import warnings
4: (0)          import numpy as np
5: (0)          def show_diff_helper(
6: (4)              frame_number: int,
7: (4)              frame_data: np.ndarray,
8: (4)              expected_frame_data: np.ndarray,
9: (4)              control_data_filename: str,
10: (0)          ):
11: (4)              """Will visually display with matplotlib differences between frame
generated and the one expected."""
12: (4)          import matplotlib.gridspec as gridspec
13: (4)          import matplotlib.pyplot as plt
14: (4)          gs = gridspec.GridSpec(2, 2)
15: (4)          fig = plt.figure()
16: (4)          fig.suptitle(f"Test difference summary at frame {frame_number}",
fontsize=16)
17: (4)          ax = fig.add_subplot(gs[0, 0])
18: (4)          ax.imshow(frame_data)
19: (4)          ax.set_title("Generated")
20: (4)          ax = fig.add_subplot(gs[0, 1])
21: (4)          ax.imshow(expected_frame_data)
22: (4)          ax.set_title("Expected")
23: (4)          ax = fig.add_subplot(gs[1, :])
24: (4)          diff_im = expected_frame_data.copy()
25: (4)          diff_im = np.where(
26: (8)              frame_data != np.array([0, 0, 0, 255]),
27: (8)              np.array([0, 255, 0, 255], dtype="uint8"),
28: (8)              np.array([0, 0, 0, 255], dtype="uint8"),
29: (4)          ) # Set any non-black pixels to green
30: (4)          np.putmask(
31: (8)              diff_im,
32: (8)              expected_frame_data != frame_data,
33: (8)              np.array([255, 0, 0, 255], dtype="uint8"),
34: (4)          ) # Set any different pixels to red
35: (4)          ax.imshow(diff_im, interpolation="nearest")
36: (4)          ax.set_title("Difference summary: (green = same, red = different)")
37: (4)          with warnings.catch_warnings():
38: (8)              warnings.simplefilter("error")
39: (8)          try:
40: (12)              plt.show()
41: (8)          except UserWarning:
42: (12)              filename = f"{control_data_filename[:-4]}-diff.pdf"
43: (12)              plt.savefig(filename)
44: (12)              logging.warning(
45: (16)                  "Interactive matplotlib interface not available,"
46: (16)                  f" diff saved to {filename}."
47: (12)          )

```

-----  
File 159 - tex\_templates.py:

```

1: (0)          """A library of LaTeX templates."""
2: (0)          from __future__ import annotations
3: (0)          __all__ = [
4: (4)              "TexTemplateLibrary",
5: (4)              "TexFontTemplates",
6: (0)          ]
7: (0)          from .tex import *
8: (0)          def _new_ams_template():
9: (4)              """Returns a simple Tex Template with only basic AMS packages"""

```

```

10: (4)             preamble = r"""
11: (0)             \usepackage[english]{babel}
12: (0)             \usepackage{amsmath}
13: (0)             \usepackage{amssymb}
14: (0)             """
15: (4)             return TexTemplate(preamble=preamble)
16: (0)             """ Tex Template preamble used by original upstream 3b1b """
17: (0)             _3b1b_preamble = r"""
18: (0)             \usepackage[english]{babel}
19: (0)             \usepackage[utf8]{inputenc}
20: (0)             \usepackage[T1]{fontenc}
21: (0)             \usepackage{lmodern}
22: (0)             \usepackage{amsmath}
23: (0)             \usepackage{amssymb}
24: (0)             \usepackage{dsfont}
25: (0)             \usepackage{setspace}
26: (0)             \usepackage{tipa}
27: (0)             \usepackage{relsize}
28: (0)             \usepackage{textcomp}
29: (0)             \usepackage{mathrsfs}
30: (0)             \usepackage{calligra}
31: (0)             \usepackage{wasysym}
32: (0)             \usepackage{ragged2e}
33: (0)             \usepackage{physics}
34: (0)             \usepackage{xcolor}
35: (0)             \usepackage{microtype}
36: (0)             \DisableLigatures{encoding = *, family = * }
37: (0)             \linespread{1}
38: (0)             """
39: (0)             class TexTemplateLibrary:
40: (4)             """
41: (4)                 A collection of basic TeX template objects
42: (4)                 Examples
43: (4)                 -----
44: (4)                 Normal usage as a value for the keyword argument tex_template of Tex() and
MathTex() mobjects::
45: (8)                 ``Tex("My TeX code", tex_template=TexTemplateLibrary.ctex)``
46: (4)                 """
47: (4)                 default = TexTemplate(preamble=_3b1b_preamble)
48: (4)                 """An instance of the default TeX template in manim"""
49: (4)                 threeb1b = TexTemplate(preamble=_3b1b_preamble)
50: (4)                 """ An instance of the default TeX template used by 3b1b """
51: (4)                 ctex = TexTemplate(
52: (8)                     tex_compiler="xelatex",
53: (8)                     output_format=".xdv",
54: (8)                     preamble=_3b1b_preamble.replace(
55: (12)                         r"\DisableLigatures{encoding = *, family = * }",
56: (12)                         r"\usepackage[UTF8]{ctex}",
57: (8)                     ),
58: (4)                 )
59: (4)                 """An instance of the TeX template used by 3b1b when using the use_ctex
flag"""
60: (4)                 simple = _new_ams_template()
61: (4)                 """An instance of a simple TeX template with only basic AMS packages
loaded"""
62: (0)                 lmtpt = _new_ams_template()
63: (0)                 lmtpt.description = "Latin Modern Typewriter Proportional"
64: (0)                 lmtpt.add_to_preamble(
65: (4)                     r"""
66: (0)                     \usepackage[T1]{fontenc}
67: (0)                     \usepackage[variablett]{lmodern}
68: (0)                     \renewcommand{\rmdefault}{\ttdefault}
69: (0)                     \usepackage[LGRgreek]{mathastext}
70: (0)                     \MTgreekfont{lmtt} % no lgr lmvtt, so use lgr lmtt
71: (0)                     \Mathastext
72: (0)                     \let\varepsilon\epsilon % only \varsigma in LGR
73: (0)                     """",
74: (0)                 )
75: (0)                 fufug = _new_ams_template()

```

```

76: (0)         fufug.description = "Fourier Utopia (Fourier upright Greek)"
77: (0)         fufug.add_to_preamble(
78: (4)             r"""
79: (0)             \usepackage[T1]{fontenc}
80: (0)             \usepackage[upright]{fourier}
81: (0)             \usepackage{mathastext}
82: (0)             """,
83: (0)
84: (0)             )
85: (0)             droidserif = _new_ams_template()
86: (0)             droidserif.description = "Droid Serif"
87: (0)             droidserif.add_to_preamble(
88: (4)                 r"""
89: (0)                 \usepackage[T1]{fontenc}
90: (0)                 \usepackage[default]{droidserif}
91: (0)                 \usepackage[LGRgreek]{mathastext}
92: (0)                 \let\varepsilon\epsilon
93: (0)                 """,
94: (0)
95: (0)                 )
96: (0)                 droidsans = _new_ams_template()
97: (0)                 droidsans.description = "Droid Sans"
98: (0)                 droidsans.add_to_preamble(
99: (4)                     r"""
100: (0)                     \usepackage[T1]{fontenc}
101: (0)                     \usepackage[default]{droidsans}
102: (0)                     \usepackage[LGRgreek]{mathastext}
103: (0)                     \let\varepsilon\epsilon
104: (0)
105: (0)                     )
106: (0)                     ncsg = _new_ams_template()
107: (4)                     ncsg.description = "New Century Schoolbook (Symbol Greek)"
108: (0)                     ncsg.add_to_preamble(
109: (0)                         r"""
110: (0)                         \usepackage[T1]{fontenc}
111: (0)                         \usepackage[newcent]
112: (0)                         \usepackage[symbolgreek]{mathastext}
113: (0)                         \linespread{1.1}
114: (0)
115: (0)                         )
116: (0)                         fceg = _new_ams_template()
117: (4)                         fceg.description = "French Cursive (Euler Greek)"
118: (0)                         fceg.add_to_preamble(
119: (0)                             r"""
120: (0)                             \usepackage[T1]{fontenc}
121: (0)                             \usepackage[default]{frcursive}
122: (0)                             \usepackage[eulergreek,noplusnominus,noequal,nohbar,%
123: (0)                             nolessnomore,noasterisk]{mathastext}
124: (0)
125: (0)                             )
126: (0)                             aksg = _new_ams_template()
127: (4)                             aksg.description = "Auriocus Kalligraphicus (Symbol Greek)"
128: (0)                             aksg.add_to_preamble(
129: (0)                               r"""
130: (0)                               \usepackage[T1]{fontenc}
131: (0)                               \usepackage[aurical]
132: (0)                               \renewcommand{\rmdefault}{AuriocusKalligraphicus}
133: (0)
134: (0)                               )
135: (0)                               palatinosg = _new_ams_template()
136: (0)                               palatinosg.description = "Palatino (Symbol Greek)"
137: (4)                               palatinosg.add_to_preamble(
138: (0)                                 r"""
139: (0)                                 \usepackage[T1]{fontenc}
140: (0)                                 \usepackage[symbolmax,defaultmathsizes]{mathastext}
141: (0)
142: (0)                                 )
143: (0)                                 comfortaa = _new_ams_template()
144: (0)                                 comfortaa.description = "Comfortaa"

```

```

145: (0)     comfortaa.add_to_preamble(
146: (4)         r"""
147: (0)             \usepackage[default]{comfortaa}
148: (0)             \usepackage[LGRgreek,defaultmathsizes,noasterisk]{mathastext}
149: (0)             \let\varphi\phi
150: (0)             \linespread{1.06}
151: (0)             """
152: (0)
153: (0)             )
154: (0)             ecfaugieeg = _new_ams_template()
155: (0)             ecfaugieeg.description = "ECF Augie (Euler Greek)"
156: (0)             ecfaugieeg.add_to_preamble(
157: (4)                 r"""
158: (0)                     \renewcommand\familydefault\fau} % emerald package
159: (0)                     \usepackage[defaultmathsizes,eulergreek]{mathastext}
160: (0)                     """
161: (0)
162: (0)                     )
163: (0)                     electrumadfcm = _new_ams_template()
164: (0)                     electrumadfcm.description = "Electrum ADF (CM Greek)"
165: (0)                     electrumadfcm.add_to_preamble(
166: (4)                         r"""
167: (0)                           \usepackage[T1]{fontenc}
168: (0)                           \usepackage[LGRgreek,basic,defaultmathsizes]{mathastext}
169: (0)                           \usepackage[lf]{electrum}
170: (0)                           \Mathastext
171: (0)                           \let\varphi\phi
172: (0)                           """
173: (0)
174: (0)                           )
175: (0)                           americantypewriter = _new_ams_template()
176: (0)                           americantypewriter.description = "American Typewriter"
177: (0)                           americantypewriter.add_to_preamble(
178: (4)                             r"""
179: (0)                               \usepackage[no-math]{fontspec}
180: (0)                               \setmainfont[Mapping=tex-text]{American Typewriter}
181: (0)                               \usepackage[defaultmathsizes]{mathastext}
182: (0)                               """
183: (0)
184: (0)                               )
185: (0)                               mpmptx = _new_ams_template()
186: (0)                               mpmptx.description = "Minion Pro and Myriad Pro (and TX fonts symbols)"
187: (0)                               mpmptx.add_to_preamble(
188: (4)                                 r"""
189: (0)                                   \usepackage{txfonts}
190: (0)                                   \usepackage[upright]{txgreeks}
191: (0)                                   \usepackage[no-math]{fontspec}
192: (0)                                   \setmainfont[Mapping=tex-text]{Minion Pro}
193: (0)                                   \setsansfont[Mapping=tex-text,Scale=MatchUppercase]{Myriad Pro}
194: (0)                                   \renewcommand\familydefault\sfdefault
195: (0)                                   \usepackage[defaultmathsizes]{mathastext}
196: (0)                                   \renewcommand\familydefault\rmdefault
197: (0)                                   """
198: (0)
199: (0)                                   )
200: (0)                                   ncsgpxm = _new_ams_template()
201: (0)                                   ncsgpxm.description = "New Century Schoolbook (Symbol Greek, PX math
symbols)"
202: (0)                                   ncsgpxm.add_to_preamble(
203: (4)                                     r"""
204: (0)                                       \usepackage[T1]{fontenc}
205: (0)                                       \usepackage{pxfonts}
206: (0)                                       \usepackage[newcent]
207: (0)                                       \usepackage[symbolgreek,defaultmathsizes]{mathastext}
208: (0)                                       \linespread{1.06}
209: (0)                                       """
210: (0)
211: (0)                                       )
212: (0)                                       vollkorntx = _new_ams_template()
213: (0)                                       vollkorntx.description = "Vollkorn (TX fonts for Greek and math symbols)"
214: (0)                                       vollkorntx.add_to_preamble(

```

```

213: (4)
      r"""
214: (0)     \usepackage[T1]{fontenc}
215: (0)     \usepackage{txfonts}
216: (0)     \usepackage[upright]{txgreeks}
217: (0)     \usepackage{vollkorn}
218: (0)     \usepackage[defaultmathsizes]{mathastext}
219: (0)     """
220: (0)
221: (0)     )
222: (0)     libertine = _new_ams_template()
223: (0)     libertine.description = "Libertine"
224: (4)     libertine.add_to_preamble(
225: (0)         r"""
226: (0)             \usepackage[T1]{fontenc}
227: (0)             \usepackage{libertine}
228: (0)             \usepackage[greek=n]{libgreek}
229: (0)             \usepackage[noasterisk,defaultmathsizes]{mathastext}
230: (0)             """
231: (0)
232: (0)             )
233: (0)             libertine.add_to_preamble(
234: (4)                 r"""
235: (0)                     \usepackage[T1]{fontenc}
236: (0)                     \usepackage{tpslifonts}
237: (0)                     \usepackage[eulergreek,defaultmathsizes]{mathastext}
238: (0)                     \MTEulerScale{1.06}
239: (0)                     \linespread{1.2}
240: (0)                     """
241: (0)
242: (0)                     )
243: (0)                     ecfwebstertx = _new_ams_template()
244: (0)                     ecfwebstertx.description = "ECF Webster (with TX fonts)"
245: (4)                     ecfwebstertx.add_to_preamble(
246: (0)                         r"""
247: (0)                             \usepackage{txfonts}
248: (0)                             \usepackage[upright]{txgreeks}
249: (0)                             \renewcommand\familydefault{\fwb} % emerald package
250: (0)                             \usepackage{mathastext}
251: (0)                             \renewcommand{\int}{\intop\limits}
252: (0)                             \linespread{1.5}
253: (0)                             """
254: (0)
255: (4)                             )
256: (0)                             ecfwebstertx.add_to_document(
257: (0)                               r"""
258: (0)                                 \mathversion{bold}
259: (0)                                 """
260: (0)
261: (0)                                 italicromandeadff = _new_ams_template()
262: (4)                                 italicromandeadff.description = "Romande ADF with Fourier (Italic)"
263: (0)                                 italicromandeadff.add_to_preamble(
264: (0)                                   r"""
265: (0)                                     \usepackage[T1]{fontenc}
266: (0)                                     \usepackage{fourier}
267: (0)                                     \usepackage{romande}
268: (0)                                     \usepackage[italic,defaultmathsizes,noasterisk]{mathastext}
269: (0)                                     \renewcommand{\itshape}{\swashstyle}
270: (0)                                     """
271: (0)
272: (0)                                     applechancery = _new_ams_template()
273: (4)                                     applechancery.description = "Apple Chancery"
274: (0)                                     applechancery.add_to_preamble(
275: (0)                                       r"""
276: (0)                                         \usepackage[no-math]{fontspec}
277: (0)                                         \setmainfont[Mapping=tex-text]{Apple Chancery}
278: (0)                                         \usepackage[defaultmathsizes]{mathastext}
279: (0)                                         """
280: (0)
281: (0)                                         applechancery.tex_compiler = "xelatex"
282: (0)                                         applechancery.output_format = ".xdv"
283: (0)                                         zapfchancery = _new_ams_template()

```

```

282: (0)         zapfchancery.description = "Zapf Chancery"
283: (0)         zapfchancery.add_to_preamble(
284: (4)             r"""
285: (0)             \DeclareFontFamily{T1}{pzc}{}{}{}
286: (0)             \DeclareFontShape{T1}{pzc}{mb}{it}{<->s*[1.2] pzc mi 8t}{}{}
287: (0)             \DeclareFontShape{T1}{pzc}{m}{it}{<->ssub * pzc/mb/it}{}{}
288: (0)             \usepackage{chancery} % = \renewcommand{\rmdefault}{pzc}
289: (0)             \renewcommand\shadefont\itdefault
290: (0)             \renewcommand\bfdefault\mddefault
291: (0)             \usepackage[defaultmathsizes]{mathastext}
292: (0)             \linespread{1.05}
293: (0)             """
294: (0)
295: (0)         italicverdana = _new_ams_template()
296: (0)         italicverdana.description = "Verdana (Italic)"
297: (0)         italicverdana.add_to_preamble(
298: (4)             r"""
299: (0)             \usepackage[no-math]{fontspec}
300: (0)             \setmainfont[Mapping=tex-text]{Verdana}
301: (0)             \usepackage[defaultmathsizes,italic]{mathastext}
302: (0)             """
303: (0)
304: (0)         italicverdana.tex_compiler = "xelatex"
305: (0)         italicverdana.output_format = ".xdv"
306: (0)         urwzccmg = _new_ams_template()
307: (0)         urwzccmg.description = "URW Zapf Chancery (CM Greek)"
308: (0)         urwzccmg.add_to_preamble(
309: (4)             r"""
310: (0)             \usepackage[T1]{fontenc}
311: (0)             \DeclareFontFamily{T1}{pzc}{}{}{}
312: (0)             \DeclareFontShape{T1}{pzc}{mb}{it}{<->s*[1.2] pzc mi 8t}{}{}
313: (0)             \DeclareFontShape{T1}{pzc}{m}{it}{<->ssub * pzc/mb/it}{}{}
314: (0)             \DeclareFontShape{T1}{pzc}{mb}{sl}{<->ssub * pzc/mb/it}{}{}
315: (0)             \DeclareFontShape{T1}{pzc}{m}{sl}{<->ssub * pzc/mb/sl}{}{}
316: (0)             \DeclareFontShape{T1}{pzc}{m}{n}{<->ssub * pzc/mb/it}{}{}
317: (0)             \usepackage{chancery}
318: (0)             \usepackage{mathastext}
319: (0)             \linespread{1.05}""",
320: (0)
321: (0)         urwzccmg.add_to_document(
322: (4)             r"""
323: (0)             \boldmath
324: (0)             """
325: (0)
326: (0)         comicksansms = _new_ams_template()
327: (0)         comicksansms.description = "Comic Sans MS"
328: (0)         comicksansms.add_to_preamble(
329: (4)             r"""
330: (0)             \usepackage[no-math]{fontspec}
331: (0)             \setmainfont[Mapping=tex-text]{Comic Sans MS}
332: (0)             \usepackage[defaultmathsizes]{mathastext}
333: (0)             """
334: (0)
335: (0)         comicksansms.tex_compiler = "xelatex"
336: (0)         comicksansms.output_format = ".xdv"
337: (0)         italicgfsdidot = _new_ams_template()
338: (0)         italicgfsdidot.description = "GFS Didot (Italic)"
339: (0)         italicgfsdidot.add_to_preamble(
340: (4)             r"""
341: (0)             \usepackage[T1]{fontenc}
342: (0)             \renewcommand\rmdefault{udidot}
343: (0)             \usepackage[LGRgreek, defaultmathsizes, italic]{mathastext}
344: (0)             \let\varphi\phi
345: (0)             """
346: (0)
347: (0)         chalkduster = _new_ams_template()
348: (0)         chalkduster.description = "Chalkduster"
349: (0)         chalkduster.add_to_preamble(
350: (4)             r"""

```

```

351: (0)          \usepackage[no-math]{fontspec}
352: (0)          \setmainfont[Mapping=tex-text]{Chalkduster}
353: (0)          \usepackage[defaultmathsizes]{mathastext}
354: (0)          """
355: (0)          )
356: (0)          chalkduster.tex_compiler = "lualatex"
357: (0)          chalkduster.output_format = ".pdf"
358: (0)          mpx = _new_ams_template()
359: (0)          mpx.description = "Minion Pro (and TX fonts symbols)"
360: (0)          mpx.add_to_preamble(
361: (4)            r"""
362: (0)          \usepackage{txfonts}
363: (0)          \usepackage[no-math]{fontspec}
364: (0)          \setmainfont[Mapping=tex-text]{Minion Pro}
365: (0)          \usepackage[defaultmathsizes]{mathastext}
366: (0)          """
367: (0)          )
368: (0)          mpx.tex_compiler = "xelatex"
369: (0)          mpx.output_format = ".xdv"
370: (0)          gnufssfs = _new_ams_template()
371: (0)          gnufssfs.description = "GNU FreeSerif and FreeSans"
372: (0)          gnufssfs.add_to_preamble(
373: (4)            r"""
374: (0)          \usepackage[no-math]{fontspec}
375: (0)          \setmainfont[ExternalLocation,
376: (16)            Mapping=tex-text,
377: (16)            BoldFont=FreeSerifBold,
378: (16)            ItalicFont=FreeSerifItalic,
379: (16)            BoldItalicFont=FreeSerifBoldItalic]{FreeSerif}
380: (0)          \setsansfont[ExternalLocation,
381: (16)            Mapping=tex-text,
382: (16)            BoldFont=FreeSansBold,
383: (16)            ItalicFont=FreeSansOblique,
384: (16)            BoldItalicFont=FreeSansBoldOblique,
385: (16)            Scale=MatchLowercase]{FreeSans}
386: (0)          \renewcommand{\familydefault}{lmss}
387: (0)          \usepackage[LGRgreek,defaultmathsizes,noasterisk]{mathastext}
388: (0)          \renewcommand{\familydefault}{\sfdefault}
389: (0)          \Mathastext
390: (0)          \let\varphi\phi % no `var' phi in LGR encoding
391: (0)          \renewcommand{\familydefault}{\rmdefault}
392: (0)          """
393: (0)
394: (0)          )
395: (0)          gnufssfs.tex_compiler = "xelatex"
396: (0)          gnufssfs.output_format = ".xdv"
397: (0)          gfsneohellenic = _new_ams_template()
398: (0)          gfsneohellenic.description = "GFS NeoHellenic"
399: (4)          gfsneohellenic.add_to_preamble(
400: (0)            r"""
401: (0)          \usepackage[T1]{fontenc}
402: (0)          \renewcommand{\rmdefault}{neohellenic}
403: (0)          \usepackage[LGRgreek]{mathastext}
404: (0)          \let\varphi\phi
405: (0)          \linespread{1.06}
406: (0)
407: (0)          )
408: (0)          ecftallpaul = _new_ams_template()
409: (0)          ecftallpaul.description = "ECF Tall Paul (with Symbol font)"
410: (4)          ecftallpaul.add_to_preamble(
411: (0)            r"""
412: (0)          \DeclareFontFamily{T1}{ftp}{}{}
413: (0)          \DeclareFontShape{T1}{ftp}{m}{n}{
414: (4)            <->s*[1.4] fpmw8t
415: (0)          }{} % increase size by factor 1.4
416: (0)          \renewcommand\familydefault{ftp} % emerald package
417: (0)          \usepackage[symbol]{mathastext}
418: (0)          \let\infty\infty
419: (0)          """
        )

```

```

420: (0)         italicdroidsans = _new_ams_template()
421: (0)         italicdroidsans.description = "Droid Sans (Italic)"
422: (0)         italicdroidsans.add_to_preamble(
423: (4)             r"""
424: (0)             \usepackage[T1]{fontenc}
425: (0)             \usepackage[default]{droidsans}
426: (0)             \usepackage[LGRgreek,defaultmathsizes,italic]{mathastext}
427: (0)             \let\varphi\phi
428: (0)             """
429: (0)
430: (0)         italicbaskerville = _new_ams_template()
431: (0)         italicbaskerville.description = "Baskerville (Italic)"
432: (0)         italicbaskerville.add_to_preamble(
433: (4)             r"""
434: (0)             \usepackage[no-math]{fontspec}
435: (0)             \setmainfont[Mapping=tex-text]{Baskerville}
436: (0)             \usepackage[defaultmathsizes,italic]{mathastext}
437: (0)             """
438: (0)
439: (0)         italicbaskerville.tex_compiler = "xelatex"
440: (0)         italicbaskerville.output_format = ".xdv"
441: (0)         ecfjdtx = _new_ams_template()
442: (0)         ecfjdtx.description = "ECF JD (with TX fonts)"
443: (0)         ecfjdtx.add_to_preamble(
444: (4)             r"""
445: (0)             \usepackage{txfonts}
446: (0)             \usepackage[upright]{txgreeks}
447: (0)             \renewcommand\familydefault\{fjd\} % emerald package
448: (0)             \usepackage{mathastext}
449: (0)             """
450: (0)
451: (0)         ecfjdtx.add_to_document(
452: (4)             r"""\mathversion{bold}
453: (0)             """
454: (0)
455: (0)         aptxgm = _new_ams_template()
456: (0)         aptxgm.description = "Antykwa Półtawskiego (TX Fonts for Greek and math
symbols)"
457: (0)         aptxgm.add_to_preamble(
458: (4)             r"""
459: (0)             \usepackage[OT4,OT1]{fontenc}
460: (0)             \usepackage{txfonts}
461: (0)             \usepackage[upright]{txgreeks}
462: (0)             \usepackage{antpol}
463: (0)             \usepackage[defaultmathsizes,nolessnomore]{mathastext}
464: (0)             """
465: (0)
466: (0)         papyrus = _new_ams_template()
467: (0)         papyrus.description = "Papyrus"
468: (0)         papyrus.add_to_preamble(
469: (4)             r"""
470: (0)             \usepackage[no-math]{fontspec}
471: (0)             \setmainfont[Mapping=tex-text]{Papyrus}
472: (0)             \usepackage[defaultmathsizes]{mathastext}
473: (0)             """
474: (0)
475: (0)         papyrus.tex_compiler = "xelatex"
476: (0)         papyrus.output_format = ".xdv"
477: (0)         gnufstx = _new_ams_template()
478: (0)         gnufstx.description = "GNU FreeSerif (and TX fonts symbols)"
479: (0)         gnufstx.add_to_preamble(
480: (4)             r"""
481: (0)             \usepackage[no-math]{fontspec}
482: (0)             \usepackage{txfonts} \%let\mathbb=\varmathbb
483: (0)             \setmainfont[ExternalLocation,
484: (16)                 Mapping=tex-text,
485: (16)                 BoldFont=FreeSerifBold,
486: (16)                 ItalicFont=FreeSerifItalic,
487: (16)                 BoldItalicFont=FreeSerifBoldItalic]{FreeSerif}

```

```

488: (0)          \usepackage[defaultmathsizes]{mathastext}
489: (0)          """
490: (0)          )
491: (0)          gnufstx.tex_compiler = "xelatex"
492: (0)          gnufstx.output_format = ".pdf"
493: (0)          ecfscmg = _new_ams_template()
494: (0)          ecfscmg.description = "ECF Skeetch (CM Greek)"
495: (0)          ecfscmg.add_to_preamble(
496: (4)          r"""
497: (0)          \usepackage[T1]{fontenc}
498: (0)          \usepackage[T1]{fontenc}
499: (0)          \DeclareFontFamily{T1}{fsk}{}{ }
500: (0)          \DeclareFontShape{T1}{fsk}{m}{n}{<->s*[1.315] fskmw8t}{}
501: (0)          \renewcommand{\rmdefault}{fsk}
502: (0)          \usepackage[noendash,defaultmathsizes,nohbar,defaultimath]{mathastext}
503: (0)          """
504: (0)          )
505: (0)          italiclmtpcm = _new_ams_template()
506: (0)          italiclmtpcm.description = "Latin Modern Typewriter Proportional (CM Greek)
(Italic)"
507: (0)          italiclmtpcm.add_to_preamble(
508: (4)          r"""
509: (0)          \usepackage[T1]{fontenc}
510: (0)          \usepackage[variablett,nomath]{lmodern}
511: (0)          \renewcommand{\familydefault}{\ttdefault}
512: (0)          \usepackage[frenchmath]{mathastext}
513: (0)          \linespread{1.08}
514: (0)          """
515: (0)          )
516: (0)          baskervaldadff = _new_ams_template()
517: (0)          baskervaldadff.description = "Baskervald ADF with Fourier"
518: (0)          baskervaldadff.add_to_preamble(
519: (4)          r"""
520: (0)          \usepackage[upright]{fourier}
521: (0)          \usepackage{baskervald}
522: (0)          \usepackage[defaultmathsizes,noasterisk]{mathastext}
523: (0)          """
524: (0)          )
525: (0)          italicdroidserifpx = _new_ams_template()
526: (0)          italicdroidserifpx.description = "Droid Serif (PX math symbols) (Italic)"
527: (0)          italicdroidserifpx.add_to_preamble(
528: (4)          r"""
529: (0)          \usepackage[T1]{fontenc}
530: (0)          \usepackage{pxfonts}
531: (0)          \usepackage[default]{droidserif}
532: (0)          \usepackage[LGRgreek,defaultmathsizes,italic,basic]{mathastext}
533: (0)          \let\varphi\phi
534: (0)          """
535: (0)          )
536: (0)          biolinum = _new_ams_template()
537: (0)          biolinum.description = "Biolinum"
538: (0)          biolinum.add_to_preamble(
539: (4)          r"""
540: (0)          \usepackage[T1]{fontenc}
541: (0)          \usepackage{libertine}
542: (0)          \renewcommand{\familydefault}{\sfdefault}
543: (0)          \usepackage[greek=n,biolinum]{libgreek}
544: (0)          \usepackage[noasterisk,defaultmathsizes]{mathastext}
545: (0)          """
546: (0)          )
547: (0)          italicvollkornf = _new_ams_template()
548: (0)          italicvollkornf.description = "Vollkorn with Fourier (Italic)"
549: (0)          italicvollkornf.add_to_preamble(
550: (4)          r"""
551: (0)          \usepackage{fourier}
552: (0)          \usepackage{vollkorn}
553: (0)          \usepackage[italic,nohbar]{mathastext}
554: (0)          """
555: (0)          )

```

```

556: (0) chalkboardse = _new_ams_template()
557: (0) chalkboardse.description = "Chalkboard SE"
558: (0) chalkboardse.add_to_preamble(
559: (4)     r"""
560: (0)     \usepackage[no-math]{fontspec}
561: (0)     \setmainfont[Mapping=tex-text]{Chalkboard SE}
562: (0)     \usepackage[defaultmathsizes]{mathastext}
563: (0)     """,
564: (0) )
565: (0) chalkboardse.tex_compiler = "xelatex"
566: (0) chalkboardse.output_format = ".xdv"
567: (0) noteworthylight = _new_ams_template()
568: (0) noteworthylight.description = "Noteworthy Light"
569: (0) noteworthylight.add_to_preamble(
570: (4)     r"""
571: (0)     \usepackage[no-math]{fontspec}
572: (0)     \setmainfont[Mapping=tex-text]{Noteworthy Light}
573: (0)     \usepackage[defaultmathsizes]{mathastext}
574: (0)     """,
575: (0) )
576: (0) epigrafica = _new_ams_template()
577: (0) epigrafica.description = "Epigrafica"
578: (0) epigrafica.add_to_preamble(
579: (4)     r"""
580: (0)     \usepackage[LGR,OT1]{fontenc}
581: (0)     \usepackage{epigrafica}
582: (0)     \usepackage[basic,LGRgreek,defaultmathsizes]{mathastext}
583: (0)     \let\varphi\phi
584: (0)     \linespread{1.2}
585: (0)     """,
586: (0) )
587: (0) librisadff = _new_ams_template()
588: (0) librisadff.description = "Libris ADF with Fourier"
589: (0) librisadff.add_to_preamble(
590: (4)     r"""
591: (0)     \usepackage[T1]{fontenc}
592: (0)     \usepackage[upright]{fourier}
593: (0)     \usepackage{libris}
594: (0)     \renewcommand{\familydefault}{\sfdefault}
595: (0)     \usepackage[noasterisk]{mathastext}
596: (0)     """,
597: (0) )
598: (0) italicvanturisadff = _new_ams_template()
599: (0) italicvanturisadff.description = "Venturis ADF with Fourier (Italic)"
600: (0) italicvanturisadff.add_to_preamble(
601: (4)     r"""
602: (0)     \usepackage{fourier}
603: (0)     \usepackage[lf]{venturis}
604: (0)     \usepackage[italic,defaultmathsizes,noasterisk]{mathastext}
605: (0)     """,
606: (0) )
607: (0) gfsbodoni = _new_ams_template()
608: (0) gfsbodoni.description = "GFS Bodoni"
609: (0) gfsbodoni.add_to_preamble(
610: (4)     r"""
611: (0)     \usepackage[T1]{fontenc}
612: (0)     \renewcommand{\rmdefault}{bodoni}
613: (0)     \usepackage[LGRgreek]{mathastext}
614: (0)     \let\varphi\phi
615: (0)     \linespread{1.06}
616: (0)     """,
617: (0) )
618: (0) brushscriptxpx = _new_ams_template()
619: (0) brushscriptxpx.description = "BrushScriptX-Italic (PX math and Greek)"
620: (0) brushscriptxpx.add_to_preamble(
621: (4)     r"""
622: (0)     \usepackage[T1]{fontenc}
623: (0)     \usepackage{pxfonts}
624: (0)     \%usepackage{pbsi}

```

```

625: (0)          \renewcommand{\rmdefault}{pbsi}
626: (0)          \renewcommand{\mddefault}{x1}
627: (0)          \renewcommand{\bfdefault}{x1}
628: (0)          \usepackage[defaultmathsizes,noasterisk]{mathastext}
629: (0)          """
630: (0)          )
631: (0)          brushscriptpx.add_to_document(
632: (4)            r"""\boldmath
633: (0)          """
634: (0)          )
635: (0)          brushscriptpx.tex_compiler = "xelatex"
636: (0)          brushscriptpx.output_format = ".xdv"
637: (0)          urwagsg = _new_ams_template()
638: (0)          urwagsg.description = "URW Avant Garde (Symbol Greek)"
639: (0)          urwagsg.add_to_preamble(
640: (4)            r"""
641: (0)            \usepackage[T1]{fontenc}
642: (0)            \usepackage{avant}
643: (0)            \renewcommand{\familydefault}{\sfdefault}
644: (0)            \usepackage[symbolgreek,defaultmathsizes]{mathastext}
645: (0)            """
646: (0)            )
647: (0)            italictimesf = _new_ams_template()
648: (0)            italictimesf.description = "Times with Fourier (Italic)"
649: (0)            italictimesf.add_to_preamble(
650: (4)              r"""
651: (0)              \usepackage{fourier}
652: (0)              \renewcommand{\rmdefault}{ptm}
653: (0)              \usepackage[italic,defaultmathsizes,noasterisk]{mathastext}
654: (0)              """
655: (0)              )
656: (0)              italichelveticaaf = _new_ams_template()
657: (0)              italichelveticaaf.description = "Helvetica with Fourier (Italic)"
658: (0)              italichelveticaaf.add_to_preamble(
659: (4)                r"""
660: (0)                \usepackage[T1]{fontenc}
661: (0)                \usepackage[scaled]{helvet}
662: (0)                \usepackage{fourier}
663: (0)                \renewcommand{\rmdefault}{phv}
664: (0)                \usepackage[italic,defaultmathsizes,noasterisk]{mathastext}
665: (0)                """
666: (0)                )
667: (0)                class TexFontTemplates:
668: (4)                  """
669: (4)                  A collection of TeX templates for the fonts described at
http://jf.burnol.free.fr/showcase.html
670: (4)                  These templates are specifically designed to allow you to typeset formulae
and mathematics using
671: (4)                  different fonts. They are based on the mathastext LaTeX package.
672: (4)                  Examples
673: (4)                  -----
674: (4)                  Normal usage as a value for the keyword argument tex_template of Tex() and
MathTex() mobjects::
675: (8)                    ``Tex("My TeX code", tex_template=TexFontTemplates.comic_sans)``
676: (4)                    Notes
677: (4)                    -----
678: (4)                    Many of these templates require that specific fonts
679: (4)                    are installed on your local machine.
680: (4)                    For example, choosing the template TexFontTemplates.comic_sans will
681: (4)                    not compile if the Comic Sans Microsoft font is not installed.
682: (4)                    To experiment, try to render the TexFontTemplateLibrary example scene:
683: (9)                      ``manim path/to/manim/example_scenes/advanced_tex_fonts.py
TexFontTemplateLibrary -p -ql``
684: (4)                      """
685: (4)                      american_typewriter = americantypewriter
686: (4)                      """American Typewriter"""
687: (4)                      antykwa = aptxgm
688: (4)                      """Antykwa Półtawskiego (TX Fonts for Greek and math symbols)"""
689: (4)                      apple_chancery = applechancery

```

```

690: (4)           """Apple Chancery"""
691: (4)           auriocus_kalligraphicus = aksg
692: (4)           """Auriocus Kalligraphicus (Symbol Greek)"""
693: (4)           baskervald_adf_fourier = baskervaldadff
694: (4)           """Baskervald ADF with Fourier"""
695: (4)           baskerville_it = italicbaskerville
696: (4)           """Baskerville (Italic)"""
697: (4)           biolinum = biolinum
698: (4)           """Biolinum"""
699: (4)           brushscriptx = brushscriptpx
700: (4)           """BrushScriptX-Italic (PX math and Greek)"""
701: (4)           chalkboard_se = chalkboardse
702: (4)           """Chalkboard SE"""
703: (4)           chalkduster = chalkduster
704: (4)           """Chalkduster"""
705: (4)           comfortaa = comfortaa
706: (4)           """Comfortaa"""
707: (4)           comic_sans = comicsansms
708: (4)           """Comic Sans MS"""
709: (4)           droid_sans = droidsans
710: (4)           """Droid Sans"""
711: (4)           droid_sans_it = italicdroidsans
712: (4)           """Droid Sans (Italic)"""
713: (4)           droid_serif = droidserif
714: (4)           """Droid Serif"""
715: (4)           droid_serif_px_it = italicdroidserifpx
716: (4)           """Droid Serif (PX math symbols) (Italic)"""
717: (4)           ecf_augie = ecfaugeeg
718: (4)           """ECF Augie (Euler Greek)"""
719: (4)           ecf_jd = ecfjdtx
720: (4)           """ECF JD (with TX fonts)"""
721: (4)           ecf_skeetch = ecfscmg
722: (4)           """ECF Skeetch (CM Greek)"""
723: (4)           ecf_tall_paul = ecftallpaul
724: (4)           """ECF Tall Paul (with Symbol font)"""
725: (4)           ecf_webster = ecfwebstertx
726: (4)           """ECF Webster (with TX fonts)"""
727: (4)           electrum_adf = electrumadfcn
728: (4)           """Electrum ADF (CM Greek)"""
729: (4)           epigrafica = epigrafica
730: (4)           """ Epigrafica """
731: (4)           fourier_utopia = fufug
732: (4)           """Fourier Utopia (Fourier upright Greek)"""
733: (4)           french_cursive = fcsg
734: (4)           """French Cursive (Euler Greek)"""
735: (4)           gfs_bodoni = gfsbodoni
736: (4)           """GFS Bodoni"""
737: (4)           gfs_didot = italicgfsdidot
738: (4)           """GFS Didot (Italic)"""
739: (4)           gfs_neoHellenic = gfsneohellenic
740: (4)           """GFS NeoHellenic"""
741: (4)           gnu_freesans_tx = gnufstx
742: (4)           """GNU FreeSerif (and TX fonts symbols)"""
743: (4)           gnu_freeserif_freesans = gnufsf
744: (4)           """GNU FreeSerif and FreeSans"""
745: (4)           helvetica_fourier_it = italichelvetica
746: (4)           """Helvetica with Fourier (Italic)"""
747: (4)           latin_modern_tw_it = italiclmtc
748: (4)           """Latin Modern Typewriter Proportional (CM Greek) (Italic)"""
749: (4)           latin_modern_tw = lmt
750: (4)           """Latin Modern Typewriter Proportional"""
751: (4)           libertine = libertine
752: (4)           """Libertine"""
753: (4)           libris_adf_fourier = librisadff
754: (4)           """Libris ADF with Fourier"""
755: (4)           minion_pro_myriad_pro = mpmp
756: (4)           """Minion Pro and Myriad Pro (and TX fonts symbols)"""
757: (4)           minion_pro_tx = mptx
758: (4)           """Minion Pro (and TX fonts symbols)"""

```

```

759: (4)             new_century_schoolbook = ncssg
760: (4)             """New Century Schoolbook (Symbol Greek)"""
761: (4)             new_century_schoolbook_px = ncsspxm
762: (4)             """New Century Schoolbook (Symbol Greek, PX math symbols)"""
763: (4)             noteworthy_light = noteworthylight
764: (4)             """Noteworthy Light"""
765: (4)             palatino = palatinosg
766: (4)             """Palatino (Symbol Greek)"""
767: (4)             papyrus = papyrus
768: (4)             """Papyrus"""
769: (4)             romande_adf_fourier_it = italicromandeadff
770: (4)             """Romande ADF with Fourier (Italic)"""
771: (4)             slitex = slitexeg
772: (4)             """SliTeX (Euler Greek)"""
773: (4)             times_fourier_it = italictimesf
774: (4)             """Times with Fourier (Italic)"""
775: (4)             urw_avant_garde = urwagsg
776: (4)             """URW Avant Garde (Symbol Greek)"""
777: (4)             urw_zapf_chancery = urwzccmg
778: (4)             """URW Zapf Chancery (CM Greek)"""
779: (4)             venturis_adf_fourier_it = italicvanturisadff
780: (4)             """Venturis ADF with Fourier (Italic)"""
781: (4)             verdana_it = italicverdana
782: (4)             """Verdana (Italic)"""
783: (4)             vollkorn_fourier_it = italicvollkornf
784: (4)             """Vollkorn with Fourier (Italic)"""
785: (4)             vollkorn = vollkorntx
786: (4)             """Vollkorn (TX fonts for Greek and math symbols)"""
787: (4)             zapf_chancery = zapfchancery
788: (4)             """Zapf Chancery"""

```

---

## File 160 - \_frames\_testers.py:

```

1: (0)         from __future__ import annotations
2: (0)         import contextlib
3: (0)         import warnings
4: (0)         from pathlib import Path
5: (0)         import numpy as np
6: (0)         from manim import logger
7: (0)         from .show_diff import show_diff_helper
8: (0)         FRAME_ABSOLUTE_TOLERANCE = 1.01
9: (0)         FRAME_MISMATCH_RATIO_TOLERANCE = 1e-5
10: (0)        class _FramesTester:
11: (4)            def __init__(self, file_path: Path, show_diff=False) -> None:
12: (8)                self._file_path = file_path
13: (8)                self._show_diff = show_diff
14: (8)                self._frames: np.ndarray
15: (8)                self._number_frames: int = 0
16: (8)                self._frames_compared = 0
17: (4)            @contextlib.contextmanager
18: (4)            def testing(self):
19: (8)                with np.load(self._file_path) as data:
20: (12)                    self._frames = data["frame_data"]
21: (12)                    if len(self._frames.shape) != 4:
22: (16)                        self._frames = np.expand_dims(self._frames, axis=0)
23: (12)                    logger.debug(self._frames.shape)
24: (12)                    self._number_frames = np.ma.size(self._frames, axis=0)
25: (12)                    yield
26: (12)                    assert self._frames_compared == self._number_frames, (
27: (16)                        f"The scene tested contained {self._frames_compared} frames, "
28: (16)                        f"when there are {self._number_frames} control frames for this
test."
29: (12)            )
30: (4)            def check_frame(self, frame_number: int, frame: np.ndarray):
31: (8)                assert frame_number < self._number_frames, (
32: (12)                    f"The tested scene is at frame number {frame_number} "
33: (12)                    f"when there are {self._number_frames} control frames."

```

```

34: (8) )
35: (8) try:
36: (12)     np.testing.assert_allclose(
37: (16)         frame,
38: (16)         self._frames[frame_number],
39: (16)         atol=FRAME_ABSOLUTE_TOLERANCE,
40: (16)         err_msg=f"Frame no {frame_number}. You can use --show_diff to
visually show the difference.", verbose=False,
41: (16)
42: (12)     )
43: (12)     self._frames_compared += 1
44: (8) except AssertionError as e:
45: (12)     number_of_matches = np.isclose(
46: (16)         frame, self._frames[frame_number],
atol=FRAME_ABSOLUTE_TOLERANCE
47: (12)             ).sum()
48: (12)             number_of_mismatches = frame.size - number_of_matches
49: (12)             if number_of_mismatches / frame.size <
FRAME_MISMATCH_RATIO_TOLERANCE:
50: (16)                 self._frames_compared += 1
51: (16)                 warnings.warn(
52: (20)                     f"Mismatch of {number_of_mismatches} pixel values in frame
{frame_number} "
53: (20)                     f"against control data in {self._file_path}. Below error
threshold, "
54: (20)                     "continuing..."
55: (16)
56: (16)             )
57: (12)             return
58: (16)             if self._show_diff:
59: (20)                 show_diff_helper(
60: (20)                     frame_number,
61: (20)                     frame,
62: (20)                     self._frames[frame_number],
63: (20)                     self._file_path.name,
64: (16)
65: (12)             )
66: (0)         raise e
67: class _ControlDataWriter(_FramesTester):
68: (4)     def __init__(self, file_path: Path, size_frame: tuple) -> None:
69: (8)         self.file_path = file_path
70: (8)         self.frames = np.empty((0, *size_frame, 4))
71: (8)         self._number_frames_written: int = 0
72: (4)     def check_frame(self, index: int, frame: np.ndarray):
73: (8)         frame = frame[np.newaxis, ...]
74: (8)         self.frames = np.concatenate((self.frames, frame))
75: (8)         self._number_frames_written += 1
76: (4)     @contextlib.contextmanager
77: (4)     def testing(self):
78: (8)         yield
79: (8)         self.save_control_data()
80: (4)     def save_control_data(self):
81: (8)         self.frames = self.frames.astype("uint8")
82: (8)         np.savez_compressed(self.file_path, frame_data=self.frames)
83: (8)         logger.info(
84: (12)             f"{self._number_frames_written} control frames saved in
{self.file_path}",
85: (8)             )

```

-----

File 161 - tex\_file\_writing.py:

```

1: (0)     """Interface for writing, compiling, and converting ``.tex`` files.
2: (0)     .. SEEALSO::
3: (4)         :mod:`.mobject.svg.tex_mobject`
4: (0)     """
5: (0)     from __future__ import annotations
6: (0)     import hashlib
7: (0)     import os
8: (0)     import re

```

```

9: (0)             import unicodedata
10: (0)            from pathlib import Path
11: (0)            from typing import Iterable
12: (0)            from manim.utils.tex import TexTemplate
13: (0)            from .. import config, logger
14: (0)            __all__ = ["tex_to_svg_file"]
15: (0)            def tex_hash(expression):
16: (4)                id_str = str(expression)
17: (4)                hasher = hashlib.sha256()
18: (4)                hasher.update(id_str.encode())
19: (4)                return hasher.hexdigest()[:16]
20: (0)            def tex_to_svg_file(
21: (4)                expression: str,
22: (4)                environment: str | None = None,
23: (4)                tex_template: TexTemplate | None = None,
24: (0)            ):
25: (4)                """Takes a tex expression and returns the svg version of the compiled tex
26: (4)                Parameters
27: (4)                -----
28: (4)                expression
29: (8)                    String containing the TeX expression to be rendered, e.g.
```\sqrt{2}`` or ``foo``
30: (4)                environment
31: (8)                    The string containing the environment in which the expression should
be typeset, e.g. ``align*``
32: (4)                tex_template
33: (8)                    Template class used to typesetting. If not set, use default template
set via `config["tex_template"]`
34: (4)                    Returns
35: (4)                    -----
36: (4)                    :class:`Path`
37: (8)                    Path to generated SVG file.
38: (4)
39: (4)                    if tex_template is None:
40: (8)                        tex_template = config["tex_template"]
41: (4)                        tex_file = generate_tex_file(expression, environment, tex_template)
42: (4)                        svg_file = tex_file.with_suffix(".svg")
43: (4)                        if svg_file.exists():
44: (8)                            return svg_file
45: (4)                        dvi_file = compile_tex(
46: (8)                            tex_file,
47: (8)                            tex_template.tex_compiler,
48: (8)                            tex_template.output_format,
49: (4)                        )
50: (4)                        svg_file = convert_to_svg(dvi_file, tex_template.output_format)
51: (4)                        if not config["no_latex_cleanup"]:
52: (8)                            delete_nonsvg_files()
53: (4)                        return svg_file
54: (0)            def generate_tex_file(
55: (4)                expression: str,
56: (4)                environment: str | None = None,
57: (4)                tex_template: TexTemplate | None = None,
58: (0)            ) -> Path:
59: (4)                """Takes a tex expression (and an optional tex environment),
60: (4)                and returns a fully formed tex file ready for compilation.
61: (4)                Parameters
62: (4)                -----
63: (4)                expression
64: (8)                    String containing the TeX expression to be rendered, e.g.
```\sqrt{2}`` or ``foo``
65: (4)                environment
66: (8)                    The string containing the environment in which the expression should
be typeset, e.g. ``align*``
67: (4)                tex_template
68: (8)                    Template class used to typesetting. If not set, use default template
set via `config["tex_template"]`
69: (4)                    Returns
70: (4)                    -----
71: (4)                    :class:`Path`
```

```

72: (8)                               Path to generated TeX file
73: (4)
74: (4)
75: (8)
76: (4)
77: (8)
environment)
78: (4)
79: (8)
80: (4)
81: (4)
82: (8)
83: (4)
84: (4)
85: (8)
86: (12)
87: (12)
88: (8)
89: (8)
90: (4)
return result
91: (0)                               def tex_compilation_command(
92: (4)                                 tex_compiler: str, output_format: str, tex_file: Path, tex_dir: Path
93: (0)                               ) -> str:
94: (4)                                 """Prepares the tex compilation command with all necessary cli flags
95: (4)                                 Parameters
96: (4)
97: (4)
tex_compiler
98: (8)                                 String containing the compiler to be used, e.g. ``pdflatex`` or
``lualatex``
99: (4)
100: (8)
``.dvi`` or ``.pdf``
101: (4)
102: (8)
103: (4)
104: (8)
105: (4)
106: (4)
107: (4)
108: (8)
109: (4)
110: (4)
if tex_compiler in {"latex", "pdflatex", "lualatex", "lualatex"}:
111: (8)                                 commands = [
112: (12)                                   tex_compiler,
113: (12)                                   "-interaction=batchmode",
114: (12)                                   f'-output-format="{output_format[1:]}',
115: (12)                                   "-halt-on-error",
116: (12)                                   f'-output-directory="{tex_dir.as_posix()}"',
117: (12)                                   f'"{tex_file.as_posix()}"',
118: (12)                                   ">",
119: (12)                                   os.devnull,
120: (8)
]
121: (4)
122: (8)
123: (12)
124: (8)
125: (12)
126: (8)
127: (12)
raise ValueError("xelatex output is either pdf or xdv")
128: (8)
129: (12)
130: (12)
131: (12)
132: (12)
133: (12)
134: (12)
135: (12)
136: (12)
137: (8)
]

```

```

138: (4)             else:
139: (8)                 raise ValueError(f"Tex compiler {tex_compiler} unknown.")
140: (4)             return " ".join(commands)
141: (0)         def insight_inputenc_error(matching):
142: (4)             code_point = chr(int(matching[1], 16))
143: (4)             name = unicodedata.name(code_point)
144: (4)             yield f"TexTemplate does not support character '{name}' (U+
{matching[1]})."
145: (4)             yield "See the documentation for manim.mobject.svg.tex_mobject for details
on using a custom TexTemplate."
146: (0)         def insight_package_not_found_error(matching):
147: (4)             yield f"You do not have package {matching[1]} installed."
148: (4)             yield f"Install {matching[1]} it using your LaTeX package manager, or
check for typos."
149: (0)     def compile_tex(tex_file: Path, tex_compiler: str, output_format: str) ->
Path:
150: (4)             """Compiles a tex_file into a .dvi or a .xdv or a .pdf
151: (4)             Parameters
152: (4)             -----
153: (4)             tex_file
154: (8)                 File name of TeX file to be typeset.
155: (4)             tex_compiler
156: (8)                 String containing the compiler to be used, e.g. ``pdflatex`` or
``lualatex``
157: (4)
158: (8)             output_format
159: (4)                 String containing the output format generated by the compiler, e.g.
``.dvi`` or ``.pdf``
160: (4)
161: (4)             Returns
162: (4)             -----
163: (4)             :class:`Path`
164: (8)                 Path to generated output file in desired format (DVI, XDV or PDF).
165: (4)
166: (4)
167: (8)             result = tex_file.with_suffix(output_format)
168: (12)             tex_dir = config.get_dir("tex_dir")
169: (12)             if not result.exists():
170: (12)                 command = tex_compilation_command(
171: (12)                     tex_compiler,
172: (8)                         output_format,
173: (8)                         tex_file,
174: (8)                         tex_dir,
175: (12)                     )
176: (12)                     exit_code = os.system(command)
177: (12)                     if exit_code != 0:
178: (16)                         log_file = tex_file.with_suffix(".log")
179: (16)                         print_all_tex_errors(log_file, tex_compiler, tex_file)
180: (16)                         raise ValueError(
181: (12)                             f"{tex_compiler} error converting to"
182: (4)                               f" {output_format[1:]}". See log output above or"
183: (0)                               f" the log file: {log_file}",
184: (4)
185: (4)             return result
186: (4)
187: (4)         def convert_to_svg(dvi_file: Path, extension: str, page: int = 1):
188: (8)             """Converts a .dvi, .xdv, or .pdf file into an svg using dvisvgm.
189: (4)             Parameters
190: (4)             -----
191: (4)             dvi_file
192: (8)                 File name of the input file to be converted.
193: (4)             extension
194: (8)                 String containing the file extension and thus indicating the file
type, e.g. ``.dvi`` or ``.pdf``
195: (4)             page
196: (8)                 Page to be converted if input file is multi-page.
197: (4)
198: (4)             Returns
199: (4)             -----
200: (4)             :class:`Path`
201: (8)                 Path to generated SVG file.
202: (4)
203: (4)             result = dvi_file.with_suffix(".svg")
204: (4)             if not result.exists():

```

```

200: (8)                     commands = [
201: (12)                   "dvisvgm",
202: (12)                   "--pdf" if extension == ".pdf" else "",
203: (12)                   "-p " + str(page),
204: (12)                   f'"{dvi_file.as_posix()}"',
205: (12)                   "-n",
206: (12)                   "-v 0",
207: (12)                   "-o " + f'"{result.as_posix()}"',
208: (12)                   ">",
209: (12)                   os.devnull,
210: (8)                 ]
211: (8)                 os.system(" ".join(commands))
212: (4)                 if not result.exists():
213: (8)                   raise ValueError(
214: (12)                   f"Your installation does not support converting {dvi_file.suffix}
files to SVG."
215: (12)                   f" Consider updating dvisvgm to at least version 2.4."
216: (12)                   f" If this does not solve the problem, please refer to our
troubleshooting guide at:"
217: (12)                   f" https://docs.manim.community/en/stable/faq/general.html#my-
installation-"
218: (12)                   f"does-not-support-converting-pdf-to-svg-help",
219: (8)                 )
220: (4)                 return result
221: (0)             def delete_nonsvg_files(additional_endings: Iterable[str] = ()) -> None:
222: (4)               """Deletes every file that does not have a suffix in `(`".svg", ".tex",
*additional_endings)``
223: (4)               Parameters
224: (4)               -----
225: (4)               additional_endings
226: (8)                 Additional endings to whitelist
227: (4)               """
228: (4)               tex_dir = config.get_dir("tex_dir")
229: (4)               file_suffix_whitelist = {".svg", ".tex", *additional_endings}
230: (4)               for f in tex_dir.iterdir():
231: (8)                 if f.suffix not in file_suffix_whitelist:
232: (12)                   f.unlink()
233: (0)             def print_all_tex_errors(log_file: Path, tex_compiler: str, tex_file: Path) ->
None:
234: (4)               if not log_file.exists():
235: (8)                 raise RuntimeError(
236: (12)                   f"{tex_compiler} failed but did not produce a log file. "
237: (12)                   "Check your LaTeX installation.",
238: (8)                 )
239: (4)               with log_file.open(encoding="utf-8") as f:
240: (8)                 tex_compilation_log = f.readlines()
241: (4)               error_indices = [
242: (8)                 index for index, line in enumerate(tex_compilation_log) if
line.startswith("!")
243: (4)               ]
244: (4)               if error_indices:
245: (8)                 with tex_file.open() as f:
246: (12)                   tex = f.readlines()
247: (8)                   for error_index in error_indices:
248: (12)                       print_tex_error(tex_compilation_log, error_index, tex)
249: (0)             LATEX_ERROR_INSIGHTS = [
250: (4)               (
251: (8)                   r"inputenc Error: Unicode character (?::*) \(\U\+([0-9a-fA-F]+)\)",
252: (8)                   insight_inputenc_error,
253: (4)               ),
254: (4)               (
255: (8)                   r"LaTeX Error: File `(.?#[clsty])` not found",
256: (8)                   insight_package_not_found_error,
257: (4)               ),
258: (0)             ]
259: (0)             def print_tex_error(tex_compilation_log, error_start_index, tex_source):
260: (4)               logger.error(
261: (8)                   f"LaTeX compilation error: {tex_compilation_log[error_start_index]
[2:]}",
```

```

262: (4)                 )
263: (4)                 line_of_tex_error = (
264: (8)                   int(
265: (12)                     [
266: (16)                       log_line
267: (16)                         for log_line in tex_compilation_log[error_start_index:]
268: (16)                           if log_line.startswith("l.")
269: (12)                             ][0]
270: (12)                             .split(" ")[0]
271: (12)                             .split(".")[1],
272: (8)                           )
273: (8)                         - 1
274: (4)                     )
275: (4)                 if line_of_tex_error >= len(tex_source):
276: (8)                   return None
277: (4)                 context = ["Context of error: \n"]
278: (4)                 if line_of_tex_error < 3:
279: (8)                   context += tex_source[: line_of_tex_error + 3]
280: (8)                   context[-4] = "-> " + context[-4]
281: (4)                 elif line_of_tex_error > len(tex_source) - 3:
282: (8)                   context += tex_source[line_of_tex_error - 1 :]
283: (8)                   context[1] = "-> " + context[1]
284: (4)                 else:
285: (8)                   context += tex_source[line_of_tex_error - 3 : line_of_tex_error + 3]
286: (8)                   context[-4] = "-> " + context[-4]
287: (4)                 context = "".join(context)
288: (4)                 logger.error(context)
289: (4)                 for insights in LATEX_ERROR_INSIGHTS:
290: (8)                   prob, get_insight = insights
291: (8)                   matching = re.search(
292: (12)                     prob,
293: (12)                     "".join(tex_compilation_log[error_start_index])[2:],
294: (8)                   )
295: (8)                   if matching is not None:
296: (12)                     for insight in get_insight(matching):
297: (16)                       logger.info(insight)

```

---

#### File 162 - frames\_comparison.py:

```

1: (0)                 from __future__ import annotations
2: (0)                 import functools
3: (0)                 import inspect
4: (0)                 from pathlib import Path
5: (0)                 from typing import Callable
6: (0)                 import cairo
7: (0)                 import pytest
8: (0)                 from _pytest.fixtures import FixtureRequest
9: (0)                 from manim import Scene
10: (0)                from manim._config import tempconfig
11: (0)                from manim._config.utils import ManimConfig
12: (0)                from manim.camera.three_d_camera import ThreeDCamera
13: (0)                from manim.renderer.cairo_renderer import CairoRenderer
14: (0)                from manim.scene.three_d_scene import ThreeDScene
15: (0)                from .frames_testers import _ControlDataWriter, _FramesTester
16: (0)                from .test_class_makers import (
17: (4)                  DummySceneFileWriter,
18: (4)                  _make_scene_file_writer_class,
19: (4)                  _make_test_renderer_class,
20: (4)                  _make_test_scene_class,
21: (0)                )
22: (0)                SCENE_PARAMETER_NAME = "scene"
23: (0)                _tests_root_dir_path = Path(__file__).absolute().parents[2]
24: (0)                PATH_CONTROL_DATA = _tests_root_dir_path / Path("control_data",
25: "graphical_units_data")
25: (0)                MIN_CAIRO_VERSION = 11800
26: (0)                def frames_comparison(
27: (4)                  func=None,

```

```

28: (4)           *,
29: (4)           last_frame: bool = True,
30: (4)           renderer_class=CairoRenderer,
31: (4)           base_scene=Scene,
32: (4)           **custom_config,
33: (0)       ):
34: (4)           """Compares the frames generated by the test with control frames
previously registered.
35: (4)           If there is no control frames for this test, the test will fail. To
generate
36: (4)           control frames for a given test, pass ``--set_test`` flag to pytest
37: (4)           while running the test.
38: (4)           Note that this decorator can be use with or without parentheses.
39: (4)           Parameters
40: (4)           -----
41: (4)           last_frame
42: (8)           whether the test should test the last frame, by default True.
43: (4)           renderer_class
44: (8)           The base renderer to use (OpenGLRenderer/CairoRenderer), by default
CairoRenderer
45: (4)
46: (8)           The base class for the scene (ThreeDScene, etc.), by default Scene
47: (4)           .. warning::
48: (8)           By default, last_frame is True, which means that only the last frame
is tested.
49: (8)           If the scene has a moving animation, then the test must set last_frame
to False.
50: (4)
51: (4)           """
52: (8)           def decorator_maker(tested_scene_construct):
53: (12)             if (
54: (12)               SCENE_PARAMETER_NAME
55: (12)               not in inspect.getfullargspec(tested_scene_construct).args
56: (12)             ):
57: (16)               raise Exception(
58: (12)                 f"Invalid graphical test function test function : must have
'{SCENE_PARAMETER_NAME}' as one of the parameters.",
59: (8)               )
60: (12)               old_sig = inspect.signature(
61: (8)                 functools.partial(tested_scene_construct, scene=None),
62: (8)               )
63: (12)               if "__module_test__" not in tested_scene_construct.__globals__:
64: (16)                 raise Exception(
65: (12)                   "There is no module test name indicated for the graphical unit
test. You have to declare __module_test__ in the test file.",
66: (8)                   )
67: (8)               module_name =
68: (8)               tested_scene_construct.__globals__.get("__module_test__")
69: (8)               test_name = tested_scene_construct.__name__[len("test_") :]
70: (12)               @functools.wraps(tested_scene_construct)
71: (16)               def wrapper(*args, request: FixtureRequest, tmp_path, **kwargs):
72: (16)                 if (
73: (12)                   renderer_class is CairoRenderer
74: (16)                   and cairo.cairo_version() < MIN_CAIRO_VERSION
75: (12)                 ):
76: (12)                   pytest.skip("Cairo version is too old. Skipping cairo
graphical tests.")
77: (16)               construct = functools.partial(tested_scene_construct, *args,
78: (12)                 **kwargs)
79: (12)               test_name_with_param = test_name + "_".join(
80: (12)                 f"_{str(tup[0])}{str(tup[1])}" for tup in kwargs.items()
81: (12)               )
82: (12)               config_tests = _config_test(last_frame)
83: (16)               config_tests["text_dir"] = tmp_path
84: (16)               config_tests["tex_dir"] = tmp_path
85: (12)               if last_frame:
86: (12)                 config_tests["frame_rate"] = 1

```

```

87: (16)             test_file_path =
tested_scene_construct.__globals__["__file__"]
88: (12)         except Exception:
89: (16)             test_file_path = None
90: (12)         real_test = _make_test_comparing_frames(
91: (16)             file_path=_control_data_path(
92: (20)                 test_file_path,
93: (20)                 module_name,
94: (20)                 test_name_with_param,
95: (20)                 setting_test,
96: (16)             ),
97: (16)             base_scene=base_scene,
98: (16)             construct=construct,
99: (16)             renderer_class=renderer_class,
100: (16)            is_set_test_data_test=setting_test,
101: (16)            last_frame=last_frame,
102: (16)            show_diff=request.config.getoption("--show_diff"),
103: (16)            size_frame=(config_tests["pixel_height"],
config_tests["pixel_width"]),
104: (12)        )
105: (12)        with tempconfig({**config_tests, **custom_config}):
106: (16)            real_test()
107: (8)            parameters = list(old_sig.parameters.values())
108: (8)            if "request" not in old_sig.parameters:
109: (12)                parameters += [inspect.Parameter("request",
inspect.Parameter.KEYWORD_ONLY)]
110: (8)                if "tmp_path" not in old_sig.parameters:
111: (12)                    parameters += [
112: (16)                        inspect.Parameter("tmp_path", inspect.Parameter.KEYWORD_ONLY),
113: (12)                    ]
114: (8)                    new_sig = old_sig.replace(parameters=parameters)
115: (8)                    wrapper.__signature__ = new_sig
116: (8)                    setattr(wrapper, "pytestmark", [])
117: (8)                    new_marks = getattr(tested_scene_construct, "pytestmark", [])
118: (8)                    wrapper.pytestmark = new_marks
119: (8)                    return wrapper
120: (4)            if callable(func):
121: (8)                return decorator_maker(func)
122: (4)            return decorator_maker
123: (0)        def _make_test_comparing_frames(
124: (4)            file_path: Path,
125: (4)            base_scene: type[Scene],
126: (4)            construct: Callable[[Scene], None],
127: (4)            renderer_class: type, # Renderer type, there is no superclass renderer
yet .....
128: (4)            is_set_test_data_test: bool,
129: (4)            last_frame: bool,
130: (4)            show_diff: bool,
131: (4)            size_frame: tuple,
132: (0)        ) -> Callable[[], None]:
133: (4)            """Create the real pytest test that will fail if the frames mismatch.
Parameters
-----
file_path
    The path of the control frames.
base_scene
    The base scene class.
construct
    The construct method (= the test function)
renderer_class
    The renderer base class.
show_diff
    whether to visually show_diff (see --show_diff)
Returns
-----
Callable[[], None]
    The pytest test.
"""
134: (4)
135: (4)
136: (4)
137: (8)
138: (4)
139: (8)
140: (4)
141: (8)
142: (4)
143: (8)
144: (4)
145: (8)
146: (4)
147: (4)
148: (4)
149: (8)
150: (4)
151: (4)
if is_set_test_data_test:

```

```

152: (8)             frames_tester = _ControlDataWriter(file_path, size_frame=size_frame)
153: (4)         else:
154: (8)             frames_tester = _FramesTester(file_path, show_diff=show_diff)
155: (4)         file_writer_class = (
156: (8)             _make_scene_file_writer_class(frames_tester)
157: (8)             if not last_frame
158: (8)             else DummySceneFileWriter
159: (4)         )
160: (4)     testRenderer = _make_test_renderer_class(renderer_class)
161: (4)     def real_test():
162: (8)         with frames_tester.testing():
163: (12)             sceneTested = _make_test_scene_class(
164: (16)                 base_scene=base_scene,
165: (16)                 construct_test=construct,
166: (16)                 test_renderer=(
167: (20)                     testRenderer(file_writer_class=file_writer_class)
168: (20)                     if base_scene is not ThreeDScene
169: (20)                     else testRenderer(
170: (24)                         file_writer_class=file_writer_class,
171: (24)                         camera_class=ThreeDCamera,
172: (20)                     )
173: (16)                 ), # testRenderer(file_writer_class=file_writer_class),
174: (12)             )
175: (12)             scene_tested = sceneTested(skip_animations=True)
176: (12)             scene_tested.render()
177: (12)             if last_frame:
178: (16)                 frames_tester.check_frame(-1,
scene_tested.renderer.get_frame())
179: (4)             return real_test
180: (0)         def _control_data_path(
181: (4)             test_file_path: str | None, module_name: str, test_name: str,
setting_test: bool
182: (0)         ) -> Path:
183: (4)             if test_file_path is None:
184: (8)                 test_file_path = __file__
185: (4)             path = Path(test_file_path).absolute().parent / "control_data" /
module_name
186: (4)
187: (8)
188: (4)
189: (8)
{path}")
190: (4)
191: (4)
192: (8)
193: (12)
{path.parent}. "
194: (12)
195: (8)
196: (4)
197: (0)         return path
198: (4)
199: (8)
200: (12)
201: (12)
202: (16)
203: (16)
204: (16)
205: (12)
206: (8)
207: (4)
)
-----
```

## File 163 - \_test\_class\_makers.py:

```

1: (0)             from __future__ import annotations
2: (0)             from typing import Callable
3: (0)             from manim.scene.scene import Scene
```

```

4: (0)          from manim.scene.scene_file_writer import SceneFileWriter
5: (0)          from ._frames_testers import _FramesTester
6: (0)          def _make_test_scene_class(
7: (4)            base_scene: type[Scene],
8: (4)            construct_test: Callable[[Scene], None],
9: (4)            test_renderer,
10: (0)           ) -> type[Scene]:
11: (4)             class _TestedScene(base_scene):
12: (8)               def __init__(self, *args, **kwargs):
13: (12)                 super().__init__(renderer=test_renderer, *args, **kwargs)
14: (8)               def construct(self):
15: (12)                 construct_test(self)
16: (12)                 if self.animations is not None:
17: (16)                   self.update_to_time(self.get_run_time(self.animations))
18: (16)                   self.renderer.render(self, 1, self.moving_mobjects)
19: (4)             return _TestedScene
20: (0)          def _make_test_renderer_class(from_renderer):
21: (4)            class _TestRenderer(from_renderer):
22: (8)              pass
23: (4)            return _TestRenderer
24: (0)          class DummySceneFileWriter(SceneFileWriter):
25: (4)            """Delegate of SceneFileWriter used to test the frames."""
26: (4)            def __init__(self, renderer, scene_name, **kwargs):
27: (8)              super().__init__(renderer, scene_name, **kwargs)
28: (8)              self.i = 0
29: (4)            def init_output_directories(self, scene_name):
30: (8)              pass
31: (4)            def add_partial_movie_file(self, hash_animation):
32: (8)              pass
33: (4)            def begin_animation(self, allow_write=True):
34: (8)              pass
35: (4)            def end_animation(self, allow_write):
36: (8)              pass
37: (4)            def combine_to_movie(self):
38: (8)              pass
39: (4)            def combine_to_section_videos(self):
40: (8)              pass
41: (4)            def clean_cache(self):
42: (8)              pass
43: (4)            def write_frame(self, frame_or_renderer):
44: (8)              self.i += 1
45: (0)          def _make_scene_file_writer_class(tester: _FramesTester) ->
46: (0)            type[SceneFileWriter]:
47: (4)              class TestSceneFileWriter(DummySceneFileWriter):
48: (8)                def write_frame(self, frame_or_renderer):
49: (12)                  tester.check_frame(self.i, frame_or_renderer)
50: (12)                  super().write_frame(frame_or_renderer)
50: (4)                return TestSceneFileWriter

```

## -----

File 164 -  
 SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRYCOMBINER\_aligner\_20\_characters\_for\_pythons\_codes.py:

```

1: (0)          import os
2: (0)          from datetime import datetime
3: (0)          def get_file_info(root_folder):
4: (4)            file_info_list = []
5: (4)            for root, dirs, files in os.walk(root_folder):
6: (8)              for file in files:
7: (12)                try:
8: (16)                  if file.endswith('.py'):
9: (20)                    file_path = os.path.join(root, file)
10: (20)                    creation_time =
datetime.fromtimestamp(os.path.getctime(file_path))
11: (20)                    modified_time =
datetime.fromtimestamp(os.path.getmtime(file_path))
12: (20)                    file_extension = os.path.splitext(file)[1].lower()

```

```

13: (20)                     file_info_list.append([file, file_path, creation_time,
modified_time, file_extension, root])
14: (12)                 except Exception as e:
15: (16)                     print(f"Error processing file {file}: {e}")
16: (4)             file_info_list.sort(key=lambda x: (x[2], x[3], len(x[0]), x[4])) # Sort
by creation, modification time, name length, extension
17: (4)             return file_info_list
18: (0)         def process_file(file_info_list):
19: (4)             combined_output = []
20: (4)             for idx, (file_name, file_path, creation_time, modified_time,
file_extension, root) in enumerate(file_info_list):
21: (8)                 with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
22: (12)                     content = f.read()
23: (12)                     content = "\n".join([line for line in content.split('\n') if
line.strip() and not line.strip().startswith("#")])
24: (12)                     content = content.replace('\t', '    ')
25: (12)                     processed_lines = []
26: (12)                     for i, line in enumerate(content.split('\n')):
27: (16)                         leading_spaces = len(line) - len(line.lstrip(' '))
28: (16)                         line_number_str = f"{i+1}: ({leading_spaces})"
29: (16)                         padding = ' ' * (20 - len(line_number_str))
30: (16)                         processed_line = f"{line_number_str}{padding}{line}"
31: (16)                         processed_lines.append(processed_line)
32: (12)                     content_with_line_numbers = "\n".join(processed_lines)
33: (12)                     combined_output.append(f"File {idx + 1} - {file_name}:\n")
34: (12)                     combined_output.append(content_with_line_numbers)
35: (12)                     combined_output.append("\n" + "-"*40 + "\n")
36: (4)             return combined_output
37: (0)         root_folder_path = '.' # Set this to the desired folder
38: (0)         file_info_list = get_file_info(root_folder_path)
39: (0)         combined_output = process_file(file_info_list)
40: (0)         output_file =
'SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRY_combined_python_files_20_chars.txt'
41: (0)         with open(output_file, 'w', encoding='utf-8') as logfile:
42: (4)             logfile.write("\n".join(combined_output))
43: (0)         print(f"Processed file info logged to {output_file}")

```

---