```
SNGT_QHENOMENOLOGY_File 1 - conf.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             import os
SNGT_QHENOMENOLOGY_2: (0)             import sys
SNGT_QHENOMENOLOGY_3: (0)             sys.path.insert(0, os.path.abspath("."))
SNGT_QHENOMENOLOGY_4: (0)             sys.path.insert(0, os.path.abspath('../../'))
SNGT_QHENOMENOLOGY_5: (0)             project = 'manim'
SNGT_QHENOMENOLOGY_6: (0)             copyright = '- This document has been placed in the public
domain.'
SNGT_QHENOMENOLOGY_7: (0)             author = 'TonyCrane'
SNGT_QHENOMENOLOGY_8: (0)             release = ''
SNGT_QHENOMENOLOGY_9: (0)             extensions = [
SNGT_QHENOMENOLOGY_10: (4)                'sphinx.ext.todo',
SNGT_QHENOMENOLOGY_11: (4)                'sphinx.ext.githubpages',
SNGT_QHENOMENOLOGY_12: (4)                'sphinx.ext.mathjax',
SNGT_QHENOMENOLOGY_13: (4)                'sphinx.ext.intersphinx',
SNGT_QHENOMENOLOGY_14: (4)                'sphinx.ext.autodoc',
SNGT_QHENOMENOLOGY_15: (4)                'sphinx.ext.coverage',
SNGT_QHENOMENOLOGY_16: (4)                'sphinx.ext.napoleon',
SNGT_QHENOMENOLOGY_17: (4)                'sphinx_copybutton',
SNGT_QHENOMENOLOGY_18: (4)                'manim_example_ext'
SNGT_QHENOMENOLOGY_19: (0)            ]
SNGT_QHENOMENOLOGY_20: (0)            autoclass_content = 'both'
SNGT_QHENOMENOLOGY_21: (0)            mathjax_path =
"https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js"
SNGT_QHENOMENOLOGY_22: (0)            templates_path = ['_templates']
SNGT_QHENOMENOLOGY_23: (0)            source_suffix = '.rst'
SNGT_QHENOMENOLOGY_24: (0)            master_doc = 'index'
SNGT_QHENOMENOLOGY_25: (0)            pygments_style = 'default'
SNGT_QHENOMENOLOGY_26: (0)            html_static_path = ["_static"]
SNGT_QHENOMENOLOGY_27: (0)            html_css_files = [
SNGT_QHENOMENOLOGY_28: (4)                "https://cdn.jsdelivr.net/gh/manim-
kindergarten/CDN@master/manimgl_assets/custom.css",
SNGT_QHENOMENOLOGY_29: (4)                "https://cdn.jsdelivr.net/gh/manim-
kindergarten/CDN@master/manimgl_assets/colors.css"
SNGT_QHENOMENOLOGY_30: (0)            ]
SNGT_QHENOMENOLOGY_31: (0)            html_theme = 'furo'  # pip install furo==2020.10.5b9
SNGT_QHENOMENOLOGY_32: (0)            html_favicon = '_static/icon.png'
SNGT_QHENOMENOLOGY_33: (0)            html_logo = '../../logo/transparent_graph.png'
SNGT_QHENOMENOLOGY_34: (0)            html_theme_options = {
SNGT_QHENOMENOLOGY_35: (4)                "sidebar_hide_name": True,
SNGT_QHENOMENOLOGY_36: (0)            }
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 2 - example.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from manimlib import *
SNGT_QHENOMENOLOGY_2: (0)             class SquareToCircle(Scene):
SNGT_QHENOMENOLOGY_3: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_4: (8)                     circle = Circle()
SNGT_QHENOMENOLOGY_5: (8)                     circle.set_fill(BLUE, opacity=0.5)
SNGT_QHENOMENOLOGY_6: (8)                     circle.set_stroke(BLUE_E, width=4)
SNGT_QHENOMENOLOGY_7: (8)                     square = Square()
SNGT_QHENOMENOLOGY_8: (8)                     self.play(ShowCreation(square))
SNGT_QHENOMENOLOGY_9: (8)                     self.wait()
SNGT_QHENOMENOLOGY_10: (8)                    self.play(ReplacementTransform(square, circle))
SNGT_QHENOMENOLOGY_11: (8)                    self.wait()
SNGT_QHENOMENOLOGY_12: (0)            class SquareToCircleEmbed(Scene):
SNGT_QHENOMENOLOGY_13: (4)                def construct(self):
SNGT_QHENOMENOLOGY_14: (8)                    circle = Circle()
SNGT_QHENOMENOLOGY_15: (8)                    circle.set_fill(BLUE, opacity=0.5)
SNGT_QHENOMENOLOGY_16: (8)                    circle.set_stroke(BLUE_E, width=4)
SNGT_QHENOMENOLOGY_17: (8)                    self.add(circle)
SNGT_QHENOMENOLOGY_18: (8)                    self.wait()
SNGT_QHENOMENOLOGY_19: (8)                    self.play(circle.animate.stretch(4, dim=0))
SNGT_QHENOMENOLOGY_20: (8)                    self.wait(1.5)
SNGT_QHENOMENOLOGY_21: (8)                    self.play(Rotate(circle, TAU / 4))
SNGT_QHENOMENOLOGY_22: (8)                    self.wait(1.5)
```

```
SNGT_QHENOMENOLOGY_23: (8)                    self.play(circle.animate.shift(2 * RIGHT),
circle.animate.scale(0.25))
SNGT_QHENOMENOLOGY_24: (8)                    self.wait(1.5)
SNGT_QHENOMENOLOGY_25: (8)                    circle.insert_n_curves(10)
SNGT_QHENOMENOLOGY_26: (8)
self.play(circle.animate.apply_complex_function(lambda z: z**2))
SNGT_QHENOMENOLOGY_27: (8)                    self.wait(2)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 3 - logo.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from manimlib.imports import *
SNGT_QHENOMENOLOGY_2: (0)             NEW_BLUE = "#68a8e1"
SNGT_QHENOMENOLOGY_3: (0)             class Thumbnail(GraphScene):
SNGT_QHENOMENOLOGY_4: (4)                 CONFIG = {
SNGT_QHENOMENOLOGY_5: (8)                     "y_max": 8,
SNGT_QHENOMENOLOGY_6: (8)                     "y_axis_height": 5,
SNGT_QHENOMENOLOGY_7: (4)                 }
SNGT_QHENOMENOLOGY_8: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_9: (8)                     self.show_function_graph()
SNGT_QHENOMENOLOGY_10: (4)                def show_function_graph(self):
SNGT_QHENOMENOLOGY_11: (8)                    self.setup_axes(animate=False)
SNGT_QHENOMENOLOGY_12: (8)                    def func(x):
SNGT_QHENOMENOLOGY_13: (12)                       return 0.1 * (x + 3-5) * (x - 3-5) * (x-5) + 5
SNGT_QHENOMENOLOGY_14: (8)                    def rect(x):
SNGT_QHENOMENOLOGY_15: (12)                       return 2.775*(x-1.5)+3.862
SNGT_QHENOMENOLOGY_16: (8)                    recta = self.get_graph(rect,x_min=-1,x_max=5)
SNGT_QHENOMENOLOGY_17: (8)                    graph = self.get_graph(func,x_min=0.2,x_max=9)
SNGT_QHENOMENOLOGY_18: (8)                    graph.set_color(NEW_BLUE)
SNGT_QHENOMENOLOGY_19: (8)                    input_tracker_p1 = ValueTracker(1.5)
SNGT_QHENOMENOLOGY_20: (8)                    input_tracker_p2 = ValueTracker(3.5)
SNGT_QHENOMENOLOGY_21: (8)                    def get_x_value(input_tracker):
SNGT_QHENOMENOLOGY_22: (12)                       return input_tracker.get_value()
SNGT_QHENOMENOLOGY_23: (8)                    def get_y_value(input_tracker):
SNGT_QHENOMENOLOGY_24: (12)                       return
graph.underlying_function(get_x_value(input_tracker))
SNGT_QHENOMENOLOGY_25: (8)                    def get_x_point(input_tracker):
SNGT_QHENOMENOLOGY_26: (12)                       return
self.coords_to_point(get_x_value(input_tracker), 0)
SNGT_QHENOMENOLOGY_27: (8)                    def get_y_point(input_tracker):
SNGT_QHENOMENOLOGY_28: (12)                       return self.coords_to_point(0,
get_y_value(input_tracker))
SNGT_QHENOMENOLOGY_29: (8)                    def get_graph_point(input_tracker):
SNGT_QHENOMENOLOGY_30: (12)                       return
self.coords_to_point(get_x_value(input_tracker), get_y_value(input_tracker))
SNGT_QHENOMENOLOGY_31: (8)                    def get_v_line(input_tracker):
SNGT_QHENOMENOLOGY_32: (12)                       return DashedLine(get_x_point(input_tracker),
get_graph_point(input_tracker), stroke_width=2)
SNGT_QHENOMENOLOGY_33: (8)                    def get_h_line(input_tracker):
SNGT_QHENOMENOLOGY_34: (12)                       return
DashedLine(get_graph_point(input_tracker), get_y_point(input_tracker), stroke_width=2)
SNGT_QHENOMENOLOGY_35: (8)                    input_triangle_p1 = RegularPolygon(n=3,
start_angle=TAU / 4)
SNGT_QHENOMENOLOGY_36: (8)                    output_triangle_p1 = RegularPolygon(n=3,
start_angle=0)
SNGT_QHENOMENOLOGY_37: (8)                    for triangle in input_triangle_p1,
output_triangle_p1:
SNGT_QHENOMENOLOGY_38: (12)                       triangle.set_fill(WHITE, 1)
SNGT_QHENOMENOLOGY_39: (12)                       triangle.set_stroke(width=0)
SNGT_QHENOMENOLOGY_40: (12)                       triangle.scale(0.1)
SNGT_QHENOMENOLOGY_41: (8)                    input_triangle_p2 = RegularPolygon(n=3,
start_angle=TAU / 4)
SNGT_QHENOMENOLOGY_42: (8)                    output_triangle_p2 = RegularPolygon(n=3,
start_angle=0)
SNGT_QHENOMENOLOGY_43: (8)                    for triangle in input_triangle_p2,
output_triangle_p2:
SNGT_QHENOMENOLOGY_44: (12)                       triangle.set_fill(WHITE, 1)
SNGT_QHENOMENOLOGY_45: (12)                       triangle.set_stroke(width=0)
```

```
SNGT_QHENOMENOLOGY_46: (12)                         triangle.scale(0.1)
SNGT_QHENOMENOLOGY_47: (8)                      x_label_p1 = Tex("a")
SNGT_QHENOMENOLOGY_48: (8)                      output_label_p1 = Tex("f(a)")
SNGT_QHENOMENOLOGY_49: (8)                      x_label_p2 = Tex("b")
SNGT_QHENOMENOLOGY_50: (8)                      output_label_p2 = Tex("f(b)")
SNGT_QHENOMENOLOGY_51: (8)                      v_line_p1 = get_v_line(input_tracker_p1)
SNGT_QHENOMENOLOGY_52: (8)                      v_line_p2 = get_v_line(input_tracker_p2)
SNGT_QHENOMENOLOGY_53: (8)                      h_line_p1 = get_h_line(input_tracker_p1)
SNGT_QHENOMENOLOGY_54: (8)                      h_line_p2 = get_h_line(input_tracker_p2)
SNGT_QHENOMENOLOGY_55: (8)                      graph_dot_p1 = Dot(color=WHITE)
SNGT_QHENOMENOLOGY_56: (8)                      graph_dot_p2 = Dot(color=WHITE)
SNGT_QHENOMENOLOGY_57: (8)                      x_label_p1.next_to(v_line_p1, DOWN)
SNGT_QHENOMENOLOGY_58: (8)                      x_label_p2.next_to(v_line_p2, DOWN)
SNGT_QHENOMENOLOGY_59: (8)                      output_label_p1.next_to(h_line_p1, LEFT)
SNGT_QHENOMENOLOGY_60: (8)                      output_label_p2.next_to(h_line_p2, LEFT)
SNGT_QHENOMENOLOGY_61: (8)                      input_triangle_p1.next_to(v_line_p1, DOWN, buff=0)
SNGT_QHENOMENOLOGY_62: (8)                      input_triangle_p2.next_to(v_line_p2, DOWN, buff=0)
SNGT_QHENOMENOLOGY_63: (8)                      output_triangle_p1.next_to(h_line_p1, LEFT, buff=0)
SNGT_QHENOMENOLOGY_64: (8)                      output_triangle_p2.next_to(h_line_p2, LEFT, buff=0)
SNGT_QHENOMENOLOGY_65: (8)
graph_dot_p1.move_to(get_graph_point(input_tracker_p1))
SNGT_QHENOMENOLOGY_66: (8)
graph_dot_p2.move_to(get_graph_point(input_tracker_p2))
SNGT_QHENOMENOLOGY_67: (8)                      self.play(
SNGT_QHENOMENOLOGY_68: (12)                         ShowCreation(graph),
SNGT_QHENOMENOLOGY_69: (8)                      )
SNGT_QHENOMENOLOGY_70: (8)                      self.add_foreground_mobject(graph_dot_p1)
SNGT_QHENOMENOLOGY_71: (8)                      self.add_foreground_mobject(graph_dot_p2)
SNGT_QHENOMENOLOGY_72: (8)                      self.play(
SNGT_QHENOMENOLOGY_73: (12)                         DrawBorderThenFill(input_triangle_p1),
SNGT_QHENOMENOLOGY_74: (12)                         Write(x_label_p1),
SNGT_QHENOMENOLOGY_75: (12)                         ShowCreation(v_line_p1),
SNGT_QHENOMENOLOGY_76: (12)                         GrowFromCenter(graph_dot_p1),
SNGT_QHENOMENOLOGY_77: (12)                         ShowCreation(h_line_p1),
SNGT_QHENOMENOLOGY_78: (12)                         Write(output_label_p1),
SNGT_QHENOMENOLOGY_79: (12)                         DrawBorderThenFill(output_triangle_p1),
SNGT_QHENOMENOLOGY_80: (12)                         DrawBorderThenFill(input_triangle_p2),
SNGT_QHENOMENOLOGY_81: (12)                         Write(x_label_p2),
SNGT_QHENOMENOLOGY_82: (12)                         ShowCreation(v_line_p2),
SNGT_QHENOMENOLOGY_83: (12)                         GrowFromCenter(graph_dot_p2),
SNGT_QHENOMENOLOGY_84: (12)                         ShowCreation(h_line_p2),
SNGT_QHENOMENOLOGY_85: (12)                         Write(output_label_p2),
SNGT_QHENOMENOLOGY_86: (12)                         DrawBorderThenFill(output_triangle_p2),
SNGT_QHENOMENOLOGY_87: (12)                         run_time=0.5
SNGT_QHENOMENOLOGY_88: (8)                      )
SNGT_QHENOMENOLOGY_89: (8)                      self.add(
SNGT_QHENOMENOLOGY_90: (12)                         input_triangle_p2,
SNGT_QHENOMENOLOGY_91: (12)                         x_label_p2,
SNGT_QHENOMENOLOGY_92: (12)                         graph_dot_p2,
SNGT_QHENOMENOLOGY_93: (12)                         v_line_p2,
SNGT_QHENOMENOLOGY_94: (12)                         h_line_p2,
SNGT_QHENOMENOLOGY_95: (12)                         output_triangle_p2,
SNGT_QHENOMENOLOGY_96: (12)                         output_label_p2,
SNGT_QHENOMENOLOGY_97: (8)                      )
SNGT_QHENOMENOLOGY_98: (8)                      pendiente_recta = self.get_secant_slope_group(
SNGT_QHENOMENOLOGY_99: (12)                         1.9, recta, dx = 1.4,
SNGT_QHENOMENOLOGY_100: (12)                        df_label = None,
SNGT_QHENOMENOLOGY_101: (12)                        dx_label = None,
SNGT_QHENOMENOLOGY_102: (12)                        dx_line_color = PURPLE,
SNGT_QHENOMENOLOGY_103: (12)                        df_line_color= ORANGE,
SNGT_QHENOMENOLOGY_104: (12)                        )
SNGT_QHENOMENOLOGY_105: (8)                     grupo_secante = self.get_secant_slope_group(
SNGT_QHENOMENOLOGY_106: (12)                        1.5, graph, dx = 2,
SNGT_QHENOMENOLOGY_107: (12)                        df_label = None,
SNGT_QHENOMENOLOGY_108: (12)                        dx_label = None,
SNGT_QHENOMENOLOGY_109: (12)                        dx_line_color = "#942357",
SNGT_QHENOMENOLOGY_110: (12)                        df_line_color= "#3f7d5c",
SNGT_QHENOMENOLOGY_111: (12)                        secant_line_color = RED,
SNGT_QHENOMENOLOGY_112: (8)                     )
```

```
SNGT_QHENOMENOLOGY_113: (8)                     self.add(
SNGT_QHENOMENOLOGY_114: (12)                        input_triangle_p2,
SNGT_QHENOMENOLOGY_115: (12)                        graph_dot_p2,
SNGT_QHENOMENOLOGY_116: (12)                        v_line_p2,
SNGT_QHENOMENOLOGY_117: (12)                        h_line_p2,
SNGT_QHENOMENOLOGY_118: (12)                        output_triangle_p2,
SNGT_QHENOMENOLOGY_119: (8)                     )
SNGT_QHENOMENOLOGY_120: (8)                     self.play(FadeIn(grupo_secante))
SNGT_QHENOMENOLOGY_121: (8)                     kwargs = {
SNGT_QHENOMENOLOGY_122: (12)                        "x_min" : 4,
SNGT_QHENOMENOLOGY_123: (12)                        "x_max" : 9,
SNGT_QHENOMENOLOGY_124: (12)                        "fill_opacity" : 0.75,
SNGT_QHENOMENOLOGY_125: (12)                        "stroke_width" : 0.25,
SNGT_QHENOMENOLOGY_126: (8)                     }
SNGT_QHENOMENOLOGY_127: (8)                     self.graph=graph
SNGT_QHENOMENOLOGY_128: (8)                     iteraciones=6
SNGT_QHENOMENOLOGY_129: (8)                     self.rect_list = self.get_riemann_rectangles_list(
SNGT_QHENOMENOLOGY_130: (12)                        graph,
iteraciones,start_color=PURPLE,end_color=ORANGE, **kwargs
SNGT_QHENOMENOLOGY_131: (8)                     )
SNGT_QHENOMENOLOGY_132: (8)                     flat_rects = self.get_riemann_rectangles(
SNGT_QHENOMENOLOGY_133: (12)                        self.get_graph(lambda x : 0), dx =
0.5,start_color=invert_color(PURPLE),end_color=invert_color(ORANGE),**kwargs
SNGT_QHENOMENOLOGY_134: (8)                     )
SNGT_QHENOMENOLOGY_135: (8)                     rects = self.rect_list[0]
SNGT_QHENOMENOLOGY_136: (8)                     self.transform_between_riemann_rects(
SNGT_QHENOMENOLOGY_137: (12)                        flat_rects, rects,
SNGT_QHENOMENOLOGY_138: (12)                        replace_mobject_with_target_in_scene = True,
SNGT_QHENOMENOLOGY_139: (12)                        run_time=0.9
SNGT_QHENOMENOLOGY_140: (8)                     )
SNGT_QHENOMENOLOGY_141: (8)                     picture = Group(*self.mobjects)
SNGT_QHENOMENOLOGY_142: (8)                     picture.scale(0.6).to_edge(LEFT, buff=SMALL_BUFF)
SNGT_QHENOMENOLOGY_143: (8)                     manim = TexText("Manim").set_height(1.5) \
SNGT_QHENOMENOLOGY_144: (36)                                            .next_to(picture,
RIGHT) \
SNGT_QHENOMENOLOGY_145: (36)                                            .shift(DOWN * 0.7)
SNGT_QHENOMENOLOGY_146: (8)                     self.add(manim)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 4 - fading.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.animation.transform import Transform
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import ORIGIN
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.utils.rate_functions import there_and_back
SNGT_QHENOMENOLOGY_9: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                  from typing import Callable
SNGT_QHENOMENOLOGY_12: (4)                  from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_13: (4)                  from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_14: (4)                  from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_15: (0)              class Fade(Transform):
SNGT_QHENOMENOLOGY_16: (4)                  def __init__(
SNGT_QHENOMENOLOGY_17: (8)                      self,
SNGT_QHENOMENOLOGY_18: (8)                      mobject: Mobject,
SNGT_QHENOMENOLOGY_19: (8)                      shift: np.ndarray = ORIGIN,
SNGT_QHENOMENOLOGY_20: (8)                      scale: float = 1,
SNGT_QHENOMENOLOGY_21: (8)                      **kwargs
SNGT_QHENOMENOLOGY_22: (4)                  ):
SNGT_QHENOMENOLOGY_23: (8)                      self.shift_vect = shift
SNGT_QHENOMENOLOGY_24: (8)                      self.scale_factor = scale
SNGT_QHENOMENOLOGY_25: (8)                      super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_26: (0)              class FadeIn(Fade):
```

```
SNGT_QHENOMENOLOGY_27: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_28: (8)                     return self.mobject.copy()
SNGT_QHENOMENOLOGY_29: (4)                 def create_starting_mobject(self) -> Mobject:
SNGT_QHENOMENOLOGY_30: (8)                     start = super().create_starting_mobject()
SNGT_QHENOMENOLOGY_31: (8)                     start.set_opacity(0)
SNGT_QHENOMENOLOGY_32: (8)                     start.scale(1.0 / self.scale_factor)
SNGT_QHENOMENOLOGY_33: (8)                     start.shift(-self.shift_vect)
SNGT_QHENOMENOLOGY_34: (8)                     return start
SNGT_QHENOMENOLOGY_35: (0)             class FadeOut(Fade):
SNGT_QHENOMENOLOGY_36: (4)                 def __init__(
SNGT_QHENOMENOLOGY_37: (8)                     self,
SNGT_QHENOMENOLOGY_38: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_39: (8)                     shift: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_40: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_41: (8)                     final_alpha_value: float = 0.0,  # Put it back in
original state when done,
SNGT_QHENOMENOLOGY_42: (8)                     **kwargs
SNGT_QHENOMENOLOGY_43: (4)                 ):
SNGT_QHENOMENOLOGY_44: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_45: (12)                        mobject, shift,
SNGT_QHENOMENOLOGY_46: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_47: (12)                        final_alpha_value=final_alpha_value,
SNGT_QHENOMENOLOGY_48: (12)                        **kwargs
SNGT_QHENOMENOLOGY_49: (8)                     )
SNGT_QHENOMENOLOGY_50: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_51: (8)                     result = self.mobject.copy()
SNGT_QHENOMENOLOGY_52: (8)                     result.set_opacity(0)
SNGT_QHENOMENOLOGY_53: (8)                     result.shift(self.shift_vect)
SNGT_QHENOMENOLOGY_54: (8)                     result.scale(self.scale_factor)
SNGT_QHENOMENOLOGY_55: (8)                     return result
SNGT_QHENOMENOLOGY_56: (0)             class FadeInFromPoint(FadeIn):
SNGT_QHENOMENOLOGY_57: (4)                 def __init__(self, mobject: Mobject, point: Vect3,
**kwargs):
SNGT_QHENOMENOLOGY_58: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_59: (12)                        mobject,
SNGT_QHENOMENOLOGY_60: (12)                        shift=mobject.get_center() - point,
SNGT_QHENOMENOLOGY_61: (12)                        scale=np.inf,
SNGT_QHENOMENOLOGY_62: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_63: (8)                     )
SNGT_QHENOMENOLOGY_64: (0)             class FadeOutToPoint(FadeOut):
SNGT_QHENOMENOLOGY_65: (4)                 def __init__(self, mobject: Mobject, point: Vect3,
**kwargs):
SNGT_QHENOMENOLOGY_66: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_67: (12)                        mobject,
SNGT_QHENOMENOLOGY_68: (12)                        shift=point - mobject.get_center(),
SNGT_QHENOMENOLOGY_69: (12)                        scale=0,
SNGT_QHENOMENOLOGY_70: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_71: (8)                     )
SNGT_QHENOMENOLOGY_72: (0)             class FadeTransform(Transform):
SNGT_QHENOMENOLOGY_73: (4)                 def __init__(
SNGT_QHENOMENOLOGY_74: (8)                     self,
SNGT_QHENOMENOLOGY_75: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_76: (8)                     target_mobject: Mobject,
SNGT_QHENOMENOLOGY_77: (8)                     stretch: bool = True,
SNGT_QHENOMENOLOGY_78: (8)                     dim_to_match: int = 1,
SNGT_QHENOMENOLOGY_79: (8)                     **kwargs
SNGT_QHENOMENOLOGY_80: (4)                 ):
SNGT_QHENOMENOLOGY_81: (8)                     self.to_add_on_completion = target_mobject
SNGT_QHENOMENOLOGY_82: (8)                     self.stretch = stretch
SNGT_QHENOMENOLOGY_83: (8)                     self.dim_to_match = dim_to_match
SNGT_QHENOMENOLOGY_84: (8)                     mobject.save_state()
SNGT_QHENOMENOLOGY_85: (8)                     super().__init__(mobject.get_group_class()(mobject,
target_mobject.copy()), **kwargs)
SNGT_QHENOMENOLOGY_86: (4)                 def begin(self) -> None:
SNGT_QHENOMENOLOGY_87: (8)                     self.ending_mobject = self.mobject.copy()
SNGT_QHENOMENOLOGY_88: (8)                     Animation.begin(self)
SNGT_QHENOMENOLOGY_89: (8)                     start, end = self.starting_mobject,
self.ending_mobject
SNGT_QHENOMENOLOGY_90: (8)                     for m0, m1 in ((start[1], start[0]), (end[0],
```

```
end[1])):
SNGT_QHENOMENOLOGY_91: (12)                         self.ghost_to(m0, m1)
SNGT_QHENOMENOLOGY_92: (4)                 def ghost_to(self, source: Mobject, target: Mobject) ->
None:
SNGT_QHENOMENOLOGY_93: (8)                     source.replace(target, stretch=self.stretch,
dim_to_match=self.dim_to_match)
SNGT_QHENOMENOLOGY_94: (8)                     source.set_uniform(**target.get_uniforms())
SNGT_QHENOMENOLOGY_95: (8)                     source.set_opacity(0)
SNGT_QHENOMENOLOGY_96: (4)                 def get_all_mobjects(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_97: (8)                     return [
SNGT_QHENOMENOLOGY_98: (12)                         self.mobject,
SNGT_QHENOMENOLOGY_99: (12)                         self.starting_mobject,
SNGT_QHENOMENOLOGY_100: (12)                        self.ending_mobject,
SNGT_QHENOMENOLOGY_101: (8)                    ]
SNGT_QHENOMENOLOGY_102: (4)                def get_all_families_zipped(self) ->
zip[tuple[Mobject]]:
SNGT_QHENOMENOLOGY_103: (8)                    return Animation.get_all_families_zipped(self)
SNGT_QHENOMENOLOGY_104: (4)                def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_105: (8)                    Animation.clean_up_from_scene(self, scene)
SNGT_QHENOMENOLOGY_106: (8)                    scene.remove(self.mobject)
SNGT_QHENOMENOLOGY_107: (8)                    self.mobject[0].restore()
SNGT_QHENOMENOLOGY_108: (8)                    if not self.remover:
SNGT_QHENOMENOLOGY_109: (12)                       scene.add(self.to_add_on_completion)
SNGT_QHENOMENOLOGY_110: (0)            class FadeTransformPieces(FadeTransform):
SNGT_QHENOMENOLOGY_111: (4)                def begin(self) -> None:
SNGT_QHENOMENOLOGY_112: (8)                    self.mobject[0].align_family(self.mobject[1])
SNGT_QHENOMENOLOGY_113: (8)                    super().begin()
SNGT_QHENOMENOLOGY_114: (4)                def ghost_to(self, source: Mobject, target: Mobject) ->
None:
SNGT_QHENOMENOLOGY_115: (8)                    for sm0, sm1 in zip(source.get_family(),
target.get_family()):
SNGT_QHENOMENOLOGY_116: (12)                       super().ghost_to(sm0, sm1)
SNGT_QHENOMENOLOGY_117: (0)            class VFadeIn(Animation):
SNGT_QHENOMENOLOGY_118: (4)                """
SNGT_QHENOMENOLOGY_119: (4)                VFadeIn and VFadeOut only work for VMobjects,
SNGT_QHENOMENOLOGY_120: (4)                """
SNGT_QHENOMENOLOGY_121: (4)                def __init__(self, vmobject: VMobject,
suspend_mobject_updating: bool = False, **kwargs):
SNGT_QHENOMENOLOGY_122: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_123: (12)                       vmobject,
SNGT_QHENOMENOLOGY_124: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_125: (12)                       **kwargs
SNGT_QHENOMENOLOGY_126: (8)                    )
SNGT_QHENOMENOLOGY_127: (4)                def interpolate_submobject(
SNGT_QHENOMENOLOGY_128: (8)                    self,
SNGT_QHENOMENOLOGY_129: (8)                    submob: VMobject,
SNGT_QHENOMENOLOGY_130: (8)                    start: VMobject,
SNGT_QHENOMENOLOGY_131: (8)                    alpha: float
SNGT_QHENOMENOLOGY_132: (4)                ) -> None:
SNGT_QHENOMENOLOGY_133: (8)                    submob.set_stroke(
SNGT_QHENOMENOLOGY_134: (12)                       opacity=interpolate(0,
start.get_stroke_opacity(), alpha)
SNGT_QHENOMENOLOGY_135: (8)                    )
SNGT_QHENOMENOLOGY_136: (8)                    submob.set_fill(
SNGT_QHENOMENOLOGY_137: (12)                       opacity=interpolate(0,
start.get_fill_opacity(), alpha)
SNGT_QHENOMENOLOGY_138: (8)                    )
SNGT_QHENOMENOLOGY_139: (0)            class VFadeOut(VFadeIn):
SNGT_QHENOMENOLOGY_140: (4)                def __init__(
SNGT_QHENOMENOLOGY_141: (8)                    self,
SNGT_QHENOMENOLOGY_142: (8)                    vmobject: VMobject,
SNGT_QHENOMENOLOGY_143: (8)                    remover: bool = True,
SNGT_QHENOMENOLOGY_144: (8)                    final_alpha_value: float = 0.0,
SNGT_QHENOMENOLOGY_145: (8)                    **kwargs
SNGT_QHENOMENOLOGY_146: (4)                ):
SNGT_QHENOMENOLOGY_147: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_148: (12)                       vmobject,
SNGT_QHENOMENOLOGY_149: (12)                       remover=remover,
```

```
SNGT_QHENOMENOLOGY_150: (12)                         final_alpha_value=final_alpha_value,
SNGT_QHENOMENOLOGY_151: (12)                         **kwargs
SNGT_QHENOMENOLOGY_152: (8)                     )
SNGT_QHENOMENOLOGY_153: (4)                 def interpolate_submobject(
SNGT_QHENOMENOLOGY_154: (8)                     self,
SNGT_QHENOMENOLOGY_155: (8)                     submob: VMobject,
SNGT_QHENOMENOLOGY_156: (8)                     start: VMobject,
SNGT_QHENOMENOLOGY_157: (8)                     alpha: float
SNGT_QHENOMENOLOGY_158: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_159: (8)                     super().interpolate_submobject(submob, start, 1 -
alpha)
SNGT_QHENOMENOLOGY_160: (0)             class VFadeInThenOut(VFadeIn):
SNGT_QHENOMENOLOGY_161: (4)                 def __init__(
SNGT_QHENOMENOLOGY_162: (8)                     self,
SNGT_QHENOMENOLOGY_163: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_164: (8)                     rate_func: Callable[[float], float] =
there_and_back,
SNGT_QHENOMENOLOGY_165: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_166: (8)                     final_alpha_value: float = 0.5,
SNGT_QHENOMENOLOGY_167: (8)                     **kwargs
SNGT_QHENOMENOLOGY_168: (4)                 ):
SNGT_QHENOMENOLOGY_169: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_170: (12)                         vmobject,
SNGT_QHENOMENOLOGY_171: (12)                         rate_func=rate_func,
SNGT_QHENOMENOLOGY_172: (12)                         remover=remover,
SNGT_QHENOMENOLOGY_173: (12)                         final_alpha_value=final_alpha_value,
SNGT_QHENOMENOLOGY_174: (12)                         **kwargs
SNGT_QHENOMENOLOGY_175: (8)                     )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 5 - growing.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.animation.transform import Transform
SNGT_QHENOMENOLOGY_3: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_4: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_5: (4)                   import numpy as np
SNGT_QHENOMENOLOGY_6: (4)                   from manimlib.mobject.geometry import Arrow
SNGT_QHENOMENOLOGY_7: (4)                   from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_8: (4)                   from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_9: (0)               class GrowFromPoint(Transform):
SNGT_QHENOMENOLOGY_10: (4)                  def __init__(
SNGT_QHENOMENOLOGY_11: (8)                      self,
SNGT_QHENOMENOLOGY_12: (8)                      mobject: Mobject,
SNGT_QHENOMENOLOGY_13: (8)                      point: np.ndarray,
SNGT_QHENOMENOLOGY_14: (8)                      point_color: ManimColor = None,
SNGT_QHENOMENOLOGY_15: (8)                      **kwargs
SNGT_QHENOMENOLOGY_16: (4)                  ):
SNGT_QHENOMENOLOGY_17: (8)                      self.point = point
SNGT_QHENOMENOLOGY_18: (8)                      self.point_color = point_color
SNGT_QHENOMENOLOGY_19: (8)                      super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_20: (4)                  def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_21: (8)                      return self.mobject.copy()
SNGT_QHENOMENOLOGY_22: (4)                  def create_starting_mobject(self) -> Mobject:
SNGT_QHENOMENOLOGY_23: (8)                      start = super().create_starting_mobject()
SNGT_QHENOMENOLOGY_24: (8)                      start.scale(0)
SNGT_QHENOMENOLOGY_25: (8)                      start.move_to(self.point)
SNGT_QHENOMENOLOGY_26: (8)                      if self.point_color is not None:
SNGT_QHENOMENOLOGY_27: (12)                         start.set_color(self.point_color)
SNGT_QHENOMENOLOGY_28: (8)                      return start
SNGT_QHENOMENOLOGY_29: (0)              class GrowFromCenter(GrowFromPoint):
SNGT_QHENOMENOLOGY_30: (4)                  def __init__(self, mobject: Mobject, **kwargs):
SNGT_QHENOMENOLOGY_31: (8)                      point = mobject.get_center()
SNGT_QHENOMENOLOGY_32: (8)                      super().__init__(mobject, point, **kwargs)
SNGT_QHENOMENOLOGY_33: (0)              class GrowFromEdge(GrowFromPoint):
SNGT_QHENOMENOLOGY_34: (4)                  def __init__(self, mobject: Mobject, edge: np.ndarray,
**kwargs):
SNGT_QHENOMENOLOGY_35: (8)                      point = mobject.get_bounding_box_point(edge)
```

```
SNGT_QHENOMENOLOGY_36: (8)                      super().__init__(mobject, point, **kwargs)
SNGT_QHENOMENOLOGY_37: (0)              class GrowArrow(GrowFromPoint):
SNGT_QHENOMENOLOGY_38: (4)                  def __init__(self, arrow: Arrow, **kwargs):
SNGT_QHENOMENOLOGY_39: (8)                      point = arrow.get_start()
SNGT_QHENOMENOLOGY_40: (8)                      super().__init__(arrow, point, **kwargs)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 6 - numbers.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.mobject.numbers import DecimalNumber
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_6: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_7: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_8: (4)                   from typing import Callable
SNGT_QHENOMENOLOGY_9: (0)               class ChangingDecimal(Animation):
SNGT_QHENOMENOLOGY_10: (4)                  def __init__(
SNGT_QHENOMENOLOGY_11: (8)                      self,
SNGT_QHENOMENOLOGY_12: (8)                      decimal_mob: DecimalNumber,
SNGT_QHENOMENOLOGY_13: (8)                      number_update_func: Callable[[float], float],
SNGT_QHENOMENOLOGY_14: (8)                      suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_15: (8)                      **kwargs
SNGT_QHENOMENOLOGY_16: (4)                  ):
SNGT_QHENOMENOLOGY_17: (8)                      assert isinstance(decimal_mob, DecimalNumber)
SNGT_QHENOMENOLOGY_18: (8)                      self.number_update_func = number_update_func
SNGT_QHENOMENOLOGY_19: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_20: (12)                         decimal_mob,
SNGT_QHENOMENOLOGY_21: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_22: (12)                         **kwargs
SNGT_QHENOMENOLOGY_23: (8)                      )
SNGT_QHENOMENOLOGY_24: (8)                      self.mobject = decimal_mob
SNGT_QHENOMENOLOGY_25: (4)                  def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_26: (8)                      self.mobject.set_value(
SNGT_QHENOMENOLOGY_27: (12)                         self.number_update_func(alpha)
SNGT_QHENOMENOLOGY_28: (8)                      )
SNGT_QHENOMENOLOGY_29: (0)              class ChangeDecimalToValue(ChangingDecimal):
SNGT_QHENOMENOLOGY_30: (4)                  def __init__(
SNGT_QHENOMENOLOGY_31: (8)                      self,
SNGT_QHENOMENOLOGY_32: (8)                      decimal_mob: DecimalNumber,
SNGT_QHENOMENOLOGY_33: (8)                      target_number: float | complex,
SNGT_QHENOMENOLOGY_34: (8)                      **kwargs
SNGT_QHENOMENOLOGY_35: (4)                  ):
SNGT_QHENOMENOLOGY_36: (8)                      start_number = decimal_mob.number
SNGT_QHENOMENOLOGY_37: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_38: (12)                         decimal_mob,
SNGT_QHENOMENOLOGY_39: (12)                         lambda a: interpolate(start_number,
target_number, a),
SNGT_QHENOMENOLOGY_40: (12)                         **kwargs
SNGT_QHENOMENOLOGY_41: (8)                      )
SNGT_QHENOMENOLOGY_42: (0)              class CountInFrom(ChangingDecimal):
SNGT_QHENOMENOLOGY_43: (4)                  def __init__(
SNGT_QHENOMENOLOGY_44: (8)                      self,
SNGT_QHENOMENOLOGY_45: (8)                      decimal_mob: DecimalNumber,
SNGT_QHENOMENOLOGY_46: (8)                      source_number: float | complex = 0,
SNGT_QHENOMENOLOGY_47: (8)                      **kwargs
SNGT_QHENOMENOLOGY_48: (4)                  ):
SNGT_QHENOMENOLOGY_49: (8)                      start_number = decimal_mob.get_value()
SNGT_QHENOMENOLOGY_50: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_51: (12)                         decimal_mob,
SNGT_QHENOMENOLOGY_52: (12)                         lambda a: interpolate(source_number,
start_number, clip(a, 0, 1)),
SNGT_QHENOMENOLOGY_53: (12)                         **kwargs
SNGT_QHENOMENOLOGY_54: (8)                      )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 7 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              import pkg_resources
SNGT_QHENOMENOLOGY_2: (0)              __version__ =
pkg_resources.get_distribution("manimgl").version
SNGT_QHENOMENOLOGY_3: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_4: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_5: (4)                  from manimlib.typing import *
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.constants import *
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.window import *
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.animation.animation import *
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.animation.composition import *
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.animation.creation import *
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.animation.fading import *
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.animation.growing import *
SNGT_QHENOMENOLOGY_13: (0)             from manimlib.animation.indication import *
SNGT_QHENOMENOLOGY_14: (0)             from manimlib.animation.movement import *
SNGT_QHENOMENOLOGY_15: (0)             from manimlib.animation.numbers import *
SNGT_QHENOMENOLOGY_16: (0)             from manimlib.animation.rotation import *
SNGT_QHENOMENOLOGY_17: (0)             from manimlib.animation.specialized import *
SNGT_QHENOMENOLOGY_18: (0)             from manimlib.animation.transform import *
SNGT_QHENOMENOLOGY_19: (0)             from manimlib.animation.transform_matching_parts import *
SNGT_QHENOMENOLOGY_20: (0)             from manimlib.animation.update import *
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.camera.camera import *
SNGT_QHENOMENOLOGY_22: (0)             from manimlib.mobject.boolean_ops import *
SNGT_QHENOMENOLOGY_23: (0)             from manimlib.mobject.changing import *
SNGT_QHENOMENOLOGY_24: (0)             from manimlib.mobject.coordinate_systems import *
SNGT_QHENOMENOLOGY_25: (0)             from manimlib.mobject.frame import *
SNGT_QHENOMENOLOGY_26: (0)             from manimlib.mobject.functions import *
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.mobject.geometry import *
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.mobject.interactive import *
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.mobject.matrix import *
SNGT_QHENOMENOLOGY_30: (0)             from manimlib.mobject.mobject import *
SNGT_QHENOMENOLOGY_31: (0)             from manimlib.mobject.mobject_update_utils import *
SNGT_QHENOMENOLOGY_32: (0)             from manimlib.mobject.number_line import *
SNGT_QHENOMENOLOGY_33: (0)             from manimlib.mobject.numbers import *
SNGT_QHENOMENOLOGY_34: (0)             from manimlib.mobject.probability import *
SNGT_QHENOMENOLOGY_35: (0)             from manimlib.mobject.shape_matchers import *
SNGT_QHENOMENOLOGY_36: (0)             from manimlib.mobject.svg.brace import *
SNGT_QHENOMENOLOGY_37: (0)             from manimlib.mobject.svg.drawings import *
SNGT_QHENOMENOLOGY_38: (0)             from manimlib.mobject.svg.string_mobject import *
SNGT_QHENOMENOLOGY_39: (0)             from manimlib.mobject.svg.svg_mobject import *
SNGT_QHENOMENOLOGY_40: (0)             from manimlib.mobject.svg.special_tex import *
SNGT_QHENOMENOLOGY_41: (0)             from manimlib.mobject.svg.tex_mobject import *
SNGT_QHENOMENOLOGY_42: (0)             from manimlib.mobject.svg.text_mobject import *
SNGT_QHENOMENOLOGY_43: (0)             from manimlib.mobject.three_dimensions import *
SNGT_QHENOMENOLOGY_44: (0)             from manimlib.mobject.types.dot_cloud import *
SNGT_QHENOMENOLOGY_45: (0)             from manimlib.mobject.types.image_mobject import *
SNGT_QHENOMENOLOGY_46: (0)             from manimlib.mobject.types.point_cloud_mobject import *
SNGT_QHENOMENOLOGY_47: (0)             from manimlib.mobject.types.surface import *
SNGT_QHENOMENOLOGY_48: (0)             from manimlib.mobject.types.vectorized_mobject import *
SNGT_QHENOMENOLOGY_49: (0)             from manimlib.mobject.value_tracker import *
SNGT_QHENOMENOLOGY_50: (0)             from manimlib.mobject.vector_field import *
SNGT_QHENOMENOLOGY_51: (0)             from manimlib.scene.interactive_scene import *
SNGT_QHENOMENOLOGY_52: (0)             from manimlib.scene.scene import *
SNGT_QHENOMENOLOGY_53: (0)             from manimlib.utils.bezier import *
SNGT_QHENOMENOLOGY_54: (0)             from manimlib.utils.cache import *
SNGT_QHENOMENOLOGY_55: (0)             from manimlib.utils.color import *
SNGT_QHENOMENOLOGY_56: (0)             from manimlib.utils.dict_ops import *
SNGT_QHENOMENOLOGY_57: (0)             from manimlib.utils.debug import *
SNGT_QHENOMENOLOGY_58: (0)             from manimlib.utils.directories import *
SNGT_QHENOMENOLOGY_59: (0)             from manimlib.utils.file_ops import *
SNGT_QHENOMENOLOGY_60: (0)             from manimlib.utils.images import *
SNGT_QHENOMENOLOGY_61: (0)             from manimlib.utils.iterables import *
SNGT_QHENOMENOLOGY_62: (0)             from manimlib.utils.paths import *
SNGT_QHENOMENOLOGY_63: (0)             from manimlib.utils.rate_functions import *
SNGT_QHENOMENOLOGY_64: (0)             from manimlib.utils.simple_functions import *
SNGT_QHENOMENOLOGY_65: (0)             from manimlib.utils.shaders import *
```

```
SNGT_QHENOMENOLOGY_66: (0)              from manimlib.utils.sounds import *
SNGT_QHENOMENOLOGY_67: (0)              from manimlib.utils.space_ops import *
SNGT_QHENOMENOLOGY_68: (0)              from manimlib.utils.tex import *
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 8 - __main__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from addict import Dict
SNGT_QHENOMENOLOGY_2: (0)               from manimlib import __version__
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.config import parse_cli
SNGT_QHENOMENOLOGY_5: (0)               import manimlib.extract_scene
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.utils.cache import clear_cache
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.window import Window
SNGT_QHENOMENOLOGY_8: (0)               from IPython.terminal.embed import KillEmbedded
SNGT_QHENOMENOLOGY_9: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                  from argparse import Namespace
SNGT_QHENOMENOLOGY_12: (0)              def run_scenes():
SNGT_QHENOMENOLOGY_13: (4)                  """
SNGT_QHENOMENOLOGY_14: (4)                  Runs the scenes in a loop and detects when a scene
reload is requested.
SNGT_QHENOMENOLOGY_15: (4)                  """
SNGT_QHENOMENOLOGY_16: (4)                  scene_config = Dict(manim_config.scene)
SNGT_QHENOMENOLOGY_17: (4)                  run_config = manim_config.run
SNGT_QHENOMENOLOGY_18: (4)                  if run_config.show_in_window:
SNGT_QHENOMENOLOGY_19: (8)                      window = Window(**manim_config.window)
SNGT_QHENOMENOLOGY_20: (8)                      scene_config.update(window=window)
SNGT_QHENOMENOLOGY_21: (4)                  while True:
SNGT_QHENOMENOLOGY_22: (8)                      try:
SNGT_QHENOMENOLOGY_23: (12)                         scenes =
manimlib.extract_scene.main(scene_config, run_config)
SNGT_QHENOMENOLOGY_24: (12)                         for scene in scenes:
SNGT_QHENOMENOLOGY_25: (16)                             scene.run()
SNGT_QHENOMENOLOGY_26: (12)                         return
SNGT_QHENOMENOLOGY_27: (8)                      except KillEmbedded:
SNGT_QHENOMENOLOGY_28: (12)                         pass
SNGT_QHENOMENOLOGY_29: (8)                      except KeyboardInterrupt:
SNGT_QHENOMENOLOGY_30: (12)                         break
SNGT_QHENOMENOLOGY_31: (0)              def main():
SNGT_QHENOMENOLOGY_32: (4)                  """
SNGT_QHENOMENOLOGY_33: (4)                  Main entry point for ManimGL.
SNGT_QHENOMENOLOGY_34: (4)                  """
SNGT_QHENOMENOLOGY_35: (4)                  print(f"ManimGL \033[32mv{__version__}\033[0m")
SNGT_QHENOMENOLOGY_36: (4)                  args = parse_cli()
SNGT_QHENOMENOLOGY_37: (4)                  if args.version and args.file is None:
SNGT_QHENOMENOLOGY_38: (8)                      return
SNGT_QHENOMENOLOGY_39: (4)                  if args.clear_cache:
SNGT_QHENOMENOLOGY_40: (8)                      clear_cache()
SNGT_QHENOMENOLOGY_41: (4)                  run_scenes()
SNGT_QHENOMENOLOGY_42: (0)              if __name__ == "__main__":
SNGT_QHENOMENOLOGY_43: (4)                  main()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 9 - creation.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from abc import ABC, abstractmethod
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.mobject.svg.string_mobject import
StringMobject
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.utils.bezier import integer_interpolate
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.utils.rate_functions import linear
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.rate_functions import double_smooth
```

```
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.utils.rate_functions import smooth
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_12: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_13: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_14: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_15: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_16: (4)                 from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_17: (4)                 from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_18: (0)             class ShowPartial(Animation, ABC):
SNGT_QHENOMENOLOGY_19: (4)                 """
SNGT_QHENOMENOLOGY_20: (4)                 Abstract class for ShowCreation and ShowPassingFlash
SNGT_QHENOMENOLOGY_21: (4)                 """
SNGT_QHENOMENOLOGY_22: (4)                 def __init__(self, mobject: Mobject,
should_match_start: bool = False, **kwargs):
SNGT_QHENOMENOLOGY_23: (8)                     self.should_match_start = should_match_start
SNGT_QHENOMENOLOGY_24: (8)                     super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_25: (4)                 def interpolate_submobject(
SNGT_QHENOMENOLOGY_26: (8)                     self,
SNGT_QHENOMENOLOGY_27: (8)                     submob: VMobject,
SNGT_QHENOMENOLOGY_28: (8)                     start_submob: VMobject,
SNGT_QHENOMENOLOGY_29: (8)                     alpha: float
SNGT_QHENOMENOLOGY_30: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_31: (8)                     submob.pointwise_become_partial(
SNGT_QHENOMENOLOGY_32: (12)                        start_submob, *self.get_bounds(alpha)
SNGT_QHENOMENOLOGY_33: (8)                     )
SNGT_QHENOMENOLOGY_34: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_35: (4)                 def get_bounds(self, alpha: float) -> tuple[float,
float]:
SNGT_QHENOMENOLOGY_36: (8)                     raise Exception("Not Implemented")
SNGT_QHENOMENOLOGY_37: (0)             class ShowCreation(ShowPartial):
SNGT_QHENOMENOLOGY_38: (4)                 def __init__(self, mobject: Mobject, lag_ratio: float =
1.0, **kwargs):
SNGT_QHENOMENOLOGY_39: (8)                     super().__init__(mobject, lag_ratio=lag_ratio,
**kwargs)
SNGT_QHENOMENOLOGY_40: (4)                 def get_bounds(self, alpha: float) -> tuple[float,
float]:
SNGT_QHENOMENOLOGY_41: (8)                     return (0, alpha)
SNGT_QHENOMENOLOGY_42: (0)             class Uncreate(ShowCreation):
SNGT_QHENOMENOLOGY_43: (4)                 def __init__(
SNGT_QHENOMENOLOGY_44: (8)                     self,
SNGT_QHENOMENOLOGY_45: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_46: (8)                     rate_func: Callable[[float], float] = lambda t:
smooth(1 - t),
SNGT_QHENOMENOLOGY_47: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_48: (8)                     should_match_start: bool = True,
SNGT_QHENOMENOLOGY_49: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_50: (4)                 ):
SNGT_QHENOMENOLOGY_51: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_52: (12)                        mobject,
SNGT_QHENOMENOLOGY_53: (12)                        rate_func=rate_func,
SNGT_QHENOMENOLOGY_54: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_55: (12)                        should_match_start=should_match_start,
SNGT_QHENOMENOLOGY_56: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_57: (8)                     )
SNGT_QHENOMENOLOGY_58: (0)             class DrawBorderThenFill(Animation):
SNGT_QHENOMENOLOGY_59: (4)                 def __init__(
SNGT_QHENOMENOLOGY_60: (8)                     self,
SNGT_QHENOMENOLOGY_61: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_62: (8)                     run_time: float = 2.0,
SNGT_QHENOMENOLOGY_63: (8)                     rate_func: Callable[[float], float] =
double_smooth,
SNGT_QHENOMENOLOGY_64: (8)                     stroke_width: float = 2.0,
SNGT_QHENOMENOLOGY_65: (8)                     stroke_color: ManimColor = None,
SNGT_QHENOMENOLOGY_66: (8)                     draw_border_animation_config: dict = {},
SNGT_QHENOMENOLOGY_67: (8)                     fill_animation_config: dict = {},
SNGT_QHENOMENOLOGY_68: (8)                     **kwargs
SNGT_QHENOMENOLOGY_69: (4)                 ):
SNGT_QHENOMENOLOGY_70: (8)                     assert isinstance(vmobject, VMobject)
SNGT_QHENOMENOLOGY_71: (8)                     self.sm_to_index = {hash(sm): 0 for sm in
```

```
vmobject.get_family()}
                    self.stroke_width = stroke_width
                    self.stroke_color = stroke_color
                    self.draw_border_animation_config =
draw_border_animation_config
                    self.fill_animation_config = fill_animation_config
                    super().__init__(
                        vmobject,
                        run_time=run_time,
                        rate_func=rate_func,
                        **kwargs
                    )
                    self.mobject = vmobject
                def begin(self) -> None:
                    self.mobject.set_animating_status(True)
                    self.outline = self.get_outline()
                    super().begin()
                    self.mobject.match_style(self.outline)
                def finish(self) -> None:
                    super().finish()
                    self.mobject.refresh_joint_angles()
                def get_outline(self) -> VMobject:
                    outline = self.mobject.copy()
                    outline.set_fill(opacity=0)
                    for sm in outline.family_members_with_points():
                        sm.set_stroke(
                            color=self.stroke_color or
sm.get_stroke_color(),
                            width=self.stroke_width,
                            behind=self.mobject.stroke_behind,
                        )
                    return outline
                def get_all_mobjects(self) -> list[Mobject]:
                    return [*super().get_all_mobjects(), self.outline]
                def interpolate_submobject(
                    self,
                    submob: VMobject,
                    start: VMobject,
                    outline: VMobject,
                    alpha: float
                ) -> None:
                    index, subalpha = integer_interpolate(0, 2, alpha)
                    if index == 1 and self.sm_to_index[hash(submob)] ==
0:
                        submob.set_data(outline.data)
                        self.sm_to_index[hash(submob)] = 1
                    if index == 0:
                        submob.pointwise_become_partial(outline, 0,
subalpha)
                    else:
                        submob.interpolate(outline, start, subalpha)
            class Write(DrawBorderThenFill):
                def __init__(
                    self,
                    vmobject: VMobject,
                    run_time: float = -1,  # If negative, this will be
reassigned
                    lag_ratio: float = -1,  # If negative, this will be
reassigned
                    rate_func: Callable[[float], float] = linear,
                    stroke_color: ManimColor = None,
                    **kwargs
                ):
                    if stroke_color is None:
                        stroke_color = vmobject.get_color()
                    family_size =
len(vmobject.family_members_with_points())
                    super().__init__(
                        vmobject,
```

```
SNGT_QHENOMENOLOGY_133: (12)                        run_time=self.compute_run_time(family_size,
run_time),
SNGT_QHENOMENOLOGY_134: (12)                        lag_ratio=self.compute_lag_ratio(family_size,
lag_ratio),
SNGT_QHENOMENOLOGY_135: (12)                        rate_func=rate_func,
SNGT_QHENOMENOLOGY_136: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_137: (12)                        **kwargs
SNGT_QHENOMENOLOGY_138: (8)                    )
SNGT_QHENOMENOLOGY_139: (4)                def compute_run_time(self, family_size: int, run_time:
float):
SNGT_QHENOMENOLOGY_140: (8)                    if run_time < 0:
SNGT_QHENOMENOLOGY_141: (12)                        return 1 if family_size < 15 else 2
SNGT_QHENOMENOLOGY_142: (8)                    return run_time
SNGT_QHENOMENOLOGY_143: (4)                def compute_lag_ratio(self, family_size: int,
lag_ratio: float):
SNGT_QHENOMENOLOGY_144: (8)                    if lag_ratio < 0:
SNGT_QHENOMENOLOGY_145: (12)                        return min(4.0 / (family_size + 1.0), 0.2)
SNGT_QHENOMENOLOGY_146: (8)                    return lag_ratio
SNGT_QHENOMENOLOGY_147: (0)            class ShowIncreasingSubsets(Animation):
SNGT_QHENOMENOLOGY_148: (4)                def __init__(
SNGT_QHENOMENOLOGY_149: (8)                    self,
SNGT_QHENOMENOLOGY_150: (8)                    group: Mobject,
SNGT_QHENOMENOLOGY_151: (8)                    int_func: Callable[[float], float] = np.round,
SNGT_QHENOMENOLOGY_152: (8)                    suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_153: (8)                    **kwargs
SNGT_QHENOMENOLOGY_154: (4)                ):
SNGT_QHENOMENOLOGY_155: (8)                    self.all_submobs = list(group.submobjects)
SNGT_QHENOMENOLOGY_156: (8)                    self.int_func = int_func
SNGT_QHENOMENOLOGY_157: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_158: (12)                        group,
SNGT_QHENOMENOLOGY_159: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_160: (12)                        **kwargs
SNGT_QHENOMENOLOGY_161: (8)                    )
SNGT_QHENOMENOLOGY_162: (4)                def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_163: (8)                    n_submobs = len(self.all_submobs)
SNGT_QHENOMENOLOGY_164: (8)                    alpha = self.rate_func(alpha)
SNGT_QHENOMENOLOGY_165: (8)                    index = int(self.int_func(alpha * n_submobs))
SNGT_QHENOMENOLOGY_166: (8)                    self.update_submobject_list(index)
SNGT_QHENOMENOLOGY_167: (4)                def update_submobject_list(self, index: int) -> None:
SNGT_QHENOMENOLOGY_168: (8)
self.mobject.set_submobjects(self.all_submobs[:index])
SNGT_QHENOMENOLOGY_169: (0)            class ShowSubmobjectsOneByOne(ShowIncreasingSubsets):
SNGT_QHENOMENOLOGY_170: (4)                def __init__(
SNGT_QHENOMENOLOGY_171: (8)                    self,
SNGT_QHENOMENOLOGY_172: (8)                    group: Mobject,
SNGT_QHENOMENOLOGY_173: (8)                    int_func: Callable[[float], float] = np.ceil,
SNGT_QHENOMENOLOGY_174: (8)                    **kwargs
SNGT_QHENOMENOLOGY_175: (4)                ):
SNGT_QHENOMENOLOGY_176: (8)                    super().__init__(group, int_func=int_func,
**kwargs)
SNGT_QHENOMENOLOGY_177: (4)                def update_submobject_list(self, index: int) -> None:
SNGT_QHENOMENOLOGY_178: (8)                    index = int(clip(index, 0, len(self.all_submobs) -
1))
SNGT_QHENOMENOLOGY_179: (8)                    if index == 0:
SNGT_QHENOMENOLOGY_180: (12)                        self.mobject.set_submobjects([])
SNGT_QHENOMENOLOGY_181: (8)                    else:
SNGT_QHENOMENOLOGY_182: (12)
self.mobject.set_submobjects([self.all_submobs[index - 1]])
SNGT_QHENOMENOLOGY_183: (0)            class AddTextWordByWord(ShowIncreasingSubsets):
SNGT_QHENOMENOLOGY_184: (4)                def __init__(
SNGT_QHENOMENOLOGY_185: (8)                    self,
SNGT_QHENOMENOLOGY_186: (8)                    string_mobject: StringMobject,
SNGT_QHENOMENOLOGY_187: (8)                    time_per_word: float = 0.2,
SNGT_QHENOMENOLOGY_188: (8)                    run_time: float = -1.0, # If negative, it will be
recomputed with time_per_word
SNGT_QHENOMENOLOGY_189: (8)                    rate_func: Callable[[float], float] = linear,
SNGT_QHENOMENOLOGY_190: (8)                    **kwargs
SNGT_QHENOMENOLOGY_191: (4)                ):
```

```
SNGT_QHENOMENOLOGY_192: (8)                     assert isinstance(string_mobject, StringMobject)
SNGT_QHENOMENOLOGY_193: (8)                     grouped_mobject = string_mobject.build_groups()
SNGT_QHENOMENOLOGY_194: (8)                     if run_time < 0:
SNGT_QHENOMENOLOGY_195: (12)                        run_time = time_per_word * len(grouped_mobject)
SNGT_QHENOMENOLOGY_196: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_197: (12)                        grouped_mobject,
SNGT_QHENOMENOLOGY_198: (12)                        run_time=run_time,
SNGT_QHENOMENOLOGY_199: (12)                        rate_func=rate_func,
SNGT_QHENOMENOLOGY_200: (12)                        **kwargs
SNGT_QHENOMENOLOGY_201: (8)                     )
SNGT_QHENOMENOLOGY_202: (8)                     self.string_mobject = string_mobject
SNGT_QHENOMENOLOGY_203: (4)                 def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_204: (8)                     scene.remove(self.mobject)
SNGT_QHENOMENOLOGY_205: (8)                     if not self.is_remover():
SNGT_QHENOMENOLOGY_206: (12)                        scene.add(self.string_mobject)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 10 - movement.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.utils.rate_functions import linear
SNGT_QHENOMENOLOGY_4: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_5: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_6: (4)                 from typing import Callable, Sequence
SNGT_QHENOMENOLOGY_7: (4)                 import numpy as np
SNGT_QHENOMENOLOGY_8: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (4)                 from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_10: (0)            class Homotopy(Animation):
SNGT_QHENOMENOLOGY_11: (4)                apply_function_config: dict = dict()
SNGT_QHENOMENOLOGY_12: (4)                def __init__(
SNGT_QHENOMENOLOGY_13: (8)                    self,
SNGT_QHENOMENOLOGY_14: (8)                    homotopy: Callable[[float, float, float, float],
Sequence[float]],
SNGT_QHENOMENOLOGY_15: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_16: (8)                    run_time: float = 3.0,
SNGT_QHENOMENOLOGY_17: (8)                    **kwargs
SNGT_QHENOMENOLOGY_18: (4)                ):
SNGT_QHENOMENOLOGY_19: (8)                    """
SNGT_QHENOMENOLOGY_20: (8)                    Homotopy is a function from
SNGT_QHENOMENOLOGY_21: (8)                    (x, y, z, t) to (x', y', z')
SNGT_QHENOMENOLOGY_22: (8)                    """
SNGT_QHENOMENOLOGY_23: (8)                    self.homotopy = homotopy
SNGT_QHENOMENOLOGY_24: (8)                    super().__init__(mobject, run_time=run_time,
**kwargs)
SNGT_QHENOMENOLOGY_25: (4)                def function_at_time_t(self, t: float) ->
Callable[[np.ndarray], Sequence[float]]:
SNGT_QHENOMENOLOGY_26: (8)                    def result(p):
SNGT_QHENOMENOLOGY_27: (12)                       return self.homotopy(*p, t)
SNGT_QHENOMENOLOGY_28: (8)                    return result
SNGT_QHENOMENOLOGY_29: (4)                def interpolate_submobject(
SNGT_QHENOMENOLOGY_30: (8)                    self,
SNGT_QHENOMENOLOGY_31: (8)                    submob: Mobject,
SNGT_QHENOMENOLOGY_32: (8)                    start: Mobject,
SNGT_QHENOMENOLOGY_33: (8)                    alpha: float
SNGT_QHENOMENOLOGY_34: (4)                ) -> None:
SNGT_QHENOMENOLOGY_35: (8)                    submob.match_points(start)
SNGT_QHENOMENOLOGY_36: (8)                    submob.apply_function(
SNGT_QHENOMENOLOGY_37: (12)                       self.function_at_time_t(alpha),
SNGT_QHENOMENOLOGY_38: (12)                       **self.apply_function_config
SNGT_QHENOMENOLOGY_39: (8)                    )
SNGT_QHENOMENOLOGY_40: (0)            class SmoothedVectorizedHomotopy(Homotopy):
SNGT_QHENOMENOLOGY_41: (4)                apply_function_config: dict = dict(make_smooth=True)
SNGT_QHENOMENOLOGY_42: (0)            class ComplexHomotopy(Homotopy):
SNGT_QHENOMENOLOGY_43: (4)                def __init__(
SNGT_QHENOMENOLOGY_44: (8)                    self,
SNGT_QHENOMENOLOGY_45: (8)                    complex_homotopy: Callable[[complex, float],
```

```
complex],
SNGT_QHENOMENOLOGY_46: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_47: (8)                    **kwargs
SNGT_QHENOMENOLOGY_48: (4)                ):
SNGT_QHENOMENOLOGY_49: (8)                    """
SNGT_QHENOMENOLOGY_50: (8)                    Given a function form (z, t) -> w, where z and w
SNGT_QHENOMENOLOGY_51: (8)                    are complex numbers and t is time, this animates
SNGT_QHENOMENOLOGY_52: (8)                    the state over time
SNGT_QHENOMENOLOGY_53: (8)                    """
SNGT_QHENOMENOLOGY_54: (8)                    def homotopy(x, y, z, t):
SNGT_QHENOMENOLOGY_55: (12)                       c = complex_homotopy(complex(x, y), t)
SNGT_QHENOMENOLOGY_56: (12)                       return (c.real, c.imag, z)
SNGT_QHENOMENOLOGY_57: (8)                    super().__init__(homotopy, mobject, **kwargs)
SNGT_QHENOMENOLOGY_58: (0)            class PhaseFlow(Animation):
SNGT_QHENOMENOLOGY_59: (4)                def __init__(
SNGT_QHENOMENOLOGY_60: (8)                    self,
SNGT_QHENOMENOLOGY_61: (8)                    function: Callable[[np.ndarray], np.ndarray],
SNGT_QHENOMENOLOGY_62: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_63: (8)                    virtual_time: float | None = None,
SNGT_QHENOMENOLOGY_64: (8)                    suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_65: (8)                    rate_func: Callable[[float], float] = linear,
SNGT_QHENOMENOLOGY_66: (8)                    run_time: float =3.0,
SNGT_QHENOMENOLOGY_67: (8)                    **kwargs
SNGT_QHENOMENOLOGY_68: (4)                ):
SNGT_QHENOMENOLOGY_69: (8)                    self.function = function
SNGT_QHENOMENOLOGY_70: (8)                    self.virtual_time = virtual_time or run_time
SNGT_QHENOMENOLOGY_71: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_72: (12)                       mobject,
SNGT_QHENOMENOLOGY_73: (12)                       rate_func=rate_func,
SNGT_QHENOMENOLOGY_74: (12)                       run_time=run_time,
SNGT_QHENOMENOLOGY_75: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_76: (12)                       **kwargs
SNGT_QHENOMENOLOGY_77: (8)                    )
SNGT_QHENOMENOLOGY_78: (4)                def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_79: (8)                    if hasattr(self, "last_alpha"):
SNGT_QHENOMENOLOGY_80: (12)                       dt = self.virtual_time * (alpha -
self.last_alpha)
SNGT_QHENOMENOLOGY_81: (12)                       self.mobject.apply_function(
SNGT_QHENOMENOLOGY_82: (16)                           lambda p: p + dt * self.function(p)
SNGT_QHENOMENOLOGY_83: (12)                       )
SNGT_QHENOMENOLOGY_84: (8)                    self.last_alpha = alpha
SNGT_QHENOMENOLOGY_85: (0)            class MoveAlongPath(Animation):
SNGT_QHENOMENOLOGY_86: (4)                def __init__(
SNGT_QHENOMENOLOGY_87: (8)                    self,
SNGT_QHENOMENOLOGY_88: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_89: (8)                    path: VMobject,
SNGT_QHENOMENOLOGY_90: (8)                    suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_91: (8)                    **kwargs
SNGT_QHENOMENOLOGY_92: (4)                ):
SNGT_QHENOMENOLOGY_93: (8)                    self.path = path
SNGT_QHENOMENOLOGY_94: (8)                    super().__init__(mobject,
suspend_mobject_updating=suspend_mobject_updating, **kwargs)
SNGT_QHENOMENOLOGY_95: (4)                def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_96: (8)                    point =
self.path.quick_point_from_proportion(self.rate_func(alpha))
SNGT_QHENOMENOLOGY_97: (8)                    self.mobject.move_to(point)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 11 - rotation.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.constants import ORIGIN, OUT
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.constants import PI, TAU
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.utils.rate_functions import linear
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.utils.rate_functions import smooth
SNGT_QHENOMENOLOGY_7: (0)             from typing import TYPE_CHECKING
```

```
SNGT_QHENOMENOLOGY_8: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                 import numpy as np
SNGT_QHENOMENOLOGY_10: (4)                from typing import Callable
SNGT_QHENOMENOLOGY_11: (4)                from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_12: (0)            class Rotating(Animation):
SNGT_QHENOMENOLOGY_13: (4)                def __init__(
SNGT_QHENOMENOLOGY_14: (8)                    self,
SNGT_QHENOMENOLOGY_15: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_16: (8)                    angle: float = TAU,
SNGT_QHENOMENOLOGY_17: (8)                    axis: np.ndarray = OUT,
SNGT_QHENOMENOLOGY_18: (8)                    about_point: np.ndarray | None = None,
SNGT_QHENOMENOLOGY_19: (8)                    about_edge: np.ndarray | None = None,
SNGT_QHENOMENOLOGY_20: (8)                    run_time: float = 5.0,
SNGT_QHENOMENOLOGY_21: (8)                    rate_func: Callable[[float], float] = linear,
SNGT_QHENOMENOLOGY_22: (8)                    suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_23: (8)                    **kwargs
SNGT_QHENOMENOLOGY_24: (4)                ):
SNGT_QHENOMENOLOGY_25: (8)                    self.angle = angle
SNGT_QHENOMENOLOGY_26: (8)                    self.axis = axis
SNGT_QHENOMENOLOGY_27: (8)                    self.about_point = about_point
SNGT_QHENOMENOLOGY_28: (8)                    self.about_edge = about_edge
SNGT_QHENOMENOLOGY_29: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_30: (12)                       mobject,
SNGT_QHENOMENOLOGY_31: (12)                       run_time=run_time,
SNGT_QHENOMENOLOGY_32: (12)                       rate_func=rate_func,
SNGT_QHENOMENOLOGY_33: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_34: (12)                       **kwargs
SNGT_QHENOMENOLOGY_35: (8)                    )
SNGT_QHENOMENOLOGY_36: (4)                def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_37: (8)                    pairs = zip(
SNGT_QHENOMENOLOGY_38: (12)                       self.mobject.family_members_with_points(),
SNGT_QHENOMENOLOGY_39: (12)
self.starting_mobject.family_members_with_points(),
SNGT_QHENOMENOLOGY_40: (8)                    )
SNGT_QHENOMENOLOGY_41: (8)                    for sm1, sm2 in pairs:
SNGT_QHENOMENOLOGY_42: (12)                       for key in sm1.pointlike_data_keys:
SNGT_QHENOMENOLOGY_43: (16)                           sm1.data[key][:] = sm2.data[key]
SNGT_QHENOMENOLOGY_44: (8)                    self.mobject.rotate(
SNGT_QHENOMENOLOGY_45: (12)                       self.rate_func(self.time_spanned_alpha(alpha))
* self.angle,
SNGT_QHENOMENOLOGY_46: (12)                       axis=self.axis,
SNGT_QHENOMENOLOGY_47: (12)                       about_point=self.about_point,
SNGT_QHENOMENOLOGY_48: (12)                       about_edge=self.about_edge,
SNGT_QHENOMENOLOGY_49: (8)                    )
SNGT_QHENOMENOLOGY_50: (0)            class Rotate(Rotating):
SNGT_QHENOMENOLOGY_51: (4)                def __init__(
SNGT_QHENOMENOLOGY_52: (8)                    self,
SNGT_QHENOMENOLOGY_53: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_54: (8)                    angle: float = PI,
SNGT_QHENOMENOLOGY_55: (8)                    axis: np.ndarray = OUT,
SNGT_QHENOMENOLOGY_56: (8)                    run_time: float = 1,
SNGT_QHENOMENOLOGY_57: (8)                    rate_func: Callable[[float], float] = smooth,
SNGT_QHENOMENOLOGY_58: (8)                    about_edge: np.ndarray = ORIGIN,
SNGT_QHENOMENOLOGY_59: (8)                    **kwargs
SNGT_QHENOMENOLOGY_60: (4)                ):
SNGT_QHENOMENOLOGY_61: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_62: (12)                       mobject, angle, axis,
SNGT_QHENOMENOLOGY_63: (12)                       run_time=run_time,
SNGT_QHENOMENOLOGY_64: (12)                       rate_func=rate_func,
SNGT_QHENOMENOLOGY_65: (12)                       about_edge=about_edge,
SNGT_QHENOMENOLOGY_66: (12)                       **kwargs
SNGT_QHENOMENOLOGY_67: (8)                    )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 12 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 13 - animation.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from copy import deepcopy
SNGT_QHENOMENOLOGY_3: (0)              from manimlib.mobject.mobject import _AnimationBuilder
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.utils.iterables import
remove_list_redundancies
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.utils.rate_functions import smooth
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_8: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_9: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_10: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_11: (4)                 from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_12: (0)             DEFAULT_ANIMATION_RUN_TIME = 1.0
SNGT_QHENOMENOLOGY_13: (0)             DEFAULT_ANIMATION_LAG_RATIO = 0
SNGT_QHENOMENOLOGY_14: (0)             class Animation(object):
SNGT_QHENOMENOLOGY_15: (4)                 def __init__(
SNGT_QHENOMENOLOGY_16: (8)                     self,
SNGT_QHENOMENOLOGY_17: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_18: (8)                     run_time: float = DEFAULT_ANIMATION_RUN_TIME,
SNGT_QHENOMENOLOGY_19: (8)                     time_span: tuple[float, float] | None = None,
SNGT_QHENOMENOLOGY_20: (8)                     lag_ratio: float = DEFAULT_ANIMATION_LAG_RATIO,
SNGT_QHENOMENOLOGY_21: (8)                     rate_func: Callable[[float], float] = smooth,
SNGT_QHENOMENOLOGY_22: (8)                     name: str = "",
SNGT_QHENOMENOLOGY_23: (8)                     remover: bool = False,
SNGT_QHENOMENOLOGY_24: (8)                     final_alpha_value: float = 1.0,
SNGT_QHENOMENOLOGY_25: (8)                     suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_26: (4)                 ):
SNGT_QHENOMENOLOGY_27: (8)                     self.mobject = mobject
SNGT_QHENOMENOLOGY_28: (8)                     self.run_time = run_time
SNGT_QHENOMENOLOGY_29: (8)                     self.time_span = time_span
SNGT_QHENOMENOLOGY_30: (8)                     self.rate_func = rate_func
SNGT_QHENOMENOLOGY_31: (8)                     self.name = name or self.__class__.__name__ +
str(self.mobject)
SNGT_QHENOMENOLOGY_32: (8)                     self.remover = remover
SNGT_QHENOMENOLOGY_33: (8)                     self.final_alpha_value = final_alpha_value
SNGT_QHENOMENOLOGY_34: (8)                     self.lag_ratio = lag_ratio
SNGT_QHENOMENOLOGY_35: (8)                     self.suspend_mobject_updating =
suspend_mobject_updating
SNGT_QHENOMENOLOGY_36: (8)                     assert isinstance(mobject, Mobject)
SNGT_QHENOMENOLOGY_37: (4)                 def __str__(self) -> str:
SNGT_QHENOMENOLOGY_38: (8)                     return self.name
SNGT_QHENOMENOLOGY_39: (4)                 def begin(self) -> None:
SNGT_QHENOMENOLOGY_40: (8)                     if self.time_span is not None:
SNGT_QHENOMENOLOGY_41: (12)                        start, end = self.time_span
SNGT_QHENOMENOLOGY_42: (12)                        self.run_time = max(end, self.run_time)
SNGT_QHENOMENOLOGY_43: (8)                     self.mobject.set_animating_status(True)
SNGT_QHENOMENOLOGY_44: (8)                     self.starting_mobject =
self.create_starting_mobject()
SNGT_QHENOMENOLOGY_45: (8)                     if self.suspend_mobject_updating:
SNGT_QHENOMENOLOGY_46: (12)                        self.mobject_was_updating = not
self.mobject.updating_suspended
SNGT_QHENOMENOLOGY_47: (12)                        self.mobject.suspend_updating()
SNGT_QHENOMENOLOGY_48: (8)                     self.families =
list(self.get_all_families_zipped())
SNGT_QHENOMENOLOGY_49: (8)                     self.interpolate(0)
SNGT_QHENOMENOLOGY_50: (4)                 def finish(self) -> None:
SNGT_QHENOMENOLOGY_51: (8)                     self.interpolate(self.final_alpha_value)
SNGT_QHENOMENOLOGY_52: (8)                     self.mobject.set_animating_status(False)
SNGT_QHENOMENOLOGY_53: (8)                     if self.suspend_mobject_updating and
self.mobject_was_updating:
SNGT_QHENOMENOLOGY_54: (12)                        self.mobject.resume_updating()
SNGT_QHENOMENOLOGY_55: (4)                 def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_56: (8)                     if self.is_remover():
SNGT_QHENOMENOLOGY_57: (12)                        scene.remove(self.mobject)
```

```
SNGT_QHENOMENOLOGY_58: (4)                 def create_starting_mobject(self) -> Mobject:
SNGT_QHENOMENOLOGY_59: (8)                     return self.mobject.copy()
SNGT_QHENOMENOLOGY_60: (4)                 def get_all_mobjects(self) -> tuple[Mobject, Mobject]:
SNGT_QHENOMENOLOGY_61: (8)                     """
SNGT_QHENOMENOLOGY_62: (8)                     Ordering must match the ording of arguments to
interpolate_submobject
SNGT_QHENOMENOLOGY_63: (8)                     """
SNGT_QHENOMENOLOGY_64: (8)                     return self.mobject, self.starting_mobject
SNGT_QHENOMENOLOGY_65: (4)                 def get_all_families_zipped(self) ->
zip[tuple[Mobject]]:
SNGT_QHENOMENOLOGY_66: (8)                     return zip(*[
SNGT_QHENOMENOLOGY_67: (12)                        mob.get_family()
SNGT_QHENOMENOLOGY_68: (12)                        for mob in self.get_all_mobjects()
SNGT_QHENOMENOLOGY_69: (8)                     ])
SNGT_QHENOMENOLOGY_70: (4)                 def update_mobjects(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_71: (8)                     """
SNGT_QHENOMENOLOGY_72: (8)                     Updates things like starting_mobject, and (for
SNGT_QHENOMENOLOGY_73: (8)                     Transforms) target_mobject.
SNGT_QHENOMENOLOGY_74: (8)                     """
SNGT_QHENOMENOLOGY_75: (8)                     for mob in self.get_all_mobjects_to_update():
SNGT_QHENOMENOLOGY_76: (12)                        mob.update(dt)
SNGT_QHENOMENOLOGY_77: (4)                 def get_all_mobjects_to_update(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_78: (8)                     items = list(filter(
SNGT_QHENOMENOLOGY_79: (12)                        lambda m: m is not self.mobject,
SNGT_QHENOMENOLOGY_80: (12)                        self.get_all_mobjects()
SNGT_QHENOMENOLOGY_81: (8)                     ))
SNGT_QHENOMENOLOGY_82: (8)                     items = remove_list_redundancies(items)
SNGT_QHENOMENOLOGY_83: (8)                     return items
SNGT_QHENOMENOLOGY_84: (4)                 def copy(self):
SNGT_QHENOMENOLOGY_85: (8)                     return deepcopy(self)
SNGT_QHENOMENOLOGY_86: (4)                 def update_rate_info(
SNGT_QHENOMENOLOGY_87: (8)                     self,
SNGT_QHENOMENOLOGY_88: (8)                     run_time: float | None = None,
SNGT_QHENOMENOLOGY_89: (8)                     rate_func: Callable[[float], float] | None = None,
SNGT_QHENOMENOLOGY_90: (8)                     lag_ratio: float | None = None,
SNGT_QHENOMENOLOGY_91: (4)                 ):
SNGT_QHENOMENOLOGY_92: (8)                     self.run_time = run_time or self.run_time
SNGT_QHENOMENOLOGY_93: (8)                     self.rate_func = rate_func or self.rate_func
SNGT_QHENOMENOLOGY_94: (8)                     self.lag_ratio = lag_ratio or self.lag_ratio
SNGT_QHENOMENOLOGY_95: (8)                     return self
SNGT_QHENOMENOLOGY_96: (4)                 def interpolate(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_97: (8)                     self.interpolate_mobject(alpha)
SNGT_QHENOMENOLOGY_98: (4)                 def update(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_99: (8)                     """
SNGT_QHENOMENOLOGY_100: (8)                    This method shouldn't exist, but it's here to
SNGT_QHENOMENOLOGY_101: (8)                    keep many old scenes from breaking
SNGT_QHENOMENOLOGY_102: (8)                    """
SNGT_QHENOMENOLOGY_103: (8)                    self.interpolate(alpha)
SNGT_QHENOMENOLOGY_104: (4)                def time_spanned_alpha(self, alpha: float) -> float:
SNGT_QHENOMENOLOGY_105: (8)                    if self.time_span is not None:
SNGT_QHENOMENOLOGY_106: (12)                       start, end = self.time_span
SNGT_QHENOMENOLOGY_107: (12)                       return clip(alpha * self.run_time - start, 0,
end - start) / (end - start)
SNGT_QHENOMENOLOGY_108: (8)                    return alpha
SNGT_QHENOMENOLOGY_109: (4)                def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_110: (8)                    for i, mobs in enumerate(self.families):
SNGT_QHENOMENOLOGY_111: (12)                       sub_alpha =
self.get_sub_alpha(self.time_spanned_alpha(alpha), i, len(self.families))
SNGT_QHENOMENOLOGY_112: (12)                       self.interpolate_submobject(*mobs, sub_alpha)
SNGT_QHENOMENOLOGY_113: (4)                def interpolate_submobject(
SNGT_QHENOMENOLOGY_114: (8)                    self,
SNGT_QHENOMENOLOGY_115: (8)                    submobject: Mobject,
SNGT_QHENOMENOLOGY_116: (8)                    starting_submobject: Mobject,
SNGT_QHENOMENOLOGY_117: (8)                    alpha: float
SNGT_QHENOMENOLOGY_118: (4)                ):
SNGT_QHENOMENOLOGY_119: (8)                    pass
SNGT_QHENOMENOLOGY_120: (4)                def get_sub_alpha(
SNGT_QHENOMENOLOGY_121: (8)                    self,
SNGT_QHENOMENOLOGY_122: (8)                    alpha: float,
```

```
SNGT_QHENOMENOLOGY_123: (8)                     index: int,
SNGT_QHENOMENOLOGY_124: (8)                     num_submobjects: int
SNGT_QHENOMENOLOGY_125: (4)                 ) -> float:
SNGT_QHENOMENOLOGY_126: (8)                     lag_ratio = self.lag_ratio
SNGT_QHENOMENOLOGY_127: (8)                     full_length = (num_submobjects - 1) * lag_ratio + 1
SNGT_QHENOMENOLOGY_128: (8)                     value = alpha * full_length
SNGT_QHENOMENOLOGY_129: (8)                     lower = index * lag_ratio
SNGT_QHENOMENOLOGY_130: (8)                     raw_sub_alpha = clip((value - lower), 0, 1)
SNGT_QHENOMENOLOGY_131: (8)                     return self.rate_func(raw_sub_alpha)
SNGT_QHENOMENOLOGY_132: (4)                 def set_run_time(self, run_time: float):
SNGT_QHENOMENOLOGY_133: (8)                     self.run_time = run_time
SNGT_QHENOMENOLOGY_134: (8)                     return self
SNGT_QHENOMENOLOGY_135: (4)                 def get_run_time(self) -> float:
SNGT_QHENOMENOLOGY_136: (8)                     if self.time_span:
SNGT_QHENOMENOLOGY_137: (12)                        return max(self.run_time, self.time_span[1])
SNGT_QHENOMENOLOGY_138: (8)                     return self.run_time
SNGT_QHENOMENOLOGY_139: (4)                 def set_rate_func(self, rate_func: Callable[[float],
float]):
SNGT_QHENOMENOLOGY_140: (8)                     self.rate_func = rate_func
SNGT_QHENOMENOLOGY_141: (8)                     return self
SNGT_QHENOMENOLOGY_142: (4)                 def get_rate_func(self) -> Callable[[float], float]:
SNGT_QHENOMENOLOGY_143: (8)                     return self.rate_func
SNGT_QHENOMENOLOGY_144: (4)                 def set_name(self, name: str):
SNGT_QHENOMENOLOGY_145: (8)                     self.name = name
SNGT_QHENOMENOLOGY_146: (8)                     return self
SNGT_QHENOMENOLOGY_147: (4)                 def is_remover(self) -> bool:
SNGT_QHENOMENOLOGY_148: (8)                     return self.remover
SNGT_QHENOMENOLOGY_149: (0)             def prepare_animation(anim: Animation | _AnimationBuilder):
SNGT_QHENOMENOLOGY_150: (4)                 if isinstance(anim, _AnimationBuilder):
SNGT_QHENOMENOLOGY_151: (8)                     return anim.build()
SNGT_QHENOMENOLOGY_152: (4)                 if isinstance(anim, Animation):
SNGT_QHENOMENOLOGY_153: (8)                     return anim
SNGT_QHENOMENOLOGY_154: (4)                 raise TypeError(f"Object {anim} cannot be converted to
an animation")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 14 - indication.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.animation.composition import AnimationGroup
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.animation.composition import Succession
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.animation.creation import ShowCreation
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.animation.creation import ShowPartial
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.animation.fading import FadeOut
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.animation.fading import FadeIn
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.animation.movement import Homotopy
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.animation.transform import Transform
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.constants import FRAME_X_RADIUS,
FRAME_Y_RADIUS
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.constants import ORIGIN, RIGHT, UP
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.constants import SMALL_BUFF
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.constants import TAU
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.constants import GREY, YELLOW
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.mobject.geometry import Circle
SNGT_QHENOMENOLOGY_19: (0)              from manimlib.mobject.geometry import Dot
SNGT_QHENOMENOLOGY_20: (0)              from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_21: (0)              from manimlib.mobject.shape_matchers import
SurroundingRectangle
SNGT_QHENOMENOLOGY_22: (0)              from manimlib.mobject.shape_matchers import Underline
SNGT_QHENOMENOLOGY_23: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_24: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_25: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_26: (0)              from manimlib.utils.rate_functions import smooth
```

```
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.utils.rate_functions import squish_rate_func
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.rate_functions import there_and_back
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.utils.rate_functions import wiggle
SNGT_QHENOMENOLOGY_30: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_31: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_32: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_33: (4)                 from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_34: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_35: (0)             class FocusOn(Transform):
SNGT_QHENOMENOLOGY_36: (4)                 def __init__(
SNGT_QHENOMENOLOGY_37: (8)                     self,
SNGT_QHENOMENOLOGY_38: (8)                     focus_point: np.ndarray | Mobject,
SNGT_QHENOMENOLOGY_39: (8)                     opacity: float = 0.2,
SNGT_QHENOMENOLOGY_40: (8)                     color: ManimColor = GREY,
SNGT_QHENOMENOLOGY_41: (8)                     run_time: float = 2,
SNGT_QHENOMENOLOGY_42: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_43: (8)                     **kwargs
SNGT_QHENOMENOLOGY_44: (4)                 ):
SNGT_QHENOMENOLOGY_45: (8)                     self.focus_point = focus_point
SNGT_QHENOMENOLOGY_46: (8)                     self.opacity = opacity
SNGT_QHENOMENOLOGY_47: (8)                     self.color = color
SNGT_QHENOMENOLOGY_48: (8)                     super().__init__(VMobject(), run_time=run_time,
remover=remover, **kwargs)
SNGT_QHENOMENOLOGY_49: (4)                 def create_target(self) -> Dot:
SNGT_QHENOMENOLOGY_50: (8)                     little_dot = Dot(radius=0)
SNGT_QHENOMENOLOGY_51: (8)                     little_dot.set_fill(self.color,
opacity=self.opacity)
SNGT_QHENOMENOLOGY_52: (8)                     little_dot.add_updater(lambda d:
d.move_to(self.focus_point))
SNGT_QHENOMENOLOGY_53: (8)                     return little_dot
SNGT_QHENOMENOLOGY_54: (4)                 def create_starting_mobject(self) -> Dot:
SNGT_QHENOMENOLOGY_55: (8)                     return Dot(
SNGT_QHENOMENOLOGY_56: (12)                        radius=FRAME_X_RADIUS + FRAME_Y_RADIUS,
SNGT_QHENOMENOLOGY_57: (12)                        stroke_width=0,
SNGT_QHENOMENOLOGY_58: (12)                        fill_color=self.color,
SNGT_QHENOMENOLOGY_59: (12)                        fill_opacity=0,
SNGT_QHENOMENOLOGY_60: (8)                     )
SNGT_QHENOMENOLOGY_61: (0)             class Indicate(Transform):
SNGT_QHENOMENOLOGY_62: (4)                 def __init__(
SNGT_QHENOMENOLOGY_63: (8)                     self,
SNGT_QHENOMENOLOGY_64: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_65: (8)                     scale_factor: float = 1.2,
SNGT_QHENOMENOLOGY_66: (8)                     color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_67: (8)                     rate_func: Callable[[float], float] =
there_and_back,
SNGT_QHENOMENOLOGY_68: (8)                     **kwargs
SNGT_QHENOMENOLOGY_69: (4)                 ):
SNGT_QHENOMENOLOGY_70: (8)                     self.scale_factor = scale_factor
SNGT_QHENOMENOLOGY_71: (8)                     self.color = color
SNGT_QHENOMENOLOGY_72: (8)                     super().__init__(mobject, rate_func=rate_func,
**kwargs)
SNGT_QHENOMENOLOGY_73: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_74: (8)                     target = self.mobject.copy()
SNGT_QHENOMENOLOGY_75: (8)                     target.scale(self.scale_factor)
SNGT_QHENOMENOLOGY_76: (8)                     target.set_color(self.color)
SNGT_QHENOMENOLOGY_77: (8)                     return target
SNGT_QHENOMENOLOGY_78: (0)             class Flash(AnimationGroup):
SNGT_QHENOMENOLOGY_79: (4)                 def __init__(
SNGT_QHENOMENOLOGY_80: (8)                     self,
SNGT_QHENOMENOLOGY_81: (8)                     point: np.ndarray | Mobject,
SNGT_QHENOMENOLOGY_82: (8)                     color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_83: (8)                     line_length: float = 0.2,
SNGT_QHENOMENOLOGY_84: (8)                     num_lines: int = 12,
SNGT_QHENOMENOLOGY_85: (8)                     flash_radius: float = 0.3,
SNGT_QHENOMENOLOGY_86: (8)                     line_stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_87: (8)                     run_time: float = 1.0,
SNGT_QHENOMENOLOGY_88: (8)                     **kwargs
SNGT_QHENOMENOLOGY_89: (4)                 ):
SNGT_QHENOMENOLOGY_90: (8)                     self.point = point
```

```
SNGT_QHENOMENOLOGY_91: (8)                      self.color = color
SNGT_QHENOMENOLOGY_92: (8)                      self.line_length = line_length
SNGT_QHENOMENOLOGY_93: (8)                      self.num_lines = num_lines
SNGT_QHENOMENOLOGY_94: (8)                      self.flash_radius = flash_radius
SNGT_QHENOMENOLOGY_95: (8)                      self.line_stroke_width = line_stroke_width
SNGT_QHENOMENOLOGY_96: (8)                      self.lines = self.create_lines()
SNGT_QHENOMENOLOGY_97: (8)                      animations = self.create_line_anims()
SNGT_QHENOMENOLOGY_98: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_99: (12)                         *animations,
SNGT_QHENOMENOLOGY_100: (12)                        group=self.lines,
SNGT_QHENOMENOLOGY_101: (12)                        run_time=run_time,
SNGT_QHENOMENOLOGY_102: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_103: (8)                     )
SNGT_QHENOMENOLOGY_104: (4)                 def create_lines(self) -> VGroup:
SNGT_QHENOMENOLOGY_105: (8)                     lines = VGroup()
SNGT_QHENOMENOLOGY_106: (8)                     for angle in np.arange(0, TAU, TAU /
self.num_lines):
SNGT_QHENOMENOLOGY_107: (12)                        line = Line(ORIGIN, self.line_length * RIGHT)
SNGT_QHENOMENOLOGY_108: (12)                        line.shift((self.flash_radius -
self.line_length) * RIGHT)
SNGT_QHENOMENOLOGY_109: (12)                        line.rotate(angle, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_110: (12)                        lines.add(line)
SNGT_QHENOMENOLOGY_111: (8)                     lines.set_stroke(
SNGT_QHENOMENOLOGY_112: (12)                        color=self.color,
SNGT_QHENOMENOLOGY_113: (12)                        width=self.line_stroke_width
SNGT_QHENOMENOLOGY_114: (8)                     )
SNGT_QHENOMENOLOGY_115: (8)                     lines.add_updater(lambda l: l.move_to(self.point))
SNGT_QHENOMENOLOGY_116: (8)                     return lines
SNGT_QHENOMENOLOGY_117: (4)                 def create_line_anims(self) -> list[Animation]:
SNGT_QHENOMENOLOGY_118: (8)                     return [
SNGT_QHENOMENOLOGY_119: (12)                        ShowCreationThenDestruction(line)
SNGT_QHENOMENOLOGY_120: (12)                        for line in self.lines
SNGT_QHENOMENOLOGY_121: (8)                     ]
SNGT_QHENOMENOLOGY_122: (0)             class CircleIndicate(Transform):
SNGT_QHENOMENOLOGY_123: (4)                 def __init__(
SNGT_QHENOMENOLOGY_124: (8)                     self,
SNGT_QHENOMENOLOGY_125: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_126: (8)                     scale_factor: float = 1.2,
SNGT_QHENOMENOLOGY_127: (8)                     rate_func: Callable[[float], float] =
there_and_back,
SNGT_QHENOMENOLOGY_128: (8)                     stroke_color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_129: (8)                     stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_130: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_131: (8)                     **kwargs
SNGT_QHENOMENOLOGY_132: (4)                 ):
SNGT_QHENOMENOLOGY_133: (8)                     circle = Circle(stroke_color=stroke_color,
stroke_width=stroke_width)
SNGT_QHENOMENOLOGY_134: (8)                     circle.surround(mobject)
SNGT_QHENOMENOLOGY_135: (8)                     pre_circle = circle.copy().set_stroke(width=0)
SNGT_QHENOMENOLOGY_136: (8)                     pre_circle.scale(1 / scale_factor)
SNGT_QHENOMENOLOGY_137: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_138: (12)                        pre_circle, circle,
SNGT_QHENOMENOLOGY_139: (12)                        rate_func=rate_func,
SNGT_QHENOMENOLOGY_140: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_141: (12)                        **kwargs
SNGT_QHENOMENOLOGY_142: (8)                     )
SNGT_QHENOMENOLOGY_143: (0)             class ShowPassingFlash(ShowPartial):
SNGT_QHENOMENOLOGY_144: (4)                 def __init__(
SNGT_QHENOMENOLOGY_145: (8)                     self,
SNGT_QHENOMENOLOGY_146: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_147: (8)                     time_width: float = 0.1,
SNGT_QHENOMENOLOGY_148: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_149: (8)                     **kwargs
SNGT_QHENOMENOLOGY_150: (4)                 ):
SNGT_QHENOMENOLOGY_151: (8)                     self.time_width = time_width
SNGT_QHENOMENOLOGY_152: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_153: (12)                        mobject,
SNGT_QHENOMENOLOGY_154: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_155: (12)                        **kwargs
```

```
SNGT_QHENOMENOLOGY_156: (8)                     )
SNGT_QHENOMENOLOGY_157: (4)                 def get_bounds(self, alpha: float) -> tuple[float,
float]:
SNGT_QHENOMENOLOGY_158: (8)                     tw = self.time_width
SNGT_QHENOMENOLOGY_159: (8)                     upper = interpolate(0, 1 + tw, alpha)
SNGT_QHENOMENOLOGY_160: (8)                     lower = upper - tw
SNGT_QHENOMENOLOGY_161: (8)                     upper = min(upper, 1)
SNGT_QHENOMENOLOGY_162: (8)                     lower = max(lower, 0)
SNGT_QHENOMENOLOGY_163: (8)                     return (lower, upper)
SNGT_QHENOMENOLOGY_164: (4)                 def finish(self) -> None:
SNGT_QHENOMENOLOGY_165: (8)                     super().finish()
SNGT_QHENOMENOLOGY_166: (8)                     for submob, start in
self.get_all_families_zipped():
SNGT_QHENOMENOLOGY_167: (12)                         submob.pointwise_become_partial(start, 0, 1)
SNGT_QHENOMENOLOGY_168: (0)             class VShowPassingFlash(Animation):
SNGT_QHENOMENOLOGY_169: (4)                 def __init__(
SNGT_QHENOMENOLOGY_170: (8)                     self,
SNGT_QHENOMENOLOGY_171: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_172: (8)                     time_width: float = 0.3,
SNGT_QHENOMENOLOGY_173: (8)                     taper_width: float = 0.05,
SNGT_QHENOMENOLOGY_174: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_175: (8)                     **kwargs
SNGT_QHENOMENOLOGY_176: (4)                 ):
SNGT_QHENOMENOLOGY_177: (8)                     self.time_width = time_width
SNGT_QHENOMENOLOGY_178: (8)                     self.taper_width = taper_width
SNGT_QHENOMENOLOGY_179: (8)                     super().__init__(vmobject, remover=remover,
**kwargs)
SNGT_QHENOMENOLOGY_180: (8)                     self.mobject = vmobject
SNGT_QHENOMENOLOGY_181: (4)                 def taper_kernel(self, x):
SNGT_QHENOMENOLOGY_182: (8)                     if x < self.taper_width:
SNGT_QHENOMENOLOGY_183: (12)                         return x
SNGT_QHENOMENOLOGY_184: (8)                     elif x > 1 - self.taper_width:
SNGT_QHENOMENOLOGY_185: (12)                         return 1.0 - x
SNGT_QHENOMENOLOGY_186: (8)                     return 1.0
SNGT_QHENOMENOLOGY_187: (4)                 def begin(self) -> None:
SNGT_QHENOMENOLOGY_188: (8)                     self.submob_to_widths = dict()
SNGT_QHENOMENOLOGY_189: (8)                     for sm in self.mobject.get_family():
SNGT_QHENOMENOLOGY_190: (12)                         widths = sm.get_stroke_widths()
SNGT_QHENOMENOLOGY_191: (12)                         self.submob_to_widths[hash(sm)] = np.array([
SNGT_QHENOMENOLOGY_192: (16)                             width * self.taper_kernel(x)
SNGT_QHENOMENOLOGY_193: (16)                             for width, x in zip(widths, np.linspace(0,
1, len(widths)))
SNGT_QHENOMENOLOGY_194: (12)                         ])
SNGT_QHENOMENOLOGY_195: (8)                     super().begin()
SNGT_QHENOMENOLOGY_196: (4)                 def interpolate_submobject(
SNGT_QHENOMENOLOGY_197: (8)                     self,
SNGT_QHENOMENOLOGY_198: (8)                     submobject: VMobject,
SNGT_QHENOMENOLOGY_199: (8)                     starting_sumobject: None,
SNGT_QHENOMENOLOGY_200: (8)                     alpha: float
SNGT_QHENOMENOLOGY_201: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_202: (8)                     widths = self.submob_to_widths[hash(submobject)]
SNGT_QHENOMENOLOGY_203: (8)                     tw = self.time_width
SNGT_QHENOMENOLOGY_204: (8)                     sigma = tw / 6
SNGT_QHENOMENOLOGY_205: (8)                     mu = interpolate(-tw / 2, 1 + tw / 2, alpha)
SNGT_QHENOMENOLOGY_206: (8)                     xs = np.linspace(0, 1, len(widths))
SNGT_QHENOMENOLOGY_207: (8)                     zs = (xs - mu) / sigma
SNGT_QHENOMENOLOGY_208: (8)                     gaussian = np.exp(-0.5 * zs * zs)
SNGT_QHENOMENOLOGY_209: (8)                     gaussian[abs(xs - mu) > 3 * sigma] = 0
SNGT_QHENOMENOLOGY_210: (8)                     if len(widths * gaussian) !=0:
SNGT_QHENOMENOLOGY_211: (12)                         submobject.set_stroke(width=widths * gaussian)
SNGT_QHENOMENOLOGY_212: (4)                 def finish(self) -> None:
SNGT_QHENOMENOLOGY_213: (8)                     super().finish()
SNGT_QHENOMENOLOGY_214: (8)                     for submob, start in
self.get_all_families_zipped():
SNGT_QHENOMENOLOGY_215: (12)                         submob.match_style(start)
SNGT_QHENOMENOLOGY_216: (0)             class FlashAround(VShowPassingFlash):
SNGT_QHENOMENOLOGY_217: (4)                 def __init__(
SNGT_QHENOMENOLOGY_218: (8)                     self,
SNGT_QHENOMENOLOGY_219: (8)                     mobject: Mobject,
```

```
SNGT_QHENOMENOLOGY_220: (8)                     time_width: float = 1.0,
SNGT_QHENOMENOLOGY_221: (8)                     taper_width: float = 0.0,
SNGT_QHENOMENOLOGY_222: (8)                     stroke_width: float = 4.0,
SNGT_QHENOMENOLOGY_223: (8)                     color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_224: (8)                     buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_225: (8)                     n_inserted_curves: int = 100,
SNGT_QHENOMENOLOGY_226: (8)                     **kwargs
SNGT_QHENOMENOLOGY_227: (4)                 ):
SNGT_QHENOMENOLOGY_228: (8)                     path = self.get_path(mobject, buff)
SNGT_QHENOMENOLOGY_229: (8)                     if mobject.is_fixed_in_frame():
SNGT_QHENOMENOLOGY_230: (12)                        path.fix_in_frame()
SNGT_QHENOMENOLOGY_231: (8)                     path.insert_n_curves(n_inserted_curves)
SNGT_QHENOMENOLOGY_232: (8)
path.set_points(path.get_points_without_null_curves())
SNGT_QHENOMENOLOGY_233: (8)                     path.set_stroke(color, stroke_width)
SNGT_QHENOMENOLOGY_234: (8)                     super().__init__(path, time_width=time_width,
taper_width=taper_width, **kwargs)
SNGT_QHENOMENOLOGY_235: (4)                 def get_path(self, mobject: Mobject, buff: float) ->
SurroundingRectangle:
SNGT_QHENOMENOLOGY_236: (8)                     return SurroundingRectangle(mobject, buff=buff)
SNGT_QHENOMENOLOGY_237: (0)             class FlashUnder(FlashAround):
SNGT_QHENOMENOLOGY_238: (4)                 def get_path(self, mobject: Mobject, buff: float) ->
Underline:
SNGT_QHENOMENOLOGY_239: (8)                     return Underline(mobject, buff=buff,
stretch_factor=1.0)
SNGT_QHENOMENOLOGY_240: (0)             class ShowCreationThenDestruction(ShowPassingFlash):
SNGT_QHENOMENOLOGY_241: (4)                 def __init__(self, vmobject: VMobject, time_width:
float = 2.0, **kwargs):
SNGT_QHENOMENOLOGY_242: (8)                     super().__init__(vmobject, time_width=time_width,
**kwargs)
SNGT_QHENOMENOLOGY_243: (0)             class ShowCreationThenFadeOut(Succession):
SNGT_QHENOMENOLOGY_244: (4)                 def __init__(self, mobject: Mobject, remover: bool =
True, **kwargs):
SNGT_QHENOMENOLOGY_245: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_246: (12)                        ShowCreation(mobject),
SNGT_QHENOMENOLOGY_247: (12)                        FadeOut(mobject),
SNGT_QHENOMENOLOGY_248: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_249: (12)                        **kwargs
SNGT_QHENOMENOLOGY_250: (8)                     )
SNGT_QHENOMENOLOGY_251: (0)             class AnimationOnSurroundingRectangle(AnimationGroup):
SNGT_QHENOMENOLOGY_252: (4)                 RectAnimationType: type = Animation
SNGT_QHENOMENOLOGY_253: (4)                 def __init__(
SNGT_QHENOMENOLOGY_254: (8)                     self,
SNGT_QHENOMENOLOGY_255: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_256: (8)                     stroke_width: float = 2.0,
SNGT_QHENOMENOLOGY_257: (8)                     stroke_color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_258: (8)                     buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_259: (8)                     **kwargs
SNGT_QHENOMENOLOGY_260: (4)                 ):
SNGT_QHENOMENOLOGY_261: (8)                     rect = SurroundingRectangle(
SNGT_QHENOMENOLOGY_262: (12)                        mobject,
SNGT_QHENOMENOLOGY_263: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_264: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_265: (12)                        buff=buff,
SNGT_QHENOMENOLOGY_266: (8)                     )
SNGT_QHENOMENOLOGY_267: (8)                     rect.add_updater(lambda r: r.move_to(mobject))
SNGT_QHENOMENOLOGY_268: (8)                     super().__init__(self.RectAnimationType(rect,
**kwargs))
SNGT_QHENOMENOLOGY_269: (0)             class
ShowPassingFlashAround(AnimationOnSurroundingRectangle):
SNGT_QHENOMENOLOGY_270: (4)                 RectAnimationType = ShowPassingFlash
SNGT_QHENOMENOLOGY_271: (0)             class
ShowCreationThenDestructionAround(AnimationOnSurroundingRectangle):
SNGT_QHENOMENOLOGY_272: (4)                 RectAnimationType = ShowCreationThenDestruction
SNGT_QHENOMENOLOGY_273: (0)             class
ShowCreationThenFadeAround(AnimationOnSurroundingRectangle):
SNGT_QHENOMENOLOGY_274: (4)                 RectAnimationType = ShowCreationThenFadeOut
SNGT_QHENOMENOLOGY_275: (0)             class ApplyWave(Homotopy):
SNGT_QHENOMENOLOGY_276: (4)                 def __init__(
```

```
SNGT_QHENOMENOLOGY_277: (8)                     self,
SNGT_QHENOMENOLOGY_278: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_279: (8)                     direction: np.ndarray = UP,
SNGT_QHENOMENOLOGY_280: (8)                     amplitude: float = 0.2,
SNGT_QHENOMENOLOGY_281: (8)                     run_time: float = 1.0,
SNGT_QHENOMENOLOGY_282: (8)                     **kwargs
SNGT_QHENOMENOLOGY_283: (4)                 ):
SNGT_QHENOMENOLOGY_284: (8)                     left_x = mobject.get_left()[0]
SNGT_QHENOMENOLOGY_285: (8)                     right_x = mobject.get_right()[0]
SNGT_QHENOMENOLOGY_286: (8)                     vect = amplitude * direction
SNGT_QHENOMENOLOGY_287: (8)                     def homotopy(x, y, z, t):
SNGT_QHENOMENOLOGY_288: (12)                        alpha = (x - left_x) / (right_x - left_x)
SNGT_QHENOMENOLOGY_289: (12)                        power = np.exp(2.0 * (alpha - 0.5))
SNGT_QHENOMENOLOGY_290: (12)                        nudge = there_and_back(t**power)
SNGT_QHENOMENOLOGY_291: (12)                        return np.array([x, y, z]) + nudge * vect
SNGT_QHENOMENOLOGY_292: (8)                     super().__init__(homotopy, mobject, **kwargs)
SNGT_QHENOMENOLOGY_293: (0)             class WiggleOutThenIn(Animation):
SNGT_QHENOMENOLOGY_294: (4)                 def __init__(
SNGT_QHENOMENOLOGY_295: (8)                     self,
SNGT_QHENOMENOLOGY_296: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_297: (8)                     scale_value: float = 1.1,
SNGT_QHENOMENOLOGY_298: (8)                     rotation_angle: float = 0.01 * TAU,
SNGT_QHENOMENOLOGY_299: (8)                     n_wiggles: int = 6,
SNGT_QHENOMENOLOGY_300: (8)                     scale_about_point: np.ndarray | None = None,
SNGT_QHENOMENOLOGY_301: (8)                     rotate_about_point: np.ndarray | None = None,
SNGT_QHENOMENOLOGY_302: (8)                     run_time: float = 2,
SNGT_QHENOMENOLOGY_303: (8)                     **kwargs
SNGT_QHENOMENOLOGY_304: (4)                 ):
SNGT_QHENOMENOLOGY_305: (8)                     self.scale_value = scale_value
SNGT_QHENOMENOLOGY_306: (8)                     self.rotation_angle = rotation_angle
SNGT_QHENOMENOLOGY_307: (8)                     self.n_wiggles = n_wiggles
SNGT_QHENOMENOLOGY_308: (8)                     self.scale_about_point = scale_about_point
SNGT_QHENOMENOLOGY_309: (8)                     self.rotate_about_point = rotate_about_point
SNGT_QHENOMENOLOGY_310: (8)                     super().__init__(mobject, run_time=run_time,
**kwargs)
SNGT_QHENOMENOLOGY_311: (4)                 def get_scale_about_point(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_312: (8)                     return self.scale_about_point or
self.mobject.get_center()
SNGT_QHENOMENOLOGY_313: (4)                 def get_rotate_about_point(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_314: (8)                     return self.rotate_about_point or
self.mobject.get_center()
SNGT_QHENOMENOLOGY_315: (4)                 def interpolate_submobject(
SNGT_QHENOMENOLOGY_316: (8)                     self,
SNGT_QHENOMENOLOGY_317: (8)                     submobject: Mobject,
SNGT_QHENOMENOLOGY_318: (8)                     starting_sumobject: Mobject,
SNGT_QHENOMENOLOGY_319: (8)                     alpha: float
SNGT_QHENOMENOLOGY_320: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_321: (8)                     submobject.match_points(starting_sumobject)
SNGT_QHENOMENOLOGY_322: (8)                     submobject.scale(
SNGT_QHENOMENOLOGY_323: (12)                        interpolate(1, self.scale_value,
there_and_back(alpha)),
SNGT_QHENOMENOLOGY_324: (12)                        about_point=self.get_scale_about_point()
SNGT_QHENOMENOLOGY_325: (8)                     )
SNGT_QHENOMENOLOGY_326: (8)                     submobject.rotate(
SNGT_QHENOMENOLOGY_327: (12)                        wiggle(alpha, self.n_wiggles) *
self.rotation_angle,
SNGT_QHENOMENOLOGY_328: (12)                        about_point=self.get_rotate_about_point()
SNGT_QHENOMENOLOGY_329: (8)                     )
SNGT_QHENOMENOLOGY_330: (0)             class TurnInsideOut(Transform):
SNGT_QHENOMENOLOGY_331: (4)                 def __init__(self, mobject: Mobject, path_arc: float =
90 * DEG, **kwargs):
SNGT_QHENOMENOLOGY_332: (8)                     super().__init__(mobject, path_arc=path_arc,
**kwargs)
SNGT_QHENOMENOLOGY_333: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_334: (8)                     result = self.mobject.copy().reverse_points()
SNGT_QHENOMENOLOGY_335: (8)                     if isinstance(result, VMobject):
SNGT_QHENOMENOLOGY_336: (12)                        result.refresh_triangulation()
SNGT_QHENOMENOLOGY_337: (8)                     return result
SNGT_QHENOMENOLOGY_338: (0)             class FlashyFadeIn(AnimationGroup):
```

```
SNGT_QHENOMENOLOGY_339: (4)                def __init__(self,
SNGT_QHENOMENOLOGY_340: (8)                    vmobject: VMobject,
SNGT_QHENOMENOLOGY_341: (8)                    stroke_width: float = 2.0,
SNGT_QHENOMENOLOGY_342: (8)                    fade_lag: float = 0.0,
SNGT_QHENOMENOLOGY_343: (8)                    time_width: float = 1.0,
SNGT_QHENOMENOLOGY_344: (8)                    **kwargs
SNGT_QHENOMENOLOGY_345: (4)                ):
SNGT_QHENOMENOLOGY_346: (8)                    outline = vmobject.copy()
SNGT_QHENOMENOLOGY_347: (8)                    outline.set_fill(opacity=0)
SNGT_QHENOMENOLOGY_348: (8)                    outline.set_stroke(width=stroke_width, opacity=1)
SNGT_QHENOMENOLOGY_349: (8)                    rate_func = kwargs.get("rate_func", smooth)
SNGT_QHENOMENOLOGY_350: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_351: (12)                       FadeIn(vmobject,
rate_func=squish_rate_func(rate_func, fade_lag, 1)),
SNGT_QHENOMENOLOGY_352: (12)                       VShowPassingFlash(outline,
time_width=time_width),
SNGT_QHENOMENOLOGY_353: (12)                       **kwargs
SNGT_QHENOMENOLOGY_354: (8)                    )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 15 - composition.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_3: (0)              from manimlib.animation.animation import prepare_animation
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.mobject.mobject import _AnimationBuilder
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.utils.bezier import integer_interpolate
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.utils.iterables import
remove_list_redundancies
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_12: (0)             from typing import TYPE_CHECKING, Union, Iterable
SNGT_QHENOMENOLOGY_13: (0)             AnimationType = Union[Animation, _AnimationBuilder]
SNGT_QHENOMENOLOGY_14: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_15: (4)                 from typing import Callable, Optional
SNGT_QHENOMENOLOGY_16: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_17: (4)                 from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_18: (0)             DEFAULT_LAGGED_START_LAG_RATIO = 0.05
SNGT_QHENOMENOLOGY_19: (0)             class AnimationGroup(Animation):
SNGT_QHENOMENOLOGY_20: (4)                 def __init__(
SNGT_QHENOMENOLOGY_21: (8)                     self,
SNGT_QHENOMENOLOGY_22: (8)                     *args: AnimationType | Iterable[AnimationType],
SNGT_QHENOMENOLOGY_23: (8)                     run_time: float = -1,  # If negative, default to
sum of inputed animation runtimes
SNGT_QHENOMENOLOGY_24: (8)                     lag_ratio: float = 0.0,
SNGT_QHENOMENOLOGY_25: (8)                     group: Optional[Mobject] = None,
SNGT_QHENOMENOLOGY_26: (8)                     group_type: Optional[type] = None,
SNGT_QHENOMENOLOGY_27: (8)                     **kwargs
SNGT_QHENOMENOLOGY_28: (4)                 ):
SNGT_QHENOMENOLOGY_29: (8)                     animations = args[0] if isinstance(args[0],
Iterable) else args
SNGT_QHENOMENOLOGY_30: (8)                     self.animations = [prepare_animation(anim) for anim
in animations]
SNGT_QHENOMENOLOGY_31: (8)                     self.build_animations_with_timings(lag_ratio)
SNGT_QHENOMENOLOGY_32: (8)                     self.max_end_time = max((awt[2] for awt in
self.anims_with_timings), default=0)
SNGT_QHENOMENOLOGY_33: (8)                     self.run_time = self.max_end_time if run_time < 0
else run_time
SNGT_QHENOMENOLOGY_34: (8)                     self.lag_ratio = lag_ratio
SNGT_QHENOMENOLOGY_35: (8)                     mobs = remove_list_redundancies([a.mobject for a in
self.animations])
SNGT_QHENOMENOLOGY_36: (8)                     if group is not None:
SNGT_QHENOMENOLOGY_37: (12)                        self.group = group
```

```
SNGT_QHENOMENOLOGY_38: (8)                      if group_type is not None:
SNGT_QHENOMENOLOGY_39: (12)                         self.group = group_type(*mobs)
SNGT_QHENOMENOLOGY_40: (8)                      elif all(isinstance(anim.mobject, VMobject) for
anim in animations):
SNGT_QHENOMENOLOGY_41: (12)                         self.group = VGroup(*mobs)
SNGT_QHENOMENOLOGY_42: (8)                      else:
SNGT_QHENOMENOLOGY_43: (12)                         self.group = Group(*mobs)
SNGT_QHENOMENOLOGY_44: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_45: (12)                         self.group,
SNGT_QHENOMENOLOGY_46: (12)                         run_time=self.run_time,
SNGT_QHENOMENOLOGY_47: (12)                         lag_ratio=lag_ratio,
SNGT_QHENOMENOLOGY_48: (12)                         **kwargs
SNGT_QHENOMENOLOGY_49: (8)                      )
SNGT_QHENOMENOLOGY_50: (4)                  def get_all_mobjects(self) -> Mobject:
SNGT_QHENOMENOLOGY_51: (8)                      return self.group
SNGT_QHENOMENOLOGY_52: (4)                  def begin(self) -> None:
SNGT_QHENOMENOLOGY_53: (8)                      self.group.set_animating_status(True)
SNGT_QHENOMENOLOGY_54: (8)                      for anim in self.animations:
SNGT_QHENOMENOLOGY_55: (12)                         anim.begin()
SNGT_QHENOMENOLOGY_56: (4)                  def finish(self) -> None:
SNGT_QHENOMENOLOGY_57: (8)                      self.group.set_animating_status(False)
SNGT_QHENOMENOLOGY_58: (8)                      for anim in self.animations:
SNGT_QHENOMENOLOGY_59: (12)                         anim.finish()
SNGT_QHENOMENOLOGY_60: (4)                  def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_61: (8)                      for anim in self.animations:
SNGT_QHENOMENOLOGY_62: (12)                         anim.clean_up_from_scene(scene)
SNGT_QHENOMENOLOGY_63: (4)                  def update_mobjects(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_64: (8)                      for anim in self.animations:
SNGT_QHENOMENOLOGY_65: (12)                         anim.update_mobjects(dt)
SNGT_QHENOMENOLOGY_66: (4)                  def calculate_max_end_time(self) -> None:
SNGT_QHENOMENOLOGY_67: (8)                      self.max_end_time = max(
SNGT_QHENOMENOLOGY_68: (12)                         (awt[2] for awt in self.anims_with_timings),
SNGT_QHENOMENOLOGY_69: (12)                         default=0,
SNGT_QHENOMENOLOGY_70: (8)                      )
SNGT_QHENOMENOLOGY_71: (8)                      if self.run_time < 0:
SNGT_QHENOMENOLOGY_72: (12)                         self.run_time = self.max_end_time
SNGT_QHENOMENOLOGY_73: (4)                  def build_animations_with_timings(self, lag_ratio:
float) -> None:
SNGT_QHENOMENOLOGY_74: (8)                      """
SNGT_QHENOMENOLOGY_75: (8)                      Creates a list of triplets of the form
SNGT_QHENOMENOLOGY_76: (8)                      (anim, start_time, end_time)
SNGT_QHENOMENOLOGY_77: (8)                      """
SNGT_QHENOMENOLOGY_78: (8)                      self.anims_with_timings = []
SNGT_QHENOMENOLOGY_79: (8)                      curr_time = 0
SNGT_QHENOMENOLOGY_80: (8)                      for anim in self.animations:
SNGT_QHENOMENOLOGY_81: (12)                         start_time = curr_time
SNGT_QHENOMENOLOGY_82: (12)                         end_time = start_time + anim.get_run_time()
SNGT_QHENOMENOLOGY_83: (12)                         self.anims_with_timings.append(
SNGT_QHENOMENOLOGY_84: (16)                             (anim, start_time, end_time)
SNGT_QHENOMENOLOGY_85: (12)                         )
SNGT_QHENOMENOLOGY_86: (12)                         curr_time = interpolate(
SNGT_QHENOMENOLOGY_87: (16)                             start_time, end_time, lag_ratio
SNGT_QHENOMENOLOGY_88: (12)                         )
SNGT_QHENOMENOLOGY_89: (4)                  def interpolate(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_90: (8)                      time = alpha * self.max_end_time
SNGT_QHENOMENOLOGY_91: (8)                      for anim, start_time, end_time in
self.anims_with_timings:
SNGT_QHENOMENOLOGY_92: (12)                         anim_time = end_time - start_time
SNGT_QHENOMENOLOGY_93: (12)                         if anim_time == 0:
SNGT_QHENOMENOLOGY_94: (16)                             sub_alpha = 0
SNGT_QHENOMENOLOGY_95: (12)                         else:
SNGT_QHENOMENOLOGY_96: (16)                             sub_alpha = clip((time - start_time) /
anim_time, 0, 1)
SNGT_QHENOMENOLOGY_97: (12)                         anim.interpolate(sub_alpha)
SNGT_QHENOMENOLOGY_98: (0)              class Succession(AnimationGroup):
SNGT_QHENOMENOLOGY_99: (4)                  def __init__(
SNGT_QHENOMENOLOGY_100: (8)                     self,
SNGT_QHENOMENOLOGY_101: (8)                     *animations: Animation,
SNGT_QHENOMENOLOGY_102: (8)                     lag_ratio: float = 1.0,
```

```
SNGT_QHENOMENOLOGY_103: (8)                     **kwargs
SNGT_QHENOMENOLOGY_104: (4)                 ):
SNGT_QHENOMENOLOGY_105: (8)                     super().__init__(*animations, lag_ratio=lag_ratio,
**kwargs)
SNGT_QHENOMENOLOGY_106: (4)                 def begin(self) -> None:
SNGT_QHENOMENOLOGY_107: (8)                     assert len(self.animations) > 0
SNGT_QHENOMENOLOGY_108: (8)                     self.active_animation = self.animations[0]
SNGT_QHENOMENOLOGY_109: (8)                     self.active_animation.begin()
SNGT_QHENOMENOLOGY_110: (4)                 def finish(self) -> None:
SNGT_QHENOMENOLOGY_111: (8)                     self.active_animation.finish()
SNGT_QHENOMENOLOGY_112: (4)                 def update_mobjects(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_113: (8)                     self.active_animation.update_mobjects(dt)
SNGT_QHENOMENOLOGY_114: (4)                 def interpolate(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_115: (8)                     index, subalpha = integer_interpolate(
SNGT_QHENOMENOLOGY_116: (12)                        0, len(self.animations), alpha
SNGT_QHENOMENOLOGY_117: (8)                     )
SNGT_QHENOMENOLOGY_118: (8)                     animation = self.animations[index]
SNGT_QHENOMENOLOGY_119: (8)                     if animation is not self.active_animation:
SNGT_QHENOMENOLOGY_120: (12)                        self.active_animation.finish()
SNGT_QHENOMENOLOGY_121: (12)                        animation.begin()
SNGT_QHENOMENOLOGY_122: (12)                        self.active_animation = animation
SNGT_QHENOMENOLOGY_123: (8)                     animation.interpolate(subalpha)
SNGT_QHENOMENOLOGY_124: (0)             class LaggedStart(AnimationGroup):
SNGT_QHENOMENOLOGY_125: (4)                 def __init__(
SNGT_QHENOMENOLOGY_126: (8)                     self,
SNGT_QHENOMENOLOGY_127: (8)                     *animations,
SNGT_QHENOMENOLOGY_128: (8)                     lag_ratio: float = DEFAULT_LAGGED_START_LAG_RATIO,
SNGT_QHENOMENOLOGY_129: (8)                     **kwargs
SNGT_QHENOMENOLOGY_130: (4)                 ):
SNGT_QHENOMENOLOGY_131: (8)                     super().__init__(*animations, lag_ratio=lag_ratio,
**kwargs)
SNGT_QHENOMENOLOGY_132: (0)             class LaggedStartMap(LaggedStart):
SNGT_QHENOMENOLOGY_133: (4)                 def __init__(
SNGT_QHENOMENOLOGY_134: (8)                     self,
SNGT_QHENOMENOLOGY_135: (8)                     anim_func: Callable[[Mobject], Animation],
SNGT_QHENOMENOLOGY_136: (8)                     group: Mobject,
SNGT_QHENOMENOLOGY_137: (8)                     run_time: float = 2.0,
SNGT_QHENOMENOLOGY_138: (8)                     lag_ratio: float = DEFAULT_LAGGED_START_LAG_RATIO,
SNGT_QHENOMENOLOGY_139: (8)                     **kwargs
SNGT_QHENOMENOLOGY_140: (4)                 ):
SNGT_QHENOMENOLOGY_141: (8)                     anim_kwargs = dict(kwargs)
SNGT_QHENOMENOLOGY_142: (8)                     anim_kwargs.pop("lag_ratio", None)
SNGT_QHENOMENOLOGY_143: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_144: (12)                        *(anim_func(submob, **anim_kwargs) for submob
in group),
SNGT_QHENOMENOLOGY_145: (12)                        run_time=run_time,
SNGT_QHENOMENOLOGY_146: (12)                        lag_ratio=lag_ratio,
SNGT_QHENOMENOLOGY_147: (12)                        group=group
SNGT_QHENOMENOLOGY_148: (8)                     )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 16 - example_scenes.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from manimlib import *
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               class OpeningManimExample(Scene):
SNGT_QHENOMENOLOGY_4: (4)                   def construct(self):
SNGT_QHENOMENOLOGY_5: (8)                       intro_words = Text("""
SNGT_QHENOMENOLOGY_6: (12)                          The original motivation for manim was to
SNGT_QHENOMENOLOGY_7: (12)                          better illustrate mathematical functions
SNGT_QHENOMENOLOGY_8: (12)                          as transformations.
SNGT_QHENOMENOLOGY_9: (8)                       """)
SNGT_QHENOMENOLOGY_10: (8)                      intro_words.to_edge(UP)
SNGT_QHENOMENOLOGY_11: (8)                      self.play(Write(intro_words))
SNGT_QHENOMENOLOGY_12: (8)                      self.wait(2)
SNGT_QHENOMENOLOGY_13: (8)                      grid = NumberPlane((-10, 10), (-5, 5))
SNGT_QHENOMENOLOGY_14: (8)                      matrix = [[1, 1], [0, 1]]
SNGT_QHENOMENOLOGY_15: (8)                      linear_transform_words = VGroup(
```

```
SNGT_QHENOMENOLOGY_16: (12)                         Text("This is what the matrix"),
SNGT_QHENOMENOLOGY_17: (12)                         IntegerMatrix(matrix),
SNGT_QHENOMENOLOGY_18: (12)                         Text("looks like")
SNGT_QHENOMENOLOGY_19: (8)                     )
SNGT_QHENOMENOLOGY_20: (8)                     linear_transform_words.arrange(RIGHT)
SNGT_QHENOMENOLOGY_21: (8)                     linear_transform_words.to_edge(UP)
SNGT_QHENOMENOLOGY_22: (8)                     linear_transform_words.set_backstroke(width=5)
SNGT_QHENOMENOLOGY_23: (8)                     self.play(
SNGT_QHENOMENOLOGY_24: (12)                         ShowCreation(grid),
SNGT_QHENOMENOLOGY_25: (12)                         FadeTransform(intro_words,
linear_transform_words)
SNGT_QHENOMENOLOGY_26: (8)                     )
SNGT_QHENOMENOLOGY_27: (8)                     self.wait()
SNGT_QHENOMENOLOGY_28: (8)                     self.play(grid.animate.apply_matrix(matrix),
run_time=3)
SNGT_QHENOMENOLOGY_29: (8)                     self.wait()
SNGT_QHENOMENOLOGY_30: (8)                     c_grid = ComplexPlane()
SNGT_QHENOMENOLOGY_31: (8)                     moving_c_grid = c_grid.copy()
SNGT_QHENOMENOLOGY_32: (8)                     moving_c_grid.prepare_for_nonlinear_transform()
SNGT_QHENOMENOLOGY_33: (8)                     c_grid.set_stroke(BLUE_E, 1)
SNGT_QHENOMENOLOGY_34: (8)                     c_grid.add_coordinate_labels(font_size=24)
SNGT_QHENOMENOLOGY_35: (8)                     complex_map_words = TexText("""
SNGT_QHENOMENOLOGY_36: (12)                         Or thinking of the plane as $\\mathds{C}$,\\\\
SNGT_QHENOMENOLOGY_37: (12)                         this is the map $z \\rightarrow z^2$
SNGT_QHENOMENOLOGY_38: (8)                     """)
SNGT_QHENOMENOLOGY_39: (8)                     complex_map_words.to_corner(UR)
SNGT_QHENOMENOLOGY_40: (8)                     complex_map_words.set_backstroke(width=5)
SNGT_QHENOMENOLOGY_41: (8)                     self.play(
SNGT_QHENOMENOLOGY_42: (12)                         FadeOut(grid),
SNGT_QHENOMENOLOGY_43: (12)                         Write(c_grid, run_time=3),
SNGT_QHENOMENOLOGY_44: (12)                         FadeIn(moving_c_grid),
SNGT_QHENOMENOLOGY_45: (12)                         FadeTransform(linear_transform_words,
complex_map_words),
SNGT_QHENOMENOLOGY_46: (8)                     )
SNGT_QHENOMENOLOGY_47: (8)                     self.wait()
SNGT_QHENOMENOLOGY_48: (8)                     self.play(
SNGT_QHENOMENOLOGY_49: (12)
moving_c_grid.animate.apply_complex_function(lambda z: z**2),
SNGT_QHENOMENOLOGY_50: (12)                         run_time=6,
SNGT_QHENOMENOLOGY_51: (8)                     )
SNGT_QHENOMENOLOGY_52: (8)                     self.wait(2)
SNGT_QHENOMENOLOGY_53: (0)             class AnimatingMethods(Scene):
SNGT_QHENOMENOLOGY_54: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_55: (8)                     grid = Tex(R"\pi").get_grid(10, 10, height=4)
SNGT_QHENOMENOLOGY_56: (8)                     self.add(grid)
SNGT_QHENOMENOLOGY_57: (8)                     self.play(grid.animate.shift(LEFT))
SNGT_QHENOMENOLOGY_58: (8)                     self.play(grid.animate.set_color(YELLOW))
SNGT_QHENOMENOLOGY_59: (8)                     self.wait()
SNGT_QHENOMENOLOGY_60: (8)
self.play(grid.animate.set_submobject_colors_by_gradient(BLUE, GREEN))
SNGT_QHENOMENOLOGY_61: (8)                     self.wait()
SNGT_QHENOMENOLOGY_62: (8)                     self.play(grid.animate.set_height(TAU -
MED_SMALL_BUFF))
SNGT_QHENOMENOLOGY_63: (8)                     self.wait()
SNGT_QHENOMENOLOGY_64: (8)
self.play(grid.animate.apply_complex_function(np.exp), run_time=5)
SNGT_QHENOMENOLOGY_65: (8)                     self.wait()
SNGT_QHENOMENOLOGY_66: (8)                     self.play(
SNGT_QHENOMENOLOGY_67: (12)                         grid.animate.apply_function(
SNGT_QHENOMENOLOGY_68: (16)                             lambda p: [
SNGT_QHENOMENOLOGY_69: (20)                                 p[0] + 0.5 * math.sin(p[1]),
SNGT_QHENOMENOLOGY_70: (20)                                 p[1] + 0.5 * math.sin(p[0]),
SNGT_QHENOMENOLOGY_71: (20)                                 p[2]
SNGT_QHENOMENOLOGY_72: (16)                             ]
SNGT_QHENOMENOLOGY_73: (12)                         ),
SNGT_QHENOMENOLOGY_74: (12)                         run_time=5,
SNGT_QHENOMENOLOGY_75: (8)                     )
SNGT_QHENOMENOLOGY_76: (8)                     self.wait()
SNGT_QHENOMENOLOGY_77: (0)             class TextExample(Scene):
```

```
SNGT_QHENOMENOLOGY_78: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_79: (8)                     text = Text("Here is a text", font="Consolas",
font_size=90)
SNGT_QHENOMENOLOGY_80: (8)                     difference = Text(
SNGT_QHENOMENOLOGY_81: (12)                        """
SNGT_QHENOMENOLOGY_82: (12)                        The most important difference between Text and
TexText is that\n
SNGT_QHENOMENOLOGY_83: (12)                        you can change the font more easily, but can't
use the LaTeX grammar
SNGT_QHENOMENOLOGY_84: (12)                        """,
SNGT_QHENOMENOLOGY_85: (12)                        font="Arial", font_size=24,
SNGT_QHENOMENOLOGY_86: (12)                        t2c={"Text": BLUE, "TexText": BLUE, "LaTeX":
ORANGE}
SNGT_QHENOMENOLOGY_87: (8)                     )
SNGT_QHENOMENOLOGY_88: (8)                     VGroup(text, difference).arrange(DOWN, buff=1)
SNGT_QHENOMENOLOGY_89: (8)                     self.play(Write(text))
SNGT_QHENOMENOLOGY_90: (8)                     self.play(FadeIn(difference, UP))
SNGT_QHENOMENOLOGY_91: (8)                     self.wait(3)
SNGT_QHENOMENOLOGY_92: (8)                     fonts = Text(
SNGT_QHENOMENOLOGY_93: (12)                        "And you can also set the font according to
different words",
SNGT_QHENOMENOLOGY_94: (12)                        font="Arial",
SNGT_QHENOMENOLOGY_95: (12)                        t2f={"font": "Consolas", "words": "Consolas"},
SNGT_QHENOMENOLOGY_96: (12)                        t2c={"font": BLUE, "words": GREEN}
SNGT_QHENOMENOLOGY_97: (8)                     )
SNGT_QHENOMENOLOGY_98: (8)                     fonts.set_width(FRAME_WIDTH - 1)
SNGT_QHENOMENOLOGY_99: (8)                     slant = Text(
SNGT_QHENOMENOLOGY_100: (12)                       "And the same as slant and weight",
SNGT_QHENOMENOLOGY_101: (12)                       font="Consolas",
SNGT_QHENOMENOLOGY_102: (12)                       t2s={"slant": ITALIC},
SNGT_QHENOMENOLOGY_103: (12)                       t2w={"weight": BOLD},
SNGT_QHENOMENOLOGY_104: (12)                       t2c={"slant": ORANGE, "weight": RED}
SNGT_QHENOMENOLOGY_105: (8)                    )
SNGT_QHENOMENOLOGY_106: (8)                    VGroup(fonts, slant).arrange(DOWN, buff=0.8)
SNGT_QHENOMENOLOGY_107: (8)                    self.play(FadeOut(text), FadeOut(difference,
shift=DOWN))
SNGT_QHENOMENOLOGY_108: (8)                    self.play(Write(fonts))
SNGT_QHENOMENOLOGY_109: (8)                    self.wait()
SNGT_QHENOMENOLOGY_110: (8)                    self.play(Write(slant))
SNGT_QHENOMENOLOGY_111: (8)                    self.wait()
SNGT_QHENOMENOLOGY_112: (0)            class TexTransformExample(Scene):
SNGT_QHENOMENOLOGY_113: (4)                def construct(self):
SNGT_QHENOMENOLOGY_114: (8)                    t2c = {
SNGT_QHENOMENOLOGY_115: (12)                       "A": BLUE,
SNGT_QHENOMENOLOGY_116: (12)                       "B": TEAL,
SNGT_QHENOMENOLOGY_117: (12)                       "C": GREEN,
SNGT_QHENOMENOLOGY_118: (8)                    }
SNGT_QHENOMENOLOGY_119: (8)                    kw = dict(font_size=72, t2c=t2c)
SNGT_QHENOMENOLOGY_120: (8)                    lines = VGroup(
SNGT_QHENOMENOLOGY_121: (12)                       Tex("A^2 + B^2 = C^2", **kw),
SNGT_QHENOMENOLOGY_122: (12)                       Tex("A^2 = C^2 - B^2", **kw),
SNGT_QHENOMENOLOGY_123: (12)                       Tex("A^2 = (C + B)(C - B)", **kw),
SNGT_QHENOMENOLOGY_124: (12)                       Tex(R"A = \sqrt{(C + B)(C - B)}", **kw),
SNGT_QHENOMENOLOGY_125: (8)                    )
SNGT_QHENOMENOLOGY_126: (8)                    lines.arrange(DOWN, buff=LARGE_BUFF)
SNGT_QHENOMENOLOGY_127: (8)                    self.add(lines[0])
SNGT_QHENOMENOLOGY_128: (8)                    self.play(
SNGT_QHENOMENOLOGY_129: (12)                       TransformMatchingStrings(
SNGT_QHENOMENOLOGY_130: (16)                           lines[0].copy(), lines[1],
SNGT_QHENOMENOLOGY_131: (16)                           matched_keys=["A^2", "B^2", "C^2"],
SNGT_QHENOMENOLOGY_132: (16)                           key_map={"+": "-"},
SNGT_QHENOMENOLOGY_133: (16)                           path_arc=90 * DEG,
SNGT_QHENOMENOLOGY_134: (12)                       ),
SNGT_QHENOMENOLOGY_135: (8)                    )
SNGT_QHENOMENOLOGY_136: (8)                    self.wait()
SNGT_QHENOMENOLOGY_137: (8)                    self.play(TransformMatchingStrings(
SNGT_QHENOMENOLOGY_138: (12)                       lines[1].copy(), lines[2],
SNGT_QHENOMENOLOGY_139: (12)                       matched_keys=["A^2"]
SNGT_QHENOMENOLOGY_140: (8)                    ))
```

```
SNGT_QHENOMENOLOGY_141: (8)                     self.wait()
SNGT_QHENOMENOLOGY_142: (8)                     self.play(
SNGT_QHENOMENOLOGY_143: (12)                        TransformMatchingStrings(
SNGT_QHENOMENOLOGY_144: (16)                            lines[2].copy(), lines[3],
SNGT_QHENOMENOLOGY_145: (16)                            key_map={"2": R"\sqrt"},
SNGT_QHENOMENOLOGY_146: (16)                            path_arc=-30 * DEG,
SNGT_QHENOMENOLOGY_147: (12)                        ),
SNGT_QHENOMENOLOGY_148: (8)                     )
SNGT_QHENOMENOLOGY_149: (8)                     self.wait(2)
SNGT_QHENOMENOLOGY_150: (8)                     self.play(LaggedStartMap(FadeOut, lines, shift=2 *
RIGHT))
SNGT_QHENOMENOLOGY_151: (8)                     source = Text("the morse code", height=1)
SNGT_QHENOMENOLOGY_152: (8)                     target = Text("here come dots", height=1)
SNGT_QHENOMENOLOGY_153: (8)                     saved_source = source.copy()
SNGT_QHENOMENOLOGY_154: (8)                     self.play(Write(source))
SNGT_QHENOMENOLOGY_155: (8)                     self.wait()
SNGT_QHENOMENOLOGY_156: (8)                     kw = dict(run_time=3, path_arc=PI / 2)
SNGT_QHENOMENOLOGY_157: (8)                     self.play(TransformMatchingShapes(source, target,
**kw))
SNGT_QHENOMENOLOGY_158: (8)                     self.wait()
SNGT_QHENOMENOLOGY_159: (8)                     self.play(TransformMatchingShapes(target,
saved_source, **kw))
SNGT_QHENOMENOLOGY_160: (8)                     self.wait()
SNGT_QHENOMENOLOGY_161: (0)             class TexIndexing(Scene):
SNGT_QHENOMENOLOGY_162: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_163: (8)                     equation = Tex(R"e^{\pi i} = -1", font_size=144)
SNGT_QHENOMENOLOGY_164: (8)                     self.add(equation)
SNGT_QHENOMENOLOGY_165: (8)                     self.play(FlashAround(equation["e"]))
SNGT_QHENOMENOLOGY_166: (8)                     self.wait()
SNGT_QHENOMENOLOGY_167: (8)                     self.play(Indicate(equation[R"\pi"]))
SNGT_QHENOMENOLOGY_168: (8)                     self.wait()
SNGT_QHENOMENOLOGY_169: (8)                     self.play(TransformFromCopy(
SNGT_QHENOMENOLOGY_170: (12)                        equation[R"e^{\pi i}"].copy().set_opacity(0.5),
SNGT_QHENOMENOLOGY_171: (12)                        equation["-1"],
SNGT_QHENOMENOLOGY_172: (12)                        path_arc=-PI / 2,
SNGT_QHENOMENOLOGY_173: (12)                        run_time=3
SNGT_QHENOMENOLOGY_174: (8)                     ))
SNGT_QHENOMENOLOGY_175: (8)                     self.play(FadeOut(equation))
SNGT_QHENOMENOLOGY_176: (8)                     equation = Tex("A^2 + B^2 = C^2", font_size=144)
SNGT_QHENOMENOLOGY_177: (8)                     self.play(Write(equation))
SNGT_QHENOMENOLOGY_178: (8)                     for part in equation[re.compile(r"\w\^2")]:
SNGT_QHENOMENOLOGY_179: (12)                        self.play(FlashAround(part))
SNGT_QHENOMENOLOGY_180: (8)                     self.wait()
SNGT_QHENOMENOLOGY_181: (8)                     self.play(FadeOut(equation))
SNGT_QHENOMENOLOGY_182: (8)                     equation = Tex(R"\sum_{n = 1}^\infty \frac{1}{n^2}
= \frac{\pi^2}{6}", font_size=72)
SNGT_QHENOMENOLOGY_183: (8)                     self.play(FadeIn(equation))
SNGT_QHENOMENOLOGY_184: (8)
self.play(equation[R"\infty"].animate.set_color(RED))  # Doesn't hit the infinity
SNGT_QHENOMENOLOGY_185: (8)                     self.wait()
SNGT_QHENOMENOLOGY_186: (8)                     self.play(FadeOut(equation))
SNGT_QHENOMENOLOGY_187: (8)                     equation = Tex(
SNGT_QHENOMENOLOGY_188: (12)                        R"\sum_{n = 1}^\infty {1 \over n^2} = {\pi^2
\over 6}",
SNGT_QHENOMENOLOGY_189: (12)                        isolate=[R"\infty"],
SNGT_QHENOMENOLOGY_190: (12)                        font_size=72
SNGT_QHENOMENOLOGY_191: (8)                     )
SNGT_QHENOMENOLOGY_192: (8)                     self.play(FadeIn(equation))
SNGT_QHENOMENOLOGY_193: (8)
self.play(equation[R"\infty"].animate.set_color(RED))  # Got it!
SNGT_QHENOMENOLOGY_194: (8)                     self.wait()
SNGT_QHENOMENOLOGY_195: (8)                     self.play(FadeOut(equation))
SNGT_QHENOMENOLOGY_196: (0)             class UpdatersExample(Scene):
SNGT_QHENOMENOLOGY_197: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_198: (8)                     square = Square()
SNGT_QHENOMENOLOGY_199: (8)                     square.set_fill(BLUE_E, 1)
SNGT_QHENOMENOLOGY_200: (8)                     brace = always_redraw(Brace, square, UP)
SNGT_QHENOMENOLOGY_201: (8)                     label = TexText("Width = 0.00")
SNGT_QHENOMENOLOGY_202: (8)                     number = label.make_number_changeable("0.00")
```

```
SNGT_QHENOMENOLOGY_203: (8)                     label.always.next_to(brace, UP)
SNGT_QHENOMENOLOGY_204: (8)                     number.f_always.set_value(square.get_width)
SNGT_QHENOMENOLOGY_205: (8)                     self.add(square, brace, label)
SNGT_QHENOMENOLOGY_206: (8)                     self.play(
SNGT_QHENOMENOLOGY_207: (12)                        square.animate.scale(2),
SNGT_QHENOMENOLOGY_208: (12)                        rate_func=there_and_back,
SNGT_QHENOMENOLOGY_209: (12)                        run_time=2,
SNGT_QHENOMENOLOGY_210: (8)                     )
SNGT_QHENOMENOLOGY_211: (8)                     self.wait()
SNGT_QHENOMENOLOGY_212: (8)                     self.play(
SNGT_QHENOMENOLOGY_213: (12)                        square.animate.set_width(5, stretch=True),
SNGT_QHENOMENOLOGY_214: (12)                        run_time=3,
SNGT_QHENOMENOLOGY_215: (8)                     )
SNGT_QHENOMENOLOGY_216: (8)                     self.wait()
SNGT_QHENOMENOLOGY_217: (8)                     self.play(
SNGT_QHENOMENOLOGY_218: (12)                        square.animate.set_width(2),
SNGT_QHENOMENOLOGY_219: (12)                        run_time=3
SNGT_QHENOMENOLOGY_220: (8)                     )
SNGT_QHENOMENOLOGY_221: (8)                     self.wait()
SNGT_QHENOMENOLOGY_222: (8)                     now = self.time
SNGT_QHENOMENOLOGY_223: (8)                     w0 = square.get_width()
SNGT_QHENOMENOLOGY_224: (8)                     square.add_updater(
SNGT_QHENOMENOLOGY_225: (12)                        lambda m: m.set_width(w0 * math.sin(self.time -
now) + w0)
SNGT_QHENOMENOLOGY_226: (8)                     )
SNGT_QHENOMENOLOGY_227: (8)                     self.wait(4 * PI)
SNGT_QHENOMENOLOGY_228: (0)             class CoordinateSystemExample(Scene):
SNGT_QHENOMENOLOGY_229: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_230: (8)                     axes = Axes(
SNGT_QHENOMENOLOGY_231: (12)                        x_range=(-1, 10),
SNGT_QHENOMENOLOGY_232: (12)                        y_range=(-2, 2, 0.5),
SNGT_QHENOMENOLOGY_233: (12)                        height=6,
SNGT_QHENOMENOLOGY_234: (12)                        width=10,
SNGT_QHENOMENOLOGY_235: (12)                        axis_config=dict(
SNGT_QHENOMENOLOGY_236: (16)                            stroke_color=GREY_A,
SNGT_QHENOMENOLOGY_237: (16)                            stroke_width=2,
SNGT_QHENOMENOLOGY_238: (16)                            numbers_to_exclude=[0],
SNGT_QHENOMENOLOGY_239: (12)                        ),
SNGT_QHENOMENOLOGY_240: (12)                        y_axis_config=dict(
SNGT_QHENOMENOLOGY_241: (16)                            big_tick_numbers=[-2, 2],
SNGT_QHENOMENOLOGY_242: (12)                        )
SNGT_QHENOMENOLOGY_243: (8)                     )
SNGT_QHENOMENOLOGY_244: (8)                     axes.add_coordinate_labels(
SNGT_QHENOMENOLOGY_245: (12)                        font_size=20,
SNGT_QHENOMENOLOGY_246: (12)                        num_decimal_places=1,
SNGT_QHENOMENOLOGY_247: (8)                     )
SNGT_QHENOMENOLOGY_248: (8)                     self.add(axes)
SNGT_QHENOMENOLOGY_249: (8)                     dot = Dot(color=RED)
SNGT_QHENOMENOLOGY_250: (8)                     dot.move_to(axes.c2p(0, 0))
SNGT_QHENOMENOLOGY_251: (8)                     self.play(FadeIn(dot, scale=0.5))
SNGT_QHENOMENOLOGY_252: (8)                     self.play(dot.animate.move_to(axes.c2p(3, 2)))
SNGT_QHENOMENOLOGY_253: (8)                     self.wait()
SNGT_QHENOMENOLOGY_254: (8)                     self.play(dot.animate.move_to(axes.c2p(5, 0.5)))
SNGT_QHENOMENOLOGY_255: (8)                     self.wait()
SNGT_QHENOMENOLOGY_256: (8)                     h_line = always_redraw(lambda:
axes.get_h_line(dot.get_left()))
SNGT_QHENOMENOLOGY_257: (8)                     v_line = always_redraw(lambda:
axes.get_v_line(dot.get_bottom()))
SNGT_QHENOMENOLOGY_258: (8)                     self.play(
SNGT_QHENOMENOLOGY_259: (12)                        ShowCreation(h_line),
SNGT_QHENOMENOLOGY_260: (12)                        ShowCreation(v_line),
SNGT_QHENOMENOLOGY_261: (8)                     )
SNGT_QHENOMENOLOGY_262: (8)                     self.play(dot.animate.move_to(axes.c2p(3, -2)))
SNGT_QHENOMENOLOGY_263: (8)                     self.wait()
SNGT_QHENOMENOLOGY_264: (8)                     self.play(dot.animate.move_to(axes.c2p(1, 1)))
SNGT_QHENOMENOLOGY_265: (8)                     self.wait()
SNGT_QHENOMENOLOGY_266: (8)                     f_always(dot.move_to, lambda: axes.c2p(1, 1))
SNGT_QHENOMENOLOGY_267: (8)                     self.play(
SNGT_QHENOMENOLOGY_268: (12)                        axes.animate.scale(0.75).to_corner(UL),
```

```
SNGT_QHENOMENOLOGY_269: (12)                        run_time=2,
SNGT_QHENOMENOLOGY_270: (8)                     )
SNGT_QHENOMENOLOGY_271: (8)                     self.wait()
SNGT_QHENOMENOLOGY_272: (8)                     self.play(FadeOut(VGroup(axes, dot, h_line,
v_line)))
SNGT_QHENOMENOLOGY_273: (0)             class GraphExample(Scene):
SNGT_QHENOMENOLOGY_274: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_275: (8)                     axes = Axes((-3, 10), (-1, 8), height=6)
SNGT_QHENOMENOLOGY_276: (8)                     axes.add_coordinate_labels()
SNGT_QHENOMENOLOGY_277: (8)                     self.play(Write(axes, lag_ratio=0.01, run_time=1))
SNGT_QHENOMENOLOGY_278: (8)                     sin_graph = axes.get_graph(
SNGT_QHENOMENOLOGY_279: (12)                        lambda x: 2 * math.sin(x),
SNGT_QHENOMENOLOGY_280: (12)                        color=BLUE,
SNGT_QHENOMENOLOGY_281: (8)                     )
SNGT_QHENOMENOLOGY_282: (8)                     relu_graph = axes.get_graph(
SNGT_QHENOMENOLOGY_283: (12)                        lambda x: max(x, 0),
SNGT_QHENOMENOLOGY_284: (12)                        use_smoothing=False,
SNGT_QHENOMENOLOGY_285: (12)                        color=YELLOW,
SNGT_QHENOMENOLOGY_286: (8)                     )
SNGT_QHENOMENOLOGY_287: (8)                     step_graph = axes.get_graph(
SNGT_QHENOMENOLOGY_288: (12)                        lambda x: 2.0 if x > 3 else 1.0,
SNGT_QHENOMENOLOGY_289: (12)                        discontinuities=[3],
SNGT_QHENOMENOLOGY_290: (12)                        color=GREEN,
SNGT_QHENOMENOLOGY_291: (8)                     )
SNGT_QHENOMENOLOGY_292: (8)                     sin_label = axes.get_graph_label(sin_graph,
"\\sin(x)")
SNGT_QHENOMENOLOGY_293: (8)                     relu_label = axes.get_graph_label(relu_graph,
Text("ReLU"))
SNGT_QHENOMENOLOGY_294: (8)                     step_label = axes.get_graph_label(step_graph,
Text("Step"), x=4)
SNGT_QHENOMENOLOGY_295: (8)                     self.play(
SNGT_QHENOMENOLOGY_296: (12)                        ShowCreation(sin_graph),
SNGT_QHENOMENOLOGY_297: (12)                        FadeIn(sin_label, RIGHT),
SNGT_QHENOMENOLOGY_298: (8)                     )
SNGT_QHENOMENOLOGY_299: (8)                     self.wait(2)
SNGT_QHENOMENOLOGY_300: (8)                     self.play(
SNGT_QHENOMENOLOGY_301: (12)                        ReplacementTransform(sin_graph, relu_graph),
SNGT_QHENOMENOLOGY_302: (12)                        FadeTransform(sin_label, relu_label),
SNGT_QHENOMENOLOGY_303: (8)                     )
SNGT_QHENOMENOLOGY_304: (8)                     self.wait()
SNGT_QHENOMENOLOGY_305: (8)                     self.play(
SNGT_QHENOMENOLOGY_306: (12)                        ReplacementTransform(relu_graph, step_graph),
SNGT_QHENOMENOLOGY_307: (12)                        FadeTransform(relu_label, step_label),
SNGT_QHENOMENOLOGY_308: (8)                     )
SNGT_QHENOMENOLOGY_309: (8)                     self.wait()
SNGT_QHENOMENOLOGY_310: (8)                     parabola = axes.get_graph(lambda x: 0.25 * x**2)
SNGT_QHENOMENOLOGY_311: (8)                     parabola.set_stroke(BLUE)
SNGT_QHENOMENOLOGY_312: (8)                     self.play(
SNGT_QHENOMENOLOGY_313: (12)                        FadeOut(step_graph),
SNGT_QHENOMENOLOGY_314: (12)                        FadeOut(step_label),
SNGT_QHENOMENOLOGY_315: (12)                        ShowCreation(parabola)
SNGT_QHENOMENOLOGY_316: (8)                     )
SNGT_QHENOMENOLOGY_317: (8)                     self.wait()
SNGT_QHENOMENOLOGY_318: (8)                     dot = Dot(color=RED)
SNGT_QHENOMENOLOGY_319: (8)                     dot.move_to(axes.i2gp(2, parabola))
SNGT_QHENOMENOLOGY_320: (8)                     self.play(FadeIn(dot, scale=0.5))
SNGT_QHENOMENOLOGY_321: (8)                     x_tracker = ValueTracker(2)
SNGT_QHENOMENOLOGY_322: (8)                     dot.add_updater(lambda d:
d.move_to(axes.i2gp(x_tracker.get_value(), parabola)))
SNGT_QHENOMENOLOGY_323: (8)                     self.play(x_tracker.animate.set_value(4),
run_time=3)
SNGT_QHENOMENOLOGY_324: (8)                     self.play(x_tracker.animate.set_value(-2),
run_time=3)
SNGT_QHENOMENOLOGY_325: (8)                     self.wait()
SNGT_QHENOMENOLOGY_326: (0)             class TexAndNumbersExample(Scene):
SNGT_QHENOMENOLOGY_327: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_328: (8)                     axes = Axes((-3, 3), (-3, 3), unit_size=1)
SNGT_QHENOMENOLOGY_329: (8)                     axes.to_edge(DOWN)
SNGT_QHENOMENOLOGY_330: (8)                     axes.add_coordinate_labels(font_size=16)
```

```
SNGT_QHENOMENOLOGY_331: (8)                     circle = Circle(radius=2)
SNGT_QHENOMENOLOGY_332: (8)                     circle.set_stroke(YELLOW, 3)
SNGT_QHENOMENOLOGY_333: (8)                     circle.move_to(axes.get_origin())
SNGT_QHENOMENOLOGY_334: (8)                     self.add(axes, circle)
SNGT_QHENOMENOLOGY_335: (8)                     tex = Tex("x^2 + y^2 = 4.00")
SNGT_QHENOMENOLOGY_336: (8)                     tex.next_to(axes, UP, buff=0.5)
SNGT_QHENOMENOLOGY_337: (8)                     value = tex.make_number_changeable("4.00")
SNGT_QHENOMENOLOGY_338: (8)                     value.add_updater(lambda v:
v.set_value(circle.get_radius()**2))
SNGT_QHENOMENOLOGY_339: (8)                     self.add(tex)
SNGT_QHENOMENOLOGY_340: (8)                     text = Text("""
SNGT_QHENOMENOLOGY_341: (12)                        You can manipulate numbers
SNGT_QHENOMENOLOGY_342: (12)                        in Tex mobjects
SNGT_QHENOMENOLOGY_343: (8)                     """, font_size=30)
SNGT_QHENOMENOLOGY_344: (8)                     text.next_to(tex, RIGHT, buff=1.5)
SNGT_QHENOMENOLOGY_345: (8)                     arrow = Arrow(text, tex)
SNGT_QHENOMENOLOGY_346: (8)                     self.add(text, arrow)
SNGT_QHENOMENOLOGY_347: (8)                     self.play(
SNGT_QHENOMENOLOGY_348: (12)                        circle.animate.set_height(2.0),
SNGT_QHENOMENOLOGY_349: (12)                        run_time=4,
SNGT_QHENOMENOLOGY_350: (12)                        rate_func=there_and_back,
SNGT_QHENOMENOLOGY_351: (8)                     )
SNGT_QHENOMENOLOGY_352: (8)                     exponents = tex.make_number_changeable("2",
replace_all=True)
SNGT_QHENOMENOLOGY_353: (8)                     self.play(
SNGT_QHENOMENOLOGY_354: (12)                        LaggedStartMap(
SNGT_QHENOMENOLOGY_355: (16)                            FlashAround, exponents,
SNGT_QHENOMENOLOGY_356: (16)                            lag_ratio=0.2, buff=0.1, color=RED
SNGT_QHENOMENOLOGY_357: (12)                        ),
SNGT_QHENOMENOLOGY_358: (12)                        exponents.animate.set_color(RED)
SNGT_QHENOMENOLOGY_359: (8)                     )
SNGT_QHENOMENOLOGY_360: (8)                     def func(x, y):
SNGT_QHENOMENOLOGY_361: (12)                        xa, ya = axes.point_to_coords(np.array([x, y,
0]))
SNGT_QHENOMENOLOGY_362: (12)                        return xa**4 + ya**4 - 4
SNGT_QHENOMENOLOGY_363: (8)                     new_curve = ImplicitFunction(func)
SNGT_QHENOMENOLOGY_364: (8)                     new_curve.match_style(circle)
SNGT_QHENOMENOLOGY_365: (8)
circle.rotate(angle_of_vector(new_curve.get_start()))  # Align
SNGT_QHENOMENOLOGY_366: (8)                     value.clear_updaters()
SNGT_QHENOMENOLOGY_367: (8)                     self.play(
SNGT_QHENOMENOLOGY_368: (12)                        *(ChangeDecimalToValue(exp, 4) for exp in
exponents),
SNGT_QHENOMENOLOGY_369: (12)                        ReplacementTransform(circle.copy(), new_curve),
SNGT_QHENOMENOLOGY_370: (12)                        circle.animate.set_stroke(width=1,
opacity=0.5),
SNGT_QHENOMENOLOGY_371: (8)                     )
SNGT_QHENOMENOLOGY_372: (0)             class SurfaceExample(ThreeDScene):
SNGT_QHENOMENOLOGY_373: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_374: (8)                     surface_text = Text("For 3d scenes, try using
surfaces")
SNGT_QHENOMENOLOGY_375: (8)                     surface_text.fix_in_frame()
SNGT_QHENOMENOLOGY_376: (8)                     surface_text.to_edge(UP)
SNGT_QHENOMENOLOGY_377: (8)                     self.add(surface_text)
SNGT_QHENOMENOLOGY_378: (8)                     self.wait(0.1)
SNGT_QHENOMENOLOGY_379: (8)                     torus1 = Torus(r1=1, r2=1)
SNGT_QHENOMENOLOGY_380: (8)                     torus2 = Torus(r1=3, r2=1)
SNGT_QHENOMENOLOGY_381: (8)                     sphere = Sphere(radius=3,
resolution=torus1.resolution)
SNGT_QHENOMENOLOGY_382: (8)                     day_texture =
"https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/Whole_world_-
_land_and_oceans.jpg/1280px-Whole_world_-_land_and_oceans.jpg"
SNGT_QHENOMENOLOGY_383: (8)                     night_texture =
"https://upload.wikimedia.org/wikipedia/commons/thumb/b/ba/The_earth_at_night.jpg/1280px-
The_earth_at_night.jpg"
SNGT_QHENOMENOLOGY_384: (8)                     surfaces = [
SNGT_QHENOMENOLOGY_385: (12)                        TexturedSurface(surface, day_texture,
night_texture)
SNGT_QHENOMENOLOGY_386: (12)                        for surface in [sphere, torus1, torus2]
```

```
SNGT_QHENOMENOLOGY_387: (8)                     ]
SNGT_QHENOMENOLOGY_388: (8)                     for mob in surfaces:
SNGT_QHENOMENOLOGY_389: (12)                        mob.shift(IN)
SNGT_QHENOMENOLOGY_390: (12)                        mob.mesh = SurfaceMesh(mob)
SNGT_QHENOMENOLOGY_391: (12)                        mob.mesh.set_stroke(BLUE, 1, opacity=0.5)
SNGT_QHENOMENOLOGY_392: (8)                     surface = surfaces[0]
SNGT_QHENOMENOLOGY_393: (8)                     self.play(
SNGT_QHENOMENOLOGY_394: (12)                        FadeIn(surface),
SNGT_QHENOMENOLOGY_395: (12)                        ShowCreation(surface.mesh, lag_ratio=0.01,
run_time=3),
SNGT_QHENOMENOLOGY_396: (8)                     )
SNGT_QHENOMENOLOGY_397: (8)                     for mob in surfaces:
SNGT_QHENOMENOLOGY_398: (12)                        mob.add(mob.mesh)
SNGT_QHENOMENOLOGY_399: (8)                     surface.save_state()
SNGT_QHENOMENOLOGY_400: (8)                     self.play(Rotate(surface, PI / 2), run_time=2)
SNGT_QHENOMENOLOGY_401: (8)                     for mob in surfaces[1:]:
SNGT_QHENOMENOLOGY_402: (12)                        mob.rotate(PI / 2)
SNGT_QHENOMENOLOGY_403: (8)                     self.play(
SNGT_QHENOMENOLOGY_404: (12)                        Transform(surface, surfaces[1]),
SNGT_QHENOMENOLOGY_405: (12)                        run_time=3
SNGT_QHENOMENOLOGY_406: (8)                     )
SNGT_QHENOMENOLOGY_407: (8)                     self.play(
SNGT_QHENOMENOLOGY_408: (12)                        Transform(surface, surfaces[2]),
SNGT_QHENOMENOLOGY_409: (12)                        self.frame.animate.increment_phi(-10 * DEG),
SNGT_QHENOMENOLOGY_410: (12)                        self.frame.animate.increment_theta(-20 * DEG),
SNGT_QHENOMENOLOGY_411: (12)                        run_time=3
SNGT_QHENOMENOLOGY_412: (8)                     )
SNGT_QHENOMENOLOGY_413: (8)                     self.frame.add_updater(lambda m, dt:
m.increment_theta(-0.1 * dt))
SNGT_QHENOMENOLOGY_414: (8)                     light_text = Text("You can move around the light
source")
SNGT_QHENOMENOLOGY_415: (8)                     light_text.move_to(surface_text)
SNGT_QHENOMENOLOGY_416: (8)                     light_text.fix_in_frame()
SNGT_QHENOMENOLOGY_417: (8)                     self.play(FadeTransform(surface_text, light_text))
SNGT_QHENOMENOLOGY_418: (8)                     light = self.camera.light_source
SNGT_QHENOMENOLOGY_419: (8)                     light_dot = GlowDot(color=WHITE, radius=0.5)
SNGT_QHENOMENOLOGY_420: (8)                     light_dot.always.move_to(light)
SNGT_QHENOMENOLOGY_421: (8)                     self.add(light, light_dot)
SNGT_QHENOMENOLOGY_422: (8)                     light.save_state()
SNGT_QHENOMENOLOGY_423: (8)                     self.play(light.animate.move_to(3 * IN),
run_time=5)
SNGT_QHENOMENOLOGY_424: (8)                     self.play(light.animate.shift(10 * OUT),
run_time=5)
SNGT_QHENOMENOLOGY_425: (8)                     drag_text = Text("Try moving the mouse while
pressing d or f")
SNGT_QHENOMENOLOGY_426: (8)                     drag_text.move_to(light_text)
SNGT_QHENOMENOLOGY_427: (8)                     drag_text.fix_in_frame()
SNGT_QHENOMENOLOGY_428: (8)                     self.play(FadeTransform(light_text, drag_text))
SNGT_QHENOMENOLOGY_429: (8)                     self.wait()
SNGT_QHENOMENOLOGY_430: (0)             class InteractiveDevelopment(Scene):
SNGT_QHENOMENOLOGY_431: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_432: (8)                     circle = Circle()
SNGT_QHENOMENOLOGY_433: (8)                     circle.set_fill(BLUE, opacity=0.5)
SNGT_QHENOMENOLOGY_434: (8)                     circle.set_stroke(BLUE_E, width=4)
SNGT_QHENOMENOLOGY_435: (8)                     square = Square()
SNGT_QHENOMENOLOGY_436: (8)                     self.play(ShowCreation(square))
SNGT_QHENOMENOLOGY_437: (8)                     self.wait()
SNGT_QHENOMENOLOGY_438: (8)                     self.embed()
SNGT_QHENOMENOLOGY_439: (8)                     self.play(ReplacementTransform(square, circle))
SNGT_QHENOMENOLOGY_440: (8)                     self.wait()
SNGT_QHENOMENOLOGY_441: (8)                     self.play(circle.animate.stretch(4, 0))
SNGT_QHENOMENOLOGY_442: (8)                     self.play(Rotate(circle, 90 * DEG))
SNGT_QHENOMENOLOGY_443: (8)                     self.play(circle.animate.shift(2 *
RIGHT).scale(0.25))
SNGT_QHENOMENOLOGY_444: (8)                     text = Text("""
SNGT_QHENOMENOLOGY_445: (12)                        In general, using the interactive shell
SNGT_QHENOMENOLOGY_446: (12)                        is very helpful when developing new scenes
SNGT_QHENOMENOLOGY_447: (8)                     """)
SNGT_QHENOMENOLOGY_448: (8)                     self.play(Write(text))
```

```
SNGT_QHENOMENOLOGY_449: (8)                     always(circle.move_to, self.mouse_point)
SNGT_QHENOMENOLOGY_450: (0)             class ControlsExample(Scene):
SNGT_QHENOMENOLOGY_451: (4)                 drag_to_pan = False
SNGT_QHENOMENOLOGY_452: (4)                 def setup(self):
SNGT_QHENOMENOLOGY_453: (8)                     self.textbox = Textbox()
SNGT_QHENOMENOLOGY_454: (8)                     self.checkbox = Checkbox()
SNGT_QHENOMENOLOGY_455: (8)                     self.color_picker = ColorSliders()
SNGT_QHENOMENOLOGY_456: (8)                     self.panel = ControlPanel(
SNGT_QHENOMENOLOGY_457: (12)                        Text("Text", font_size=24), self.textbox,
Line(),
SNGT_QHENOMENOLOGY_458: (12)                        Text("Show/Hide Text", font_size=24),
self.checkbox, Line(),
SNGT_QHENOMENOLOGY_459: (12)                        Text("Color of Text", font_size=24),
self.color_picker
SNGT_QHENOMENOLOGY_460: (8)                     )
SNGT_QHENOMENOLOGY_461: (8)                     self.add(self.panel)
SNGT_QHENOMENOLOGY_462: (4)                 def construct(self):
SNGT_QHENOMENOLOGY_463: (8)                     text = Text("text", font_size=96)
SNGT_QHENOMENOLOGY_464: (8)                     def text_updater(old_text):
SNGT_QHENOMENOLOGY_465: (12)                        assert(isinstance(old_text, Text))
SNGT_QHENOMENOLOGY_466: (12)                        new_text = Text(self.textbox.get_value(),
font_size=old_text.font_size)
SNGT_QHENOMENOLOGY_467: (12)                        new_text.move_to(old_text)
SNGT_QHENOMENOLOGY_468: (12)                        if self.checkbox.get_value():
SNGT_QHENOMENOLOGY_469: (16)                            new_text.set_fill(
SNGT_QHENOMENOLOGY_470: (20)
color=self.color_picker.get_picked_color(),
SNGT_QHENOMENOLOGY_471: (20)
opacity=self.color_picker.get_picked_opacity()
SNGT_QHENOMENOLOGY_472: (16)                            )
SNGT_QHENOMENOLOGY_473: (12)                        else:
SNGT_QHENOMENOLOGY_474: (16)                            new_text.set_opacity(0)
SNGT_QHENOMENOLOGY_475: (12)                        old_text.become(new_text)
SNGT_QHENOMENOLOGY_476: (8)                     text.add_updater(text_updater)
SNGT_QHENOMENOLOGY_477: (8)                     self.add(MotionMobject(text))
SNGT_QHENOMENOLOGY_478: (8)                     self.textbox.set_value("Manim")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 17 - manim_example_ext.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from docutils import nodes
SNGT_QHENOMENOLOGY_2: (0)               from docutils.parsers.rst import directives, Directive
SNGT_QHENOMENOLOGY_3: (0)               import jinja2
SNGT_QHENOMENOLOGY_4: (0)               import os
SNGT_QHENOMENOLOGY_5: (0)               class skip_manim_node(nodes.Admonition, nodes.Element):
SNGT_QHENOMENOLOGY_6: (4)                   pass
SNGT_QHENOMENOLOGY_7: (0)               def visit(self, node, name=""):
SNGT_QHENOMENOLOGY_8: (4)                   self.visit_admonition(node, name)
SNGT_QHENOMENOLOGY_9: (0)               def depart(self, node):
SNGT_QHENOMENOLOGY_10: (4)                  self.depart_admonition(node)
SNGT_QHENOMENOLOGY_11: (0)              class ManimExampleDirective(Directive):
SNGT_QHENOMENOLOGY_12: (4)                  has_content = True
SNGT_QHENOMENOLOGY_13: (4)                  required_arguments = 1
SNGT_QHENOMENOLOGY_14: (4)                  optional_arguments = 0
SNGT_QHENOMENOLOGY_15: (4)                  option_spec = {
SNGT_QHENOMENOLOGY_16: (8)                      "hide_code": bool,
SNGT_QHENOMENOLOGY_17: (8)                      "media": str,
SNGT_QHENOMENOLOGY_18: (4)                  }
SNGT_QHENOMENOLOGY_19: (4)                  final_argument_whitespace = True
SNGT_QHENOMENOLOGY_20: (4)                  def run(self):
SNGT_QHENOMENOLOGY_21: (8)                      hide_code = "hide_code" in self.options
SNGT_QHENOMENOLOGY_22: (8)                      scene_name = self.arguments[0]
SNGT_QHENOMENOLOGY_23: (8)                      media_file_name = self.options["media"]
SNGT_QHENOMENOLOGY_24: (8)                      source_block = [
SNGT_QHENOMENOLOGY_25: (12)                         ".. code-block:: python",
SNGT_QHENOMENOLOGY_26: (12)                         "",
SNGT_QHENOMENOLOGY_27: (12)                         *["    " + line for line in self.content],
SNGT_QHENOMENOLOGY_28: (8)                      ]
```

```
SNGT_QHENOMENOLOGY_29: (8)                    source_block = "\n".join(source_block)
SNGT_QHENOMENOLOGY_30: (8)                    state_machine = self.state_machine
SNGT_QHENOMENOLOGY_31: (8)                    document = state_machine.document
SNGT_QHENOMENOLOGY_32: (8)                    if any(media_file_name.endswith(ext) for ext in
[".png", ".jpg", ".gif"]):
SNGT_QHENOMENOLOGY_33: (12)                       is_video = False
SNGT_QHENOMENOLOGY_34: (8)                    else:
SNGT_QHENOMENOLOGY_35: (12)                       is_video = True
SNGT_QHENOMENOLOGY_36: (8)                    rendered_template =
jinja2.Template(TEMPLATE).render(
SNGT_QHENOMENOLOGY_37: (12)                       scene_name=scene_name,
SNGT_QHENOMENOLOGY_38: (12)                       scene_name_lowercase=scene_name.lower(),
SNGT_QHENOMENOLOGY_39: (12)                       hide_code=hide_code,
SNGT_QHENOMENOLOGY_40: (12)                       is_video=is_video,
SNGT_QHENOMENOLOGY_41: (12)                       media_file_name=media_file_name,
SNGT_QHENOMENOLOGY_42: (12)                       source_block=source_block,
SNGT_QHENOMENOLOGY_43: (8)                    )
SNGT_QHENOMENOLOGY_44: (8)                    state_machine.insert_input(
SNGT_QHENOMENOLOGY_45: (12)                       rendered_template.split("\n"),
source=document.attributes["source"]
SNGT_QHENOMENOLOGY_46: (8)                    )
SNGT_QHENOMENOLOGY_47: (8)                    return []
SNGT_QHENOMENOLOGY_48: (0)            def setup(app):
SNGT_QHENOMENOLOGY_49: (4)                app.add_node(skip_manim_node, html=(visit, depart))
SNGT_QHENOMENOLOGY_50: (4)                setup.app = app
SNGT_QHENOMENOLOGY_51: (4)                setup.config = app.config
SNGT_QHENOMENOLOGY_52: (4)                setup.confdir = app.confdir
SNGT_QHENOMENOLOGY_53: (4)                app.add_directive("manim-example",
ManimExampleDirective)
SNGT_QHENOMENOLOGY_54: (4)                metadata = {"parallel_read_safe": False,
"parallel_write_safe": True}
SNGT_QHENOMENOLOGY_55: (4)                return metadata
SNGT_QHENOMENOLOGY_56: (0)            TEMPLATE = r"""
SNGT_QHENOMENOLOGY_57: (0)            {% if not hide_code %}
SNGT_QHENOMENOLOGY_58: (0)            .. raw:: html
SNGT_QHENOMENOLOGY_59: (4)                <div class="manim-example">
SNGT_QHENOMENOLOGY_60: (0)            {% endif %}
SNGT_QHENOMENOLOGY_61: (0)            {% if is_video %}
SNGT_QHENOMENOLOGY_62: (0)            .. raw:: html
SNGT_QHENOMENOLOGY_63: (4)                <video id="{{ scene_name_lowercase }}" class="manim-
video" controls loop autoplay src="{{ media_file_name }}"></video>
SNGT_QHENOMENOLOGY_64: (0)            {% else %}
SNGT_QHENOMENOLOGY_65: (0)            .. image:: {{ media_file_name }}
SNGT_QHENOMENOLOGY_66: (4)                :align: center
SNGT_QHENOMENOLOGY_67: (4)                :name: {{ scene_name_lowercase }}
SNGT_QHENOMENOLOGY_68: (0)            {% endif %}
SNGT_QHENOMENOLOGY_69: (0)            {% if not hide_code %}
SNGT_QHENOMENOLOGY_70: (0)            .. raw:: html
SNGT_QHENOMENOLOGY_71: (4)                <h5 class="example-header">{{ scene_name }}<a
class="headerlink" href="#{{ scene_name_lowercase }}">¶</a></h5>
SNGT_QHENOMENOLOGY_72: (0)            {{ source_block }}
SNGT_QHENOMENOLOGY_73: (0)            {% endif %}
SNGT_QHENOMENOLOGY_74: (0)            .. raw:: html
SNGT_QHENOMENOLOGY_75: (4)                </div>
SNGT_QHENOMENOLOGY_76: (0)            """
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 18 - frame.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from manimlib.constants import BLACK, GREY_E
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.constants import FRAME_HEIGHT
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_5: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_6: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_7: (4)                 from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_8: (0)             class ScreenRectangle(Rectangle):
SNGT_QHENOMENOLOGY_9: (4)                 def __init__(
```

```
SNGT_QHENOMENOLOGY_10: (8)                      self,
SNGT_QHENOMENOLOGY_11: (8)                      aspect_ratio: float = 16.0 / 9.0,
SNGT_QHENOMENOLOGY_12: (8)                      height: float = 4,
SNGT_QHENOMENOLOGY_13: (8)                      **kwargs
SNGT_QHENOMENOLOGY_14: (4)                  ):
SNGT_QHENOMENOLOGY_15: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_16: (12)                         width=aspect_ratio * height,
SNGT_QHENOMENOLOGY_17: (12)                         height=height,
SNGT_QHENOMENOLOGY_18: (12)                         **kwargs
SNGT_QHENOMENOLOGY_19: (8)                      )
SNGT_QHENOMENOLOGY_20: (0)              class FullScreenRectangle(ScreenRectangle):
SNGT_QHENOMENOLOGY_21: (4)                  def __init__(
SNGT_QHENOMENOLOGY_22: (8)                      self,
SNGT_QHENOMENOLOGY_23: (8)                      height: float = FRAME_HEIGHT,
SNGT_QHENOMENOLOGY_24: (8)                      fill_color: ManimColor = GREY_E,
SNGT_QHENOMENOLOGY_25: (8)                      fill_opacity: float = 1,
SNGT_QHENOMENOLOGY_26: (8)                      stroke_width: float = 0,
SNGT_QHENOMENOLOGY_27: (8)                      **kwargs,
SNGT_QHENOMENOLOGY_28: (4)                  ):
SNGT_QHENOMENOLOGY_29: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_30: (12)                         height=height,
SNGT_QHENOMENOLOGY_31: (12)                         fill_color=fill_color,
SNGT_QHENOMENOLOGY_32: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_33: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_34: (12)                         **kwargs
SNGT_QHENOMENOLOGY_35: (8)                      )
SNGT_QHENOMENOLOGY_36: (0)              class FullScreenFadeRectangle(FullScreenRectangle):
SNGT_QHENOMENOLOGY_37: (4)                  def __init__(
SNGT_QHENOMENOLOGY_38: (8)                      self,
SNGT_QHENOMENOLOGY_39: (8)                      stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_40: (8)                      fill_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_41: (8)                      fill_opacity: float = 0.7,
SNGT_QHENOMENOLOGY_42: (8)                      **kwargs,
SNGT_QHENOMENOLOGY_43: (4)                  ):
SNGT_QHENOMENOLOGY_44: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_45: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_46: (12)                         fill_color=fill_color,
SNGT_QHENOMENOLOGY_47: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_48: (8)                      )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 19 - config.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import argparse
SNGT_QHENOMENOLOGY_3: (0)               import colour
SNGT_QHENOMENOLOGY_4: (0)               import importlib
SNGT_QHENOMENOLOGY_5: (0)               import inspect
SNGT_QHENOMENOLOGY_6: (0)               import os
SNGT_QHENOMENOLOGY_7: (0)               import sys
SNGT_QHENOMENOLOGY_8: (0)               import yaml
SNGT_QHENOMENOLOGY_9: (0)               from pathlib import Path
SNGT_QHENOMENOLOGY_10: (0)              from ast import literal_eval
SNGT_QHENOMENOLOGY_11: (0)              from addict import Dict
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.logger import log
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.dict_ops import merge_dicts_recursively
SNGT_QHENOMENOLOGY_14: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_15: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_16: (4)                  from argparse import Namespace
SNGT_QHENOMENOLOGY_17: (4)                  from typing import Optional
SNGT_QHENOMENOLOGY_18: (0)              def initialize_manim_config() -> Dict:
SNGT_QHENOMENOLOGY_19: (4)                  """
SNGT_QHENOMENOLOGY_20: (4)                  Return default configuration for various classes in
manim, such as
SNGT_QHENOMENOLOGY_21: (4)                  Scene, Window, Camera, and SceneFileWriter, as well as
configuration
SNGT_QHENOMENOLOGY_22: (4)                  determining how the scene is run (e.g. written to file
or previewed in window).
```

```
SNGT_QHENOMENOLOGY_23: (4)                 The result is initially on the contents of
default_config.yml in the manimlib directory,
SNGT_QHENOMENOLOGY_24: (4)                 which can be further updated by a custom configuration
file custom_config.yml.
SNGT_QHENOMENOLOGY_25: (4)                 It is further updated based on command line argument.
SNGT_QHENOMENOLOGY_26: (4)                 """
SNGT_QHENOMENOLOGY_27: (4)                 args = parse_cli()
SNGT_QHENOMENOLOGY_28: (4)                 global_defaults_file = os.path.join(get_manim_dir(),
"manimlib", "default_config.yml")
SNGT_QHENOMENOLOGY_29: (4)                 config = Dict(merge_dicts_recursively(
SNGT_QHENOMENOLOGY_30: (8)                     load_yaml(global_defaults_file),
SNGT_QHENOMENOLOGY_31: (8)                     load_yaml("custom_config.yml"),  # From current
working directory
SNGT_QHENOMENOLOGY_32: (8)                     load_yaml(args.config_file) if args.config_file
else dict(),
SNGT_QHENOMENOLOGY_33: (4)                 ))
SNGT_QHENOMENOLOGY_34: (4)                 log.setLevel(args.log_level or config["log_level"])
SNGT_QHENOMENOLOGY_35: (4)                 update_directory_config(config)
SNGT_QHENOMENOLOGY_36: (4)                 update_window_config(config, args)
SNGT_QHENOMENOLOGY_37: (4)                 update_camera_config(config, args)
SNGT_QHENOMENOLOGY_38: (4)                 update_file_writer_config(config, args)
SNGT_QHENOMENOLOGY_39: (4)                 update_scene_config(config, args)
SNGT_QHENOMENOLOGY_40: (4)                 update_run_config(config, args)
SNGT_QHENOMENOLOGY_41: (4)                 update_embed_config(config, args)
SNGT_QHENOMENOLOGY_42: (4)                 return config
SNGT_QHENOMENOLOGY_43: (0)             def parse_cli():
SNGT_QHENOMENOLOGY_44: (4)                 try:
SNGT_QHENOMENOLOGY_45: (8)                     parser = argparse.ArgumentParser()
SNGT_QHENOMENOLOGY_46: (8)                     module_location =
parser.add_mutually_exclusive_group()
SNGT_QHENOMENOLOGY_47: (8)                     module_location.add_argument(
SNGT_QHENOMENOLOGY_48: (12)                        "file",
SNGT_QHENOMENOLOGY_49: (12)                        nargs="?",
SNGT_QHENOMENOLOGY_50: (12)                        help="Path to file holding the python code for
the scene",
SNGT_QHENOMENOLOGY_51: (8)                     )
SNGT_QHENOMENOLOGY_52: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_53: (12)                        "scene_names",
SNGT_QHENOMENOLOGY_54: (12)                        nargs="*",
SNGT_QHENOMENOLOGY_55: (12)                        help="Name of the Scene class you want to see",
SNGT_QHENOMENOLOGY_56: (8)                     )
SNGT_QHENOMENOLOGY_57: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_58: (12)                        "-w", "--write_file",
SNGT_QHENOMENOLOGY_59: (12)                        action="store_true",
SNGT_QHENOMENOLOGY_60: (12)                        help="Render the scene as a movie file",
SNGT_QHENOMENOLOGY_61: (8)                     )
SNGT_QHENOMENOLOGY_62: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_63: (12)                        "-s", "--skip_animations",
SNGT_QHENOMENOLOGY_64: (12)                        action="store_true",
SNGT_QHENOMENOLOGY_65: (12)                        help="Save the last frame",
SNGT_QHENOMENOLOGY_66: (8)                     )
SNGT_QHENOMENOLOGY_67: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_68: (12)                        "-l", "--low_quality",
SNGT_QHENOMENOLOGY_69: (12)                        action="store_true",
SNGT_QHENOMENOLOGY_70: (12)                        help="Render at 480p",
SNGT_QHENOMENOLOGY_71: (8)                     )
SNGT_QHENOMENOLOGY_72: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_73: (12)                        "-m", "--medium_quality",
SNGT_QHENOMENOLOGY_74: (12)                        action="store_true",
SNGT_QHENOMENOLOGY_75: (12)                        help="Render at 720p",
SNGT_QHENOMENOLOGY_76: (8)                     )
SNGT_QHENOMENOLOGY_77: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_78: (12)                        "--hd",
SNGT_QHENOMENOLOGY_79: (12)                        action="store_true",
SNGT_QHENOMENOLOGY_80: (12)                        help="Render at a 1080p",
SNGT_QHENOMENOLOGY_81: (8)                     )
SNGT_QHENOMENOLOGY_82: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_83: (12)                        "--uhd",
SNGT_QHENOMENOLOGY_84: (12)                        action="store_true",
```

```
SNGT_QHENOMENOLOGY_85: (12)                     help="Render at a 4k",
SNGT_QHENOMENOLOGY_86: (8)                  )
SNGT_QHENOMENOLOGY_87: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_88: (12)                     "-f", "--full_screen",
SNGT_QHENOMENOLOGY_89: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_90: (12)                     help="Show window in full screen",
SNGT_QHENOMENOLOGY_91: (8)                  )
SNGT_QHENOMENOLOGY_92: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_93: (12)                     "-p", "--presenter_mode",
SNGT_QHENOMENOLOGY_94: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_95: (12)                     help="Scene will stay paused during wait calls
until " + \
SNGT_QHENOMENOLOGY_96: (17)                          "space bar or right arrow is hit, like a
slide show"
SNGT_QHENOMENOLOGY_97: (8)                  )
SNGT_QHENOMENOLOGY_98: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_99: (12)                     "-i", "--gif",
SNGT_QHENOMENOLOGY_100: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_101: (12)                    help="Save the video as gif",
SNGT_QHENOMENOLOGY_102: (8)                 )
SNGT_QHENOMENOLOGY_103: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_104: (12)                    "-t", "--transparent",
SNGT_QHENOMENOLOGY_105: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_106: (12)                    help="Render to a movie file with an alpha
channel",
SNGT_QHENOMENOLOGY_107: (8)                 )
SNGT_QHENOMENOLOGY_108: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_109: (12)                    "--vcodec",
SNGT_QHENOMENOLOGY_110: (12)                    help="Video codec to use with ffmpeg",
SNGT_QHENOMENOLOGY_111: (8)                 )
SNGT_QHENOMENOLOGY_112: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_113: (12)                    "--pix_fmt",
SNGT_QHENOMENOLOGY_114: (12)                    help="Pixel format to use for the output of
ffmpeg, defaults to `yuv420p`",
SNGT_QHENOMENOLOGY_115: (8)                 )
SNGT_QHENOMENOLOGY_116: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_117: (12)                    "-q", "--quiet",
SNGT_QHENOMENOLOGY_118: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_119: (12)                    help="",
SNGT_QHENOMENOLOGY_120: (8)                 )
SNGT_QHENOMENOLOGY_121: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_122: (12)                    "-a", "--write_all",
SNGT_QHENOMENOLOGY_123: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_124: (12)                    help="Write all the scenes from a file",
SNGT_QHENOMENOLOGY_125: (8)                 )
SNGT_QHENOMENOLOGY_126: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_127: (12)                    "-o", "--open",
SNGT_QHENOMENOLOGY_128: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_129: (12)                    help="Automatically open the saved file once
its done",
SNGT_QHENOMENOLOGY_130: (8)                 )
SNGT_QHENOMENOLOGY_131: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_132: (12)                    "--finder",
SNGT_QHENOMENOLOGY_133: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_134: (12)                    help="Show the output file in finder",
SNGT_QHENOMENOLOGY_135: (8)                 )
SNGT_QHENOMENOLOGY_136: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_137: (12)                    "--subdivide",
SNGT_QHENOMENOLOGY_138: (12)                    action="store_true",
SNGT_QHENOMENOLOGY_139: (12)                    help="Divide the output animation into
individual movie files " +
SNGT_QHENOMENOLOGY_140: (17)                         "for each animation",
SNGT_QHENOMENOLOGY_141: (8)                 )
SNGT_QHENOMENOLOGY_142: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_143: (12)                    "--file_name",
SNGT_QHENOMENOLOGY_144: (12)                    help="Name for the movie or image file",
SNGT_QHENOMENOLOGY_145: (8)                 )
SNGT_QHENOMENOLOGY_146: (8)                 parser.add_argument(
SNGT_QHENOMENOLOGY_147: (12)                    "-n", "--start_at_animation_number",
```

```
SNGT_QHENOMENOLOGY_148: (12)                     help="Start rendering not from the first
animation, but " + \
SNGT_QHENOMENOLOGY_149: (17)                          "from another, specified by its index.  If
you pass " + \
SNGT_QHENOMENOLOGY_150: (17)                          "in two comma separated values, e.g.
\"3,6\", it will end " + \
SNGT_QHENOMENOLOGY_151: (17)                          "the rendering at the second value",
SNGT_QHENOMENOLOGY_152: (8)                  )
SNGT_QHENOMENOLOGY_153: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_154: (12)                     "-e", "--embed",
SNGT_QHENOMENOLOGY_155: (12)                     metavar="LINE_NUMBER",
SNGT_QHENOMENOLOGY_156: (12)                     help="Adds a breakpoint at the inputted file
dropping into an " + \
SNGT_QHENOMENOLOGY_157: (17)                          "interactive iPython session at that point
of the code."
SNGT_QHENOMENOLOGY_158: (8)                  )
SNGT_QHENOMENOLOGY_159: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_160: (12)                     "-r", "--resolution",
SNGT_QHENOMENOLOGY_161: (12)                     help="Resolution, passed as \"WxH\", e.g.
\"1920x1080\"",
SNGT_QHENOMENOLOGY_162: (8)                  )
SNGT_QHENOMENOLOGY_163: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_164: (12)                     "--fps",
SNGT_QHENOMENOLOGY_165: (12)                     help="Frame rate, as an integer",
SNGT_QHENOMENOLOGY_166: (8)                  )
SNGT_QHENOMENOLOGY_167: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_168: (12)                     "-c", "--color",
SNGT_QHENOMENOLOGY_169: (12)                     help="Background color",
SNGT_QHENOMENOLOGY_170: (8)                  )
SNGT_QHENOMENOLOGY_171: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_172: (12)                     "--leave_progress_bars",
SNGT_QHENOMENOLOGY_173: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_174: (12)                     help="Leave progress bars displayed in
terminal",
SNGT_QHENOMENOLOGY_175: (8)                  )
SNGT_QHENOMENOLOGY_176: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_177: (12)                     "--show_animation_progress",
SNGT_QHENOMENOLOGY_178: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_179: (12)                     help="Show progress bar for each animation",
SNGT_QHENOMENOLOGY_180: (8)                  )
SNGT_QHENOMENOLOGY_181: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_182: (12)                     "--prerun",
SNGT_QHENOMENOLOGY_183: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_184: (12)                     help="Calculate total framecount, to display in
a progress bar, by doing " + \
SNGT_QHENOMENOLOGY_185: (17)                          "an initial run of the scene which skips
animations."
SNGT_QHENOMENOLOGY_186: (8)                  )
SNGT_QHENOMENOLOGY_187: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_188: (12)                     "--video_dir",
SNGT_QHENOMENOLOGY_189: (12)                     help="Directory to write video",
SNGT_QHENOMENOLOGY_190: (8)                  )
SNGT_QHENOMENOLOGY_191: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_192: (12)                     "--config_file",
SNGT_QHENOMENOLOGY_193: (12)                     help="Path to the custom configuration file",
SNGT_QHENOMENOLOGY_194: (8)                  )
SNGT_QHENOMENOLOGY_195: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_196: (12)                     "-v", "--version",
SNGT_QHENOMENOLOGY_197: (12)                     action="store_true",
SNGT_QHENOMENOLOGY_198: (12)                     help="Display the version of manimgl"
SNGT_QHENOMENOLOGY_199: (8)                  )
SNGT_QHENOMENOLOGY_200: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_201: (12)                     "--log-level",
SNGT_QHENOMENOLOGY_202: (12)                     help="Level of messages to Display, can be
DEBUG / INFO / WARNING / ERROR / CRITICAL"
SNGT_QHENOMENOLOGY_203: (8)                  )
SNGT_QHENOMENOLOGY_204: (8)                  parser.add_argument(
SNGT_QHENOMENOLOGY_205: (12)                     "--clear-cache",
SNGT_QHENOMENOLOGY_206: (12)                     action="store_true",
```

```
SNGT_QHENOMENOLOGY_207: (12)                         help="Erase the cache used for Tex and Text
Mobjects"
SNGT_QHENOMENOLOGY_208: (8)                     )
SNGT_QHENOMENOLOGY_209: (8)                     parser.add_argument(
SNGT_QHENOMENOLOGY_210: (12)                         "--autoreload",
SNGT_QHENOMENOLOGY_211: (12)                         action="store_true",
SNGT_QHENOMENOLOGY_212: (12)                         help="Automatically reload Python modules to
pick up code changes " +
SNGT_QHENOMENOLOGY_213: (17)                              "across different files",
SNGT_QHENOMENOLOGY_214: (8)                     )
SNGT_QHENOMENOLOGY_215: (8)                     args = parser.parse_args()
SNGT_QHENOMENOLOGY_216: (8)                     args.write_file = any([args.write_file, args.open,
args.finder])
SNGT_QHENOMENOLOGY_217: (8)                     return args
SNGT_QHENOMENOLOGY_218: (4)                 except argparse.ArgumentError as err:
SNGT_QHENOMENOLOGY_219: (8)                     log.error(str(err))
SNGT_QHENOMENOLOGY_220: (8)                     sys.exit(2)
SNGT_QHENOMENOLOGY_221: (0)             def update_directory_config(config: Dict):
SNGT_QHENOMENOLOGY_222: (4)                 dir_config = config.directories
SNGT_QHENOMENOLOGY_223: (4)                 base = dir_config.base
SNGT_QHENOMENOLOGY_224: (4)                 for key, subdir in dir_config.subdirs.items():
SNGT_QHENOMENOLOGY_225: (8)                     dir_config[key] = os.path.join(base, subdir)
SNGT_QHENOMENOLOGY_226: (0)             def update_window_config(config: Dict, args: Namespace):
SNGT_QHENOMENOLOGY_227: (4)                 window_config = config.window
SNGT_QHENOMENOLOGY_228: (4)                 for key in "position", "size":
SNGT_QHENOMENOLOGY_229: (8)                     if window_config.get(key):
SNGT_QHENOMENOLOGY_230: (12)                         window_config[key] =
literal_eval(window_config[key])
SNGT_QHENOMENOLOGY_231: (4)                 if args.full_screen:
SNGT_QHENOMENOLOGY_232: (8)                     window_config.full_screen = True
SNGT_QHENOMENOLOGY_233: (0)             def update_camera_config(config: Dict, args: Namespace):
SNGT_QHENOMENOLOGY_234: (4)                 camera_config = config.camera
SNGT_QHENOMENOLOGY_235: (4)                 arg_resolution = get_resolution_from_args(args,
config.resolution_options)
SNGT_QHENOMENOLOGY_236: (4)                 camera_config.resolution = arg_resolution or
literal_eval(camera_config.resolution)
SNGT_QHENOMENOLOGY_237: (4)                 if args.fps:
SNGT_QHENOMENOLOGY_238: (8)                     camera_config.fps = args.fps
SNGT_QHENOMENOLOGY_239: (4)                 if args.color:
SNGT_QHENOMENOLOGY_240: (8)                     try:
SNGT_QHENOMENOLOGY_241: (12)                         camera_config.background_color =
colour.Color(args.color)
SNGT_QHENOMENOLOGY_242: (8)                     except Exception:
SNGT_QHENOMENOLOGY_243: (12)                         log.error("Please use a valid color")
SNGT_QHENOMENOLOGY_244: (12)                         log.error(err)
SNGT_QHENOMENOLOGY_245: (12)                         sys.exit(2)
SNGT_QHENOMENOLOGY_246: (4)                 if args.transparent:
SNGT_QHENOMENOLOGY_247: (8)                     camera_config.background_opacity = 0.0
SNGT_QHENOMENOLOGY_248: (0)             def update_file_writer_config(config: Dict, args:
Namespace):
SNGT_QHENOMENOLOGY_249: (4)                 file_writer_config = config.file_writer
SNGT_QHENOMENOLOGY_250: (4)                 file_writer_config.update(
SNGT_QHENOMENOLOGY_251: (8)                     write_to_movie=(not args.skip_animations and
args.write_file),
SNGT_QHENOMENOLOGY_252: (8)                     subdivide_output=args.subdivide,
SNGT_QHENOMENOLOGY_253: (8)                     save_last_frame=(args.skip_animations and
args.write_file),
SNGT_QHENOMENOLOGY_254: (8)                     png_mode=("RGBA" if args.transparent else "RGB"),
SNGT_QHENOMENOLOGY_255: (8)                     movie_file_extension=(get_file_ext(args)),
SNGT_QHENOMENOLOGY_256: (8)                     output_directory=get_output_directory(args,
config),
SNGT_QHENOMENOLOGY_257: (8)                     file_name=args.file_name,
SNGT_QHENOMENOLOGY_258: (8)                     open_file_upon_completion=args.open,
SNGT_QHENOMENOLOGY_259: (8)                     show_file_location_upon_completion=args.finder,
SNGT_QHENOMENOLOGY_260: (8)                     quiet=args.quiet,
SNGT_QHENOMENOLOGY_261: (4)                 )
SNGT_QHENOMENOLOGY_262: (4)                 if args.vcodec:
SNGT_QHENOMENOLOGY_263: (8)                     file_writer_config.video_codec = args.vcodec
SNGT_QHENOMENOLOGY_264: (4)                 elif args.transparent:
```

```
SNGT_QHENOMENOLOGY_265: (8)                     file_writer_config.video_codec = 'prores_ks'
SNGT_QHENOMENOLOGY_266: (8)                     file_writer_config.pixel_format = ''
SNGT_QHENOMENOLOGY_267: (4)                 elif args.gif:
SNGT_QHENOMENOLOGY_268: (8)                     file_writer_config.video_codec = ''
SNGT_QHENOMENOLOGY_269: (4)                 if args.pix_fmt:
SNGT_QHENOMENOLOGY_270: (8)                     file_writer_config.pixel_format = args.pix_fmt
SNGT_QHENOMENOLOGY_271: (0)             def update_scene_config(config: Dict, args: Namespace):
SNGT_QHENOMENOLOGY_272: (4)                 scene_config = config.scene
SNGT_QHENOMENOLOGY_273: (4)                 start, end = get_animations_numbers(args)
SNGT_QHENOMENOLOGY_274: (4)                 scene_config.update(
SNGT_QHENOMENOLOGY_275: (8)                     camera_config=dict(),
SNGT_QHENOMENOLOGY_276: (8)                     file_writer_config=dict(),
SNGT_QHENOMENOLOGY_277: (8)                     skip_animations=args.skip_animations,
SNGT_QHENOMENOLOGY_278: (8)                     start_at_animation_number=start,
SNGT_QHENOMENOLOGY_279: (8)                     end_at_animation_number=end,
SNGT_QHENOMENOLOGY_280: (8)                     presenter_mode=args.presenter_mode,
SNGT_QHENOMENOLOGY_281: (4)                 )
SNGT_QHENOMENOLOGY_282: (4)                 if args.leave_progress_bars:
SNGT_QHENOMENOLOGY_283: (8)                     scene_config.leave_progress_bars = True
SNGT_QHENOMENOLOGY_284: (4)                 if args.show_animation_progress:
SNGT_QHENOMENOLOGY_285: (8)                     scene_config.show_animation_progress = True
SNGT_QHENOMENOLOGY_286: (0)             def update_run_config(config: Dict, args: Namespace):
SNGT_QHENOMENOLOGY_287: (4)                 config.run = Dict(
SNGT_QHENOMENOLOGY_288: (8)                     file_name=args.file,
SNGT_QHENOMENOLOGY_289: (8)                     embed_line=(int(args.embed) if args.embed is not
None else None),
SNGT_QHENOMENOLOGY_290: (8)                     is_reload=False,
SNGT_QHENOMENOLOGY_291: (8)                     prerun=args.prerun,
SNGT_QHENOMENOLOGY_292: (8)                     scene_names=args.scene_names,
SNGT_QHENOMENOLOGY_293: (8)                     quiet=args.quiet or args.write_all,
SNGT_QHENOMENOLOGY_294: (8)                     write_all=args.write_all,
SNGT_QHENOMENOLOGY_295: (8)                     show_in_window=not args.write_file
SNGT_QHENOMENOLOGY_296: (4)                 )
SNGT_QHENOMENOLOGY_297: (0)             def update_embed_config(config: Dict, args: Namespace):
SNGT_QHENOMENOLOGY_298: (4)                 if args.autoreload:
SNGT_QHENOMENOLOGY_299: (8)                     config.embed.autoreload = True
SNGT_QHENOMENOLOGY_300: (0)             def load_yaml(file_path: str):
SNGT_QHENOMENOLOGY_301: (4)                 try:
SNGT_QHENOMENOLOGY_302: (8)                     with open(file_path, "r") as file:
SNGT_QHENOMENOLOGY_303: (12)                        return yaml.safe_load(file) or {}
SNGT_QHENOMENOLOGY_304: (4)                 except FileNotFoundError:
SNGT_QHENOMENOLOGY_305: (8)                     return {}
SNGT_QHENOMENOLOGY_306: (0)             def get_manim_dir():
SNGT_QHENOMENOLOGY_307: (4)                 manimlib_module = importlib.import_module("manimlib")
SNGT_QHENOMENOLOGY_308: (4)                 manimlib_dir =
os.path.dirname(inspect.getabsfile(manimlib_module))
SNGT_QHENOMENOLOGY_309: (4)                 return os.path.abspath(os.path.join(manimlib_dir,
".."))
SNGT_QHENOMENOLOGY_310: (0)             def get_resolution_from_args(args: Optional[Namespace],
resolution_options: dict) -> Optional[tuple[int, int]]:
SNGT_QHENOMENOLOGY_311: (4)                 if args.resolution:
SNGT_QHENOMENOLOGY_312: (8)                     return tuple(map(int, args.resolution.split("x")))
SNGT_QHENOMENOLOGY_313: (4)                 if args.low_quality:
SNGT_QHENOMENOLOGY_314: (8)                     return literal_eval(resolution_options["low"])
SNGT_QHENOMENOLOGY_315: (4)                 if args.medium_quality:
SNGT_QHENOMENOLOGY_316: (8)                     return literal_eval(resolution_options["med"])
SNGT_QHENOMENOLOGY_317: (4)                 if args.hd:
SNGT_QHENOMENOLOGY_318: (8)                     return literal_eval(resolution_options["high"])
SNGT_QHENOMENOLOGY_319: (4)                 if args.uhd:
SNGT_QHENOMENOLOGY_320: (8)                     return literal_eval(resolution_options["4k"])
SNGT_QHENOMENOLOGY_321: (4)                 return None
SNGT_QHENOMENOLOGY_322: (0)             def get_file_ext(args: Namespace) -> str:
SNGT_QHENOMENOLOGY_323: (4)                 if args.transparent:
SNGT_QHENOMENOLOGY_324: (8)                     file_ext = ".mov"
SNGT_QHENOMENOLOGY_325: (4)                 elif args.gif:
SNGT_QHENOMENOLOGY_326: (8)                     file_ext = ".gif"
SNGT_QHENOMENOLOGY_327: (4)                 else:
SNGT_QHENOMENOLOGY_328: (8)                     file_ext = ".mp4"
SNGT_QHENOMENOLOGY_329: (4)                 return file_ext
```

```
SNGT_QHENOMENOLOGY_330: (0)             def get_animations_numbers(args: Namespace) -> tuple[int |
None, int | None]:
SNGT_QHENOMENOLOGY_331: (4)                 stan = args.start_at_animation_number
SNGT_QHENOMENOLOGY_332: (4)                 if stan is None:
SNGT_QHENOMENOLOGY_333: (8)                     return (None, None)
SNGT_QHENOMENOLOGY_334: (4)                 elif "," in stan:
SNGT_QHENOMENOLOGY_335: (8)                     return tuple(map(int, stan.split(",")))
SNGT_QHENOMENOLOGY_336: (4)                 else:
SNGT_QHENOMENOLOGY_337: (8)                     return int(stan), None
SNGT_QHENOMENOLOGY_338: (0)             def get_output_directory(args: Namespace, config: Dict) ->
str:
SNGT_QHENOMENOLOGY_339: (4)                 dir_config = config.directories
SNGT_QHENOMENOLOGY_340: (4)                 out_dir = args.video_dir or dir_config.output
SNGT_QHENOMENOLOGY_341: (4)                 if dir_config.mirror_module_path and args.file:
SNGT_QHENOMENOLOGY_342: (8)                     file_path = Path(args.file).absolute()
SNGT_QHENOMENOLOGY_343: (8)                     rel_path =
file_path.relative_to(dir_config.removed_mirror_prefix)
SNGT_QHENOMENOLOGY_344: (8)                     rel_path = Path(str(rel_path).lstrip("_"))
SNGT_QHENOMENOLOGY_345: (8)                     out_dir = Path(out_dir, rel_path).with_suffix("")
SNGT_QHENOMENOLOGY_346: (4)                 return out_dir
SNGT_QHENOMENOLOGY_347: (0)             manim_config: Dict = initialize_manim_config()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 20 - logger.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               import logging
SNGT_QHENOMENOLOGY_2: (0)               from rich.logging import RichHandler
SNGT_QHENOMENOLOGY_3: (0)               __all__ = ["log"]
SNGT_QHENOMENOLOGY_4: (0)               FORMAT = "%(message)s"
SNGT_QHENOMENOLOGY_5: (0)               logging.basicConfig(
SNGT_QHENOMENOLOGY_6: (4)                   level=logging.WARNING, format=FORMAT, datefmt="[%X]",
handlers=[RichHandler()]
SNGT_QHENOMENOLOGY_7: (0)               )
SNGT_QHENOMENOLOGY_8: (0)               log = logging.getLogger("manimgl")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 21 - update.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_3: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_4: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_5: (4)                   from typing import Callable
SNGT_QHENOMENOLOGY_6: (4)                   from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_7: (0)               class UpdateFromFunc(Animation):
SNGT_QHENOMENOLOGY_8: (4)                   """
SNGT_QHENOMENOLOGY_9: (4)                   update_function of the form func(mobject), presumably
SNGT_QHENOMENOLOGY_10: (4)                  to be used when the state of one mobject is dependent
SNGT_QHENOMENOLOGY_11: (4)                  on another simultaneously animated mobject
SNGT_QHENOMENOLOGY_12: (4)                  """
SNGT_QHENOMENOLOGY_13: (4)                  def __init__(
SNGT_QHENOMENOLOGY_14: (8)                      self,
SNGT_QHENOMENOLOGY_15: (8)                      mobject: Mobject,
SNGT_QHENOMENOLOGY_16: (8)                      update_function: Callable[[Mobject], Mobject |
None],
SNGT_QHENOMENOLOGY_17: (8)                      suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_18: (8)                      **kwargs
SNGT_QHENOMENOLOGY_19: (4)                  ):
SNGT_QHENOMENOLOGY_20: (8)                      self.update_function = update_function
SNGT_QHENOMENOLOGY_21: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_22: (12)                         mobject,
SNGT_QHENOMENOLOGY_23: (12)
suspend_mobject_updating=suspend_mobject_updating,
SNGT_QHENOMENOLOGY_24: (12)                         **kwargs
SNGT_QHENOMENOLOGY_25: (8)                      )
SNGT_QHENOMENOLOGY_26: (4)                  def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_27: (8)                      self.update_function(self.mobject)
```

```
SNGT_QHENOMENOLOGY_28: (0)             class UpdateFromAlphaFunc(Animation):
SNGT_QHENOMENOLOGY_29: (4)                 def __init__(
SNGT_QHENOMENOLOGY_30: (8)                     self,
SNGT_QHENOMENOLOGY_31: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_32: (8)                     update_function: Callable[[Mobject, float], Mobject
| None],
SNGT_QHENOMENOLOGY_33: (8)                     suspend_mobject_updating: bool = False,
SNGT_QHENOMENOLOGY_34: (8)                     **kwargs
SNGT_QHENOMENOLOGY_35: (4)                 ):
SNGT_QHENOMENOLOGY_36: (8)                     self.update_function = update_function
SNGT_QHENOMENOLOGY_37: (8)                     super().__init__(mobject,
suspend_mobject_updating=suspend_mobject_updating, **kwargs)
SNGT_QHENOMENOLOGY_38: (4)                 def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_39: (8)                     self.update_function(self.mobject, alpha)
SNGT_QHENOMENOLOGY_40: (0)             class MaintainPositionRelativeTo(Animation):
SNGT_QHENOMENOLOGY_41: (4)                 def __init__(
SNGT_QHENOMENOLOGY_42: (8)                     self,
SNGT_QHENOMENOLOGY_43: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_44: (8)                     tracked_mobject: Mobject,
SNGT_QHENOMENOLOGY_45: (8)                     **kwargs
SNGT_QHENOMENOLOGY_46: (4)                 ):
SNGT_QHENOMENOLOGY_47: (8)                     self.tracked_mobject = tracked_mobject
SNGT_QHENOMENOLOGY_48: (8)                     self.diff = mobject.get_center() -
tracked_mobject.get_center()
SNGT_QHENOMENOLOGY_49: (8)                     super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_50: (4)                 def interpolate_mobject(self, alpha: float) -> None:
SNGT_QHENOMENOLOGY_51: (8)                     target = self.tracked_mobject.get_center()
SNGT_QHENOMENOLOGY_52: (8)                     location = self.mobject.get_center()
SNGT_QHENOMENOLOGY_53: (8)                     self.mobject.shift(target - location + self.diff)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 22 - camera.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import moderngl
SNGT_QHENOMENOLOGY_3: (0)              import numpy as np
SNGT_QHENOMENOLOGY_4: (0)              import OpenGL.GL as gl
SNGT_QHENOMENOLOGY_5: (0)              from PIL import Image
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.camera.camera_frame import CameraFrame
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.constants import BLACK
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.constants import DEFAULT_RESOLUTION
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.constants import FRAME_HEIGHT
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.constants import FRAME_WIDTH
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.mobject.mobject import Point
SNGT_QHENOMENOLOGY_13: (0)             from manimlib.utils.color import color_to_rgba
SNGT_QHENOMENOLOGY_14: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_15: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_16: (4)                 from typing import Optional
SNGT_QHENOMENOLOGY_17: (4)                 from manimlib.typing import ManimColor, Vect3
SNGT_QHENOMENOLOGY_18: (4)                 from manimlib.window import Window
SNGT_QHENOMENOLOGY_19: (0)             class Camera(object):
SNGT_QHENOMENOLOGY_20: (4)                 def __init__(
SNGT_QHENOMENOLOGY_21: (8)                     self,
SNGT_QHENOMENOLOGY_22: (8)                     window: Optional[Window] = None,
SNGT_QHENOMENOLOGY_23: (8)                     background_image: Optional[str] = None,
SNGT_QHENOMENOLOGY_24: (8)                     frame_config: dict = dict(),
SNGT_QHENOMENOLOGY_25: (8)                     resolution=DEFAULT_RESOLUTION,
SNGT_QHENOMENOLOGY_26: (8)                     fps: int = 30,
SNGT_QHENOMENOLOGY_27: (8)                     background_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_28: (8)                     background_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_29: (8)                     max_allowable_norm: float = FRAME_WIDTH,
SNGT_QHENOMENOLOGY_30: (8)                     image_mode: str = "RGBA",
SNGT_QHENOMENOLOGY_31: (8)                     n_channels: int = 4,
SNGT_QHENOMENOLOGY_32: (8)                     pixel_array_dtype: type = np.uint8,
SNGT_QHENOMENOLOGY_33: (8)                     light_source_position: Vect3 = np.array([-10, 10,
10]),
SNGT_QHENOMENOLOGY_34: (8)                     samples: int = 0,
```

```
SNGT_QHENOMENOLOGY_35: (4)                  ):
SNGT_QHENOMENOLOGY_36: (8)                      self.window = window
SNGT_QHENOMENOLOGY_37: (8)                      self.background_image = background_image
SNGT_QHENOMENOLOGY_38: (8)                      self.default_pixel_shape = resolution  # Rename?
SNGT_QHENOMENOLOGY_39: (8)                      self.fps = fps
SNGT_QHENOMENOLOGY_40: (8)                      self.max_allowable_norm = max_allowable_norm
SNGT_QHENOMENOLOGY_41: (8)                      self.image_mode = image_mode
SNGT_QHENOMENOLOGY_42: (8)                      self.n_channels = n_channels
SNGT_QHENOMENOLOGY_43: (8)                      self.pixel_array_dtype = pixel_array_dtype
SNGT_QHENOMENOLOGY_44: (8)                      self.light_source_position = light_source_position
SNGT_QHENOMENOLOGY_45: (8)                      self.samples = samples
SNGT_QHENOMENOLOGY_46: (8)                      self.rgb_max_val: float =
np.iinfo(self.pixel_array_dtype).max
SNGT_QHENOMENOLOGY_47: (8)                      self.background_rgba: list[float] =
list(color_to_rgba(
SNGT_QHENOMENOLOGY_48: (12)                         background_color, background_opacity
SNGT_QHENOMENOLOGY_49: (8)                      ))
SNGT_QHENOMENOLOGY_50: (8)                      self.uniforms = dict()
SNGT_QHENOMENOLOGY_51: (8)                      self.init_frame(**frame_config)
SNGT_QHENOMENOLOGY_52: (8)                      self.init_context()
SNGT_QHENOMENOLOGY_53: (8)                      self.init_fbo()
SNGT_QHENOMENOLOGY_54: (8)                      self.init_light_source()
SNGT_QHENOMENOLOGY_55: (4)                  def init_frame(self, **config) -> None:
SNGT_QHENOMENOLOGY_56: (8)                      self.frame = CameraFrame(**config)
SNGT_QHENOMENOLOGY_57: (4)                  def init_context(self) -> None:
SNGT_QHENOMENOLOGY_58: (8)                      if self.window is None:
SNGT_QHENOMENOLOGY_59: (12)                         self.ctx: moderngl.Context =
moderngl.create_standalone_context()
SNGT_QHENOMENOLOGY_60: (8)                      else:
SNGT_QHENOMENOLOGY_61: (12)                         self.ctx: moderngl.Context = self.window.ctx
SNGT_QHENOMENOLOGY_62: (8)                      self.ctx.enable(moderngl.PROGRAM_POINT_SIZE)
SNGT_QHENOMENOLOGY_63: (8)                      self.ctx.enable(moderngl.BLEND)
SNGT_QHENOMENOLOGY_64: (4)                  def init_fbo(self) -> None:
SNGT_QHENOMENOLOGY_65: (8)                      self.fbo_for_files = self.get_fbo(self.samples)
SNGT_QHENOMENOLOGY_66: (8)                      self.draw_fbo = self.get_fbo(samples=0)
SNGT_QHENOMENOLOGY_67: (8)                      if self.window is None:
SNGT_QHENOMENOLOGY_68: (12)                         self.window_fbo = None
SNGT_QHENOMENOLOGY_69: (12)                         self.fbo = self.fbo_for_files
SNGT_QHENOMENOLOGY_70: (8)                      else:
SNGT_QHENOMENOLOGY_71: (12)                         self.window_fbo = self.ctx.detect_framebuffer()
SNGT_QHENOMENOLOGY_72: (12)                         self.fbo = self.window_fbo
SNGT_QHENOMENOLOGY_73: (8)                      self.fbo.use()
SNGT_QHENOMENOLOGY_74: (4)                  def init_light_source(self) -> None:
SNGT_QHENOMENOLOGY_75: (8)                      self.light_source =
Point(self.light_source_position)
SNGT_QHENOMENOLOGY_76: (4)                  def use_window_fbo(self, use: bool = True):
SNGT_QHENOMENOLOGY_77: (8)                      assert self.window is not None
SNGT_QHENOMENOLOGY_78: (8)                      if use:
SNGT_QHENOMENOLOGY_79: (12)                         self.fbo = self.window_fbo
SNGT_QHENOMENOLOGY_80: (8)                      else:
SNGT_QHENOMENOLOGY_81: (12)                         self.fbo = self.fbo_for_files
SNGT_QHENOMENOLOGY_82: (4)                  def get_fbo(
SNGT_QHENOMENOLOGY_83: (8)                      self,
SNGT_QHENOMENOLOGY_84: (8)                      samples: int = 0
SNGT_QHENOMENOLOGY_85: (4)                  ) -> moderngl.Framebuffer:
SNGT_QHENOMENOLOGY_86: (8)                      return self.ctx.framebuffer(
SNGT_QHENOMENOLOGY_87: (12)                         color_attachments=self.ctx.texture(
SNGT_QHENOMENOLOGY_88: (16)                             self.default_pixel_shape,
SNGT_QHENOMENOLOGY_89: (16)                             components=self.n_channels,
SNGT_QHENOMENOLOGY_90: (16)                             samples=samples,
SNGT_QHENOMENOLOGY_91: (12)                         ),
SNGT_QHENOMENOLOGY_92: (12)                         depth_attachment=self.ctx.depth_renderbuffer(
SNGT_QHENOMENOLOGY_93: (16)                             self.default_pixel_shape,
SNGT_QHENOMENOLOGY_94: (16)                             samples=samples
SNGT_QHENOMENOLOGY_95: (12)                         )
SNGT_QHENOMENOLOGY_96: (8)                      )
SNGT_QHENOMENOLOGY_97: (4)                  def clear(self) -> None:
SNGT_QHENOMENOLOGY_98: (8)                      self.fbo.clear(*self.background_rgba)
SNGT_QHENOMENOLOGY_99: (8)                      if self.window:
```

```
SNGT_QHENOMENOLOGY_100: (12)                        self.window.clear(*self.background_rgba)
SNGT_QHENOMENOLOGY_101: (4)                 def blit(self, src_fbo, dst_fbo):
SNGT_QHENOMENOLOGY_102: (8)                     """
SNGT_QHENOMENOLOGY_103: (8)                     Copy blocks between fbo's using Blit
SNGT_QHENOMENOLOGY_104: (8)                     """
SNGT_QHENOMENOLOGY_105: (8)                     gl.glBindFramebuffer(gl.GL_READ_FRAMEBUFFER,
src_fbo.glo)
SNGT_QHENOMENOLOGY_106: (8)                     gl.glBindFramebuffer(gl.GL_DRAW_FRAMEBUFFER,
dst_fbo.glo)
SNGT_QHENOMENOLOGY_107: (8)                     gl.glBlitFramebuffer(
SNGT_QHENOMENOLOGY_108: (12)                        *src_fbo.viewport,
SNGT_QHENOMENOLOGY_109: (12)                        *dst_fbo.viewport,
SNGT_QHENOMENOLOGY_110: (12)                        gl.GL_COLOR_BUFFER_BIT, gl.GL_LINEAR
SNGT_QHENOMENOLOGY_111: (8)                     )
SNGT_QHENOMENOLOGY_112: (4)                 def get_raw_fbo_data(self, dtype: str = 'f1') -> bytes:
SNGT_QHENOMENOLOGY_113: (8)                     self.blit(self.fbo, self.draw_fbo)
SNGT_QHENOMENOLOGY_114: (8)                     return self.draw_fbo.read(
SNGT_QHENOMENOLOGY_115: (12)                        viewport=self.draw_fbo.viewport,
SNGT_QHENOMENOLOGY_116: (12)                        components=self.n_channels,
SNGT_QHENOMENOLOGY_117: (12)                        dtype=dtype,
SNGT_QHENOMENOLOGY_118: (8)                     )
SNGT_QHENOMENOLOGY_119: (4)                 def get_image(self) -> Image.Image:
SNGT_QHENOMENOLOGY_120: (8)                     return Image.frombytes(
SNGT_QHENOMENOLOGY_121: (12)                        'RGBA',
SNGT_QHENOMENOLOGY_122: (12)                        self.get_pixel_shape(),
SNGT_QHENOMENOLOGY_123: (12)                        self.get_raw_fbo_data(),
SNGT_QHENOMENOLOGY_124: (12)                        'raw', 'RGBA', 0, -1
SNGT_QHENOMENOLOGY_125: (8)                     )
SNGT_QHENOMENOLOGY_126: (4)                 def get_pixel_array(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_127: (8)                     raw = self.get_raw_fbo_data(dtype='f4')
SNGT_QHENOMENOLOGY_128: (8)                     flat_arr = np.frombuffer(raw, dtype='f4')
SNGT_QHENOMENOLOGY_129: (8)                     arr =
flat_arr.reshape([*reversed(self.draw_fbo.size), self.n_channels])
SNGT_QHENOMENOLOGY_130: (8)                     arr = arr[::-1]
SNGT_QHENOMENOLOGY_131: (8)                     return (self.rgb_max_val *
arr).astype(self.pixel_array_dtype)
SNGT_QHENOMENOLOGY_132: (4)                 def get_texture(self) -> moderngl.Texture:
SNGT_QHENOMENOLOGY_133: (8)                     texture = self.ctx.texture(
SNGT_QHENOMENOLOGY_134: (12)                        size=self.fbo.size,
SNGT_QHENOMENOLOGY_135: (12)                        components=4,
SNGT_QHENOMENOLOGY_136: (12)                        data=self.get_raw_fbo_data(),
SNGT_QHENOMENOLOGY_137: (12)                        dtype='f4'
SNGT_QHENOMENOLOGY_138: (8)                     )
SNGT_QHENOMENOLOGY_139: (8)                     return texture
SNGT_QHENOMENOLOGY_140: (4)                 def get_pixel_size(self) -> float:
SNGT_QHENOMENOLOGY_141: (8)                     return self.frame.get_width() /
self.get_pixel_shape()[0]
SNGT_QHENOMENOLOGY_142: (4)                 def get_pixel_shape(self) -> tuple[int, int]:
SNGT_QHENOMENOLOGY_143: (8)                     return self.fbo.size
SNGT_QHENOMENOLOGY_144: (4)                 def get_pixel_width(self) -> int:
SNGT_QHENOMENOLOGY_145: (8)                     return self.get_pixel_shape()[0]
SNGT_QHENOMENOLOGY_146: (4)                 def get_pixel_height(self) -> int:
SNGT_QHENOMENOLOGY_147: (8)                     return self.get_pixel_shape()[1]
SNGT_QHENOMENOLOGY_148: (4)                 def get_aspect_ratio(self):
SNGT_QHENOMENOLOGY_149: (8)                     pw, ph = self.get_pixel_shape()
SNGT_QHENOMENOLOGY_150: (8)                     return pw / ph
SNGT_QHENOMENOLOGY_151: (4)                 def get_frame_height(self) -> float:
SNGT_QHENOMENOLOGY_152: (8)                     return self.frame.get_height()
SNGT_QHENOMENOLOGY_153: (4)                 def get_frame_width(self) -> float:
SNGT_QHENOMENOLOGY_154: (8)                     return self.frame.get_width()
SNGT_QHENOMENOLOGY_155: (4)                 def get_frame_shape(self) -> tuple[float, float]:
SNGT_QHENOMENOLOGY_156: (8)                     return (self.get_frame_width(),
self.get_frame_height())
SNGT_QHENOMENOLOGY_157: (4)                 def get_frame_center(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_158: (8)                     return self.frame.get_center()
SNGT_QHENOMENOLOGY_159: (4)                 def get_location(self) -> tuple[float, float, float]:
SNGT_QHENOMENOLOGY_160: (8)                     return self.frame.get_implied_camera_location()
SNGT_QHENOMENOLOGY_161: (4)                 def resize_frame_shape(self, fixed_dimension: bool =
False) -> None:
```

```
SNGT_QHENOMENOLOGY_162: (8)                     """
SNGT_QHENOMENOLOGY_163: (8)                     Changes frame_shape to match the aspect ratio
SNGT_QHENOMENOLOGY_164: (8)                     of the pixels, where fixed_dimension determines
SNGT_QHENOMENOLOGY_165: (8)                     whether frame_height or frame_width
SNGT_QHENOMENOLOGY_166: (8)                     remains fixed while the other changes accordingly.
SNGT_QHENOMENOLOGY_167: (8)                     """
SNGT_QHENOMENOLOGY_168: (8)                     frame_height = self.get_frame_height()
SNGT_QHENOMENOLOGY_169: (8)                     frame_width = self.get_frame_width()
SNGT_QHENOMENOLOGY_170: (8)                     aspect_ratio = self.get_aspect_ratio()
SNGT_QHENOMENOLOGY_171: (8)                     if not fixed_dimension:
SNGT_QHENOMENOLOGY_172: (12)                        frame_height = frame_width / aspect_ratio
SNGT_QHENOMENOLOGY_173: (8)                     else:
SNGT_QHENOMENOLOGY_174: (12)                        frame_width = aspect_ratio * frame_height
SNGT_QHENOMENOLOGY_175: (8)                     self.frame.set_height(frame_height, stretch=True)
SNGT_QHENOMENOLOGY_176: (8)                     self.frame.set_width(frame_width, stretch=True)
SNGT_QHENOMENOLOGY_177: (4)                 def capture(self, *mobjects: Mobject) -> None:
SNGT_QHENOMENOLOGY_178: (8)                     self.clear()
SNGT_QHENOMENOLOGY_179: (8)                     self.refresh_uniforms()
SNGT_QHENOMENOLOGY_180: (8)                     self.fbo.use()
SNGT_QHENOMENOLOGY_181: (8)                     for mobject in mobjects:
SNGT_QHENOMENOLOGY_182: (12)                        mobject.render(self.ctx, self.uniforms)
SNGT_QHENOMENOLOGY_183: (8)                     if self.window:
SNGT_QHENOMENOLOGY_184: (12)                        self.window.swap_buffers()
SNGT_QHENOMENOLOGY_185: (12)                        if self.fbo is not self.window_fbo:
SNGT_QHENOMENOLOGY_186: (16)                            self.blit(self.fbo, self.window_fbo)
SNGT_QHENOMENOLOGY_187: (16)                            self.window.swap_buffers()
SNGT_QHENOMENOLOGY_188: (4)                 def refresh_uniforms(self) -> None:
SNGT_QHENOMENOLOGY_189: (8)                     frame = self.frame
SNGT_QHENOMENOLOGY_190: (8)                     view_matrix = frame.get_view_matrix()
SNGT_QHENOMENOLOGY_191: (8)                     light_pos = self.light_source.get_location()
SNGT_QHENOMENOLOGY_192: (8)                     cam_pos = self.frame.get_implied_camera_location()
SNGT_QHENOMENOLOGY_193: (8)                     self.uniforms.update(
SNGT_QHENOMENOLOGY_194: (12)                        view=tuple(view_matrix.T.flatten()),
SNGT_QHENOMENOLOGY_195: (12)                        frame_scale=frame.get_scale(),
SNGT_QHENOMENOLOGY_196: (12)                        frame_rescale_factors=(
SNGT_QHENOMENOLOGY_197: (16)                            2.0 / FRAME_WIDTH,
SNGT_QHENOMENOLOGY_198: (16)                            2.0 / FRAME_HEIGHT,
SNGT_QHENOMENOLOGY_199: (16)                            frame.get_scale() /

frame.get_focal_distance(),
SNGT_QHENOMENOLOGY_200: (12)                        ),
SNGT_QHENOMENOLOGY_201: (12)                        pixel_size=self.get_pixel_size(),
SNGT_QHENOMENOLOGY_202: (12)                        camera_position=tuple(cam_pos),
SNGT_QHENOMENOLOGY_203: (12)                        light_position=tuple(light_pos),
SNGT_QHENOMENOLOGY_204: (8)                     )
SNGT_QHENOMENOLOGY_205: (0)             class ThreeDCamera(Camera):
SNGT_QHENOMENOLOGY_206: (4)                 def __init__(self, samples: int = 4, **kwargs):
SNGT_QHENOMENOLOGY_207: (8)                     super().__init__(samples=samples, **kwargs)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 23 - matrix.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.constants import DOWN, LEFT, RIGHT, ORIGIN
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.mobject.numbers import DecimalNumber
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_9: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                  from typing import Sequence, Union, Optional
SNGT_QHENOMENOLOGY_12: (4)                  from manimlib.typing import ManimColor, Vect3,
VectNArray, Self
SNGT_QHENOMENOLOGY_13: (4)                  StringMatrixType = Union[Sequence[Sequence[str]],
np.ndarray[int, np.dtype[np.str_]]]
```

```
SNGT_QHENOMENOLOGY_14: (4)                 FloatMatrixType = Union[Sequence[Sequence[float]],
VectNArray]
SNGT_QHENOMENOLOGY_15: (4)                 VMobjectMatrixType = Sequence[Sequence[VMobject]]
SNGT_QHENOMENOLOGY_16: (4)                 GenericMatrixType = Union[FloatMatrixType,
StringMatrixType, VMobjectMatrixType]
SNGT_QHENOMENOLOGY_17: (0)             class Matrix(VMobject):
SNGT_QHENOMENOLOGY_18: (4)                 def __init__(
SNGT_QHENOMENOLOGY_19: (8)                     self,
SNGT_QHENOMENOLOGY_20: (8)                     matrix: GenericMatrixType,
SNGT_QHENOMENOLOGY_21: (8)                     v_buff: float = 0.5,
SNGT_QHENOMENOLOGY_22: (8)                     h_buff: float = 0.5,
SNGT_QHENOMENOLOGY_23: (8)                     bracket_h_buff: float = 0.2,
SNGT_QHENOMENOLOGY_24: (8)                     bracket_v_buff: float = 0.25,
SNGT_QHENOMENOLOGY_25: (8)                     height: float | None = None,
SNGT_QHENOMENOLOGY_26: (8)                     element_config: dict = dict(),
SNGT_QHENOMENOLOGY_27: (8)                     element_alignment_corner: Vect3 = DOWN,
SNGT_QHENOMENOLOGY_28: (8)                     ellipses_row: Optional[int] = None,
SNGT_QHENOMENOLOGY_29: (8)                     ellipses_col: Optional[int] = None,
SNGT_QHENOMENOLOGY_30: (4)                 ):
SNGT_QHENOMENOLOGY_31: (8)                     """
SNGT_QHENOMENOLOGY_32: (8)                     Matrix can either include numbers, tex_strings,
SNGT_QHENOMENOLOGY_33: (8)                     or mobjects
SNGT_QHENOMENOLOGY_34: (8)                     """
SNGT_QHENOMENOLOGY_35: (8)                     super().__init__()
SNGT_QHENOMENOLOGY_36: (8)                     self.mob_matrix = self.create_mobject_matrix(
SNGT_QHENOMENOLOGY_37: (12)                        matrix, v_buff, h_buff,
element_alignment_corner,
SNGT_QHENOMENOLOGY_38: (12)                        **element_config
SNGT_QHENOMENOLOGY_39: (8)                     )
SNGT_QHENOMENOLOGY_40: (8)                     n_cols = len(self.mob_matrix[0])
SNGT_QHENOMENOLOGY_41: (8)                     self.elements = [elem for row in self.mob_matrix
for elem in row]
SNGT_QHENOMENOLOGY_42: (8)                     self.columns = VGroup(*(
SNGT_QHENOMENOLOGY_43: (12)                        VGroup(*(row[i] for row in self.mob_matrix))
SNGT_QHENOMENOLOGY_44: (12)                        for i in range(n_cols)
SNGT_QHENOMENOLOGY_45: (8)                     ))
SNGT_QHENOMENOLOGY_46: (8)                     self.rows = VGroup(*(VGroup(*row) for row in
self.mob_matrix))
SNGT_QHENOMENOLOGY_47: (8)                     if height is not None:
SNGT_QHENOMENOLOGY_48: (12)                        self.rows.set_height(height - 2 *
bracket_v_buff)
SNGT_QHENOMENOLOGY_49: (8)                     self.brackets = self.create_brackets(self.rows,
bracket_v_buff, bracket_h_buff)
SNGT_QHENOMENOLOGY_50: (8)                     self.ellipses = []
SNGT_QHENOMENOLOGY_51: (8)                     self.add(*self.elements)
SNGT_QHENOMENOLOGY_52: (8)                     self.add(*self.brackets)
SNGT_QHENOMENOLOGY_53: (8)                     self.center()
SNGT_QHENOMENOLOGY_54: (8)                     self.swap_entries_for_ellipses(
SNGT_QHENOMENOLOGY_55: (12)                        ellipses_row,
SNGT_QHENOMENOLOGY_56: (12)                        ellipses_col,
SNGT_QHENOMENOLOGY_57: (8)                     )
SNGT_QHENOMENOLOGY_58: (4)                 def copy(self, deep: bool = False):
SNGT_QHENOMENOLOGY_59: (8)                     result = super().copy(deep)
SNGT_QHENOMENOLOGY_60: (8)                     self_family = self.get_family()
SNGT_QHENOMENOLOGY_61: (8)                     copy_family = result.get_family()
SNGT_QHENOMENOLOGY_62: (8)                     for attr in ["elements", "ellipses"]:
SNGT_QHENOMENOLOGY_63: (12)                        setattr(result, attr, [
SNGT_QHENOMENOLOGY_64: (16)                            copy_family[self_family.index(mob)]
SNGT_QHENOMENOLOGY_65: (16)                            for mob in getattr(self, attr)
SNGT_QHENOMENOLOGY_66: (12)                        ])
SNGT_QHENOMENOLOGY_67: (8)                     return result
SNGT_QHENOMENOLOGY_68: (4)                 def create_mobject_matrix(
SNGT_QHENOMENOLOGY_69: (8)                     self,
SNGT_QHENOMENOLOGY_70: (8)                     matrix: GenericMatrixType,
SNGT_QHENOMENOLOGY_71: (8)                     v_buff: float,
SNGT_QHENOMENOLOGY_72: (8)                     h_buff: float,
SNGT_QHENOMENOLOGY_73: (8)                     aligned_corner: Vect3,
SNGT_QHENOMENOLOGY_74: (8)                     **element_config
SNGT_QHENOMENOLOGY_75: (4)                 ) -> VMobjectMatrixType:
```

```
SNGT_QHENOMENOLOGY_76: (8)                    """
SNGT_QHENOMENOLOGY_77: (8)                    Creates and organizes the matrix of mobjects
SNGT_QHENOMENOLOGY_78: (8)                    """
SNGT_QHENOMENOLOGY_79: (8)                    mob_matrix = [
SNGT_QHENOMENOLOGY_80: (12)                       [
SNGT_QHENOMENOLOGY_81: (16)                           self.element_to_mobject(element,
**element_config)
SNGT_QHENOMENOLOGY_82: (16)                           for element in row
SNGT_QHENOMENOLOGY_83: (12)                       ]
SNGT_QHENOMENOLOGY_84: (12)                       for row in matrix
SNGT_QHENOMENOLOGY_85: (8)                    ]
SNGT_QHENOMENOLOGY_86: (8)                    max_width = max(elem.get_width() for row in
mob_matrix for elem in row)
SNGT_QHENOMENOLOGY_87: (8)                    max_height = max(elem.get_height() for row in
mob_matrix for elem in row)
SNGT_QHENOMENOLOGY_88: (8)                    x_step = (max_width + h_buff) * RIGHT
SNGT_QHENOMENOLOGY_89: (8)                    y_step = (max_height + v_buff) * DOWN
SNGT_QHENOMENOLOGY_90: (8)                    for i, row in enumerate(mob_matrix):
SNGT_QHENOMENOLOGY_91: (12)                       for j, elem in enumerate(row):
SNGT_QHENOMENOLOGY_92: (16)                           elem.move_to(i * y_step + j * x_step,
aligned_corner)
SNGT_QHENOMENOLOGY_93: (8)                    return mob_matrix
SNGT_QHENOMENOLOGY_94: (4)                def element_to_mobject(self, element, **config) ->
VMobject:
SNGT_QHENOMENOLOGY_95: (8)                    if isinstance(element, VMobject):
SNGT_QHENOMENOLOGY_96: (12)                       return element
SNGT_QHENOMENOLOGY_97: (8)                    elif isinstance(element, float | complex):
SNGT_QHENOMENOLOGY_98: (12)                       return DecimalNumber(element, **config)
SNGT_QHENOMENOLOGY_99: (8)                    else:
SNGT_QHENOMENOLOGY_100: (12)                      return Tex(str(element), **config)
SNGT_QHENOMENOLOGY_101: (4)               def create_brackets(self, rows, v_buff: float, h_buff:
float) -> VGroup:
SNGT_QHENOMENOLOGY_102: (8)                   brackets = Tex("".join((
SNGT_QHENOMENOLOGY_103: (12)                      R"\left[\begin{array}{c}",
SNGT_QHENOMENOLOGY_104: (12)                      *len(rows) * [R"\quad \\"],
SNGT_QHENOMENOLOGY_105: (12)                      R"\end{array}\right]",
SNGT_QHENOMENOLOGY_106: (8)                   )))
SNGT_QHENOMENOLOGY_107: (8)                   brackets.set_height(rows.get_height() + v_buff)
SNGT_QHENOMENOLOGY_108: (8)                   l_bracket = brackets[:len(brackets) // 2]
SNGT_QHENOMENOLOGY_109: (8)                   r_bracket = brackets[len(brackets) // 2:]
SNGT_QHENOMENOLOGY_110: (8)                   l_bracket.next_to(rows, LEFT, h_buff)
SNGT_QHENOMENOLOGY_111: (8)                   r_bracket.next_to(rows, RIGHT, h_buff)
SNGT_QHENOMENOLOGY_112: (8)                   return VGroup(l_bracket, r_bracket)
SNGT_QHENOMENOLOGY_113: (4)               def get_column(self, index: int):
SNGT_QHENOMENOLOGY_114: (8)                   if not 0 <= index < len(self.columns):
SNGT_QHENOMENOLOGY_115: (12)                      raise IndexError(f"Index {index} out of bound
for matrix with {len(self.columns)} columns")
SNGT_QHENOMENOLOGY_116: (8)                   return self.columns[index]
SNGT_QHENOMENOLOGY_117: (4)               def get_row(self, index: int):
SNGT_QHENOMENOLOGY_118: (8)                   if not 0 <= index < len(self.rows):
SNGT_QHENOMENOLOGY_119: (12)                      raise IndexError(f"Index {index} out of bound
for matrix with {len(self.rows)} rows")
SNGT_QHENOMENOLOGY_120: (8)                   return self.rows[index]
SNGT_QHENOMENOLOGY_121: (4)               def get_columns(self) -> VGroup:
SNGT_QHENOMENOLOGY_122: (8)                   return self.columns
SNGT_QHENOMENOLOGY_123: (4)               def get_rows(self) -> VGroup:
SNGT_QHENOMENOLOGY_124: (8)                   return self.rows
SNGT_QHENOMENOLOGY_125: (4)               def set_column_colors(self, *colors: ManimColor) ->
Self:
SNGT_QHENOMENOLOGY_126: (8)                   columns = self.get_columns()
SNGT_QHENOMENOLOGY_127: (8)                   for color, column in zip(colors, columns):
SNGT_QHENOMENOLOGY_128: (12)                      column.set_color(color)
SNGT_QHENOMENOLOGY_129: (8)                   return self
SNGT_QHENOMENOLOGY_130: (4)               def add_background_to_entries(self) -> Self:
SNGT_QHENOMENOLOGY_131: (8)                   for mob in self.get_entries():
SNGT_QHENOMENOLOGY_132: (12)                      mob.add_background_rectangle()
SNGT_QHENOMENOLOGY_133: (8)                   return self
SNGT_QHENOMENOLOGY_134: (4)               def swap_entry_for_dots(self, entry, dots):
SNGT_QHENOMENOLOGY_135: (8)                   dots.move_to(entry)
```

```
SNGT_QHENOMENOLOGY_136: (8)                     entry.become(dots)
SNGT_QHENOMENOLOGY_137: (8)                     if entry in self.elements:
SNGT_QHENOMENOLOGY_138: (12)                        self.elements.remove(entry)
SNGT_QHENOMENOLOGY_139: (8)                     if entry not in self.ellipses:
SNGT_QHENOMENOLOGY_140: (12)                        self.ellipses.append(entry)
SNGT_QHENOMENOLOGY_141: (4)                 def swap_entries_for_ellipses(
SNGT_QHENOMENOLOGY_142: (8)                     self,
SNGT_QHENOMENOLOGY_143: (8)                     row_index: Optional[int] = None,
SNGT_QHENOMENOLOGY_144: (8)                     col_index: Optional[int] = None,
SNGT_QHENOMENOLOGY_145: (8)                     height_ratio: float = 0.65,
SNGT_QHENOMENOLOGY_146: (8)                     width_ratio: float = 0.4
SNGT_QHENOMENOLOGY_147: (4)                 ):
SNGT_QHENOMENOLOGY_148: (8)                     rows = self.get_rows()
SNGT_QHENOMENOLOGY_149: (8)                     cols = self.get_columns()
SNGT_QHENOMENOLOGY_150: (8)                     avg_row_height = rows.get_height() / len(rows)
SNGT_QHENOMENOLOGY_151: (8)                     vdots_height = height_ratio * avg_row_height
SNGT_QHENOMENOLOGY_152: (8)                     avg_col_width = cols.get_width() / len(cols)
SNGT_QHENOMENOLOGY_153: (8)                     hdots_width = width_ratio * avg_col_width
SNGT_QHENOMENOLOGY_154: (8)                     use_vdots = row_index is not None and -len(rows) <=
row_index < len(rows)
SNGT_QHENOMENOLOGY_155: (8)                     use_hdots = col_index is not None and -len(cols) <=
col_index < len(cols)
SNGT_QHENOMENOLOGY_156: (8)                     if use_vdots:
SNGT_QHENOMENOLOGY_157: (12)                        for column in cols:
SNGT_QHENOMENOLOGY_158: (16)                            dots = Tex(R"\vdots")
SNGT_QHENOMENOLOGY_159: (16)                            dots.set_height(vdots_height)
SNGT_QHENOMENOLOGY_160: (16)                            self.swap_entry_for_dots(column[row_index],
dots)
SNGT_QHENOMENOLOGY_161: (8)                     if use_hdots:
SNGT_QHENOMENOLOGY_162: (12)                        for row in rows:
SNGT_QHENOMENOLOGY_163: (16)                            dots = Tex(R"\hdots")
SNGT_QHENOMENOLOGY_164: (16)                            dots.set_width(hdots_width)
SNGT_QHENOMENOLOGY_165: (16)                            self.swap_entry_for_dots(row[col_index],
dots)
SNGT_QHENOMENOLOGY_166: (8)                     if use_vdots and use_hdots:
SNGT_QHENOMENOLOGY_167: (12)                        rows[row_index][col_index].rotate(-45 * DEG)
SNGT_QHENOMENOLOGY_168: (8)                     return self
SNGT_QHENOMENOLOGY_169: (4)                 def get_mob_matrix(self) -> VMobjectMatrixType:
SNGT_QHENOMENOLOGY_170: (8)                     return self.mob_matrix
SNGT_QHENOMENOLOGY_171: (4)                 def get_entries(self) -> VGroup:
SNGT_QHENOMENOLOGY_172: (8)                     return VGroup(*self.elements)
SNGT_QHENOMENOLOGY_173: (4)                 def get_brackets(self) -> VGroup:
SNGT_QHENOMENOLOGY_174: (8)                     return VGroup(*self.brackets)
SNGT_QHENOMENOLOGY_175: (4)                 def get_ellipses(self) -> VGroup:
SNGT_QHENOMENOLOGY_176: (8)                     return VGroup(*self.ellipses)
SNGT_QHENOMENOLOGY_177: (0)             class DecimalMatrix(Matrix):
SNGT_QHENOMENOLOGY_178: (4)                 def __init__(
SNGT_QHENOMENOLOGY_179: (8)                     self,
SNGT_QHENOMENOLOGY_180: (8)                     matrix: FloatMatrixType,
SNGT_QHENOMENOLOGY_181: (8)                     num_decimal_places: int = 2,
SNGT_QHENOMENOLOGY_182: (8)                     decimal_config: dict = dict(),
SNGT_QHENOMENOLOGY_183: (8)                     **config
SNGT_QHENOMENOLOGY_184: (4)                 ):
SNGT_QHENOMENOLOGY_185: (8)                     self.float_matrix = matrix
SNGT_QHENOMENOLOGY_186: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_187: (12)                        matrix,
SNGT_QHENOMENOLOGY_188: (12)                        element_config=dict(
SNGT_QHENOMENOLOGY_189: (16)                            num_decimal_places=num_decimal_places,
SNGT_QHENOMENOLOGY_190: (16)                            **decimal_config
SNGT_QHENOMENOLOGY_191: (12)                        ),
SNGT_QHENOMENOLOGY_192: (12)                        **config
SNGT_QHENOMENOLOGY_193: (8)                     )
SNGT_QHENOMENOLOGY_194: (4)                 def element_to_mobject(self, element, **decimal_config)
-> DecimalNumber:
SNGT_QHENOMENOLOGY_195: (8)                     return DecimalNumber(element, **decimal_config)
SNGT_QHENOMENOLOGY_196: (0)             class IntegerMatrix(DecimalMatrix):
SNGT_QHENOMENOLOGY_197: (4)                 def __init__(
SNGT_QHENOMENOLOGY_198: (8)                     self,
SNGT_QHENOMENOLOGY_199: (8)                     matrix: FloatMatrixType,
```

```
SNGT_QHENOMENOLOGY_200: (8)                     num_decimal_places: int = 0,
SNGT_QHENOMENOLOGY_201: (8)                     decimal_config: dict = dict(),
SNGT_QHENOMENOLOGY_202: (8)                     **config
SNGT_QHENOMENOLOGY_203: (4)                 ):
SNGT_QHENOMENOLOGY_204: (8)                     super().__init__(matrix, num_decimal_places,
decimal_config, **config)
SNGT_QHENOMENOLOGY_205: (0)             class TexMatrix(Matrix):
SNGT_QHENOMENOLOGY_206: (4)                 def __init__(
SNGT_QHENOMENOLOGY_207: (8)                     self,
SNGT_QHENOMENOLOGY_208: (8)                     matrix: StringMatrixType,
SNGT_QHENOMENOLOGY_209: (8)                     tex_config: dict = dict(),
SNGT_QHENOMENOLOGY_210: (8)                     **config,
SNGT_QHENOMENOLOGY_211: (4)                 ):
SNGT_QHENOMENOLOGY_212: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_213: (12)                        matrix,
SNGT_QHENOMENOLOGY_214: (12)                        element_config=tex_config,
SNGT_QHENOMENOLOGY_215: (12)                        **config
SNGT_QHENOMENOLOGY_216: (8)                     )
SNGT_QHENOMENOLOGY_217: (0)             class MobjectMatrix(Matrix):
SNGT_QHENOMENOLOGY_218: (4)                 def __init__(
SNGT_QHENOMENOLOGY_219: (8)                     self,
SNGT_QHENOMENOLOGY_220: (8)                     group: VGroup,
SNGT_QHENOMENOLOGY_221: (8)                     n_rows: int | None = None,
SNGT_QHENOMENOLOGY_222: (8)                     n_cols: int | None = None,
SNGT_QHENOMENOLOGY_223: (8)                     height: float = 4.0,
SNGT_QHENOMENOLOGY_224: (8)                     element_alignment_corner=ORIGIN,
SNGT_QHENOMENOLOGY_225: (8)                     **config,
SNGT_QHENOMENOLOGY_226: (4)                 ):
SNGT_QHENOMENOLOGY_227: (8)                     n_mobs = len(group)
SNGT_QHENOMENOLOGY_228: (8)                     if n_rows is None:
SNGT_QHENOMENOLOGY_229: (12)                        n_rows = int(np.sqrt(n_mobs)) if n_cols is None
else n_mobs // n_cols
SNGT_QHENOMENOLOGY_230: (8)                     if n_cols is None:
SNGT_QHENOMENOLOGY_231: (12)                        n_cols = n_mobs // n_rows
SNGT_QHENOMENOLOGY_232: (8)                     if len(group) < n_rows * n_cols:
SNGT_QHENOMENOLOGY_233: (12)                        raise Exception("Input to MobjectMatrix must
have at least n_rows * n_cols entries")
SNGT_QHENOMENOLOGY_234: (8)                     mob_matrix = [
SNGT_QHENOMENOLOGY_235: (12)                        [group[n * n_cols + k] for k in range(n_cols)]
SNGT_QHENOMENOLOGY_236: (12)                        for n in range(n_rows)
SNGT_QHENOMENOLOGY_237: (8)                     ]
SNGT_QHENOMENOLOGY_238: (8)                     config.update(
SNGT_QHENOMENOLOGY_239: (12)                        height=height,
SNGT_QHENOMENOLOGY_240: (12)
element_alignment_corner=element_alignment_corner,
SNGT_QHENOMENOLOGY_241: (8)                     )
SNGT_QHENOMENOLOGY_242: (8)                     super().__init__(mob_matrix,  **config)
SNGT_QHENOMENOLOGY_243: (4)                 def element_to_mobject(self, element: VMobject,
**config) -> VMobject:
SNGT_QHENOMENOLOGY_244: (8)                     return element
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 24 - mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import copy
SNGT_QHENOMENOLOGY_3: (0)               from functools import wraps
SNGT_QHENOMENOLOGY_4: (0)               import itertools as it
SNGT_QHENOMENOLOGY_5: (0)               import os
SNGT_QHENOMENOLOGY_6: (0)               import pickle
SNGT_QHENOMENOLOGY_7: (0)               import random
SNGT_QHENOMENOLOGY_8: (0)               import sys
SNGT_QHENOMENOLOGY_9: (0)               import moderngl
SNGT_QHENOMENOLOGY_10: (0)              import numbers
SNGT_QHENOMENOLOGY_11: (0)              import numpy as np
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.constants import DEFAULT_MOBJECT_TO_EDGE_BUFF
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.constants import
DEFAULT_MOBJECT_TO_MOBJECT_BUFF
```

```
SNGT_QHENOMENOLOGY_14: (0)             from manimlib.constants import DOWN, IN, LEFT, ORIGIN, OUT,
RIGHT, UP
SNGT_QHENOMENOLOGY_15: (0)             from manimlib.constants import FRAME_X_RADIUS,
FRAME_Y_RADIUS
SNGT_QHENOMENOLOGY_16: (0)             from manimlib.constants import MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_17: (0)             from manimlib.constants import TAU
SNGT_QHENOMENOLOGY_18: (0)             from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_19: (0)             from manimlib.event_handler import EVENT_DISPATCHER
SNGT_QHENOMENOLOGY_20: (0)             from manimlib.event_handler.event_listner import
EventListener
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.event_handler.event_type import EventType
SNGT_QHENOMENOLOGY_22: (0)             from manimlib.logger import log
SNGT_QHENOMENOLOGY_23: (0)             from manimlib.shader_wrapper import ShaderWrapper
SNGT_QHENOMENOLOGY_24: (0)             from manimlib.utils.color import color_gradient
SNGT_QHENOMENOLOGY_25: (0)             from manimlib.utils.color import color_to_rgb
SNGT_QHENOMENOLOGY_26: (0)             from manimlib.utils.color import get_colormap_list
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.utils.color import rgb_to_hex
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.iterables import arrays_match
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.utils.iterables import array_is_constant
SNGT_QHENOMENOLOGY_30: (0)             from manimlib.utils.iterables import batch_by_property
SNGT_QHENOMENOLOGY_31: (0)             from manimlib.utils.iterables import list_update
SNGT_QHENOMENOLOGY_32: (0)             from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_33: (0)             from manimlib.utils.iterables import resize_array
SNGT_QHENOMENOLOGY_34: (0)             from manimlib.utils.iterables import
resize_preserving_order
SNGT_QHENOMENOLOGY_35: (0)             from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_36: (0)             from manimlib.utils.bezier import integer_interpolate
SNGT_QHENOMENOLOGY_37: (0)             from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_38: (0)             from manimlib.utils.paths import straight_path
SNGT_QHENOMENOLOGY_39: (0)             from manimlib.utils.shaders import get_colormap_code
SNGT_QHENOMENOLOGY_40: (0)             from manimlib.utils.space_ops import angle_of_vector
SNGT_QHENOMENOLOGY_41: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_42: (0)             from manimlib.utils.space_ops import
rotation_matrix_transpose
SNGT_QHENOMENOLOGY_43: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_44: (0)             from typing import TypeVar, Generic, Iterable
SNGT_QHENOMENOLOGY_45: (0)             SubmobjectType = TypeVar('SubmobjectType', bound='Mobject')
SNGT_QHENOMENOLOGY_46: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_47: (4)                 from typing import Callable, Iterator, Union, Tuple,
Optional, Any
SNGT_QHENOMENOLOGY_48: (4)                 import numpy.typing as npt
SNGT_QHENOMENOLOGY_49: (4)                 from manimlib.typing import ManimColor, Vect3, Vect4,
Vect3Array, UniformDict, Self
SNGT_QHENOMENOLOGY_50: (4)                 from moderngl.context import Context
SNGT_QHENOMENOLOGY_51: (4)                 T = TypeVar('T')
SNGT_QHENOMENOLOGY_52: (4)                 TimeBasedUpdater = Callable[["Mobject", float],
"Mobject" | None]
SNGT_QHENOMENOLOGY_53: (4)                 NonTimeUpdater = Callable[["Mobject"], "Mobject" |
None]
SNGT_QHENOMENOLOGY_54: (4)                 Updater = Union[TimeBasedUpdater, NonTimeUpdater]
SNGT_QHENOMENOLOGY_55: (0)             class Mobject(object):
SNGT_QHENOMENOLOGY_56: (4)                 """
SNGT_QHENOMENOLOGY_57: (4)                 Mathematical Object
SNGT_QHENOMENOLOGY_58: (4)                 """
SNGT_QHENOMENOLOGY_59: (4)                 dim: int = 3
SNGT_QHENOMENOLOGY_60: (4)                 shader_folder: str = ""
SNGT_QHENOMENOLOGY_61: (4)                 render_primitive: int = moderngl.TRIANGLE_STRIP
SNGT_QHENOMENOLOGY_62: (4)                 data_dtype: np.dtype = np.dtype([
SNGT_QHENOMENOLOGY_63: (8)                     ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_64: (8)                     ('rgba', np.float32, (4,)),
SNGT_QHENOMENOLOGY_65: (4)                 ])
SNGT_QHENOMENOLOGY_66: (4)                 aligned_data_keys = ['point']
SNGT_QHENOMENOLOGY_67: (4)                 pointlike_data_keys = ['point']
SNGT_QHENOMENOLOGY_68: (4)                 def __init__(
SNGT_QHENOMENOLOGY_69: (8)                     self,
SNGT_QHENOMENOLOGY_70: (8)                     color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_71: (8)                     opacity: float = 1.0,
SNGT_QHENOMENOLOGY_72: (8)                     shading: Tuple[float, float, float] = (0.0, 0.0,
```

```
0.0),
SNGT_QHENOMENOLOGY_73: (8)                      texture_paths: dict[str, str] | None = None,
SNGT_QHENOMENOLOGY_74: (8)                      is_fixed_in_frame: bool = False,
SNGT_QHENOMENOLOGY_75: (8)                      depth_test: bool = False,
SNGT_QHENOMENOLOGY_76: (8)                      z_index: int = 0,
SNGT_QHENOMENOLOGY_77: (4)                  ):
SNGT_QHENOMENOLOGY_78: (8)                      self.color = color
SNGT_QHENOMENOLOGY_79: (8)                      self.opacity = opacity
SNGT_QHENOMENOLOGY_80: (8)                      self.shading = shading
SNGT_QHENOMENOLOGY_81: (8)                      self.texture_paths = texture_paths
SNGT_QHENOMENOLOGY_82: (8)                      self.depth_test = depth_test
SNGT_QHENOMENOLOGY_83: (8)                      self.z_index = z_index
SNGT_QHENOMENOLOGY_84: (8)                      self.submobjects: list[Mobject] = []
SNGT_QHENOMENOLOGY_85: (8)                      self.parents: list[Mobject] = []
SNGT_QHENOMENOLOGY_86: (8)                      self.family: list[Mobject] | None = [self]
SNGT_QHENOMENOLOGY_87: (8)                      self.locked_data_keys: set[str] = set()
SNGT_QHENOMENOLOGY_88: (8)                      self.const_data_keys: set[str] = set()
SNGT_QHENOMENOLOGY_89: (8)                      self.locked_uniform_keys: set[str] = set()
SNGT_QHENOMENOLOGY_90: (8)                      self.saved_state = None
SNGT_QHENOMENOLOGY_91: (8)                      self.target = None
SNGT_QHENOMENOLOGY_92: (8)                      self.bounding_box: Vect3Array = np.zeros((3, 3))
SNGT_QHENOMENOLOGY_93: (8)                      self.shader_wrapper: Optional[ShaderWrapper] = None
SNGT_QHENOMENOLOGY_94: (8)                      self._is_animating: bool = False
SNGT_QHENOMENOLOGY_95: (8)                      self._needs_new_bounding_box: bool = True
SNGT_QHENOMENOLOGY_96: (8)                      self._data_has_changed: bool = True
SNGT_QHENOMENOLOGY_97: (8)                      self.shader_code_replacements: dict[str, str] =
dict()
SNGT_QHENOMENOLOGY_98: (8)                      self.init_data()
SNGT_QHENOMENOLOGY_99: (8)                      self.init_uniforms()
SNGT_QHENOMENOLOGY_100: (8)                     self.init_updaters()
SNGT_QHENOMENOLOGY_101: (8)                     self.init_event_listners()
SNGT_QHENOMENOLOGY_102: (8)                     self.init_points()
SNGT_QHENOMENOLOGY_103: (8)                     self.init_colors()
SNGT_QHENOMENOLOGY_104: (8)                     if self.depth_test:
SNGT_QHENOMENOLOGY_105: (12)                        self.apply_depth_test()
SNGT_QHENOMENOLOGY_106: (8)                     if is_fixed_in_frame:
SNGT_QHENOMENOLOGY_107: (12)                        self.fix_in_frame()
SNGT_QHENOMENOLOGY_108: (4)                 def __str__(self):
SNGT_QHENOMENOLOGY_109: (8)                     return self.__class__.__name__
SNGT_QHENOMENOLOGY_110: (4)                 def __add__(self, other: Mobject) -> Mobject:
SNGT_QHENOMENOLOGY_111: (8)                     assert isinstance(other, Mobject)
SNGT_QHENOMENOLOGY_112: (8)                     return self.get_group_class()(self, other)
SNGT_QHENOMENOLOGY_113: (4)                 def __mul__(self, other: int) -> Mobject:
SNGT_QHENOMENOLOGY_114: (8)                     assert isinstance(other, int)
SNGT_QHENOMENOLOGY_115: (8)                     return self.replicate(other)
SNGT_QHENOMENOLOGY_116: (4)                 def init_data(self, length: int = 0):
SNGT_QHENOMENOLOGY_117: (8)                     self.data = np.zeros(length, dtype=self.data_dtype)
SNGT_QHENOMENOLOGY_118: (8)                     self._data_defaults = np.ones(1,
dtype=self.data.dtype)
SNGT_QHENOMENOLOGY_119: (4)                 def init_uniforms(self):
SNGT_QHENOMENOLOGY_120: (8)                     self.uniforms: UniformDict = {
SNGT_QHENOMENOLOGY_121: (12)                        "is_fixed_in_frame": 0.0,
SNGT_QHENOMENOLOGY_122: (12)                        "shading": np.array(self.shading, dtype=float),
SNGT_QHENOMENOLOGY_123: (12)                        "clip_plane": np.zeros(4),
SNGT_QHENOMENOLOGY_124: (8)                     }
SNGT_QHENOMENOLOGY_125: (4)                 def init_colors(self):
SNGT_QHENOMENOLOGY_126: (8)                     self.set_color(self.color, self.opacity)
SNGT_QHENOMENOLOGY_127: (4)                 def init_points(self):
SNGT_QHENOMENOLOGY_128: (8)                     pass
SNGT_QHENOMENOLOGY_129: (4)                 def set_uniforms(self, uniforms: dict) -> Self:
SNGT_QHENOMENOLOGY_130: (8)                     for key, value in uniforms.items():
SNGT_QHENOMENOLOGY_131: (12)                        if isinstance(value, np.ndarray):
SNGT_QHENOMENOLOGY_132: (16)                            value = value.copy()
SNGT_QHENOMENOLOGY_133: (12)                        self.uniforms[key] = value
SNGT_QHENOMENOLOGY_134: (8)                     return self
SNGT_QHENOMENOLOGY_135: (4)                 @property
SNGT_QHENOMENOLOGY_136: (4)                 def animate(self) -> _AnimationBuilder:
SNGT_QHENOMENOLOGY_137: (8)                     """
SNGT_QHENOMENOLOGY_138: (8)                     Methods called with Mobject.animate.method() can be
```

```
passed
SNGT_QHENOMENOLOGY_139: (8)                     into a Scene.play call, as if you were calling
SNGT_QHENOMENOLOGY_140: (8)                     ApplyMethod(mobject.method)
SNGT_QHENOMENOLOGY_141: (8)                     Borrowed from
https://github.com/ManimCommunity/manim/
SNGT_QHENOMENOLOGY_142: (8)                     """
SNGT_QHENOMENOLOGY_143: (8)                     return _AnimationBuilder(self)
SNGT_QHENOMENOLOGY_144: (4)                 @property
SNGT_QHENOMENOLOGY_145: (4)                 def always(self) -> _UpdaterBuilder:
SNGT_QHENOMENOLOGY_146: (8)                     """
SNGT_QHENOMENOLOGY_147: (8)                     Methods called with mobject.always.method(*args,
**kwargs)
SNGT_QHENOMENOLOGY_148: (8)                     will result in the call mobject.method(*args,
**kwargs)
SNGT_QHENOMENOLOGY_149: (8)                     on every frame
SNGT_QHENOMENOLOGY_150: (8)                     """
SNGT_QHENOMENOLOGY_151: (8)                     return _UpdaterBuilder(self)
SNGT_QHENOMENOLOGY_152: (4)                 @property
SNGT_QHENOMENOLOGY_153: (4)                 def f_always(self) -> _FunctionalUpdaterBuilder:
SNGT_QHENOMENOLOGY_154: (8)                     """
SNGT_QHENOMENOLOGY_155: (8)                     Similar to Mobject.always, but with the intent that
arguments
SNGT_QHENOMENOLOGY_156: (8)                     are functions returning the corresponding type fit
for the method
SNGT_QHENOMENOLOGY_157: (8)                     Methods called with
SNGT_QHENOMENOLOGY_158: (8)                     mobject.f_always.method(
SNGT_QHENOMENOLOGY_159: (12)                        func1, func2, ...,
SNGT_QHENOMENOLOGY_160: (12)                        kwarg1=kw_func1,
SNGT_QHENOMENOLOGY_161: (12)                        kwarg2=kw_func2,
SNGT_QHENOMENOLOGY_162: (12)                        ...
SNGT_QHENOMENOLOGY_163: (8)                     )
SNGT_QHENOMENOLOGY_164: (8)                     will result in the call
SNGT_QHENOMENOLOGY_165: (8)                     mobject.method(
SNGT_QHENOMENOLOGY_166: (12)                        func1(), func2(), ...,
SNGT_QHENOMENOLOGY_167: (12)                        kwarg1=kw_func1(),
SNGT_QHENOMENOLOGY_168: (12)                        kwarg2=kw_func2(),
SNGT_QHENOMENOLOGY_169: (12)                        ...
SNGT_QHENOMENOLOGY_170: (8)                     )
SNGT_QHENOMENOLOGY_171: (8)                     on every frame
SNGT_QHENOMENOLOGY_172: (8)                     """
SNGT_QHENOMENOLOGY_173: (8)                     return _FunctionalUpdaterBuilder(self)
SNGT_QHENOMENOLOGY_174: (4)                 def note_changed_data(self, recurse_up: bool = True) ->
Self:
SNGT_QHENOMENOLOGY_175: (8)                     self._data_has_changed = True
SNGT_QHENOMENOLOGY_176: (8)                     if recurse_up:
SNGT_QHENOMENOLOGY_177: (12)                        for mob in self.parents:
SNGT_QHENOMENOLOGY_178: (16)                            mob.note_changed_data()
SNGT_QHENOMENOLOGY_179: (8)                     return self
SNGT_QHENOMENOLOGY_180: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_181: (4)                 def affects_data(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_182: (8)                     @wraps(func)
SNGT_QHENOMENOLOGY_183: (8)                     def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_184: (12)                        result = func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_185: (12)                        self.note_changed_data()
SNGT_QHENOMENOLOGY_186: (12)                        return result
SNGT_QHENOMENOLOGY_187: (8)                     return wrapper
SNGT_QHENOMENOLOGY_188: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_189: (4)                 def affects_family_data(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_190: (8)                     @wraps(func)
SNGT_QHENOMENOLOGY_191: (8)                     def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_192: (12)                        result = func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_193: (12)                        for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_194: (16)                            mob.note_changed_data()
SNGT_QHENOMENOLOGY_195: (12)                        return result
SNGT_QHENOMENOLOGY_196: (8)                     return wrapper
SNGT_QHENOMENOLOGY_197: (4)                 @affects_data
SNGT_QHENOMENOLOGY_198: (4)                 def set_data(self, data: np.ndarray) -> Self:
```

```
SNGT_QHENOMENOLOGY_199: (8)                     assert data.dtype == self.data.dtype
SNGT_QHENOMENOLOGY_200: (8)                     self.resize_points(len(data))
SNGT_QHENOMENOLOGY_201: (8)                     self.data[:] = data
SNGT_QHENOMENOLOGY_202: (8)                     return self
SNGT_QHENOMENOLOGY_203: (4)                 @affects_data
SNGT_QHENOMENOLOGY_204: (4)                 def resize_points(
SNGT_QHENOMENOLOGY_205: (8)                     self,
SNGT_QHENOMENOLOGY_206: (8)                     new_length: int,
SNGT_QHENOMENOLOGY_207: (8)                     resize_func: Callable[[np.ndarray, int],
np.ndarray] = resize_array
SNGT_QHENOMENOLOGY_208: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_209: (8)                     if new_length == 0:
SNGT_QHENOMENOLOGY_210: (12)                        if len(self.data) > 0:
SNGT_QHENOMENOLOGY_211: (16)                            self._data_defaults[:1] = self.data[:1]
SNGT_QHENOMENOLOGY_212: (8)                     elif self.get_num_points() == 0:
SNGT_QHENOMENOLOGY_213: (12)                        self.data = self._data_defaults.copy()
SNGT_QHENOMENOLOGY_214: (8)                     self.data = resize_func(self.data, new_length)
SNGT_QHENOMENOLOGY_215: (8)                     self.refresh_bounding_box()
SNGT_QHENOMENOLOGY_216: (8)                     return self
SNGT_QHENOMENOLOGY_217: (4)                 @affects_data
SNGT_QHENOMENOLOGY_218: (4)                 def set_points(self, points: Vect3Array | list[Vect3])
-> Self:
SNGT_QHENOMENOLOGY_219: (8)                     self.resize_points(len(points),
resize_func=resize_preserving_order)
SNGT_QHENOMENOLOGY_220: (8)                     self.data["point"][:] = points
SNGT_QHENOMENOLOGY_221: (8)                     return self
SNGT_QHENOMENOLOGY_222: (4)                 @affects_data
SNGT_QHENOMENOLOGY_223: (4)                 def append_points(self, new_points: Vect3Array) ->
Self:
SNGT_QHENOMENOLOGY_224: (8)                     n = self.get_num_points()
SNGT_QHENOMENOLOGY_225: (8)                     self.resize_points(n + len(new_points))
SNGT_QHENOMENOLOGY_226: (8)                     self.data[n:] = self.data[n - 1]
SNGT_QHENOMENOLOGY_227: (8)                     self.data["point"][n:] = new_points
SNGT_QHENOMENOLOGY_228: (8)                     self.refresh_bounding_box()
SNGT_QHENOMENOLOGY_229: (8)                     return self
SNGT_QHENOMENOLOGY_230: (4)                 @affects_family_data
SNGT_QHENOMENOLOGY_231: (4)                 def reverse_points(self) -> Self:
SNGT_QHENOMENOLOGY_232: (8)                     for mob in self.get_family():
SNGT_QHENOMENOLOGY_233: (12)                        mob.data[:] = mob.data[::-1]
SNGT_QHENOMENOLOGY_234: (8)                     return self
SNGT_QHENOMENOLOGY_235: (4)                 @affects_family_data
SNGT_QHENOMENOLOGY_236: (4)                 def apply_points_function(
SNGT_QHENOMENOLOGY_237: (8)                     self,
SNGT_QHENOMENOLOGY_238: (8)                     func: Callable[[np.ndarray], np.ndarray],
SNGT_QHENOMENOLOGY_239: (8)                     about_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_240: (8)                     about_edge: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_241: (8)                     works_on_bounding_box: bool = False
SNGT_QHENOMENOLOGY_242: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_243: (8)                     if about_point is None and about_edge is not None:
SNGT_QHENOMENOLOGY_244: (12)                        about_point =
self.get_bounding_box_point(about_edge)
SNGT_QHENOMENOLOGY_245: (8)                     for mob in self.get_family():
SNGT_QHENOMENOLOGY_246: (12)                        arrs = []
SNGT_QHENOMENOLOGY_247: (12)                        if mob.has_points():
SNGT_QHENOMENOLOGY_248: (16)                            for key in mob.pointlike_data_keys:
SNGT_QHENOMENOLOGY_249: (20)                                arrs.append(mob.data[key])
SNGT_QHENOMENOLOGY_250: (12)                        if works_on_bounding_box:
SNGT_QHENOMENOLOGY_251: (16)                            arrs.append(mob.get_bounding_box())
SNGT_QHENOMENOLOGY_252: (12)                        for arr in arrs:
SNGT_QHENOMENOLOGY_253: (16)                            if about_point is None:
SNGT_QHENOMENOLOGY_254: (20)                                arr[:] = func(arr)
SNGT_QHENOMENOLOGY_255: (16)                            else:
SNGT_QHENOMENOLOGY_256: (20)                                arr[:] = func(arr - about_point) +
about_point
SNGT_QHENOMENOLOGY_257: (8)                     if not works_on_bounding_box:
SNGT_QHENOMENOLOGY_258: (12)                        self.refresh_bounding_box(recurse_down=True)
SNGT_QHENOMENOLOGY_259: (8)                     else:
SNGT_QHENOMENOLOGY_260: (12)                        for parent in self.parents:
SNGT_QHENOMENOLOGY_261: (16)                            parent.refresh_bounding_box()
```

```
SNGT_QHENOMENOLOGY_262: (8)                     return self
SNGT_QHENOMENOLOGY_263: (4)                 def match_points(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_264: (8)                     self.set_points(mobject.get_points())
SNGT_QHENOMENOLOGY_265: (8)                     return self
SNGT_QHENOMENOLOGY_266: (4)                 def get_points(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_267: (8)                     return self.data["point"]
SNGT_QHENOMENOLOGY_268: (4)                 def clear_points(self) -> Self:
SNGT_QHENOMENOLOGY_269: (8)                     self.resize_points(0)
SNGT_QHENOMENOLOGY_270: (8)                     return self
SNGT_QHENOMENOLOGY_271: (4)                 def get_num_points(self) -> int:
SNGT_QHENOMENOLOGY_272: (8)                     return len(self.get_points())
SNGT_QHENOMENOLOGY_273: (4)                 def get_all_points(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_274: (8)                     if self.submobjects:
SNGT_QHENOMENOLOGY_275: (12)                        return np.vstack([sm.get_points() for sm in
self.get_family()])
SNGT_QHENOMENOLOGY_276: (8)                     else:
SNGT_QHENOMENOLOGY_277: (12)                        return self.get_points()
SNGT_QHENOMENOLOGY_278: (4)                 def has_points(self) -> bool:
SNGT_QHENOMENOLOGY_279: (8)                     return len(self.get_points()) > 0
SNGT_QHENOMENOLOGY_280: (4)                 def get_bounding_box(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_281: (8)                     if self._needs_new_bounding_box:
SNGT_QHENOMENOLOGY_282: (12)                        self.bounding_box[:] =
self.compute_bounding_box()
SNGT_QHENOMENOLOGY_283: (12)                        self._needs_new_bounding_box = False
SNGT_QHENOMENOLOGY_284: (8)                     return self.bounding_box
SNGT_QHENOMENOLOGY_285: (4)                 def compute_bounding_box(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_286: (8)                     all_points = np.vstack([
SNGT_QHENOMENOLOGY_287: (12)                        self.get_points(),
SNGT_QHENOMENOLOGY_288: (12)                        *(
SNGT_QHENOMENOLOGY_289: (16)                            mob.get_bounding_box()
SNGT_QHENOMENOLOGY_290: (16)                            for mob in self.get_family()[1:]
SNGT_QHENOMENOLOGY_291: (16)                            if mob.has_points()
SNGT_QHENOMENOLOGY_292: (12)                        )
SNGT_QHENOMENOLOGY_293: (8)                     ])
SNGT_QHENOMENOLOGY_294: (8)                     if len(all_points) == 0:
SNGT_QHENOMENOLOGY_295: (12)                        return np.zeros((3, self.dim))
SNGT_QHENOMENOLOGY_296: (8)                     else:
SNGT_QHENOMENOLOGY_297: (12)                        mins = all_points.min(0)
SNGT_QHENOMENOLOGY_298: (12)                        maxs = all_points.max(0)
SNGT_QHENOMENOLOGY_299: (12)                        mids = (mins + maxs) / 2
SNGT_QHENOMENOLOGY_300: (12)                        return np.array([mins, mids, maxs])
SNGT_QHENOMENOLOGY_301: (4)                 def refresh_bounding_box(
SNGT_QHENOMENOLOGY_302: (8)                     self,
SNGT_QHENOMENOLOGY_303: (8)                     recurse_down: bool = False,
SNGT_QHENOMENOLOGY_304: (8)                     recurse_up: bool = True
SNGT_QHENOMENOLOGY_305: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_306: (8)                     for mob in self.get_family(recurse_down):
SNGT_QHENOMENOLOGY_307: (12)                        mob._needs_new_bounding_box = True
SNGT_QHENOMENOLOGY_308: (8)                     if recurse_up:
SNGT_QHENOMENOLOGY_309: (12)                        for parent in self.parents:
SNGT_QHENOMENOLOGY_310: (16)                            parent.refresh_bounding_box()
SNGT_QHENOMENOLOGY_311: (8)                     return self
SNGT_QHENOMENOLOGY_312: (4)                 def are_points_touching(
SNGT_QHENOMENOLOGY_313: (8)                     self,
SNGT_QHENOMENOLOGY_314: (8)                     points: Vect3Array,
SNGT_QHENOMENOLOGY_315: (8)                     buff: float = 0
SNGT_QHENOMENOLOGY_316: (4)                 ) -> np.ndarray:
SNGT_QHENOMENOLOGY_317: (8)                     bb = self.get_bounding_box()
SNGT_QHENOMENOLOGY_318: (8)                     mins = (bb[0] - buff)
SNGT_QHENOMENOLOGY_319: (8)                     maxs = (bb[2] + buff)
SNGT_QHENOMENOLOGY_320: (8)                     return ((points >= mins) * (points <= maxs)).all(1)
SNGT_QHENOMENOLOGY_321: (4)                 def is_point_touching(
SNGT_QHENOMENOLOGY_322: (8)                     self,
SNGT_QHENOMENOLOGY_323: (8)                     point: Vect3,
SNGT_QHENOMENOLOGY_324: (8)                     buff: float = 0
SNGT_QHENOMENOLOGY_325: (4)                 ) -> bool:
SNGT_QHENOMENOLOGY_326: (8)                     return self.are_points_touching(np.array(point,
ndmin=2), buff)[0]
SNGT_QHENOMENOLOGY_327: (4)                 def is_touching(self, mobject: Mobject, buff: float =
```

```
1e-2) -> bool:
SNGT_QHENOMENOLOGY_328: (8)                     bb1 = self.get_bounding_box()
SNGT_QHENOMENOLOGY_329: (8)                     bb2 = mobject.get_bounding_box()
SNGT_QHENOMENOLOGY_330: (8)                     return not any((
SNGT_QHENOMENOLOGY_331: (12)                        (bb2[2] < bb1[0] - buff).any(),  # E.g. Right
of mobject is left of self's left
SNGT_QHENOMENOLOGY_332: (12)                        (bb2[0] > bb1[2] + buff).any(),  # E.g. Left of
mobject is right of self's right
SNGT_QHENOMENOLOGY_333: (8)                     ))
SNGT_QHENOMENOLOGY_334: (4)                 def __getitem__(self, value: int | slice) -> Mobject:
SNGT_QHENOMENOLOGY_335: (8)                     if isinstance(value, slice):
SNGT_QHENOMENOLOGY_336: (12)                        GroupClass = self.get_group_class()
SNGT_QHENOMENOLOGY_337: (12)                        return
GroupClass(*self.split().__getitem__(value))
SNGT_QHENOMENOLOGY_338: (8)                     return self.split().__getitem__(value)
SNGT_QHENOMENOLOGY_339: (4)                 def __iter__(self) -> Iterator[Self]:
SNGT_QHENOMENOLOGY_340: (8)                     return iter(self.split())
SNGT_QHENOMENOLOGY_341: (4)                 def __len__(self) -> int:
SNGT_QHENOMENOLOGY_342: (8)                     return len(self.split())
SNGT_QHENOMENOLOGY_343: (4)                 def split(self) -> list[Self]:
SNGT_QHENOMENOLOGY_344: (8)                     return self.submobjects
SNGT_QHENOMENOLOGY_345: (4)                 @affects_data
SNGT_QHENOMENOLOGY_346: (4)                 def note_changed_family(self, only_changed_order=False)
-> Self:
SNGT_QHENOMENOLOGY_347: (8)                     self.family = None
SNGT_QHENOMENOLOGY_348: (8)                     if not only_changed_order:
SNGT_QHENOMENOLOGY_349: (12)                        self.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_350: (12)                        self.refresh_bounding_box()
SNGT_QHENOMENOLOGY_351: (8)                     for parent in self.parents:
SNGT_QHENOMENOLOGY_352: (12)                        parent.note_changed_family()
SNGT_QHENOMENOLOGY_353: (8)                     return self
SNGT_QHENOMENOLOGY_354: (4)                 def get_family(self, recurse: bool = True) ->
list[Mobject]:
SNGT_QHENOMENOLOGY_355: (8)                     if not recurse:
SNGT_QHENOMENOLOGY_356: (12)                        return [self]
SNGT_QHENOMENOLOGY_357: (8)                     if self.family is None:
SNGT_QHENOMENOLOGY_358: (12)                        sub_families = (sm.get_family() for sm in
self.submobjects)
SNGT_QHENOMENOLOGY_359: (12)                        self.family = [self, *it.chain(*sub_families)]
SNGT_QHENOMENOLOGY_360: (8)                     return self.family
SNGT_QHENOMENOLOGY_361: (4)                 def family_members_with_points(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_362: (8)                     return [m for m in self.get_family() if len(m.data)
> 0]
SNGT_QHENOMENOLOGY_363: (4)                 def get_ancestors(self, extended: bool = False) ->
list[Mobject]:
SNGT_QHENOMENOLOGY_364: (8)                     """
SNGT_QHENOMENOLOGY_365: (8)                     Returns parents, grandparents, etc.
SNGT_QHENOMENOLOGY_366: (8)                     Order of result should be from higher members of
the hierarchy down.
SNGT_QHENOMENOLOGY_367: (8)                     If extended is set to true, it includes the
ancestors of all family members,
SNGT_QHENOMENOLOGY_368: (8)                     e.g. any other parents of a submobject
SNGT_QHENOMENOLOGY_369: (8)                     """
SNGT_QHENOMENOLOGY_370: (8)                     ancestors = []
SNGT_QHENOMENOLOGY_371: (8)                     to_process =
list(self.get_family(recurse=extended))
SNGT_QHENOMENOLOGY_372: (8)                     excluded = set(to_process)
SNGT_QHENOMENOLOGY_373: (8)                     while to_process:
SNGT_QHENOMENOLOGY_374: (12)                        for p in to_process.pop().parents:
SNGT_QHENOMENOLOGY_375: (16)                            if p not in excluded:
SNGT_QHENOMENOLOGY_376: (20)                                ancestors.append(p)
SNGT_QHENOMENOLOGY_377: (20)                                to_process.append(p)
SNGT_QHENOMENOLOGY_378: (8)                     ancestors.reverse()
SNGT_QHENOMENOLOGY_379: (8)                     return list(dict.fromkeys(ancestors))
SNGT_QHENOMENOLOGY_380: (4)                 def add(self, *mobjects: Mobject) -> Self:
SNGT_QHENOMENOLOGY_381: (8)                     if self in mobjects:
SNGT_QHENOMENOLOGY_382: (12)                        raise Exception("Mobject cannot contain self")
SNGT_QHENOMENOLOGY_383: (8)                     for mobject in mobjects:
SNGT_QHENOMENOLOGY_384: (12)                        if mobject not in self.submobjects:
```

```
SNGT_QHENOMENOLOGY_385: (16)                            self.submobjects.append(mobject)
SNGT_QHENOMENOLOGY_386: (12)                        if self not in mobject.parents:
SNGT_QHENOMENOLOGY_387: (16)                            mobject.parents.append(self)
SNGT_QHENOMENOLOGY_388: (8)                     self.note_changed_family()
SNGT_QHENOMENOLOGY_389: (8)                     return self
SNGT_QHENOMENOLOGY_390: (4)                 def remove(
SNGT_QHENOMENOLOGY_391: (8)                     self,
SNGT_QHENOMENOLOGY_392: (8)                     *to_remove: Mobject,
SNGT_QHENOMENOLOGY_393: (8)                     reassemble: bool = True,
SNGT_QHENOMENOLOGY_394: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_395: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_396: (8)                     for parent in self.get_family(recurse):
SNGT_QHENOMENOLOGY_397: (12)                        for child in to_remove:
SNGT_QHENOMENOLOGY_398: (16)                            if child in parent.submobjects:
SNGT_QHENOMENOLOGY_399: (20)                                parent.submobjects.remove(child)
SNGT_QHENOMENOLOGY_400: (16)                            if parent in child.parents:
SNGT_QHENOMENOLOGY_401: (20)                                child.parents.remove(parent)
SNGT_QHENOMENOLOGY_402: (12)                        if reassemble:
SNGT_QHENOMENOLOGY_403: (16)                            parent.note_changed_family()
SNGT_QHENOMENOLOGY_404: (8)                     return self
SNGT_QHENOMENOLOGY_405: (4)                 def clear(self) -> Self:
SNGT_QHENOMENOLOGY_406: (8)                     self.remove(*self.submobjects, recurse=False)
SNGT_QHENOMENOLOGY_407: (8)                     return self
SNGT_QHENOMENOLOGY_408: (4)                 def add_to_back(self, *mobjects: Mobject) -> Self:
SNGT_QHENOMENOLOGY_409: (8)                     self.set_submobjects(list_update(mobjects,
self.submobjects))
SNGT_QHENOMENOLOGY_410: (8)                     return self
SNGT_QHENOMENOLOGY_411: (4)                 def replace_submobject(self, index: int, new_submob:
Mobject) -> Self:
SNGT_QHENOMENOLOGY_412: (8)                     old_submob = self.submobjects[index]
SNGT_QHENOMENOLOGY_413: (8)                     if self in old_submob.parents:
SNGT_QHENOMENOLOGY_414: (12)                        old_submob.parents.remove(self)
SNGT_QHENOMENOLOGY_415: (8)                     self.submobjects[index] = new_submob
SNGT_QHENOMENOLOGY_416: (8)                     new_submob.parents.append(self)
SNGT_QHENOMENOLOGY_417: (8)                     self.note_changed_family()
SNGT_QHENOMENOLOGY_418: (8)                     return self
SNGT_QHENOMENOLOGY_419: (4)                 def insert_submobject(self, index: int, new_submob:
Mobject) -> Self:
SNGT_QHENOMENOLOGY_420: (8)                     self.submobjects.insert(index, new_submob)
SNGT_QHENOMENOLOGY_421: (8)                     self.note_changed_family()
SNGT_QHENOMENOLOGY_422: (8)                     return self
SNGT_QHENOMENOLOGY_423: (4)                 def set_submobjects(self, submobject_list:
list[Mobject]) -> Self:
SNGT_QHENOMENOLOGY_424: (8)                     if self.submobjects == submobject_list:
SNGT_QHENOMENOLOGY_425: (12)                        return self
SNGT_QHENOMENOLOGY_426: (8)                     self.clear()
SNGT_QHENOMENOLOGY_427: (8)                     self.add(*submobject_list)
SNGT_QHENOMENOLOGY_428: (8)                     return self
SNGT_QHENOMENOLOGY_429: (4)                 def digest_mobject_attrs(self) -> Self:
SNGT_QHENOMENOLOGY_430: (8)                     """
SNGT_QHENOMENOLOGY_431: (8)                     Ensures all attributes which are mobjects are
included
SNGT_QHENOMENOLOGY_432: (8)                     in the submobjects list.
SNGT_QHENOMENOLOGY_433: (8)                     """
SNGT_QHENOMENOLOGY_434: (8)                     mobject_attrs = [x for x in
list(self.__dict__.values()) if isinstance(x, Mobject)]
SNGT_QHENOMENOLOGY_435: (8)                     self.set_submobjects(list_update(self.submobjects,
mobject_attrs))
SNGT_QHENOMENOLOGY_436: (8)                     return self
SNGT_QHENOMENOLOGY_437: (4)                 def arrange(
SNGT_QHENOMENOLOGY_438: (8)                     self,
SNGT_QHENOMENOLOGY_439: (8)                     direction: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_440: (8)                     center: bool = True,
SNGT_QHENOMENOLOGY_441: (8)                     **kwargs
SNGT_QHENOMENOLOGY_442: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_443: (8)                     for m1, m2 in zip(self.submobjects,
self.submobjects[1:]):
SNGT_QHENOMENOLOGY_444: (12)                        m2.next_to(m1, direction, **kwargs)
SNGT_QHENOMENOLOGY_445: (8)                     if center:
```

```
SNGT_QHENOMENOLOGY_446: (12)                     self.center()
SNGT_QHENOMENOLOGY_447: (8)                 return self
SNGT_QHENOMENOLOGY_448: (4)             def arrange_in_grid(
SNGT_QHENOMENOLOGY_449: (8)                 self,
SNGT_QHENOMENOLOGY_450: (8)                 n_rows: int | None = None,
SNGT_QHENOMENOLOGY_451: (8)                 n_cols: int | None = None,
SNGT_QHENOMENOLOGY_452: (8)                 buff: float | None = None,
SNGT_QHENOMENOLOGY_453: (8)                 h_buff: float | None = None,
SNGT_QHENOMENOLOGY_454: (8)                 v_buff: float | None = None,
SNGT_QHENOMENOLOGY_455: (8)                 buff_ratio: float | None = None,
SNGT_QHENOMENOLOGY_456: (8)                 h_buff_ratio: float = 0.5,
SNGT_QHENOMENOLOGY_457: (8)                 v_buff_ratio: float = 0.5,
SNGT_QHENOMENOLOGY_458: (8)                 aligned_edge: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_459: (8)                 fill_rows_first: bool = True
SNGT_QHENOMENOLOGY_460: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_461: (8)                 submobs = self.submobjects
SNGT_QHENOMENOLOGY_462: (8)                 n_submobs = len(submobs)
SNGT_QHENOMENOLOGY_463: (8)                 if n_rows is None:
SNGT_QHENOMENOLOGY_464: (12)                     n_rows = int(np.sqrt(n_submobs)) if n_cols is
None else n_submobs // n_cols
SNGT_QHENOMENOLOGY_465: (8)                 if n_cols is None:
SNGT_QHENOMENOLOGY_466: (12)                     n_cols = n_submobs // n_rows
SNGT_QHENOMENOLOGY_467: (8)                 if buff is not None:
SNGT_QHENOMENOLOGY_468: (12)                     h_buff = buff
SNGT_QHENOMENOLOGY_469: (12)                     v_buff = buff
SNGT_QHENOMENOLOGY_470: (8)                 else:
SNGT_QHENOMENOLOGY_471: (12)                     if buff_ratio is not None:
SNGT_QHENOMENOLOGY_472: (16)                         v_buff_ratio = buff_ratio
SNGT_QHENOMENOLOGY_473: (16)                         h_buff_ratio = buff_ratio
SNGT_QHENOMENOLOGY_474: (12)                     if h_buff is None:
SNGT_QHENOMENOLOGY_475: (16)                         h_buff = h_buff_ratio * self[0].get_width()
SNGT_QHENOMENOLOGY_476: (12)                     if v_buff is None:
SNGT_QHENOMENOLOGY_477: (16)                         v_buff = v_buff_ratio *
self[0].get_height()
SNGT_QHENOMENOLOGY_478: (8)                 x_unit = h_buff + max([sm.get_width() for sm in
submobs])
SNGT_QHENOMENOLOGY_479: (8)                 y_unit = v_buff + max([sm.get_height() for sm in
submobs])
SNGT_QHENOMENOLOGY_480: (8)                 for index, sm in enumerate(submobs):
SNGT_QHENOMENOLOGY_481: (12)                     if fill_rows_first:
SNGT_QHENOMENOLOGY_482: (16)                         x, y = index % n_cols, index // n_cols
SNGT_QHENOMENOLOGY_483: (12)                     else:
SNGT_QHENOMENOLOGY_484: (16)                         x, y = index // n_rows, index % n_rows
SNGT_QHENOMENOLOGY_485: (12)                     sm.move_to(ORIGIN, aligned_edge)
SNGT_QHENOMENOLOGY_486: (12)                     sm.shift(x * x_unit * RIGHT + y * y_unit *
DOWN)
SNGT_QHENOMENOLOGY_487: (8)                 self.center()
SNGT_QHENOMENOLOGY_488: (8)                 return self
SNGT_QHENOMENOLOGY_489: (4)             def arrange_to_fit_dim(self, length: float, dim: int,
about_edge=ORIGIN) -> Self:
SNGT_QHENOMENOLOGY_490: (8)                 ref_point = self.get_bounding_box_point(about_edge)
SNGT_QHENOMENOLOGY_491: (8)                 n_submobs = len(self.submobjects)
SNGT_QHENOMENOLOGY_492: (8)                 if n_submobs <= 1:
SNGT_QHENOMENOLOGY_493: (12)                     return
SNGT_QHENOMENOLOGY_494: (8)                 total_length = sum(sm.length_over_dim(dim) for sm
in self.submobjects)
SNGT_QHENOMENOLOGY_495: (8)                 buff = (length - total_length) / (n_submobs - 1)
SNGT_QHENOMENOLOGY_496: (8)                 vect = np.zeros(self.dim)
SNGT_QHENOMENOLOGY_497: (8)                 vect[dim] = 1
SNGT_QHENOMENOLOGY_498: (8)                 x = 0
SNGT_QHENOMENOLOGY_499: (8)                 for submob in self.submobjects:
SNGT_QHENOMENOLOGY_500: (12)                     submob.set_coord(x, dim, -vect)
SNGT_QHENOMENOLOGY_501: (12)                     x += submob.length_over_dim(dim) + buff
SNGT_QHENOMENOLOGY_502: (8)                 self.move_to(ref_point, about_edge)
SNGT_QHENOMENOLOGY_503: (8)                 return self
SNGT_QHENOMENOLOGY_504: (4)             def arrange_to_fit_width(self, width: float,
about_edge=ORIGIN) -> Self:
SNGT_QHENOMENOLOGY_505: (8)                 return self.arrange_to_fit_dim(width, 0,
about_edge)
```

```
SNGT_QHENOMENOLOGY_506: (4)                 def arrange_to_fit_height(self, height: float,
about_edge=ORIGIN) -> Self:
SNGT_QHENOMENOLOGY_507: (8)                     return self.arrange_to_fit_dim(height, 1,
about_edge)
SNGT_QHENOMENOLOGY_508: (4)                 def arrange_to_fit_depth(self, depth: float,
about_edge=ORIGIN) -> Self:
SNGT_QHENOMENOLOGY_509: (8)                     return self.arrange_to_fit_dim(depth, 2,
about_edge)
SNGT_QHENOMENOLOGY_510: (4)                 def sort(
SNGT_QHENOMENOLOGY_511: (8)                     self,
SNGT_QHENOMENOLOGY_512: (8)                     point_to_num_func: Callable[[np.ndarray], float] =
lambda p: p[0],
SNGT_QHENOMENOLOGY_513: (8)                     submob_func: Callable[[Mobject]] | None = None
SNGT_QHENOMENOLOGY_514: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_515: (8)                     if submob_func is not None:
SNGT_QHENOMENOLOGY_516: (12)                        self.submobjects.sort(key=submob_func)
SNGT_QHENOMENOLOGY_517: (8)                     else:
SNGT_QHENOMENOLOGY_518: (12)                        self.submobjects.sort(key=lambda m:
point_to_num_func(m.get_center()))
SNGT_QHENOMENOLOGY_519: (8)                     self.note_changed_family(only_changed_order=True)
SNGT_QHENOMENOLOGY_520: (8)                     return self
SNGT_QHENOMENOLOGY_521: (4)                 def shuffle(self, recurse: bool = False) -> Self:
SNGT_QHENOMENOLOGY_522: (8)                     if recurse:
SNGT_QHENOMENOLOGY_523: (12)                        for submob in self.submobjects:
SNGT_QHENOMENOLOGY_524: (16)                            submob.shuffle(recurse=True)
SNGT_QHENOMENOLOGY_525: (8)                     random.shuffle(self.submobjects)
SNGT_QHENOMENOLOGY_526: (8)                     self.note_changed_family(only_changed_order=True)
SNGT_QHENOMENOLOGY_527: (8)                     return self
SNGT_QHENOMENOLOGY_528: (4)                 def reverse_submobjects(self) -> Self:
SNGT_QHENOMENOLOGY_529: (8)                     self.submobjects.reverse()
SNGT_QHENOMENOLOGY_530: (8)                     self.note_changed_family(only_changed_order=True)
SNGT_QHENOMENOLOGY_531: (8)                     return self
SNGT_QHENOMENOLOGY_532: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_533: (4)                 def stash_mobject_pointers(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_534: (8)                     @wraps(func)
SNGT_QHENOMENOLOGY_535: (8)                     def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_536: (12)                        uncopied_attrs = ["parents", "target",
"saved_state"]
SNGT_QHENOMENOLOGY_537: (12)                        stash = dict()
SNGT_QHENOMENOLOGY_538: (12)                        for attr in uncopied_attrs:
SNGT_QHENOMENOLOGY_539: (16)                            if hasattr(self, attr):
SNGT_QHENOMENOLOGY_540: (20)                                value = getattr(self, attr)
SNGT_QHENOMENOLOGY_541: (20)                                stash[attr] = value
SNGT_QHENOMENOLOGY_542: (20)                                null_value = [] if isinstance(value,
list) else None
SNGT_QHENOMENOLOGY_543: (20)                                setattr(self, attr, null_value)
SNGT_QHENOMENOLOGY_544: (12)                        result = func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_545: (12)                        self.__dict__.update(stash)
SNGT_QHENOMENOLOGY_546: (12)                        return result
SNGT_QHENOMENOLOGY_547: (8)                     return wrapper
SNGT_QHENOMENOLOGY_548: (4)                 @stash_mobject_pointers
SNGT_QHENOMENOLOGY_549: (4)                 def serialize(self) -> bytes:
SNGT_QHENOMENOLOGY_550: (8)                     return pickle.dumps(self)
SNGT_QHENOMENOLOGY_551: (4)                 def deserialize(self, data: bytes) -> Self:
SNGT_QHENOMENOLOGY_552: (8)                     self.become(pickle.loads(data))
SNGT_QHENOMENOLOGY_553: (8)                     return self
SNGT_QHENOMENOLOGY_554: (4)                 @stash_mobject_pointers
SNGT_QHENOMENOLOGY_555: (4)                 def deepcopy(self) -> Self:
SNGT_QHENOMENOLOGY_556: (8)                     return copy.deepcopy(self)
SNGT_QHENOMENOLOGY_557: (4)                 def copy(self, deep: bool = False) -> Self:
SNGT_QHENOMENOLOGY_558: (8)                     if deep:
SNGT_QHENOMENOLOGY_559: (12)                        return self.deepcopy()
SNGT_QHENOMENOLOGY_560: (8)                     result = copy.copy(self)
SNGT_QHENOMENOLOGY_561: (8)                     result.parents = []
SNGT_QHENOMENOLOGY_562: (8)                     result.target = None
SNGT_QHENOMENOLOGY_563: (8)                     result.saved_state = None
SNGT_QHENOMENOLOGY_564: (8)                     result.uniforms = {
SNGT_QHENOMENOLOGY_565: (12)                        key: value.copy() if isinstance(value,
```

```
np.ndarray) else value
SNGT_QHENOMENOLOGY_566: (12)                     for key, value in self.uniforms.items()
SNGT_QHENOMENOLOGY_567: (8)                 }
SNGT_QHENOMENOLOGY_568: (8)                 result.submobjects = [sm.copy() for sm in
self.submobjects]
SNGT_QHENOMENOLOGY_569: (8)                 for sm in result.submobjects:
SNGT_QHENOMENOLOGY_570: (12)                     sm.parents = [result]
SNGT_QHENOMENOLOGY_571: (8)                 result.family = [result, *it.chain(*
(sm.get_family() for sm in result.submobjects))]
SNGT_QHENOMENOLOGY_572: (8)                 result.updaters = list(self.updaters)
SNGT_QHENOMENOLOGY_573: (8)                 result._data_has_changed = True
SNGT_QHENOMENOLOGY_574: (8)                 result.shader_wrapper = None
SNGT_QHENOMENOLOGY_575: (8)                 family = self.get_family()
SNGT_QHENOMENOLOGY_576: (8)                 for attr, value in self.__dict__.items():
SNGT_QHENOMENOLOGY_577: (12)                     if isinstance(value, Mobject) and value is not
self:
SNGT_QHENOMENOLOGY_578: (16)                         if value in family:
SNGT_QHENOMENOLOGY_579: (20)                             setattr(result, attr,
result.family[family.index(value)])
SNGT_QHENOMENOLOGY_580: (12)                     elif isinstance(value, np.ndarray):
SNGT_QHENOMENOLOGY_581: (16)                         setattr(result, attr, value.copy())
SNGT_QHENOMENOLOGY_582: (8)                 return result
SNGT_QHENOMENOLOGY_583: (4)             def generate_target(self, use_deepcopy: bool = False) -
> Self:
SNGT_QHENOMENOLOGY_584: (8)                 self.target = self.copy(deep=use_deepcopy)
SNGT_QHENOMENOLOGY_585: (8)                 self.target.saved_state = self.saved_state
SNGT_QHENOMENOLOGY_586: (8)                 return self.target
SNGT_QHENOMENOLOGY_587: (4)             def save_state(self, use_deepcopy: bool = False) ->
Self:
SNGT_QHENOMENOLOGY_588: (8)                 self.saved_state = self.copy(deep=use_deepcopy)
SNGT_QHENOMENOLOGY_589: (8)                 self.saved_state.target = self.target
SNGT_QHENOMENOLOGY_590: (8)                 return self
SNGT_QHENOMENOLOGY_591: (4)             def restore(self) -> Self:
SNGT_QHENOMENOLOGY_592: (8)                 if not hasattr(self, "saved_state") or
self.saved_state is None:
SNGT_QHENOMENOLOGY_593: (12)                     raise Exception("Trying to restore without
having saved")
SNGT_QHENOMENOLOGY_594: (8)                 self.become(self.saved_state)
SNGT_QHENOMENOLOGY_595: (8)                 return self
SNGT_QHENOMENOLOGY_596: (4)             def become(self, mobject: Mobject,
match_updaters=False) -> Self:
SNGT_QHENOMENOLOGY_597: (8)                 """
SNGT_QHENOMENOLOGY_598: (8)                 Edit all data and submobjects to be idential
SNGT_QHENOMENOLOGY_599: (8)                 to another mobject
SNGT_QHENOMENOLOGY_600: (8)                 """
SNGT_QHENOMENOLOGY_601: (8)                 self.align_family(mobject)
SNGT_QHENOMENOLOGY_602: (8)                 family1 = self.get_family()
SNGT_QHENOMENOLOGY_603: (8)                 family2 = mobject.get_family()
SNGT_QHENOMENOLOGY_604: (8)                 for sm1, sm2 in zip(family1, family2):
SNGT_QHENOMENOLOGY_605: (12)                     sm1.set_data(sm2.data)
SNGT_QHENOMENOLOGY_606: (12)                     sm1.set_uniforms(sm2.uniforms)
SNGT_QHENOMENOLOGY_607: (12)                     sm1.bounding_box[:] = sm2.bounding_box
SNGT_QHENOMENOLOGY_608: (12)                     sm1.shader_folder = sm2.shader_folder
SNGT_QHENOMENOLOGY_609: (12)                     sm1.texture_paths = sm2.texture_paths
SNGT_QHENOMENOLOGY_610: (12)                     sm1.depth_test = sm2.depth_test
SNGT_QHENOMENOLOGY_611: (12)                     sm1.render_primitive = sm2.render_primitive
SNGT_QHENOMENOLOGY_612: (12)                     sm1._needs_new_bounding_box =
sm2._needs_new_bounding_box
SNGT_QHENOMENOLOGY_613: (8)                 for attr, value in list(mobject.__dict__.items()):
SNGT_QHENOMENOLOGY_614: (12)                     if isinstance(value, Mobject) and value in
family2:
SNGT_QHENOMENOLOGY_615: (16)                         setattr(self, attr,
family1[family2.index(value)])
SNGT_QHENOMENOLOGY_616: (8)                 if match_updaters:
SNGT_QHENOMENOLOGY_617: (12)                     self.match_updaters(mobject)
SNGT_QHENOMENOLOGY_618: (8)                 return self
SNGT_QHENOMENOLOGY_619: (4)             def looks_identical(self, mobject: Mobject) -> bool:
SNGT_QHENOMENOLOGY_620: (8)                 fam1 = self.family_members_with_points()
SNGT_QHENOMENOLOGY_621: (8)                 fam2 = mobject.family_members_with_points()
```

```
SNGT_QHENOMENOLOGY_622: (8)                     if len(fam1) != len(fam2):
SNGT_QHENOMENOLOGY_623: (12)                        return False
SNGT_QHENOMENOLOGY_624: (8)                     for m1, m2 in zip(fam1, fam2):
SNGT_QHENOMENOLOGY_625: (12)                        if m1.get_num_points() != m2.get_num_points():
SNGT_QHENOMENOLOGY_626: (16)                            return False
SNGT_QHENOMENOLOGY_627: (12)                        if not m1.data.dtype == m2.data.dtype:
SNGT_QHENOMENOLOGY_628: (16)                            return False
SNGT_QHENOMENOLOGY_629: (12)                        for key in m1.data.dtype.names:
SNGT_QHENOMENOLOGY_630: (16)                            if not np.isclose(m1.data[key],
m2.data[key]).all():
SNGT_QHENOMENOLOGY_631: (20)                                return False
SNGT_QHENOMENOLOGY_632: (12)                        if set(m1.uniforms).difference(m2.uniforms):
SNGT_QHENOMENOLOGY_633: (16)                            return False
SNGT_QHENOMENOLOGY_634: (12)                        for key in m1.uniforms:
SNGT_QHENOMENOLOGY_635: (16)                            value1 = m1.uniforms[key]
SNGT_QHENOMENOLOGY_636: (16)                            value2 = m2.uniforms[key]
SNGT_QHENOMENOLOGY_637: (16)                            if isinstance(value1, np.ndarray) and
isinstance(value2, np.ndarray) and not value1.size == value2.size:
SNGT_QHENOMENOLOGY_638: (20)                                return False
SNGT_QHENOMENOLOGY_639: (16)                            if not np.isclose(value1, value2).all():
SNGT_QHENOMENOLOGY_640: (20)                                return False
SNGT_QHENOMENOLOGY_641: (8)                     return True
SNGT_QHENOMENOLOGY_642: (4)                 def has_same_shape_as(self, mobject: Mobject) -> bool:
SNGT_QHENOMENOLOGY_643: (8)                     points1, points2 = (
SNGT_QHENOMENOLOGY_644: (12)                        (m.get_all_points() - m.get_center()) /
m.get_height()
SNGT_QHENOMENOLOGY_645: (12)                        for m in (self, mobject)
SNGT_QHENOMENOLOGY_646: (8)                     )
SNGT_QHENOMENOLOGY_647: (8)                     if len(points1) != len(points2):
SNGT_QHENOMENOLOGY_648: (12)                        return False
SNGT_QHENOMENOLOGY_649: (8)                     return bool(np.isclose(points1, points2,
atol=self.get_width() * 1e-2).all())
SNGT_QHENOMENOLOGY_650: (4)                 def replicate(self, n: int) -> Self:
SNGT_QHENOMENOLOGY_651: (8)                     group_class = self.get_group_class()
SNGT_QHENOMENOLOGY_652: (8)                     return group_class(*(self.copy() for _ in
range(n)))
SNGT_QHENOMENOLOGY_653: (4)                 def get_grid(
SNGT_QHENOMENOLOGY_654: (8)                     self,
SNGT_QHENOMENOLOGY_655: (8)                     n_rows: int,
SNGT_QHENOMENOLOGY_656: (8)                     n_cols: int,
SNGT_QHENOMENOLOGY_657: (8)                     height: float | None = None,
SNGT_QHENOMENOLOGY_658: (8)                     width: float | None = None,
SNGT_QHENOMENOLOGY_659: (8)                     group_by_rows: bool = False,
SNGT_QHENOMENOLOGY_660: (8)                     group_by_cols: bool = False,
SNGT_QHENOMENOLOGY_661: (8)                     **kwargs
SNGT_QHENOMENOLOGY_662: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_663: (8)                     """
SNGT_QHENOMENOLOGY_664: (8)                     Returns a new mobject containing multiple copies of
this one
SNGT_QHENOMENOLOGY_665: (8)                     arranged in a grid
SNGT_QHENOMENOLOGY_666: (8)                     """
SNGT_QHENOMENOLOGY_667: (8)                     total = n_rows * n_cols
SNGT_QHENOMENOLOGY_668: (8)                     grid = self.replicate(total)
SNGT_QHENOMENOLOGY_669: (8)                     if group_by_cols:
SNGT_QHENOMENOLOGY_670: (12)                        kwargs["fill_rows_first"] = False
SNGT_QHENOMENOLOGY_671: (8)                     grid.arrange_in_grid(n_rows, n_cols, **kwargs)
SNGT_QHENOMENOLOGY_672: (8)                     if height is not None:
SNGT_QHENOMENOLOGY_673: (12)                        grid.set_height(height)
SNGT_QHENOMENOLOGY_674: (8)                     if width is not None:
SNGT_QHENOMENOLOGY_675: (12)                        grid.set_height(width)
SNGT_QHENOMENOLOGY_676: (8)                     group_class = self.get_group_class()
SNGT_QHENOMENOLOGY_677: (8)                     if group_by_rows:
SNGT_QHENOMENOLOGY_678: (12)                        return group_class(*(grid[n:n + n_cols] for n
in range(0, total, n_cols)))
SNGT_QHENOMENOLOGY_679: (8)                     elif group_by_cols:
SNGT_QHENOMENOLOGY_680: (12)                        return group_class(*(grid[n:n + n_rows] for n
in range(0, total, n_rows)))
SNGT_QHENOMENOLOGY_681: (8)                     else:
SNGT_QHENOMENOLOGY_682: (12)                        return grid
```

```
SNGT_QHENOMENOLOGY_683: (4)                 def init_updaters(self):
SNGT_QHENOMENOLOGY_684: (8)                     self.updaters: list[Updater] = list()
SNGT_QHENOMENOLOGY_685: (8)                     self._has_updaters_in_family: Optional[bool] =
False
SNGT_QHENOMENOLOGY_686: (8)                     self.updating_suspended: bool = False
SNGT_QHENOMENOLOGY_687: (4)                 def update(self, dt: float = 0, recurse: bool = True) -
> Self:
SNGT_QHENOMENOLOGY_688: (8)                     if not self.has_updaters() or
self.updating_suspended:
SNGT_QHENOMENOLOGY_689: (12)                        return self
SNGT_QHENOMENOLOGY_690: (8)                     if recurse:
SNGT_QHENOMENOLOGY_691: (12)                        for submob in self.submobjects:
SNGT_QHENOMENOLOGY_692: (16)                            submob.update(dt, recurse)
SNGT_QHENOMENOLOGY_693: (8)                     for updater in self.updaters:
SNGT_QHENOMENOLOGY_694: (12)                        if "dt" in updater.__code__.co_varnames:
SNGT_QHENOMENOLOGY_695: (16)                            updater(self, dt=dt)
SNGT_QHENOMENOLOGY_696: (12)                        else:
SNGT_QHENOMENOLOGY_697: (16)                            updater(self)
SNGT_QHENOMENOLOGY_698: (8)                     return self
SNGT_QHENOMENOLOGY_699: (4)                 def get_updaters(self) -> list[Updater]:
SNGT_QHENOMENOLOGY_700: (8)                     return self.updaters
SNGT_QHENOMENOLOGY_701: (4)                 def add_updater(self, update_func: Updater, call: bool
= True) -> Self:
SNGT_QHENOMENOLOGY_702: (8)                     self.updaters.append(update_func)
SNGT_QHENOMENOLOGY_703: (8)                     if call:
SNGT_QHENOMENOLOGY_704: (12)                        self.update(dt=0)
SNGT_QHENOMENOLOGY_705: (8)                     self.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_706: (8)                     return self
SNGT_QHENOMENOLOGY_707: (4)                 def insert_updater(self, update_func: Updater,
index=0):
SNGT_QHENOMENOLOGY_708: (8)                     self.updaters.insert(index, update_func)
SNGT_QHENOMENOLOGY_709: (8)                     self.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_710: (8)                     return self
SNGT_QHENOMENOLOGY_711: (4)                 def remove_updater(self, update_func: Updater) -> Self:
SNGT_QHENOMENOLOGY_712: (8)                     while update_func in self.updaters:
SNGT_QHENOMENOLOGY_713: (12)                        self.updaters.remove(update_func)
SNGT_QHENOMENOLOGY_714: (8)                     self.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_715: (8)                     return self
SNGT_QHENOMENOLOGY_716: (4)                 def clear_updaters(self, recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_717: (8)                     for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_718: (12)                        mob.updaters = []
SNGT_QHENOMENOLOGY_719: (12)                        mob._has_updaters_in_family = False
SNGT_QHENOMENOLOGY_720: (8)                     for parent in self.get_ancestors():
SNGT_QHENOMENOLOGY_721: (12)                        parent._has_updaters_in_family = False
SNGT_QHENOMENOLOGY_722: (8)                     return self
SNGT_QHENOMENOLOGY_723: (4)                 def match_updaters(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_724: (8)                     self.updaters = list(mobject.updaters)
SNGT_QHENOMENOLOGY_725: (8)                     self.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_726: (8)                     return self
SNGT_QHENOMENOLOGY_727: (4)                 def suspend_updating(self, recurse: bool = True) ->
Self:
SNGT_QHENOMENOLOGY_728: (8)                     self.updating_suspended = True
SNGT_QHENOMENOLOGY_729: (8)                     if recurse:
SNGT_QHENOMENOLOGY_730: (12)                        for submob in self.submobjects:
SNGT_QHENOMENOLOGY_731: (16)                            submob.suspend_updating(recurse)
SNGT_QHENOMENOLOGY_732: (8)                     return self
SNGT_QHENOMENOLOGY_733: (4)                 def resume_updating(self, recurse: bool = True,
call_updater: bool = True) -> Self:
SNGT_QHENOMENOLOGY_734: (8)                     self.updating_suspended = False
SNGT_QHENOMENOLOGY_735: (8)                     if recurse:
SNGT_QHENOMENOLOGY_736: (12)                        for submob in self.submobjects:
SNGT_QHENOMENOLOGY_737: (16)                            submob.resume_updating(recurse)
SNGT_QHENOMENOLOGY_738: (8)                     for parent in self.parents:
SNGT_QHENOMENOLOGY_739: (12)                        parent.resume_updating(recurse=False,
call_updater=False)
SNGT_QHENOMENOLOGY_740: (8)                     if call_updater:
SNGT_QHENOMENOLOGY_741: (12)                        self.update(dt=0, recurse=recurse)
SNGT_QHENOMENOLOGY_742: (8)                     return self
SNGT_QHENOMENOLOGY_743: (4)                 def has_updaters(self) -> bool:
```

```
SNGT_QHENOMENOLOGY_744: (8)                     if self._has_updaters_in_family is None:
SNGT_QHENOMENOLOGY_745: (12)                         self._has_updaters_in_family =
bool(self.updaters) or any(
SNGT_QHENOMENOLOGY_746: (16)                             sm.has_updaters() for sm in
self.submobjects
SNGT_QHENOMENOLOGY_747: (12)                         )
SNGT_QHENOMENOLOGY_748: (8)                     return self._has_updaters_in_family
SNGT_QHENOMENOLOGY_749: (4)                 def refresh_has_updater_status(self) -> Self:
SNGT_QHENOMENOLOGY_750: (8)                     self._has_updaters_in_family = None
SNGT_QHENOMENOLOGY_751: (8)                     for parent in self.parents:
SNGT_QHENOMENOLOGY_752: (12)                         parent.refresh_has_updater_status()
SNGT_QHENOMENOLOGY_753: (8)                     return self
SNGT_QHENOMENOLOGY_754: (4)                 def is_changing(self) -> bool:
SNGT_QHENOMENOLOGY_755: (8)                     return self._is_animating or self.has_updaters()
SNGT_QHENOMENOLOGY_756: (4)                 def set_animating_status(self, is_animating: bool,
recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_757: (8)                     for mob in (*self.get_family(recurse),
*self.get_ancestors()):
SNGT_QHENOMENOLOGY_758: (12)                         mob._is_animating = is_animating
SNGT_QHENOMENOLOGY_759: (8)                     return self
SNGT_QHENOMENOLOGY_760: (4)                 def shift(self, vector: Vect3) -> Self:
SNGT_QHENOMENOLOGY_761: (8)                     self.apply_points_function(
SNGT_QHENOMENOLOGY_762: (12)                         lambda points: points + vector,
SNGT_QHENOMENOLOGY_763: (12)                         about_edge=None,
SNGT_QHENOMENOLOGY_764: (12)                         works_on_bounding_box=True,
SNGT_QHENOMENOLOGY_765: (8)                     )
SNGT_QHENOMENOLOGY_766: (8)                     return self
SNGT_QHENOMENOLOGY_767: (4)                 def scale(
SNGT_QHENOMENOLOGY_768: (8)                     self,
SNGT_QHENOMENOLOGY_769: (8)                     scale_factor: float | npt.ArrayLike,
SNGT_QHENOMENOLOGY_770: (8)                     min_scale_factor: float = 1e-8,
SNGT_QHENOMENOLOGY_771: (8)                     about_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_772: (8)                     about_edge: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_773: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_774: (8)                     """
SNGT_QHENOMENOLOGY_775: (8)                     Default behavior is to scale about the center of
the mobject.
SNGT_QHENOMENOLOGY_776: (8)                     The argument about_edge can be a vector, indicating
which side of
SNGT_QHENOMENOLOGY_777: (8)                     the mobject to scale about, e.g.,
mob.scale(about_edge = RIGHT)
SNGT_QHENOMENOLOGY_778: (8)                     scales about mob.get_right().
SNGT_QHENOMENOLOGY_779: (8)                     Otherwise, if about_point is given a value, scaling
is done with
SNGT_QHENOMENOLOGY_780: (8)                     respect to that point.
SNGT_QHENOMENOLOGY_781: (8)                     """
SNGT_QHENOMENOLOGY_782: (8)                     if isinstance(scale_factor, numbers.Number):
SNGT_QHENOMENOLOGY_783: (12)                         scale_factor = max(scale_factor,
min_scale_factor)
SNGT_QHENOMENOLOGY_784: (8)                     else:
SNGT_QHENOMENOLOGY_785: (12)                         scale_factor =
np.array(scale_factor).clip(min=min_scale_factor)
SNGT_QHENOMENOLOGY_786: (8)                     self.apply_points_function(
SNGT_QHENOMENOLOGY_787: (12)                         lambda points: scale_factor * points,
SNGT_QHENOMENOLOGY_788: (12)                         about_point=about_point,
SNGT_QHENOMENOLOGY_789: (12)                         about_edge=about_edge,
SNGT_QHENOMENOLOGY_790: (12)                         works_on_bounding_box=True,
SNGT_QHENOMENOLOGY_791: (8)                     )
SNGT_QHENOMENOLOGY_792: (8)                     for mob in self.get_family():
SNGT_QHENOMENOLOGY_793: (12)                         mob._handle_scale_side_effects(scale_factor)
SNGT_QHENOMENOLOGY_794: (8)                     return self
SNGT_QHENOMENOLOGY_795: (4)                 def _handle_scale_side_effects(self, scale_factor):
SNGT_QHENOMENOLOGY_796: (8)                     pass
SNGT_QHENOMENOLOGY_797: (4)                 def stretch(self, factor: float, dim: int, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_798: (8)                     def func(points):
SNGT_QHENOMENOLOGY_799: (12)                         points[:, dim] *= factor
SNGT_QHENOMENOLOGY_800: (12)                         return points
SNGT_QHENOMENOLOGY_801: (8)                     self.apply_points_function(func,
```

```
works_on_bounding_box=True, **kwargs)
SNGT_QHENOMENOLOGY_802: (8)                     return self
SNGT_QHENOMENOLOGY_803: (4)                 def rotate_about_origin(self, angle: float, axis: Vect3
= OUT) -> Self:
SNGT_QHENOMENOLOGY_804: (8)                     return self.rotate(angle, axis, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_805: (4)                 def rotate(
SNGT_QHENOMENOLOGY_806: (8)                     self,
SNGT_QHENOMENOLOGY_807: (8)                     angle: float,
SNGT_QHENOMENOLOGY_808: (8)                     axis: Vect3 = OUT,
SNGT_QHENOMENOLOGY_809: (8)                     about_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_810: (8)                     **kwargs
SNGT_QHENOMENOLOGY_811: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_812: (8)                     rot_matrix_T = rotation_matrix_transpose(angle,
axis)
SNGT_QHENOMENOLOGY_813: (8)                     self.apply_points_function(
SNGT_QHENOMENOLOGY_814: (12)                        lambda points: np.dot(points, rot_matrix_T),
SNGT_QHENOMENOLOGY_815: (12)                        about_point,
SNGT_QHENOMENOLOGY_816: (12)                        **kwargs
SNGT_QHENOMENOLOGY_817: (8)                     )
SNGT_QHENOMENOLOGY_818: (8)                     return self
SNGT_QHENOMENOLOGY_819: (4)                 def flip(self, axis: Vect3 = UP, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_820: (8)                     return self.rotate(TAU / 2, axis, **kwargs)
SNGT_QHENOMENOLOGY_821: (4)                 def apply_function(self, function:
Callable[[np.ndarray], np.ndarray], **kwargs) -> Self:
SNGT_QHENOMENOLOGY_822: (8)                     if len(kwargs) == 0:
SNGT_QHENOMENOLOGY_823: (12)                        kwargs["about_point"] = ORIGIN
SNGT_QHENOMENOLOGY_824: (8)                     self.apply_points_function(
SNGT_QHENOMENOLOGY_825: (12)                        lambda points: np.array([function(p) for p in
points]),
SNGT_QHENOMENOLOGY_826: (12)                        **kwargs
SNGT_QHENOMENOLOGY_827: (8)                     )
SNGT_QHENOMENOLOGY_828: (8)                     return self
SNGT_QHENOMENOLOGY_829: (4)                 def apply_function_to_position(self, function:
Callable[[np.ndarray], np.ndarray]) -> Self:
SNGT_QHENOMENOLOGY_830: (8)                     self.move_to(function(self.get_center()))
SNGT_QHENOMENOLOGY_831: (8)                     return self
SNGT_QHENOMENOLOGY_832: (4)                 def apply_function_to_submobject_positions(
SNGT_QHENOMENOLOGY_833: (8)                     self,
SNGT_QHENOMENOLOGY_834: (8)                     function: Callable[[np.ndarray], np.ndarray]
SNGT_QHENOMENOLOGY_835: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_836: (8)                     for submob in self.submobjects:
SNGT_QHENOMENOLOGY_837: (12)                        submob.apply_function_to_position(function)
SNGT_QHENOMENOLOGY_838: (8)                     return self
SNGT_QHENOMENOLOGY_839: (4)                 def apply_matrix(self, matrix: npt.ArrayLike, **kwargs)
-> Self:
SNGT_QHENOMENOLOGY_840: (8)                     if ("about_point" not in kwargs) and ("about_edge"
not in kwargs):
SNGT_QHENOMENOLOGY_841: (12)                        kwargs["about_point"] = ORIGIN
SNGT_QHENOMENOLOGY_842: (8)                     full_matrix = np.identity(self.dim)
SNGT_QHENOMENOLOGY_843: (8)                     matrix = np.array(matrix)
SNGT_QHENOMENOLOGY_844: (8)                     full_matrix[:matrix.shape[0], :matrix.shape[1]] =
matrix
SNGT_QHENOMENOLOGY_845: (8)                     self.apply_points_function(
SNGT_QHENOMENOLOGY_846: (12)                        lambda points: np.dot(points, full_matrix.T),
SNGT_QHENOMENOLOGY_847: (12)                        **kwargs
SNGT_QHENOMENOLOGY_848: (8)                     )
SNGT_QHENOMENOLOGY_849: (8)                     return self
SNGT_QHENOMENOLOGY_850: (4)                 def apply_complex_function(self, function:
Callable[[complex], complex], **kwargs) -> Self:
SNGT_QHENOMENOLOGY_851: (8)                     def R3_func(point):
SNGT_QHENOMENOLOGY_852: (12)                        x, y, z = point
SNGT_QHENOMENOLOGY_853: (12)                        xy_complex = function(complex(x, y))
SNGT_QHENOMENOLOGY_854: (12)                        return [
SNGT_QHENOMENOLOGY_855: (16)                            xy_complex.real,
SNGT_QHENOMENOLOGY_856: (16)                            xy_complex.imag,
SNGT_QHENOMENOLOGY_857: (16)                            z
SNGT_QHENOMENOLOGY_858: (12)                        ]
SNGT_QHENOMENOLOGY_859: (8)                     return self.apply_function(R3_func, **kwargs)
SNGT_QHENOMENOLOGY_860: (4)                 def wag(
```

```
SNGT_QHENOMENOLOGY_861: (8)                     self,
SNGT_QHENOMENOLOGY_862: (8)                     direction: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_863: (8)                     axis: Vect3 = DOWN,
SNGT_QHENOMENOLOGY_864: (8)                     wag_factor: float = 1.0
SNGT_QHENOMENOLOGY_865: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_866: (8)                     for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_867: (12)                        alphas = np.dot(mob.get_points(),
np.transpose(axis))
SNGT_QHENOMENOLOGY_868: (12)                        alphas -= min(alphas)
SNGT_QHENOMENOLOGY_869: (12)                        alphas /= max(alphas)
SNGT_QHENOMENOLOGY_870: (12)                        alphas = alphas**wag_factor
SNGT_QHENOMENOLOGY_871: (12)                        mob.set_points(mob.get_points() + np.dot(
SNGT_QHENOMENOLOGY_872: (16)                            alphas.reshape((len(alphas), 1)),
SNGT_QHENOMENOLOGY_873: (16)                            np.array(direction).reshape((1, mob.dim))
SNGT_QHENOMENOLOGY_874: (12)                        ))
SNGT_QHENOMENOLOGY_875: (8)                     return self
SNGT_QHENOMENOLOGY_876: (4)                 def center(self) -> Self:
SNGT_QHENOMENOLOGY_877: (8)                     self.shift(-self.get_center())
SNGT_QHENOMENOLOGY_878: (8)                     return self
SNGT_QHENOMENOLOGY_879: (4)                 def align_on_border(
SNGT_QHENOMENOLOGY_880: (8)                     self,
SNGT_QHENOMENOLOGY_881: (8)                     direction: Vect3,
SNGT_QHENOMENOLOGY_882: (8)                     buff: float = DEFAULT_MOBJECT_TO_EDGE_BUFF
SNGT_QHENOMENOLOGY_883: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_884: (8)                     """
SNGT_QHENOMENOLOGY_885: (8)                     Direction just needs to be a vector pointing
towards side or
SNGT_QHENOMENOLOGY_886: (8)                     corner in the 2d plane.
SNGT_QHENOMENOLOGY_887: (8)                     """
SNGT_QHENOMENOLOGY_888: (8)                     target_point = np.sign(direction) *
(FRAME_X_RADIUS, FRAME_Y_RADIUS, 0)
SNGT_QHENOMENOLOGY_889: (8)                     point_to_align =
self.get_bounding_box_point(direction)
SNGT_QHENOMENOLOGY_890: (8)                     shift_val = target_point - point_to_align - buff *
np.array(direction)
SNGT_QHENOMENOLOGY_891: (8)                     shift_val = shift_val * abs(np.sign(direction))
SNGT_QHENOMENOLOGY_892: (8)                     self.shift(shift_val)
SNGT_QHENOMENOLOGY_893: (8)                     return self
SNGT_QHENOMENOLOGY_894: (4)                 def to_corner(
SNGT_QHENOMENOLOGY_895: (8)                     self,
SNGT_QHENOMENOLOGY_896: (8)                     corner: Vect3 = LEFT + DOWN,
SNGT_QHENOMENOLOGY_897: (8)                     buff: float = DEFAULT_MOBJECT_TO_EDGE_BUFF
SNGT_QHENOMENOLOGY_898: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_899: (8)                     return self.align_on_border(corner, buff)
SNGT_QHENOMENOLOGY_900: (4)                 def to_edge(
SNGT_QHENOMENOLOGY_901: (8)                     self,
SNGT_QHENOMENOLOGY_902: (8)                     edge: Vect3 = LEFT,
SNGT_QHENOMENOLOGY_903: (8)                     buff: float = DEFAULT_MOBJECT_TO_EDGE_BUFF
SNGT_QHENOMENOLOGY_904: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_905: (8)                     return self.align_on_border(edge, buff)
SNGT_QHENOMENOLOGY_906: (4)                 def next_to(
SNGT_QHENOMENOLOGY_907: (8)                     self,
SNGT_QHENOMENOLOGY_908: (8)                     mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_909: (8)                     direction: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_910: (8)                     buff: float = DEFAULT_MOBJECT_TO_MOBJECT_BUFF,
SNGT_QHENOMENOLOGY_911: (8)                     aligned_edge: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_912: (8)                     submobject_to_align: Mobject | None = None,
SNGT_QHENOMENOLOGY_913: (8)                     index_of_submobject_to_align: int | slice | None =
None,
SNGT_QHENOMENOLOGY_914: (8)                     coor_mask: Vect3 = np.array([1, 1, 1]),
SNGT_QHENOMENOLOGY_915: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_916: (8)                     if isinstance(mobject_or_point, Mobject):
SNGT_QHENOMENOLOGY_917: (12)                        mob = mobject_or_point
SNGT_QHENOMENOLOGY_918: (12)                        if index_of_submobject_to_align is not None:
SNGT_QHENOMENOLOGY_919: (16)                            target_aligner =
mob[index_of_submobject_to_align]
SNGT_QHENOMENOLOGY_920: (12)                        else:
SNGT_QHENOMENOLOGY_921: (16)                            target_aligner = mob
SNGT_QHENOMENOLOGY_922: (12)                        target_point =
```

```
target_aligner.get_bounding_box_point(
SNGT_QHENOMENOLOGY_923: (16)                         aligned_edge + direction
SNGT_QHENOMENOLOGY_924: (12)                     )
SNGT_QHENOMENOLOGY_925: (8)                 else:
SNGT_QHENOMENOLOGY_926: (12)                     target_point = mobject_or_point
SNGT_QHENOMENOLOGY_927: (8)                 if submobject_to_align is not None:
SNGT_QHENOMENOLOGY_928: (12)                     aligner = submobject_to_align
SNGT_QHENOMENOLOGY_929: (8)                 elif index_of_submobject_to_align is not None:
SNGT_QHENOMENOLOGY_930: (12)                     aligner = self[index_of_submobject_to_align]
SNGT_QHENOMENOLOGY_931: (8)                 else:
SNGT_QHENOMENOLOGY_932: (12)                     aligner = self
SNGT_QHENOMENOLOGY_933: (8)                 point_to_align =
aligner.get_bounding_box_point(aligned_edge - direction)
SNGT_QHENOMENOLOGY_934: (8)                 self.shift((target_point - point_to_align + buff *
direction) * coor_mask)
SNGT_QHENOMENOLOGY_935: (8)                 return self
SNGT_QHENOMENOLOGY_936: (4)             def shift_onto_screen(self, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_937: (8)                 space_lengths = [FRAME_X_RADIUS, FRAME_Y_RADIUS]
SNGT_QHENOMENOLOGY_938: (8)                 for vect in UP, DOWN, LEFT, RIGHT:
SNGT_QHENOMENOLOGY_939: (12)                     dim = np.argmax(np.abs(vect))
SNGT_QHENOMENOLOGY_940: (12)                     buff = kwargs.get("buff",
DEFAULT_MOBJECT_TO_EDGE_BUFF)
SNGT_QHENOMENOLOGY_941: (12)                     max_val = space_lengths[dim] - buff
SNGT_QHENOMENOLOGY_942: (12)                     edge_center = self.get_edge_center(vect)
SNGT_QHENOMENOLOGY_943: (12)                     if np.dot(edge_center, vect) > max_val:
SNGT_QHENOMENOLOGY_944: (16)                         self.to_edge(vect, **kwargs)
SNGT_QHENOMENOLOGY_945: (8)                 return self
SNGT_QHENOMENOLOGY_946: (4)             def is_off_screen(self) -> bool:
SNGT_QHENOMENOLOGY_947: (8)                 if self.get_left()[0] > FRAME_X_RADIUS:
SNGT_QHENOMENOLOGY_948: (12)                     return True
SNGT_QHENOMENOLOGY_949: (8)                 if self.get_right()[0] < -FRAME_X_RADIUS:
SNGT_QHENOMENOLOGY_950: (12)                     return True
SNGT_QHENOMENOLOGY_951: (8)                 if self.get_bottom()[1] > FRAME_Y_RADIUS:
SNGT_QHENOMENOLOGY_952: (12)                     return True
SNGT_QHENOMENOLOGY_953: (8)                 if self.get_top()[1] < -FRAME_Y_RADIUS:
SNGT_QHENOMENOLOGY_954: (12)                     return True
SNGT_QHENOMENOLOGY_955: (8)                 return False
SNGT_QHENOMENOLOGY_956: (4)             def stretch_about_point(self, factor: float, dim: int,
point: Vect3) -> Self:
SNGT_QHENOMENOLOGY_957: (8)                 return self.stretch(factor, dim, about_point=point)
SNGT_QHENOMENOLOGY_958: (4)             def stretch_in_place(self, factor: float, dim: int) ->
Self:
SNGT_QHENOMENOLOGY_959: (8)                 return self.stretch(factor, dim)
SNGT_QHENOMENOLOGY_960: (4)             def rescale_to_fit(self, length: float, dim: int,
stretch: bool = False, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_961: (8)                 old_length = self.length_over_dim(dim)
SNGT_QHENOMENOLOGY_962: (8)                 if old_length == 0:
SNGT_QHENOMENOLOGY_963: (12)                     return self
SNGT_QHENOMENOLOGY_964: (8)                 if stretch:
SNGT_QHENOMENOLOGY_965: (12)                     self.stretch(length / old_length, dim,
**kwargs)
SNGT_QHENOMENOLOGY_966: (8)                 else:
SNGT_QHENOMENOLOGY_967: (12)                     self.scale(length / old_length, **kwargs)
SNGT_QHENOMENOLOGY_968: (8)                 return self
SNGT_QHENOMENOLOGY_969: (4)             def stretch_to_fit_width(self, width: float, **kwargs)
-> Self:
SNGT_QHENOMENOLOGY_970: (8)                 return self.rescale_to_fit(width, 0, stretch=True,
**kwargs)
SNGT_QHENOMENOLOGY_971: (4)             def stretch_to_fit_height(self, height: float,
**kwargs) -> Self:
SNGT_QHENOMENOLOGY_972: (8)                 return self.rescale_to_fit(height, 1, stretch=True,
**kwargs)
SNGT_QHENOMENOLOGY_973: (4)             def stretch_to_fit_depth(self, depth: float, **kwargs)
-> Self:
SNGT_QHENOMENOLOGY_974: (8)                 return self.rescale_to_fit(depth, 2, stretch=True,
**kwargs)
SNGT_QHENOMENOLOGY_975: (4)             def set_width(self, width: float, stretch: bool =
False, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_976: (8)                 return self.rescale_to_fit(width, 0,
```

```
stretch=stretch, **kwargs)
SNGT_QHENOMENOLOGY_977: (4)                def set_height(self, height: float, stretch: bool =
False, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_978: (8)                    return self.rescale_to_fit(height, 1,
stretch=stretch, **kwargs)
SNGT_QHENOMENOLOGY_979: (4)                def set_depth(self, depth: float, stretch: bool =
False, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_980: (8)                    return self.rescale_to_fit(depth, 2,
stretch=stretch, **kwargs)
SNGT_QHENOMENOLOGY_981: (4)                def set_max_width(self, max_width: float, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_982: (8)                    if self.get_width() > max_width:
SNGT_QHENOMENOLOGY_983: (12)                       self.set_width(max_width, **kwargs)
SNGT_QHENOMENOLOGY_984: (8)                    return self
SNGT_QHENOMENOLOGY_985: (4)                def set_max_height(self, max_height: float, **kwargs) -
> Self:
SNGT_QHENOMENOLOGY_986: (8)                    if self.get_height() > max_height:
SNGT_QHENOMENOLOGY_987: (12)                       self.set_height(max_height, **kwargs)
SNGT_QHENOMENOLOGY_988: (8)                    return self
SNGT_QHENOMENOLOGY_989: (4)                def set_max_depth(self, max_depth: float, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_990: (8)                    if self.get_depth() > max_depth:
SNGT_QHENOMENOLOGY_991: (12)                       self.set_depth(max_depth, **kwargs)
SNGT_QHENOMENOLOGY_992: (8)                    return self
SNGT_QHENOMENOLOGY_993: (4)                def set_min_width(self, min_width: float, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_994: (8)                    if self.get_width() < min_width:
SNGT_QHENOMENOLOGY_995: (12)                       self.set_width(min_width, **kwargs)
SNGT_QHENOMENOLOGY_996: (8)                    return self
SNGT_QHENOMENOLOGY_997: (4)                def set_min_height(self, min_height: float, **kwargs) -
> Self:
SNGT_QHENOMENOLOGY_998: (8)                    if self.get_height() < min_height:
SNGT_QHENOMENOLOGY_999: (12)                       self.set_height(min_height, **kwargs)
SNGT_QHENOMENOLOGY_1000: (8)                   return self
SNGT_QHENOMENOLOGY_1001: (4)               def set_min_depth(self, min_depth: float, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_1002: (8)                   if self.get_depth() < min_depth:
SNGT_QHENOMENOLOGY_1003: (12)                      self.set_depth(min_depth, **kwargs)
SNGT_QHENOMENOLOGY_1004: (8)                   return self
SNGT_QHENOMENOLOGY_1005: (4)               def set_shape(
SNGT_QHENOMENOLOGY_1006: (8)                   self,
SNGT_QHENOMENOLOGY_1007: (8)                   width: Optional[float] = None,
SNGT_QHENOMENOLOGY_1008: (8)                   height: Optional[float] = None,
SNGT_QHENOMENOLOGY_1009: (8)                   depth: Optional[float] = None,
SNGT_QHENOMENOLOGY_1010: (8)                   **kwargs
SNGT_QHENOMENOLOGY_1011: (4)               ) -> Self:
SNGT_QHENOMENOLOGY_1012: (8)                   if width is not None:
SNGT_QHENOMENOLOGY_1013: (12)                      self.set_width(width, stretch=True, **kwargs)
SNGT_QHENOMENOLOGY_1014: (8)                   if height is not None:
SNGT_QHENOMENOLOGY_1015: (12)                      self.set_height(height, stretch=True, **kwargs)
SNGT_QHENOMENOLOGY_1016: (8)                   if depth is not None:
SNGT_QHENOMENOLOGY_1017: (12)                      self.set_depth(depth, stretch=True, **kwargs)
SNGT_QHENOMENOLOGY_1018: (8)                   return self
SNGT_QHENOMENOLOGY_1019: (4)               def set_coord(self, value: float, dim: int, direction:
Vect3 = ORIGIN) -> Self:
SNGT_QHENOMENOLOGY_1020: (8)                   curr = self.get_coord(dim, direction)
SNGT_QHENOMENOLOGY_1021: (8)                   shift_vect = np.zeros(self.dim)
SNGT_QHENOMENOLOGY_1022: (8)                   shift_vect[dim] = value - curr
SNGT_QHENOMENOLOGY_1023: (8)                   self.shift(shift_vect)
SNGT_QHENOMENOLOGY_1024: (8)                   return self
SNGT_QHENOMENOLOGY_1025: (4)               def set_x(self, x: float, direction: Vect3 = ORIGIN) ->
Self:
SNGT_QHENOMENOLOGY_1026: (8)                   return self.set_coord(x, 0, direction)
SNGT_QHENOMENOLOGY_1027: (4)               def set_y(self, y: float, direction: Vect3 = ORIGIN) ->
Self:
SNGT_QHENOMENOLOGY_1028: (8)                   return self.set_coord(y, 1, direction)
SNGT_QHENOMENOLOGY_1029: (4)               def set_z(self, z: float, direction: Vect3 = ORIGIN) ->
Self:
SNGT_QHENOMENOLOGY_1030: (8)                   return self.set_coord(z, 2, direction)
```

```
SNGT_QHENOMENOLOGY_1031: (4)                def set_z_index(self, z_index: int) -> Self:
SNGT_QHENOMENOLOGY_1032: (8)                    self.z_index = z_index
SNGT_QHENOMENOLOGY_1033: (8)                    return self
SNGT_QHENOMENOLOGY_1034: (4)                def space_out_submobjects(self, factor: float = 1.5,
**kwargs) -> Self:
SNGT_QHENOMENOLOGY_1035: (8)                    self.scale(factor, **kwargs)
SNGT_QHENOMENOLOGY_1036: (8)                    for submob in self.submobjects:
SNGT_QHENOMENOLOGY_1037: (12)                       submob.scale(1. / factor)
SNGT_QHENOMENOLOGY_1038: (8)                    return self
SNGT_QHENOMENOLOGY_1039: (4)                def move_to(
SNGT_QHENOMENOLOGY_1040: (8)                    self,
SNGT_QHENOMENOLOGY_1041: (8)                    point_or_mobject: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1042: (8)                    aligned_edge: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_1043: (8)                    coor_mask: Vect3 = np.array([1, 1, 1])
SNGT_QHENOMENOLOGY_1044: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1045: (8)                    if isinstance(point_or_mobject, Mobject):
SNGT_QHENOMENOLOGY_1046: (12)                       target =
point_or_mobject.get_bounding_box_point(aligned_edge)
SNGT_QHENOMENOLOGY_1047: (8)                    else:
SNGT_QHENOMENOLOGY_1048: (12)                       target = point_or_mobject
SNGT_QHENOMENOLOGY_1049: (8)                    point_to_align =
self.get_bounding_box_point(aligned_edge)
SNGT_QHENOMENOLOGY_1050: (8)                    self.shift((target - point_to_align) * coor_mask)
SNGT_QHENOMENOLOGY_1051: (8)                    return self
SNGT_QHENOMENOLOGY_1052: (4)                def replace(self, mobject: Mobject, dim_to_match: int =
0, stretch: bool = False) -> Self:
SNGT_QHENOMENOLOGY_1053: (8)                    if not mobject.get_num_points() and not
mobject.submobjects:
SNGT_QHENOMENOLOGY_1054: (12)                       self.scale(0)
SNGT_QHENOMENOLOGY_1055: (12)                       return self
SNGT_QHENOMENOLOGY_1056: (8)                    if stretch:
SNGT_QHENOMENOLOGY_1057: (12)                       for i in range(self.dim):
SNGT_QHENOMENOLOGY_1058: (16)
self.rescale_to_fit(mobject.length_over_dim(i), i, stretch=True)
SNGT_QHENOMENOLOGY_1059: (8)                    else:
SNGT_QHENOMENOLOGY_1060: (12)                       self.rescale_to_fit(
SNGT_QHENOMENOLOGY_1061: (16)                           mobject.length_over_dim(dim_to_match),
SNGT_QHENOMENOLOGY_1062: (16)                           dim_to_match,
SNGT_QHENOMENOLOGY_1063: (16)                           stretch=False
SNGT_QHENOMENOLOGY_1064: (12)                       )
SNGT_QHENOMENOLOGY_1065: (8)                    self.shift(mobject.get_center() -
self.get_center())
SNGT_QHENOMENOLOGY_1066: (8)                    return self
SNGT_QHENOMENOLOGY_1067: (4)                def surround(
SNGT_QHENOMENOLOGY_1068: (8)                    self,
SNGT_QHENOMENOLOGY_1069: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_1070: (8)                    dim_to_match: int = 0,
SNGT_QHENOMENOLOGY_1071: (8)                    stretch: bool = False,
SNGT_QHENOMENOLOGY_1072: (8)                    buff: float = MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_1073: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1074: (8)                    self.replace(mobject, dim_to_match, stretch)
SNGT_QHENOMENOLOGY_1075: (8)                    length = mobject.length_over_dim(dim_to_match)
SNGT_QHENOMENOLOGY_1076: (8)                    self.scale((length + buff) / length)
SNGT_QHENOMENOLOGY_1077: (8)                    return self
SNGT_QHENOMENOLOGY_1078: (4)                def put_start_and_end_on(self, start: Vect3, end:
Vect3) -> Self:
SNGT_QHENOMENOLOGY_1079: (8)                    curr_start, curr_end = self.get_start_and_end()
SNGT_QHENOMENOLOGY_1080: (8)                    curr_vect = curr_end - curr_start
SNGT_QHENOMENOLOGY_1081: (8)                    if np.all(curr_vect == 0):
SNGT_QHENOMENOLOGY_1082: (12)                       raise Exception("Cannot position endpoints of
closed loop")
SNGT_QHENOMENOLOGY_1083: (8)                    target_vect = end - start
SNGT_QHENOMENOLOGY_1084: (8)                    self.scale(
SNGT_QHENOMENOLOGY_1085: (12)                       get_norm(target_vect) / get_norm(curr_vect),
SNGT_QHENOMENOLOGY_1086: (12)                       about_point=curr_start,
SNGT_QHENOMENOLOGY_1087: (8)                    )
SNGT_QHENOMENOLOGY_1088: (8)                    self.rotate(
SNGT_QHENOMENOLOGY_1089: (12)                       angle_of_vector(target_vect) -
angle_of_vector(curr_vect),
```

```
SNGT_QHENOMENOLOGY_1090: (8)                    )
SNGT_QHENOMENOLOGY_1091: (8)                    self.rotate(
SNGT_QHENOMENOLOGY_1092: (12)                       np.arctan2(curr_vect[2],
get_norm(curr_vect[:2])) - np.arctan2(target_vect[2], get_norm(target_vect[:2])),
SNGT_QHENOMENOLOGY_1093: (12)                       axis=np.array([-target_vect[1], target_vect[0],
0]),
SNGT_QHENOMENOLOGY_1094: (8)                    )
SNGT_QHENOMENOLOGY_1095: (8)                    self.shift(start - self.get_start())
SNGT_QHENOMENOLOGY_1096: (8)                    return self
SNGT_QHENOMENOLOGY_1097: (4)                @affects_family_data
SNGT_QHENOMENOLOGY_1098: (4)                def set_rgba_array(
SNGT_QHENOMENOLOGY_1099: (8)                    self,
SNGT_QHENOMENOLOGY_1100: (8)                    rgba_array: npt.ArrayLike,
SNGT_QHENOMENOLOGY_1101: (8)                    name: str = "rgba",
SNGT_QHENOMENOLOGY_1102: (8)                    recurse: bool = False
SNGT_QHENOMENOLOGY_1103: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1104: (8)                    for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1105: (12)                       data = mob.data if mob.get_num_points() > 0
else mob._data_defaults
SNGT_QHENOMENOLOGY_1106: (12)                       data[name][:] = rgba_array
SNGT_QHENOMENOLOGY_1107: (8)                    return self
SNGT_QHENOMENOLOGY_1108: (4)                def set_color_by_rgba_func(
SNGT_QHENOMENOLOGY_1109: (8)                    self,
SNGT_QHENOMENOLOGY_1110: (8)                    func: Callable[[Vect3], Vect4],
SNGT_QHENOMENOLOGY_1111: (8)                    recurse: bool = True
SNGT_QHENOMENOLOGY_1112: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1113: (8)                    """
SNGT_QHENOMENOLOGY_1114: (8)                    Func should take in a point in R3 and output an
rgba value
SNGT_QHENOMENOLOGY_1115: (8)                    """
SNGT_QHENOMENOLOGY_1116: (8)                    for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1117: (12)                       rgba_array = [func(point) for point in
mob.get_points()]
SNGT_QHENOMENOLOGY_1118: (12)                       mob.set_rgba_array(rgba_array)
SNGT_QHENOMENOLOGY_1119: (8)                    return self
SNGT_QHENOMENOLOGY_1120: (4)                def set_color_by_rgb_func(
SNGT_QHENOMENOLOGY_1121: (8)                    self,
SNGT_QHENOMENOLOGY_1122: (8)                    func: Callable[[Vect3], Vect3],
SNGT_QHENOMENOLOGY_1123: (8)                    opacity: float = 1,
SNGT_QHENOMENOLOGY_1124: (8)                    recurse: bool = True
SNGT_QHENOMENOLOGY_1125: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1126: (8)                    """
SNGT_QHENOMENOLOGY_1127: (8)                    Func should take in a point in R3 and output an rgb
value
SNGT_QHENOMENOLOGY_1128: (8)                    """
SNGT_QHENOMENOLOGY_1129: (8)                    for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1130: (12)                       rgba_array = [[*func(point), opacity] for point
in mob.get_points()]
SNGT_QHENOMENOLOGY_1131: (12)                       mob.set_rgba_array(rgba_array)
SNGT_QHENOMENOLOGY_1132: (8)                    return self
SNGT_QHENOMENOLOGY_1133: (4)                @affects_family_data
SNGT_QHENOMENOLOGY_1134: (4)                def set_rgba_array_by_color(
SNGT_QHENOMENOLOGY_1135: (8)                    self,
SNGT_QHENOMENOLOGY_1136: (8)                    color: ManimColor | Iterable[ManimColor] | None =
None,
SNGT_QHENOMENOLOGY_1137: (8)                    opacity: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_1138: (8)                    name: str = "rgba",
SNGT_QHENOMENOLOGY_1139: (8)                    recurse: bool = True
SNGT_QHENOMENOLOGY_1140: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1141: (8)                    for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1142: (12)                       data = mob.data if mob.has_points() > 0 else
mob._data_defaults
SNGT_QHENOMENOLOGY_1143: (12)                       if color is not None:
SNGT_QHENOMENOLOGY_1144: (16)                           rgbs = np.array(list(map(color_to_rgb,
listify(color))))
SNGT_QHENOMENOLOGY_1145: (16)                           if 1 < len(rgbs):
SNGT_QHENOMENOLOGY_1146: (20)                               rgbs = resize_with_interpolation(rgbs,
len(data))
SNGT_QHENOMENOLOGY_1147: (16)                           data[name][:, :3] = rgbs
```

```
SNGT_QHENOMENOLOGY_1148: (12)                     if opacity is not None:
SNGT_QHENOMENOLOGY_1149: (16)                         if not isinstance(opacity, (float, int)):
SNGT_QHENOMENOLOGY_1150: (20)                             opacity =
resize_with_interpolation(np.array(opacity), len(data))
SNGT_QHENOMENOLOGY_1151: (16)                         data[name][:, 3] = opacity
SNGT_QHENOMENOLOGY_1152: (8)                 return self
SNGT_QHENOMENOLOGY_1153: (4)             def set_color(
SNGT_QHENOMENOLOGY_1154: (8)                 self,
SNGT_QHENOMENOLOGY_1155: (8)                 color: ManimColor | Iterable[ManimColor] | None,
SNGT_QHENOMENOLOGY_1156: (8)                 opacity: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_1157: (8)                 recurse: bool = True
SNGT_QHENOMENOLOGY_1158: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_1159: (8)                 self.set_rgba_array_by_color(color, opacity,
recurse=False)
SNGT_QHENOMENOLOGY_1160: (8)                 if recurse:
SNGT_QHENOMENOLOGY_1161: (12)                     for submob in self.submobjects:
SNGT_QHENOMENOLOGY_1162: (16)                         submob.set_color(color, recurse=True)
SNGT_QHENOMENOLOGY_1163: (8)                 return self
SNGT_QHENOMENOLOGY_1164: (4)             def set_opacity(
SNGT_QHENOMENOLOGY_1165: (8)                 self,
SNGT_QHENOMENOLOGY_1166: (8)                 opacity: float | Iterable[float] | None,
SNGT_QHENOMENOLOGY_1167: (8)                 recurse: bool = True
SNGT_QHENOMENOLOGY_1168: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_1169: (8)                 self.set_rgba_array_by_color(color=None,
opacity=opacity, recurse=False)
SNGT_QHENOMENOLOGY_1170: (8)                 if recurse:
SNGT_QHENOMENOLOGY_1171: (12)                     for submob in self.submobjects:
SNGT_QHENOMENOLOGY_1172: (16)                         submob.set_opacity(opacity, recurse=True)
SNGT_QHENOMENOLOGY_1173: (8)                 return self
SNGT_QHENOMENOLOGY_1174: (4)             def get_color(self) -> str:
SNGT_QHENOMENOLOGY_1175: (8)                 return rgb_to_hex(self.data["rgba"][0, :3])
SNGT_QHENOMENOLOGY_1176: (4)             def get_opacity(self) -> float:
SNGT_QHENOMENOLOGY_1177: (8)                 return float(self.data["rgba"][0, 3])
SNGT_QHENOMENOLOGY_1178: (4)             def get_opacities(self) -> float:
SNGT_QHENOMENOLOGY_1179: (8)                 return self.data["rgba"][:, 3]
SNGT_QHENOMENOLOGY_1180: (4)             def set_color_by_gradient(self, *colors: ManimColor) ->
Self:
SNGT_QHENOMENOLOGY_1181: (8)                 if self.has_points():
SNGT_QHENOMENOLOGY_1182: (12)                     self.set_color(colors)
SNGT_QHENOMENOLOGY_1183: (8)                 else:
SNGT_QHENOMENOLOGY_1184: (12)                     self.set_submobject_colors_by_gradient(*colors)
SNGT_QHENOMENOLOGY_1185: (8)                 return self
SNGT_QHENOMENOLOGY_1186: (4)             def set_submobject_colors_by_gradient(self, *colors:
ManimColor) -> Self:
SNGT_QHENOMENOLOGY_1187: (8)                 if len(colors) == 0:
SNGT_QHENOMENOLOGY_1188: (12)                     raise Exception("Need at least one color")
SNGT_QHENOMENOLOGY_1189: (8)                 elif len(colors) == 1:
SNGT_QHENOMENOLOGY_1190: (12)                     return self.set_color(*colors)
SNGT_QHENOMENOLOGY_1191: (8)                 mobs = self.submobjects
SNGT_QHENOMENOLOGY_1192: (8)                 new_colors = color_gradient(colors, len(mobs))
SNGT_QHENOMENOLOGY_1193: (8)                 for mob, color in zip(mobs, new_colors):
SNGT_QHENOMENOLOGY_1194: (12)                     mob.set_color(color)
SNGT_QHENOMENOLOGY_1195: (8)                 return self
SNGT_QHENOMENOLOGY_1196: (4)             def fade(self, darkness: float = 0.5, recurse: bool =
True) -> Self:
SNGT_QHENOMENOLOGY_1197: (8)                 self.set_opacity(1.0 - darkness, recurse=recurse)
SNGT_QHENOMENOLOGY_1198: (4)             def get_shading(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_1199: (8)                 return self.uniforms["shading"]
SNGT_QHENOMENOLOGY_1200: (4)             def set_shading(
SNGT_QHENOMENOLOGY_1201: (8)                 self,
SNGT_QHENOMENOLOGY_1202: (8)                 reflectiveness: float | None = None,
SNGT_QHENOMENOLOGY_1203: (8)                 gloss: float | None = None,
SNGT_QHENOMENOLOGY_1204: (8)                 shadow: float | None = None,
SNGT_QHENOMENOLOGY_1205: (8)                 recurse: bool = True
SNGT_QHENOMENOLOGY_1206: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_1207: (8)                 """
SNGT_QHENOMENOLOGY_1208: (8)                 Larger reflectiveness makes things brighter when
facing the light
SNGT_QHENOMENOLOGY_1209: (8)                 Larger shadow makes faces opposite the light darker
```

```
SNGT_QHENOMENOLOGY_1210: (8)                    Makes parts bright where light gets reflected
toward the camera
SNGT_QHENOMENOLOGY_1211: (8)                    """
SNGT_QHENOMENOLOGY_1212: (8)                    for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1213: (12)                       shading = mob.uniforms["shading"]
SNGT_QHENOMENOLOGY_1214: (12)                       for i, value in enumerate([reflectiveness,
gloss, shadow]):
SNGT_QHENOMENOLOGY_1215: (16)                           if value is not None:
SNGT_QHENOMENOLOGY_1216: (20)                               shading[i] = value
SNGT_QHENOMENOLOGY_1217: (12)                       mob.set_uniform(shading=shading, recurse=False)
SNGT_QHENOMENOLOGY_1218: (8)                    return self
SNGT_QHENOMENOLOGY_1219: (4)                def get_reflectiveness(self) -> float:
SNGT_QHENOMENOLOGY_1220: (8)                    return self.get_shading()[0]
SNGT_QHENOMENOLOGY_1221: (4)                def get_gloss(self) -> float:
SNGT_QHENOMENOLOGY_1222: (8)                    return self.get_shading()[1]
SNGT_QHENOMENOLOGY_1223: (4)                def get_shadow(self) -> float:
SNGT_QHENOMENOLOGY_1224: (8)                    return self.get_shading()[2]
SNGT_QHENOMENOLOGY_1225: (4)                def set_reflectiveness(self, reflectiveness: float,
recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_1226: (8)                    self.set_shading(reflectiveness=reflectiveness,
recurse=recurse)
SNGT_QHENOMENOLOGY_1227: (8)                    return self
SNGT_QHENOMENOLOGY_1228: (4)                def set_gloss(self, gloss: float, recurse: bool = True)
-> Self:
SNGT_QHENOMENOLOGY_1229: (8)                    self.set_shading(gloss=gloss, recurse=recurse)
SNGT_QHENOMENOLOGY_1230: (8)                    return self
SNGT_QHENOMENOLOGY_1231: (4)                def set_shadow(self, shadow: float, recurse: bool =
True) -> Self:
SNGT_QHENOMENOLOGY_1232: (8)                    self.set_shading(shadow=shadow, recurse=recurse)
SNGT_QHENOMENOLOGY_1233: (8)                    return self
SNGT_QHENOMENOLOGY_1234: (4)                def add_background_rectangle(
SNGT_QHENOMENOLOGY_1235: (8)                    self,
SNGT_QHENOMENOLOGY_1236: (8)                    color: ManimColor | None = None,
SNGT_QHENOMENOLOGY_1237: (8)                    opacity: float = 1.0,
SNGT_QHENOMENOLOGY_1238: (8)                    **kwargs
SNGT_QHENOMENOLOGY_1239: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1240: (8)                    from manimlib.mobject.shape_matchers import
BackgroundRectangle
SNGT_QHENOMENOLOGY_1241: (8)                    self.background_rectangle = BackgroundRectangle(
SNGT_QHENOMENOLOGY_1242: (12)                       self, color=color,
SNGT_QHENOMENOLOGY_1243: (12)                       fill_opacity=opacity,
SNGT_QHENOMENOLOGY_1244: (12)                       **kwargs
SNGT_QHENOMENOLOGY_1245: (8)                    )
SNGT_QHENOMENOLOGY_1246: (8)                    self.add_to_back(self.background_rectangle)
SNGT_QHENOMENOLOGY_1247: (8)                    return self
SNGT_QHENOMENOLOGY_1248: (4)                def add_background_rectangle_to_submobjects(self,
**kwargs) -> Self:
SNGT_QHENOMENOLOGY_1249: (8)                    for submobject in self.submobjects:
SNGT_QHENOMENOLOGY_1250: (12)                       submobject.add_background_rectangle(**kwargs)
SNGT_QHENOMENOLOGY_1251: (8)                    return self
SNGT_QHENOMENOLOGY_1252: (4)                def
add_background_rectangle_to_family_members_with_points(self, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_1253: (8)                    for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_1254: (12)                       mob.add_background_rectangle(**kwargs)
SNGT_QHENOMENOLOGY_1255: (8)                    return self
SNGT_QHENOMENOLOGY_1256: (4)                def get_bounding_box_point(self, direction: Vect3) ->
Vect3:
SNGT_QHENOMENOLOGY_1257: (8)                    bb = self.get_bounding_box()
SNGT_QHENOMENOLOGY_1258: (8)                    indices = (np.sign(direction) + 1).astype(int)
SNGT_QHENOMENOLOGY_1259: (8)                    return np.array([
SNGT_QHENOMENOLOGY_1260: (12)                       bb[indices[i]][i]
SNGT_QHENOMENOLOGY_1261: (12)                       for i in range(3)
SNGT_QHENOMENOLOGY_1262: (8)                    ])
SNGT_QHENOMENOLOGY_1263: (4)                def get_edge_center(self, direction: Vect3) -> Vect3:
SNGT_QHENOMENOLOGY_1264: (8)                    return self.get_bounding_box_point(direction)
SNGT_QHENOMENOLOGY_1265: (4)                def get_corner(self, direction: Vect3) -> Vect3:
SNGT_QHENOMENOLOGY_1266: (8)                    return self.get_bounding_box_point(direction)
SNGT_QHENOMENOLOGY_1267: (4)                def get_all_corners(self):
SNGT_QHENOMENOLOGY_1268: (8)                    bb = self.get_bounding_box()
```

```
SNGT_QHENOMENOLOGY_1269: (8)                    return np.array([
SNGT_QHENOMENOLOGY_1270: (12)                       [bb[indices[-i + 1]][i] for i in range(3)]
SNGT_QHENOMENOLOGY_1271: (12)                       for indices in it.product([0, 2], repeat=3)
SNGT_QHENOMENOLOGY_1272: (8)                    ])
SNGT_QHENOMENOLOGY_1273: (4)                def get_center(self) -> Vect3:
SNGT_QHENOMENOLOGY_1274: (8)                    return self.get_bounding_box()[1]
SNGT_QHENOMENOLOGY_1275: (4)                def get_center_of_mass(self) -> Vect3:
SNGT_QHENOMENOLOGY_1276: (8)                    return self.get_all_points().mean(0)
SNGT_QHENOMENOLOGY_1277: (4)                def get_boundary_point(self, direction: Vect3) ->
Vect3:
SNGT_QHENOMENOLOGY_1278: (8)                    all_points = self.get_all_points()
SNGT_QHENOMENOLOGY_1279: (8)                    boundary_directions = all_points -
self.get_center()
SNGT_QHENOMENOLOGY_1280: (8)                    norms = np.linalg.norm(boundary_directions, axis=1)
SNGT_QHENOMENOLOGY_1281: (8)                    boundary_directions /= np.repeat(norms,
3).reshape((len(norms), 3))
SNGT_QHENOMENOLOGY_1282: (8)                    index = np.argmax(np.dot(boundary_directions,
np.array(direction).T))
SNGT_QHENOMENOLOGY_1283: (8)                    return all_points[index]
SNGT_QHENOMENOLOGY_1284: (4)                def get_continuous_bounding_box_point(self, direction:
Vect3) -> Vect3:
SNGT_QHENOMENOLOGY_1285: (8)                    dl, center, ur = self.get_bounding_box()
SNGT_QHENOMENOLOGY_1286: (8)                    corner_vect = (ur - center)
SNGT_QHENOMENOLOGY_1287: (8)                    return center + direction /
np.max(np.abs(np.true_divide(
SNGT_QHENOMENOLOGY_1288: (12)                       direction, corner_vect,
SNGT_QHENOMENOLOGY_1289: (12)                       out=np.zeros(len(direction)),
SNGT_QHENOMENOLOGY_1290: (12)                       where=((corner_vect) != 0)
SNGT_QHENOMENOLOGY_1291: (8)                    )))
SNGT_QHENOMENOLOGY_1292: (4)                def get_top(self) -> Vect3:
SNGT_QHENOMENOLOGY_1293: (8)                    return self.get_edge_center(UP)
SNGT_QHENOMENOLOGY_1294: (4)                def get_bottom(self) -> Vect3:
SNGT_QHENOMENOLOGY_1295: (8)                    return self.get_edge_center(DOWN)
SNGT_QHENOMENOLOGY_1296: (4)                def get_right(self) -> Vect3:
SNGT_QHENOMENOLOGY_1297: (8)                    return self.get_edge_center(RIGHT)
SNGT_QHENOMENOLOGY_1298: (4)                def get_left(self) -> Vect3:
SNGT_QHENOMENOLOGY_1299: (8)                    return self.get_edge_center(LEFT)
SNGT_QHENOMENOLOGY_1300: (4)                def get_zenith(self) -> Vect3:
SNGT_QHENOMENOLOGY_1301: (8)                    return self.get_edge_center(OUT)
SNGT_QHENOMENOLOGY_1302: (4)                def get_nadir(self) -> Vect3:
SNGT_QHENOMENOLOGY_1303: (8)                    return self.get_edge_center(IN)
SNGT_QHENOMENOLOGY_1304: (4)                def length_over_dim(self, dim: int) -> float:
SNGT_QHENOMENOLOGY_1305: (8)                    bb = self.get_bounding_box()
SNGT_QHENOMENOLOGY_1306: (8)                    return abs((bb[2] - bb[0])[dim])
SNGT_QHENOMENOLOGY_1307: (4)                def get_width(self) -> float:
SNGT_QHENOMENOLOGY_1308: (8)                    return self.length_over_dim(0)
SNGT_QHENOMENOLOGY_1309: (4)                def get_height(self) -> float:
SNGT_QHENOMENOLOGY_1310: (8)                    return self.length_over_dim(1)
SNGT_QHENOMENOLOGY_1311: (4)                def get_depth(self) -> float:
SNGT_QHENOMENOLOGY_1312: (8)                    return self.length_over_dim(2)
SNGT_QHENOMENOLOGY_1313: (4)                def get_shape(self) -> Tuple[float]:
SNGT_QHENOMENOLOGY_1314: (8)                    return tuple(self.length_over_dim(dim) for dim in
range(3))
SNGT_QHENOMENOLOGY_1315: (4)                def get_coord(self, dim: int, direction: Vect3 =
ORIGIN) -> float:
SNGT_QHENOMENOLOGY_1316: (8)                    """
SNGT_QHENOMENOLOGY_1317: (8)                    Meant to generalize get_x, get_y, get_z
SNGT_QHENOMENOLOGY_1318: (8)                    """
SNGT_QHENOMENOLOGY_1319: (8)                    return self.get_bounding_box_point(direction)[dim]
SNGT_QHENOMENOLOGY_1320: (4)                def get_x(self, direction=ORIGIN) -> float:
SNGT_QHENOMENOLOGY_1321: (8)                    return self.get_coord(0, direction)
SNGT_QHENOMENOLOGY_1322: (4)                def get_y(self, direction=ORIGIN) -> float:
SNGT_QHENOMENOLOGY_1323: (8)                    return self.get_coord(1, direction)
SNGT_QHENOMENOLOGY_1324: (4)                def get_z(self, direction=ORIGIN) -> float:
SNGT_QHENOMENOLOGY_1325: (8)                    return self.get_coord(2, direction)
SNGT_QHENOMENOLOGY_1326: (4)                def get_start(self) -> Vect3:
SNGT_QHENOMENOLOGY_1327: (8)                    self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_1328: (8)                    return self.get_points()[0].copy()
SNGT_QHENOMENOLOGY_1329: (4)                def get_end(self) -> Vect3:
```

```
SNGT_QHENOMENOLOGY_1330: (8)                 self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_1331: (8)                 return self.get_points()[-1].copy()
SNGT_QHENOMENOLOGY_1332: (4)             def get_start_and_end(self) -> tuple[Vect3, Vect3]:
SNGT_QHENOMENOLOGY_1333: (8)                 self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_1334: (8)                 points = self.get_points()
SNGT_QHENOMENOLOGY_1335: (8)                 return (points[0].copy(), points[-1].copy())
SNGT_QHENOMENOLOGY_1336: (4)             def point_from_proportion(self, alpha: float) -> Vect3:
SNGT_QHENOMENOLOGY_1337: (8)                 points = self.get_points()
SNGT_QHENOMENOLOGY_1338: (8)                 i, subalpha = integer_interpolate(0, len(points) -
1, alpha)
SNGT_QHENOMENOLOGY_1339: (8)                 return interpolate(points[i], points[i + 1],
subalpha)
SNGT_QHENOMENOLOGY_1340: (4)             def pfp(self, alpha):
SNGT_QHENOMENOLOGY_1341: (8)                 """Abbreviation for point_from_proportion"""
SNGT_QHENOMENOLOGY_1342: (8)                 return self.point_from_proportion(alpha)
SNGT_QHENOMENOLOGY_1343: (4)             def get_pieces(self, n_pieces: int) -> Group:
SNGT_QHENOMENOLOGY_1344: (8)                 template = self.copy()
SNGT_QHENOMENOLOGY_1345: (8)                 template.set_submobjects([])
SNGT_QHENOMENOLOGY_1346: (8)                 alphas = np.linspace(0, 1, n_pieces + 1)
SNGT_QHENOMENOLOGY_1347: (8)                 return Group(*[
SNGT_QHENOMENOLOGY_1348: (12)                    template.copy().pointwise_become_partial(
SNGT_QHENOMENOLOGY_1349: (16)                        self, a1, a2
SNGT_QHENOMENOLOGY_1350: (12)                    )
SNGT_QHENOMENOLOGY_1351: (12)                    for a1, a2 in zip(alphas[:-1], alphas[1:])
SNGT_QHENOMENOLOGY_1352: (8)                 ])
SNGT_QHENOMENOLOGY_1353: (4)             def get_z_index_reference_point(self) -> Vect3:
SNGT_QHENOMENOLOGY_1354: (8)                 z_index_group = getattr(self, "z_index_group",
self)
SNGT_QHENOMENOLOGY_1355: (8)                 return z_index_group.get_center()
SNGT_QHENOMENOLOGY_1356: (4)             def match_color(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1357: (8)                 return self.set_color(mobject.get_color())
SNGT_QHENOMENOLOGY_1358: (4)             def match_style(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1359: (8)                 self.set_color(mobject.get_color())
SNGT_QHENOMENOLOGY_1360: (8)                 self.set_opacity(mobject.get_opacity())
SNGT_QHENOMENOLOGY_1361: (8)                 self.set_shading(*mobject.get_shading())
SNGT_QHENOMENOLOGY_1362: (8)                 return self
SNGT_QHENOMENOLOGY_1363: (4)             def match_dim_size(self, mobject: Mobject, dim: int,
**kwargs) -> Self:
SNGT_QHENOMENOLOGY_1364: (8)                 return self.rescale_to_fit(
SNGT_QHENOMENOLOGY_1365: (12)                    mobject.length_over_dim(dim), dim,
SNGT_QHENOMENOLOGY_1366: (12)                    **kwargs
SNGT_QHENOMENOLOGY_1367: (8)                 )
SNGT_QHENOMENOLOGY_1368: (4)             def match_width(self, mobject: Mobject, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_1369: (8)                 return self.match_dim_size(mobject, 0, **kwargs)
SNGT_QHENOMENOLOGY_1370: (4)             def match_height(self, mobject: Mobject, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_1371: (8)                 return self.match_dim_size(mobject, 1, **kwargs)
SNGT_QHENOMENOLOGY_1372: (4)             def match_depth(self, mobject: Mobject, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_1373: (8)                 return self.match_dim_size(mobject, 2, **kwargs)
SNGT_QHENOMENOLOGY_1374: (4)             def match_coord(
SNGT_QHENOMENOLOGY_1375: (8)                 self,
SNGT_QHENOMENOLOGY_1376: (8)                 mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1377: (8)                 dim: int,
SNGT_QHENOMENOLOGY_1378: (8)                 direction: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_1379: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_1380: (8)                 if isinstance(mobject_or_point, Mobject):
SNGT_QHENOMENOLOGY_1381: (12)                    coord = mobject_or_point.get_coord(dim,
direction)
SNGT_QHENOMENOLOGY_1382: (8)                 else:
SNGT_QHENOMENOLOGY_1383: (12)                    coord = mobject_or_point[dim]
SNGT_QHENOMENOLOGY_1384: (8)                 return self.set_coord(coord, dim=dim,
direction=direction)
SNGT_QHENOMENOLOGY_1385: (4)             def match_x(
SNGT_QHENOMENOLOGY_1386: (8)                 self,
SNGT_QHENOMENOLOGY_1387: (8)                 mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1388: (8)                 direction: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_1389: (4)             ) -> Self:
```

```
SNGT_QHENOMENOLOGY_1390: (8)                    return self.match_coord(mobject_or_point, 0,
direction)
SNGT_QHENOMENOLOGY_1391: (4)                def match_y(
SNGT_QHENOMENOLOGY_1392: (8)                    self,
SNGT_QHENOMENOLOGY_1393: (8)                    mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1394: (8)                    direction: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_1395: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1396: (8)                    return self.match_coord(mobject_or_point, 1,
direction)
SNGT_QHENOMENOLOGY_1397: (4)                def match_z(
SNGT_QHENOMENOLOGY_1398: (8)                    self,
SNGT_QHENOMENOLOGY_1399: (8)                    mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1400: (8)                    direction: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_1401: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1402: (8)                    return self.match_coord(mobject_or_point, 2,
direction)
SNGT_QHENOMENOLOGY_1403: (4)                def align_to(
SNGT_QHENOMENOLOGY_1404: (8)                    self,
SNGT_QHENOMENOLOGY_1405: (8)                    mobject_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_1406: (8)                    direction: Vect3 = ORIGIN
SNGT_QHENOMENOLOGY_1407: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1408: (8)                    """
SNGT_QHENOMENOLOGY_1409: (8)                    Examples:
SNGT_QHENOMENOLOGY_1410: (8)                    mob1.align_to(mob2, UP) moves mob1 vertically so
that its
SNGT_QHENOMENOLOGY_1411: (8)                    top edge lines ups with mob2's top edge.
SNGT_QHENOMENOLOGY_1412: (8)                    mob1.align_to(mob2, alignment_vect = RIGHT) moves
mob1
SNGT_QHENOMENOLOGY_1413: (8)                    horizontally so that it's center is directly
above/below
SNGT_QHENOMENOLOGY_1414: (8)                    the center of mob2
SNGT_QHENOMENOLOGY_1415: (8)                    """
SNGT_QHENOMENOLOGY_1416: (8)                    if isinstance(mobject_or_point, Mobject):
SNGT_QHENOMENOLOGY_1417: (12)                       point =
mobject_or_point.get_bounding_box_point(direction)
SNGT_QHENOMENOLOGY_1418: (8)                    else:
SNGT_QHENOMENOLOGY_1419: (12)                       point = mobject_or_point
SNGT_QHENOMENOLOGY_1420: (8)                    for dim in range(self.dim):
SNGT_QHENOMENOLOGY_1421: (12)                       if direction[dim] != 0:
SNGT_QHENOMENOLOGY_1422: (16)                           self.set_coord(point[dim], dim, direction)
SNGT_QHENOMENOLOGY_1423: (8)                    return self
SNGT_QHENOMENOLOGY_1424: (4)                def get_group_class(self):
SNGT_QHENOMENOLOGY_1425: (8)                    return Group
SNGT_QHENOMENOLOGY_1426: (4)                def is_aligned_with(self, mobject: Mobject) -> bool:
SNGT_QHENOMENOLOGY_1427: (8)                    if len(self.data) != len(mobject.data):
SNGT_QHENOMENOLOGY_1428: (12)                       return False
SNGT_QHENOMENOLOGY_1429: (8)                    if len(self.submobjects) !=
len(mobject.submobjects):
SNGT_QHENOMENOLOGY_1430: (12)                       return False
SNGT_QHENOMENOLOGY_1431: (8)                    return all(
SNGT_QHENOMENOLOGY_1432: (12)                       sm1.is_aligned_with(sm2)
SNGT_QHENOMENOLOGY_1433: (12)                       for sm1, sm2 in zip(self.submobjects,
mobject.submobjects)
SNGT_QHENOMENOLOGY_1434: (8)                    )
SNGT_QHENOMENOLOGY_1435: (4)                def align_data_and_family(self, mobject: Mobject) ->
Self:
SNGT_QHENOMENOLOGY_1436: (8)                    self.align_family(mobject)
SNGT_QHENOMENOLOGY_1437: (8)                    self.align_data(mobject)
SNGT_QHENOMENOLOGY_1438: (8)                    return self
SNGT_QHENOMENOLOGY_1439: (4)                def align_data(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1440: (8)                    for mob1, mob2 in zip(self.get_family(),
mobject.get_family()):
SNGT_QHENOMENOLOGY_1441: (12)                       mob1.align_points(mob2)
SNGT_QHENOMENOLOGY_1442: (8)                    return self
SNGT_QHENOMENOLOGY_1443: (4)                def align_points(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1444: (8)                    max_len = max(self.get_num_points(),
mobject.get_num_points())
SNGT_QHENOMENOLOGY_1445: (8)                    for mob in (self, mobject):
SNGT_QHENOMENOLOGY_1446: (12)                       mob.resize_points(max_len,
```

```
resize_func=resize_preserving_order)
SNGT_QHENOMENOLOGY_1447: (8)                    return self
SNGT_QHENOMENOLOGY_1448: (4)                def align_family(self, mobject: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1449: (8)                    mob1 = self
SNGT_QHENOMENOLOGY_1450: (8)                    mob2 = mobject
SNGT_QHENOMENOLOGY_1451: (8)                    n1 = len(mob1)
SNGT_QHENOMENOLOGY_1452: (8)                    n2 = len(mob2)
SNGT_QHENOMENOLOGY_1453: (8)                    if n1 != n2:
SNGT_QHENOMENOLOGY_1454: (12)                       mob1.add_n_more_submobjects(max(0, n2 - n1))
SNGT_QHENOMENOLOGY_1455: (12)                       mob2.add_n_more_submobjects(max(0, n1 - n2))
SNGT_QHENOMENOLOGY_1456: (8)                    for sm1, sm2 in zip(mob1.submobjects,
mob2.submobjects):
SNGT_QHENOMENOLOGY_1457: (12)                       sm1.align_family(sm2)
SNGT_QHENOMENOLOGY_1458: (8)                    return self
SNGT_QHENOMENOLOGY_1459: (4)                def push_self_into_submobjects(self) -> Self:
SNGT_QHENOMENOLOGY_1460: (8)                    copy = self.copy()
SNGT_QHENOMENOLOGY_1461: (8)                    copy.set_submobjects([])
SNGT_QHENOMENOLOGY_1462: (8)                    self.resize_points(0)
SNGT_QHENOMENOLOGY_1463: (8)                    self.add(copy)
SNGT_QHENOMENOLOGY_1464: (8)                    return self
SNGT_QHENOMENOLOGY_1465: (4)                def add_n_more_submobjects(self, n: int) -> Self:
SNGT_QHENOMENOLOGY_1466: (8)                    if n == 0:
SNGT_QHENOMENOLOGY_1467: (12)                       return self
SNGT_QHENOMENOLOGY_1468: (8)                    curr = len(self.submobjects)
SNGT_QHENOMENOLOGY_1469: (8)                    if curr == 0:
SNGT_QHENOMENOLOGY_1470: (12)                       null_mob = self.copy()
SNGT_QHENOMENOLOGY_1471: (12)                       null_mob.set_points([self.get_center()])
SNGT_QHENOMENOLOGY_1472: (12)                       self.set_submobjects([
SNGT_QHENOMENOLOGY_1473: (16)                           null_mob.copy()
SNGT_QHENOMENOLOGY_1474: (16)                           for k in range(n)
SNGT_QHENOMENOLOGY_1475: (12)                       ])
SNGT_QHENOMENOLOGY_1476: (12)                       return self
SNGT_QHENOMENOLOGY_1477: (8)                    target = curr + n
SNGT_QHENOMENOLOGY_1478: (8)                    repeat_indices = (np.arange(target) * curr) //
target
SNGT_QHENOMENOLOGY_1479: (8)                    split_factors = [
SNGT_QHENOMENOLOGY_1480: (12)                       (repeat_indices == i).sum()
SNGT_QHENOMENOLOGY_1481: (12)                       for i in range(curr)
SNGT_QHENOMENOLOGY_1482: (8)                    ]
SNGT_QHENOMENOLOGY_1483: (8)                    new_submobs = []
SNGT_QHENOMENOLOGY_1484: (8)                    for submob, sf in zip(self.submobjects,
split_factors):
SNGT_QHENOMENOLOGY_1485: (12)                       new_submobs.append(submob)
SNGT_QHENOMENOLOGY_1486: (12)                       for k in range(1, sf):
SNGT_QHENOMENOLOGY_1487: (16)                           new_submobs.append(submob.invisible_copy())
SNGT_QHENOMENOLOGY_1488: (8)                    self.set_submobjects(new_submobs)
SNGT_QHENOMENOLOGY_1489: (8)                    return self
SNGT_QHENOMENOLOGY_1490: (4)                def invisible_copy(self) -> Self:
SNGT_QHENOMENOLOGY_1491: (8)                    return self.copy().set_opacity(0)
SNGT_QHENOMENOLOGY_1492: (4)                def interpolate(
SNGT_QHENOMENOLOGY_1493: (8)                    self,
SNGT_QHENOMENOLOGY_1494: (8)                    mobject1: Mobject,
SNGT_QHENOMENOLOGY_1495: (8)                    mobject2: Mobject,
SNGT_QHENOMENOLOGY_1496: (8)                    alpha: float,
SNGT_QHENOMENOLOGY_1497: (8)                    path_func: Callable[[np.ndarray, np.ndarray,
float], np.ndarray] = straight_path
SNGT_QHENOMENOLOGY_1498: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1499: (8)                    keys = [k for k in self.data.dtype.names if k not
in self.locked_data_keys]
SNGT_QHENOMENOLOGY_1500: (8)                    if keys:
SNGT_QHENOMENOLOGY_1501: (12)                       self.note_changed_data()
SNGT_QHENOMENOLOGY_1502: (8)                    for key in keys:
SNGT_QHENOMENOLOGY_1503: (12)                       md1 = mobject1.data[key]
SNGT_QHENOMENOLOGY_1504: (12)                       md2 = mobject2.data[key]
SNGT_QHENOMENOLOGY_1505: (12)                       if key in self.const_data_keys:
SNGT_QHENOMENOLOGY_1506: (16)                           md1 = md1[0]
SNGT_QHENOMENOLOGY_1507: (16)                           md2 = md2[0]
SNGT_QHENOMENOLOGY_1508: (12)                       if key in self.pointlike_data_keys:
SNGT_QHENOMENOLOGY_1509: (16)                           self.data[key] = path_func(md1, md2, alpha)
```

```
SNGT_QHENOMENOLOGY_1510: (12)                     else:
SNGT_QHENOMENOLOGY_1511: (16)                         self.data[key] = (1 - alpha) * md1 + alpha
* md2
SNGT_QHENOMENOLOGY_1512: (8)                 for key in self.uniforms:
SNGT_QHENOMENOLOGY_1513: (12)                     if key in self.locked_uniform_keys:
SNGT_QHENOMENOLOGY_1514: (16)                         continue
SNGT_QHENOMENOLOGY_1515: (12)                     if key not in mobject1.uniforms or key not in
mobject2.uniforms:
SNGT_QHENOMENOLOGY_1516: (16)                         continue
SNGT_QHENOMENOLOGY_1517: (12)                     self.uniforms[key] = (1 - alpha) *
mobject1.uniforms[key] + alpha * mobject2.uniforms[key]
SNGT_QHENOMENOLOGY_1518: (8)                 self.bounding_box[:] =
path_func(mobject1.bounding_box, mobject2.bounding_box, alpha)
SNGT_QHENOMENOLOGY_1519: (8)                 return self
SNGT_QHENOMENOLOGY_1520: (4)             def pointwise_become_partial(self, mobject, a, b) ->
Self:
SNGT_QHENOMENOLOGY_1521: (8)                 """
SNGT_QHENOMENOLOGY_1522: (8)                 Set points in such a way as to become only
SNGT_QHENOMENOLOGY_1523: (8)                 part of mobject.
SNGT_QHENOMENOLOGY_1524: (8)                 Inputs 0 <= a < b <= 1 determine what portion
SNGT_QHENOMENOLOGY_1525: (8)                 of mobject to become.
SNGT_QHENOMENOLOGY_1526: (8)                 """
SNGT_QHENOMENOLOGY_1527: (8)                 return self
SNGT_QHENOMENOLOGY_1528: (4)             def lock_data(self, keys: Iterable[str]) -> Self:
SNGT_QHENOMENOLOGY_1529: (8)                 """
SNGT_QHENOMENOLOGY_1530: (8)                 To speed up some animations, particularly
transformations,
SNGT_QHENOMENOLOGY_1531: (8)                 it can be handy to acknowledge which pieces of data
SNGT_QHENOMENOLOGY_1532: (8)                 won't change during the animation so that calls to
SNGT_QHENOMENOLOGY_1533: (8)                 interpolate can skip this, and so that it's not
SNGT_QHENOMENOLOGY_1534: (8)                 read into the shader_wrapper objects needlessly
SNGT_QHENOMENOLOGY_1535: (8)                 """
SNGT_QHENOMENOLOGY_1536: (8)                 if self.has_updaters():
SNGT_QHENOMENOLOGY_1537: (12)                     return self
SNGT_QHENOMENOLOGY_1538: (8)                 self.locked_data_keys = set(keys)
SNGT_QHENOMENOLOGY_1539: (8)                 return self
SNGT_QHENOMENOLOGY_1540: (4)             def lock_uniforms(self, keys: Iterable[str]) -> Self:
SNGT_QHENOMENOLOGY_1541: (8)                 if self.has_updaters():
SNGT_QHENOMENOLOGY_1542: (12)                     return self
SNGT_QHENOMENOLOGY_1543: (8)                 self.locked_uniform_keys = set(keys)
SNGT_QHENOMENOLOGY_1544: (8)                 return self
SNGT_QHENOMENOLOGY_1545: (4)             def lock_matching_data(self, mobject1: Mobject,
mobject2: Mobject) -> Self:
SNGT_QHENOMENOLOGY_1546: (8)                 tuples = zip(
SNGT_QHENOMENOLOGY_1547: (12)                     self.get_family(),
SNGT_QHENOMENOLOGY_1548: (12)                     mobject1.get_family(),
SNGT_QHENOMENOLOGY_1549: (12)                     mobject2.get_family(),
SNGT_QHENOMENOLOGY_1550: (8)                 )
SNGT_QHENOMENOLOGY_1551: (8)                 for sm, sm1, sm2 in tuples:
SNGT_QHENOMENOLOGY_1552: (12)                     if not sm.data.dtype == sm1.data.dtype ==
sm2.data.dtype:
SNGT_QHENOMENOLOGY_1553: (16)                         continue
SNGT_QHENOMENOLOGY_1554: (12)                     sm.lock_data(
SNGT_QHENOMENOLOGY_1555: (16)                         key for key in sm.data.dtype.names
SNGT_QHENOMENOLOGY_1556: (16)                         if arrays_match(sm1.data[key],
sm2.data[key])
SNGT_QHENOMENOLOGY_1557: (12)                     )
SNGT_QHENOMENOLOGY_1558: (12)                     sm.lock_uniforms(
SNGT_QHENOMENOLOGY_1559: (16)                         key for key in self.uniforms
SNGT_QHENOMENOLOGY_1560: (16)                         if all(listify(mobject1.uniforms.get(key,
0) == mobject2.uniforms.get(key, 0)))
SNGT_QHENOMENOLOGY_1561: (12)                     )
SNGT_QHENOMENOLOGY_1562: (12)                     sm.const_data_keys = set(
SNGT_QHENOMENOLOGY_1563: (16)                         key for key in sm.data.dtype.names
SNGT_QHENOMENOLOGY_1564: (16)                         if key not in sm.locked_data_keys
SNGT_QHENOMENOLOGY_1565: (16)                         if all(
SNGT_QHENOMENOLOGY_1566: (20)                             array_is_constant(mob.data[key])
SNGT_QHENOMENOLOGY_1567: (20)                             for mob in (sm, sm1, sm2)
SNGT_QHENOMENOLOGY_1568: (16)                         )
```

```
SNGT_QHENOMENOLOGY_1569: (12)                     )
SNGT_QHENOMENOLOGY_1570: (8)                 return self
SNGT_QHENOMENOLOGY_1571: (4)             def unlock_data(self) -> Self:
SNGT_QHENOMENOLOGY_1572: (8)                 for mob in self.get_family():
SNGT_QHENOMENOLOGY_1573: (12)                    mob.locked_data_keys = set()
SNGT_QHENOMENOLOGY_1574: (12)                    mob.const_data_keys = set()
SNGT_QHENOMENOLOGY_1575: (12)                    mob.locked_uniform_keys = set()
SNGT_QHENOMENOLOGY_1576: (8)                 return self
SNGT_QHENOMENOLOGY_1577: (4)             @staticmethod
SNGT_QHENOMENOLOGY_1578: (4)             def affects_shader_info_id(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_1579: (8)                 @wraps(func)
SNGT_QHENOMENOLOGY_1580: (8)                 def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_1581: (12)                    result = func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_1582: (12)                    self.refresh_shader_wrapper_id()
SNGT_QHENOMENOLOGY_1583: (12)                    return result
SNGT_QHENOMENOLOGY_1584: (8)                 return wrapper
SNGT_QHENOMENOLOGY_1585: (4)             @affects_shader_info_id
SNGT_QHENOMENOLOGY_1586: (4)             def set_uniform(self, recurse: bool = True,
**new_uniforms) -> Self:
SNGT_QHENOMENOLOGY_1587: (8)                 for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1588: (12)                    mob.uniforms.update(new_uniforms)
SNGT_QHENOMENOLOGY_1589: (8)                 return self
SNGT_QHENOMENOLOGY_1590: (4)             @affects_shader_info_id
SNGT_QHENOMENOLOGY_1591: (4)             def fix_in_frame(self, recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_1592: (8)                 self.set_uniform(recurse, is_fixed_in_frame=1.0)
SNGT_QHENOMENOLOGY_1593: (8)                 return self
SNGT_QHENOMENOLOGY_1594: (4)             @affects_shader_info_id
SNGT_QHENOMENOLOGY_1595: (4)             def unfix_from_frame(self, recurse: bool = True) ->
Self:
SNGT_QHENOMENOLOGY_1596: (8)                 self.set_uniform(recurse, is_fixed_in_frame=0.0)
SNGT_QHENOMENOLOGY_1597: (8)                 return self
SNGT_QHENOMENOLOGY_1598: (4)             def is_fixed_in_frame(self) -> bool:
SNGT_QHENOMENOLOGY_1599: (8)                 return bool(self.uniforms["is_fixed_in_frame"])
SNGT_QHENOMENOLOGY_1600: (4)             @affects_shader_info_id
SNGT_QHENOMENOLOGY_1601: (4)             def apply_depth_test(self, recurse: bool = True) ->
Self:
SNGT_QHENOMENOLOGY_1602: (8)                 for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1603: (12)                    mob.depth_test = True
SNGT_QHENOMENOLOGY_1604: (8)                 return self
SNGT_QHENOMENOLOGY_1605: (4)             @affects_shader_info_id
SNGT_QHENOMENOLOGY_1606: (4)             def deactivate_depth_test(self, recurse: bool = True) -
> Self:
SNGT_QHENOMENOLOGY_1607: (8)                 for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1608: (12)                    mob.depth_test = False
SNGT_QHENOMENOLOGY_1609: (8)                 return self
SNGT_QHENOMENOLOGY_1610: (4)             def set_clip_plane(
SNGT_QHENOMENOLOGY_1611: (8)                 self,
SNGT_QHENOMENOLOGY_1612: (8)                 vect: Vect3 | None = None,
SNGT_QHENOMENOLOGY_1613: (8)                 threshold: float | None = None,
SNGT_QHENOMENOLOGY_1614: (8)                 recurse=True
SNGT_QHENOMENOLOGY_1615: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_1616: (8)                 for submob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1617: (12)                    if vect is not None:
SNGT_QHENOMENOLOGY_1618: (16)                        submob.uniforms["clip_plane"][:3] = vect
SNGT_QHENOMENOLOGY_1619: (12)                    if threshold is not None:
SNGT_QHENOMENOLOGY_1620: (16)                        submob.uniforms["clip_plane"][3] =
threshold
SNGT_QHENOMENOLOGY_1621: (8)                 return self
SNGT_QHENOMENOLOGY_1622: (4)             def deactivate_clip_plane(self) -> Self:
SNGT_QHENOMENOLOGY_1623: (8)                 self.uniforms["clip_plane"][:] = 0
SNGT_QHENOMENOLOGY_1624: (8)                 return self
SNGT_QHENOMENOLOGY_1625: (4)             @affects_data
SNGT_QHENOMENOLOGY_1626: (4)             def replace_shader_code(self, old: str, new: str) ->
Self:
SNGT_QHENOMENOLOGY_1627: (8)                 for mob in self.get_family():
SNGT_QHENOMENOLOGY_1628: (12)                    mob.shader_code_replacements[old] = new
SNGT_QHENOMENOLOGY_1629: (12)                    mob.shader_wrapper = None
SNGT_QHENOMENOLOGY_1630: (8)                 return self
```

```
SNGT_QHENOMENOLOGY_1631: (4)                def set_color_by_code(self, glsl_code: str) -> Self:
SNGT_QHENOMENOLOGY_1632: (8)                    """
SNGT_QHENOMENOLOGY_1633: (8)                    Takes a snippet of code and inserts it into a
SNGT_QHENOMENOLOGY_1634: (8)                    context which has the following variables:
SNGT_QHENOMENOLOGY_1635: (8)                    vec4 color, vec3 point, vec3 unit_normal.
SNGT_QHENOMENOLOGY_1636: (8)                    The code should change the color variable
SNGT_QHENOMENOLOGY_1637: (8)                    """
SNGT_QHENOMENOLOGY_1638: (8)                    self.replace_shader_code(
SNGT_QHENOMENOLOGY_1639: (12)                       "///// INSERT COLOR FUNCTION HERE /////",
SNGT_QHENOMENOLOGY_1640: (12)                       glsl_code
SNGT_QHENOMENOLOGY_1641: (8)                    )
SNGT_QHENOMENOLOGY_1642: (8)                    return self
SNGT_QHENOMENOLOGY_1643: (4)                def set_color_by_xyz_func(
SNGT_QHENOMENOLOGY_1644: (8)                    self,
SNGT_QHENOMENOLOGY_1645: (8)                    glsl_snippet: str,
SNGT_QHENOMENOLOGY_1646: (8)                    min_value: float = -5.0,
SNGT_QHENOMENOLOGY_1647: (8)                    max_value: float = 5.0,
SNGT_QHENOMENOLOGY_1648: (8)                    colormap: str = "viridis"
SNGT_QHENOMENOLOGY_1649: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1650: (8)                    """
SNGT_QHENOMENOLOGY_1651: (8)                    Pass in a glsl expression in terms of x, y and z
which returns
SNGT_QHENOMENOLOGY_1652: (8)                    a float.
SNGT_QHENOMENOLOGY_1653: (8)                    """
SNGT_QHENOMENOLOGY_1654: (8)                    for char in "xyz":
SNGT_QHENOMENOLOGY_1655: (12)                       glsl_snippet = glsl_snippet.replace(char,
"point." + char)
SNGT_QHENOMENOLOGY_1656: (8)                    rgb_list = get_colormap_list(colormap)
SNGT_QHENOMENOLOGY_1657: (8)                    self.set_color_by_code(
SNGT_QHENOMENOLOGY_1658: (12)                       "color.rgb = float_to_color({}, {}, {},
{});".format(
SNGT_QHENOMENOLOGY_1659: (16)                           glsl_snippet,
SNGT_QHENOMENOLOGY_1660: (16)                           float(min_value),
SNGT_QHENOMENOLOGY_1661: (16)                           float(max_value),
SNGT_QHENOMENOLOGY_1662: (16)                           get_colormap_code(rgb_list)
SNGT_QHENOMENOLOGY_1663: (12)                       )
SNGT_QHENOMENOLOGY_1664: (8)                    )
SNGT_QHENOMENOLOGY_1665: (8)                    return self
SNGT_QHENOMENOLOGY_1666: (4)                def init_shader_wrapper(self, ctx: Context):
SNGT_QHENOMENOLOGY_1667: (8)                    self.shader_wrapper = ShaderWrapper(
SNGT_QHENOMENOLOGY_1668: (12)                       ctx=ctx,
SNGT_QHENOMENOLOGY_1669: (12)                       vert_data=self.data,
SNGT_QHENOMENOLOGY_1670: (12)                       shader_folder=self.shader_folder,
SNGT_QHENOMENOLOGY_1671: (12)                       mobject_uniforms=self.uniforms,
SNGT_QHENOMENOLOGY_1672: (12)                       texture_paths=self.texture_paths,
SNGT_QHENOMENOLOGY_1673: (12)                       depth_test=self.depth_test,
SNGT_QHENOMENOLOGY_1674: (12)                       render_primitive=self.render_primitive,
SNGT_QHENOMENOLOGY_1675: (12)
code_replacements=self.shader_code_replacements,
SNGT_QHENOMENOLOGY_1676: (8)                    )
SNGT_QHENOMENOLOGY_1677: (4)                def refresh_shader_wrapper_id(self):
SNGT_QHENOMENOLOGY_1678: (8)                    for submob in self.get_family():
SNGT_QHENOMENOLOGY_1679: (12)                       if submob.shader_wrapper is not None:
SNGT_QHENOMENOLOGY_1680: (16)                           submob.shader_wrapper.depth_test =
submob.depth_test
SNGT_QHENOMENOLOGY_1681: (16)                           submob.shader_wrapper.refresh_id()
SNGT_QHENOMENOLOGY_1682: (8)                    for mob in (self, *self.get_ancestors()):
SNGT_QHENOMENOLOGY_1683: (12)                       mob._data_has_changed = True
SNGT_QHENOMENOLOGY_1684: (8)                    return self
SNGT_QHENOMENOLOGY_1685: (4)                def get_shader_wrapper(self, ctx: Context) ->
ShaderWrapper:
SNGT_QHENOMENOLOGY_1686: (8)                    if self.shader_wrapper is None:
SNGT_QHENOMENOLOGY_1687: (12)                       self.init_shader_wrapper(ctx)
SNGT_QHENOMENOLOGY_1688: (8)                    return self.shader_wrapper
SNGT_QHENOMENOLOGY_1689: (4)                def get_shader_wrapper_list(self, ctx: Context) ->
list[ShaderWrapper]:
SNGT_QHENOMENOLOGY_1690: (8)                    family = self.family_members_with_points()
SNGT_QHENOMENOLOGY_1691: (8)                    batches = batch_by_property(family, lambda sm:
sm.get_shader_wrapper(ctx).get_id())
```

```
SNGT_QHENOMENOLOGY_1692: (8)                    result = []
SNGT_QHENOMENOLOGY_1693: (8)                    for submobs, sid in batches:
SNGT_QHENOMENOLOGY_1694: (12)                       shader_wrapper = submobs[0].shader_wrapper
SNGT_QHENOMENOLOGY_1695: (12)                       data_list = [sm.get_shader_data() for sm in
submobs]
SNGT_QHENOMENOLOGY_1696: (12)                       shader_wrapper.read_in(data_list)
SNGT_QHENOMENOLOGY_1697: (12)                       result.append(shader_wrapper)
SNGT_QHENOMENOLOGY_1698: (8)                    return result
SNGT_QHENOMENOLOGY_1699: (4)                def get_shader_data(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_1700: (8)                    indices = self.get_shader_vert_indices()
SNGT_QHENOMENOLOGY_1701: (8)                    if indices is not None:
SNGT_QHENOMENOLOGY_1702: (12)                       return self.data[indices]
SNGT_QHENOMENOLOGY_1703: (8)                    else:
SNGT_QHENOMENOLOGY_1704: (12)                       return self.data
SNGT_QHENOMENOLOGY_1705: (4)                def get_uniforms(self):
SNGT_QHENOMENOLOGY_1706: (8)                    return self.uniforms
SNGT_QHENOMENOLOGY_1707: (4)                def get_shader_vert_indices(self) ->
Optional[np.ndarray]:
SNGT_QHENOMENOLOGY_1708: (8)                    return None
SNGT_QHENOMENOLOGY_1709: (4)                def render(self, ctx: Context, camera_uniforms: dict):
SNGT_QHENOMENOLOGY_1710: (8)                    if self._data_has_changed:
SNGT_QHENOMENOLOGY_1711: (12)                       self.shader_wrappers =
self.get_shader_wrapper_list(ctx)
SNGT_QHENOMENOLOGY_1712: (12)                       self._data_has_changed = False
SNGT_QHENOMENOLOGY_1713: (8)                    for shader_wrapper in self.shader_wrappers:
SNGT_QHENOMENOLOGY_1714: (12)
shader_wrapper.update_program_uniforms(camera_uniforms)
SNGT_QHENOMENOLOGY_1715: (12)                       shader_wrapper.pre_render()
SNGT_QHENOMENOLOGY_1716: (12)                       shader_wrapper.render()
SNGT_QHENOMENOLOGY_1717: (4)                """
SNGT_QHENOMENOLOGY_1718: (8)                    Event handling follows the Event Bubbling model of
DOM in javascript.
SNGT_QHENOMENOLOGY_1719: (8)                    Return false to stop the event bubbling.
SNGT_QHENOMENOLOGY_1720: (8)                    To learn more visit
https://www.quirksmode.org/js/events_order.html
SNGT_QHENOMENOLOGY_1721: (8)                    Event Callback Argument is a callable function
taking two arguments:
SNGT_QHENOMENOLOGY_1722: (12)                       1. Mobject
SNGT_QHENOMENOLOGY_1723: (12)                       2. EventData
SNGT_QHENOMENOLOGY_1724: (4)                """
SNGT_QHENOMENOLOGY_1725: (4)                def init_event_listners(self):
SNGT_QHENOMENOLOGY_1726: (8)                    self.event_listners: list[EventListener] = []
SNGT_QHENOMENOLOGY_1727: (4)                def add_event_listner(
SNGT_QHENOMENOLOGY_1728: (8)                    self,
SNGT_QHENOMENOLOGY_1729: (8)                    event_type: EventType,
SNGT_QHENOMENOLOGY_1730: (8)                    event_callback: Callable[[Mobject, dict[str]]]
SNGT_QHENOMENOLOGY_1731: (4)                ):
SNGT_QHENOMENOLOGY_1732: (8)                    event_listner = EventListener(self, event_type,
event_callback)
SNGT_QHENOMENOLOGY_1733: (8)                    self.event_listners.append(event_listner)
SNGT_QHENOMENOLOGY_1734: (8)                    EVENT_DISPATCHER.add_listner(event_listner)
SNGT_QHENOMENOLOGY_1735: (8)                    return self
SNGT_QHENOMENOLOGY_1736: (4)                def remove_event_listner(
SNGT_QHENOMENOLOGY_1737: (8)                    self,
SNGT_QHENOMENOLOGY_1738: (8)                    event_type: EventType,
SNGT_QHENOMENOLOGY_1739: (8)                    event_callback: Callable[[Mobject, dict[str]]]
SNGT_QHENOMENOLOGY_1740: (4)                ):
SNGT_QHENOMENOLOGY_1741: (8)                    event_listner = EventListener(self, event_type,
event_callback)
SNGT_QHENOMENOLOGY_1742: (8)                    while event_listner in self.event_listners:
SNGT_QHENOMENOLOGY_1743: (12)                       self.event_listners.remove(event_listner)
SNGT_QHENOMENOLOGY_1744: (8)                    EVENT_DISPATCHER.remove_listner(event_listner)
SNGT_QHENOMENOLOGY_1745: (8)                    return self
SNGT_QHENOMENOLOGY_1746: (4)                def clear_event_listners(self, recurse: bool = True):
SNGT_QHENOMENOLOGY_1747: (8)                    self.event_listners = []
SNGT_QHENOMENOLOGY_1748: (8)                    if recurse:
SNGT_QHENOMENOLOGY_1749: (12)                       for submob in self.submobjects:
SNGT_QHENOMENOLOGY_1750: (16)
submob.clear_event_listners(recurse=recurse)
```

```
SNGT_QHENOMENOLOGY_1751: (8)                    return self
SNGT_QHENOMENOLOGY_1752: (4)                def get_event_listners(self):
SNGT_QHENOMENOLOGY_1753: (8)                    return self.event_listners
SNGT_QHENOMENOLOGY_1754: (4)                def get_family_event_listners(self):
SNGT_QHENOMENOLOGY_1755: (8)                    return list(it.chain(*[sm.get_event_listners() for
sm in self.get_family()]))
SNGT_QHENOMENOLOGY_1756: (4)                def get_has_event_listner(self):
SNGT_QHENOMENOLOGY_1757: (8)                    return any(
SNGT_QHENOMENOLOGY_1758: (12)                       mob.get_event_listners()
SNGT_QHENOMENOLOGY_1759: (12)                       for mob in self.get_family()
SNGT_QHENOMENOLOGY_1760: (8)                    )
SNGT_QHENOMENOLOGY_1761: (4)                def add_mouse_motion_listner(self, callback):
SNGT_QHENOMENOLOGY_1762: (8)                    self.add_event_listner(EventType.MouseMotionEvent,
callback)
SNGT_QHENOMENOLOGY_1763: (4)                def remove_mouse_motion_listner(self, callback):
SNGT_QHENOMENOLOGY_1764: (8)
self.remove_event_listner(EventType.MouseMotionEvent, callback)
SNGT_QHENOMENOLOGY_1765: (4)                def add_mouse_press_listner(self, callback):
SNGT_QHENOMENOLOGY_1766: (8)                    self.add_event_listner(EventType.MousePressEvent,
callback)
SNGT_QHENOMENOLOGY_1767: (4)                def remove_mouse_press_listner(self, callback):
SNGT_QHENOMENOLOGY_1768: (8)
self.remove_event_listner(EventType.MousePressEvent, callback)
SNGT_QHENOMENOLOGY_1769: (4)                def add_mouse_release_listner(self, callback):
SNGT_QHENOMENOLOGY_1770: (8)                    self.add_event_listner(EventType.MouseReleaseEvent,
callback)
SNGT_QHENOMENOLOGY_1771: (4)                def remove_mouse_release_listner(self, callback):
SNGT_QHENOMENOLOGY_1772: (8)
self.remove_event_listner(EventType.MouseReleaseEvent, callback)
SNGT_QHENOMENOLOGY_1773: (4)                def add_mouse_drag_listner(self, callback):
SNGT_QHENOMENOLOGY_1774: (8)                    self.add_event_listner(EventType.MouseDragEvent,
callback)
SNGT_QHENOMENOLOGY_1775: (4)                def remove_mouse_drag_listner(self, callback):
SNGT_QHENOMENOLOGY_1776: (8)                    self.remove_event_listner(EventType.MouseDragEvent,
callback)
SNGT_QHENOMENOLOGY_1777: (4)                def add_mouse_scroll_listner(self, callback):
SNGT_QHENOMENOLOGY_1778: (8)                    self.add_event_listner(EventType.MouseScrollEvent,
callback)
SNGT_QHENOMENOLOGY_1779: (4)                def remove_mouse_scroll_listner(self, callback):
SNGT_QHENOMENOLOGY_1780: (8)
self.remove_event_listner(EventType.MouseScrollEvent, callback)
SNGT_QHENOMENOLOGY_1781: (4)                def add_key_press_listner(self, callback):
SNGT_QHENOMENOLOGY_1782: (8)                    self.add_event_listner(EventType.KeyPressEvent,
callback)
SNGT_QHENOMENOLOGY_1783: (4)                def remove_key_press_listner(self, callback):
SNGT_QHENOMENOLOGY_1784: (8)                    self.remove_event_listner(EventType.KeyPressEvent,
callback)
SNGT_QHENOMENOLOGY_1785: (4)                def add_key_release_listner(self, callback):
SNGT_QHENOMENOLOGY_1786: (8)                    self.add_event_listner(EventType.KeyReleaseEvent,
callback)
SNGT_QHENOMENOLOGY_1787: (4)                def remove_key_release_listner(self, callback):
SNGT_QHENOMENOLOGY_1788: (8)
self.remove_event_listner(EventType.KeyReleaseEvent, callback)
SNGT_QHENOMENOLOGY_1789: (4)                def throw_error_if_no_points(self):
SNGT_QHENOMENOLOGY_1790: (8)                    if not self.has_points():
SNGT_QHENOMENOLOGY_1791: (12)                       message = "Cannot call Mobject.{} " +\
SNGT_QHENOMENOLOGY_1792: (22)                                 "for a Mobject with no points"
SNGT_QHENOMENOLOGY_1793: (12)                       caller_name = sys._getframe(1).f_code.co_name
SNGT_QHENOMENOLOGY_1794: (12)                       raise Exception(message.format(caller_name))
SNGT_QHENOMENOLOGY_1795: (0)            class Group(Mobject, Generic[SubmobjectType]):
SNGT_QHENOMENOLOGY_1796: (4)                def __init__(self, *mobjects: SubmobjectType |
Iterable[SubmobjectType], **kwargs):
SNGT_QHENOMENOLOGY_1797: (8)                    super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_1798: (8)                    self._ingest_args(*mobjects)
SNGT_QHENOMENOLOGY_1799: (4)                def _ingest_args(self, *args: Mobject |
Iterable[Mobject]):
SNGT_QHENOMENOLOGY_1800: (8)                    if len(args) == 0:
SNGT_QHENOMENOLOGY_1801: (12)                       return
SNGT_QHENOMENOLOGY_1802: (8)                    if all(isinstance(mob, Mobject) for mob in args):
```

```
SNGT_QHENOMENOLOGY_1803: (12)                     self.add(*args)
SNGT_QHENOMENOLOGY_1804: (8)                 elif isinstance(args[0], Iterable):
SNGT_QHENOMENOLOGY_1805: (12)                     self.add(*args[0])
SNGT_QHENOMENOLOGY_1806: (8)                 else:
SNGT_QHENOMENOLOGY_1807: (12)                     raise Exception(f"Invalid argument to Group of
type {type(args[0])}")
SNGT_QHENOMENOLOGY_1808: (4)             def __add__(self, other: Mobject | Group) -> Self:
SNGT_QHENOMENOLOGY_1809: (8)                 assert isinstance(other, Mobject)
SNGT_QHENOMENOLOGY_1810: (8)                 return self.add(other)
SNGT_QHENOMENOLOGY_1811: (4)             def __getitem__(self, index) -> SubmobjectType:
SNGT_QHENOMENOLOGY_1812: (8)                 return super().__getitem__(index)
SNGT_QHENOMENOLOGY_1813: (0)         class Point(Mobject):
SNGT_QHENOMENOLOGY_1814: (4)             def __init__(
SNGT_QHENOMENOLOGY_1815: (8)                 self,
SNGT_QHENOMENOLOGY_1816: (8)                 location: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_1817: (8)                 artificial_width: float = 1e-6,
SNGT_QHENOMENOLOGY_1818: (8)                 artificial_height: float = 1e-6,
SNGT_QHENOMENOLOGY_1819: (8)                 **kwargs
SNGT_QHENOMENOLOGY_1820: (4)             ):
SNGT_QHENOMENOLOGY_1821: (8)                 self.artificial_width = artificial_width
SNGT_QHENOMENOLOGY_1822: (8)                 self.artificial_height = artificial_height
SNGT_QHENOMENOLOGY_1823: (8)                 super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_1824: (8)                 self.set_location(location)
SNGT_QHENOMENOLOGY_1825: (4)             def get_width(self) -> float:
SNGT_QHENOMENOLOGY_1826: (8)                 return self.artificial_width
SNGT_QHENOMENOLOGY_1827: (4)             def get_height(self) -> float:
SNGT_QHENOMENOLOGY_1828: (8)                 return self.artificial_height
SNGT_QHENOMENOLOGY_1829: (4)             def get_location(self) -> Vect3:
SNGT_QHENOMENOLOGY_1830: (8)                 return self.get_points()[0].copy()
SNGT_QHENOMENOLOGY_1831: (4)             def get_bounding_box_point(self, *args, **kwargs) ->
Vect3:
SNGT_QHENOMENOLOGY_1832: (8)                 return self.get_location()
SNGT_QHENOMENOLOGY_1833: (4)             def set_location(self, new_loc: npt.ArrayLike) -> Self:
SNGT_QHENOMENOLOGY_1834: (8)                 self.set_points(np.array(new_loc, ndmin=2,
dtype=float))
SNGT_QHENOMENOLOGY_1835: (8)                 return self
SNGT_QHENOMENOLOGY_1836: (0)         class _AnimationBuilder:
SNGT_QHENOMENOLOGY_1837: (4)             def __init__(self, mobject: Mobject):
SNGT_QHENOMENOLOGY_1838: (8)                 self.mobject = mobject
SNGT_QHENOMENOLOGY_1839: (8)                 self.overridden_animation = None
SNGT_QHENOMENOLOGY_1840: (8)                 self.mobject.generate_target()
SNGT_QHENOMENOLOGY_1841: (8)                 self.is_chaining = False
SNGT_QHENOMENOLOGY_1842: (8)                 self.methods: list[Callable] = []
SNGT_QHENOMENOLOGY_1843: (8)                 self.anim_args = {}
SNGT_QHENOMENOLOGY_1844: (8)                 self.can_pass_args = True
SNGT_QHENOMENOLOGY_1845: (4)             def __getattr__(self, method_name: str):
SNGT_QHENOMENOLOGY_1846: (8)                 method = getattr(self.mobject.target, method_name)
SNGT_QHENOMENOLOGY_1847: (8)                 self.methods.append(method)
SNGT_QHENOMENOLOGY_1848: (8)                 has_overridden_animation = hasattr(method,
"_override_animate")
SNGT_QHENOMENOLOGY_1849: (8)                 if (self.is_chaining and has_overridden_animation)
or self.overridden_animation:
SNGT_QHENOMENOLOGY_1850: (12)                     raise NotImplementedError(
SNGT_QHENOMENOLOGY_1851: (16)                         "Method chaining is currently not supported
for " + \
SNGT_QHENOMENOLOGY_1852: (16)                         "overridden animations"
SNGT_QHENOMENOLOGY_1853: (12)                     )
SNGT_QHENOMENOLOGY_1854: (8)                 def update_target(*method_args, **method_kwargs):
SNGT_QHENOMENOLOGY_1855: (12)                     if has_overridden_animation:
SNGT_QHENOMENOLOGY_1856: (16)                         self.overridden_animation =
method._override_animate(
SNGT_QHENOMENOLOGY_1857: (20)                             self.mobject, *method_args,
**method_kwargs
SNGT_QHENOMENOLOGY_1858: (16)                         )
SNGT_QHENOMENOLOGY_1859: (12)                     else:
SNGT_QHENOMENOLOGY_1860: (16)                         method(*method_args, **method_kwargs)
SNGT_QHENOMENOLOGY_1861: (12)                     return self
SNGT_QHENOMENOLOGY_1862: (8)                 self.is_chaining = True
SNGT_QHENOMENOLOGY_1863: (8)                 return update_target
```

```
SNGT_QHENOMENOLOGY_1864: (4)               def __call__(self, **kwargs):
SNGT_QHENOMENOLOGY_1865: (8)                   return self.set_anim_args(**kwargs)
SNGT_QHENOMENOLOGY_1866: (4)               def set_anim_args(self, **kwargs):
SNGT_QHENOMENOLOGY_1867: (8)                   '''
SNGT_QHENOMENOLOGY_1868: (8)                   You can change the args of
:class:`~manimlib.animation.transform.Transform`, such as
SNGT_QHENOMENOLOGY_1869: (8)                   - ``run_time``
SNGT_QHENOMENOLOGY_1870: (8)                   - ``time_span``
SNGT_QHENOMENOLOGY_1871: (8)                   - ``rate_func``
SNGT_QHENOMENOLOGY_1872: (8)                   - ``lag_ratio``
SNGT_QHENOMENOLOGY_1873: (8)                   - ``path_arc``
SNGT_QHENOMENOLOGY_1874: (8)                   - ``path_func``
SNGT_QHENOMENOLOGY_1875: (8)                   and so on.
SNGT_QHENOMENOLOGY_1876: (8)                   '''
SNGT_QHENOMENOLOGY_1877: (8)                   if not self.can_pass_args:
SNGT_QHENOMENOLOGY_1878: (12)                      raise ValueError(
SNGT_QHENOMENOLOGY_1879: (16)                          "Animation arguments can only be passed by
calling ``animate`` " + \
SNGT_QHENOMENOLOGY_1880: (16)                          "or ``set_anim_args`` and can only be
passed once",
SNGT_QHENOMENOLOGY_1881: (12)                      )
SNGT_QHENOMENOLOGY_1882: (8)                   self.anim_args = kwargs
SNGT_QHENOMENOLOGY_1883: (8)                   self.can_pass_args = False
SNGT_QHENOMENOLOGY_1884: (8)                   return self
SNGT_QHENOMENOLOGY_1885: (4)               def build(self):
SNGT_QHENOMENOLOGY_1886: (8)                   from manimlib.animation.transform import
_MethodAnimation
SNGT_QHENOMENOLOGY_1887: (8)                   if self.overridden_animation:
SNGT_QHENOMENOLOGY_1888: (12)                      return self.overridden_animation
SNGT_QHENOMENOLOGY_1889: (8)                   return _MethodAnimation(self.mobject, self.methods,
**self.anim_args)
SNGT_QHENOMENOLOGY_1890: (0)           def override_animate(method):
SNGT_QHENOMENOLOGY_1891: (4)               def decorator(animation_method):
SNGT_QHENOMENOLOGY_1892: (8)                   method._override_animate = animation_method
SNGT_QHENOMENOLOGY_1893: (8)                   return animation_method
SNGT_QHENOMENOLOGY_1894: (4)               return decorator
SNGT_QHENOMENOLOGY_1895: (0)           class _UpdaterBuilder:
SNGT_QHENOMENOLOGY_1896: (4)               def __init__(self, mobject: Mobject):
SNGT_QHENOMENOLOGY_1897: (8)                   self.mobject = mobject
SNGT_QHENOMENOLOGY_1898: (4)               def __getattr__(self, method_name: str):
SNGT_QHENOMENOLOGY_1899: (8)                   def add_updater(*method_args, **method_kwargs):
SNGT_QHENOMENOLOGY_1900: (12)                      self.mobject.add_updater(
SNGT_QHENOMENOLOGY_1901: (16)                          lambda m: getattr(m, method_name)
(*method_args, **method_kwargs)
SNGT_QHENOMENOLOGY_1902: (12)                      )
SNGT_QHENOMENOLOGY_1903: (12)                      return self
SNGT_QHENOMENOLOGY_1904: (8)                   return add_updater
SNGT_QHENOMENOLOGY_1905: (0)           class _FunctionalUpdaterBuilder:
SNGT_QHENOMENOLOGY_1906: (4)               def __init__(self, mobject: Mobject):
SNGT_QHENOMENOLOGY_1907: (8)                   self.mobject = mobject
SNGT_QHENOMENOLOGY_1908: (4)               def __getattr__(self, method_name: str):
SNGT_QHENOMENOLOGY_1909: (8)                   def add_updater(*method_args, **method_kwargs):
SNGT_QHENOMENOLOGY_1910: (12)                      self.mobject.add_updater(
SNGT_QHENOMENOLOGY_1911: (16)                          lambda m: getattr(m, method_name)(
SNGT_QHENOMENOLOGY_1912: (20)                              *(arg() for arg in method_args),
SNGT_QHENOMENOLOGY_1913: (20)                              **{
SNGT_QHENOMENOLOGY_1914: (24)                                  key: value()
SNGT_QHENOMENOLOGY_1915: (24)                                  for key, value in
method_kwargs.items()
SNGT_QHENOMENOLOGY_1916: (20)                              }
SNGT_QHENOMENOLOGY_1917: (16)                          )
SNGT_QHENOMENOLOGY_1918: (12)                      )
SNGT_QHENOMENOLOGY_1919: (12)                      return self
SNGT_QHENOMENOLOGY_1920: (8)                   return add_updater
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 25 - numbers.py:
SNGT_QHENOMENOLOGY_
```

```
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from functools import lru_cache
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import DOWN, LEFT, RIGHT, UP
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.svg.text_mobject import Text
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.paths import straight_path
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_11: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_12: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_13: (4)                  from typing import TypeVar, Callable
SNGT_QHENOMENOLOGY_14: (4)                  from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_15: (4)                  from manimlib.typing import ManimColor, Vect3, Self
SNGT_QHENOMENOLOGY_16: (4)                  T = TypeVar("T", bound=VMobject)
SNGT_QHENOMENOLOGY_17: (0)              @lru_cache()
SNGT_QHENOMENOLOGY_18: (0)              def char_to_cahced_mob(char: str, **text_config):
SNGT_QHENOMENOLOGY_19: (4)                  if "\\" in char:
SNGT_QHENOMENOLOGY_20: (8)                      return Tex(char, **text_config)
SNGT_QHENOMENOLOGY_21: (4)                  else:
SNGT_QHENOMENOLOGY_22: (8)                      return Text(char, **text_config)
SNGT_QHENOMENOLOGY_23: (0)              class DecimalNumber(VMobject):
SNGT_QHENOMENOLOGY_24: (4)                  def __init__(
SNGT_QHENOMENOLOGY_25: (8)                      self,
SNGT_QHENOMENOLOGY_26: (8)                      number: float | complex = 0,
SNGT_QHENOMENOLOGY_27: (8)                      color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_28: (8)                      stroke_width: float = 0,
SNGT_QHENOMENOLOGY_29: (8)                      fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_30: (8)                      fill_border_width: float = 0.5,
SNGT_QHENOMENOLOGY_31: (8)                      num_decimal_places: int = 2,
SNGT_QHENOMENOLOGY_32: (8)                      include_sign: bool = False,
SNGT_QHENOMENOLOGY_33: (8)                      group_with_commas: bool = True,
SNGT_QHENOMENOLOGY_34: (8)                      digit_buff_per_font_unit: float = 0.001,
SNGT_QHENOMENOLOGY_35: (8)                      show_ellipsis: bool = False,
SNGT_QHENOMENOLOGY_36: (8)                      unit: str | None = None,  # Aligned to bottom
unless it starts with "^"
SNGT_QHENOMENOLOGY_37: (8)                      include_background_rectangle: bool = False,
SNGT_QHENOMENOLOGY_38: (8)                      edge_to_fix: Vect3 = LEFT,
SNGT_QHENOMENOLOGY_39: (8)                      font_size: float = 48,
SNGT_QHENOMENOLOGY_40: (8)                      text_config: dict = dict(),  # Do not pass in
font_size here
SNGT_QHENOMENOLOGY_41: (8)                      **kwargs
SNGT_QHENOMENOLOGY_42: (4)                  ):
SNGT_QHENOMENOLOGY_43: (8)                      self.num_decimal_places = num_decimal_places
SNGT_QHENOMENOLOGY_44: (8)                      self.include_sign = include_sign
SNGT_QHENOMENOLOGY_45: (8)                      self.group_with_commas = group_with_commas
SNGT_QHENOMENOLOGY_46: (8)                      self.digit_buff_per_font_unit =
digit_buff_per_font_unit
SNGT_QHENOMENOLOGY_47: (8)                      self.show_ellipsis = show_ellipsis
SNGT_QHENOMENOLOGY_48: (8)                      self.unit = unit
SNGT_QHENOMENOLOGY_49: (8)                      self.include_background_rectangle =
include_background_rectangle
SNGT_QHENOMENOLOGY_50: (8)                      self.edge_to_fix = edge_to_fix
SNGT_QHENOMENOLOGY_51: (8)                      self.font_size = font_size
SNGT_QHENOMENOLOGY_52: (8)                      self.text_config = dict(text_config)
SNGT_QHENOMENOLOGY_53: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_54: (12)                         color=color,
SNGT_QHENOMENOLOGY_55: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_56: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_57: (12)                         fill_border_width=fill_border_width,
SNGT_QHENOMENOLOGY_58: (12)                         **kwargs
SNGT_QHENOMENOLOGY_59: (8)                      )
SNGT_QHENOMENOLOGY_60: (8)                      self.set_submobjects_from_number(number)
SNGT_QHENOMENOLOGY_61: (8)                      self.init_colors()
SNGT_QHENOMENOLOGY_62: (4)                  def set_submobjects_from_number(self, number: float |
complex) -> None:
SNGT_QHENOMENOLOGY_63: (8)                      self.number = number
```

```
SNGT_QHENOMENOLOGY_64: (8)                    self.num_string = self.get_num_string(number)
SNGT_QHENOMENOLOGY_65: (8)                    submob_templates = list(map(self.char_to_mob,
self.num_string))
SNGT_QHENOMENOLOGY_66: (8)                    if self.show_ellipsis:
SNGT_QHENOMENOLOGY_67: (12)                       dots = self.char_to_mob("...")
SNGT_QHENOMENOLOGY_68: (12)                       dots.arrange(RIGHT, buff=2 *
dots[0].get_width())
SNGT_QHENOMENOLOGY_69: (12)                       submob_templates.append(dots)
SNGT_QHENOMENOLOGY_70: (8)                    if self.unit is not None:
SNGT_QHENOMENOLOGY_71: (12)
submob_templates.append(self.char_to_mob(self.unit))
SNGT_QHENOMENOLOGY_72: (8)                    font_size = self.get_font_size()
SNGT_QHENOMENOLOGY_73: (8)                    if len(submob_templates) == len(self.submobjects):
SNGT_QHENOMENOLOGY_74: (12)                       for sm, smt in zip(self.submobjects,
submob_templates):
SNGT_QHENOMENOLOGY_75: (16)                           sm.become(smt)
SNGT_QHENOMENOLOGY_76: (16)                           sm.scale(font_size / smt.font_size)
SNGT_QHENOMENOLOGY_77: (8)                    else:
SNGT_QHENOMENOLOGY_78: (12)                       self.set_submobjects([
SNGT_QHENOMENOLOGY_79: (16)                           smt.copy().scale(font_size / smt.font_size)
SNGT_QHENOMENOLOGY_80: (16)                           for smt in submob_templates
SNGT_QHENOMENOLOGY_81: (12)                       ])
SNGT_QHENOMENOLOGY_82: (8)                    digit_buff = self.digit_buff_per_font_unit *
font_size
SNGT_QHENOMENOLOGY_83: (8)                    self.arrange(RIGHT, buff=digit_buff,
aligned_edge=DOWN)
SNGT_QHENOMENOLOGY_84: (8)                    for i, c in enumerate(self.num_string):
SNGT_QHENOMENOLOGY_85: (12)                       if c == "-" and len(self.num_string) > i + 1:
SNGT_QHENOMENOLOGY_86: (16)                           self[i].align_to(self[i + 1], UP)
SNGT_QHENOMENOLOGY_87: (16)                           self[i].shift(self[i + 1].get_height() *
DOWN / 2)
SNGT_QHENOMENOLOGY_88: (12)                       elif c == ",":
SNGT_QHENOMENOLOGY_89: (16)                           self[i].shift(self[i].get_height() * DOWN /
2)
SNGT_QHENOMENOLOGY_90: (8)                    if self.unit and self.unit.startswith("^"):
SNGT_QHENOMENOLOGY_91: (12)                       self[-1].align_to(self, UP)
SNGT_QHENOMENOLOGY_92: (8)                    if self.include_background_rectangle:
SNGT_QHENOMENOLOGY_93: (12)                       self.add_background_rectangle()
SNGT_QHENOMENOLOGY_94: (4)                def get_num_string(self, number: float | complex) ->
str:
SNGT_QHENOMENOLOGY_95: (8)                    if isinstance(number, complex):
SNGT_QHENOMENOLOGY_96: (12)                       formatter = self.get_complex_formatter()
SNGT_QHENOMENOLOGY_97: (8)                    else:
SNGT_QHENOMENOLOGY_98: (12)                       formatter = self.get_formatter()
SNGT_QHENOMENOLOGY_99: (8)                    if self.num_decimal_places == 0 and
isinstance(number, float):
SNGT_QHENOMENOLOGY_100: (12)                      number = int(number)
SNGT_QHENOMENOLOGY_101: (8)                   num_string = formatter.format(number)
SNGT_QHENOMENOLOGY_102: (8)                   rounded_num = np.round(number,
self.num_decimal_places)
SNGT_QHENOMENOLOGY_103: (8)                   if num_string.startswith("-") and rounded_num == 0:
SNGT_QHENOMENOLOGY_104: (12)                      if self.include_sign:
SNGT_QHENOMENOLOGY_105: (16)                          num_string = "+" + num_string[1:]
SNGT_QHENOMENOLOGY_106: (12)                      else:
SNGT_QHENOMENOLOGY_107: (16)                          num_string = num_string[1:]
SNGT_QHENOMENOLOGY_108: (8)                   num_string = num_string.replace("-", "−")
SNGT_QHENOMENOLOGY_109: (8)                   return num_string
SNGT_QHENOMENOLOGY_110: (4)               def char_to_mob(self, char: str) -> Text:
SNGT_QHENOMENOLOGY_111: (8)                   return char_to_cahced_mob(char, **self.text_config)
SNGT_QHENOMENOLOGY_112: (4)               def interpolate(
SNGT_QHENOMENOLOGY_113: (8)                   self,
SNGT_QHENOMENOLOGY_114: (8)                   mobject1: Mobject,
SNGT_QHENOMENOLOGY_115: (8)                   mobject2: Mobject,
SNGT_QHENOMENOLOGY_116: (8)                   alpha: float,
SNGT_QHENOMENOLOGY_117: (8)                   path_func: Callable[[np.ndarray, np.ndarray,
float], np.ndarray] = straight_path
SNGT_QHENOMENOLOGY_118: (4)               ) -> Self:
SNGT_QHENOMENOLOGY_119: (8)                   super().interpolate(mobject1, mobject2, alpha,
path_func)
```

```
SNGT_QHENOMENOLOGY_120: (8)                     if hasattr(mobject1, "font_size") and
hasattr(mobject2, "font_size"):
SNGT_QHENOMENOLOGY_121: (12)                         self.font_size =
interpolate(mobject1.font_size, mobject2.font_size, alpha)
SNGT_QHENOMENOLOGY_122: (4)                 def get_font_size(self) -> float:
SNGT_QHENOMENOLOGY_123: (8)                     return self.font_size
SNGT_QHENOMENOLOGY_124: (4)                 def get_formatter(self, **kwargs) -> str:
SNGT_QHENOMENOLOGY_125: (8)                     """
SNGT_QHENOMENOLOGY_126: (8)                     Configuration is based first off instance
attributes,
SNGT_QHENOMENOLOGY_127: (8)                     but overwritten by any kew word argument.  Relevant
SNGT_QHENOMENOLOGY_128: (8)                     key words:
SNGT_QHENOMENOLOGY_129: (8)                     - include_sign
SNGT_QHENOMENOLOGY_130: (8)                     - group_with_commas
SNGT_QHENOMENOLOGY_131: (8)                     - num_decimal_places
SNGT_QHENOMENOLOGY_132: (8)                     - field_name (e.g. 0 or 0.real)
SNGT_QHENOMENOLOGY_133: (8)                     """
SNGT_QHENOMENOLOGY_134: (8)                     config = dict([
SNGT_QHENOMENOLOGY_135: (12)                         (attr, getattr(self, attr))
SNGT_QHENOMENOLOGY_136: (12)                         for attr in [
SNGT_QHENOMENOLOGY_137: (16)                             "include_sign",
SNGT_QHENOMENOLOGY_138: (16)                             "group_with_commas",
SNGT_QHENOMENOLOGY_139: (16)                             "num_decimal_places",
SNGT_QHENOMENOLOGY_140: (12)                         ]
SNGT_QHENOMENOLOGY_141: (8)                     ])
SNGT_QHENOMENOLOGY_142: (8)                     config.update(kwargs)
SNGT_QHENOMENOLOGY_143: (8)                     ndp = config["num_decimal_places"]
SNGT_QHENOMENOLOGY_144: (8)                     return "".join([
SNGT_QHENOMENOLOGY_145: (12)                         "{",
SNGT_QHENOMENOLOGY_146: (12)                         config.get("field_name", ""),
SNGT_QHENOMENOLOGY_147: (12)                         ":",
SNGT_QHENOMENOLOGY_148: (12)                         "+" if config["include_sign"] else "",
SNGT_QHENOMENOLOGY_149: (12)                         "," if config["group_with_commas"] else "",
SNGT_QHENOMENOLOGY_150: (12)                         f".{ndp}f" if ndp > 0 else "d",
SNGT_QHENOMENOLOGY_151: (12)                         "}",
SNGT_QHENOMENOLOGY_152: (8)                     ])
SNGT_QHENOMENOLOGY_153: (4)                 def get_complex_formatter(self, **kwargs) -> str:
SNGT_QHENOMENOLOGY_154: (8)                     return "".join([
SNGT_QHENOMENOLOGY_155: (12)                         self.get_formatter(field_name="0.real"),
SNGT_QHENOMENOLOGY_156: (12)                         self.get_formatter(field_name="0.imag",
include_sign=True),
SNGT_QHENOMENOLOGY_157: (12)                         "i"
SNGT_QHENOMENOLOGY_158: (8)                     ])
SNGT_QHENOMENOLOGY_159: (4)                 def get_tex(self):
SNGT_QHENOMENOLOGY_160: (8)                     return self.num_string
SNGT_QHENOMENOLOGY_161: (4)                 def set_value(self, number: float | complex) -> Self:
SNGT_QHENOMENOLOGY_162: (8)                     move_to_point =
self.get_edge_center(self.edge_to_fix)
SNGT_QHENOMENOLOGY_163: (8)                     style = self.family_members_with_points()
[0].get_style()
SNGT_QHENOMENOLOGY_164: (8)                     self.set_submobjects_from_number(number)
SNGT_QHENOMENOLOGY_165: (8)                     self.move_to(move_to_point, self.edge_to_fix)
SNGT_QHENOMENOLOGY_166: (8)                     self.set_style(**style)
SNGT_QHENOMENOLOGY_167: (8)                     for submob in self.get_family():
SNGT_QHENOMENOLOGY_168: (12)                         submob.uniforms.update(self.uniforms)
SNGT_QHENOMENOLOGY_169: (8)                     return self
SNGT_QHENOMENOLOGY_170: (4)                 def _handle_scale_side_effects(self, scale_factor:
float) -> Self:
SNGT_QHENOMENOLOGY_171: (8)                     self.font_size *= scale_factor
SNGT_QHENOMENOLOGY_172: (8)                     return self
SNGT_QHENOMENOLOGY_173: (4)                 def get_value(self) -> float | complex:
SNGT_QHENOMENOLOGY_174: (8)                     return self.number
SNGT_QHENOMENOLOGY_175: (4)                 def increment_value(self, delta_t: float | complex = 1)
-> Self:
SNGT_QHENOMENOLOGY_176: (8)                     self.set_value(self.get_value() + delta_t)
SNGT_QHENOMENOLOGY_177: (8)                     return self
SNGT_QHENOMENOLOGY_178: (0)             class Integer(DecimalNumber):
SNGT_QHENOMENOLOGY_179: (4)                 def __init__(
SNGT_QHENOMENOLOGY_180: (8)                     self,
```

```
SNGT_QHENOMENOLOGY_181: (8)                     number: int = 0,
SNGT_QHENOMENOLOGY_182: (8)                     num_decimal_places: int = 0,
SNGT_QHENOMENOLOGY_183: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_184: (4)                 ):
SNGT_QHENOMENOLOGY_185: (8)                     super().__init__(number,
num_decimal_places=num_decimal_places, **kwargs)
SNGT_QHENOMENOLOGY_186: (4)                 def get_value(self) -> int:
SNGT_QHENOMENOLOGY_187: (8)                     return int(np.round(super().get_value()))
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 26 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 27 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from manimlib.event_handler.event_dispatcher import
EventDispatcher
SNGT_QHENOMENOLOGY_2: (0)             EVENT_DISPATCHER = EventDispatcher()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 28 - changing.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import numpy as np
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.constants import BLUE_B, BLUE_D, BLUE_E,
GREY_BROWN, WHITE
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.utils.rate_functions import smooth
SNGT_QHENOMENOLOGY_8: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_9: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_10: (4)                 from typing import Callable, List, Iterable
SNGT_QHENOMENOLOGY_11: (4)                 from manimlib.typing import ManimColor, Vect3, Self
SNGT_QHENOMENOLOGY_12: (0)             class AnimatedBoundary(VGroup):
SNGT_QHENOMENOLOGY_13: (4)                 def __init__(
SNGT_QHENOMENOLOGY_14: (8)                     self,
SNGT_QHENOMENOLOGY_15: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_16: (8)                     colors: List[ManimColor] = [BLUE_D, BLUE_B, BLUE_E,
GREY_BROWN],
SNGT_QHENOMENOLOGY_17: (8)                     max_stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_18: (8)                     cycle_rate: float = 0.5,
SNGT_QHENOMENOLOGY_19: (8)                     back_and_forth: bool = True,
SNGT_QHENOMENOLOGY_20: (8)                     draw_rate_func: Callable[[float], float] = smooth,
SNGT_QHENOMENOLOGY_21: (8)                     fade_rate_func: Callable[[float], float] = smooth,
SNGT_QHENOMENOLOGY_22: (8)                     **kwargs
SNGT_QHENOMENOLOGY_23: (4)                 ):
SNGT_QHENOMENOLOGY_24: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_25: (8)                     self.vmobject: VMobject = vmobject
SNGT_QHENOMENOLOGY_26: (8)                     self.colors = colors
SNGT_QHENOMENOLOGY_27: (8)                     self.max_stroke_width = max_stroke_width
SNGT_QHENOMENOLOGY_28: (8)                     self.cycle_rate = cycle_rate
SNGT_QHENOMENOLOGY_29: (8)                     self.back_and_forth = back_and_forth
SNGT_QHENOMENOLOGY_30: (8)                     self.draw_rate_func = draw_rate_func
SNGT_QHENOMENOLOGY_31: (8)                     self.fade_rate_func = fade_rate_func
SNGT_QHENOMENOLOGY_32: (8)                     self.boundary_copies: list[VMobject] = [
SNGT_QHENOMENOLOGY_33: (12)                         vmobject.copy().set_style(
SNGT_QHENOMENOLOGY_34: (16)                             stroke_width=0,
SNGT_QHENOMENOLOGY_35: (16)                             fill_opacity=0
SNGT_QHENOMENOLOGY_36: (12)                         )
SNGT_QHENOMENOLOGY_37: (12)                         for x in range(2)
SNGT_QHENOMENOLOGY_38: (8)                     ]
```

```
SNGT_QHENOMENOLOGY_39: (8)                      self.add(*self.boundary_copies)
SNGT_QHENOMENOLOGY_40: (8)                      self.total_time: float = 0
SNGT_QHENOMENOLOGY_41: (8)                      self.add_updater(
SNGT_QHENOMENOLOGY_42: (12)                         lambda m, dt: self.update_boundary_copies(dt)
SNGT_QHENOMENOLOGY_43: (8)                      )
SNGT_QHENOMENOLOGY_44: (4)                  def update_boundary_copies(self, dt: float) -> Self:
SNGT_QHENOMENOLOGY_45: (8)                      time = self.total_time * self.cycle_rate
SNGT_QHENOMENOLOGY_46: (8)                      growing, fading = self.boundary_copies
SNGT_QHENOMENOLOGY_47: (8)                      colors = self.colors
SNGT_QHENOMENOLOGY_48: (8)                      msw = self.max_stroke_width
SNGT_QHENOMENOLOGY_49: (8)                      vmobject = self.vmobject
SNGT_QHENOMENOLOGY_50: (8)                      index = int(time % len(colors))
SNGT_QHENOMENOLOGY_51: (8)                      alpha = time % 1
SNGT_QHENOMENOLOGY_52: (8)                      draw_alpha = self.draw_rate_func(alpha)
SNGT_QHENOMENOLOGY_53: (8)                      fade_alpha = self.fade_rate_func(alpha)
SNGT_QHENOMENOLOGY_54: (8)                      if self.back_and_forth and int(time) % 2 == 1:
SNGT_QHENOMENOLOGY_55: (12)                         bounds = (1 - draw_alpha, 1)
SNGT_QHENOMENOLOGY_56: (8)                      else:
SNGT_QHENOMENOLOGY_57: (12)                         bounds = (0, draw_alpha)
SNGT_QHENOMENOLOGY_58: (8)                      self.full_family_become_partial(growing, vmobject,
*bounds)
SNGT_QHENOMENOLOGY_59: (8)                      growing.set_stroke(colors[index], width=msw)
SNGT_QHENOMENOLOGY_60: (8)                      if time >= 1:
SNGT_QHENOMENOLOGY_61: (12)                         self.full_family_become_partial(fading,
vmobject, 0, 1)
SNGT_QHENOMENOLOGY_62: (12)                         fading.set_stroke(
SNGT_QHENOMENOLOGY_63: (16)                             color=colors[index - 1],
SNGT_QHENOMENOLOGY_64: (16)                             width=(1 - fade_alpha) * msw
SNGT_QHENOMENOLOGY_65: (12)                         )
SNGT_QHENOMENOLOGY_66: (8)                      self.total_time += dt
SNGT_QHENOMENOLOGY_67: (8)                      return self
SNGT_QHENOMENOLOGY_68: (4)                  def full_family_become_partial(
SNGT_QHENOMENOLOGY_69: (8)                      self,
SNGT_QHENOMENOLOGY_70: (8)                      mob1: VMobject,
SNGT_QHENOMENOLOGY_71: (8)                      mob2: VMobject,
SNGT_QHENOMENOLOGY_72: (8)                      a: float,
SNGT_QHENOMENOLOGY_73: (8)                      b: float
SNGT_QHENOMENOLOGY_74: (4)                  ) -> Self:
SNGT_QHENOMENOLOGY_75: (8)                      family1 = mob1.family_members_with_points()
SNGT_QHENOMENOLOGY_76: (8)                      family2 = mob2.family_members_with_points()
SNGT_QHENOMENOLOGY_77: (8)                      for sm1, sm2 in zip(family1, family2):
SNGT_QHENOMENOLOGY_78: (12)                         sm1.pointwise_become_partial(sm2, a, b)
SNGT_QHENOMENOLOGY_79: (8)                      return self
SNGT_QHENOMENOLOGY_80: (0)              class TracedPath(VMobject):
SNGT_QHENOMENOLOGY_81: (4)                  def __init__(
SNGT_QHENOMENOLOGY_82: (8)                      self,
SNGT_QHENOMENOLOGY_83: (8)                      traced_point_func: Callable[[], Vect3],
SNGT_QHENOMENOLOGY_84: (8)                      time_traced: float = np.inf,
SNGT_QHENOMENOLOGY_85: (8)                      time_per_anchor: float = 1.0 / 15,
SNGT_QHENOMENOLOGY_86: (8)                      stroke_width: float | Iterable[float] = 2.0,
SNGT_QHENOMENOLOGY_87: (8)                      stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_88: (8)                      **kwargs
SNGT_QHENOMENOLOGY_89: (4)                  ):
SNGT_QHENOMENOLOGY_90: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_91: (8)                      self.traced_point_func = traced_point_func
SNGT_QHENOMENOLOGY_92: (8)                      self.time_traced = time_traced
SNGT_QHENOMENOLOGY_93: (8)                      self.time_per_anchor = time_per_anchor
SNGT_QHENOMENOLOGY_94: (8)                      self.time: float = 0
SNGT_QHENOMENOLOGY_95: (8)                      self.traced_points: list[np.ndarray] = []
SNGT_QHENOMENOLOGY_96: (8)                      self.add_updater(lambda m, dt: m.update_path(dt))
SNGT_QHENOMENOLOGY_97: (8)                      self.set_stroke(stroke_color, stroke_width)
SNGT_QHENOMENOLOGY_98: (4)                  def update_path(self, dt: float) -> Self:
SNGT_QHENOMENOLOGY_99: (8)                      if dt == 0:
SNGT_QHENOMENOLOGY_100: (12)                        return self
SNGT_QHENOMENOLOGY_101: (8)                     point = self.traced_point_func().copy()
SNGT_QHENOMENOLOGY_102: (8)                     self.traced_points.append(point)
SNGT_QHENOMENOLOGY_103: (8)                     if self.time_traced < np.inf:
SNGT_QHENOMENOLOGY_104: (12)                        n_relevant_points = int(self.time_traced / dt +
0.5)
```

```
SNGT_QHENOMENOLOGY_105: (12)                         n_tps = len(self.traced_points)
SNGT_QHENOMENOLOGY_106: (12)                         if n_tps < n_relevant_points:
SNGT_QHENOMENOLOGY_107: (16)                             points = self.traced_points + [point] *
(n_relevant_points - n_tps)
SNGT_QHENOMENOLOGY_108: (12)                         else:
SNGT_QHENOMENOLOGY_109: (16)                             points = self.traced_points[n_tps -
n_relevant_points:]
SNGT_QHENOMENOLOGY_110: (12)                         if n_tps > 10 * n_relevant_points:
SNGT_QHENOMENOLOGY_111: (16)                             self.traced_points = self.traced_points[-
n_relevant_points:]
SNGT_QHENOMENOLOGY_112: (8)                     else:
SNGT_QHENOMENOLOGY_113: (12)                         points = self.traced_points
SNGT_QHENOMENOLOGY_114: (8)                     if points:
SNGT_QHENOMENOLOGY_115: (12)                         self.set_points_smoothly(points)
SNGT_QHENOMENOLOGY_116: (8)                     self.time += dt
SNGT_QHENOMENOLOGY_117: (8)                     return self
SNGT_QHENOMENOLOGY_118: (0)             class TracingTail(TracedPath):
SNGT_QHENOMENOLOGY_119: (4)                 def __init__(
SNGT_QHENOMENOLOGY_120: (8)                     self,
SNGT_QHENOMENOLOGY_121: (8)                     mobject_or_func: Mobject | Callable[[],
np.ndarray],
SNGT_QHENOMENOLOGY_122: (8)                     time_traced: float = 1.0,
SNGT_QHENOMENOLOGY_123: (8)                     stroke_width: float | Iterable[float] = (0, 3),
SNGT_QHENOMENOLOGY_124: (8)                     stroke_opacity: float | Iterable[float] = (0, 1),
SNGT_QHENOMENOLOGY_125: (8)                     stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_126: (8)                     **kwargs
SNGT_QHENOMENOLOGY_127: (4)                 ):
SNGT_QHENOMENOLOGY_128: (8)                     if isinstance(mobject_or_func, Mobject):
SNGT_QHENOMENOLOGY_129: (12)                         func = mobject_or_func.get_center
SNGT_QHENOMENOLOGY_130: (8)                     else:
SNGT_QHENOMENOLOGY_131: (12)                         func = mobject_or_func
SNGT_QHENOMENOLOGY_132: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_133: (12)                         func,
SNGT_QHENOMENOLOGY_134: (12)                         time_traced=time_traced,
SNGT_QHENOMENOLOGY_135: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_136: (12)                         stroke_opacity=stroke_opacity,
SNGT_QHENOMENOLOGY_137: (12)                         stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_138: (12)                         **kwargs
SNGT_QHENOMENOLOGY_139: (8)                     )
SNGT_QHENOMENOLOGY_140: (8)                     self.add_updater(lambda m:
m.set_stroke(width=stroke_width, opacity=stroke_opacity))
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 29 - geometry.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import math
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import DL, DOWN, DR, LEFT, ORIGIN,
OUT, RIGHT, UL, UP, UR
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import GREY_A, RED, WHITE, BLACK
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import MED_SMALL_BUFF, SMALL_BUFF
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import DEG, PI, TAU
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.types.vectorized_mobject import
DashedVMobject
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.bezier import
quadratic_bezier_points_for_arc
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.iterables import adjacent_n_tuples
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.iterables import adjacent_pairs
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.simple_functions import fdiv
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.space_ops import angle_between_vectors
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.utils.space_ops import angle_of_vector
```

```
SNGT_QHENOMENOLOGY_19: (0)              from manimlib.utils.space_ops import cross2d
SNGT_QHENOMENOLOGY_20: (0)              from manimlib.utils.space_ops import compass_directions
SNGT_QHENOMENOLOGY_21: (0)              from manimlib.utils.space_ops import find_intersection
SNGT_QHENOMENOLOGY_22: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_23: (0)              from manimlib.utils.space_ops import normalize
SNGT_QHENOMENOLOGY_24: (0)              from manimlib.utils.space_ops import rotate_vector
SNGT_QHENOMENOLOGY_25: (0)              from manimlib.utils.space_ops import
rotation_matrix_transpose
SNGT_QHENOMENOLOGY_26: (0)              from manimlib.utils.space_ops import
rotation_between_vectors
SNGT_QHENOMENOLOGY_27: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_28: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_29: (4)                  from typing import Iterable, Optional
SNGT_QHENOMENOLOGY_30: (4)                  from manimlib.typing import ManimColor, Vect3,
Vect3Array, Self
SNGT_QHENOMENOLOGY_31: (0)              DEFAULT_DOT_RADIUS = 0.08
SNGT_QHENOMENOLOGY_32: (0)              DEFAULT_SMALL_DOT_RADIUS = 0.04
SNGT_QHENOMENOLOGY_33: (0)              DEFAULT_DASH_LENGTH = 0.05
SNGT_QHENOMENOLOGY_34: (0)              DEFAULT_ARROW_TIP_LENGTH = 0.35
SNGT_QHENOMENOLOGY_35: (0)              DEFAULT_ARROW_TIP_WIDTH = 0.35
SNGT_QHENOMENOLOGY_36: (0)              class TipableVMobject(VMobject):
SNGT_QHENOMENOLOGY_37: (4)                  """
SNGT_QHENOMENOLOGY_38: (4)                  Meant for shared functionality between Arc and Line.
SNGT_QHENOMENOLOGY_39: (4)                  Functionality can be classified broadly into these
groups:
SNGT_QHENOMENOLOGY_40: (8)                      * Adding, Creating, Modifying tips
SNGT_QHENOMENOLOGY_41: (12)                         - add_tip calls create_tip, before pushing the
new tip
SNGT_QHENOMENOLOGY_42: (16)                             into the TipableVMobject's list of
submobjects
SNGT_QHENOMENOLOGY_43: (12)                         - stylistic and positional configuration
SNGT_QHENOMENOLOGY_44: (8)                      * Checking for tips
SNGT_QHENOMENOLOGY_45: (12)                         - Boolean checks for whether the
TipableVMobject has a tip
SNGT_QHENOMENOLOGY_46: (16)                             and a starting tip
SNGT_QHENOMENOLOGY_47: (8)                      * Getters
SNGT_QHENOMENOLOGY_48: (12)                         - Straightforward accessors, returning
information pertaining
SNGT_QHENOMENOLOGY_49: (16)                             to the TipableVMobject instance's tip(s),
its length etc
SNGT_QHENOMENOLOGY_50: (4)                  """
SNGT_QHENOMENOLOGY_51: (4)                  tip_config: dict = dict(
SNGT_QHENOMENOLOGY_52: (8)                      fill_opacity=1.0,
SNGT_QHENOMENOLOGY_53: (8)                      stroke_width=0.0,
SNGT_QHENOMENOLOGY_54: (8)                      tip_style=0.0,  # triangle=0, inner_smooth=1, dot=2
SNGT_QHENOMENOLOGY_55: (4)                  )
SNGT_QHENOMENOLOGY_56: (4)                  def add_tip(self, at_start: bool = False, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_57: (8)                      """
SNGT_QHENOMENOLOGY_58: (8)                      Adds a tip to the TipableVMobject instance,
recognising
SNGT_QHENOMENOLOGY_59: (8)                      that the endpoints might need to be switched if
it's
SNGT_QHENOMENOLOGY_60: (8)                      a 'starting tip' or not.
SNGT_QHENOMENOLOGY_61: (8)                      """
SNGT_QHENOMENOLOGY_62: (8)                      tip = self.create_tip(at_start, **kwargs)
SNGT_QHENOMENOLOGY_63: (8)                      self.reset_endpoints_based_on_tip(tip, at_start)
SNGT_QHENOMENOLOGY_64: (8)                      self.asign_tip_attr(tip, at_start)
SNGT_QHENOMENOLOGY_65: (8)                      tip.set_color(self.get_stroke_color())
SNGT_QHENOMENOLOGY_66: (8)                      self.add(tip)
SNGT_QHENOMENOLOGY_67: (8)                      return self
SNGT_QHENOMENOLOGY_68: (4)                  def create_tip(self, at_start: bool = False, **kwargs)
-> ArrowTip:
SNGT_QHENOMENOLOGY_69: (8)                      """
SNGT_QHENOMENOLOGY_70: (8)                      Stylises the tip, positions it spacially, and
returns
SNGT_QHENOMENOLOGY_71: (8)                      the newly instantiated tip to the caller.
SNGT_QHENOMENOLOGY_72: (8)                      """
SNGT_QHENOMENOLOGY_73: (8)                      tip = self.get_unpositioned_tip(**kwargs)
```

```
SNGT_QHENOMENOLOGY_74: (8)                      self.position_tip(tip, at_start)
SNGT_QHENOMENOLOGY_75: (8)                      return tip
SNGT_QHENOMENOLOGY_76: (4)                  def get_unpositioned_tip(self, **kwargs) -> ArrowTip:
SNGT_QHENOMENOLOGY_77: (8)                      """
SNGT_QHENOMENOLOGY_78: (8)                      Returns a tip that has been stylistically
configured,
SNGT_QHENOMENOLOGY_79: (8)                      but has not yet been given a position in space.
SNGT_QHENOMENOLOGY_80: (8)                      """
SNGT_QHENOMENOLOGY_81: (8)                      config = dict()
SNGT_QHENOMENOLOGY_82: (8)                      config.update(self.tip_config)
SNGT_QHENOMENOLOGY_83: (8)                      config.update(kwargs)
SNGT_QHENOMENOLOGY_84: (8)                      return ArrowTip(**config)
SNGT_QHENOMENOLOGY_85: (4)                  def position_tip(self, tip: ArrowTip, at_start: bool =
False) -> ArrowTip:
SNGT_QHENOMENOLOGY_86: (8)                      if at_start:
SNGT_QHENOMENOLOGY_87: (12)                         anchor = self.get_start()
SNGT_QHENOMENOLOGY_88: (12)                         handle = self.get_first_handle()
SNGT_QHENOMENOLOGY_89: (8)                      else:
SNGT_QHENOMENOLOGY_90: (12)                         handle = self.get_last_handle()
SNGT_QHENOMENOLOGY_91: (12)                         anchor = self.get_end()
SNGT_QHENOMENOLOGY_92: (8)                      tip.rotate(angle_of_vector(handle - anchor) - PI -
tip.get_angle())
SNGT_QHENOMENOLOGY_93: (8)                      tip.shift(anchor - tip.get_tip_point())
SNGT_QHENOMENOLOGY_94: (8)                      return tip
SNGT_QHENOMENOLOGY_95: (4)                  def reset_endpoints_based_on_tip(self, tip: ArrowTip,
at_start: bool) -> Self:
SNGT_QHENOMENOLOGY_96: (8)                      if self.get_length() == 0:
SNGT_QHENOMENOLOGY_97: (12)                         return self
SNGT_QHENOMENOLOGY_98: (8)                      if at_start:
SNGT_QHENOMENOLOGY_99: (12)                         start = tip.get_base()
SNGT_QHENOMENOLOGY_100: (12)                        end = self.get_end()
SNGT_QHENOMENOLOGY_101: (8)                     else:
SNGT_QHENOMENOLOGY_102: (12)                        start = self.get_start()
SNGT_QHENOMENOLOGY_103: (12)                        end = tip.get_base()
SNGT_QHENOMENOLOGY_104: (8)                     self.put_start_and_end_on(start, end)
SNGT_QHENOMENOLOGY_105: (8)                     return self
SNGT_QHENOMENOLOGY_106: (4)                 def asign_tip_attr(self, tip: ArrowTip, at_start: bool)
-> Self:
SNGT_QHENOMENOLOGY_107: (8)                     if at_start:
SNGT_QHENOMENOLOGY_108: (12)                        self.start_tip = tip
SNGT_QHENOMENOLOGY_109: (8)                     else:
SNGT_QHENOMENOLOGY_110: (12)                        self.tip = tip
SNGT_QHENOMENOLOGY_111: (8)                     return self
SNGT_QHENOMENOLOGY_112: (4)                 def has_tip(self) -> bool:
SNGT_QHENOMENOLOGY_113: (8)                     return hasattr(self, "tip") and self.tip in self
SNGT_QHENOMENOLOGY_114: (4)                 def has_start_tip(self) -> bool:
SNGT_QHENOMENOLOGY_115: (8)                     return hasattr(self, "start_tip") and
self.start_tip in self
SNGT_QHENOMENOLOGY_116: (4)                 def pop_tips(self) -> VGroup:
SNGT_QHENOMENOLOGY_117: (8)                     start, end = self.get_start_and_end()
SNGT_QHENOMENOLOGY_118: (8)                     result = VGroup()
SNGT_QHENOMENOLOGY_119: (8)                     if self.has_tip():
SNGT_QHENOMENOLOGY_120: (12)                        result.add(self.tip)
SNGT_QHENOMENOLOGY_121: (12)                        self.remove(self.tip)
SNGT_QHENOMENOLOGY_122: (8)                     if self.has_start_tip():
SNGT_QHENOMENOLOGY_123: (12)                        result.add(self.start_tip)
SNGT_QHENOMENOLOGY_124: (12)                        self.remove(self.start_tip)
SNGT_QHENOMENOLOGY_125: (8)                     self.put_start_and_end_on(start, end)
SNGT_QHENOMENOLOGY_126: (8)                     return result
SNGT_QHENOMENOLOGY_127: (4)                 def get_tips(self) -> VGroup:
SNGT_QHENOMENOLOGY_128: (8)                     """
SNGT_QHENOMENOLOGY_129: (8)                     Returns a VGroup (collection of VMobjects)
containing
SNGT_QHENOMENOLOGY_130: (8)                     the TipableVMObject instance's tips.
SNGT_QHENOMENOLOGY_131: (8)                     """
SNGT_QHENOMENOLOGY_132: (8)                     result = VGroup()
SNGT_QHENOMENOLOGY_133: (8)                     if hasattr(self, "tip"):
SNGT_QHENOMENOLOGY_134: (12)                        result.add(self.tip)
SNGT_QHENOMENOLOGY_135: (8)                     if hasattr(self, "start_tip"):
```

```
SNGT_QHENOMENOLOGY_136: (12)                     result.add(self.start_tip)
SNGT_QHENOMENOLOGY_137: (8)                 return result
SNGT_QHENOMENOLOGY_138: (4)             def get_tip(self) -> ArrowTip:
SNGT_QHENOMENOLOGY_139: (8)                 """Returns the TipableVMobject instance's (first)
tip,
SNGT_QHENOMENOLOGY_140: (8)                 otherwise throws an exception."""
SNGT_QHENOMENOLOGY_141: (8)                 tips = self.get_tips()
SNGT_QHENOMENOLOGY_142: (8)                 if len(tips) == 0:
SNGT_QHENOMENOLOGY_143: (12)                    raise Exception("tip not found")
SNGT_QHENOMENOLOGY_144: (8)                 else:
SNGT_QHENOMENOLOGY_145: (12)                    return tips[0]
SNGT_QHENOMENOLOGY_146: (4)             def get_default_tip_length(self) -> float:
SNGT_QHENOMENOLOGY_147: (8)                 return self.tip_length
SNGT_QHENOMENOLOGY_148: (4)             def get_first_handle(self) -> Vect3:
SNGT_QHENOMENOLOGY_149: (8)                 return self.get_points()[1]
SNGT_QHENOMENOLOGY_150: (4)             def get_last_handle(self) -> Vect3:
SNGT_QHENOMENOLOGY_151: (8)                 return self.get_points()[-2]
SNGT_QHENOMENOLOGY_152: (4)             def get_end(self) -> Vect3:
SNGT_QHENOMENOLOGY_153: (8)                 if self.has_tip():
SNGT_QHENOMENOLOGY_154: (12)                    return self.tip.get_start()
SNGT_QHENOMENOLOGY_155: (8)                 else:
SNGT_QHENOMENOLOGY_156: (12)                    return VMobject.get_end(self)
SNGT_QHENOMENOLOGY_157: (4)             def get_start(self) -> Vect3:
SNGT_QHENOMENOLOGY_158: (8)                 if self.has_start_tip():
SNGT_QHENOMENOLOGY_159: (12)                    return self.start_tip.get_start()
SNGT_QHENOMENOLOGY_160: (8)                 else:
SNGT_QHENOMENOLOGY_161: (12)                    return VMobject.get_start(self)
SNGT_QHENOMENOLOGY_162: (4)             def get_length(self) -> float:
SNGT_QHENOMENOLOGY_163: (8)                 start, end = self.get_start_and_end()
SNGT_QHENOMENOLOGY_164: (8)                 return get_norm(start - end)
SNGT_QHENOMENOLOGY_165: (0)         class Arc(TipableVMobject):
SNGT_QHENOMENOLOGY_166: (4)             def __init__(
SNGT_QHENOMENOLOGY_167: (8)                 self,
SNGT_QHENOMENOLOGY_168: (8)                 start_angle: float = 0,
SNGT_QHENOMENOLOGY_169: (8)                 angle: float = TAU / 4,
SNGT_QHENOMENOLOGY_170: (8)                 radius: float = 1.0,
SNGT_QHENOMENOLOGY_171: (8)                 n_components: int = 8,
SNGT_QHENOMENOLOGY_172: (8)                 arc_center: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_173: (8)                 **kwargs
SNGT_QHENOMENOLOGY_174: (4)             ):
SNGT_QHENOMENOLOGY_175: (8)                 super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_176: (8)
self.set_points(quadratic_bezier_points_for_arc(angle, n_components))
SNGT_QHENOMENOLOGY_177: (8)                 self.rotate(start_angle, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_178: (8)                 self.scale(radius, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_179: (8)                 self.shift(arc_center)
SNGT_QHENOMENOLOGY_180: (4)             def get_arc_center(self) -> Vect3:
SNGT_QHENOMENOLOGY_181: (8)                 """
SNGT_QHENOMENOLOGY_182: (8)                 Looks at the normals to the first two
SNGT_QHENOMENOLOGY_183: (8)                 anchors, and finds their intersection points
SNGT_QHENOMENOLOGY_184: (8)                 """
SNGT_QHENOMENOLOGY_185: (8)                 a1, h, a2 = self.get_points()[:3]
SNGT_QHENOMENOLOGY_186: (8)                 t1 = h - a1
SNGT_QHENOMENOLOGY_187: (8)                 t2 = h - a2
SNGT_QHENOMENOLOGY_188: (8)                 n1 = rotate_vector(t1, TAU / 4)
SNGT_QHENOMENOLOGY_189: (8)                 n2 = rotate_vector(t2, TAU / 4)
SNGT_QHENOMENOLOGY_190: (8)                 return find_intersection(a1, n1, a2, n2)
SNGT_QHENOMENOLOGY_191: (4)             def get_start_angle(self) -> float:
SNGT_QHENOMENOLOGY_192: (8)                 angle = angle_of_vector(self.get_start() -
self.get_arc_center())
SNGT_QHENOMENOLOGY_193: (8)                 return angle % TAU
SNGT_QHENOMENOLOGY_194: (4)             def get_stop_angle(self) -> float:
SNGT_QHENOMENOLOGY_195: (8)                 angle = angle_of_vector(self.get_end() -
self.get_arc_center())
SNGT_QHENOMENOLOGY_196: (8)                 return angle % TAU
SNGT_QHENOMENOLOGY_197: (4)             def move_arc_center_to(self, point: Vect3) -> Self:
SNGT_QHENOMENOLOGY_198: (8)                 self.shift(point - self.get_arc_center())
SNGT_QHENOMENOLOGY_199: (8)                 return self
SNGT_QHENOMENOLOGY_200: (0)         class ArcBetweenPoints(Arc):
```

```
SNGT_QHENOMENOLOGY_201: (4)                 def __init__(
SNGT_QHENOMENOLOGY_202: (8)                     self,
SNGT_QHENOMENOLOGY_203: (8)                     start: Vect3,
SNGT_QHENOMENOLOGY_204: (8)                     end: Vect3,
SNGT_QHENOMENOLOGY_205: (8)                     angle: float = TAU / 4,
SNGT_QHENOMENOLOGY_206: (8)                     **kwargs
SNGT_QHENOMENOLOGY_207: (4)                 ):
SNGT_QHENOMENOLOGY_208: (8)                     super().__init__(angle=angle, **kwargs)
SNGT_QHENOMENOLOGY_209: (8)                     if angle == 0:
SNGT_QHENOMENOLOGY_210: (12)                        self.set_points_as_corners([LEFT, RIGHT])
SNGT_QHENOMENOLOGY_211: (8)                     self.put_start_and_end_on(start, end)
SNGT_QHENOMENOLOGY_212: (0)             class CurvedArrow(ArcBetweenPoints):
SNGT_QHENOMENOLOGY_213: (4)                 def __init__(
SNGT_QHENOMENOLOGY_214: (8)                     self,
SNGT_QHENOMENOLOGY_215: (8)                     start_point: Vect3,
SNGT_QHENOMENOLOGY_216: (8)                     end_point: Vect3,
SNGT_QHENOMENOLOGY_217: (8)                     **kwargs
SNGT_QHENOMENOLOGY_218: (4)                 ):
SNGT_QHENOMENOLOGY_219: (8)                     super().__init__(start_point, end_point, **kwargs)
SNGT_QHENOMENOLOGY_220: (8)                     self.add_tip()
SNGT_QHENOMENOLOGY_221: (0)             class CurvedDoubleArrow(CurvedArrow):
SNGT_QHENOMENOLOGY_222: (4)                 def __init__(
SNGT_QHENOMENOLOGY_223: (8)                     self,
SNGT_QHENOMENOLOGY_224: (8)                     start_point: Vect3,
SNGT_QHENOMENOLOGY_225: (8)                     end_point: Vect3,
SNGT_QHENOMENOLOGY_226: (8)                     **kwargs
SNGT_QHENOMENOLOGY_227: (4)                 ):
SNGT_QHENOMENOLOGY_228: (8)                     super().__init__(start_point, end_point, **kwargs)
SNGT_QHENOMENOLOGY_229: (8)                     self.add_tip(at_start=True)
SNGT_QHENOMENOLOGY_230: (0)             class Circle(Arc):
SNGT_QHENOMENOLOGY_231: (4)                 def __init__(
SNGT_QHENOMENOLOGY_232: (8)                     self,
SNGT_QHENOMENOLOGY_233: (8)                     start_angle: float = 0,
SNGT_QHENOMENOLOGY_234: (8)                     stroke_color: ManimColor = RED,
SNGT_QHENOMENOLOGY_235: (8)                     **kwargs
SNGT_QHENOMENOLOGY_236: (4)                 ):
SNGT_QHENOMENOLOGY_237: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_238: (12)                        start_angle, TAU,
SNGT_QHENOMENOLOGY_239: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_240: (12)                        **kwargs
SNGT_QHENOMENOLOGY_241: (8)                     )
SNGT_QHENOMENOLOGY_242: (4)                 def surround(
SNGT_QHENOMENOLOGY_243: (8)                     self,
SNGT_QHENOMENOLOGY_244: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_245: (8)                     dim_to_match: int = 0,
SNGT_QHENOMENOLOGY_246: (8)                     stretch: bool = False,
SNGT_QHENOMENOLOGY_247: (8)                     buff: float = MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_248: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_249: (8)                     self.replace(mobject, dim_to_match, stretch)
SNGT_QHENOMENOLOGY_250: (8)                     self.stretch((self.get_width() + 2 * buff) /
self.get_width(), 0)
SNGT_QHENOMENOLOGY_251: (8)                     self.stretch((self.get_height() + 2 * buff) /
self.get_height(), 1)
SNGT_QHENOMENOLOGY_252: (8)                     return self
SNGT_QHENOMENOLOGY_253: (4)                 def point_at_angle(self, angle: float) -> Vect3:
SNGT_QHENOMENOLOGY_254: (8)                     start_angle = self.get_start_angle()
SNGT_QHENOMENOLOGY_255: (8)                     return self.point_from_proportion(
SNGT_QHENOMENOLOGY_256: (12)                        ((angle - start_angle) % TAU) / TAU
SNGT_QHENOMENOLOGY_257: (8)                     )
SNGT_QHENOMENOLOGY_258: (4)                 def get_radius(self) -> float:
SNGT_QHENOMENOLOGY_259: (8)                     return get_norm(self.get_start() -
self.get_center())
SNGT_QHENOMENOLOGY_260: (0)             class Dot(Circle):
SNGT_QHENOMENOLOGY_261: (4)                 def __init__(
SNGT_QHENOMENOLOGY_262: (8)                     self,
SNGT_QHENOMENOLOGY_263: (8)                     point: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_264: (8)                     radius: float = DEFAULT_DOT_RADIUS,
SNGT_QHENOMENOLOGY_265: (8)                     stroke_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_266: (8)                     stroke_width: float = 0.0,
```

```
SNGT_QHENOMENOLOGY_267: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_268: (8)                     fill_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_269: (8)                     **kwargs
SNGT_QHENOMENOLOGY_270: (4)                 ):
SNGT_QHENOMENOLOGY_271: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_272: (12)                        arc_center=point,
SNGT_QHENOMENOLOGY_273: (12)                        radius=radius,
SNGT_QHENOMENOLOGY_274: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_275: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_276: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_277: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_278: (12)                        **kwargs
SNGT_QHENOMENOLOGY_279: (8)                     )
SNGT_QHENOMENOLOGY_280: (0)             class SmallDot(Dot):
SNGT_QHENOMENOLOGY_281: (4)                 def __init__(
SNGT_QHENOMENOLOGY_282: (8)                     self,
SNGT_QHENOMENOLOGY_283: (8)                     point: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_284: (8)                     radius: float = DEFAULT_SMALL_DOT_RADIUS,
SNGT_QHENOMENOLOGY_285: (8)                     **kwargs
SNGT_QHENOMENOLOGY_286: (4)                 ):
SNGT_QHENOMENOLOGY_287: (8)                     super().__init__(point, radius=radius, **kwargs)
SNGT_QHENOMENOLOGY_288: (0)             class Ellipse(Circle):
SNGT_QHENOMENOLOGY_289: (4)                 def __init__(
SNGT_QHENOMENOLOGY_290: (8)                     self,
SNGT_QHENOMENOLOGY_291: (8)                     width: float = 2.0,
SNGT_QHENOMENOLOGY_292: (8)                     height: float = 1.0,
SNGT_QHENOMENOLOGY_293: (8)                     **kwargs
SNGT_QHENOMENOLOGY_294: (4)                 ):
SNGT_QHENOMENOLOGY_295: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_296: (8)                     self.set_width(width, stretch=True)
SNGT_QHENOMENOLOGY_297: (8)                     self.set_height(height, stretch=True)
SNGT_QHENOMENOLOGY_298: (0)             class AnnularSector(VMobject):
SNGT_QHENOMENOLOGY_299: (4)                 def __init__(
SNGT_QHENOMENOLOGY_300: (8)                     self,
SNGT_QHENOMENOLOGY_301: (8)                     angle: float = TAU / 4,
SNGT_QHENOMENOLOGY_302: (8)                     start_angle: float = 0.0,
SNGT_QHENOMENOLOGY_303: (8)                     inner_radius: float = 1.0,
SNGT_QHENOMENOLOGY_304: (8)                     outer_radius: float = 2.0,
SNGT_QHENOMENOLOGY_305: (8)                     arc_center: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_306: (8)                     fill_color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_307: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_308: (8)                     stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_309: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_310: (4)                 ):
SNGT_QHENOMENOLOGY_311: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_312: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_313: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_314: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_315: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_316: (8)                     )
SNGT_QHENOMENOLOGY_317: (8)                     inner_arc, outer_arc = [
SNGT_QHENOMENOLOGY_318: (12)                        Arc(
SNGT_QHENOMENOLOGY_319: (16)                            start_angle=start_angle,
SNGT_QHENOMENOLOGY_320: (16)                            angle=angle,
SNGT_QHENOMENOLOGY_321: (16)                            radius=radius,
SNGT_QHENOMENOLOGY_322: (16)                            arc_center=arc_center,
SNGT_QHENOMENOLOGY_323: (12)                        )
SNGT_QHENOMENOLOGY_324: (12)                        for radius in (inner_radius, outer_radius)
SNGT_QHENOMENOLOGY_325: (8)                     ]
SNGT_QHENOMENOLOGY_326: (8)                     self.set_points(inner_arc.get_points()[::-1])  #
Reverse
SNGT_QHENOMENOLOGY_327: (8)                     self.add_line_to(outer_arc.get_points()[0])
SNGT_QHENOMENOLOGY_328: (8)                     self.add_subpath(outer_arc.get_points())
SNGT_QHENOMENOLOGY_329: (8)                     self.add_line_to(inner_arc.get_points()[-1])
SNGT_QHENOMENOLOGY_330: (0)             class Sector(AnnularSector):
SNGT_QHENOMENOLOGY_331: (4)                 def __init__(
SNGT_QHENOMENOLOGY_332: (8)                     self,
SNGT_QHENOMENOLOGY_333: (8)                     angle: float = TAU / 4,
SNGT_QHENOMENOLOGY_334: (8)                     radius: float = 1.0,
```

```
SNGT_QHENOMENOLOGY_335: (8)                     **kwargs
SNGT_QHENOMENOLOGY_336: (4)                 ):
SNGT_QHENOMENOLOGY_337: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_338: (12)                        angle,
SNGT_QHENOMENOLOGY_339: (12)                        inner_radius=0,
SNGT_QHENOMENOLOGY_340: (12)                        outer_radius=radius,
SNGT_QHENOMENOLOGY_341: (12)                        **kwargs
SNGT_QHENOMENOLOGY_342: (8)                     )
SNGT_QHENOMENOLOGY_343: (0)             class Annulus(VMobject):
SNGT_QHENOMENOLOGY_344: (4)                 def __init__(
SNGT_QHENOMENOLOGY_345: (8)                     self,
SNGT_QHENOMENOLOGY_346: (8)                     inner_radius: float = 1.0,
SNGT_QHENOMENOLOGY_347: (8)                     outer_radius: float = 2.0,
SNGT_QHENOMENOLOGY_348: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_349: (8)                     stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_350: (8)                     fill_color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_351: (8)                     center: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_352: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_353: (4)                 ):
SNGT_QHENOMENOLOGY_354: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_355: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_356: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_357: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_358: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_359: (8)                     )
SNGT_QHENOMENOLOGY_360: (8)                     self.radius = outer_radius
SNGT_QHENOMENOLOGY_361: (8)                     outer_path = outer_radius *
quadratic_bezier_points_for_arc(TAU)
SNGT_QHENOMENOLOGY_362: (8)                     inner_path = inner_radius *
quadratic_bezier_points_for_arc(-TAU)
SNGT_QHENOMENOLOGY_363: (8)                     self.add_subpath(outer_path)
SNGT_QHENOMENOLOGY_364: (8)                     self.add_subpath(inner_path)
SNGT_QHENOMENOLOGY_365: (8)                     self.shift(center)
SNGT_QHENOMENOLOGY_366: (0)             class Line(TipableVMobject):
SNGT_QHENOMENOLOGY_367: (4)                 def __init__(
SNGT_QHENOMENOLOGY_368: (8)                     self,
SNGT_QHENOMENOLOGY_369: (8)                     start: Vect3 | Mobject = LEFT,
SNGT_QHENOMENOLOGY_370: (8)                     end: Vect3 | Mobject = RIGHT,
SNGT_QHENOMENOLOGY_371: (8)                     buff: float = 0.0,
SNGT_QHENOMENOLOGY_372: (8)                     path_arc: float = 0.0,
SNGT_QHENOMENOLOGY_373: (8)                     **kwargs
SNGT_QHENOMENOLOGY_374: (4)                 ):
SNGT_QHENOMENOLOGY_375: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_376: (8)                     self.path_arc = path_arc
SNGT_QHENOMENOLOGY_377: (8)                     self.buff = buff
SNGT_QHENOMENOLOGY_378: (8)                     self.set_start_and_end_attrs(start, end)
SNGT_QHENOMENOLOGY_379: (8)                     self.set_points_by_ends(self.start, self.end, buff,
path_arc)
SNGT_QHENOMENOLOGY_380: (4)                 def set_points_by_ends(
SNGT_QHENOMENOLOGY_381: (8)                     self,
SNGT_QHENOMENOLOGY_382: (8)                     start: Vect3,
SNGT_QHENOMENOLOGY_383: (8)                     end: Vect3,
SNGT_QHENOMENOLOGY_384: (8)                     buff: float = 0,
SNGT_QHENOMENOLOGY_385: (8)                     path_arc: float = 0
SNGT_QHENOMENOLOGY_386: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_387: (8)                     self.clear_points()
SNGT_QHENOMENOLOGY_388: (8)                     self.start_new_path(start)
SNGT_QHENOMENOLOGY_389: (8)                     self.add_arc_to(end, path_arc)
SNGT_QHENOMENOLOGY_390: (8)                     if buff > 0:
SNGT_QHENOMENOLOGY_391: (12)                        length = self.get_arc_length()
SNGT_QHENOMENOLOGY_392: (12)                        alpha = min(buff / length, 0.5)
SNGT_QHENOMENOLOGY_393: (12)                        self.pointwise_become_partial(self, alpha, 1 -
alpha)
SNGT_QHENOMENOLOGY_394: (8)                     return self
SNGT_QHENOMENOLOGY_395: (4)                 def set_path_arc(self, new_value: float) -> Self:
SNGT_QHENOMENOLOGY_396: (8)                     self.path_arc = new_value
SNGT_QHENOMENOLOGY_397: (8)                     self.init_points()
SNGT_QHENOMENOLOGY_398: (8)                     return self
SNGT_QHENOMENOLOGY_399: (4)                 def set_start_and_end_attrs(self, start: Vect3 |
```

```
Mobject, end: Vect3 | Mobject):
SNGT_QHENOMENOLOGY_400: (8)                     rough_start = self.pointify(start)
SNGT_QHENOMENOLOGY_401: (8)                     rough_end = self.pointify(end)
SNGT_QHENOMENOLOGY_402: (8)                     vect = normalize(rough_end - rough_start)
SNGT_QHENOMENOLOGY_403: (8)                     self.start = self.pointify(start, vect)
SNGT_QHENOMENOLOGY_404: (8)                     self.end = self.pointify(end, -vect)
SNGT_QHENOMENOLOGY_405: (4)                 def pointify(
SNGT_QHENOMENOLOGY_406: (8)                     self,
SNGT_QHENOMENOLOGY_407: (8)                     mob_or_point: Mobject | Vect3,
SNGT_QHENOMENOLOGY_408: (8)                     direction: Vect3 | None = None
SNGT_QHENOMENOLOGY_409: (4)                 ) -> Vect3:
SNGT_QHENOMENOLOGY_410: (8)                     """
SNGT_QHENOMENOLOGY_411: (8)                     Take an argument passed into Line (or subclass) and
turn
SNGT_QHENOMENOLOGY_412: (8)                     it into a 3d point.
SNGT_QHENOMENOLOGY_413: (8)                     """
SNGT_QHENOMENOLOGY_414: (8)                     if isinstance(mob_or_point, Mobject):
SNGT_QHENOMENOLOGY_415: (12)                        mob = mob_or_point
SNGT_QHENOMENOLOGY_416: (12)                        if direction is None:
SNGT_QHENOMENOLOGY_417: (16)                            return mob.get_center()
SNGT_QHENOMENOLOGY_418: (12)                        else:
SNGT_QHENOMENOLOGY_419: (16)                            return
mob.get_continuous_bounding_box_point(direction)
SNGT_QHENOMENOLOGY_420: (8)                     else:
SNGT_QHENOMENOLOGY_421: (12)                        point = mob_or_point
SNGT_QHENOMENOLOGY_422: (12)                        result = np.zeros(self.dim)
SNGT_QHENOMENOLOGY_423: (12)                        result[:len(point)] = point
SNGT_QHENOMENOLOGY_424: (12)                        return result
SNGT_QHENOMENOLOGY_425: (4)                 def put_start_and_end_on(self, start: Vect3, end:
Vect3) -> Self:
SNGT_QHENOMENOLOGY_426: (8)                     curr_start, curr_end = self.get_start_and_end()
SNGT_QHENOMENOLOGY_427: (8)                     if np.isclose(curr_start, curr_end).all():
SNGT_QHENOMENOLOGY_428: (12)                        self.set_points_by_ends(start, end, buff=0,
path_arc=self.path_arc)
SNGT_QHENOMENOLOGY_429: (12)                        return self
SNGT_QHENOMENOLOGY_430: (8)                     return super().put_start_and_end_on(start, end)
SNGT_QHENOMENOLOGY_431: (4)                 def get_vector(self) -> Vect3:
SNGT_QHENOMENOLOGY_432: (8)                     return self.get_end() - self.get_start()
SNGT_QHENOMENOLOGY_433: (4)                 def get_unit_vector(self) -> Vect3:
SNGT_QHENOMENOLOGY_434: (8)                     return normalize(self.get_vector())
SNGT_QHENOMENOLOGY_435: (4)                 def get_angle(self) -> float:
SNGT_QHENOMENOLOGY_436: (8)                     return angle_of_vector(self.get_vector())
SNGT_QHENOMENOLOGY_437: (4)                 def get_projection(self, point: Vect3) -> Vect3:
SNGT_QHENOMENOLOGY_438: (8)                     """
SNGT_QHENOMENOLOGY_439: (8)                     Return projection of a point onto the line
SNGT_QHENOMENOLOGY_440: (8)                     """
SNGT_QHENOMENOLOGY_441: (8)                     unit_vect = self.get_unit_vector()
SNGT_QHENOMENOLOGY_442: (8)                     start = self.get_start()
SNGT_QHENOMENOLOGY_443: (8)                     return start + np.dot(point - start, unit_vect) *
unit_vect
SNGT_QHENOMENOLOGY_444: (4)                 def get_slope(self) -> float:
SNGT_QHENOMENOLOGY_445: (8)                     return np.tan(self.get_angle())
SNGT_QHENOMENOLOGY_446: (4)                 def set_angle(self, angle: float, about_point:
Optional[Vect3] = None) -> Self:
SNGT_QHENOMENOLOGY_447: (8)                     if about_point is None:
SNGT_QHENOMENOLOGY_448: (12)                        about_point = self.get_start()
SNGT_QHENOMENOLOGY_449: (8)                     self.rotate(
SNGT_QHENOMENOLOGY_450: (12)                        angle - self.get_angle(),
SNGT_QHENOMENOLOGY_451: (12)                        about_point=about_point,
SNGT_QHENOMENOLOGY_452: (8)                     )
SNGT_QHENOMENOLOGY_453: (8)                     return self
SNGT_QHENOMENOLOGY_454: (4)                 def set_length(self, length: float, **kwargs):
SNGT_QHENOMENOLOGY_455: (8)                     self.scale(length / self.get_length(), **kwargs)
SNGT_QHENOMENOLOGY_456: (8)                     return self
SNGT_QHENOMENOLOGY_457: (4)                 def get_arc_length(self) -> float:
SNGT_QHENOMENOLOGY_458: (8)                     arc_len = get_norm(self.get_vector())
SNGT_QHENOMENOLOGY_459: (8)                     if self.path_arc > 0:
SNGT_QHENOMENOLOGY_460: (12)                        arc_len *= self.path_arc / (2 *
math.sin(self.path_arc / 2))
```

```
SNGT_QHENOMENOLOGY_461: (8)                     return arc_len
SNGT_QHENOMENOLOGY_462: (0)             class DashedLine(Line):
SNGT_QHENOMENOLOGY_463: (4)                 def __init__(
SNGT_QHENOMENOLOGY_464: (8)                     self,
SNGT_QHENOMENOLOGY_465: (8)                     start: Vect3 = LEFT,
SNGT_QHENOMENOLOGY_466: (8)                     end: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_467: (8)                     dash_length: float = DEFAULT_DASH_LENGTH,
SNGT_QHENOMENOLOGY_468: (8)                     positive_space_ratio: float = 0.5,
SNGT_QHENOMENOLOGY_469: (8)                     **kwargs
SNGT_QHENOMENOLOGY_470: (4)                 ):
SNGT_QHENOMENOLOGY_471: (8)                     super().__init__(start, end, **kwargs)
SNGT_QHENOMENOLOGY_472: (8)                     num_dashes = self.calculate_num_dashes(dash_length,
positive_space_ratio)
SNGT_QHENOMENOLOGY_473: (8)                     dashes = DashedVMobject(
SNGT_QHENOMENOLOGY_474: (12)                        self,
SNGT_QHENOMENOLOGY_475: (12)                        num_dashes=num_dashes,
SNGT_QHENOMENOLOGY_476: (12)                        positive_space_ratio=positive_space_ratio
SNGT_QHENOMENOLOGY_477: (8)                     )
SNGT_QHENOMENOLOGY_478: (8)                     self.clear_points()
SNGT_QHENOMENOLOGY_479: (8)                     self.add(*dashes)
SNGT_QHENOMENOLOGY_480: (4)                 def calculate_num_dashes(self, dash_length: float,
positive_space_ratio: float) -> int:
SNGT_QHENOMENOLOGY_481: (8)                     try:
SNGT_QHENOMENOLOGY_482: (12)                        full_length = dash_length /
positive_space_ratio
SNGT_QHENOMENOLOGY_483: (12)                        return int(np.ceil(self.get_length() /
full_length))
SNGT_QHENOMENOLOGY_484: (8)                     except ZeroDivisionError:
SNGT_QHENOMENOLOGY_485: (12)                        return 1
SNGT_QHENOMENOLOGY_486: (4)                 def get_start(self) -> Vect3:
SNGT_QHENOMENOLOGY_487: (8)                     if len(self.submobjects) > 0:
SNGT_QHENOMENOLOGY_488: (12)                        return self.submobjects[0].get_start()
SNGT_QHENOMENOLOGY_489: (8)                     else:
SNGT_QHENOMENOLOGY_490: (12)                        return Line.get_start(self)
SNGT_QHENOMENOLOGY_491: (4)                 def get_end(self) -> Vect3:
SNGT_QHENOMENOLOGY_492: (8)                     if len(self.submobjects) > 0:
SNGT_QHENOMENOLOGY_493: (12)                        return self.submobjects[-1].get_end()
SNGT_QHENOMENOLOGY_494: (8)                     else:
SNGT_QHENOMENOLOGY_495: (12)                        return Line.get_end(self)
SNGT_QHENOMENOLOGY_496: (4)                 def get_first_handle(self) -> Vect3:
SNGT_QHENOMENOLOGY_497: (8)                     return self.submobjects[0].get_points()[1]
SNGT_QHENOMENOLOGY_498: (4)                 def get_last_handle(self) -> Vect3:
SNGT_QHENOMENOLOGY_499: (8)                     return self.submobjects[-1].get_points()[-2]
SNGT_QHENOMENOLOGY_500: (0)             class TangentLine(Line):
SNGT_QHENOMENOLOGY_501: (4)                 def __init__(
SNGT_QHENOMENOLOGY_502: (8)                     self,
SNGT_QHENOMENOLOGY_503: (8)                     vmob: VMobject,
SNGT_QHENOMENOLOGY_504: (8)                     alpha: float,
SNGT_QHENOMENOLOGY_505: (8)                     length: float = 2,
SNGT_QHENOMENOLOGY_506: (8)                     d_alpha: float = 1e-6,
SNGT_QHENOMENOLOGY_507: (8)                     **kwargs
SNGT_QHENOMENOLOGY_508: (4)                 ):
SNGT_QHENOMENOLOGY_509: (8)                     a1 = clip(alpha - d_alpha, 0, 1)
SNGT_QHENOMENOLOGY_510: (8)                     a2 = clip(alpha + d_alpha, 0, 1)
SNGT_QHENOMENOLOGY_511: (8)                     super().__init__(vmob.pfp(a1), vmob.pfp(a2),
**kwargs)
SNGT_QHENOMENOLOGY_512: (8)                     self.scale(length / self.get_length())
SNGT_QHENOMENOLOGY_513: (0)             class Elbow(VMobject):
SNGT_QHENOMENOLOGY_514: (4)                 def __init__(
SNGT_QHENOMENOLOGY_515: (8)                     self,
SNGT_QHENOMENOLOGY_516: (8)                     width: float = 0.2,
SNGT_QHENOMENOLOGY_517: (8)                     angle: float = 0,
SNGT_QHENOMENOLOGY_518: (8)                     **kwargs
SNGT_QHENOMENOLOGY_519: (4)                 ):
SNGT_QHENOMENOLOGY_520: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_521: (8)                     self.set_points_as_corners([UP, UR, RIGHT])
SNGT_QHENOMENOLOGY_522: (8)                     self.set_width(width, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_523: (8)                     self.rotate(angle, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_524: (0)             class StrokeArrow(Line):
```

```
SNGT_QHENOMENOLOGY_525: (4)                 def __init__(
SNGT_QHENOMENOLOGY_526: (8)                     self,
SNGT_QHENOMENOLOGY_527: (8)                     start: Vect3 | Mobject,
SNGT_QHENOMENOLOGY_528: (8)                     end: Vect3 | Mobject,
SNGT_QHENOMENOLOGY_529: (8)                     stroke_color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_530: (8)                     stroke_width: float = 5,
SNGT_QHENOMENOLOGY_531: (8)                     buff: float = 0.25,
SNGT_QHENOMENOLOGY_532: (8)                     tip_width_ratio: float = 5,
SNGT_QHENOMENOLOGY_533: (8)                     tip_len_to_width: float = 0.0075,
SNGT_QHENOMENOLOGY_534: (8)                     max_tip_length_to_length_ratio: float = 0.3,
SNGT_QHENOMENOLOGY_535: (8)                     max_width_to_length_ratio: float = 8.0,
SNGT_QHENOMENOLOGY_536: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_537: (4)                 ):
SNGT_QHENOMENOLOGY_538: (8)                     self.tip_width_ratio = tip_width_ratio
SNGT_QHENOMENOLOGY_539: (8)                     self.tip_len_to_width = tip_len_to_width
SNGT_QHENOMENOLOGY_540: (8)                     self.max_tip_length_to_length_ratio =
max_tip_length_to_length_ratio
SNGT_QHENOMENOLOGY_541: (8)                     self.max_width_to_length_ratio =
max_width_to_length_ratio
SNGT_QHENOMENOLOGY_542: (8)                     self.n_tip_points = 3
SNGT_QHENOMENOLOGY_543: (8)                     self.original_stroke_width = stroke_width
SNGT_QHENOMENOLOGY_544: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_545: (12)                        start, end,
SNGT_QHENOMENOLOGY_546: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_547: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_548: (12)                        buff=buff,
SNGT_QHENOMENOLOGY_549: (12)                        **kwargs
SNGT_QHENOMENOLOGY_550: (8)                     )
SNGT_QHENOMENOLOGY_551: (4)                 def set_points_by_ends(
SNGT_QHENOMENOLOGY_552: (8)                     self,
SNGT_QHENOMENOLOGY_553: (8)                     start: Vect3,
SNGT_QHENOMENOLOGY_554: (8)                     end: Vect3,
SNGT_QHENOMENOLOGY_555: (8)                     buff: float = 0,
SNGT_QHENOMENOLOGY_556: (8)                     path_arc: float = 0
SNGT_QHENOMENOLOGY_557: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_558: (8)                     super().set_points_by_ends(start, end, buff,
path_arc)
SNGT_QHENOMENOLOGY_559: (8)                     self.insert_tip_anchor()
SNGT_QHENOMENOLOGY_560: (8)                     self.create_tip_with_stroke_width()
SNGT_QHENOMENOLOGY_561: (8)                     return self
SNGT_QHENOMENOLOGY_562: (4)                 def insert_tip_anchor(self) -> Self:
SNGT_QHENOMENOLOGY_563: (8)                     prev_end = self.get_end()
SNGT_QHENOMENOLOGY_564: (8)                     arc_len = self.get_arc_length()
SNGT_QHENOMENOLOGY_565: (8)                     tip_len = self.get_stroke_width() *
self.tip_width_ratio * self.tip_len_to_width
SNGT_QHENOMENOLOGY_566: (8)                     if tip_len >= self.max_tip_length_to_length_ratio *
arc_len or arc_len == 0:
SNGT_QHENOMENOLOGY_567: (12)                        alpha = self.max_tip_length_to_length_ratio
SNGT_QHENOMENOLOGY_568: (8)                     else:
SNGT_QHENOMENOLOGY_569: (12)                        alpha = tip_len / arc_len
SNGT_QHENOMENOLOGY_570: (8)                     if self.path_arc > 0 and self.buff > 0:
SNGT_QHENOMENOLOGY_571: (12)                        self.insert_n_curves(10)  # Is this needed?
SNGT_QHENOMENOLOGY_572: (8)                     self.pointwise_become_partial(self, 0.0, 1.0 -
alpha)
SNGT_QHENOMENOLOGY_573: (8)                     self.add_line_to(self.get_end())
SNGT_QHENOMENOLOGY_574: (8)                     self.add_line_to(prev_end)
SNGT_QHENOMENOLOGY_575: (8)                     self.n_tip_points = 3
SNGT_QHENOMENOLOGY_576: (8)                     return self
SNGT_QHENOMENOLOGY_577: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_578: (4)                 def create_tip_with_stroke_width(self) -> Self:
SNGT_QHENOMENOLOGY_579: (8)                     if self.get_num_points() < 3:
SNGT_QHENOMENOLOGY_580: (12)                        return self
SNGT_QHENOMENOLOGY_581: (8)                     stroke_width = min(
SNGT_QHENOMENOLOGY_582: (12)                        self.original_stroke_width,
SNGT_QHENOMENOLOGY_583: (12)                        self.max_width_to_length_ratio *
self.get_length(),
SNGT_QHENOMENOLOGY_584: (8)                     )
SNGT_QHENOMENOLOGY_585: (8)                     tip_width = self.tip_width_ratio * stroke_width
SNGT_QHENOMENOLOGY_586: (8)                     ntp = self.n_tip_points
```

```
SNGT_QHENOMENOLOGY_587: (8)                     self.data['stroke_width'][:-ntp] =
self.data['stroke_width'][0]
SNGT_QHENOMENOLOGY_588: (8)                     self.data['stroke_width'][-ntp:, 0] = tip_width *
np.linspace(1, 0, ntp)
SNGT_QHENOMENOLOGY_589: (8)                     return self
SNGT_QHENOMENOLOGY_590: (4)                 def reset_tip(self) -> Self:
SNGT_QHENOMENOLOGY_591: (8)                     self.set_points_by_ends(
SNGT_QHENOMENOLOGY_592: (12)                        self.get_start(), self.get_end(),
SNGT_QHENOMENOLOGY_593: (12)                        path_arc=self.path_arc
SNGT_QHENOMENOLOGY_594: (8)                     )
SNGT_QHENOMENOLOGY_595: (8)                     return self
SNGT_QHENOMENOLOGY_596: (4)                 def set_stroke(
SNGT_QHENOMENOLOGY_597: (8)                     self,
SNGT_QHENOMENOLOGY_598: (8)                     color: ManimColor | Iterable[ManimColor] | None =
None,
SNGT_QHENOMENOLOGY_599: (8)                     width: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_600: (8)                     *args, **kwargs
SNGT_QHENOMENOLOGY_601: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_602: (8)                     super().set_stroke(color=color, width=width, *args,
**kwargs)
SNGT_QHENOMENOLOGY_603: (8)                     self.original_stroke_width =
self.get_stroke_width()
SNGT_QHENOMENOLOGY_604: (8)                     if self.has_points():
SNGT_QHENOMENOLOGY_605: (12)                        self.reset_tip()
SNGT_QHENOMENOLOGY_606: (8)                     return self
SNGT_QHENOMENOLOGY_607: (4)                 def _handle_scale_side_effects(self, scale_factor:
float) -> Self:
SNGT_QHENOMENOLOGY_608: (8)                     if scale_factor != 1.0:
SNGT_QHENOMENOLOGY_609: (12)                        self.reset_tip()
SNGT_QHENOMENOLOGY_610: (8)                     return self
SNGT_QHENOMENOLOGY_611: (0)             class Arrow(Line):
SNGT_QHENOMENOLOGY_612: (4)                 tickness_multiplier = 0.015
SNGT_QHENOMENOLOGY_613: (4)                 def __init__(
SNGT_QHENOMENOLOGY_614: (8)                     self,
SNGT_QHENOMENOLOGY_615: (8)                     start: Vect3 | Mobject = LEFT,
SNGT_QHENOMENOLOGY_616: (8)                     end: Vect3 | Mobject = LEFT,
SNGT_QHENOMENOLOGY_617: (8)                     buff: float = MED_SMALL_BUFF,
SNGT_QHENOMENOLOGY_618: (8)                     path_arc: float = 0,
SNGT_QHENOMENOLOGY_619: (8)                     fill_color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_620: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_621: (8)                     stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_622: (8)                     thickness: float = 3.0,
SNGT_QHENOMENOLOGY_623: (8)                     tip_width_ratio: float = 5,
SNGT_QHENOMENOLOGY_624: (8)                     tip_angle: float = PI / 3,
SNGT_QHENOMENOLOGY_625: (8)                     max_tip_length_to_length_ratio: float = 0.5,
SNGT_QHENOMENOLOGY_626: (8)                     max_width_to_length_ratio: float = 0.1,
SNGT_QHENOMENOLOGY_627: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_628: (4)                 ):
SNGT_QHENOMENOLOGY_629: (8)                     self.thickness = thickness
SNGT_QHENOMENOLOGY_630: (8)                     self.tip_width_ratio = tip_width_ratio
SNGT_QHENOMENOLOGY_631: (8)                     self.tip_angle = tip_angle
SNGT_QHENOMENOLOGY_632: (8)                     self.max_tip_length_to_length_ratio =
max_tip_length_to_length_ratio
SNGT_QHENOMENOLOGY_633: (8)                     self.max_width_to_length_ratio =
max_width_to_length_ratio
SNGT_QHENOMENOLOGY_634: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_635: (12)                        start, end,
SNGT_QHENOMENOLOGY_636: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_637: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_638: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_639: (12)                        buff=buff,
SNGT_QHENOMENOLOGY_640: (12)                        path_arc=path_arc,
SNGT_QHENOMENOLOGY_641: (12)                        **kwargs
SNGT_QHENOMENOLOGY_642: (8)                     )
SNGT_QHENOMENOLOGY_643: (4)                 def get_key_dimensions(self, length):
SNGT_QHENOMENOLOGY_644: (8)                     width = self.thickness * self.tickness_multiplier
SNGT_QHENOMENOLOGY_645: (8)                     w_ratio = fdiv(self.max_width_to_length_ratio,
fdiv(width, length))
SNGT_QHENOMENOLOGY_646: (8)                     if w_ratio < 1:
```

```
SNGT_QHENOMENOLOGY_647: (12)                    width *= w_ratio
SNGT_QHENOMENOLOGY_648: (8)                 tip_width = self.tip_width_ratio * width
SNGT_QHENOMENOLOGY_649: (8)                 tip_length = tip_width / (2 * np.tan(self.tip_angle
/ 2))
SNGT_QHENOMENOLOGY_650: (8)                 t_ratio = fdiv(self.max_tip_length_to_length_ratio,
fdiv(tip_length, length))
SNGT_QHENOMENOLOGY_651: (8)                 if t_ratio < 1:
SNGT_QHENOMENOLOGY_652: (12)                    tip_length *= t_ratio
SNGT_QHENOMENOLOGY_653: (12)                    tip_width *= t_ratio
SNGT_QHENOMENOLOGY_654: (8)                 return width, tip_width, tip_length
SNGT_QHENOMENOLOGY_655: (4)             def set_points_by_ends(
SNGT_QHENOMENOLOGY_656: (8)                 self,
SNGT_QHENOMENOLOGY_657: (8)                 start: Vect3,
SNGT_QHENOMENOLOGY_658: (8)                 end: Vect3,
SNGT_QHENOMENOLOGY_659: (8)                 buff: float = 0,
SNGT_QHENOMENOLOGY_660: (8)                 path_arc: float = 0
SNGT_QHENOMENOLOGY_661: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_662: (8)                 vect = end - start
SNGT_QHENOMENOLOGY_663: (8)                 length = max(get_norm(vect), 1e-8)  # More
systematic min?
SNGT_QHENOMENOLOGY_664: (8)                 unit_vect = normalize(vect)
SNGT_QHENOMENOLOGY_665: (8)                 width, tip_width, tip_length =
self.get_key_dimensions(length - buff)
SNGT_QHENOMENOLOGY_666: (8)                 if path_arc == 0:
SNGT_QHENOMENOLOGY_667: (12)                    start = start + buff * unit_vect
SNGT_QHENOMENOLOGY_668: (12)                    end = end - buff * unit_vect
SNGT_QHENOMENOLOGY_669: (8)                 else:
SNGT_QHENOMENOLOGY_670: (12)                    R = length / 2 / math.sin(path_arc / 2)
SNGT_QHENOMENOLOGY_671: (12)                    midpoint = 0.5 * (start + end)
SNGT_QHENOMENOLOGY_672: (12)                    center = midpoint + rotate_vector(0.5 * vect,
PI / 2) / math.tan(path_arc / 2)
SNGT_QHENOMENOLOGY_673: (12)                    sign = 1
SNGT_QHENOMENOLOGY_674: (12)                    start = center + rotate_vector(start - center,
buff / R)
SNGT_QHENOMENOLOGY_675: (12)                    end = center + rotate_vector(end - center, -
buff / R)
SNGT_QHENOMENOLOGY_676: (12)                    path_arc -= (2 * buff + tip_length) / R
SNGT_QHENOMENOLOGY_677: (8)                 vect = end - start
SNGT_QHENOMENOLOGY_678: (8)                 length = get_norm(vect)
SNGT_QHENOMENOLOGY_679: (8)                 if path_arc == 0:
SNGT_QHENOMENOLOGY_680: (12)                    points1 = (length - tip_length) *
np.array([RIGHT, 0.5 * RIGHT, ORIGIN])
SNGT_QHENOMENOLOGY_681: (12)                    points1 += width * UP / 2
SNGT_QHENOMENOLOGY_682: (12)                    points2 = points1[::-1] + width * DOWN
SNGT_QHENOMENOLOGY_683: (8)                 else:
SNGT_QHENOMENOLOGY_684: (12)                    points1 =
quadratic_bezier_points_for_arc(path_arc)
SNGT_QHENOMENOLOGY_685: (12)                    points2 = np.array(points1[::-1])
SNGT_QHENOMENOLOGY_686: (12)                    points1 *= (R + width / 2)
SNGT_QHENOMENOLOGY_687: (12)                    points2 *= (R - width / 2)
SNGT_QHENOMENOLOGY_688: (12)                    rot_T = rotation_matrix_transpose(PI / 2 -
path_arc, OUT)
SNGT_QHENOMENOLOGY_689: (12)                    for points in points1, points2:
SNGT_QHENOMENOLOGY_690: (16)                        points[:] = np.dot(points, rot_T)
SNGT_QHENOMENOLOGY_691: (16)                        points += R * DOWN
SNGT_QHENOMENOLOGY_692: (8)                 self.set_points(points1)
SNGT_QHENOMENOLOGY_693: (8)                 self.add_line_to(tip_width * UP / 2)
SNGT_QHENOMENOLOGY_694: (8)                 self.add_line_to(tip_length * LEFT)
SNGT_QHENOMENOLOGY_695: (8)                 self.tip_index = len(self.get_points()) - 1
SNGT_QHENOMENOLOGY_696: (8)                 self.add_line_to(tip_width * DOWN / 2)
SNGT_QHENOMENOLOGY_697: (8)                 self.add_line_to(points2[0])
SNGT_QHENOMENOLOGY_698: (8)                 self.add_subpath(points2)
SNGT_QHENOMENOLOGY_699: (8)                 self.add_line_to(points1[0])
SNGT_QHENOMENOLOGY_700: (8)                 self.rotate(angle_of_vector(vect) -
self.get_angle())
SNGT_QHENOMENOLOGY_701: (8)                 self.rotate(
SNGT_QHENOMENOLOGY_702: (12)                    PI / 2 - np.arccos(normalize(vect)[2]),
SNGT_QHENOMENOLOGY_703: (12)                    axis=rotate_vector(self.get_unit_vector(), -PI
/ 2),
```

```
SNGT_QHENOMENOLOGY_704: (8)                     )
SNGT_QHENOMENOLOGY_705: (8)                     self.shift(start - self.get_start())
SNGT_QHENOMENOLOGY_706: (8)                     return self
SNGT_QHENOMENOLOGY_707: (4)                 def reset_points_around_ends(self) -> Self:
SNGT_QHENOMENOLOGY_708: (8)                     self.set_points_by_ends(
SNGT_QHENOMENOLOGY_709: (12)                        self.get_start().copy(),
SNGT_QHENOMENOLOGY_710: (12)                        self.get_end().copy(),
SNGT_QHENOMENOLOGY_711: (12)                        path_arc=self.path_arc
SNGT_QHENOMENOLOGY_712: (8)                     )
SNGT_QHENOMENOLOGY_713: (8)                     return self
SNGT_QHENOMENOLOGY_714: (4)                 def get_start(self) -> Vect3:
SNGT_QHENOMENOLOGY_715: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_716: (8)                     return 0.5 * (points[0] + points[-3])
SNGT_QHENOMENOLOGY_717: (4)                 def get_end(self) -> Vect3:
SNGT_QHENOMENOLOGY_718: (8)                     return self.get_points()[self.tip_index]
SNGT_QHENOMENOLOGY_719: (4)                 def get_start_and_end(self):
SNGT_QHENOMENOLOGY_720: (8)                     return (self.get_start(), self.get_end())
SNGT_QHENOMENOLOGY_721: (4)                 def put_start_and_end_on(self, start: Vect3, end:
Vect3) -> Self:
SNGT_QHENOMENOLOGY_722: (8)                     self.set_points_by_ends(start, end, buff=0,
path_arc=self.path_arc)
SNGT_QHENOMENOLOGY_723: (8)                     return self
SNGT_QHENOMENOLOGY_724: (4)                 def scale(self, *args, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_725: (8)                     super().scale(*args, **kwargs)
SNGT_QHENOMENOLOGY_726: (8)                     self.reset_points_around_ends()
SNGT_QHENOMENOLOGY_727: (8)                     return self
SNGT_QHENOMENOLOGY_728: (4)                 def set_thickness(self, thickness: float) -> Self:
SNGT_QHENOMENOLOGY_729: (8)                     self.thickness = thickness
SNGT_QHENOMENOLOGY_730: (8)                     self.reset_points_around_ends()
SNGT_QHENOMENOLOGY_731: (8)                     return self
SNGT_QHENOMENOLOGY_732: (4)                 def set_path_arc(self, path_arc: float) -> Self:
SNGT_QHENOMENOLOGY_733: (8)                     self.path_arc = path_arc
SNGT_QHENOMENOLOGY_734: (8)                     self.reset_points_around_ends()
SNGT_QHENOMENOLOGY_735: (8)                     return self
SNGT_QHENOMENOLOGY_736: (4)                 def set_perpendicular_to_camera(self, camera_frame):
SNGT_QHENOMENOLOGY_737: (8)                     to_cam = camera_frame.get_implied_camera_location()
- self.get_center()
SNGT_QHENOMENOLOGY_738: (8)                     normal = self.get_unit_normal()
SNGT_QHENOMENOLOGY_739: (8)                     axis = normalize(self.get_vector())
SNGT_QHENOMENOLOGY_740: (8)                     trg_normal = to_cam - np.dot(to_cam, axis) * axis
SNGT_QHENOMENOLOGY_741: (8)                     mat = rotation_between_vectors(normal, trg_normal)
SNGT_QHENOMENOLOGY_742: (8)                     self.apply_matrix(mat,
about_point=self.get_start())
SNGT_QHENOMENOLOGY_743: (8)                     return self
SNGT_QHENOMENOLOGY_744: (0)             class Vector(Arrow):
SNGT_QHENOMENOLOGY_745: (4)                 def __init__(
SNGT_QHENOMENOLOGY_746: (8)                     self,
SNGT_QHENOMENOLOGY_747: (8)                     direction: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_748: (8)                     buff: float = 0.0,
SNGT_QHENOMENOLOGY_749: (8)                     **kwargs
SNGT_QHENOMENOLOGY_750: (4)                 ):
SNGT_QHENOMENOLOGY_751: (8)                     if len(direction) == 2:
SNGT_QHENOMENOLOGY_752: (12)                        direction = np.hstack([direction, 0])
SNGT_QHENOMENOLOGY_753: (8)                     super().__init__(ORIGIN, direction, buff=buff,
**kwargs)
SNGT_QHENOMENOLOGY_754: (0)             class CubicBezier(VMobject):
SNGT_QHENOMENOLOGY_755: (4)                 def __init__(
SNGT_QHENOMENOLOGY_756: (8)                     self,
SNGT_QHENOMENOLOGY_757: (8)                     a0: Vect3,
SNGT_QHENOMENOLOGY_758: (8)                     h0: Vect3,
SNGT_QHENOMENOLOGY_759: (8)                     h1: Vect3,
SNGT_QHENOMENOLOGY_760: (8)                     a1: Vect3,
SNGT_QHENOMENOLOGY_761: (8)                     **kwargs
SNGT_QHENOMENOLOGY_762: (4)                 ):
SNGT_QHENOMENOLOGY_763: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_764: (8)                     self.add_cubic_bezier_curve(a0, h0, h1, a1)
SNGT_QHENOMENOLOGY_765: (0)             class Polygon(VMobject):
SNGT_QHENOMENOLOGY_766: (4)                 def __init__(
SNGT_QHENOMENOLOGY_767: (8)                     self,
```

```
SNGT_QHENOMENOLOGY_768: (8)                     *vertices: Vect3,
SNGT_QHENOMENOLOGY_769: (8)                     **kwargs
SNGT_QHENOMENOLOGY_770: (4)                 ):
SNGT_QHENOMENOLOGY_771: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_772: (8)                     self.set_points_as_corners([*vertices,
vertices[0]])
SNGT_QHENOMENOLOGY_773: (4)                 def get_vertices(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_774: (8)                     return self.get_start_anchors()
SNGT_QHENOMENOLOGY_775: (4)                 def round_corners(self, radius: Optional[float] = None)
-> Self:
SNGT_QHENOMENOLOGY_776: (8)                     if radius is None:
SNGT_QHENOMENOLOGY_777: (12)                        verts = self.get_vertices()
SNGT_QHENOMENOLOGY_778: (12)                        min_edge_length = min(
SNGT_QHENOMENOLOGY_779: (16)                            get_norm(v1 - v2)
SNGT_QHENOMENOLOGY_780: (16)                            for v1, v2 in zip(verts, verts[1:])
SNGT_QHENOMENOLOGY_781: (16)                            if not np.isclose(v1, v2).all()
SNGT_QHENOMENOLOGY_782: (12)                        )
SNGT_QHENOMENOLOGY_783: (12)                        radius = 0.25 * min_edge_length
SNGT_QHENOMENOLOGY_784: (8)                     vertices = self.get_vertices()
SNGT_QHENOMENOLOGY_785: (8)                     arcs = []
SNGT_QHENOMENOLOGY_786: (8)                     for v1, v2, v3 in adjacent_n_tuples(vertices, 3):
SNGT_QHENOMENOLOGY_787: (12)                        vect1 = normalize(v2 - v1)
SNGT_QHENOMENOLOGY_788: (12)                        vect2 = normalize(v3 - v2)
SNGT_QHENOMENOLOGY_789: (12)                        angle = angle_between_vectors(vect1, vect2)
SNGT_QHENOMENOLOGY_790: (12)                        cut_off_length = radius * np.tan(angle / 2)
SNGT_QHENOMENOLOGY_791: (12)                        sign = float(np.sign(radius * cross2d(vect1,
vect2)))
SNGT_QHENOMENOLOGY_792: (12)                        arc = ArcBetweenPoints(
SNGT_QHENOMENOLOGY_793: (16)                            v2 - vect1 * cut_off_length,
SNGT_QHENOMENOLOGY_794: (16)                            v2 + vect2 * cut_off_length,
SNGT_QHENOMENOLOGY_795: (16)                            angle=sign * angle,
SNGT_QHENOMENOLOGY_796: (16)                            n_components=2,
SNGT_QHENOMENOLOGY_797: (12)                        )
SNGT_QHENOMENOLOGY_798: (12)                        arcs.append(arc)
SNGT_QHENOMENOLOGY_799: (8)                     self.clear_points()
SNGT_QHENOMENOLOGY_800: (8)                     arcs = [arcs[-1], *arcs[:-1]]
SNGT_QHENOMENOLOGY_801: (8)                     for arc1, arc2 in adjacent_pairs(arcs):
SNGT_QHENOMENOLOGY_802: (12)                        self.add_subpath(arc1.get_points())
SNGT_QHENOMENOLOGY_803: (12)                        self.add_line_to(arc2.get_start())
SNGT_QHENOMENOLOGY_804: (8)                     return self
SNGT_QHENOMENOLOGY_805: (0)             class Polyline(VMobject):
SNGT_QHENOMENOLOGY_806: (4)                 def __init__(
SNGT_QHENOMENOLOGY_807: (8)                     self,
SNGT_QHENOMENOLOGY_808: (8)                     *vertices: Vect3,
SNGT_QHENOMENOLOGY_809: (8)                     **kwargs
SNGT_QHENOMENOLOGY_810: (4)                 ):
SNGT_QHENOMENOLOGY_811: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_812: (8)                     self.set_points_as_corners(vertices)
SNGT_QHENOMENOLOGY_813: (0)             class RegularPolygon(Polygon):
SNGT_QHENOMENOLOGY_814: (4)                 def __init__(
SNGT_QHENOMENOLOGY_815: (8)                     self,
SNGT_QHENOMENOLOGY_816: (8)                     n: int = 6,
SNGT_QHENOMENOLOGY_817: (8)                     radius: float = 1.0,
SNGT_QHENOMENOLOGY_818: (8)                     start_angle: float | None = None,
SNGT_QHENOMENOLOGY_819: (8)                     **kwargs
SNGT_QHENOMENOLOGY_820: (4)                 ):
SNGT_QHENOMENOLOGY_821: (8)                     if start_angle is None:
SNGT_QHENOMENOLOGY_822: (12)                        start_angle = (n % 2) * 90 * DEG
SNGT_QHENOMENOLOGY_823: (8)                     start_vect = rotate_vector(radius * RIGHT,
start_angle)
SNGT_QHENOMENOLOGY_824: (8)                     vertices = compass_directions(n, start_vect)
SNGT_QHENOMENOLOGY_825: (8)                     super().__init__(*vertices, **kwargs)
SNGT_QHENOMENOLOGY_826: (0)             class Triangle(RegularPolygon):
SNGT_QHENOMENOLOGY_827: (4)                 def __init__(self, **kwargs):
SNGT_QHENOMENOLOGY_828: (8)                     super().__init__(n=3, **kwargs)
SNGT_QHENOMENOLOGY_829: (0)             class ArrowTip(Triangle):
SNGT_QHENOMENOLOGY_830: (4)                 def __init__(
SNGT_QHENOMENOLOGY_831: (8)                     self,
SNGT_QHENOMENOLOGY_832: (8)                     angle: float = 0,
```

```
SNGT_QHENOMENOLOGY_833: (8)                     width: float = DEFAULT_ARROW_TIP_WIDTH,
SNGT_QHENOMENOLOGY_834: (8)                     length: float = DEFAULT_ARROW_TIP_LENGTH,
SNGT_QHENOMENOLOGY_835: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_836: (8)                     fill_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_837: (8)                     stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_838: (8)                     tip_style: int = 0,  # triangle=0, inner_smooth=1,
dot=2
SNGT_QHENOMENOLOGY_839: (8)                     **kwargs
SNGT_QHENOMENOLOGY_840: (4)                 ):
SNGT_QHENOMENOLOGY_841: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_842: (12)                        start_angle=0,
SNGT_QHENOMENOLOGY_843: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_844: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_845: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_846: (12)                        **kwargs
SNGT_QHENOMENOLOGY_847: (8)                     )
SNGT_QHENOMENOLOGY_848: (8)                     self.set_height(width)
SNGT_QHENOMENOLOGY_849: (8)                     self.set_width(length, stretch=True)
SNGT_QHENOMENOLOGY_850: (8)                     if tip_style == 1:
SNGT_QHENOMENOLOGY_851: (12)                        self.set_height(length * 0.9, stretch=True)
SNGT_QHENOMENOLOGY_852: (12)                        self.data["point"][4] += np.array([0.6 *
length, 0, 0])
SNGT_QHENOMENOLOGY_853: (8)                     elif tip_style == 2:
SNGT_QHENOMENOLOGY_854: (12)                        h = length / 2
SNGT_QHENOMENOLOGY_855: (12)
self.set_points(Dot().set_width(h).get_points())
SNGT_QHENOMENOLOGY_856: (8)                     self.rotate(angle)
SNGT_QHENOMENOLOGY_857: (4)                 def get_base(self) -> Vect3:
SNGT_QHENOMENOLOGY_858: (8)                     return self.point_from_proportion(0.5)
SNGT_QHENOMENOLOGY_859: (4)                 def get_tip_point(self) -> Vect3:
SNGT_QHENOMENOLOGY_860: (8)                     return self.get_points()[0]
SNGT_QHENOMENOLOGY_861: (4)                 def get_vector(self) -> Vect3:
SNGT_QHENOMENOLOGY_862: (8)                     return self.get_tip_point() - self.get_base()
SNGT_QHENOMENOLOGY_863: (4)                 def get_angle(self) -> float:
SNGT_QHENOMENOLOGY_864: (8)                     return angle_of_vector(self.get_vector())
SNGT_QHENOMENOLOGY_865: (4)                 def get_length(self) -> float:
SNGT_QHENOMENOLOGY_866: (8)                     return get_norm(self.get_vector())
SNGT_QHENOMENOLOGY_867: (0)             class Rectangle(Polygon):
SNGT_QHENOMENOLOGY_868: (4)                 def __init__(
SNGT_QHENOMENOLOGY_869: (8)                     self,
SNGT_QHENOMENOLOGY_870: (8)                     width: float = 4.0,
SNGT_QHENOMENOLOGY_871: (8)                     height: float = 2.0,
SNGT_QHENOMENOLOGY_872: (8)                     **kwargs
SNGT_QHENOMENOLOGY_873: (4)                 ):
SNGT_QHENOMENOLOGY_874: (8)                     super().__init__(UR, UL, DL, DR, **kwargs)
SNGT_QHENOMENOLOGY_875: (8)                     self.set_width(width, stretch=True)
SNGT_QHENOMENOLOGY_876: (8)                     self.set_height(height, stretch=True)
SNGT_QHENOMENOLOGY_877: (4)                 def surround(self, mobject, buff=SMALL_BUFF) -> Self:
SNGT_QHENOMENOLOGY_878: (8)                     target_shape = np.array(mobject.get_shape()) + 2 *
buff
SNGT_QHENOMENOLOGY_879: (8)                     self.set_shape(*target_shape)
SNGT_QHENOMENOLOGY_880: (8)                     self.move_to(mobject)
SNGT_QHENOMENOLOGY_881: (8)                     return self
SNGT_QHENOMENOLOGY_882: (0)             class Square(Rectangle):
SNGT_QHENOMENOLOGY_883: (4)                 def __init__(self, side_length: float = 2.0, **kwargs):
SNGT_QHENOMENOLOGY_884: (8)                     super().__init__(side_length, side_length,
**kwargs)
SNGT_QHENOMENOLOGY_885: (0)             class RoundedRectangle(Rectangle):
SNGT_QHENOMENOLOGY_886: (4)                 def __init__(
SNGT_QHENOMENOLOGY_887: (8)                     self,
SNGT_QHENOMENOLOGY_888: (8)                     width: float = 4.0,
SNGT_QHENOMENOLOGY_889: (8)                     height: float = 2.0,
SNGT_QHENOMENOLOGY_890: (8)                     corner_radius: float = 0.5,
SNGT_QHENOMENOLOGY_891: (8)                     **kwargs
SNGT_QHENOMENOLOGY_892: (4)                 ):
SNGT_QHENOMENOLOGY_893: (8)                     super().__init__(width, height, **kwargs)
SNGT_QHENOMENOLOGY_894: (8)                     self.round_corners(corner_radius)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 30 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 31 - constants.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_4: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_5: (4)                  from typing import List
SNGT_QHENOMENOLOGY_6: (4)                  from manimlib.typing import ManimColor, Vect3
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_8: (0)              DEFAULT_RESOLUTION: tuple[int, int] =
manim_config.camera.resolution
SNGT_QHENOMENOLOGY_9: (0)              DEFAULT_PIXEL_WIDTH: int = DEFAULT_RESOLUTION[0]
SNGT_QHENOMENOLOGY_10: (0)             DEFAULT_PIXEL_HEIGHT: int = DEFAULT_RESOLUTION[1]
SNGT_QHENOMENOLOGY_11: (0)             ASPECT_RATIO: float = DEFAULT_PIXEL_WIDTH /
DEFAULT_PIXEL_HEIGHT
SNGT_QHENOMENOLOGY_12: (0)             FRAME_HEIGHT: float = manim_config.sizes.frame_height
SNGT_QHENOMENOLOGY_13: (0)             FRAME_WIDTH: float = FRAME_HEIGHT * ASPECT_RATIO
SNGT_QHENOMENOLOGY_14: (0)             FRAME_SHAPE: tuple[float, float] = (FRAME_WIDTH,
FRAME_HEIGHT)
SNGT_QHENOMENOLOGY_15: (0)             FRAME_Y_RADIUS: float = FRAME_HEIGHT / 2
SNGT_QHENOMENOLOGY_16: (0)             FRAME_X_RADIUS: float = FRAME_WIDTH / 2
SNGT_QHENOMENOLOGY_17: (0)             SMALL_BUFF: float = manim_config.sizes.small_buff
SNGT_QHENOMENOLOGY_18: (0)             MED_SMALL_BUFF: float = manim_config.sizes.med_small_buff
SNGT_QHENOMENOLOGY_19: (0)             MED_LARGE_BUFF: float = manim_config.sizes.med_large_buff
SNGT_QHENOMENOLOGY_20: (0)             LARGE_BUFF: float = manim_config.sizes.large_buff
SNGT_QHENOMENOLOGY_21: (0)             DEFAULT_MOBJECT_TO_EDGE_BUFF: float =
manim_config.sizes.default_mobject_to_edge_buff
SNGT_QHENOMENOLOGY_22: (0)             DEFAULT_MOBJECT_TO_MOBJECT_BUFF: float =
manim_config.sizes.default_mobject_to_mobject_buff
SNGT_QHENOMENOLOGY_23: (0)             ORIGIN: Vect3 = np.array([0., 0., 0.])
SNGT_QHENOMENOLOGY_24: (0)             UP: Vect3 = np.array([0., 1., 0.])
SNGT_QHENOMENOLOGY_25: (0)             DOWN: Vect3 = np.array([0., -1., 0.])
SNGT_QHENOMENOLOGY_26: (0)             RIGHT: Vect3 = np.array([1., 0., 0.])
SNGT_QHENOMENOLOGY_27: (0)             LEFT: Vect3 = np.array([-1., 0., 0.])
SNGT_QHENOMENOLOGY_28: (0)             IN: Vect3 = np.array([0., 0., -1.])
SNGT_QHENOMENOLOGY_29: (0)             OUT: Vect3 = np.array([0., 0., 1.])
SNGT_QHENOMENOLOGY_30: (0)             X_AXIS: Vect3 = np.array([1., 0., 0.])
SNGT_QHENOMENOLOGY_31: (0)             Y_AXIS: Vect3 = np.array([0., 1., 0.])
SNGT_QHENOMENOLOGY_32: (0)             Z_AXIS: Vect3 = np.array([0., 0., 1.])
SNGT_QHENOMENOLOGY_33: (0)             NULL_POINTS = np.array([[0., 0., 0.]])
SNGT_QHENOMENOLOGY_34: (0)             UL: Vect3 = UP + LEFT
SNGT_QHENOMENOLOGY_35: (0)             UR: Vect3 = UP + RIGHT
SNGT_QHENOMENOLOGY_36: (0)             DL: Vect3 = DOWN + LEFT
SNGT_QHENOMENOLOGY_37: (0)             DR: Vect3 = DOWN + RIGHT
SNGT_QHENOMENOLOGY_38: (0)             TOP: Vect3 = FRAME_Y_RADIUS * UP
SNGT_QHENOMENOLOGY_39: (0)             BOTTOM: Vect3 = FRAME_Y_RADIUS * DOWN
SNGT_QHENOMENOLOGY_40: (0)             LEFT_SIDE: Vect3 = FRAME_X_RADIUS * LEFT
SNGT_QHENOMENOLOGY_41: (0)             RIGHT_SIDE: Vect3 = FRAME_X_RADIUS * RIGHT
SNGT_QHENOMENOLOGY_42: (0)             PI: float = np.pi
SNGT_QHENOMENOLOGY_43: (0)             TAU: float = 2 * PI
SNGT_QHENOMENOLOGY_44: (0)             DEG: float = TAU / 360
SNGT_QHENOMENOLOGY_45: (0)             DEGREES = DEG  # Many older animations use the full name
SNGT_QHENOMENOLOGY_46: (0)             RADIANS: float = 1
SNGT_QHENOMENOLOGY_47: (0)             NORMAL: str = "NORMAL"
SNGT_QHENOMENOLOGY_48: (0)             ITALIC: str = "ITALIC"
SNGT_QHENOMENOLOGY_49: (0)             OBLIQUE: str = "OBLIQUE"
SNGT_QHENOMENOLOGY_50: (0)             BOLD: str = "BOLD"
SNGT_QHENOMENOLOGY_51: (0)             DEFAULT_STROKE_WIDTH: float =
manim_config.vmobject.default_stroke_width
SNGT_QHENOMENOLOGY_52: (0)             BLUE_E: ManimColor = manim_config.colors.blue_e
SNGT_QHENOMENOLOGY_53: (0)             BLUE_D: ManimColor = manim_config.colors.blue_d
SNGT_QHENOMENOLOGY_54: (0)             BLUE_C: ManimColor = manim_config.colors.blue_c
```

```
SNGT_QHENOMENOLOGY_55: (0)              BLUE_B: ManimColor = manim_config.colors.blue_b
SNGT_QHENOMENOLOGY_56: (0)              BLUE_A: ManimColor = manim_config.colors.blue_a
SNGT_QHENOMENOLOGY_57: (0)              TEAL_E: ManimColor = manim_config.colors.teal_e
SNGT_QHENOMENOLOGY_58: (0)              TEAL_D: ManimColor = manim_config.colors.teal_d
SNGT_QHENOMENOLOGY_59: (0)              TEAL_C: ManimColor = manim_config.colors.teal_c
SNGT_QHENOMENOLOGY_60: (0)              TEAL_B: ManimColor = manim_config.colors.teal_b
SNGT_QHENOMENOLOGY_61: (0)              TEAL_A: ManimColor = manim_config.colors.teal_a
SNGT_QHENOMENOLOGY_62: (0)              GREEN_E: ManimColor = manim_config.colors.green_e
SNGT_QHENOMENOLOGY_63: (0)              GREEN_D: ManimColor = manim_config.colors.green_d
SNGT_QHENOMENOLOGY_64: (0)              GREEN_C: ManimColor = manim_config.colors.green_c
SNGT_QHENOMENOLOGY_65: (0)              GREEN_B: ManimColor = manim_config.colors.green_b
SNGT_QHENOMENOLOGY_66: (0)              GREEN_A: ManimColor = manim_config.colors.green_a
SNGT_QHENOMENOLOGY_67: (0)              YELLOW_E: ManimColor = manim_config.colors.yellow_e
SNGT_QHENOMENOLOGY_68: (0)              YELLOW_D: ManimColor = manim_config.colors.yellow_d
SNGT_QHENOMENOLOGY_69: (0)              YELLOW_C: ManimColor = manim_config.colors.yellow_c
SNGT_QHENOMENOLOGY_70: (0)              YELLOW_B: ManimColor = manim_config.colors.yellow_b
SNGT_QHENOMENOLOGY_71: (0)              YELLOW_A: ManimColor = manim_config.colors.yellow_a
SNGT_QHENOMENOLOGY_72: (0)              GOLD_E: ManimColor = manim_config.colors.gold_e
SNGT_QHENOMENOLOGY_73: (0)              GOLD_D: ManimColor = manim_config.colors.gold_d
SNGT_QHENOMENOLOGY_74: (0)              GOLD_C: ManimColor = manim_config.colors.gold_c
SNGT_QHENOMENOLOGY_75: (0)              GOLD_B: ManimColor = manim_config.colors.gold_b
SNGT_QHENOMENOLOGY_76: (0)              GOLD_A: ManimColor = manim_config.colors.gold_a
SNGT_QHENOMENOLOGY_77: (0)              RED_E: ManimColor = manim_config.colors.red_e
SNGT_QHENOMENOLOGY_78: (0)              RED_D: ManimColor = manim_config.colors.red_d
SNGT_QHENOMENOLOGY_79: (0)              RED_C: ManimColor = manim_config.colors.red_c
SNGT_QHENOMENOLOGY_80: (0)              RED_B: ManimColor = manim_config.colors.red_b
SNGT_QHENOMENOLOGY_81: (0)              RED_A: ManimColor = manim_config.colors.red_a
SNGT_QHENOMENOLOGY_82: (0)              MAROON_E: ManimColor = manim_config.colors.maroon_e
SNGT_QHENOMENOLOGY_83: (0)              MAROON_D: ManimColor = manim_config.colors.maroon_d
SNGT_QHENOMENOLOGY_84: (0)              MAROON_C: ManimColor = manim_config.colors.maroon_c
SNGT_QHENOMENOLOGY_85: (0)              MAROON_B: ManimColor = manim_config.colors.maroon_b
SNGT_QHENOMENOLOGY_86: (0)              MAROON_A: ManimColor = manim_config.colors.maroon_a
SNGT_QHENOMENOLOGY_87: (0)              PURPLE_E: ManimColor = manim_config.colors.purple_e
SNGT_QHENOMENOLOGY_88: (0)              PURPLE_D: ManimColor = manim_config.colors.purple_d
SNGT_QHENOMENOLOGY_89: (0)              PURPLE_C: ManimColor = manim_config.colors.purple_c
SNGT_QHENOMENOLOGY_90: (0)              PURPLE_B: ManimColor = manim_config.colors.purple_b
SNGT_QHENOMENOLOGY_91: (0)              PURPLE_A: ManimColor = manim_config.colors.purple_a
SNGT_QHENOMENOLOGY_92: (0)              GREY_E: ManimColor = manim_config.colors.grey_e
SNGT_QHENOMENOLOGY_93: (0)              GREY_D: ManimColor = manim_config.colors.grey_d
SNGT_QHENOMENOLOGY_94: (0)              GREY_C: ManimColor = manim_config.colors.grey_c
SNGT_QHENOMENOLOGY_95: (0)              GREY_B: ManimColor = manim_config.colors.grey_b
SNGT_QHENOMENOLOGY_96: (0)              GREY_A: ManimColor = manim_config.colors.grey_a
SNGT_QHENOMENOLOGY_97: (0)              WHITE: ManimColor = manim_config.colors.white
SNGT_QHENOMENOLOGY_98: (0)              BLACK: ManimColor = manim_config.colors.black
SNGT_QHENOMENOLOGY_99: (0)              GREY_BROWN: ManimColor = manim_config.colors.grey_brown
SNGT_QHENOMENOLOGY_100: (0)             DARK_BROWN: ManimColor = manim_config.colors.dark_brown
SNGT_QHENOMENOLOGY_101: (0)             LIGHT_BROWN: ManimColor = manim_config.colors.light_brown
SNGT_QHENOMENOLOGY_102: (0)             PINK: ManimColor = manim_config.colors.pink
SNGT_QHENOMENOLOGY_103: (0)             LIGHT_PINK: ManimColor = manim_config.colors.light_pink
SNGT_QHENOMENOLOGY_104: (0)             GREEN_SCREEN: ManimColor = manim_config.colors.green_screen
SNGT_QHENOMENOLOGY_105: (0)             ORANGE: ManimColor = manim_config.colors.orange
SNGT_QHENOMENOLOGY_106: (0)             MANIM_COLORS: List[ManimColor] =
list(manim_config.colors.values())
SNGT_QHENOMENOLOGY_107: (0)             BLUE: ManimColor = BLUE_C
SNGT_QHENOMENOLOGY_108: (0)             TEAL: ManimColor = TEAL_C
SNGT_QHENOMENOLOGY_109: (0)             GREEN: ManimColor = GREEN_C
SNGT_QHENOMENOLOGY_110: (0)             YELLOW: ManimColor = YELLOW_C
SNGT_QHENOMENOLOGY_111: (0)             GOLD: ManimColor = GOLD_C
SNGT_QHENOMENOLOGY_112: (0)             RED: ManimColor = RED_C
SNGT_QHENOMENOLOGY_113: (0)             MAROON: ManimColor = MAROON_C
SNGT_QHENOMENOLOGY_114: (0)             PURPLE: ManimColor = PURPLE_C
SNGT_QHENOMENOLOGY_115: (0)             GREY: ManimColor = GREY_C
SNGT_QHENOMENOLOGY_116: (0)             COLORMAP_3B1B: List[ManimColor] = [BLUE_E, GREEN, YELLOW,
RED]
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 32 - transform.py:
SNGT_QHENOMENOLOGY_
```

```
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import inspect
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import OUT
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.paths import path_along_arc
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.paths import straight_path
SNGT_QHENOMENOLOGY_11: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_12: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_13: (4)                  from typing import Callable
SNGT_QHENOMENOLOGY_14: (4)                  import numpy.typing as npt
SNGT_QHENOMENOLOGY_15: (4)                  from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_16: (4)                  from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_17: (0)              class Transform(Animation):
SNGT_QHENOMENOLOGY_18: (4)                  replace_mobject_with_target_in_scene: bool = False
SNGT_QHENOMENOLOGY_19: (4)                  def __init__(
SNGT_QHENOMENOLOGY_20: (8)                      self,
SNGT_QHENOMENOLOGY_21: (8)                      mobject: Mobject,
SNGT_QHENOMENOLOGY_22: (8)                      target_mobject: Mobject | None = None,
SNGT_QHENOMENOLOGY_23: (8)                      path_arc: float = 0.0,
SNGT_QHENOMENOLOGY_24: (8)                      path_arc_axis: np.ndarray = OUT,
SNGT_QHENOMENOLOGY_25: (8)                      path_func: Callable | None = None,
SNGT_QHENOMENOLOGY_26: (8)                      **kwargs
SNGT_QHENOMENOLOGY_27: (4)                  ):
SNGT_QHENOMENOLOGY_28: (8)                      self.target_mobject = target_mobject
SNGT_QHENOMENOLOGY_29: (8)                      self.path_arc = path_arc
SNGT_QHENOMENOLOGY_30: (8)                      self.path_arc_axis = path_arc_axis
SNGT_QHENOMENOLOGY_31: (8)                      self.path_func = path_func
SNGT_QHENOMENOLOGY_32: (8)                      super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_33: (8)                      self.init_path_func()
SNGT_QHENOMENOLOGY_34: (4)                  def init_path_func(self) -> None:
SNGT_QHENOMENOLOGY_35: (8)                      if self.path_func is not None:
SNGT_QHENOMENOLOGY_36: (12)                         return
SNGT_QHENOMENOLOGY_37: (8)                      elif self.path_arc == 0:
SNGT_QHENOMENOLOGY_38: (12)                         self.path_func = straight_path
SNGT_QHENOMENOLOGY_39: (8)                      else:
SNGT_QHENOMENOLOGY_40: (12)                         self.path_func = path_along_arc(
SNGT_QHENOMENOLOGY_41: (16)                             self.path_arc,
SNGT_QHENOMENOLOGY_42: (16)                             self.path_arc_axis,
SNGT_QHENOMENOLOGY_43: (12)                         )
SNGT_QHENOMENOLOGY_44: (4)                  def begin(self) -> None:
SNGT_QHENOMENOLOGY_45: (8)                      self.target_mobject = self.create_target()
SNGT_QHENOMENOLOGY_46: (8)                      self.check_target_mobject_validity()
SNGT_QHENOMENOLOGY_47: (8)                      if
self.mobject.is_aligned_with(self.target_mobject):
SNGT_QHENOMENOLOGY_48: (12)                         self.target_copy = self.target_mobject
SNGT_QHENOMENOLOGY_49: (8)                      else:
SNGT_QHENOMENOLOGY_50: (12)                         self.target_copy = self.target_mobject.copy()
SNGT_QHENOMENOLOGY_51: (8)
self.mobject.align_data_and_family(self.target_copy)
SNGT_QHENOMENOLOGY_52: (8)                      super().begin()
SNGT_QHENOMENOLOGY_53: (8)                      if not self.mobject.has_updaters():
SNGT_QHENOMENOLOGY_54: (12)                         self.mobject.lock_matching_data(
SNGT_QHENOMENOLOGY_55: (16)                             self.starting_mobject,
SNGT_QHENOMENOLOGY_56: (16)                             self.target_copy,
SNGT_QHENOMENOLOGY_57: (12)                         )
SNGT_QHENOMENOLOGY_58: (4)                  def finish(self) -> None:
SNGT_QHENOMENOLOGY_59: (8)                      super().finish()
SNGT_QHENOMENOLOGY_60: (8)                      self.mobject.unlock_data()
SNGT_QHENOMENOLOGY_61: (4)                  def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_62: (8)                      return self.target_mobject
SNGT_QHENOMENOLOGY_63: (4)                  def check_target_mobject_validity(self) -> None:
SNGT_QHENOMENOLOGY_64: (8)                      if self.target_mobject is None:
SNGT_QHENOMENOLOGY_65: (12)                         raise Exception(
SNGT_QHENOMENOLOGY_66: (16)                             f"{self.__class__.__name__}.create_target
not properly implemented"
```

```
SNGT_QHENOMENOLOGY_67: (12)                         )
SNGT_QHENOMENOLOGY_68: (4)                 def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_69: (8)                     super().clean_up_from_scene(scene)
SNGT_QHENOMENOLOGY_70: (8)                     if self.replace_mobject_with_target_in_scene:
SNGT_QHENOMENOLOGY_71: (12)                        scene.remove(self.mobject)
SNGT_QHENOMENOLOGY_72: (12)                        scene.add(self.target_mobject)
SNGT_QHENOMENOLOGY_73: (4)                 def update_config(self, **kwargs) -> None:
SNGT_QHENOMENOLOGY_74: (8)                     Animation.update_config(self, **kwargs)
SNGT_QHENOMENOLOGY_75: (8)                     if "path_arc" in kwargs:
SNGT_QHENOMENOLOGY_76: (12)                        self.path_func = path_along_arc(
SNGT_QHENOMENOLOGY_77: (16)                            kwargs["path_arc"],
SNGT_QHENOMENOLOGY_78: (16)                            kwargs.get("path_arc_axis", OUT)
SNGT_QHENOMENOLOGY_79: (12)                        )
SNGT_QHENOMENOLOGY_80: (4)                 def get_all_mobjects(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_81: (8)                     return [
SNGT_QHENOMENOLOGY_82: (12)                        self.mobject,
SNGT_QHENOMENOLOGY_83: (12)                        self.starting_mobject,
SNGT_QHENOMENOLOGY_84: (12)                        self.target_mobject,
SNGT_QHENOMENOLOGY_85: (12)                        self.target_copy,
SNGT_QHENOMENOLOGY_86: (8)                     ]
SNGT_QHENOMENOLOGY_87: (4)                 def get_all_families_zipped(self) ->
zip[tuple[Mobject]]:
SNGT_QHENOMENOLOGY_88: (8)                     return zip(*[
SNGT_QHENOMENOLOGY_89: (12)                        mob.get_family()
SNGT_QHENOMENOLOGY_90: (12)                        for mob in [
SNGT_QHENOMENOLOGY_91: (16)                            self.mobject,
SNGT_QHENOMENOLOGY_92: (16)                            self.starting_mobject,
SNGT_QHENOMENOLOGY_93: (16)                            self.target_copy,
SNGT_QHENOMENOLOGY_94: (12)                        ]
SNGT_QHENOMENOLOGY_95: (8)                     ])
SNGT_QHENOMENOLOGY_96: (4)                 def interpolate_submobject(
SNGT_QHENOMENOLOGY_97: (8)                     self,
SNGT_QHENOMENOLOGY_98: (8)                     submob: Mobject,
SNGT_QHENOMENOLOGY_99: (8)                     start: Mobject,
SNGT_QHENOMENOLOGY_100: (8)                    target_copy: Mobject,
SNGT_QHENOMENOLOGY_101: (8)                    alpha: float
SNGT_QHENOMENOLOGY_102: (4)                ):
SNGT_QHENOMENOLOGY_103: (8)                    submob.interpolate(start, target_copy, alpha,
self.path_func)
SNGT_QHENOMENOLOGY_104: (8)                    return self
SNGT_QHENOMENOLOGY_105: (0)            class ReplacementTransform(Transform):
SNGT_QHENOMENOLOGY_106: (4)                replace_mobject_with_target_in_scene: bool = True
SNGT_QHENOMENOLOGY_107: (0)            class TransformFromCopy(Transform):
SNGT_QHENOMENOLOGY_108: (4)                replace_mobject_with_target_in_scene: bool = True
SNGT_QHENOMENOLOGY_109: (4)                def __init__(self, mobject: Mobject, target_mobject:
Mobject, **kwargs):
SNGT_QHENOMENOLOGY_110: (8)                    super().__init__(mobject.copy(), target_mobject,
**kwargs)
SNGT_QHENOMENOLOGY_111: (0)            class MoveToTarget(Transform):
SNGT_QHENOMENOLOGY_112: (4)                def __init__(self, mobject: Mobject, **kwargs):
SNGT_QHENOMENOLOGY_113: (8)                    self.check_validity_of_input(mobject)
SNGT_QHENOMENOLOGY_114: (8)                    super().__init__(mobject, mobject.target, **kwargs)
SNGT_QHENOMENOLOGY_115: (4)                def check_validity_of_input(self, mobject: Mobject) ->
None:
SNGT_QHENOMENOLOGY_116: (8)                    if not hasattr(mobject, "target"):
SNGT_QHENOMENOLOGY_117: (12)                       raise Exception(
SNGT_QHENOMENOLOGY_118: (16)                           "MoveToTarget called on mobject without
attribute 'target'"
SNGT_QHENOMENOLOGY_119: (12)                       )
SNGT_QHENOMENOLOGY_120: (0)            class _MethodAnimation(MoveToTarget):
SNGT_QHENOMENOLOGY_121: (4)                def __init__(self, mobject: Mobject, methods:
list[Callable], **kwargs):
SNGT_QHENOMENOLOGY_122: (8)                    self.methods = methods
SNGT_QHENOMENOLOGY_123: (8)                    super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_124: (0)            class ApplyMethod(Transform):
SNGT_QHENOMENOLOGY_125: (4)                def __init__(self, method: Callable, *args, **kwargs):
SNGT_QHENOMENOLOGY_126: (8)                    """
SNGT_QHENOMENOLOGY_127: (8)                    method is a method of Mobject, *args are arguments
for
```

```
SNGT_QHENOMENOLOGY_128: (8)                     that method.  Key word arguments should be passed
in
SNGT_QHENOMENOLOGY_129: (8)                     as the last arg, as a dict, since **kwargs is for
SNGT_QHENOMENOLOGY_130: (8)                     configuration of the transform itself
SNGT_QHENOMENOLOGY_131: (8)                     Relies on the fact that mobject methods return the
mobject
SNGT_QHENOMENOLOGY_132: (8)                     """
SNGT_QHENOMENOLOGY_133: (8)                     self.check_validity_of_input(method)
SNGT_QHENOMENOLOGY_134: (8)                     self.method = method
SNGT_QHENOMENOLOGY_135: (8)                     self.method_args = args
SNGT_QHENOMENOLOGY_136: (8)                     super().__init__(method.__self__, **kwargs)
SNGT_QHENOMENOLOGY_137: (4)                 def check_validity_of_input(self, method: Callable) ->
None:
SNGT_QHENOMENOLOGY_138: (8)                     if not inspect.ismethod(method):
SNGT_QHENOMENOLOGY_139: (12)                        raise Exception(
SNGT_QHENOMENOLOGY_140: (16)                            "Whoops, looks like you accidentally
invoked "
SNGT_QHENOMENOLOGY_141: (16)                            "the method you want to animate"
SNGT_QHENOMENOLOGY_142: (12)                        )
SNGT_QHENOMENOLOGY_143: (8)                     assert isinstance(method.__self__, Mobject)
SNGT_QHENOMENOLOGY_144: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_145: (8)                     method = self.method
SNGT_QHENOMENOLOGY_146: (8)                     args = list(self.method_args)
SNGT_QHENOMENOLOGY_147: (8)                     if len(args) > 0 and isinstance(args[-1], dict):
SNGT_QHENOMENOLOGY_148: (12)                        method_kwargs = args.pop()
SNGT_QHENOMENOLOGY_149: (8)                     else:
SNGT_QHENOMENOLOGY_150: (12)                        method_kwargs = {}
SNGT_QHENOMENOLOGY_151: (8)                     target = method.__self__.copy()
SNGT_QHENOMENOLOGY_152: (8)                     method.__func__(target, *args, **method_kwargs)
SNGT_QHENOMENOLOGY_153: (8)                     return target
SNGT_QHENOMENOLOGY_154: (0)             class ApplyPointwiseFunction(ApplyMethod):
SNGT_QHENOMENOLOGY_155: (4)                 def __init__(
SNGT_QHENOMENOLOGY_156: (8)                     self,
SNGT_QHENOMENOLOGY_157: (8)                     function: Callable[[np.ndarray], np.ndarray],
SNGT_QHENOMENOLOGY_158: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_159: (8)                     run_time: float = 3.0,
SNGT_QHENOMENOLOGY_160: (8)                     **kwargs
SNGT_QHENOMENOLOGY_161: (4)                 ):
SNGT_QHENOMENOLOGY_162: (8)                     super().__init__(mobject.apply_function, function,
run_time=run_time, **kwargs)
SNGT_QHENOMENOLOGY_163: (0)             class ApplyPointwiseFunctionToCenter(Transform):
SNGT_QHENOMENOLOGY_164: (4)                 def __init__(
SNGT_QHENOMENOLOGY_165: (8)                     self,
SNGT_QHENOMENOLOGY_166: (8)                     function: Callable[[np.ndarray], np.ndarray],
SNGT_QHENOMENOLOGY_167: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_168: (8)                     **kwargs
SNGT_QHENOMENOLOGY_169: (4)                 ):
SNGT_QHENOMENOLOGY_170: (8)                     self.function = function
SNGT_QHENOMENOLOGY_171: (8)                     super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_172: (4)                 def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_173: (8)                     return
self.mobject.copy().move_to(self.function(self.mobject.get_center()))
SNGT_QHENOMENOLOGY_174: (0)             class FadeToColor(ApplyMethod):
SNGT_QHENOMENOLOGY_175: (4)                 def __init__(
SNGT_QHENOMENOLOGY_176: (8)                     self,
SNGT_QHENOMENOLOGY_177: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_178: (8)                     color: ManimColor,
SNGT_QHENOMENOLOGY_179: (8)                     **kwargs
SNGT_QHENOMENOLOGY_180: (4)                 ):
SNGT_QHENOMENOLOGY_181: (8)                     super().__init__(mobject.set_color, color,
**kwargs)
SNGT_QHENOMENOLOGY_182: (0)             class ScaleInPlace(ApplyMethod):
SNGT_QHENOMENOLOGY_183: (4)                 def __init__(
SNGT_QHENOMENOLOGY_184: (8)                     self,
SNGT_QHENOMENOLOGY_185: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_186: (8)                     scale_factor: npt.ArrayLike,
SNGT_QHENOMENOLOGY_187: (8)                     **kwargs
SNGT_QHENOMENOLOGY_188: (4)                 ):
SNGT_QHENOMENOLOGY_189: (8)                     super().__init__(mobject.scale, scale_factor,
```

```
**kwargs)
SNGT_QHENOMENOLOGY_190: (0)            class ShrinkToCenter(ScaleInPlace):
SNGT_QHENOMENOLOGY_191: (4)                def __init__(self, mobject: Mobject, **kwargs):
SNGT_QHENOMENOLOGY_192: (8)                    super().__init__(mobject, 0, **kwargs)
SNGT_QHENOMENOLOGY_193: (0)            class Restore(Transform):
SNGT_QHENOMENOLOGY_194: (4)                def __init__(self, mobject: Mobject, **kwargs):
SNGT_QHENOMENOLOGY_195: (8)                    if not hasattr(mobject, "saved_state") or
mobject.saved_state is None:
SNGT_QHENOMENOLOGY_196: (12)                       raise Exception("Trying to restore without
having saved")
SNGT_QHENOMENOLOGY_197: (8)                    super().__init__(mobject, mobject.saved_state,
**kwargs)
SNGT_QHENOMENOLOGY_198: (0)            class ApplyFunction(Transform):
SNGT_QHENOMENOLOGY_199: (4)                def __init__(
SNGT_QHENOMENOLOGY_200: (8)                    self,
SNGT_QHENOMENOLOGY_201: (8)                    function: Callable[[Mobject], Mobject],
SNGT_QHENOMENOLOGY_202: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_203: (8)                    **kwargs
SNGT_QHENOMENOLOGY_204: (4)                ):
SNGT_QHENOMENOLOGY_205: (8)                    self.function = function
SNGT_QHENOMENOLOGY_206: (8)                    super().__init__(mobject, **kwargs)
SNGT_QHENOMENOLOGY_207: (4)                def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_208: (8)                    target = self.function(self.mobject.copy())
SNGT_QHENOMENOLOGY_209: (8)                    if not isinstance(target, Mobject):
SNGT_QHENOMENOLOGY_210: (12)                       raise Exception("Functions passed to
ApplyFunction must return object of type Mobject")
SNGT_QHENOMENOLOGY_211: (8)                    return target
SNGT_QHENOMENOLOGY_212: (0)            class ApplyMatrix(ApplyPointwiseFunction):
SNGT_QHENOMENOLOGY_213: (4)                def __init__(
SNGT_QHENOMENOLOGY_214: (8)                    self,
SNGT_QHENOMENOLOGY_215: (8)                    matrix: npt.ArrayLike,
SNGT_QHENOMENOLOGY_216: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_217: (8)                    **kwargs
SNGT_QHENOMENOLOGY_218: (4)                ):
SNGT_QHENOMENOLOGY_219: (8)                    matrix = self.initialize_matrix(matrix)
SNGT_QHENOMENOLOGY_220: (8)                    def func(p):
SNGT_QHENOMENOLOGY_221: (12)                       return np.dot(p, matrix.T)
SNGT_QHENOMENOLOGY_222: (8)                    super().__init__(func, mobject, **kwargs)
SNGT_QHENOMENOLOGY_223: (4)                def initialize_matrix(self, matrix: npt.ArrayLike) ->
np.ndarray:
SNGT_QHENOMENOLOGY_224: (8)                    matrix = np.array(matrix)
SNGT_QHENOMENOLOGY_225: (8)                    if matrix.shape == (2, 2):
SNGT_QHENOMENOLOGY_226: (12)                       new_matrix = np.identity(3)
SNGT_QHENOMENOLOGY_227: (12)                       new_matrix[:2, :2] = matrix
SNGT_QHENOMENOLOGY_228: (12)                       matrix = new_matrix
SNGT_QHENOMENOLOGY_229: (8)                    elif matrix.shape != (3, 3):
SNGT_QHENOMENOLOGY_230: (12)                       raise Exception("Matrix has bad dimensions")
SNGT_QHENOMENOLOGY_231: (8)                    return matrix
SNGT_QHENOMENOLOGY_232: (0)            class ApplyComplexFunction(ApplyMethod):
SNGT_QHENOMENOLOGY_233: (4)                def __init__(
SNGT_QHENOMENOLOGY_234: (8)                    self,
SNGT_QHENOMENOLOGY_235: (8)                    function: Callable[[complex], complex],
SNGT_QHENOMENOLOGY_236: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_237: (8)                    **kwargs
SNGT_QHENOMENOLOGY_238: (4)                ):
SNGT_QHENOMENOLOGY_239: (8)                    self.function = function
SNGT_QHENOMENOLOGY_240: (8)                    method = mobject.apply_complex_function
SNGT_QHENOMENOLOGY_241: (8)                    super().__init__(method, function, **kwargs)
SNGT_QHENOMENOLOGY_242: (4)                def init_path_func(self) -> None:
SNGT_QHENOMENOLOGY_243: (8)                    func1 = self.function(complex(1))
SNGT_QHENOMENOLOGY_244: (8)                    self.path_arc = np.log(func1).imag
SNGT_QHENOMENOLOGY_245: (8)                    super().init_path_func()
SNGT_QHENOMENOLOGY_246: (0)            class CyclicReplace(Transform):
SNGT_QHENOMENOLOGY_247: (4)                def __init__(self, *mobjects: Mobject, path_arc=90 *
DEG, **kwargs):
SNGT_QHENOMENOLOGY_248: (8)                    super().__init__(Group(*mobjects),
path_arc=path_arc, **kwargs)
SNGT_QHENOMENOLOGY_249: (4)                def create_target(self) -> Mobject:
SNGT_QHENOMENOLOGY_250: (8)                    group = self.mobject
```

```
SNGT_QHENOMENOLOGY_251: (8)                     target = group.copy()
SNGT_QHENOMENOLOGY_252: (8)                     cycled_targets = [target[-1], *target[:-1]]
SNGT_QHENOMENOLOGY_253: (8)                     for m1, m2 in zip(cycled_targets, group):
SNGT_QHENOMENOLOGY_254: (12)                        m1.move_to(m2)
SNGT_QHENOMENOLOGY_255: (8)                     return target
SNGT_QHENOMENOLOGY_256: (0)             class Swap(CyclicReplace):
SNGT_QHENOMENOLOGY_257: (4)                 """Alternate name for CyclicReplace"""
SNGT_QHENOMENOLOGY_258: (4)                 pass
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 33 - functions.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from isosurfaces import plot_isoline
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import FRAME_X_RADIUS,
FRAME_Y_RADIUS
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import YELLOW
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_7: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                   from typing import Callable, Sequence, Tuple
SNGT_QHENOMENOLOGY_10: (4)                  from manimlib.typing import ManimColor, Vect3
SNGT_QHENOMENOLOGY_11: (0)              class ParametricCurve(VMobject):
SNGT_QHENOMENOLOGY_12: (4)                  def __init__(
SNGT_QHENOMENOLOGY_13: (8)                      self,
SNGT_QHENOMENOLOGY_14: (8)                      t_func: Callable[[float], Sequence[float] | Vect3],
SNGT_QHENOMENOLOGY_15: (8)                      t_range: Tuple[float, float, float] = (0, 1, 0.1),
SNGT_QHENOMENOLOGY_16: (8)                      epsilon: float = 1e-8,
SNGT_QHENOMENOLOGY_17: (8)                      discontinuities: Sequence[float] = [],
SNGT_QHENOMENOLOGY_18: (8)                      use_smoothing: bool = True,
SNGT_QHENOMENOLOGY_19: (8)                      **kwargs
SNGT_QHENOMENOLOGY_20: (4)                  ):
SNGT_QHENOMENOLOGY_21: (8)                      self.t_func = t_func
SNGT_QHENOMENOLOGY_22: (8)                      self.t_range = t_range
SNGT_QHENOMENOLOGY_23: (8)                      self.epsilon = epsilon
SNGT_QHENOMENOLOGY_24: (8)                      self.discontinuities = discontinuities
SNGT_QHENOMENOLOGY_25: (8)                      self.use_smoothing = use_smoothing
SNGT_QHENOMENOLOGY_26: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_27: (4)                  def get_point_from_function(self, t: float) -> Vect3:
SNGT_QHENOMENOLOGY_28: (8)                      return np.array(self.t_func(t))
SNGT_QHENOMENOLOGY_29: (4)                  def init_points(self):
SNGT_QHENOMENOLOGY_30: (8)                      t_min, t_max, step = self.t_range
SNGT_QHENOMENOLOGY_31: (8)                      jumps = np.array(self.discontinuities)
SNGT_QHENOMENOLOGY_32: (8)                      jumps = jumps[(jumps > t_min) & (jumps < t_max)]
SNGT_QHENOMENOLOGY_33: (8)                      boundary_times = [t_min, t_max, *(jumps -
self.epsilon), *(jumps + self.epsilon)]
SNGT_QHENOMENOLOGY_34: (8)                      boundary_times.sort()
SNGT_QHENOMENOLOGY_35: (8)                      for t1, t2 in zip(boundary_times[0::2],
boundary_times[1::2]):
SNGT_QHENOMENOLOGY_36: (12)                         t_range = [*np.arange(t1, t2, step), t2]
SNGT_QHENOMENOLOGY_37: (12)                         points = np.array([self.t_func(t) for t in
t_range])
SNGT_QHENOMENOLOGY_38: (12)                         self.start_new_path(points[0])
SNGT_QHENOMENOLOGY_39: (12)                         self.add_points_as_corners(points[1:])
SNGT_QHENOMENOLOGY_40: (8)                      if self.use_smoothing:
SNGT_QHENOMENOLOGY_41: (12)                         self.make_smooth(approx=True)
SNGT_QHENOMENOLOGY_42: (8)                      if not self.has_points():
SNGT_QHENOMENOLOGY_43: (12)                         self.set_points(np.array([self.t_func(t_min)]))
SNGT_QHENOMENOLOGY_44: (8)                      return self
SNGT_QHENOMENOLOGY_45: (4)                  def get_t_func(self):
SNGT_QHENOMENOLOGY_46: (8)                      return self.t_func
SNGT_QHENOMENOLOGY_47: (4)                  def get_function(self):
SNGT_QHENOMENOLOGY_48: (8)                      if hasattr(self, "underlying_function"):
SNGT_QHENOMENOLOGY_49: (12)                         return self.underlying_function
SNGT_QHENOMENOLOGY_50: (8)                      if hasattr(self, "function"):
SNGT_QHENOMENOLOGY_51: (12)                         return self.function
```

```
SNGT_QHENOMENOLOGY_52: (4)                 def get_x_range(self):
SNGT_QHENOMENOLOGY_53: (8)                     if hasattr(self, "x_range"):
SNGT_QHENOMENOLOGY_54: (12)                        return self.x_range
SNGT_QHENOMENOLOGY_55: (0)             class FunctionGraph(ParametricCurve):
SNGT_QHENOMENOLOGY_56: (4)                 def __init__(
SNGT_QHENOMENOLOGY_57: (8)                     self,
SNGT_QHENOMENOLOGY_58: (8)                     function: Callable[[float], float],
SNGT_QHENOMENOLOGY_59: (8)                     x_range: Tuple[float, float, float] = (-8, 8,
0.25),
SNGT_QHENOMENOLOGY_60: (8)                     color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_61: (8)                     **kwargs
SNGT_QHENOMENOLOGY_62: (4)                 ):
SNGT_QHENOMENOLOGY_63: (8)                     self.function = function
SNGT_QHENOMENOLOGY_64: (8)                     self.x_range = x_range
SNGT_QHENOMENOLOGY_65: (8)                     def parametric_function(t):
SNGT_QHENOMENOLOGY_66: (12)                        return [t, function(t), 0]
SNGT_QHENOMENOLOGY_67: (8)                     super().__init__(parametric_function, self.x_range,
**kwargs)
SNGT_QHENOMENOLOGY_68: (0)             class ImplicitFunction(VMobject):
SNGT_QHENOMENOLOGY_69: (4)                 def __init__(
SNGT_QHENOMENOLOGY_70: (8)                     self,
SNGT_QHENOMENOLOGY_71: (8)                     func: Callable[[float, float], float],
SNGT_QHENOMENOLOGY_72: (8)                     x_range: Tuple[float, float] = (-FRAME_X_RADIUS,
FRAME_X_RADIUS),
SNGT_QHENOMENOLOGY_73: (8)                     y_range: Tuple[float, float] = (-FRAME_Y_RADIUS,
FRAME_Y_RADIUS),
SNGT_QHENOMENOLOGY_74: (8)                     min_depth: int = 5,
SNGT_QHENOMENOLOGY_75: (8)                     max_quads: int = 1500,
SNGT_QHENOMENOLOGY_76: (8)                     use_smoothing: bool = False,
SNGT_QHENOMENOLOGY_77: (8)                     joint_type: str = 'no_joint',
SNGT_QHENOMENOLOGY_78: (8)                     **kwargs
SNGT_QHENOMENOLOGY_79: (4)                 ):
SNGT_QHENOMENOLOGY_80: (8)                     super().__init__(joint_type=joint_type, **kwargs)
SNGT_QHENOMENOLOGY_81: (8)                     p_min, p_max = (
SNGT_QHENOMENOLOGY_82: (12)                        np.array([x_range[0], y_range[0]]),
SNGT_QHENOMENOLOGY_83: (12)                        np.array([x_range[1], y_range[1]]),
SNGT_QHENOMENOLOGY_84: (8)                     )
SNGT_QHENOMENOLOGY_85: (8)                     curves = plot_isoline(
SNGT_QHENOMENOLOGY_86: (12)                        fn=lambda u: func(u[0], u[1]),
SNGT_QHENOMENOLOGY_87: (12)                        pmin=p_min,
SNGT_QHENOMENOLOGY_88: (12)                        pmax=p_max,
SNGT_QHENOMENOLOGY_89: (12)                        min_depth=min_depth,
SNGT_QHENOMENOLOGY_90: (12)                        max_quads=max_quads,
SNGT_QHENOMENOLOGY_91: (8)                     )  # returns a list of lists of 2D points
SNGT_QHENOMENOLOGY_92: (8)                     curves = [
SNGT_QHENOMENOLOGY_93: (12)                        np.pad(curve, [(0, 0), (0, 1)])
SNGT_QHENOMENOLOGY_94: (12)                        for curve in curves
SNGT_QHENOMENOLOGY_95: (12)                        if curve != []
SNGT_QHENOMENOLOGY_96: (8)                     ]  # add z coord as 0
SNGT_QHENOMENOLOGY_97: (8)                     for curve in curves:
SNGT_QHENOMENOLOGY_98: (12)                        self.start_new_path(curve[0])
SNGT_QHENOMENOLOGY_99: (12)                        self.add_points_as_corners(curve[1:])
SNGT_QHENOMENOLOGY_100: (8)                    if use_smoothing:
SNGT_QHENOMENOLOGY_101: (12)                       self.make_smooth()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 34 - event_type.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from enum import Enum
SNGT_QHENOMENOLOGY_2: (0)              class EventType(Enum):
SNGT_QHENOMENOLOGY_3: (4)                  MouseMotionEvent = 'mouse_motion_event'
SNGT_QHENOMENOLOGY_4: (4)                  MousePressEvent = 'mouse_press_event'
SNGT_QHENOMENOLOGY_5: (4)                  MouseReleaseEvent = 'mouse_release_event'
SNGT_QHENOMENOLOGY_6: (4)                  MouseDragEvent = 'mouse_drag_event'
SNGT_QHENOMENOLOGY_7: (4)                  MouseScrollEvent = 'mouse_scroll_event'
SNGT_QHENOMENOLOGY_8: (4)                  KeyPressEvent = 'key_press_event'
SNGT_QHENOMENOLOGY_9: (4)                  KeyReleaseEvent = 'key_release_event'
SNGT_QHENOMENOLOGY_
```

```
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 35 - specialized.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from manimlib.animation.composition import LaggedStart
SNGT_QHENOMENOLOGY_3: (0)              from manimlib.animation.transform import Restore
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.constants import BLACK, WHITE
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.mobject.geometry import Circle
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_7: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                  import numpy as np
SNGT_QHENOMENOLOGY_10: (4)                 from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_11: (0)             class Broadcast(LaggedStart):
SNGT_QHENOMENOLOGY_12: (4)                 def __init__(
SNGT_QHENOMENOLOGY_13: (8)                     self,
SNGT_QHENOMENOLOGY_14: (8)                     focal_point: np.ndarray,
SNGT_QHENOMENOLOGY_15: (8)                     small_radius: float = 0.0,
SNGT_QHENOMENOLOGY_16: (8)                     big_radius: float = 5.0,
SNGT_QHENOMENOLOGY_17: (8)                     n_circles: int = 5,
SNGT_QHENOMENOLOGY_18: (8)                     start_stroke_width: float = 8.0,
SNGT_QHENOMENOLOGY_19: (8)                     color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_20: (8)                     run_time: float = 3.0,
SNGT_QHENOMENOLOGY_21: (8)                     lag_ratio: float = 0.2,
SNGT_QHENOMENOLOGY_22: (8)                     remover: bool = True,
SNGT_QHENOMENOLOGY_23: (8)                     **kwargs
SNGT_QHENOMENOLOGY_24: (4)                 ):
SNGT_QHENOMENOLOGY_25: (8)                     self.focal_point = focal_point
SNGT_QHENOMENOLOGY_26: (8)                     self.small_radius = small_radius
SNGT_QHENOMENOLOGY_27: (8)                     self.big_radius = big_radius
SNGT_QHENOMENOLOGY_28: (8)                     self.n_circles = n_circles
SNGT_QHENOMENOLOGY_29: (8)                     self.start_stroke_width = start_stroke_width
SNGT_QHENOMENOLOGY_30: (8)                     self.color = color
SNGT_QHENOMENOLOGY_31: (8)                     circles = VGroup()
SNGT_QHENOMENOLOGY_32: (8)                     for x in range(n_circles):
SNGT_QHENOMENOLOGY_33: (12)                        circle = Circle(
SNGT_QHENOMENOLOGY_34: (16)                            radius=big_radius,
SNGT_QHENOMENOLOGY_35: (16)                            stroke_color=BLACK,
SNGT_QHENOMENOLOGY_36: (16)                            stroke_width=0,
SNGT_QHENOMENOLOGY_37: (12)                        )
SNGT_QHENOMENOLOGY_38: (12)                        circle.add_updater(lambda c:
c.move_to(focal_point))
SNGT_QHENOMENOLOGY_39: (12)                        circle.save_state()
SNGT_QHENOMENOLOGY_40: (12)                        circle.set_width(small_radius * 2)
SNGT_QHENOMENOLOGY_41: (12)                        circle.set_stroke(color, start_stroke_width)
SNGT_QHENOMENOLOGY_42: (12)                        circles.add(circle)
SNGT_QHENOMENOLOGY_43: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_44: (12)                        *map(Restore, circles),
SNGT_QHENOMENOLOGY_45: (12)                        run_time=run_time,
SNGT_QHENOMENOLOGY_46: (12)                        lag_ratio=lag_ratio,
SNGT_QHENOMENOLOGY_47: (12)                        remover=remover,
SNGT_QHENOMENOLOGY_48: (12)                        **kwargs
SNGT_QHENOMENOLOGY_49: (8)                     )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 36 - boolean_ops.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              import pathops
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_5: (0)              def _convert_vmobject_to_skia_path(vmobject: VMobject) ->
pathops.Path:
SNGT_QHENOMENOLOGY_6: (4)                  path = pathops.Path()
SNGT_QHENOMENOLOGY_7: (4)                  for submob in vmobject.family_members_with_points():
```

```
SNGT_QHENOMENOLOGY_8: (8)                       for subpath in submob.get_subpaths():
SNGT_QHENOMENOLOGY_9: (12)                          quads =
vmobject.get_bezier_tuples_from_points(subpath)
SNGT_QHENOMENOLOGY_10: (12)                         start = subpath[0]
SNGT_QHENOMENOLOGY_11: (12)                         path.moveTo(*start[:2])
SNGT_QHENOMENOLOGY_12: (12)                         for p0, p1, p2 in quads:
SNGT_QHENOMENOLOGY_13: (16)                             path.quadTo(*p1[:2], *p2[:2])
SNGT_QHENOMENOLOGY_14: (12)                         if vmobject.consider_points_equal(subpath[0],
subpath[-1]):
SNGT_QHENOMENOLOGY_15: (16)                             path.close()
SNGT_QHENOMENOLOGY_16: (4)                  return path
SNGT_QHENOMENOLOGY_17: (0)              def _convert_skia_path_to_vmobject(
SNGT_QHENOMENOLOGY_18: (4)                  path: pathops.Path,
SNGT_QHENOMENOLOGY_19: (4)                  vmobject: VMobject
SNGT_QHENOMENOLOGY_20: (0)              ) -> VMobject:
SNGT_QHENOMENOLOGY_21: (4)                  PathVerb = pathops.PathVerb
SNGT_QHENOMENOLOGY_22: (4)                  current_path_start = np.array([0.0, 0.0, 0.0])
SNGT_QHENOMENOLOGY_23: (4)                  for path_verb, points in path:
SNGT_QHENOMENOLOGY_24: (8)                      if path_verb == PathVerb.CLOSE:
SNGT_QHENOMENOLOGY_25: (12)                         vmobject.add_line_to(current_path_start)
SNGT_QHENOMENOLOGY_26: (8)                      else:
SNGT_QHENOMENOLOGY_27: (12)                         points = np.hstack((np.array(points),
np.zeros((len(points), 1))))
SNGT_QHENOMENOLOGY_28: (12)                         if path_verb == PathVerb.MOVE:
SNGT_QHENOMENOLOGY_29: (16)                             for point in points:
SNGT_QHENOMENOLOGY_30: (20)                                 current_path_start = point
SNGT_QHENOMENOLOGY_31: (20)                                 vmobject.start_new_path(point)
SNGT_QHENOMENOLOGY_32: (12)                         elif path_verb == PathVerb.CUBIC:
SNGT_QHENOMENOLOGY_33: (16)                             vmobject.add_cubic_bezier_curve_to(*points)
SNGT_QHENOMENOLOGY_34: (12)                         elif path_verb == PathVerb.LINE:
SNGT_QHENOMENOLOGY_35: (16)                             vmobject.add_line_to(points[0])
SNGT_QHENOMENOLOGY_36: (12)                         elif path_verb == PathVerb.QUAD:
SNGT_QHENOMENOLOGY_37: (16)
vmobject.add_quadratic_bezier_curve_to(*points)
SNGT_QHENOMENOLOGY_38: (12)                         else:
SNGT_QHENOMENOLOGY_39: (16)                             raise Exception(f"Unsupported:
{path_verb}")
SNGT_QHENOMENOLOGY_40: (4)                  return vmobject.reverse_points()
SNGT_QHENOMENOLOGY_41: (0)              class Union(VMobject):
SNGT_QHENOMENOLOGY_42: (4)                  def __init__(self, *vmobjects: VMobject, **kwargs):
SNGT_QHENOMENOLOGY_43: (8)                      if len(vmobjects) < 2:
SNGT_QHENOMENOLOGY_44: (12)                         raise ValueError("At least 2 mobjects needed
for Union.")
SNGT_QHENOMENOLOGY_45: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_46: (8)                      outpen = pathops.Path()
SNGT_QHENOMENOLOGY_47: (8)                      paths = [
SNGT_QHENOMENOLOGY_48: (12)                         _convert_vmobject_to_skia_path(vmobject)
SNGT_QHENOMENOLOGY_49: (12)                         for vmobject in vmobjects
SNGT_QHENOMENOLOGY_50: (8)                      ]
SNGT_QHENOMENOLOGY_51: (8)                      pathops.union(paths, outpen.getPen())
SNGT_QHENOMENOLOGY_52: (8)                      _convert_skia_path_to_vmobject(outpen, self)
SNGT_QHENOMENOLOGY_53: (0)              class Difference(VMobject):
SNGT_QHENOMENOLOGY_54: (4)                  def __init__(self, subject: VMobject, clip: VMobject,
**kwargs):
SNGT_QHENOMENOLOGY_55: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_56: (8)                      outpen = pathops.Path()
SNGT_QHENOMENOLOGY_57: (8)                      pathops.difference(
SNGT_QHENOMENOLOGY_58: (12)                         [_convert_vmobject_to_skia_path(subject)],
SNGT_QHENOMENOLOGY_59: (12)                         [_convert_vmobject_to_skia_path(clip)],
SNGT_QHENOMENOLOGY_60: (12)                         outpen.getPen(),
SNGT_QHENOMENOLOGY_61: (8)                      )
SNGT_QHENOMENOLOGY_62: (8)                      _convert_skia_path_to_vmobject(outpen, self)
SNGT_QHENOMENOLOGY_63: (0)              class Intersection(VMobject):
SNGT_QHENOMENOLOGY_64: (4)                  def __init__(self, *vmobjects: VMobject, **kwargs):
SNGT_QHENOMENOLOGY_65: (8)                      if len(vmobjects) < 2:
SNGT_QHENOMENOLOGY_66: (12)                         raise ValueError("At least 2 mobjects needed
for Intersection.")
SNGT_QHENOMENOLOGY_67: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_68: (8)                      outpen = pathops.Path()
```

```
SNGT_QHENOMENOLOGY_69: (8)                    pathops.intersection(
SNGT_QHENOMENOLOGY_70: (12)                       [_convert_vmobject_to_skia_path(vmobjects[0])],
SNGT_QHENOMENOLOGY_71: (12)                       [_convert_vmobject_to_skia_path(vmobjects[1])],
SNGT_QHENOMENOLOGY_72: (12)                       outpen.getPen(),
SNGT_QHENOMENOLOGY_73: (8)                    )
SNGT_QHENOMENOLOGY_74: (8)                    new_outpen = outpen
SNGT_QHENOMENOLOGY_75: (8)                    for _i in range(2, len(vmobjects)):
SNGT_QHENOMENOLOGY_76: (12)                       new_outpen = pathops.Path()
SNGT_QHENOMENOLOGY_77: (12)                       pathops.intersection(
SNGT_QHENOMENOLOGY_78: (16)                           [outpen],
SNGT_QHENOMENOLOGY_79: (16)
[_convert_vmobject_to_skia_path(vmobjects[_i])],
SNGT_QHENOMENOLOGY_80: (16)                           new_outpen.getPen(),
SNGT_QHENOMENOLOGY_81: (12)                       )
SNGT_QHENOMENOLOGY_82: (12)                       outpen = new_outpen
SNGT_QHENOMENOLOGY_83: (8)                    _convert_skia_path_to_vmobject(outpen, self)
SNGT_QHENOMENOLOGY_84: (0)            class Exclusion(VMobject):
SNGT_QHENOMENOLOGY_85: (4)                def __init__(self, *vmobjects: VMobject, **kwargs):
SNGT_QHENOMENOLOGY_86: (8)                    if len(vmobjects) < 2:
SNGT_QHENOMENOLOGY_87: (12)                       raise ValueError("At least 2 mobjects needed
for Exclusion.")
SNGT_QHENOMENOLOGY_88: (8)                    super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_89: (8)                    outpen = pathops.Path()
SNGT_QHENOMENOLOGY_90: (8)                    pathops.xor(
SNGT_QHENOMENOLOGY_91: (12)                       [_convert_vmobject_to_skia_path(vmobjects[0])],
SNGT_QHENOMENOLOGY_92: (12)                       [_convert_vmobject_to_skia_path(vmobjects[1])],
SNGT_QHENOMENOLOGY_93: (12)                       outpen.getPen(),
SNGT_QHENOMENOLOGY_94: (8)                    )
SNGT_QHENOMENOLOGY_95: (8)                    new_outpen = outpen
SNGT_QHENOMENOLOGY_96: (8)                    for _i in range(2, len(vmobjects)):
SNGT_QHENOMENOLOGY_97: (12)                       new_outpen = pathops.Path()
SNGT_QHENOMENOLOGY_98: (12)                       pathops.xor(
SNGT_QHENOMENOLOGY_99: (16)                           [outpen],
SNGT_QHENOMENOLOGY_100: (16)
[_convert_vmobject_to_skia_path(vmobjects[_i])],
SNGT_QHENOMENOLOGY_101: (16)                          new_outpen.getPen(),
SNGT_QHENOMENOLOGY_102: (12)                      )
SNGT_QHENOMENOLOGY_103: (12)                      outpen = new_outpen
SNGT_QHENOMENOLOGY_104: (8)                   _convert_skia_path_to_vmobject(outpen, self)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 37 - interactive.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import numpy as np
SNGT_QHENOMENOLOGY_3: (0)             from pyglet.window import key as PygletWindowKeys
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.constants import FRAME_HEIGHT, FRAME_WIDTH
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.constants import DOWN, LEFT, ORIGIN, RIGHT,
UP
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.constants import MED_LARGE_BUFF,
MED_SMALL_BUFF, SMALL_BUFF
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.constants import BLACK, BLUE, GREEN, GREY_A,
GREY_C, RED, WHITE
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_10: (0)            from manimlib.mobject.geometry import Circle
SNGT_QHENOMENOLOGY_11: (0)            from manimlib.mobject.geometry import Dot
SNGT_QHENOMENOLOGY_12: (0)            from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_13: (0)            from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_14: (0)            from manimlib.mobject.geometry import RoundedRectangle
SNGT_QHENOMENOLOGY_15: (0)            from manimlib.mobject.geometry import Square
SNGT_QHENOMENOLOGY_16: (0)            from manimlib.mobject.svg.text_mobject import Text
SNGT_QHENOMENOLOGY_17: (0)            from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_18: (0)            from manimlib.mobject.value_tracker import ValueTracker
SNGT_QHENOMENOLOGY_19: (0)            from manimlib.utils.color import rgb_to_hex
SNGT_QHENOMENOLOGY_20: (0)            from manimlib.utils.space_ops import
get_closest_point_on_line
```

```
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_22: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_23: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_24: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_25: (4)                 from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_26: (0)             class MotionMobject(Mobject):
SNGT_QHENOMENOLOGY_27: (4)                 """
SNGT_QHENOMENOLOGY_28: (8)                     You could hold and drag this object to any position
SNGT_QHENOMENOLOGY_29: (4)                 """
SNGT_QHENOMENOLOGY_30: (4)                 def __init__(self, mobject: Mobject, **kwargs):
SNGT_QHENOMENOLOGY_31: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_32: (8)                     assert isinstance(mobject, Mobject)
SNGT_QHENOMENOLOGY_33: (8)                     self.mobject = mobject
SNGT_QHENOMENOLOGY_34: (8)
self.mobject.add_mouse_drag_listner(self.mob_on_mouse_drag)
SNGT_QHENOMENOLOGY_35: (8)                     self.mobject.add_updater(lambda mob: None)
SNGT_QHENOMENOLOGY_36: (8)                     self.add(mobject)
SNGT_QHENOMENOLOGY_37: (4)                 def mob_on_mouse_drag(self, mob: Mobject, event_data:
dict[str, np.ndarray]) -> bool:
SNGT_QHENOMENOLOGY_38: (8)                     mob.move_to(event_data["point"])
SNGT_QHENOMENOLOGY_39: (8)                     return False
SNGT_QHENOMENOLOGY_40: (0)             class Button(Mobject):
SNGT_QHENOMENOLOGY_41: (4)                 """
SNGT_QHENOMENOLOGY_42: (8)                     Pass any mobject and register an on_click method
SNGT_QHENOMENOLOGY_43: (8)                     The on_click method takes mobject as argument like
updater
SNGT_QHENOMENOLOGY_44: (4)                 """
SNGT_QHENOMENOLOGY_45: (4)                 def __init__(self, mobject: Mobject, on_click:
Callable[[Mobject]], **kwargs):
SNGT_QHENOMENOLOGY_46: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_47: (8)                     assert isinstance(mobject, Mobject)
SNGT_QHENOMENOLOGY_48: (8)                     self.on_click = on_click
SNGT_QHENOMENOLOGY_49: (8)                     self.mobject = mobject
SNGT_QHENOMENOLOGY_50: (8)
self.mobject.add_mouse_press_listner(self.mob_on_mouse_press)
SNGT_QHENOMENOLOGY_51: (8)                     self.add(self.mobject)
SNGT_QHENOMENOLOGY_52: (4)                 def mob_on_mouse_press(self, mob: Mobject, event_data)
-> bool:
SNGT_QHENOMENOLOGY_53: (8)                     self.on_click(mob)
SNGT_QHENOMENOLOGY_54: (8)                     return False
SNGT_QHENOMENOLOGY_55: (0)             class ControlMobject(ValueTracker):
SNGT_QHENOMENOLOGY_56: (4)                 def __init__(self, value: float, *mobjects: Mobject,
**kwargs):
SNGT_QHENOMENOLOGY_57: (8)                     super().__init__(value=value, **kwargs)
SNGT_QHENOMENOLOGY_58: (8)                     self.add(*mobjects)
SNGT_QHENOMENOLOGY_59: (8)                     self.add_updater(lambda mob: None)
SNGT_QHENOMENOLOGY_60: (8)                     self.fix_in_frame()
SNGT_QHENOMENOLOGY_61: (4)                 def set_value(self, value: float):
SNGT_QHENOMENOLOGY_62: (8)                     self.assert_value(value)
SNGT_QHENOMENOLOGY_63: (8)                     self.set_value_anim(value)
SNGT_QHENOMENOLOGY_64: (8)                     return ValueTracker.set_value(self, value)
SNGT_QHENOMENOLOGY_65: (4)                 def assert_value(self, value):
SNGT_QHENOMENOLOGY_66: (8)                     pass
SNGT_QHENOMENOLOGY_67: (4)                 def set_value_anim(self, value):
SNGT_QHENOMENOLOGY_68: (8)                     pass
SNGT_QHENOMENOLOGY_69: (0)             class EnableDisableButton(ControlMobject):
SNGT_QHENOMENOLOGY_70: (4)                 def __init__(
SNGT_QHENOMENOLOGY_71: (8)                     self,
SNGT_QHENOMENOLOGY_72: (8)                     value: bool = True,
SNGT_QHENOMENOLOGY_73: (8)                     value_type: np.dtype = np.dtype(bool),
SNGT_QHENOMENOLOGY_74: (8)                     rect_kwargs: dict = {
SNGT_QHENOMENOLOGY_75: (12)                        "width": 0.5,
SNGT_QHENOMENOLOGY_76: (12)                        "height": 0.5,
SNGT_QHENOMENOLOGY_77: (12)                        "fill_opacity": 1.0
SNGT_QHENOMENOLOGY_78: (8)                     },
SNGT_QHENOMENOLOGY_79: (8)                     enable_color: ManimColor = GREEN,
SNGT_QHENOMENOLOGY_80: (8)                     disable_color: ManimColor = RED,
SNGT_QHENOMENOLOGY_81: (8)                     **kwargs
SNGT_QHENOMENOLOGY_82: (4)                 ):
```

```
SNGT_QHENOMENOLOGY_83: (8)                      self.value = value
SNGT_QHENOMENOLOGY_84: (8)                      self.value_type = value_type
SNGT_QHENOMENOLOGY_85: (8)                      self.rect_kwargs = rect_kwargs
SNGT_QHENOMENOLOGY_86: (8)                      self.enable_color = enable_color
SNGT_QHENOMENOLOGY_87: (8)                      self.disable_color = disable_color
SNGT_QHENOMENOLOGY_88: (8)                      self.box = Rectangle(**self.rect_kwargs)
SNGT_QHENOMENOLOGY_89: (8)                      super().__init__(value, self.box, **kwargs)
SNGT_QHENOMENOLOGY_90: (8)                      self.add_mouse_press_listner(self.on_mouse_press)
SNGT_QHENOMENOLOGY_91: (4)                  def assert_value(self, value: bool) -> None:
SNGT_QHENOMENOLOGY_92: (8)                      assert isinstance(value, bool)
SNGT_QHENOMENOLOGY_93: (4)                  def set_value_anim(self, value: bool) -> None:
SNGT_QHENOMENOLOGY_94: (8)                      if value:
SNGT_QHENOMENOLOGY_95: (12)                         self.box.set_fill(self.enable_color)
SNGT_QHENOMENOLOGY_96: (8)                      else:
SNGT_QHENOMENOLOGY_97: (12)                         self.box.set_fill(self.disable_color)
SNGT_QHENOMENOLOGY_98: (4)                  def toggle_value(self) -> None:
SNGT_QHENOMENOLOGY_99: (8)                      super().set_value(not self.get_value())
SNGT_QHENOMENOLOGY_100: (4)                 def on_mouse_press(self, mob: Mobject, event_data) ->
bool:
SNGT_QHENOMENOLOGY_101: (8)                     mob.toggle_value()
SNGT_QHENOMENOLOGY_102: (8)                     return False
SNGT_QHENOMENOLOGY_103: (0)             class Checkbox(ControlMobject):
SNGT_QHENOMENOLOGY_104: (4)                 def __init__(
SNGT_QHENOMENOLOGY_105: (8)                     self,
SNGT_QHENOMENOLOGY_106: (8)                     value: bool = True,
SNGT_QHENOMENOLOGY_107: (8)                     value_type: np.dtype = np.dtype(bool),
SNGT_QHENOMENOLOGY_108: (8)                     rect_kwargs: dict = {
SNGT_QHENOMENOLOGY_109: (12)                        "width": 0.5,
SNGT_QHENOMENOLOGY_110: (12)                        "height": 0.5,
SNGT_QHENOMENOLOGY_111: (12)                        "fill_opacity": 0.0
SNGT_QHENOMENOLOGY_112: (8)                     },
SNGT_QHENOMENOLOGY_113: (8)                     checkmark_kwargs: dict = {
SNGT_QHENOMENOLOGY_114: (12)                        "stroke_color": GREEN,
SNGT_QHENOMENOLOGY_115: (12)                        "stroke_width": 6,
SNGT_QHENOMENOLOGY_116: (8)                     },
SNGT_QHENOMENOLOGY_117: (8)                     cross_kwargs: dict = {
SNGT_QHENOMENOLOGY_118: (12)                        "stroke_color": RED,
SNGT_QHENOMENOLOGY_119: (12)                        "stroke_width": 6,
SNGT_QHENOMENOLOGY_120: (8)                     },
SNGT_QHENOMENOLOGY_121: (8)                     box_content_buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_122: (8)                     **kwargs
SNGT_QHENOMENOLOGY_123: (4)                 ):
SNGT_QHENOMENOLOGY_124: (8)                     self.value_type = value_type
SNGT_QHENOMENOLOGY_125: (8)                     self.rect_kwargs = rect_kwargs
SNGT_QHENOMENOLOGY_126: (8)                     self.checkmark_kwargs = checkmark_kwargs
SNGT_QHENOMENOLOGY_127: (8)                     self.cross_kwargs = cross_kwargs
SNGT_QHENOMENOLOGY_128: (8)                     self.box_content_buff = box_content_buff
SNGT_QHENOMENOLOGY_129: (8)                     self.box = Rectangle(**self.rect_kwargs)
SNGT_QHENOMENOLOGY_130: (8)                     self.box_content = self.get_checkmark() if value
else self.get_cross()
SNGT_QHENOMENOLOGY_131: (8)                     super().__init__(value, self.box, self.box_content,
**kwargs)
SNGT_QHENOMENOLOGY_132: (8)                     self.add_mouse_press_listner(self.on_mouse_press)
SNGT_QHENOMENOLOGY_133: (4)                 def assert_value(self, value: bool) -> None:
SNGT_QHENOMENOLOGY_134: (8)                     assert isinstance(value, bool)
SNGT_QHENOMENOLOGY_135: (4)                 def toggle_value(self) -> None:
SNGT_QHENOMENOLOGY_136: (8)                     super().set_value(not self.get_value())
SNGT_QHENOMENOLOGY_137: (4)                 def set_value_anim(self, value: bool) -> None:
SNGT_QHENOMENOLOGY_138: (8)                     if value:
SNGT_QHENOMENOLOGY_139: (12)                        self.box_content.become(self.get_checkmark())
SNGT_QHENOMENOLOGY_140: (8)                     else:
SNGT_QHENOMENOLOGY_141: (12)                        self.box_content.become(self.get_cross())
SNGT_QHENOMENOLOGY_142: (4)                 def on_mouse_press(self, mob: Mobject, event_data) ->
None:
SNGT_QHENOMENOLOGY_143: (8)                     mob.toggle_value()
SNGT_QHENOMENOLOGY_144: (8)                     return False
SNGT_QHENOMENOLOGY_145: (4)                 def get_checkmark(self) -> VGroup:
SNGT_QHENOMENOLOGY_146: (8)                     checkmark = VGroup(
SNGT_QHENOMENOLOGY_147: (12)                        Line(UP / 2 + 2 * LEFT, DOWN + LEFT,
```

```
**self.checkmark_kwargs),
SNGT_QHENOMENOLOGY_148: (12)                         Line(DOWN + LEFT, UP + RIGHT,
**self.checkmark_kwargs)
SNGT_QHENOMENOLOGY_149: (8)                     )
SNGT_QHENOMENOLOGY_150: (8)
checkmark.stretch_to_fit_width(self.box.get_width())
SNGT_QHENOMENOLOGY_151: (8)
checkmark.stretch_to_fit_height(self.box.get_height())
SNGT_QHENOMENOLOGY_152: (8)                     checkmark.scale(0.5)
SNGT_QHENOMENOLOGY_153: (8)                     checkmark.move_to(self.box)
SNGT_QHENOMENOLOGY_154: (8)                     return checkmark
SNGT_QHENOMENOLOGY_155: (4)                 def get_cross(self) -> VGroup:
SNGT_QHENOMENOLOGY_156: (8)                     cross = VGroup(
SNGT_QHENOMENOLOGY_157: (12)                         Line(UP + LEFT, DOWN + RIGHT,
**self.cross_kwargs),
SNGT_QHENOMENOLOGY_158: (12)                         Line(UP + RIGHT, DOWN + LEFT,
**self.cross_kwargs)
SNGT_QHENOMENOLOGY_159: (8)                     )
SNGT_QHENOMENOLOGY_160: (8)                     cross.stretch_to_fit_width(self.box.get_width())
SNGT_QHENOMENOLOGY_161: (8)                     cross.stretch_to_fit_height(self.box.get_height())
SNGT_QHENOMENOLOGY_162: (8)                     cross.scale(0.5)
SNGT_QHENOMENOLOGY_163: (8)                     cross.move_to(self.box)
SNGT_QHENOMENOLOGY_164: (8)                     return cross
SNGT_QHENOMENOLOGY_165: (0)             class LinearNumberSlider(ControlMobject):
SNGT_QHENOMENOLOGY_166: (4)                 def __init__(
SNGT_QHENOMENOLOGY_167: (8)                     self,
SNGT_QHENOMENOLOGY_168: (8)                     value: float = 0,
SNGT_QHENOMENOLOGY_169: (8)                     value_type: type = np.float64,
SNGT_QHENOMENOLOGY_170: (8)                     min_value: float = -10.0,
SNGT_QHENOMENOLOGY_171: (8)                     max_value: float = 10.0,
SNGT_QHENOMENOLOGY_172: (8)                     step: float = 1.0,
SNGT_QHENOMENOLOGY_173: (8)                     rounded_rect_kwargs: dict = {
SNGT_QHENOMENOLOGY_174: (12)                         "height": 0.075,
SNGT_QHENOMENOLOGY_175: (12)                         "width": 2,
SNGT_QHENOMENOLOGY_176: (12)                         "corner_radius": 0.0375
SNGT_QHENOMENOLOGY_177: (8)                     },
SNGT_QHENOMENOLOGY_178: (8)                     circle_kwargs: dict = {
SNGT_QHENOMENOLOGY_179: (12)                         "radius": 0.1,
SNGT_QHENOMENOLOGY_180: (12)                         "stroke_color": GREY_A,
SNGT_QHENOMENOLOGY_181: (12)                         "fill_color": GREY_A,
SNGT_QHENOMENOLOGY_182: (12)                         "fill_opacity": 1.0
SNGT_QHENOMENOLOGY_183: (8)                     },
SNGT_QHENOMENOLOGY_184: (8)                     **kwargs
SNGT_QHENOMENOLOGY_185: (4)                 ):
SNGT_QHENOMENOLOGY_186: (8)                     self.value_type = value_type
SNGT_QHENOMENOLOGY_187: (8)                     self.min_value = min_value
SNGT_QHENOMENOLOGY_188: (8)                     self.max_value = max_value
SNGT_QHENOMENOLOGY_189: (8)                     self.step = step
SNGT_QHENOMENOLOGY_190: (8)                     self.rounded_rect_kwargs = rounded_rect_kwargs
SNGT_QHENOMENOLOGY_191: (8)                     self.circle_kwargs = circle_kwargs
SNGT_QHENOMENOLOGY_192: (8)                     self.bar =
RoundedRectangle(**self.rounded_rect_kwargs)
SNGT_QHENOMENOLOGY_193: (8)                     self.slider = Circle(**self.circle_kwargs)
SNGT_QHENOMENOLOGY_194: (8)                     self.slider_axis = Line(
SNGT_QHENOMENOLOGY_195: (12)                         start=self.bar.get_bounding_box_point(LEFT),
SNGT_QHENOMENOLOGY_196: (12)                         end=self.bar.get_bounding_box_point(RIGHT)
SNGT_QHENOMENOLOGY_197: (8)                     )
SNGT_QHENOMENOLOGY_198: (8)                     self.slider_axis.set_opacity(0.0)
SNGT_QHENOMENOLOGY_199: (8)                     self.slider.move_to(self.slider_axis)
SNGT_QHENOMENOLOGY_200: (8)
self.slider.add_mouse_drag_listner(self.slider_on_mouse_drag)
SNGT_QHENOMENOLOGY_201: (8)                     super().__init__(value, self.bar, self.slider,
self.slider_axis, **kwargs)
SNGT_QHENOMENOLOGY_202: (4)                 def assert_value(self, value: float) -> None:
SNGT_QHENOMENOLOGY_203: (8)                     assert self.min_value <= value <= self.max_value
SNGT_QHENOMENOLOGY_204: (4)                 def set_value_anim(self, value: float) -> None:
SNGT_QHENOMENOLOGY_205: (8)                     prop = (value - self.min_value) / (self.max_value -
self.min_value)
SNGT_QHENOMENOLOGY_206: (8)
```

```
self.slider.move_to(self.slider_axis.point_from_proportion(prop))
SNGT_QHENOMENOLOGY_207: (4)                def slider_on_mouse_drag(self, mob, event_data:
dict[str, np.ndarray]) -> bool:
SNGT_QHENOMENOLOGY_208: (8)
self.set_value(self.get_value_from_point(event_data["point"]))
SNGT_QHENOMENOLOGY_209: (8)                    return False
SNGT_QHENOMENOLOGY_210: (4)                def get_value_from_point(self, point: np.ndarray) ->
float:
SNGT_QHENOMENOLOGY_211: (8)                    start, end = self.slider_axis.get_start_and_end()
SNGT_QHENOMENOLOGY_212: (8)                    point_on_line = get_closest_point_on_line(start,
end, point)
SNGT_QHENOMENOLOGY_213: (8)                    prop = get_norm(point_on_line - start) /
get_norm(end - start)
SNGT_QHENOMENOLOGY_214: (8)                    value = self.min_value + prop * (self.max_value -
self.min_value)
SNGT_QHENOMENOLOGY_215: (8)                    no_of_steps = int((value - self.min_value) /
self.step)
SNGT_QHENOMENOLOGY_216: (8)                    value_nearest_to_step = self.min_value +
no_of_steps * self.step
SNGT_QHENOMENOLOGY_217: (8)                    return value_nearest_to_step
SNGT_QHENOMENOLOGY_218: (0)            class ColorSliders(Group):
SNGT_QHENOMENOLOGY_219: (4)                def __init__(
SNGT_QHENOMENOLOGY_220: (8)                    self,
SNGT_QHENOMENOLOGY_221: (8)                    sliders_kwargs: dict = {},
SNGT_QHENOMENOLOGY_222: (8)                    rect_kwargs: dict = {
SNGT_QHENOMENOLOGY_223: (12)                       "width": 2.0,
SNGT_QHENOMENOLOGY_224: (12)                       "height": 0.5,
SNGT_QHENOMENOLOGY_225: (12)                       "stroke_opacity": 1.0
SNGT_QHENOMENOLOGY_226: (8)                    },
SNGT_QHENOMENOLOGY_227: (8)                    background_grid_kwargs: dict = {
SNGT_QHENOMENOLOGY_228: (12)                       "colors": [GREY_A, GREY_C],
SNGT_QHENOMENOLOGY_229: (12)                       "single_square_len": 0.1
SNGT_QHENOMENOLOGY_230: (8)                    },
SNGT_QHENOMENOLOGY_231: (8)                    sliders_buff: float = MED_LARGE_BUFF,
SNGT_QHENOMENOLOGY_232: (8)                    default_rgb_value: int = 255,
SNGT_QHENOMENOLOGY_233: (8)                    default_a_value: int = 1,
SNGT_QHENOMENOLOGY_234: (8)                    **kwargs
SNGT_QHENOMENOLOGY_235: (4)                ):
SNGT_QHENOMENOLOGY_236: (8)                    self.sliders_kwargs = sliders_kwargs
SNGT_QHENOMENOLOGY_237: (8)                    self.rect_kwargs = rect_kwargs
SNGT_QHENOMENOLOGY_238: (8)                    self.background_grid_kwargs =
background_grid_kwargs
SNGT_QHENOMENOLOGY_239: (8)                    self.sliders_buff = sliders_buff
SNGT_QHENOMENOLOGY_240: (8)                    self.default_rgb_value = default_rgb_value
SNGT_QHENOMENOLOGY_241: (8)                    self.default_a_value = default_a_value
SNGT_QHENOMENOLOGY_242: (8)                    rgb_kwargs = {"value": self.default_rgb_value,
"min_value": 0, "max_value": 255, "step": 1}
SNGT_QHENOMENOLOGY_243: (8)                    a_kwargs = {"value": self.default_a_value,
"min_value": 0, "max_value": 1, "step": 0.04}
SNGT_QHENOMENOLOGY_244: (8)                    self.r_slider =
LinearNumberSlider(**self.sliders_kwargs, **rgb_kwargs)
SNGT_QHENOMENOLOGY_245: (8)                    self.g_slider =
LinearNumberSlider(**self.sliders_kwargs, **rgb_kwargs)
SNGT_QHENOMENOLOGY_246: (8)                    self.b_slider =
LinearNumberSlider(**self.sliders_kwargs, **rgb_kwargs)
SNGT_QHENOMENOLOGY_247: (8)                    self.a_slider =
LinearNumberSlider(**self.sliders_kwargs, **a_kwargs)
SNGT_QHENOMENOLOGY_248: (8)                    self.sliders = Group(
SNGT_QHENOMENOLOGY_249: (12)                       self.r_slider,
SNGT_QHENOMENOLOGY_250: (12)                       self.g_slider,
SNGT_QHENOMENOLOGY_251: (12)                       self.b_slider,
SNGT_QHENOMENOLOGY_252: (12)                       self.a_slider
SNGT_QHENOMENOLOGY_253: (8)                    )
SNGT_QHENOMENOLOGY_254: (8)                    self.sliders.arrange(DOWN, buff=self.sliders_buff)
SNGT_QHENOMENOLOGY_255: (8)                    self.r_slider.slider.set_color(RED)
SNGT_QHENOMENOLOGY_256: (8)                    self.g_slider.slider.set_color(GREEN)
SNGT_QHENOMENOLOGY_257: (8)                    self.b_slider.slider.set_color(BLUE)
SNGT_QHENOMENOLOGY_258: (8)                    self.a_slider.slider.set_color_by_gradient(BLACK,
WHITE)
```

```
SNGT_QHENOMENOLOGY_259: (8)                     self.selected_color_box =
Rectangle(**self.rect_kwargs)
SNGT_QHENOMENOLOGY_260: (8)                     self.selected_color_box.add_updater(
SNGT_QHENOMENOLOGY_261: (12)                        lambda mob: mob.set_fill(
SNGT_QHENOMENOLOGY_262: (16)                            self.get_picked_color(),
self.get_picked_opacity()
SNGT_QHENOMENOLOGY_263: (12)                        )
SNGT_QHENOMENOLOGY_264: (8)                     )
SNGT_QHENOMENOLOGY_265: (8)                     self.background = self.get_background()
SNGT_QHENOMENOLOGY_266: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_267: (12)                        Group(self.background,
self.selected_color_box).fix_in_frame(),
SNGT_QHENOMENOLOGY_268: (12)                        self.sliders,
SNGT_QHENOMENOLOGY_269: (12)                        **kwargs
SNGT_QHENOMENOLOGY_270: (8)                     )
SNGT_QHENOMENOLOGY_271: (8)                     self.arrange(DOWN)
SNGT_QHENOMENOLOGY_272: (4)                 def get_background(self) -> VGroup:
SNGT_QHENOMENOLOGY_273: (8)                     single_square_len =
self.background_grid_kwargs["single_square_len"]
SNGT_QHENOMENOLOGY_274: (8)                     colors = self.background_grid_kwargs["colors"]
SNGT_QHENOMENOLOGY_275: (8)                     width = self.rect_kwargs["width"]
SNGT_QHENOMENOLOGY_276: (8)                     height = self.rect_kwargs["height"]
SNGT_QHENOMENOLOGY_277: (8)                     rows = int(height / single_square_len)
SNGT_QHENOMENOLOGY_278: (8)                     cols = int(width / single_square_len)
SNGT_QHENOMENOLOGY_279: (8)                     cols = (cols + 1) if (cols % 2 == 0) else cols
SNGT_QHENOMENOLOGY_280: (8)                     single_square = Square(single_square_len)
SNGT_QHENOMENOLOGY_281: (8)                     grid = single_square.get_grid(n_rows=rows,
n_cols=cols, buff=0.0)
SNGT_QHENOMENOLOGY_282: (8)                     grid.stretch_to_fit_width(width)
SNGT_QHENOMENOLOGY_283: (8)                     grid.stretch_to_fit_height(height)
SNGT_QHENOMENOLOGY_284: (8)                     grid.move_to(self.selected_color_box)
SNGT_QHENOMENOLOGY_285: (8)                     for idx, square in enumerate(grid):
SNGT_QHENOMENOLOGY_286: (12)                        assert isinstance(square, Square)
SNGT_QHENOMENOLOGY_287: (12)                        square.set_stroke(width=0.0, opacity=0.0)
SNGT_QHENOMENOLOGY_288: (12)                        square.set_fill(colors[idx % len(colors)], 1.0)
SNGT_QHENOMENOLOGY_289: (8)                     return grid
SNGT_QHENOMENOLOGY_290: (4)                 def set_value(self, r: float, g: float, b: float, a:
float):
SNGT_QHENOMENOLOGY_291: (8)                     self.r_slider.set_value(r)
SNGT_QHENOMENOLOGY_292: (8)                     self.g_slider.set_value(g)
SNGT_QHENOMENOLOGY_293: (8)                     self.b_slider.set_value(b)
SNGT_QHENOMENOLOGY_294: (8)                     self.a_slider.set_value(a)
SNGT_QHENOMENOLOGY_295: (4)                 def get_value(self) -> np.ndarary:
SNGT_QHENOMENOLOGY_296: (8)                     r = self.r_slider.get_value() / 255
SNGT_QHENOMENOLOGY_297: (8)                     g = self.g_slider.get_value() / 255
SNGT_QHENOMENOLOGY_298: (8)                     b = self.b_slider.get_value() / 255
SNGT_QHENOMENOLOGY_299: (8)                     alpha = self.a_slider.get_value()
SNGT_QHENOMENOLOGY_300: (8)                     return np.array((r, g, b, alpha))
SNGT_QHENOMENOLOGY_301: (4)                 def get_picked_color(self) -> str:
SNGT_QHENOMENOLOGY_302: (8)                     rgba = self.get_value()
SNGT_QHENOMENOLOGY_303: (8)                     return rgb_to_hex(rgba[:3])
SNGT_QHENOMENOLOGY_304: (4)                 def get_picked_opacity(self) -> float:
SNGT_QHENOMENOLOGY_305: (8)                     rgba = self.get_value()
SNGT_QHENOMENOLOGY_306: (8)                     return rgba[3]
SNGT_QHENOMENOLOGY_307: (0)             class Textbox(ControlMobject):
SNGT_QHENOMENOLOGY_308: (4)                 def __init__(
SNGT_QHENOMENOLOGY_309: (8)                     self,
SNGT_QHENOMENOLOGY_310: (8)                     value: str = "",
SNGT_QHENOMENOLOGY_311: (8)                     value_type: np.dtype = np.dtype(object),
SNGT_QHENOMENOLOGY_312: (8)                     box_kwargs: dict = {
SNGT_QHENOMENOLOGY_313: (12)                        "width": 2.0,
SNGT_QHENOMENOLOGY_314: (12)                        "height": 1.0,
SNGT_QHENOMENOLOGY_315: (12)                        "fill_color": WHITE,
SNGT_QHENOMENOLOGY_316: (12)                        "fill_opacity": 1.0,
SNGT_QHENOMENOLOGY_317: (8)                     },
SNGT_QHENOMENOLOGY_318: (8)                     text_kwargs: dict = {
SNGT_QHENOMENOLOGY_319: (12)                        "color": BLUE
SNGT_QHENOMENOLOGY_320: (8)                     },
SNGT_QHENOMENOLOGY_321: (8)                     text_buff: float = MED_SMALL_BUFF,
```

```
SNGT_QHENOMENOLOGY_322: (8)                     isInitiallyActive: bool = False,
SNGT_QHENOMENOLOGY_323: (8)                     active_color: ManimColor = BLUE,
SNGT_QHENOMENOLOGY_324: (8)                     deactive_color: ManimColor = RED,
SNGT_QHENOMENOLOGY_325: (8)                     **kwargs
SNGT_QHENOMENOLOGY_326: (4)                 ):
SNGT_QHENOMENOLOGY_327: (8)                     self.value_type = value_type
SNGT_QHENOMENOLOGY_328: (8)                     self.box_kwargs = box_kwargs
SNGT_QHENOMENOLOGY_329: (8)                     self.text_kwargs = text_kwargs
SNGT_QHENOMENOLOGY_330: (8)                     self.text_buff = text_buff
SNGT_QHENOMENOLOGY_331: (8)                     self.isInitiallyActive = isInitiallyActive
SNGT_QHENOMENOLOGY_332: (8)                     self.active_color = active_color
SNGT_QHENOMENOLOGY_333: (8)                     self.deactive_color = deactive_color
SNGT_QHENOMENOLOGY_334: (8)                     self.isActive = self.isInitiallyActive
SNGT_QHENOMENOLOGY_335: (8)                     self.box = Rectangle(**self.box_kwargs)
SNGT_QHENOMENOLOGY_336: (8)
self.box.add_mouse_press_listner(self.box_on_mouse_press)
SNGT_QHENOMENOLOGY_337: (8)                     self.text = Text(value, **self.text_kwargs)
SNGT_QHENOMENOLOGY_338: (8)                     super().__init__(value, self.box, self.text,
**kwargs)
SNGT_QHENOMENOLOGY_339: (8)                     self.update_text(value)
SNGT_QHENOMENOLOGY_340: (8)                     self.active_anim(self.isActive)
SNGT_QHENOMENOLOGY_341: (8)                     self.add_key_press_listner(self.on_key_press)
SNGT_QHENOMENOLOGY_342: (4)                 def set_value_anim(self, value: str) -> None:
SNGT_QHENOMENOLOGY_343: (8)                     self.update_text(value)
SNGT_QHENOMENOLOGY_344: (4)                 def update_text(self, value: str) -> None:
SNGT_QHENOMENOLOGY_345: (8)                     text = self.text
SNGT_QHENOMENOLOGY_346: (8)                     self.remove(text)
SNGT_QHENOMENOLOGY_347: (8)                     text.__init__(value, **self.text_kwargs)
SNGT_QHENOMENOLOGY_348: (8)                     height = text.get_height()
SNGT_QHENOMENOLOGY_349: (8)                     text.set_width(self.box.get_width() - 2 *
self.text_buff)
SNGT_QHENOMENOLOGY_350: (8)                     if text.get_height() > height:
SNGT_QHENOMENOLOGY_351: (12)                        text.set_height(height)
SNGT_QHENOMENOLOGY_352: (8)                     text.add_updater(lambda mob: mob.move_to(self.box))
SNGT_QHENOMENOLOGY_353: (8)                     text.fix_in_frame()
SNGT_QHENOMENOLOGY_354: (8)                     self.add(text)
SNGT_QHENOMENOLOGY_355: (4)                 def active_anim(self, isActive: bool) -> None:
SNGT_QHENOMENOLOGY_356: (8)                     if isActive:
SNGT_QHENOMENOLOGY_357: (12)                        self.box.set_stroke(self.active_color)
SNGT_QHENOMENOLOGY_358: (8)                     else:
SNGT_QHENOMENOLOGY_359: (12)                        self.box.set_stroke(self.deactive_color)
SNGT_QHENOMENOLOGY_360: (4)                 def box_on_mouse_press(self, mob, event_data) -> bool:
SNGT_QHENOMENOLOGY_361: (8)                     self.isActive = not self.isActive
SNGT_QHENOMENOLOGY_362: (8)                     self.active_anim(self.isActive)
SNGT_QHENOMENOLOGY_363: (8)                     return False
SNGT_QHENOMENOLOGY_364: (4)                 def on_key_press(self, mob: Mobject, event_data:
dict[str, int]) -> bool | None:
SNGT_QHENOMENOLOGY_365: (8)                     symbol = event_data["symbol"]
SNGT_QHENOMENOLOGY_366: (8)                     modifiers = event_data["modifiers"]
SNGT_QHENOMENOLOGY_367: (8)                     char = chr(symbol)
SNGT_QHENOMENOLOGY_368: (8)                     if mob.isActive:
SNGT_QHENOMENOLOGY_369: (12)                        old_value = mob.get_value()
SNGT_QHENOMENOLOGY_370: (12)                        new_value = old_value
SNGT_QHENOMENOLOGY_371: (12)                        if char.isalnum():
SNGT_QHENOMENOLOGY_372: (16)                            if (modifiers & PygletWindowKeys.MOD_SHIFT)
or (modifiers & PygletWindowKeys.MOD_CAPSLOCK):
SNGT_QHENOMENOLOGY_373: (20)                                new_value = old_value + char.upper()
SNGT_QHENOMENOLOGY_374: (16)                            else:
SNGT_QHENOMENOLOGY_375: (20)                                new_value = old_value + char.lower()
SNGT_QHENOMENOLOGY_376: (12)                        elif symbol in [PygletWindowKeys.SPACE]:
SNGT_QHENOMENOLOGY_377: (16)                            new_value = old_value + char
SNGT_QHENOMENOLOGY_378: (12)                        elif symbol == PygletWindowKeys.TAB:
SNGT_QHENOMENOLOGY_379: (16)                            new_value = old_value + '\t'
SNGT_QHENOMENOLOGY_380: (12)                        elif symbol == PygletWindowKeys.BACKSPACE:
SNGT_QHENOMENOLOGY_381: (16)                            new_value = old_value[:-1] or ''
SNGT_QHENOMENOLOGY_382: (12)                        mob.set_value(new_value)
SNGT_QHENOMENOLOGY_383: (12)                        return False
SNGT_QHENOMENOLOGY_384: (0)             class ControlPanel(Group):
SNGT_QHENOMENOLOGY_385: (4)                 def __init__(
```

```
SNGT_QHENOMENOLOGY_386: (8)                     self,
SNGT_QHENOMENOLOGY_387: (8)                     *controls: ControlMobject,
SNGT_QHENOMENOLOGY_388: (8)                     panel_kwargs: dict = {
SNGT_QHENOMENOLOGY_389: (12)                        "width": FRAME_WIDTH / 4,
SNGT_QHENOMENOLOGY_390: (12)                        "height": MED_SMALL_BUFF + FRAME_HEIGHT,
SNGT_QHENOMENOLOGY_391: (12)                        "fill_color": GREY_C,
SNGT_QHENOMENOLOGY_392: (12)                        "fill_opacity": 1.0,
SNGT_QHENOMENOLOGY_393: (12)                        "stroke_width": 0.0
SNGT_QHENOMENOLOGY_394: (8)                     },
SNGT_QHENOMENOLOGY_395: (8)                     opener_kwargs: dict = {
SNGT_QHENOMENOLOGY_396: (12)                        "width": FRAME_WIDTH / 8,
SNGT_QHENOMENOLOGY_397: (12)                        "height": 0.5,
SNGT_QHENOMENOLOGY_398: (12)                        "fill_color": GREY_C,
SNGT_QHENOMENOLOGY_399: (12)                        "fill_opacity": 1.0
SNGT_QHENOMENOLOGY_400: (8)                     },
SNGT_QHENOMENOLOGY_401: (8)                     opener_text_kwargs: dict = {
SNGT_QHENOMENOLOGY_402: (12)                        "text": "Control Panel",
SNGT_QHENOMENOLOGY_403: (12)                        "font_size": 20
SNGT_QHENOMENOLOGY_404: (8)                     },
SNGT_QHENOMENOLOGY_405: (8)                     **kwargs
SNGT_QHENOMENOLOGY_406: (4)                 ):
SNGT_QHENOMENOLOGY_407: (8)                     self.panel_kwargs = panel_kwargs
SNGT_QHENOMENOLOGY_408: (8)                     self.opener_kwargs = opener_kwargs
SNGT_QHENOMENOLOGY_409: (8)                     self.opener_text_kwargs = opener_text_kwargs
SNGT_QHENOMENOLOGY_410: (8)                     self.panel = Rectangle(**self.panel_kwargs)
SNGT_QHENOMENOLOGY_411: (8)                     self.panel.to_corner(UP + LEFT, buff=0)
SNGT_QHENOMENOLOGY_412: (8)                     self.panel.shift(self.panel.get_height() * UP)
SNGT_QHENOMENOLOGY_413: (8)
self.panel.add_mouse_scroll_listner(self.panel_on_mouse_scroll)
SNGT_QHENOMENOLOGY_414: (8)                     self.panel_opener_rect =
Rectangle(**self.opener_kwargs)
SNGT_QHENOMENOLOGY_415: (8)                     self.panel_info_text =
Text(**self.opener_text_kwargs)
SNGT_QHENOMENOLOGY_416: (8)
self.panel_info_text.move_to(self.panel_opener_rect)
SNGT_QHENOMENOLOGY_417: (8)                     self.panel_opener = Group(self.panel_opener_rect,
self.panel_info_text)
SNGT_QHENOMENOLOGY_418: (8)                     self.panel_opener.next_to(self.panel, DOWN,
aligned_edge=DOWN)
SNGT_QHENOMENOLOGY_419: (8)
self.panel_opener.add_mouse_drag_listner(self.panel_opener_on_mouse_drag)
SNGT_QHENOMENOLOGY_420: (8)                     self.controls = Group(*controls)
SNGT_QHENOMENOLOGY_421: (8)                     self.controls.arrange(DOWN, center=False,
aligned_edge=ORIGIN)
SNGT_QHENOMENOLOGY_422: (8)                     self.controls.move_to(self.panel)
SNGT_QHENOMENOLOGY_423: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_424: (12)                        self.panel, self.panel_opener,
SNGT_QHENOMENOLOGY_425: (12)                        self.controls,
SNGT_QHENOMENOLOGY_426: (12)                        **kwargs
SNGT_QHENOMENOLOGY_427: (8)                     )
SNGT_QHENOMENOLOGY_428: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_429: (8)                     self.fix_in_frame()
SNGT_QHENOMENOLOGY_430: (4)                 def move_panel_and_controls_to_panel_opener(self) ->
None:
SNGT_QHENOMENOLOGY_431: (8)                     self.panel.next_to(
SNGT_QHENOMENOLOGY_432: (12)                        self.panel_opener_rect,
SNGT_QHENOMENOLOGY_433: (12)                        direction=UP,
SNGT_QHENOMENOLOGY_434: (12)                        buff=0
SNGT_QHENOMENOLOGY_435: (8)                     )
SNGT_QHENOMENOLOGY_436: (8)                     controls_old_x = self.controls.get_x()
SNGT_QHENOMENOLOGY_437: (8)                     self.controls.next_to(
SNGT_QHENOMENOLOGY_438: (12)                        self.panel_opener_rect,
SNGT_QHENOMENOLOGY_439: (12)                        direction=UP,
SNGT_QHENOMENOLOGY_440: (12)                        buff=MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_441: (8)                     )
SNGT_QHENOMENOLOGY_442: (8)                     self.controls.set_x(controls_old_x)
SNGT_QHENOMENOLOGY_443: (4)                 def add_controls(self, *new_controls: ControlMobject) -
> None:
SNGT_QHENOMENOLOGY_444: (8)                     self.controls.add(*new_controls)
```

```
SNGT_QHENOMENOLOGY_445: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_446: (4)                 def remove_controls(self, *controls_to_remove:
ControlMobject) -> None:
SNGT_QHENOMENOLOGY_447: (8)                     self.controls.remove(*controls_to_remove)
SNGT_QHENOMENOLOGY_448: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_449: (4)                 def open_panel(self):
SNGT_QHENOMENOLOGY_450: (8)                     panel_opener_x = self.panel_opener.get_x()
SNGT_QHENOMENOLOGY_451: (8)                     self.panel_opener.to_corner(DOWN + LEFT, buff=0.0)
SNGT_QHENOMENOLOGY_452: (8)                     self.panel_opener.set_x(panel_opener_x)
SNGT_QHENOMENOLOGY_453: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_454: (8)                     return self
SNGT_QHENOMENOLOGY_455: (4)                 def close_panel(self):
SNGT_QHENOMENOLOGY_456: (8)                     panel_opener_x = self.panel_opener.get_x()
SNGT_QHENOMENOLOGY_457: (8)                     self.panel_opener.to_corner(UP + LEFT, buff=0.0)
SNGT_QHENOMENOLOGY_458: (8)                     self.panel_opener.set_x(panel_opener_x)
SNGT_QHENOMENOLOGY_459: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_460: (8)                     return self
SNGT_QHENOMENOLOGY_461: (4)                 def panel_opener_on_mouse_drag(self, mob, event_data:
dict[str, np.ndarray]) -> bool:
SNGT_QHENOMENOLOGY_462: (8)                     point = event_data["point"]
SNGT_QHENOMENOLOGY_463: (8)                     self.panel_opener.match_y(Dot(point))
SNGT_QHENOMENOLOGY_464: (8)                     self.move_panel_and_controls_to_panel_opener()
SNGT_QHENOMENOLOGY_465: (8)                     return False
SNGT_QHENOMENOLOGY_466: (4)                 def panel_on_mouse_scroll(self, mob, event_data:
dict[str, np.ndarray]) -> bool:
SNGT_QHENOMENOLOGY_467: (8)                     offset = event_data["offset"]
SNGT_QHENOMENOLOGY_468: (8)                     factor = 10 * offset[1]
SNGT_QHENOMENOLOGY_469: (8)                     self.controls.set_y(self.controls.get_y() + factor)
SNGT_QHENOMENOLOGY_470: (8)                     return False
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 38 - number_line.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.constants import DOWN, LEFT, RIGHT, UP
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import GREY_B
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.numbers import DecimalNumber
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.bezier import outer_interpolate
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.utils.dict_ops import merge_dicts_recursively
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.simple_functions import fdiv
SNGT_QHENOMENOLOGY_13: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_14: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_15: (4)                  from typing import Iterable, Optional
SNGT_QHENOMENOLOGY_16: (4)                  from manimlib.typing import ManimColor, Vect3,
Vect3Array, VectN, RangeSpecifier
SNGT_QHENOMENOLOGY_17: (0)              class NumberLine(Line):
SNGT_QHENOMENOLOGY_18: (4)                  def __init__(
SNGT_QHENOMENOLOGY_19: (8)                      self,
SNGT_QHENOMENOLOGY_20: (8)                      x_range: RangeSpecifier = (-8, 8, 1),
SNGT_QHENOMENOLOGY_21: (8)                      color: ManimColor = GREY_B,
SNGT_QHENOMENOLOGY_22: (8)                      stroke_width: float = 2.0,
SNGT_QHENOMENOLOGY_23: (8)                      unit_size: float = 1.0,
SNGT_QHENOMENOLOGY_24: (8)                      width: Optional[float] = None,
SNGT_QHENOMENOLOGY_25: (8)                      include_ticks: bool = True,
SNGT_QHENOMENOLOGY_26: (8)                      tick_size: float = 0.1,
SNGT_QHENOMENOLOGY_27: (8)                      longer_tick_multiple: float = 1.5,
SNGT_QHENOMENOLOGY_28: (8)                      tick_offset: float = 0.0,
SNGT_QHENOMENOLOGY_29: (8)                      big_tick_spacing: Optional[float] = None,
SNGT_QHENOMENOLOGY_30: (8)                      big_tick_numbers: list[float] = [],
SNGT_QHENOMENOLOGY_31: (8)                      include_numbers: bool = False,
SNGT_QHENOMENOLOGY_32: (8)                      line_to_number_direction: Vect3 = DOWN,
SNGT_QHENOMENOLOGY_33: (8)                      line_to_number_buff: float = MED_SMALL_BUFF,
```

```
SNGT_QHENOMENOLOGY_34: (8)                    include_tip: bool = False,
SNGT_QHENOMENOLOGY_35: (8)                    tip_config: dict = dict(
SNGT_QHENOMENOLOGY_36: (12)                       width=0.25,
SNGT_QHENOMENOLOGY_37: (12)                       length=0.25,
SNGT_QHENOMENOLOGY_38: (8)                    ),
SNGT_QHENOMENOLOGY_39: (8)                    decimal_number_config: dict = dict(
SNGT_QHENOMENOLOGY_40: (12)                       num_decimal_places=0,
SNGT_QHENOMENOLOGY_41: (12)                       font_size=36,
SNGT_QHENOMENOLOGY_42: (8)                    ),
SNGT_QHENOMENOLOGY_43: (8)                    numbers_to_exclude: list | None = None,
SNGT_QHENOMENOLOGY_44: (8)                    **kwargs,
SNGT_QHENOMENOLOGY_45: (4)                ):
SNGT_QHENOMENOLOGY_46: (8)                    self.x_range = x_range
SNGT_QHENOMENOLOGY_47: (8)                    self.tick_size = tick_size
SNGT_QHENOMENOLOGY_48: (8)                    self.longer_tick_multiple = longer_tick_multiple
SNGT_QHENOMENOLOGY_49: (8)                    self.tick_offset = tick_offset
SNGT_QHENOMENOLOGY_50: (8)                    if big_tick_spacing is not None:
SNGT_QHENOMENOLOGY_51: (12)                       self.big_tick_numbers = np.arange(
SNGT_QHENOMENOLOGY_52: (16)                           x_range[0],
SNGT_QHENOMENOLOGY_53: (16)                           x_range[1] + big_tick_spacing,
SNGT_QHENOMENOLOGY_54: (16)                           big_tick_spacing,
SNGT_QHENOMENOLOGY_55: (12)                       )
SNGT_QHENOMENOLOGY_56: (8)                    else:
SNGT_QHENOMENOLOGY_57: (12)                       self.big_tick_numbers = list(big_tick_numbers)
SNGT_QHENOMENOLOGY_58: (8)                    self.line_to_number_direction =
line_to_number_direction
SNGT_QHENOMENOLOGY_59: (8)                    self.line_to_number_buff = line_to_number_buff
SNGT_QHENOMENOLOGY_60: (8)                    self.include_tip = include_tip
SNGT_QHENOMENOLOGY_61: (8)                    self.tip_config = dict(tip_config)
SNGT_QHENOMENOLOGY_62: (8)                    self.decimal_number_config =
dict(decimal_number_config)
SNGT_QHENOMENOLOGY_63: (8)                    self.numbers_to_exclude = numbers_to_exclude
SNGT_QHENOMENOLOGY_64: (8)                    self.x_min, self.x_max = x_range[:2]
SNGT_QHENOMENOLOGY_65: (8)                    self.x_step = 1 if len(x_range) == 2 else
x_range[2]
SNGT_QHENOMENOLOGY_66: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_67: (12)                       self.x_min * RIGHT, self.x_max * RIGHT,
SNGT_QHENOMENOLOGY_68: (12)                       color=color,
SNGT_QHENOMENOLOGY_69: (12)                       stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_70: (12)                       **kwargs
SNGT_QHENOMENOLOGY_71: (8)                    )
SNGT_QHENOMENOLOGY_72: (8)                    if width:
SNGT_QHENOMENOLOGY_73: (12)                       self.set_width(width)
SNGT_QHENOMENOLOGY_74: (8)                    else:
SNGT_QHENOMENOLOGY_75: (12)                       self.scale(unit_size)
SNGT_QHENOMENOLOGY_76: (8)                    self.center()
SNGT_QHENOMENOLOGY_77: (8)                    if include_tip:
SNGT_QHENOMENOLOGY_78: (12)                       self.add_tip()
SNGT_QHENOMENOLOGY_79: (12)                       self.tip.set_stroke(
SNGT_QHENOMENOLOGY_80: (16)                           self.stroke_color,
SNGT_QHENOMENOLOGY_81: (16)                           self.stroke_width,
SNGT_QHENOMENOLOGY_82: (12)                       )
SNGT_QHENOMENOLOGY_83: (8)                    if include_ticks:
SNGT_QHENOMENOLOGY_84: (12)                       self.add_ticks()
SNGT_QHENOMENOLOGY_85: (8)                    if include_numbers:
SNGT_QHENOMENOLOGY_86: (12)
self.add_numbers(excluding=self.numbers_to_exclude)
SNGT_QHENOMENOLOGY_87: (4)                def get_tick_range(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_88: (8)                    if self.include_tip:
SNGT_QHENOMENOLOGY_89: (12)                       x_max = self.x_max
SNGT_QHENOMENOLOGY_90: (8)                    else:
SNGT_QHENOMENOLOGY_91: (12)                       x_max = self.x_max + self.x_step
SNGT_QHENOMENOLOGY_92: (8)                    result = np.arange(self.x_min, x_max, self.x_step)
SNGT_QHENOMENOLOGY_93: (8)                    return result[result <= self.x_max]
SNGT_QHENOMENOLOGY_94: (4)                def add_ticks(self) -> None:
SNGT_QHENOMENOLOGY_95: (8)                    ticks = VGroup()
SNGT_QHENOMENOLOGY_96: (8)                    for x in self.get_tick_range():
SNGT_QHENOMENOLOGY_97: (12)                       size = self.tick_size
SNGT_QHENOMENOLOGY_98: (12)                       if np.isclose(self.big_tick_numbers, x).any():
```

```
SNGT_QHENOMENOLOGY_99: (16)                             size *= self.longer_tick_multiple
SNGT_QHENOMENOLOGY_100: (12)                        ticks.add(self.get_tick(x, size))
SNGT_QHENOMENOLOGY_101: (8)                     self.add(ticks)
SNGT_QHENOMENOLOGY_102: (8)                     self.ticks = ticks
SNGT_QHENOMENOLOGY_103: (4)                 def get_tick(self, x: float, size: float | None = None)
-> Line:
SNGT_QHENOMENOLOGY_104: (8)                     if size is None:
SNGT_QHENOMENOLOGY_105: (12)                        size = self.tick_size
SNGT_QHENOMENOLOGY_106: (8)                     result = Line(size * DOWN, size * UP)
SNGT_QHENOMENOLOGY_107: (8)                     result.rotate(self.get_angle())
SNGT_QHENOMENOLOGY_108: (8)                     result.move_to(self.number_to_point(x))
SNGT_QHENOMENOLOGY_109: (8)                     result.match_style(self)
SNGT_QHENOMENOLOGY_110: (8)                     return result
SNGT_QHENOMENOLOGY_111: (4)                 def get_tick_marks(self) -> VGroup:
SNGT_QHENOMENOLOGY_112: (8)                     return self.ticks
SNGT_QHENOMENOLOGY_113: (4)                 def number_to_point(self, number: float | VectN) ->
Vect3 | Vect3Array:
SNGT_QHENOMENOLOGY_114: (8)                     start = self.get_points()[0]
SNGT_QHENOMENOLOGY_115: (8)                     end = self.get_points()[-1]
SNGT_QHENOMENOLOGY_116: (8)                     alpha = (number - self.x_min) / (self.x_max -
self.x_min)
SNGT_QHENOMENOLOGY_117: (8)                     return outer_interpolate(start, end, alpha)
SNGT_QHENOMENOLOGY_118: (4)                 def point_to_number(self, point: Vect3 | Vect3Array) ->
float | VectN:
SNGT_QHENOMENOLOGY_119: (8)                     start = self.get_points()[0]
SNGT_QHENOMENOLOGY_120: (8)                     end = self.get_points()[-1]
SNGT_QHENOMENOLOGY_121: (8)                     vect = end - start
SNGT_QHENOMENOLOGY_122: (8)                     proportion = fdiv(
SNGT_QHENOMENOLOGY_123: (12)                        np.dot(point - start, vect),
SNGT_QHENOMENOLOGY_124: (12)                        np.dot(end - start, vect),
SNGT_QHENOMENOLOGY_125: (8)                     )
SNGT_QHENOMENOLOGY_126: (8)                     return interpolate(self.x_min, self.x_max,
proportion)
SNGT_QHENOMENOLOGY_127: (4)                 def n2p(self, number: float | VectN) -> Vect3 |
Vect3Array:
SNGT_QHENOMENOLOGY_128: (8)                     """Abbreviation for number_to_point"""
SNGT_QHENOMENOLOGY_129: (8)                     return self.number_to_point(number)
SNGT_QHENOMENOLOGY_130: (4)                 def p2n(self, point: Vect3 | Vect3Array) -> float |
VectN:
SNGT_QHENOMENOLOGY_131: (8)                     """Abbreviation for point_to_number"""
SNGT_QHENOMENOLOGY_132: (8)                     return self.point_to_number(point)
SNGT_QHENOMENOLOGY_133: (4)                 def get_unit_size(self) -> float:
SNGT_QHENOMENOLOGY_134: (8)                     return self.get_length() / (self.x_max -
self.x_min)
SNGT_QHENOMENOLOGY_135: (4)                 def get_number_mobject(
SNGT_QHENOMENOLOGY_136: (8)                     self,
SNGT_QHENOMENOLOGY_137: (8)                     x: float,
SNGT_QHENOMENOLOGY_138: (8)                     direction: Vect3 | None = None,
SNGT_QHENOMENOLOGY_139: (8)                     buff: float | None = None,
SNGT_QHENOMENOLOGY_140: (8)                     unit: float = 1.0,
SNGT_QHENOMENOLOGY_141: (8)                     unit_tex: str = "",
SNGT_QHENOMENOLOGY_142: (8)                     **number_config
SNGT_QHENOMENOLOGY_143: (4)                 ) -> DecimalNumber:
SNGT_QHENOMENOLOGY_144: (8)                     number_config = merge_dicts_recursively(
SNGT_QHENOMENOLOGY_145: (12)                        self.decimal_number_config, number_config,
SNGT_QHENOMENOLOGY_146: (8)                     )
SNGT_QHENOMENOLOGY_147: (8)                     if direction is None:
SNGT_QHENOMENOLOGY_148: (12)                        direction = self.line_to_number_direction
SNGT_QHENOMENOLOGY_149: (8)                     if buff is None:
SNGT_QHENOMENOLOGY_150: (12)                        buff = self.line_to_number_buff
SNGT_QHENOMENOLOGY_151: (8)                     if unit_tex:
SNGT_QHENOMENOLOGY_152: (12)                        number_config["unit"] = unit_tex
SNGT_QHENOMENOLOGY_153: (8)                     num_mob = DecimalNumber(x / unit, **number_config)
SNGT_QHENOMENOLOGY_154: (8)                     num_mob.next_to(
SNGT_QHENOMENOLOGY_155: (12)                        self.number_to_point(x),
SNGT_QHENOMENOLOGY_156: (12)                        direction=direction,
SNGT_QHENOMENOLOGY_157: (12)                        buff=buff
SNGT_QHENOMENOLOGY_158: (8)                     )
SNGT_QHENOMENOLOGY_159: (8)                     if x < 0 and direction[0] == 0:
```

```
SNGT_QHENOMENOLOGY_160: (12)                         num_mob.shift(num_mob[0].get_width() * LEFT /
2)
SNGT_QHENOMENOLOGY_161: (8)                     if x == unit and unit_tex:
SNGT_QHENOMENOLOGY_162: (12)                         center = num_mob.get_center()
SNGT_QHENOMENOLOGY_163: (12)                         num_mob.remove(num_mob[0])
SNGT_QHENOMENOLOGY_164: (12)                         num_mob.move_to(center)
SNGT_QHENOMENOLOGY_165: (8)                     return num_mob
SNGT_QHENOMENOLOGY_166: (4)                 def add_numbers(
SNGT_QHENOMENOLOGY_167: (8)                     self,
SNGT_QHENOMENOLOGY_168: (8)                     x_values: Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_169: (8)                     excluding: Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_170: (8)                     font_size: int = 24,
SNGT_QHENOMENOLOGY_171: (8)                     **kwargs
SNGT_QHENOMENOLOGY_172: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_173: (8)                     if x_values is None:
SNGT_QHENOMENOLOGY_174: (12)                         x_values = self.get_tick_range()
SNGT_QHENOMENOLOGY_175: (8)                     kwargs["font_size"] = font_size
SNGT_QHENOMENOLOGY_176: (8)                     if excluding is None:
SNGT_QHENOMENOLOGY_177: (12)                         excluding = self.numbers_to_exclude
SNGT_QHENOMENOLOGY_178: (8)                     numbers = VGroup()
SNGT_QHENOMENOLOGY_179: (8)                     for x in x_values:
SNGT_QHENOMENOLOGY_180: (12)                         if excluding is not None and x in excluding:
SNGT_QHENOMENOLOGY_181: (16)                             continue
SNGT_QHENOMENOLOGY_182: (12)                         numbers.add(self.get_number_mobject(x,
**kwargs))
SNGT_QHENOMENOLOGY_183: (8)                     self.add(numbers)
SNGT_QHENOMENOLOGY_184: (8)                     self.numbers = numbers
SNGT_QHENOMENOLOGY_185: (8)                     return numbers
SNGT_QHENOMENOLOGY_186: (0)             class UnitInterval(NumberLine):
SNGT_QHENOMENOLOGY_187: (4)                 def __init__(
SNGT_QHENOMENOLOGY_188: (8)                     self,
SNGT_QHENOMENOLOGY_189: (8)                     x_range: RangeSpecifier = (0, 1, 0.1),
SNGT_QHENOMENOLOGY_190: (8)                     unit_size: float = 10,
SNGT_QHENOMENOLOGY_191: (8)                     big_tick_numbers: list[float] = [0, 1],
SNGT_QHENOMENOLOGY_192: (8)                     decimal_number_config: dict = dict(
SNGT_QHENOMENOLOGY_193: (12)                         num_decimal_places=1,
SNGT_QHENOMENOLOGY_194: (8)                     )
SNGT_QHENOMENOLOGY_195: (4)                 ):
SNGT_QHENOMENOLOGY_196: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_197: (12)                         x_range=x_range,
SNGT_QHENOMENOLOGY_198: (12)                         unit_size=unit_size,
SNGT_QHENOMENOLOGY_199: (12)                         big_tick_numbers=big_tick_numbers,
SNGT_QHENOMENOLOGY_200: (12)                         decimal_number_config=decimal_number_config,
SNGT_QHENOMENOLOGY_201: (8)                     )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 39 - camera_frame.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import math
SNGT_QHENOMENOLOGY_3: (0)             import warnings
SNGT_QHENOMENOLOGY_4: (0)             import numpy as np
SNGT_QHENOMENOLOGY_5: (0)             from scipy.spatial.transform import Rotation
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.constants import DEG, RADIANS
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.constants import FRAME_SHAPE
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.constants import DOWN, LEFT, ORIGIN, OUT,
RIGHT, UP
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.constants import PI
SNGT_QHENOMENOLOGY_10: (0)            from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_11: (0)            from manimlib.utils.space_ops import normalize
SNGT_QHENOMENOLOGY_12: (0)            from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_13: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_14: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_15: (4)                from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_16: (0)            class CameraFrame(Mobject):
SNGT_QHENOMENOLOGY_17: (4)                def __init__(
SNGT_QHENOMENOLOGY_18: (8)                    self,
SNGT_QHENOMENOLOGY_19: (8)                    frame_shape: tuple[float, float] = FRAME_SHAPE,
```

```
SNGT_QHENOMENOLOGY_20: (8)                      center_point: Vect3 = ORIGIN,
SNGT_QHENOMENOLOGY_21: (8)                      fovy: float = 45 * DEG,
SNGT_QHENOMENOLOGY_22: (8)                      euler_axes: str = "zxz",
SNGT_QHENOMENOLOGY_23: (8)                      z_index=-1,
SNGT_QHENOMENOLOGY_24: (8)                      **kwargs,
SNGT_QHENOMENOLOGY_25: (4)                  ):
SNGT_QHENOMENOLOGY_26: (8)                      super().__init__(z_index=z_index, **kwargs)
SNGT_QHENOMENOLOGY_27: (8)                      self.uniforms["orientation"] =
Rotation.identity().as_quat()
SNGT_QHENOMENOLOGY_28: (8)                      self.uniforms["fovy"] = fovy
SNGT_QHENOMENOLOGY_29: (8)                      self.default_orientation = Rotation.identity()
SNGT_QHENOMENOLOGY_30: (8)                      self.view_matrix = np.identity(4)
SNGT_QHENOMENOLOGY_31: (8)                      self.id4x4 = np.identity(4)
SNGT_QHENOMENOLOGY_32: (8)                      self.camera_location = OUT  # This will be updated
by set_points
SNGT_QHENOMENOLOGY_33: (8)                      self.euler_axes = euler_axes
SNGT_QHENOMENOLOGY_34: (8)                      self.set_points(np.array([ORIGIN, LEFT, RIGHT,
DOWN, UP]))
SNGT_QHENOMENOLOGY_35: (8)                      self.set_width(frame_shape[0], stretch=True)
SNGT_QHENOMENOLOGY_36: (8)                      self.set_height(frame_shape[1], stretch=True)
SNGT_QHENOMENOLOGY_37: (8)                      self.move_to(center_point)
SNGT_QHENOMENOLOGY_38: (4)                  def set_orientation(self, rotation: Rotation):
SNGT_QHENOMENOLOGY_39: (8)                      self.uniforms["orientation"][:] =
rotation.as_quat()
SNGT_QHENOMENOLOGY_40: (8)                      return self
SNGT_QHENOMENOLOGY_41: (4)                  def get_orientation(self):
SNGT_QHENOMENOLOGY_42: (8)                      return
Rotation.from_quat(self.uniforms["orientation"])
SNGT_QHENOMENOLOGY_43: (4)                  def make_orientation_default(self):
SNGT_QHENOMENOLOGY_44: (8)                      self.default_orientation = self.get_orientation()
SNGT_QHENOMENOLOGY_45: (8)                      return self
SNGT_QHENOMENOLOGY_46: (4)                  def to_default_state(self):
SNGT_QHENOMENOLOGY_47: (8)                      self.set_shape(*FRAME_SHAPE)
SNGT_QHENOMENOLOGY_48: (8)                      self.center()
SNGT_QHENOMENOLOGY_49: (8)                      self.set_orientation(self.default_orientation)
SNGT_QHENOMENOLOGY_50: (8)                      return self
SNGT_QHENOMENOLOGY_51: (4)                  def get_euler_angles(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_52: (8)                      orientation = self.get_orientation()
SNGT_QHENOMENOLOGY_53: (8)                      if np.isclose(orientation.as_quat(), [0, 0, 0,
1]).all():
SNGT_QHENOMENOLOGY_54: (12)                         return np.zeros(3)
SNGT_QHENOMENOLOGY_55: (8)                      with warnings.catch_warnings():
SNGT_QHENOMENOLOGY_56: (12)                         warnings.simplefilter('ignore', UserWarning)  #
Ignore UserWarnings
SNGT_QHENOMENOLOGY_57: (12)                         angles = orientation.as_euler(self.euler_axes)
[::-1]
SNGT_QHENOMENOLOGY_58: (8)                      if self.euler_axes == "zxz":
SNGT_QHENOMENOLOGY_59: (12)                         if np.isclose(angles[1], 0, atol=1e-2):
SNGT_QHENOMENOLOGY_60: (16)                             angles[0] = angles[0] + angles[2]
SNGT_QHENOMENOLOGY_61: (16)                             angles[2] = 0
SNGT_QHENOMENOLOGY_62: (12)                         if np.isclose(angles[1], PI, atol=1e-2):
SNGT_QHENOMENOLOGY_63: (16)                             angles[0] = angles[0] - angles[2]
SNGT_QHENOMENOLOGY_64: (16)                             angles[2] = 0
SNGT_QHENOMENOLOGY_65: (8)                      return angles
SNGT_QHENOMENOLOGY_66: (4)                  def get_theta(self):
SNGT_QHENOMENOLOGY_67: (8)                      return self.get_euler_angles()[0]
SNGT_QHENOMENOLOGY_68: (4)                  def get_phi(self):
SNGT_QHENOMENOLOGY_69: (8)                      return self.get_euler_angles()[1]
SNGT_QHENOMENOLOGY_70: (4)                  def get_gamma(self):
SNGT_QHENOMENOLOGY_71: (8)                      return self.get_euler_angles()[2]
SNGT_QHENOMENOLOGY_72: (4)                  def get_scale(self):
SNGT_QHENOMENOLOGY_73: (8)                      return self.get_height() / FRAME_SHAPE[1]
SNGT_QHENOMENOLOGY_74: (4)                  def get_inverse_camera_rotation_matrix(self):
SNGT_QHENOMENOLOGY_75: (8)                      return self.get_orientation().as_matrix().T
SNGT_QHENOMENOLOGY_76: (4)                  def get_view_matrix(self, refresh=False):
SNGT_QHENOMENOLOGY_77: (8)                      """
SNGT_QHENOMENOLOGY_78: (8)                      Returns a 4x4 for the affine transformation mapping
a point
SNGT_QHENOMENOLOGY_79: (8)                      into the camera's internal coordinate system
```

```
SNGT_QHENOMENOLOGY_80: (8)                      """
SNGT_QHENOMENOLOGY_81: (8)                      if self._data_has_changed:
SNGT_QHENOMENOLOGY_82: (12)                         shift = self.id4x4.copy()
SNGT_QHENOMENOLOGY_83: (12)                         rotation = self.id4x4.copy()
SNGT_QHENOMENOLOGY_84: (12)                         scale = self.get_scale()
SNGT_QHENOMENOLOGY_85: (12)                         shift[:3, 3] = -self.get_center()
SNGT_QHENOMENOLOGY_86: (12)                         rotation[:3, :3] =
self.get_inverse_camera_rotation_matrix()
SNGT_QHENOMENOLOGY_87: (12)                         np.dot(rotation, shift, out=self.view_matrix)
SNGT_QHENOMENOLOGY_88: (12)                         if scale > 0:
SNGT_QHENOMENOLOGY_89: (16)                             self.view_matrix[:3, :4] /= scale
SNGT_QHENOMENOLOGY_90: (8)                      return self.view_matrix
SNGT_QHENOMENOLOGY_91: (4)                  def get_inv_view_matrix(self):
SNGT_QHENOMENOLOGY_92: (8)                      return np.linalg.inv(self.get_view_matrix())
SNGT_QHENOMENOLOGY_93: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_94: (4)                  def interpolate(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_95: (8)                      super().interpolate(*args, **kwargs)
SNGT_QHENOMENOLOGY_96: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_97: (4)                  def rotate(self, angle: float, axis: np.ndarray = OUT,
**kwargs):
SNGT_QHENOMENOLOGY_98: (8)                      rot = Rotation.from_rotvec(angle * normalize(axis))
SNGT_QHENOMENOLOGY_99: (8)                      self.set_orientation(rot * self.get_orientation())
SNGT_QHENOMENOLOGY_100: (8)                     return self
SNGT_QHENOMENOLOGY_101: (4)                 def set_euler_angles(
SNGT_QHENOMENOLOGY_102: (8)                     self,
SNGT_QHENOMENOLOGY_103: (8)                     theta: float | None = None,
SNGT_QHENOMENOLOGY_104: (8)                     phi: float | None = None,
SNGT_QHENOMENOLOGY_105: (8)                     gamma: float | None = None,
SNGT_QHENOMENOLOGY_106: (8)                     units: float = RADIANS
SNGT_QHENOMENOLOGY_107: (4)                 ):
SNGT_QHENOMENOLOGY_108: (8)                     eulers = self.get_euler_angles()  # theta, phi,
gamma
SNGT_QHENOMENOLOGY_109: (8)                     for i, var in enumerate([theta, phi, gamma]):
SNGT_QHENOMENOLOGY_110: (12)                        if var is not None:
SNGT_QHENOMENOLOGY_111: (16)                            eulers[i] = var * units
SNGT_QHENOMENOLOGY_112: (8)                     if all(eulers == 0):
SNGT_QHENOMENOLOGY_113: (12)                        rot = Rotation.identity()
SNGT_QHENOMENOLOGY_114: (8)                     else:
SNGT_QHENOMENOLOGY_115: (12)                        rot = Rotation.from_euler(self.euler_axes,
eulers[::-1])
SNGT_QHENOMENOLOGY_116: (8)                     self.set_orientation(rot)
SNGT_QHENOMENOLOGY_117: (8)                     return self
SNGT_QHENOMENOLOGY_118: (4)                 def increment_euler_angles(
SNGT_QHENOMENOLOGY_119: (8)                     self,
SNGT_QHENOMENOLOGY_120: (8)                     dtheta: float = 0,
SNGT_QHENOMENOLOGY_121: (8)                     dphi: float = 0,
SNGT_QHENOMENOLOGY_122: (8)                     dgamma: float = 0,
SNGT_QHENOMENOLOGY_123: (8)                     units: float = RADIANS
SNGT_QHENOMENOLOGY_124: (4)                 ):
SNGT_QHENOMENOLOGY_125: (8)                     angles = self.get_euler_angles()
SNGT_QHENOMENOLOGY_126: (8)                     new_angles = angles + np.array([dtheta, dphi,
dgamma]) * units
SNGT_QHENOMENOLOGY_127: (8)                     if self.euler_axes == "zxz":
SNGT_QHENOMENOLOGY_128: (12)                        new_angles[1] = clip(new_angles[1], 0, PI)
SNGT_QHENOMENOLOGY_129: (8)                     elif self.euler_axes == "zxy":
SNGT_QHENOMENOLOGY_130: (12)                        new_angles[1] = clip(new_angles[1], -PI / 2, PI
/ 2)
SNGT_QHENOMENOLOGY_131: (8)                     new_rot = Rotation.from_euler(self.euler_axes,
new_angles[::-1])
SNGT_QHENOMENOLOGY_132: (8)                     self.set_orientation(new_rot)
SNGT_QHENOMENOLOGY_133: (8)                     return self
SNGT_QHENOMENOLOGY_134: (4)                 def set_euler_axes(self, seq: str):
SNGT_QHENOMENOLOGY_135: (8)                     self.euler_axes = seq
SNGT_QHENOMENOLOGY_136: (4)                 def reorient(
SNGT_QHENOMENOLOGY_137: (8)                     self,
SNGT_QHENOMENOLOGY_138: (8)                     theta_degrees: float | None = None,
SNGT_QHENOMENOLOGY_139: (8)                     phi_degrees: float | None = None,
SNGT_QHENOMENOLOGY_140: (8)                     gamma_degrees: float | None = None,
SNGT_QHENOMENOLOGY_141: (8)                     center: Vect3 | tuple[float, float, float] | None =
```

```
None,
SNGT_QHENOMENOLOGY_142: (8)                     height: float | None = None
SNGT_QHENOMENOLOGY_143: (4)                 ):
SNGT_QHENOMENOLOGY_144: (8)                     """
SNGT_QHENOMENOLOGY_145: (8)                     Shortcut for set_euler_angles, defaulting to taking
SNGT_QHENOMENOLOGY_146: (8)                     in angles in degrees
SNGT_QHENOMENOLOGY_147: (8)                     """
SNGT_QHENOMENOLOGY_148: (8)                     self.set_euler_angles(theta_degrees, phi_degrees,
gamma_degrees, units=DEG)
SNGT_QHENOMENOLOGY_149: (8)                     if center is not None:
SNGT_QHENOMENOLOGY_150: (12)                        self.move_to(np.array(center))
SNGT_QHENOMENOLOGY_151: (8)                     if height is not None:
SNGT_QHENOMENOLOGY_152: (12)                        self.set_height(height)
SNGT_QHENOMENOLOGY_153: (8)                     return self
SNGT_QHENOMENOLOGY_154: (4)                 def set_theta(self, theta: float):
SNGT_QHENOMENOLOGY_155: (8)                     return self.set_euler_angles(theta=theta)
SNGT_QHENOMENOLOGY_156: (4)                 def set_phi(self, phi: float):
SNGT_QHENOMENOLOGY_157: (8)                     return self.set_euler_angles(phi=phi)
SNGT_QHENOMENOLOGY_158: (4)                 def set_gamma(self, gamma: float):
SNGT_QHENOMENOLOGY_159: (8)                     return self.set_euler_angles(gamma=gamma)
SNGT_QHENOMENOLOGY_160: (4)                 def increment_theta(self, dtheta: float,
units=RADIANS):
SNGT_QHENOMENOLOGY_161: (8)                     self.increment_euler_angles(dtheta=dtheta,
units=units)
SNGT_QHENOMENOLOGY_162: (8)                     return self
SNGT_QHENOMENOLOGY_163: (4)                 def increment_phi(self, dphi: float, units=RADIANS):
SNGT_QHENOMENOLOGY_164: (8)                     self.increment_euler_angles(dphi=dphi, units=units)
SNGT_QHENOMENOLOGY_165: (8)                     return self
SNGT_QHENOMENOLOGY_166: (4)                 def increment_gamma(self, dgamma: float,
units=RADIANS):
SNGT_QHENOMENOLOGY_167: (8)                     self.increment_euler_angles(dgamma=dgamma,
units=units)
SNGT_QHENOMENOLOGY_168: (8)                     return self
SNGT_QHENOMENOLOGY_169: (4)                 def add_ambient_rotation(self, angular_speed=1 * DEG):
SNGT_QHENOMENOLOGY_170: (8)                     self.add_updater(lambda m, dt:
m.increment_theta(angular_speed * dt))
SNGT_QHENOMENOLOGY_171: (8)                     return self
SNGT_QHENOMENOLOGY_172: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_173: (4)                 def set_focal_distance(self, focal_distance: float):
SNGT_QHENOMENOLOGY_174: (8)                     self.uniforms["fovy"] = 2 * math.atan(0.5 *
self.get_height() / focal_distance)
SNGT_QHENOMENOLOGY_175: (8)                     return self
SNGT_QHENOMENOLOGY_176: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_177: (4)                 def set_field_of_view(self, field_of_view: float):
SNGT_QHENOMENOLOGY_178: (8)                     self.uniforms["fovy"] = field_of_view
SNGT_QHENOMENOLOGY_179: (8)                     return self
SNGT_QHENOMENOLOGY_180: (4)                 def get_shape(self):
SNGT_QHENOMENOLOGY_181: (8)                     return (self.get_width(), self.get_height())
SNGT_QHENOMENOLOGY_182: (4)                 def get_aspect_ratio(self):
SNGT_QHENOMENOLOGY_183: (8)                     width, height = self.get_shape()
SNGT_QHENOMENOLOGY_184: (8)                     return width / height
SNGT_QHENOMENOLOGY_185: (4)                 def get_center(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_186: (8)                     return self.get_points()[0]
SNGT_QHENOMENOLOGY_187: (4)                 def get_width(self) -> float:
SNGT_QHENOMENOLOGY_188: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_189: (8)                     return points[2, 0] - points[1, 0]
SNGT_QHENOMENOLOGY_190: (4)                 def get_height(self) -> float:
SNGT_QHENOMENOLOGY_191: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_192: (8)                     return points[4, 1] - points[3, 1]
SNGT_QHENOMENOLOGY_193: (4)                 def get_focal_distance(self) -> float:
SNGT_QHENOMENOLOGY_194: (8)                     return 0.5 * self.get_height() / math.tan(0.5 *
self.uniforms["fovy"])
SNGT_QHENOMENOLOGY_195: (4)                 def get_field_of_view(self) -> float:
SNGT_QHENOMENOLOGY_196: (8)                     return self.uniforms["fovy"]
SNGT_QHENOMENOLOGY_197: (4)                 def get_implied_camera_location(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_198: (8)                     if self._data_has_changed:
SNGT_QHENOMENOLOGY_199: (12)                        to_camera =
self.get_inverse_camera_rotation_matrix()[2]
SNGT_QHENOMENOLOGY_200: (12)                        dist = self.get_focal_distance()
```

```
SNGT_QHENOMENOLOGY_201: (12)                         self.camera_location = self.get_center() + dist
* to_camera
SNGT_QHENOMENOLOGY_202: (8)                     return self.camera_location
SNGT_QHENOMENOLOGY_203: (4)                 def to_fixed_frame_point(self, point: Vect3, relative:
bool = False):
SNGT_QHENOMENOLOGY_204: (8)                     view = self.get_view_matrix()
SNGT_QHENOMENOLOGY_205: (8)                     point4d = [*point, 0 if relative else 1]
SNGT_QHENOMENOLOGY_206: (8)                     return np.dot(point4d, view.T)[:3]
SNGT_QHENOMENOLOGY_207: (4)                 def from_fixed_frame_point(self, point: Vect3,
relative: bool = False):
SNGT_QHENOMENOLOGY_208: (8)                     inv_view = self.get_inv_view_matrix()
SNGT_QHENOMENOLOGY_209: (8)                     point4d = [*point, 0 if relative else 1]
SNGT_QHENOMENOLOGY_210: (8)                     return np.dot(point4d, inv_view.T)[:3]
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 40 - extract_scene.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import copy
SNGT_QHENOMENOLOGY_3: (0)             import inspect
SNGT_QHENOMENOLOGY_4: (0)             import sys
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.module_loader import ModuleLoader
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.logger import log
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.scene.interactive_scene import
InteractiveScene
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_10: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_11: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_12: (4)                Module = importlib.util.types.ModuleType
SNGT_QHENOMENOLOGY_13: (4)                from typing import Optional
SNGT_QHENOMENOLOGY_14: (4)                from addict import Dict
SNGT_QHENOMENOLOGY_15: (0)            class BlankScene(InteractiveScene):
SNGT_QHENOMENOLOGY_16: (4)                def construct(self):
SNGT_QHENOMENOLOGY_17: (8)                    exec(manim_config.universal_import_line)
SNGT_QHENOMENOLOGY_18: (8)                    self.embed()
SNGT_QHENOMENOLOGY_19: (0)            def is_child_scene(obj, module):
SNGT_QHENOMENOLOGY_20: (4)                if not inspect.isclass(obj):
SNGT_QHENOMENOLOGY_21: (8)                    return False
SNGT_QHENOMENOLOGY_22: (4)                if not issubclass(obj, Scene):
SNGT_QHENOMENOLOGY_23: (8)                    return False
SNGT_QHENOMENOLOGY_24: (4)                if obj == Scene:
SNGT_QHENOMENOLOGY_25: (8)                    return False
SNGT_QHENOMENOLOGY_26: (4)                if not obj.__module__.startswith(module.__name__):
SNGT_QHENOMENOLOGY_27: (8)                    return False
SNGT_QHENOMENOLOGY_28: (4)                return True
SNGT_QHENOMENOLOGY_29: (0)            def prompt_user_for_choice(scene_classes):
SNGT_QHENOMENOLOGY_30: (4)                name_to_class = {}
SNGT_QHENOMENOLOGY_31: (4)                max_digits = len(str(len(scene_classes)))
SNGT_QHENOMENOLOGY_32: (4)                for idx, scene_class in enumerate(scene_classes,
start=1):
SNGT_QHENOMENOLOGY_33: (8)                    name = scene_class.__name__
SNGT_QHENOMENOLOGY_34: (8)                    print(f"{str(idx).zfill(max_digits)}: {name}")
SNGT_QHENOMENOLOGY_35: (8)                    name_to_class[name] = scene_class
SNGT_QHENOMENOLOGY_36: (4)                try:
SNGT_QHENOMENOLOGY_37: (8)                    user_input = input("\nSelect which scene to render
(by name or number): ")
SNGT_QHENOMENOLOGY_38: (8)                    return [
SNGT_QHENOMENOLOGY_39: (12)                       name_to_class[split_str] if not
split_str.isnumeric() else scene_classes[int(split_str) - 1]
SNGT_QHENOMENOLOGY_40: (12)                       for split_str in user_input.replace(" ",
"").split(",")
SNGT_QHENOMENOLOGY_41: (8)                    ]
SNGT_QHENOMENOLOGY_42: (4)                except IndexError:
SNGT_QHENOMENOLOGY_43: (8)                    log.error("Invalid scene number")
SNGT_QHENOMENOLOGY_44: (8)                    sys.exit(2)
SNGT_QHENOMENOLOGY_45: (4)                except KeyError:
SNGT_QHENOMENOLOGY_46: (8)                    log.error("Invalid scene name")
```

```
SNGT_QHENOMENOLOGY_47: (8)                    sys.exit(2)
SNGT_QHENOMENOLOGY_48: (4)                except EOFError:
SNGT_QHENOMENOLOGY_49: (8)                    sys.exit(1)
SNGT_QHENOMENOLOGY_50: (0)            def compute_total_frames(scene_class, scene_config):
SNGT_QHENOMENOLOGY_51: (4)                """
SNGT_QHENOMENOLOGY_52: (4)                When a scene is being written to file, a copy of the
scene is run with
SNGT_QHENOMENOLOGY_53: (4)                skip_animations set to true so as to count how many
frames it will require.
SNGT_QHENOMENOLOGY_54: (4)                This allows for a total progress bar on rendering, and
also allows runtime
SNGT_QHENOMENOLOGY_55: (4)                errors to be exposed preemptively for long running
scenes.
SNGT_QHENOMENOLOGY_56: (4)                """
SNGT_QHENOMENOLOGY_57: (4)                pre_config = copy.deepcopy(scene_config)
SNGT_QHENOMENOLOGY_58: (4)                pre_config["file_writer_config"]["write_to_movie"] =
False
SNGT_QHENOMENOLOGY_59: (4)                pre_config["file_writer_config"]["save_last_frame"] =
False
SNGT_QHENOMENOLOGY_60: (4)                pre_config["file_writer_config"]["quiet"] = True
SNGT_QHENOMENOLOGY_61: (4)                pre_config["skip_animations"] = True
SNGT_QHENOMENOLOGY_62: (4)                pre_scene = scene_class(**pre_config)
SNGT_QHENOMENOLOGY_63: (4)                pre_scene.run()
SNGT_QHENOMENOLOGY_64: (4)                total_time = pre_scene.time - pre_scene.skip_time
SNGT_QHENOMENOLOGY_65: (4)                return int(total_time * manim_config.camera.fps)
SNGT_QHENOMENOLOGY_66: (0)            def scene_from_class(scene_class, scene_config: Dict,
run_config: Dict):
SNGT_QHENOMENOLOGY_67: (4)                fw_config = manim_config.file_writer
SNGT_QHENOMENOLOGY_68: (4)                if fw_config.write_to_movie and run_config.prerun:
SNGT_QHENOMENOLOGY_69: (8)                    scene_config.file_writer_config.total_frames =
compute_total_frames(scene_class, scene_config)
SNGT_QHENOMENOLOGY_70: (4)                return scene_class(**scene_config)
SNGT_QHENOMENOLOGY_71: (0)            def note_missing_scenes(arg_names, module_names):
SNGT_QHENOMENOLOGY_72: (4)                for name in arg_names:
SNGT_QHENOMENOLOGY_73: (8)                    if name not in module_names:
SNGT_QHENOMENOLOGY_74: (12)                       log.error(f"No scene named {name} found")
SNGT_QHENOMENOLOGY_75: (0)            def get_scenes_to_render(all_scene_classes: list,
scene_config: Dict, run_config: Dict):
SNGT_QHENOMENOLOGY_76: (4)                if run_config["write_all"] or len(all_scene_classes) ==
1:
SNGT_QHENOMENOLOGY_77: (8)                    classes_to_run = all_scene_classes
SNGT_QHENOMENOLOGY_78: (4)                else:
SNGT_QHENOMENOLOGY_79: (8)                    name_to_class = {sc.__name__: sc for sc in
all_scene_classes}
SNGT_QHENOMENOLOGY_80: (8)                    classes_to_run = [name_to_class.get(name) for name
in run_config.scene_names]
SNGT_QHENOMENOLOGY_81: (8)                    classes_to_run = list(filter(lambda x: x,
classes_to_run))  # Remove Nones
SNGT_QHENOMENOLOGY_82: (8)                    note_missing_scenes(run_config.scene_names,
name_to_class.keys())
SNGT_QHENOMENOLOGY_83: (4)                if len(classes_to_run) == 0:
SNGT_QHENOMENOLOGY_84: (8)                    classes_to_run =
prompt_user_for_choice(all_scene_classes)
SNGT_QHENOMENOLOGY_85: (4)                return [
SNGT_QHENOMENOLOGY_86: (8)                    scene_from_class(scene_class, scene_config,
run_config)
SNGT_QHENOMENOLOGY_87: (8)                    for scene_class in classes_to_run
SNGT_QHENOMENOLOGY_88: (4)                ]
SNGT_QHENOMENOLOGY_89: (0)            def get_scene_classes(module: Optional[Module]):
SNGT_QHENOMENOLOGY_90: (4)                if module is None:
SNGT_QHENOMENOLOGY_91: (8)                    return [BlankScene]
SNGT_QHENOMENOLOGY_92: (4)                if hasattr(module, "SCENES_IN_ORDER"):
SNGT_QHENOMENOLOGY_93: (8)                    return module.SCENES_IN_ORDER
SNGT_QHENOMENOLOGY_94: (4)                else:
SNGT_QHENOMENOLOGY_95: (8)                    return [
SNGT_QHENOMENOLOGY_96: (12)                       member[1]
SNGT_QHENOMENOLOGY_97: (12)                       for member in inspect.getmembers(
SNGT_QHENOMENOLOGY_98: (16)                           module,
SNGT_QHENOMENOLOGY_99: (16)                           lambda x: is_child_scene(x, module)
```

```
SNGT_QHENOMENOLOGY_100: (12)                        )
SNGT_QHENOMENOLOGY_101: (8)                    ]
SNGT_QHENOMENOLOGY_102: (0)            def get_indent(code_lines: list[str], line_number: int) ->
str:
SNGT_QHENOMENOLOGY_103: (4)                """
SNGT_QHENOMENOLOGY_104: (4)                Find the indent associated with a given line of python
code,
SNGT_QHENOMENOLOGY_105: (4)                as a string of spaces
SNGT_QHENOMENOLOGY_106: (4)                """
SNGT_QHENOMENOLOGY_107: (4)                try:
SNGT_QHENOMENOLOGY_108: (8)                    line = next(filter(lambda line: line.strip(),
code_lines[line_number - 1::-1]))
SNGT_QHENOMENOLOGY_109: (4)                except StopIteration:
SNGT_QHENOMENOLOGY_110: (8)                    return ""
SNGT_QHENOMENOLOGY_111: (4)                n_spaces = len(line) - len(line.lstrip())
SNGT_QHENOMENOLOGY_112: (4)                if line.endswith(":"):
SNGT_QHENOMENOLOGY_113: (8)                    n_spaces += 4
SNGT_QHENOMENOLOGY_114: (4)                return n_spaces * " "
SNGT_QHENOMENOLOGY_115: (0)            def insert_embed_line_to_module(module: Module,
line_number: int):
SNGT_QHENOMENOLOGY_116: (4)                """
SNGT_QHENOMENOLOGY_117: (4)                This is hacky, but convenient. When user includes the
argument "-e", it will try
SNGT_QHENOMENOLOGY_118: (4)                to recreate a file that inserts the line `self.embed()`
into the end of the scene's
SNGT_QHENOMENOLOGY_119: (4)                construct method. If there is an argument passed in, it
will insert the line after
SNGT_QHENOMENOLOGY_120: (4)                the last line in the sourcefile which includes that
string.
SNGT_QHENOMENOLOGY_121: (4)                """
SNGT_QHENOMENOLOGY_122: (4)                lines = inspect.getsource(module).splitlines()
SNGT_QHENOMENOLOGY_123: (4)                indent = get_indent(lines, line_number)
SNGT_QHENOMENOLOGY_124: (4)                lines.insert(line_number, indent + "self.embed()")
SNGT_QHENOMENOLOGY_125: (4)                new_code = "\n".join(lines)
SNGT_QHENOMENOLOGY_126: (4)                code_object = compile(new_code, module.__name__,
'exec')
SNGT_QHENOMENOLOGY_127: (4)                exec(code_object, module.__dict__)
SNGT_QHENOMENOLOGY_128: (0)            def get_module(file_name: Optional[str], embed_line:
Optional[int], is_reload: bool = False) -> Module:
SNGT_QHENOMENOLOGY_129: (4)                module = ModuleLoader.get_module(file_name, is_reload)
SNGT_QHENOMENOLOGY_130: (4)                if embed_line:
SNGT_QHENOMENOLOGY_131: (8)                    insert_embed_line_to_module(module, embed_line)
SNGT_QHENOMENOLOGY_132: (4)                return module
SNGT_QHENOMENOLOGY_133: (0)            def main(scene_config: Dict, run_config: Dict):
SNGT_QHENOMENOLOGY_134: (4)                module = get_module(run_config.file_name,
run_config.embed_line, run_config.is_reload)
SNGT_QHENOMENOLOGY_135: (4)                all_scene_classes = get_scene_classes(module)
SNGT_QHENOMENOLOGY_136: (4)                scenes = get_scenes_to_render(all_scene_classes,
scene_config, run_config)
SNGT_QHENOMENOLOGY_137: (4)                if len(scenes) == 0:
SNGT_QHENOMENOLOGY_138: (8)                    print("No scenes found to run")
SNGT_QHENOMENOLOGY_139: (4)                return scenes
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 41 - event_listner.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_3: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_4: (4)                  from typing import Callable
SNGT_QHENOMENOLOGY_5: (4)                  from manimlib.event_handler.event_type import EventType
SNGT_QHENOMENOLOGY_6: (4)                  from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_7: (0)              class EventListener(object):
SNGT_QHENOMENOLOGY_8: (4)                  def __init__(
SNGT_QHENOMENOLOGY_9: (8)                      self,
SNGT_QHENOMENOLOGY_10: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_11: (8)                     event_type: EventType,
SNGT_QHENOMENOLOGY_12: (8)                     event_callback: Callable[[Mobject, dict[str]]]
```

```
SNGT_QHENOMENOLOGY_13: (4)                 ):
SNGT_QHENOMENOLOGY_14: (8)                     self.mobject = mobject
SNGT_QHENOMENOLOGY_15: (8)                     self.event_type = event_type
SNGT_QHENOMENOLOGY_16: (8)                     self.callback = event_callback
SNGT_QHENOMENOLOGY_17: (4)                 def __eq__(self, o: object) -> bool:
SNGT_QHENOMENOLOGY_18: (8)                     return_val = False
SNGT_QHENOMENOLOGY_19: (8)                     try:
SNGT_QHENOMENOLOGY_20: (12)                        return_val = self.callback == o.callback \
SNGT_QHENOMENOLOGY_21: (16)                            and self.mobject == o.mobject \
SNGT_QHENOMENOLOGY_22: (16)                            and self.event_type == o.event_type
SNGT_QHENOMENOLOGY_23: (8)                     except:
SNGT_QHENOMENOLOGY_24: (12)                        pass
SNGT_QHENOMENOLOGY_25: (8)                     return return_val
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 42 - event_dispatcher.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              from manimlib.event_handler.event_listner import
EventListener
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.event_handler.event_type import EventType
SNGT_QHENOMENOLOGY_5: (0)              class EventDispatcher(object):
SNGT_QHENOMENOLOGY_6: (4)                  def __init__(self):
SNGT_QHENOMENOLOGY_7: (8)                      self.event_listners: dict[
SNGT_QHENOMENOLOGY_8: (12)                         EventType, list[EventListener]
SNGT_QHENOMENOLOGY_9: (8)                      ] = {
SNGT_QHENOMENOLOGY_10: (12)                        event_type: []
SNGT_QHENOMENOLOGY_11: (12)                        for event_type in EventType
SNGT_QHENOMENOLOGY_12: (8)                     }
SNGT_QHENOMENOLOGY_13: (8)                     self.mouse_point = np.array((0., 0., 0.))
SNGT_QHENOMENOLOGY_14: (8)                     self.mouse_drag_point = np.array((0., 0., 0.))
SNGT_QHENOMENOLOGY_15: (8)                     self.pressed_keys: set[int] = set()
SNGT_QHENOMENOLOGY_16: (8)                     self.draggable_object_listners: list[EventListener]
= []
SNGT_QHENOMENOLOGY_17: (4)                 def add_listner(self, event_listner: EventListener):
SNGT_QHENOMENOLOGY_18: (8)                     assert isinstance(event_listner, EventListener)
SNGT_QHENOMENOLOGY_19: (8)
self.event_listners[event_listner.event_type].append(event_listner)
SNGT_QHENOMENOLOGY_20: (8)                     return self
SNGT_QHENOMENOLOGY_21: (4)                 def remove_listner(self, event_listner: EventListener):
SNGT_QHENOMENOLOGY_22: (8)                     assert isinstance(event_listner, EventListener)
SNGT_QHENOMENOLOGY_23: (8)                     try:
SNGT_QHENOMENOLOGY_24: (12)                        while event_listner in
self.event_listners[event_listner.event_type]:
SNGT_QHENOMENOLOGY_25: (16)
self.event_listners[event_listner.event_type].remove(event_listner)
SNGT_QHENOMENOLOGY_26: (8)                     except:
SNGT_QHENOMENOLOGY_27: (12)                        pass
SNGT_QHENOMENOLOGY_28: (8)                     return self
SNGT_QHENOMENOLOGY_29: (4)                 def dispatch(self, event_type: EventType,
**event_data):
SNGT_QHENOMENOLOGY_30: (8)                     if event_type == EventType.MouseMotionEvent:
SNGT_QHENOMENOLOGY_31: (12)                        self.mouse_point = event_data["point"]
SNGT_QHENOMENOLOGY_32: (8)                     elif event_type == EventType.MouseDragEvent:
SNGT_QHENOMENOLOGY_33: (12)                        self.mouse_drag_point = event_data["point"]
SNGT_QHENOMENOLOGY_34: (8)                     elif event_type == EventType.KeyPressEvent:
SNGT_QHENOMENOLOGY_35: (12)                        self.pressed_keys.add(event_data["symbol"])  #
Modifiers?
SNGT_QHENOMENOLOGY_36: (8)                     elif event_type == EventType.KeyReleaseEvent:
SNGT_QHENOMENOLOGY_37: (12)
self.pressed_keys.difference_update({event_data["symbol"]})  # Modifiers?
SNGT_QHENOMENOLOGY_38: (8)                     elif event_type == EventType.MousePressEvent:
SNGT_QHENOMENOLOGY_39: (12)                        self.draggable_object_listners = [
SNGT_QHENOMENOLOGY_40: (16)                            listner
SNGT_QHENOMENOLOGY_41: (16)                            for listner in
self.event_listners[EventType.MouseDragEvent]
SNGT_QHENOMENOLOGY_42: (16)                            if
```

```
listner.mobject.is_point_touching(self.mouse_point)
SNGT_QHENOMENOLOGY_43: (12)                         ]
SNGT_QHENOMENOLOGY_44: (8)                     elif event_type == EventType.MouseReleaseEvent:
SNGT_QHENOMENOLOGY_45: (12)                         self.draggable_object_listners = []
SNGT_QHENOMENOLOGY_46: (8)                     propagate_event = None
SNGT_QHENOMENOLOGY_47: (8)                     if event_type == EventType.MouseDragEvent:
SNGT_QHENOMENOLOGY_48: (12)                         for listner in self.draggable_object_listners:
SNGT_QHENOMENOLOGY_49: (16)                             assert isinstance(listner, EventListener)
SNGT_QHENOMENOLOGY_50: (16)                             propagate_event =
listner.callback(listner.mobject, event_data)
SNGT_QHENOMENOLOGY_51: (16)                             if propagate_event is not None and
propagate_event is False:
SNGT_QHENOMENOLOGY_52: (20)                                 return propagate_event
SNGT_QHENOMENOLOGY_53: (8)                     elif event_type.value.startswith('mouse'):
SNGT_QHENOMENOLOGY_54: (12)                         for listner in self.event_listners[event_type]:
SNGT_QHENOMENOLOGY_55: (16)                             if
listner.mobject.is_point_touching(self.mouse_point):
SNGT_QHENOMENOLOGY_56: (20)                                 propagate_event = listner.callback(
SNGT_QHENOMENOLOGY_57: (24)                                     listner.mobject, event_data)
SNGT_QHENOMENOLOGY_58: (20)                                 if propagate_event is not None and
propagate_event is False:
SNGT_QHENOMENOLOGY_59: (24)                                     return propagate_event
SNGT_QHENOMENOLOGY_60: (8)                     elif event_type.value.startswith('key'):
SNGT_QHENOMENOLOGY_61: (12)                         for listner in self.event_listners[event_type]:
SNGT_QHENOMENOLOGY_62: (16)                             propagate_event =
listner.callback(listner.mobject, event_data)
SNGT_QHENOMENOLOGY_63: (16)                             if propagate_event is not None and
propagate_event is False:
SNGT_QHENOMENOLOGY_64: (20)                                 return propagate_event
SNGT_QHENOMENOLOGY_65: (8)                     return propagate_event
SNGT_QHENOMENOLOGY_66: (4)                 def get_listners_count(self) -> int:
SNGT_QHENOMENOLOGY_67: (8)                     return sum([len(value) for key, value in
self.event_listners.items()])
SNGT_QHENOMENOLOGY_68: (4)                 def get_mouse_point(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_69: (8)                     return self.mouse_point
SNGT_QHENOMENOLOGY_70: (4)                 def get_mouse_drag_point(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_71: (8)                     return self.mouse_drag_point
SNGT_QHENOMENOLOGY_72: (4)                 def is_key_pressed(self, symbol: int) -> bool:
SNGT_QHENOMENOLOGY_73: (8)                     return (symbol in self.pressed_keys)
SNGT_QHENOMENOLOGY_74: (4)                 __iadd__ = add_listner
SNGT_QHENOMENOLOGY_75: (4)                 __isub__ = remove_listner
SNGT_QHENOMENOLOGY_76: (4)                 __call__ = dispatch
SNGT_QHENOMENOLOGY_77: (4)                 __len__ = get_listners_count
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 43 - coordinate_systems.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from abc import ABC, abstractmethod
SNGT_QHENOMENOLOGY_3: (0)              import numbers
SNGT_QHENOMENOLOGY_4: (0)              import numpy as np
SNGT_QHENOMENOLOGY_5: (0)              import itertools as it
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.constants import BLACK, BLUE, BLUE_D, BLUE_E,
GREEN, GREY_A, WHITE, RED
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.constants import DEG, PI
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.constants import DL, UL, DOWN, DR, LEFT,
ORIGIN, OUT, RIGHT, UP
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.constants import FRAME_X_RADIUS,
FRAME_Y_RADIUS
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.constants import MED_SMALL_BUFF, SMALL_BUFF
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.mobject.functions import ParametricCurve
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.mobject.geometry import Arrow
SNGT_QHENOMENOLOGY_13: (0)             from manimlib.mobject.geometry import DashedLine
SNGT_QHENOMENOLOGY_14: (0)             from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_15: (0)             from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_16: (0)             from manimlib.mobject.number_line import NumberLine
SNGT_QHENOMENOLOGY_17: (0)             from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_18: (0)             from manimlib.mobject.types.dot_cloud import DotCloud
```

```
SNGT_QHENOMENOLOGY_19: (0)             from manimlib.mobject.types.surface import
ParametricSurface
SNGT_QHENOMENOLOGY_20: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_22: (0)             from manimlib.utils.bezier import inverse_interpolate
SNGT_QHENOMENOLOGY_23: (0)             from manimlib.utils.dict_ops import merge_dicts_recursively
SNGT_QHENOMENOLOGY_24: (0)             from manimlib.utils.simple_functions import binary_search
SNGT_QHENOMENOLOGY_25: (0)             from manimlib.utils.space_ops import angle_of_vector
SNGT_QHENOMENOLOGY_26: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.utils.space_ops import rotate_vector
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.space_ops import normalize
SNGT_QHENOMENOLOGY_29: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_30: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_31: (4)                 from typing import Callable, Iterable, Sequence, Type,
TypeVar, Optional
SNGT_QHENOMENOLOGY_32: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_33: (4)                 from manimlib.typing import ManimColor, Vect3,
Vect3Array, VectN, RangeSpecifier, Self
SNGT_QHENOMENOLOGY_34: (4)                 T = TypeVar("T", bound=Mobject)
SNGT_QHENOMENOLOGY_35: (0)             EPSILON = 1e-8
SNGT_QHENOMENOLOGY_36: (0)             DEFAULT_X_RANGE = (-8.0, 8.0, 1.0)
SNGT_QHENOMENOLOGY_37: (0)             DEFAULT_Y_RANGE = (-4.0, 4.0, 1.0)
SNGT_QHENOMENOLOGY_38: (0)             def full_range_specifier(range_args):
SNGT_QHENOMENOLOGY_39: (4)                 if len(range_args) == 2:
SNGT_QHENOMENOLOGY_40: (8)                     return (*range_args, 1)
SNGT_QHENOMENOLOGY_41: (4)                 return range_args
SNGT_QHENOMENOLOGY_42: (0)             class CoordinateSystem(ABC):
SNGT_QHENOMENOLOGY_43: (4)                 """
SNGT_QHENOMENOLOGY_44: (4)                 Abstract class for Axes and NumberPlane
SNGT_QHENOMENOLOGY_45: (4)                 """
SNGT_QHENOMENOLOGY_46: (4)                 dimension: int = 2
SNGT_QHENOMENOLOGY_47: (4)                 def __init__(
SNGT_QHENOMENOLOGY_48: (8)                     self,
SNGT_QHENOMENOLOGY_49: (8)                     x_range: RangeSpecifier = DEFAULT_X_RANGE,
SNGT_QHENOMENOLOGY_50: (8)                     y_range: RangeSpecifier = DEFAULT_Y_RANGE,
SNGT_QHENOMENOLOGY_51: (8)                     num_sampled_graph_points_per_tick: int = 5,
SNGT_QHENOMENOLOGY_52: (4)                 ):
SNGT_QHENOMENOLOGY_53: (8)                     self.x_range = full_range_specifier(x_range)
SNGT_QHENOMENOLOGY_54: (8)                     self.y_range = full_range_specifier(y_range)
SNGT_QHENOMENOLOGY_55: (8)                     self.num_sampled_graph_points_per_tick =
num_sampled_graph_points_per_tick
SNGT_QHENOMENOLOGY_56: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_57: (4)                 def coords_to_point(self, *coords: float | VectN) ->
Vect3 | Vect3Array:
SNGT_QHENOMENOLOGY_58: (8)                     raise Exception("Not implemented")
SNGT_QHENOMENOLOGY_59: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_60: (4)                 def point_to_coords(self, point: Vect3 | Vect3Array) ->
tuple[float | VectN, ...]:
SNGT_QHENOMENOLOGY_61: (8)                     raise Exception("Not implemented")
SNGT_QHENOMENOLOGY_62: (4)                 def c2p(self, *coords: float) -> Vect3 | Vect3Array:
SNGT_QHENOMENOLOGY_63: (8)                     """Abbreviation for coords_to_point"""
SNGT_QHENOMENOLOGY_64: (8)                     return self.coords_to_point(*coords)
SNGT_QHENOMENOLOGY_65: (4)                 def p2c(self, point: Vect3) -> tuple[float | VectN,
...]:
SNGT_QHENOMENOLOGY_66: (8)                     """Abbreviation for point_to_coords"""
SNGT_QHENOMENOLOGY_67: (8)                     return self.point_to_coords(point)
SNGT_QHENOMENOLOGY_68: (4)                 def get_origin(self) -> Vect3:
SNGT_QHENOMENOLOGY_69: (8)                     return self.c2p(*[0] * self.dimension)
SNGT_QHENOMENOLOGY_70: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_71: (4)                 def get_axes(self) -> VGroup:
SNGT_QHENOMENOLOGY_72: (8)                     raise Exception("Not implemented")
SNGT_QHENOMENOLOGY_73: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_74: (4)                 def get_all_ranges(self) -> list[np.ndarray]:
SNGT_QHENOMENOLOGY_75: (8)                     raise Exception("Not implemented")
SNGT_QHENOMENOLOGY_76: (4)                 def get_axis(self, index: int) -> NumberLine:
SNGT_QHENOMENOLOGY_77: (8)                     return self.get_axes()[index]
SNGT_QHENOMENOLOGY_78: (4)                 def get_x_axis(self) -> NumberLine:
```

```
SNGT_QHENOMENOLOGY_79: (8)                    return self.get_axis(0)
SNGT_QHENOMENOLOGY_80: (4)                def get_y_axis(self) -> NumberLine:
SNGT_QHENOMENOLOGY_81: (8)                    return self.get_axis(1)
SNGT_QHENOMENOLOGY_82: (4)                def get_z_axis(self) -> NumberLine:
SNGT_QHENOMENOLOGY_83: (8)                    return self.get_axis(2)
SNGT_QHENOMENOLOGY_84: (4)                def get_x_axis_label(
SNGT_QHENOMENOLOGY_85: (8)                    self,
SNGT_QHENOMENOLOGY_86: (8)                    label_tex: str,
SNGT_QHENOMENOLOGY_87: (8)                    edge: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_88: (8)                    direction: Vect3 = DL,
SNGT_QHENOMENOLOGY_89: (8)                    **kwargs
SNGT_QHENOMENOLOGY_90: (4)                ) -> Tex:
SNGT_QHENOMENOLOGY_91: (8)                    return self.get_axis_label(
SNGT_QHENOMENOLOGY_92: (12)                       label_tex, self.get_x_axis(),
SNGT_QHENOMENOLOGY_93: (12)                       edge, direction, **kwargs
SNGT_QHENOMENOLOGY_94: (8)                    )
SNGT_QHENOMENOLOGY_95: (4)                def get_y_axis_label(
SNGT_QHENOMENOLOGY_96: (8)                    self,
SNGT_QHENOMENOLOGY_97: (8)                    label_tex: str,
SNGT_QHENOMENOLOGY_98: (8)                    edge: Vect3 = UP,
SNGT_QHENOMENOLOGY_99: (8)                    direction: Vect3 = DR,
SNGT_QHENOMENOLOGY_100: (8)                   **kwargs
SNGT_QHENOMENOLOGY_101: (4)               ) -> Tex:
SNGT_QHENOMENOLOGY_102: (8)                   return self.get_axis_label(
SNGT_QHENOMENOLOGY_103: (12)                      label_tex, self.get_y_axis(),
SNGT_QHENOMENOLOGY_104: (12)                      edge, direction, **kwargs
SNGT_QHENOMENOLOGY_105: (8)                   )
SNGT_QHENOMENOLOGY_106: (4)               def get_axis_label(
SNGT_QHENOMENOLOGY_107: (8)                   self,
SNGT_QHENOMENOLOGY_108: (8)                   label_tex: str,
SNGT_QHENOMENOLOGY_109: (8)                   axis: Vect3,
SNGT_QHENOMENOLOGY_110: (8)                   edge: Vect3,
SNGT_QHENOMENOLOGY_111: (8)                   direction: Vect3,
SNGT_QHENOMENOLOGY_112: (8)                   buff: float = MED_SMALL_BUFF,
SNGT_QHENOMENOLOGY_113: (8)                   ensure_on_screen: bool = False
SNGT_QHENOMENOLOGY_114: (4)               ) -> Tex:
SNGT_QHENOMENOLOGY_115: (8)                   label = Tex(label_tex)
SNGT_QHENOMENOLOGY_116: (8)                   label.next_to(
SNGT_QHENOMENOLOGY_117: (12)                      axis.get_edge_center(edge), direction,
SNGT_QHENOMENOLOGY_118: (12)                      buff=buff
SNGT_QHENOMENOLOGY_119: (8)                   )
SNGT_QHENOMENOLOGY_120: (8)                   if ensure_on_screen:
SNGT_QHENOMENOLOGY_121: (12)                      label.shift_onto_screen(buff=MED_SMALL_BUFF)
SNGT_QHENOMENOLOGY_122: (8)                   return label
SNGT_QHENOMENOLOGY_123: (4)               def get_axis_labels(
SNGT_QHENOMENOLOGY_124: (8)                   self,
SNGT_QHENOMENOLOGY_125: (8)                   x_label_tex: str = "x",
SNGT_QHENOMENOLOGY_126: (8)                   y_label_tex: str = "y"
SNGT_QHENOMENOLOGY_127: (4)               ) -> VGroup:
SNGT_QHENOMENOLOGY_128: (8)                   self.axis_labels = VGroup(
SNGT_QHENOMENOLOGY_129: (12)                      self.get_x_axis_label(x_label_tex),
SNGT_QHENOMENOLOGY_130: (12)                      self.get_y_axis_label(y_label_tex),
SNGT_QHENOMENOLOGY_131: (8)                   )
SNGT_QHENOMENOLOGY_132: (8)                   return self.axis_labels
SNGT_QHENOMENOLOGY_133: (4)               def get_line_from_axis_to_point(
SNGT_QHENOMENOLOGY_134: (8)                   self,
SNGT_QHENOMENOLOGY_135: (8)                   index: int,
SNGT_QHENOMENOLOGY_136: (8)                   point: Vect3,
SNGT_QHENOMENOLOGY_137: (8)                   line_func: Type[T] = DashedLine,
SNGT_QHENOMENOLOGY_138: (8)                   color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_139: (8)                   stroke_width: float = 2
SNGT_QHENOMENOLOGY_140: (4)               ) -> T:
SNGT_QHENOMENOLOGY_141: (8)                   axis = self.get_axis(index)
SNGT_QHENOMENOLOGY_142: (8)                   line = line_func(axis.get_projection(point), point)
SNGT_QHENOMENOLOGY_143: (8)                   line.set_stroke(color, stroke_width)
SNGT_QHENOMENOLOGY_144: (8)                   return line
SNGT_QHENOMENOLOGY_145: (4)               def get_v_line(self, point: Vect3, **kwargs):
SNGT_QHENOMENOLOGY_146: (8)                   return self.get_line_from_axis_to_point(0, point,
**kwargs)
```

```
SNGT_QHENOMENOLOGY_147: (4)                def get_h_line(self, point: Vect3, **kwargs):
SNGT_QHENOMENOLOGY_148: (8)                    return self.get_line_from_axis_to_point(1, point,
**kwargs)
SNGT_QHENOMENOLOGY_149: (4)                def get_graph(
SNGT_QHENOMENOLOGY_150: (8)                    self,
SNGT_QHENOMENOLOGY_151: (8)                    function: Callable[[float], float],
SNGT_QHENOMENOLOGY_152: (8)                    x_range: Sequence[float] | None = None,
SNGT_QHENOMENOLOGY_153: (8)                    bind: bool = False,
SNGT_QHENOMENOLOGY_154: (8)                    **kwargs
SNGT_QHENOMENOLOGY_155: (4)                ) -> ParametricCurve:
SNGT_QHENOMENOLOGY_156: (8)                    x_range = x_range or self.x_range
SNGT_QHENOMENOLOGY_157: (8)                    t_range = np.ones(3)
SNGT_QHENOMENOLOGY_158: (8)                    t_range[:len(x_range)] = x_range
SNGT_QHENOMENOLOGY_159: (8)                    t_range[2] /=
self.num_sampled_graph_points_per_tick
SNGT_QHENOMENOLOGY_160: (8)                    def parametric_function(t: float) -> Vect3:
SNGT_QHENOMENOLOGY_161: (12)                       return self.c2p(t, function(t))
SNGT_QHENOMENOLOGY_162: (8)                    graph = ParametricCurve(
SNGT_QHENOMENOLOGY_163: (12)                       parametric_function,
SNGT_QHENOMENOLOGY_164: (12)                       t_range=tuple(t_range),
SNGT_QHENOMENOLOGY_165: (12)                       **kwargs
SNGT_QHENOMENOLOGY_166: (8)                    )
SNGT_QHENOMENOLOGY_167: (8)                    graph.underlying_function = function
SNGT_QHENOMENOLOGY_168: (8)                    graph.x_range = x_range
SNGT_QHENOMENOLOGY_169: (8)                    if bind:
SNGT_QHENOMENOLOGY_170: (12)                       self.bind_graph_to_func(graph, function)
SNGT_QHENOMENOLOGY_171: (8)                    return graph
SNGT_QHENOMENOLOGY_172: (4)                def get_parametric_curve(
SNGT_QHENOMENOLOGY_173: (8)                    self,
SNGT_QHENOMENOLOGY_174: (8)                    function: Callable[[float], Vect3],
SNGT_QHENOMENOLOGY_175: (8)                    **kwargs
SNGT_QHENOMENOLOGY_176: (4)                ) -> ParametricCurve:
SNGT_QHENOMENOLOGY_177: (8)                    dim = self.dimension
SNGT_QHENOMENOLOGY_178: (8)                    graph = ParametricCurve(
SNGT_QHENOMENOLOGY_179: (12)                       lambda t: self.coords_to_point(*function(t)
[:dim]),
SNGT_QHENOMENOLOGY_180: (12)                       **kwargs
SNGT_QHENOMENOLOGY_181: (8)                    )
SNGT_QHENOMENOLOGY_182: (8)                    graph.underlying_function = function
SNGT_QHENOMENOLOGY_183: (8)                    return graph
SNGT_QHENOMENOLOGY_184: (4)                def input_to_graph_point(
SNGT_QHENOMENOLOGY_185: (8)                    self,
SNGT_QHENOMENOLOGY_186: (8)                    x: float,
SNGT_QHENOMENOLOGY_187: (8)                    graph: ParametricCurve
SNGT_QHENOMENOLOGY_188: (4)                ) -> Vect3 | None:
SNGT_QHENOMENOLOGY_189: (8)                    if hasattr(graph, "underlying_function"):
SNGT_QHENOMENOLOGY_190: (12)                       return self.coords_to_point(x,
graph.underlying_function(x))
SNGT_QHENOMENOLOGY_191: (8)                    else:
SNGT_QHENOMENOLOGY_192: (12)                       alpha = binary_search(
SNGT_QHENOMENOLOGY_193: (16)                          function=lambda a: self.point_to_coords(
SNGT_QHENOMENOLOGY_194: (20)                              graph.quick_point_from_proportion(a)
SNGT_QHENOMENOLOGY_195: (16)                          )[0],
SNGT_QHENOMENOLOGY_196: (16)                          target=x,
SNGT_QHENOMENOLOGY_197: (16)                          lower_bound=self.x_range[0],
SNGT_QHENOMENOLOGY_198: (16)                          upper_bound=self.x_range[1],
SNGT_QHENOMENOLOGY_199: (12)                       )
SNGT_QHENOMENOLOGY_200: (12)                       if alpha is not None:
SNGT_QHENOMENOLOGY_201: (16)                          return
graph.quick_point_from_proportion(alpha)
SNGT_QHENOMENOLOGY_202: (12)                       else:
SNGT_QHENOMENOLOGY_203: (16)                          return None
SNGT_QHENOMENOLOGY_204: (4)                def i2gp(self, x: float, graph: ParametricCurve) ->
Vect3 | None:
SNGT_QHENOMENOLOGY_205: (8)                    """
SNGT_QHENOMENOLOGY_206: (8)                    Alias for input_to_graph_point
SNGT_QHENOMENOLOGY_207: (8)                    """
SNGT_QHENOMENOLOGY_208: (8)                    return self.input_to_graph_point(x, graph)
SNGT_QHENOMENOLOGY_209: (4)                def bind_graph_to_func(
```

```
SNGT_QHENOMENOLOGY_210: (8)                     self,
SNGT_QHENOMENOLOGY_211: (8)                     graph: VMobject,
SNGT_QHENOMENOLOGY_212: (8)                     func: Callable[[VectN], VectN],
SNGT_QHENOMENOLOGY_213: (8)                     jagged: bool = False,
SNGT_QHENOMENOLOGY_214: (8)                     get_discontinuities: Optional[Callable[[], Vect3]]
= None
SNGT_QHENOMENOLOGY_215: (4)                 ) -> VMobject:
SNGT_QHENOMENOLOGY_216: (8)                     """
SNGT_QHENOMENOLOGY_217: (8)                     Use for graphing functions which might change over
time, or change with
SNGT_QHENOMENOLOGY_218: (8)                     conditions
SNGT_QHENOMENOLOGY_219: (8)                     """
SNGT_QHENOMENOLOGY_220: (8)                     x_values = np.array([self.x_axis.p2n(p) for p in
graph.get_points()])
SNGT_QHENOMENOLOGY_221: (8)                     def get_graph_points():
SNGT_QHENOMENOLOGY_222: (12)                        xs = x_values
SNGT_QHENOMENOLOGY_223: (12)                        if get_discontinuities:
SNGT_QHENOMENOLOGY_224: (16)                            ds = get_discontinuities()
SNGT_QHENOMENOLOGY_225: (16)                            ep = 1e-6
SNGT_QHENOMENOLOGY_226: (16)                            added_xs = it.chain(*((d - ep, d + ep) for
d in ds))
SNGT_QHENOMENOLOGY_227: (16)                            xs[:] = sorted([*x_values, *added_xs])
[:len(x_values)]
SNGT_QHENOMENOLOGY_228: (12)                        return self.c2p(xs, func(xs))
SNGT_QHENOMENOLOGY_229: (8)                     graph.add_updater(
SNGT_QHENOMENOLOGY_230: (12)                        lambda g:
g.set_points_as_corners(get_graph_points())
SNGT_QHENOMENOLOGY_231: (8)                     )
SNGT_QHENOMENOLOGY_232: (8)                     if not jagged:
SNGT_QHENOMENOLOGY_233: (12)                        graph.add_updater(lambda g:
g.make_smooth(approx=True))
SNGT_QHENOMENOLOGY_234: (8)                     return graph
SNGT_QHENOMENOLOGY_235: (4)                 def get_graph_label(
SNGT_QHENOMENOLOGY_236: (8)                     self,
SNGT_QHENOMENOLOGY_237: (8)                     graph: ParametricCurve,
SNGT_QHENOMENOLOGY_238: (8)                     label: str | Mobject = "f(x)",
SNGT_QHENOMENOLOGY_239: (8)                     x: float | None = None,
SNGT_QHENOMENOLOGY_240: (8)                     direction: Vect3 = RIGHT,
SNGT_QHENOMENOLOGY_241: (8)                     buff: float = MED_SMALL_BUFF,
SNGT_QHENOMENOLOGY_242: (8)                     color: ManimColor | None = None
SNGT_QHENOMENOLOGY_243: (4)                 ) -> Tex | Mobject:
SNGT_QHENOMENOLOGY_244: (8)                     if isinstance(label, str):
SNGT_QHENOMENOLOGY_245: (12)                        label = Tex(label)
SNGT_QHENOMENOLOGY_246: (8)                     if color is None:
SNGT_QHENOMENOLOGY_247: (12)                        label.match_color(graph)
SNGT_QHENOMENOLOGY_248: (8)                     if x is None:
SNGT_QHENOMENOLOGY_249: (12)                        max_y = FRAME_Y_RADIUS - label.get_height()
SNGT_QHENOMENOLOGY_250: (12)                        max_x = FRAME_X_RADIUS - label.get_width()
SNGT_QHENOMENOLOGY_251: (12)                        for x0 in np.arange(*self.x_range)[::-1]:
SNGT_QHENOMENOLOGY_252: (16)                            pt = self.i2gp(x0, graph)
SNGT_QHENOMENOLOGY_253: (16)                            if abs(pt[0]) < max_x and abs(pt[1]) <
max_y:
SNGT_QHENOMENOLOGY_254: (20)                                x = x0
SNGT_QHENOMENOLOGY_255: (20)                                break
SNGT_QHENOMENOLOGY_256: (12)                        if x is None:
SNGT_QHENOMENOLOGY_257: (16)                            x = self.x_range[1]
SNGT_QHENOMENOLOGY_258: (8)                     point = self.input_to_graph_point(x, graph)
SNGT_QHENOMENOLOGY_259: (8)                     angle = self.angle_of_tangent(x, graph)
SNGT_QHENOMENOLOGY_260: (8)                     normal = rotate_vector(RIGHT, angle + 90 * DEG)
SNGT_QHENOMENOLOGY_261: (8)                     if normal[1] < 0:
SNGT_QHENOMENOLOGY_262: (12)                        normal *= -1
SNGT_QHENOMENOLOGY_263: (8)                     label.next_to(point, normal, buff=buff)
SNGT_QHENOMENOLOGY_264: (8)                     label.shift_onto_screen()
SNGT_QHENOMENOLOGY_265: (8)                     return label
SNGT_QHENOMENOLOGY_266: (4)                 def get_v_line_to_graph(self, x: float, graph:
ParametricCurve, **kwargs):
SNGT_QHENOMENOLOGY_267: (8)                     return self.get_v_line(self.i2gp(x, graph),
**kwargs)
SNGT_QHENOMENOLOGY_268: (4)                 def get_h_line_to_graph(self, x: float, graph:
```

```
ParametricCurve, **kwargs):
SNGT_QHENOMENOLOGY_269: (8)                 return self.get_h_line(self.i2gp(x, graph),
**kwargs)
SNGT_QHENOMENOLOGY_270: (4)             def get_scatterplot(self,
SNGT_QHENOMENOLOGY_271: (24)                                x_values: Vect3Array,
SNGT_QHENOMENOLOGY_272: (24)                                y_values: Vect3Array,
SNGT_QHENOMENOLOGY_273: (24)                                **dot_config):
SNGT_QHENOMENOLOGY_274: (8)                 return DotCloud(self.c2p(x_values, y_values),
**dot_config)
SNGT_QHENOMENOLOGY_275: (4)             def angle_of_tangent(
SNGT_QHENOMENOLOGY_276: (8)                 self,
SNGT_QHENOMENOLOGY_277: (8)                 x: float,
SNGT_QHENOMENOLOGY_278: (8)                 graph: ParametricCurve,
SNGT_QHENOMENOLOGY_279: (8)                 dx: float = EPSILON
SNGT_QHENOMENOLOGY_280: (4)             ) -> float:
SNGT_QHENOMENOLOGY_281: (8)                 p0 = self.input_to_graph_point(x, graph)
SNGT_QHENOMENOLOGY_282: (8)                 p1 = self.input_to_graph_point(x + dx, graph)
SNGT_QHENOMENOLOGY_283: (8)                 return angle_of_vector(p1 - p0)
SNGT_QHENOMENOLOGY_284: (4)             def slope_of_tangent(
SNGT_QHENOMENOLOGY_285: (8)                 self,
SNGT_QHENOMENOLOGY_286: (8)                 x: float,
SNGT_QHENOMENOLOGY_287: (8)                 graph: ParametricCurve,
SNGT_QHENOMENOLOGY_288: (8)                 **kwargs
SNGT_QHENOMENOLOGY_289: (4)             ) -> float:
SNGT_QHENOMENOLOGY_290: (8)                 return np.tan(self.angle_of_tangent(x, graph,
**kwargs))
SNGT_QHENOMENOLOGY_291: (4)             def get_tangent_line(
SNGT_QHENOMENOLOGY_292: (8)                 self,
SNGT_QHENOMENOLOGY_293: (8)                 x: float,
SNGT_QHENOMENOLOGY_294: (8)                 graph: ParametricCurve,
SNGT_QHENOMENOLOGY_295: (8)                 length: float = 5,
SNGT_QHENOMENOLOGY_296: (8)                 line_func: Type[T] = Line
SNGT_QHENOMENOLOGY_297: (4)             ) -> T:
SNGT_QHENOMENOLOGY_298: (8)                 line = line_func(LEFT, RIGHT)
SNGT_QHENOMENOLOGY_299: (8)                 line.set_width(length)
SNGT_QHENOMENOLOGY_300: (8)                 line.rotate(self.angle_of_tangent(x, graph))
SNGT_QHENOMENOLOGY_301: (8)                 line.move_to(self.input_to_graph_point(x, graph))
SNGT_QHENOMENOLOGY_302: (8)                 return line
SNGT_QHENOMENOLOGY_303: (4)             def get_riemann_rectangles(
SNGT_QHENOMENOLOGY_304: (8)                 self,
SNGT_QHENOMENOLOGY_305: (8)                 graph: ParametricCurve,
SNGT_QHENOMENOLOGY_306: (8)                 x_range: Sequence[float] = None,
SNGT_QHENOMENOLOGY_307: (8)                 dx: float | None = None,
SNGT_QHENOMENOLOGY_308: (8)                 input_sample_type: str = "left",
SNGT_QHENOMENOLOGY_309: (8)                 stroke_width: float = 1,
SNGT_QHENOMENOLOGY_310: (8)                 stroke_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_311: (8)                 fill_opacity: float = 1,
SNGT_QHENOMENOLOGY_312: (8)                 colors: Iterable[ManimColor] = (BLUE, GREEN),
SNGT_QHENOMENOLOGY_313: (8)                 negative_color: ManimColor = RED,
SNGT_QHENOMENOLOGY_314: (8)                 stroke_background: bool = True,
SNGT_QHENOMENOLOGY_315: (8)                 show_signed_area: bool = True
SNGT_QHENOMENOLOGY_316: (4)             ) -> VGroup:
SNGT_QHENOMENOLOGY_317: (8)                 if x_range is None:
SNGT_QHENOMENOLOGY_318: (12)                    x_range = self.x_range[:2]
SNGT_QHENOMENOLOGY_319: (8)                 if dx is None:
SNGT_QHENOMENOLOGY_320: (12)                    dx = self.x_range[2]
SNGT_QHENOMENOLOGY_321: (8)                 if len(x_range) < 3:
SNGT_QHENOMENOLOGY_322: (12)                    x_range = [*x_range, dx]
SNGT_QHENOMENOLOGY_323: (8)                 rects = []
SNGT_QHENOMENOLOGY_324: (8)                 x_range[1] = x_range[1] + dx
SNGT_QHENOMENOLOGY_325: (8)                 xs = np.arange(*x_range)
SNGT_QHENOMENOLOGY_326: (8)                 for x0, x1 in zip(xs, xs[1:]):
SNGT_QHENOMENOLOGY_327: (12)                    if input_sample_type == "left":
SNGT_QHENOMENOLOGY_328: (16)                        sample = x0
SNGT_QHENOMENOLOGY_329: (12)                    elif input_sample_type == "right":
SNGT_QHENOMENOLOGY_330: (16)                        sample = x1
SNGT_QHENOMENOLOGY_331: (12)                    elif input_sample_type == "center":
SNGT_QHENOMENOLOGY_332: (16)                        sample = 0.5 * x0 + 0.5 * x1
SNGT_QHENOMENOLOGY_333: (12)                    else:
```

```
SNGT_QHENOMENOLOGY_334: (16)                            raise Exception("Invalid input sample
type")
SNGT_QHENOMENOLOGY_335: (12)                        height_vect = self.i2gp(sample, graph) -
self.c2p(sample, 0)
SNGT_QHENOMENOLOGY_336: (12)                        rect = Rectangle(
SNGT_QHENOMENOLOGY_337: (16)                            width=self.x_axis.n2p(x1)[0] -
self.x_axis.n2p(x0)[0],
SNGT_QHENOMENOLOGY_338: (16)                            height=get_norm(height_vect),
SNGT_QHENOMENOLOGY_339: (12)                        )
SNGT_QHENOMENOLOGY_340: (12)                        rect.positive = height_vect[1] > 0
SNGT_QHENOMENOLOGY_341: (12)                        rect.move_to(self.c2p(x0, 0), DL if
rect.positive else UL)
SNGT_QHENOMENOLOGY_342: (12)                        rects.append(rect)
SNGT_QHENOMENOLOGY_343: (8)                    result = VGroup(*rects)
SNGT_QHENOMENOLOGY_344: (8)                    result.set_submobject_colors_by_gradient(*colors)
SNGT_QHENOMENOLOGY_345: (8)                    result.set_style(
SNGT_QHENOMENOLOGY_346: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_347: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_348: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_349: (12)                        stroke_behind=stroke_background
SNGT_QHENOMENOLOGY_350: (8)                    )
SNGT_QHENOMENOLOGY_351: (8)                    for rect in result:
SNGT_QHENOMENOLOGY_352: (12)                        if not rect.positive:
SNGT_QHENOMENOLOGY_353: (16)                            rect.set_fill(negative_color)
SNGT_QHENOMENOLOGY_354: (8)                    return result
SNGT_QHENOMENOLOGY_355: (4)                def get_area_under_graph(self, graph, x_range,
fill_color=BLUE, fill_opacity=0.5):
SNGT_QHENOMENOLOGY_356: (8)                    if not hasattr(graph, "x_range"):
SNGT_QHENOMENOLOGY_357: (12)                        raise Exception("Argument `graph` must have
attribute `x_range`")
SNGT_QHENOMENOLOGY_358: (8)                    alpha_bounds = [
SNGT_QHENOMENOLOGY_359: (12)                        inverse_interpolate(*graph.x_range, x)
SNGT_QHENOMENOLOGY_360: (12)                        for x in x_range
SNGT_QHENOMENOLOGY_361: (8)                    ]
SNGT_QHENOMENOLOGY_362: (8)                    sub_graph = graph.copy()
SNGT_QHENOMENOLOGY_363: (8)                    sub_graph.pointwise_become_partial(graph,
*alpha_bounds)
SNGT_QHENOMENOLOGY_364: (8)                    sub_graph.add_line_to(self.c2p(x_range[1], 0))
SNGT_QHENOMENOLOGY_365: (8)                    sub_graph.add_line_to(self.c2p(x_range[0], 0))
SNGT_QHENOMENOLOGY_366: (8)                    sub_graph.add_line_to(sub_graph.get_start())
SNGT_QHENOMENOLOGY_367: (8)                    sub_graph.set_stroke(width=0)
SNGT_QHENOMENOLOGY_368: (8)                    sub_graph.set_fill(fill_color, fill_opacity)
SNGT_QHENOMENOLOGY_369: (8)                    return sub_graph
SNGT_QHENOMENOLOGY_370: (0)            class Axes(VGroup, CoordinateSystem):
SNGT_QHENOMENOLOGY_371: (4)                default_axis_config: dict = dict()
SNGT_QHENOMENOLOGY_372: (4)                default_x_axis_config: dict = dict()
SNGT_QHENOMENOLOGY_373: (4)                default_y_axis_config: dict =
dict(line_to_number_direction=LEFT)
SNGT_QHENOMENOLOGY_374: (4)                def __init__(
SNGT_QHENOMENOLOGY_375: (8)                    self,
SNGT_QHENOMENOLOGY_376: (8)                    x_range: RangeSpecifier = DEFAULT_X_RANGE,
SNGT_QHENOMENOLOGY_377: (8)                    y_range: RangeSpecifier = DEFAULT_Y_RANGE,
SNGT_QHENOMENOLOGY_378: (8)                    axis_config: dict = dict(),
SNGT_QHENOMENOLOGY_379: (8)                    x_axis_config: dict = dict(),
SNGT_QHENOMENOLOGY_380: (8)                    y_axis_config: dict = dict(),
SNGT_QHENOMENOLOGY_381: (8)                    height: float | None = None,
SNGT_QHENOMENOLOGY_382: (8)                    width: float | None = None,
SNGT_QHENOMENOLOGY_383: (8)                    unit_size: float = 1.0,
SNGT_QHENOMENOLOGY_384: (8)                    **kwargs
SNGT_QHENOMENOLOGY_385: (4)                ):
SNGT_QHENOMENOLOGY_386: (8)                    CoordinateSystem.__init__(self, x_range, y_range,
**kwargs)
SNGT_QHENOMENOLOGY_387: (8)                    kwargs.pop("num_sampled_graph_points_per_tick",
None)
SNGT_QHENOMENOLOGY_388: (8)                    VGroup.__init__(self, **kwargs)
SNGT_QHENOMENOLOGY_389: (8)                    axis_config = dict(**axis_config,
unit_size=unit_size)
SNGT_QHENOMENOLOGY_390: (8)                    self.x_axis = self.create_axis(
SNGT_QHENOMENOLOGY_391: (12)                        self.x_range,
```

```
SNGT_QHENOMENOLOGY_392: (12)                        axis_config=merge_dicts_recursively(
SNGT_QHENOMENOLOGY_393: (16)                            self.default_axis_config,
SNGT_QHENOMENOLOGY_394: (16)                            self.default_x_axis_config,
SNGT_QHENOMENOLOGY_395: (16)                            axis_config,
SNGT_QHENOMENOLOGY_396: (16)                            x_axis_config
SNGT_QHENOMENOLOGY_397: (12)                        ),
SNGT_QHENOMENOLOGY_398: (12)                        length=width,
SNGT_QHENOMENOLOGY_399: (8)                    )
SNGT_QHENOMENOLOGY_400: (8)                    self.y_axis = self.create_axis(
SNGT_QHENOMENOLOGY_401: (12)                        self.y_range,
SNGT_QHENOMENOLOGY_402: (12)                        axis_config=merge_dicts_recursively(
SNGT_QHENOMENOLOGY_403: (16)                            self.default_axis_config,
SNGT_QHENOMENOLOGY_404: (16)                            self.default_y_axis_config,
SNGT_QHENOMENOLOGY_405: (16)                            axis_config,
SNGT_QHENOMENOLOGY_406: (16)                            y_axis_config
SNGT_QHENOMENOLOGY_407: (12)                        ),
SNGT_QHENOMENOLOGY_408: (12)                        length=height,
SNGT_QHENOMENOLOGY_409: (8)                    )
SNGT_QHENOMENOLOGY_410: (8)                    self.y_axis.rotate(90 * DEG, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_411: (8)                    self.axes = VGroup(self.x_axis, self.y_axis)
SNGT_QHENOMENOLOGY_412: (8)                    self.add(*self.axes)
SNGT_QHENOMENOLOGY_413: (8)                    self.center()
SNGT_QHENOMENOLOGY_414: (4)                def create_axis(
SNGT_QHENOMENOLOGY_415: (8)                    self,
SNGT_QHENOMENOLOGY_416: (8)                    range_terms: RangeSpecifier,
SNGT_QHENOMENOLOGY_417: (8)                    axis_config: dict,
SNGT_QHENOMENOLOGY_418: (8)                    length: float | None
SNGT_QHENOMENOLOGY_419: (4)                ) -> NumberLine:
SNGT_QHENOMENOLOGY_420: (8)                    axis = NumberLine(range_terms, width=length,
**axis_config)
SNGT_QHENOMENOLOGY_421: (8)                    axis.shift(-axis.n2p(0))
SNGT_QHENOMENOLOGY_422: (8)                    return axis
SNGT_QHENOMENOLOGY_423: (4)                def coords_to_point(self, *coords: float | VectN) ->
Vect3 | Vect3Array:
SNGT_QHENOMENOLOGY_424: (8)                    origin = self.x_axis.number_to_point(0)
SNGT_QHENOMENOLOGY_425: (8)                    return origin + sum(
SNGT_QHENOMENOLOGY_426: (12)                        axis.number_to_point(coord) - origin
SNGT_QHENOMENOLOGY_427: (12)                        for axis, coord in zip(self.get_axes(), coords)
SNGT_QHENOMENOLOGY_428: (8)                    )
SNGT_QHENOMENOLOGY_429: (4)                def point_to_coords(self, point: Vect3 | Vect3Array) ->
tuple[float | VectN, ...]:
SNGT_QHENOMENOLOGY_430: (8)                    return tuple([
SNGT_QHENOMENOLOGY_431: (12)                        axis.point_to_number(point)
SNGT_QHENOMENOLOGY_432: (12)                        for axis in self.get_axes()
SNGT_QHENOMENOLOGY_433: (8)                    ])
SNGT_QHENOMENOLOGY_434: (4)                def get_axes(self) -> VGroup:
SNGT_QHENOMENOLOGY_435: (8)                    return self.axes
SNGT_QHENOMENOLOGY_436: (4)                def get_all_ranges(self) -> list[Sequence[float]]:
SNGT_QHENOMENOLOGY_437: (8)                    return [self.x_range, self.y_range]
SNGT_QHENOMENOLOGY_438: (4)                def add_coordinate_labels(
SNGT_QHENOMENOLOGY_439: (8)                    self,
SNGT_QHENOMENOLOGY_440: (8)                    x_values: Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_441: (8)                    y_values: Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_442: (8)                    excluding: Iterable[float] = [0],
SNGT_QHENOMENOLOGY_443: (8)                    **kwargs
SNGT_QHENOMENOLOGY_444: (4)                ) -> VGroup:
SNGT_QHENOMENOLOGY_445: (8)                    axes = self.get_axes()
SNGT_QHENOMENOLOGY_446: (8)                    self.coordinate_labels = VGroup()
SNGT_QHENOMENOLOGY_447: (8)                    for axis, values in zip(axes, [x_values,
y_values]):
SNGT_QHENOMENOLOGY_448: (12)                        labels = axis.add_numbers(values,
excluding=excluding, **kwargs)
SNGT_QHENOMENOLOGY_449: (12)                        self.coordinate_labels.add(labels)
SNGT_QHENOMENOLOGY_450: (8)                    return self.coordinate_labels
SNGT_QHENOMENOLOGY_451: (0)            class ThreeDAxes(Axes):
SNGT_QHENOMENOLOGY_452: (4)                dimension: int = 3
SNGT_QHENOMENOLOGY_453: (4)                default_z_axis_config: dict = dict()
SNGT_QHENOMENOLOGY_454: (4)                def __init__(
SNGT_QHENOMENOLOGY_455: (8)                    self,
```

```
SNGT_QHENOMENOLOGY_456: (8)                     x_range: RangeSpecifier = (-6.0, 6.0, 1.0),
SNGT_QHENOMENOLOGY_457: (8)                     y_range: RangeSpecifier = (-5.0, 5.0, 1.0),
SNGT_QHENOMENOLOGY_458: (8)                     z_range: RangeSpecifier = (-4.0, 4.0, 1.0),
SNGT_QHENOMENOLOGY_459: (8)                     z_axis_config: dict = dict(),
SNGT_QHENOMENOLOGY_460: (8)                     z_normal: Vect3 = DOWN,
SNGT_QHENOMENOLOGY_461: (8)                     depth: float | None = None,
SNGT_QHENOMENOLOGY_462: (8)                     **kwargs
SNGT_QHENOMENOLOGY_463: (4)                 ):
SNGT_QHENOMENOLOGY_464: (8)                     Axes.__init__(self, x_range, y_range, **kwargs)
SNGT_QHENOMENOLOGY_465: (8)                     self.z_range = full_range_specifier(z_range)
SNGT_QHENOMENOLOGY_466: (8)                     self.z_axis = self.create_axis(
SNGT_QHENOMENOLOGY_467: (12)                        self.z_range,
SNGT_QHENOMENOLOGY_468: (12)                        axis_config=merge_dicts_recursively(
SNGT_QHENOMENOLOGY_469: (16)                            self.default_axis_config,
SNGT_QHENOMENOLOGY_470: (16)                            self.default_z_axis_config,
SNGT_QHENOMENOLOGY_471: (16)                            kwargs.get("axis_config", {}),
SNGT_QHENOMENOLOGY_472: (16)                            z_axis_config
SNGT_QHENOMENOLOGY_473: (12)                        ),
SNGT_QHENOMENOLOGY_474: (12)                        length=depth,
SNGT_QHENOMENOLOGY_475: (8)                     )
SNGT_QHENOMENOLOGY_476: (8)                     self.z_axis.rotate(-PI / 2, UP, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_477: (8)                     self.z_axis.rotate(
SNGT_QHENOMENOLOGY_478: (12)                        angle_of_vector(z_normal), OUT,
SNGT_QHENOMENOLOGY_479: (12)                        about_point=ORIGIN
SNGT_QHENOMENOLOGY_480: (8)                     )
SNGT_QHENOMENOLOGY_481: (8)                     self.z_axis.shift(self.x_axis.n2p(0))
SNGT_QHENOMENOLOGY_482: (8)                     self.axes.add(self.z_axis)
SNGT_QHENOMENOLOGY_483: (8)                     self.add(self.z_axis)
SNGT_QHENOMENOLOGY_484: (4)                 def get_all_ranges(self) -> list[Sequence[float]]:
SNGT_QHENOMENOLOGY_485: (8)                     return [self.x_range, self.y_range, self.z_range]
SNGT_QHENOMENOLOGY_486: (4)                 def add_axis_labels(self, x_tex="x", y_tex="y",
z_tex="z", font_size=24, buff=0.2):
SNGT_QHENOMENOLOGY_487: (8)                     x_label, y_label, z_label = labels = VGroup(*(
SNGT_QHENOMENOLOGY_488: (12)                        Tex(tex, font_size=font_size)
SNGT_QHENOMENOLOGY_489: (12)                        for tex in [x_tex, y_tex, z_tex]
SNGT_QHENOMENOLOGY_490: (8)                     ))
SNGT_QHENOMENOLOGY_491: (8)                     z_label.rotate(PI / 2, RIGHT)
SNGT_QHENOMENOLOGY_492: (8)                     for label, axis in zip(labels, self):
SNGT_QHENOMENOLOGY_493: (12)                        label.next_to(axis,
normalize(np.round(axis.get_vector()), 2), buff=buff)
SNGT_QHENOMENOLOGY_494: (12)                        axis.add(label)
SNGT_QHENOMENOLOGY_495: (8)                     self.axis_labels = labels
SNGT_QHENOMENOLOGY_496: (4)                 def get_graph(
SNGT_QHENOMENOLOGY_497: (8)                     self,
SNGT_QHENOMENOLOGY_498: (8)                     func,
SNGT_QHENOMENOLOGY_499: (8)                     color=BLUE_E,
SNGT_QHENOMENOLOGY_500: (8)                     opacity=0.9,
SNGT_QHENOMENOLOGY_501: (8)                     u_range=None,
SNGT_QHENOMENOLOGY_502: (8)                     v_range=None,
SNGT_QHENOMENOLOGY_503: (8)                     **kwargs
SNGT_QHENOMENOLOGY_504: (4)                 ) -> ParametricSurface:
SNGT_QHENOMENOLOGY_505: (8)                     xu = self.x_axis.get_unit_size()
SNGT_QHENOMENOLOGY_506: (8)                     yu = self.y_axis.get_unit_size()
SNGT_QHENOMENOLOGY_507: (8)                     zu = self.z_axis.get_unit_size()
SNGT_QHENOMENOLOGY_508: (8)                     x0, y0, z0 = self.get_origin()
SNGT_QHENOMENOLOGY_509: (8)                     u_range = u_range or self.x_range[:2]
SNGT_QHENOMENOLOGY_510: (8)                     v_range = v_range or self.y_range[:2]
SNGT_QHENOMENOLOGY_511: (8)                     return ParametricSurface(
SNGT_QHENOMENOLOGY_512: (12)                        lambda u, v: [xu * u + x0, yu * v + y0, zu *
func(u, v) + z0],
SNGT_QHENOMENOLOGY_513: (12)                        u_range=u_range,
SNGT_QHENOMENOLOGY_514: (12)                        v_range=v_range,
SNGT_QHENOMENOLOGY_515: (12)                        color=color,
SNGT_QHENOMENOLOGY_516: (12)                        opacity=opacity,
SNGT_QHENOMENOLOGY_517: (12)                        **kwargs
SNGT_QHENOMENOLOGY_518: (8)                     )
SNGT_QHENOMENOLOGY_519: (4)                 def get_parametric_surface(
SNGT_QHENOMENOLOGY_520: (8)                     self,
SNGT_QHENOMENOLOGY_521: (8)                     func,
```

```
SNGT_QHENOMENOLOGY_522: (8)                     color=BLUE_E,
SNGT_QHENOMENOLOGY_523: (8)                     opacity=0.9,
SNGT_QHENOMENOLOGY_524: (8)                     **kwargs
SNGT_QHENOMENOLOGY_525: (4)                 ) -> ParametricSurface:
SNGT_QHENOMENOLOGY_526: (8)                     surface = ParametricSurface(func, color=color,
opacity=opacity, **kwargs)
SNGT_QHENOMENOLOGY_527: (8)                     axes = [self.x_axis, self.y_axis, self.z_axis]
SNGT_QHENOMENOLOGY_528: (8)                     for dim, axis in zip(range(3), axes):
SNGT_QHENOMENOLOGY_529: (12)                        surface.stretch(axis.get_unit_size(), dim,
about_point=ORIGIN)
SNGT_QHENOMENOLOGY_530: (8)                     surface.shift(self.get_origin())
SNGT_QHENOMENOLOGY_531: (8)                     return surface
SNGT_QHENOMENOLOGY_532: (0)             class NumberPlane(Axes):
SNGT_QHENOMENOLOGY_533: (4)                 default_axis_config: dict = dict(
SNGT_QHENOMENOLOGY_534: (8)                     stroke_color=WHITE,
SNGT_QHENOMENOLOGY_535: (8)                     stroke_width=2,
SNGT_QHENOMENOLOGY_536: (8)                     include_ticks=False,
SNGT_QHENOMENOLOGY_537: (8)                     include_tip=False,
SNGT_QHENOMENOLOGY_538: (8)                     line_to_number_buff=SMALL_BUFF,
SNGT_QHENOMENOLOGY_539: (8)                     line_to_number_direction=DL,
SNGT_QHENOMENOLOGY_540: (4)                 )
SNGT_QHENOMENOLOGY_541: (4)                 default_y_axis_config: dict = dict(
SNGT_QHENOMENOLOGY_542: (8)                     line_to_number_direction=DL,
SNGT_QHENOMENOLOGY_543: (4)                 )
SNGT_QHENOMENOLOGY_544: (4)                 def __init__(
SNGT_QHENOMENOLOGY_545: (8)                     self,
SNGT_QHENOMENOLOGY_546: (8)                     x_range: RangeSpecifier = (-8.0, 8.0, 1.0),
SNGT_QHENOMENOLOGY_547: (8)                     y_range: RangeSpecifier = (-4.0, 4.0, 1.0),
SNGT_QHENOMENOLOGY_548: (8)                     background_line_style: dict = dict(
SNGT_QHENOMENOLOGY_549: (12)                        stroke_color=BLUE_D,
SNGT_QHENOMENOLOGY_550: (12)                        stroke_width=2,
SNGT_QHENOMENOLOGY_551: (12)                        stroke_opacity=1,
SNGT_QHENOMENOLOGY_552: (8)                     ),
SNGT_QHENOMENOLOGY_553: (8)                     faded_line_style: dict = dict(),
SNGT_QHENOMENOLOGY_554: (8)                     faded_line_ratio: int = 4,
SNGT_QHENOMENOLOGY_555: (8)                     make_smooth_after_applying_functions: bool = True,
SNGT_QHENOMENOLOGY_556: (8)                     **kwargs
SNGT_QHENOMENOLOGY_557: (4)                 ):
SNGT_QHENOMENOLOGY_558: (8)                     super().__init__(x_range, y_range, **kwargs)
SNGT_QHENOMENOLOGY_559: (8)                     self.background_line_style =
dict(background_line_style)
SNGT_QHENOMENOLOGY_560: (8)                     self.faded_line_style = dict(faded_line_style)
SNGT_QHENOMENOLOGY_561: (8)                     self.faded_line_ratio = faded_line_ratio
SNGT_QHENOMENOLOGY_562: (8)                     self.make_smooth_after_applying_functions =
make_smooth_after_applying_functions
SNGT_QHENOMENOLOGY_563: (8)                     self.init_background_lines()
SNGT_QHENOMENOLOGY_564: (4)                 def init_background_lines(self) -> None:
SNGT_QHENOMENOLOGY_565: (8)                     if not self.faded_line_style:
SNGT_QHENOMENOLOGY_566: (12)                        style = dict(self.background_line_style)
SNGT_QHENOMENOLOGY_567: (12)                        for key in style:
SNGT_QHENOMENOLOGY_568: (16)                            if isinstance(style[key], numbers.Number):
SNGT_QHENOMENOLOGY_569: (20)                                style[key] *= 0.5
SNGT_QHENOMENOLOGY_570: (12)                        self.faded_line_style = style
SNGT_QHENOMENOLOGY_571: (8)                     self.background_lines, self.faded_lines =
self.get_lines()
SNGT_QHENOMENOLOGY_572: (8)
self.background_lines.set_style(**self.background_line_style)
SNGT_QHENOMENOLOGY_573: (8)                     self.faded_lines.set_style(**self.faded_line_style)
SNGT_QHENOMENOLOGY_574: (8)                     self.add_to_back(
SNGT_QHENOMENOLOGY_575: (12)                        self.faded_lines,
SNGT_QHENOMENOLOGY_576: (12)                        self.background_lines,
SNGT_QHENOMENOLOGY_577: (8)                     )
SNGT_QHENOMENOLOGY_578: (4)                 def get_lines(self) -> tuple[VGroup, VGroup]:
SNGT_QHENOMENOLOGY_579: (8)                     x_axis = self.get_x_axis()
SNGT_QHENOMENOLOGY_580: (8)                     y_axis = self.get_y_axis()
SNGT_QHENOMENOLOGY_581: (8)                     x_lines1, x_lines2 =
self.get_lines_parallel_to_axis(x_axis, y_axis)
SNGT_QHENOMENOLOGY_582: (8)                     y_lines1, y_lines2 =
self.get_lines_parallel_to_axis(y_axis, x_axis)
```

```
SNGT_QHENOMENOLOGY_583: (8)                     lines1 = VGroup(*x_lines1, *y_lines1)
SNGT_QHENOMENOLOGY_584: (8)                     lines2 = VGroup(*x_lines2, *y_lines2)
SNGT_QHENOMENOLOGY_585: (8)                     return lines1, lines2
SNGT_QHENOMENOLOGY_586: (4)                 def get_lines_parallel_to_axis(
SNGT_QHENOMENOLOGY_587: (8)                     self,
SNGT_QHENOMENOLOGY_588: (8)                     axis1: NumberLine,
SNGT_QHENOMENOLOGY_589: (8)                     axis2: NumberLine
SNGT_QHENOMENOLOGY_590: (4)                 ) -> tuple[VGroup, VGroup]:
SNGT_QHENOMENOLOGY_591: (8)                     freq = axis2.x_step
SNGT_QHENOMENOLOGY_592: (8)                     ratio = self.faded_line_ratio
SNGT_QHENOMENOLOGY_593: (8)                     line = Line(axis1.get_start(), axis1.get_end())
SNGT_QHENOMENOLOGY_594: (8)                     dense_freq = (1 + ratio)
SNGT_QHENOMENOLOGY_595: (8)                     step = (1 / dense_freq) * freq
SNGT_QHENOMENOLOGY_596: (8)                     lines1 = VGroup()
SNGT_QHENOMENOLOGY_597: (8)                     lines2 = VGroup()
SNGT_QHENOMENOLOGY_598: (8)                     inputs = np.arange(axis2.x_min, axis2.x_max + step,
step)
SNGT_QHENOMENOLOGY_599: (8)                     for i, x in enumerate(inputs):
SNGT_QHENOMENOLOGY_600: (12)                        if abs(x) < 1e-8:
SNGT_QHENOMENOLOGY_601: (16)                            continue
SNGT_QHENOMENOLOGY_602: (12)                        new_line = line.copy()
SNGT_QHENOMENOLOGY_603: (12)                        new_line.shift(axis2.n2p(x) - axis2.n2p(0))
SNGT_QHENOMENOLOGY_604: (12)                        if i % (1 + ratio) == 0:
SNGT_QHENOMENOLOGY_605: (16)                            lines1.add(new_line)
SNGT_QHENOMENOLOGY_606: (12)                        else:
SNGT_QHENOMENOLOGY_607: (16)                            lines2.add(new_line)
SNGT_QHENOMENOLOGY_608: (8)                     return lines1, lines2
SNGT_QHENOMENOLOGY_609: (4)                 def get_x_unit_size(self) -> float:
SNGT_QHENOMENOLOGY_610: (8)                     return self.get_x_axis().get_unit_size()
SNGT_QHENOMENOLOGY_611: (4)                 def get_y_unit_size(self) -> list:
SNGT_QHENOMENOLOGY_612: (8)                     return self.get_x_axis().get_unit_size()
SNGT_QHENOMENOLOGY_613: (4)                 def get_axes(self) -> VGroup:
SNGT_QHENOMENOLOGY_614: (8)                     return self.axes
SNGT_QHENOMENOLOGY_615: (4)                 def get_vector(self, coords: Iterable[float], **kwargs)
-> Arrow:
SNGT_QHENOMENOLOGY_616: (8)                     kwargs["buff"] = 0
SNGT_QHENOMENOLOGY_617: (8)                     return Arrow(self.c2p(0, 0), self.c2p(*coords),
**kwargs)
SNGT_QHENOMENOLOGY_618: (4)                 def prepare_for_nonlinear_transform(self,
num_inserted_curves: int = 50) -> Self:
SNGT_QHENOMENOLOGY_619: (8)                     for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_620: (12)                        num_curves = mob.get_num_curves()
SNGT_QHENOMENOLOGY_621: (12)                        if num_inserted_curves > num_curves:
SNGT_QHENOMENOLOGY_622: (16)                            mob.insert_n_curves(num_inserted_curves -
num_curves)
SNGT_QHENOMENOLOGY_623: (12)                        mob.make_smooth_after_applying_functions = True
SNGT_QHENOMENOLOGY_624: (8)                     return self
SNGT_QHENOMENOLOGY_625: (0)             class ComplexPlane(NumberPlane):
SNGT_QHENOMENOLOGY_626: (4)                 def number_to_point(self, number: complex | float) ->
Vect3:
SNGT_QHENOMENOLOGY_627: (8)                     number = complex(number)
SNGT_QHENOMENOLOGY_628: (8)                     return self.coords_to_point(number.real,
number.imag)
SNGT_QHENOMENOLOGY_629: (4)                 def n2p(self, number: complex | float) -> Vect3:
SNGT_QHENOMENOLOGY_630: (8)                     return self.number_to_point(number)
SNGT_QHENOMENOLOGY_631: (4)                 def point_to_number(self, point: Vect3) -> complex:
SNGT_QHENOMENOLOGY_632: (8)                     x, y = self.point_to_coords(point)
SNGT_QHENOMENOLOGY_633: (8)                     return complex(x, y)
SNGT_QHENOMENOLOGY_634: (4)                 def p2n(self, point: Vect3) -> complex:
SNGT_QHENOMENOLOGY_635: (8)                     return self.point_to_number(point)
SNGT_QHENOMENOLOGY_636: (4)                 def get_default_coordinate_values(
SNGT_QHENOMENOLOGY_637: (8)                     self,
SNGT_QHENOMENOLOGY_638: (8)                     skip_first: bool = True
SNGT_QHENOMENOLOGY_639: (4)                 ) -> list[complex]:
SNGT_QHENOMENOLOGY_640: (8)                     x_numbers = self.get_x_axis().get_tick_range()[1:]
SNGT_QHENOMENOLOGY_641: (8)                     y_numbers = self.get_y_axis().get_tick_range()[1:]
SNGT_QHENOMENOLOGY_642: (8)                     y_numbers = [complex(0, y) for y in y_numbers if y
!= 0]
SNGT_QHENOMENOLOGY_643: (8)                     return [*x_numbers, *y_numbers]
```

```
SNGT_QHENOMENOLOGY_644: (4)                 def add_coordinate_labels(
SNGT_QHENOMENOLOGY_645: (8)                     self,
SNGT_QHENOMENOLOGY_646: (8)                     numbers: list[complex] | None = None,
SNGT_QHENOMENOLOGY_647: (8)                     skip_first: bool = True,
SNGT_QHENOMENOLOGY_648: (8)                     font_size: int = 36,
SNGT_QHENOMENOLOGY_649: (8)                     **kwargs
SNGT_QHENOMENOLOGY_650: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_651: (8)                     if numbers is None:
SNGT_QHENOMENOLOGY_652: (12)                        numbers =
self.get_default_coordinate_values(skip_first)
SNGT_QHENOMENOLOGY_653: (8)                     self.coordinate_labels = VGroup()
SNGT_QHENOMENOLOGY_654: (8)                     for number in numbers:
SNGT_QHENOMENOLOGY_655: (12)                        z = complex(number)
SNGT_QHENOMENOLOGY_656: (12)                        if abs(z.imag) > abs(z.real):
SNGT_QHENOMENOLOGY_657: (16)                            axis = self.get_y_axis()
SNGT_QHENOMENOLOGY_658: (16)                            value = z.imag
SNGT_QHENOMENOLOGY_659: (16)                            kwargs["unit_tex"] = "i"
SNGT_QHENOMENOLOGY_660: (12)                        else:
SNGT_QHENOMENOLOGY_661: (16)                            axis = self.get_x_axis()
SNGT_QHENOMENOLOGY_662: (16)                            value = z.real
SNGT_QHENOMENOLOGY_663: (12)                        number_mob = axis.get_number_mobject(value,
font_size=font_size, **kwargs)
SNGT_QHENOMENOLOGY_664: (12)                        if z.imag == -1:
SNGT_QHENOMENOLOGY_665: (16)                            number_mob.remove(number_mob[1])
SNGT_QHENOMENOLOGY_666: (16)                            number_mob[0].next_to(
SNGT_QHENOMENOLOGY_667: (20)                                number_mob[1], LEFT,
SNGT_QHENOMENOLOGY_668: (20)                                buff=number_mob[0].get_width() / 4
SNGT_QHENOMENOLOGY_669: (16)                            )
SNGT_QHENOMENOLOGY_670: (12)                        self.coordinate_labels.add(number_mob)
SNGT_QHENOMENOLOGY_671: (8)                     self.add(self.coordinate_labels)
SNGT_QHENOMENOLOGY_672: (8)                     return self
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 44 - mobject_update_utils.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import inspect
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import RIGHT
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_7: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                   from typing import Callable
SNGT_QHENOMENOLOGY_10: (4)                  import numpy as np
SNGT_QHENOMENOLOGY_11: (4)                  from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_12: (0)              def assert_is_mobject_method(method):
SNGT_QHENOMENOLOGY_13: (4)                  assert inspect.ismethod(method)
SNGT_QHENOMENOLOGY_14: (4)                  mobject = method.__self__
SNGT_QHENOMENOLOGY_15: (4)                  assert isinstance(mobject, Mobject)
SNGT_QHENOMENOLOGY_16: (0)              def always(method, *args, **kwargs):
SNGT_QHENOMENOLOGY_17: (4)                  assert_is_mobject_method(method)
SNGT_QHENOMENOLOGY_18: (4)                  mobject = method.__self__
SNGT_QHENOMENOLOGY_19: (4)                  func = method.__func__
SNGT_QHENOMENOLOGY_20: (4)                  mobject.add_updater(lambda m: func(m, *args, **kwargs))
SNGT_QHENOMENOLOGY_21: (4)                  return mobject
SNGT_QHENOMENOLOGY_22: (0)              def f_always(method, *arg_generators, **kwargs):
SNGT_QHENOMENOLOGY_23: (4)                  """
SNGT_QHENOMENOLOGY_24: (4)                  More functional version of always, where instead
SNGT_QHENOMENOLOGY_25: (4)                  of taking in args, it takes in functions which output
SNGT_QHENOMENOLOGY_26: (4)                  the relevant arguments.
SNGT_QHENOMENOLOGY_27: (4)                  """
SNGT_QHENOMENOLOGY_28: (4)                  assert_is_mobject_method(method)
SNGT_QHENOMENOLOGY_29: (4)                  mobject = method.__self__
SNGT_QHENOMENOLOGY_30: (4)                  func = method.__func__
SNGT_QHENOMENOLOGY_31: (4)                  def updater(mob):
SNGT_QHENOMENOLOGY_32: (8)                      args = [
SNGT_QHENOMENOLOGY_33: (12)                         arg_generator()
```

```
SNGT_QHENOMENOLOGY_34: (12)                         for arg_generator in arg_generators
SNGT_QHENOMENOLOGY_35: (8)                     ]
SNGT_QHENOMENOLOGY_36: (8)                     func(mob, *args, **kwargs)
SNGT_QHENOMENOLOGY_37: (4)                 mobject.add_updater(updater)
SNGT_QHENOMENOLOGY_38: (4)                 return mobject
SNGT_QHENOMENOLOGY_39: (0)             def always_redraw(func: Callable[..., Mobject], *args,
**kwargs) -> Mobject:
SNGT_QHENOMENOLOGY_40: (4)                 mob = func(*args, **kwargs)
SNGT_QHENOMENOLOGY_41: (4)                 mob.add_updater(lambda m: mob.become(func(*args,
**kwargs)))
SNGT_QHENOMENOLOGY_42: (4)                 return mob
SNGT_QHENOMENOLOGY_43: (0)             def always_shift(
SNGT_QHENOMENOLOGY_44: (4)                 mobject: Mobject,
SNGT_QHENOMENOLOGY_45: (4)                 direction: np.ndarray = RIGHT,
SNGT_QHENOMENOLOGY_46: (4)                 rate: float = 0.1
SNGT_QHENOMENOLOGY_47: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_48: (4)                 mobject.add_updater(
SNGT_QHENOMENOLOGY_49: (8)                     lambda m, dt: m.shift(dt * rate * direction)
SNGT_QHENOMENOLOGY_50: (4)                 )
SNGT_QHENOMENOLOGY_51: (4)                 return mobject
SNGT_QHENOMENOLOGY_52: (0)             def always_rotate(
SNGT_QHENOMENOLOGY_53: (4)                 mobject: Mobject,
SNGT_QHENOMENOLOGY_54: (4)                 rate: float = 20 * DEG,
SNGT_QHENOMENOLOGY_55: (4)                 **kwargs
SNGT_QHENOMENOLOGY_56: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_57: (4)                 mobject.add_updater(
SNGT_QHENOMENOLOGY_58: (8)                     lambda m, dt: m.rotate(dt * rate, **kwargs)
SNGT_QHENOMENOLOGY_59: (4)                 )
SNGT_QHENOMENOLOGY_60: (4)                 return mobject
SNGT_QHENOMENOLOGY_61: (0)             def turn_animation_into_updater(
SNGT_QHENOMENOLOGY_62: (4)                 animation: Animation,
SNGT_QHENOMENOLOGY_63: (4)                 cycle: bool = False,
SNGT_QHENOMENOLOGY_64: (4)                 **kwargs
SNGT_QHENOMENOLOGY_65: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_66: (4)                 """
SNGT_QHENOMENOLOGY_67: (4)                 Add an updater to the animation's mobject which applies
SNGT_QHENOMENOLOGY_68: (4)                 the interpolation and update functions of the animation
SNGT_QHENOMENOLOGY_69: (4)                 If cycle is True, this repeats over and over.
Otherwise,
SNGT_QHENOMENOLOGY_70: (4)                 the updater will be popped uplon completion
SNGT_QHENOMENOLOGY_71: (4)                 """
SNGT_QHENOMENOLOGY_72: (4)                 mobject = animation.mobject
SNGT_QHENOMENOLOGY_73: (4)                 animation.update_rate_info(**kwargs)
SNGT_QHENOMENOLOGY_74: (4)                 animation.suspend_mobject_updating = False
SNGT_QHENOMENOLOGY_75: (4)                 animation.begin()
SNGT_QHENOMENOLOGY_76: (4)                 animation.total_time = 0
SNGT_QHENOMENOLOGY_77: (4)                 def update(m, dt):
SNGT_QHENOMENOLOGY_78: (8)                     run_time = animation.get_run_time()
SNGT_QHENOMENOLOGY_79: (8)                     time_ratio = animation.total_time / run_time
SNGT_QHENOMENOLOGY_80: (8)                     if cycle:
SNGT_QHENOMENOLOGY_81: (12)                        alpha = time_ratio % 1
SNGT_QHENOMENOLOGY_82: (8)                     else:
SNGT_QHENOMENOLOGY_83: (12)                        alpha = clip(time_ratio, 0, 1)
SNGT_QHENOMENOLOGY_84: (12)                        if alpha >= 1:
SNGT_QHENOMENOLOGY_85: (16)                            animation.finish()
SNGT_QHENOMENOLOGY_86: (16)                            m.remove_updater(update)
SNGT_QHENOMENOLOGY_87: (16)                            return
SNGT_QHENOMENOLOGY_88: (8)                     animation.interpolate(alpha)
SNGT_QHENOMENOLOGY_89: (8)                     animation.update_mobjects(dt)
SNGT_QHENOMENOLOGY_90: (8)                     animation.total_time += dt
SNGT_QHENOMENOLOGY_91: (4)                 mobject.add_updater(update)
SNGT_QHENOMENOLOGY_92: (4)                 return mobject
SNGT_QHENOMENOLOGY_93: (0)             def cycle_animation(animation: Animation, **kwargs) ->
Mobject:
SNGT_QHENOMENOLOGY_94: (4)                 return turn_animation_into_updater(
SNGT_QHENOMENOLOGY_95: (8)                     animation, cycle=True, **kwargs
SNGT_QHENOMENOLOGY_96: (4)                 )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 45 - transform_matching_parts.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import itertools as it
SNGT_QHENOMENOLOGY_3: (0)              from difflib import SequenceMatcher
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.animation.composition import AnimationGroup
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.animation.fading import FadeInFromPoint
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.animation.fading import FadeOutToPoint
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.animation.transform import Transform
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.mobject.svg.string_mobject import
StringMobject
SNGT_QHENOMENOLOGY_11: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_12: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_13: (4)                 from typing import Iterable
SNGT_QHENOMENOLOGY_14: (4)                 from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_15: (0)             class TransformMatchingParts(AnimationGroup):
SNGT_QHENOMENOLOGY_16: (4)                 def __init__(
SNGT_QHENOMENOLOGY_17: (8)                     self,
SNGT_QHENOMENOLOGY_18: (8)                     source: Mobject,
SNGT_QHENOMENOLOGY_19: (8)                     target: Mobject,
SNGT_QHENOMENOLOGY_20: (8)                     matched_pairs: Iterable[tuple[Mobject, Mobject]] =
[],
SNGT_QHENOMENOLOGY_21: (8)                     match_animation: type = Transform,
SNGT_QHENOMENOLOGY_22: (8)                     mismatch_animation: type = Transform,
SNGT_QHENOMENOLOGY_23: (8)                     run_time: float = 2,
SNGT_QHENOMENOLOGY_24: (8)                     lag_ratio: float = 0,
SNGT_QHENOMENOLOGY_25: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_26: (4)                 ):
SNGT_QHENOMENOLOGY_27: (8)                     self.source = source
SNGT_QHENOMENOLOGY_28: (8)                     self.target = target
SNGT_QHENOMENOLOGY_29: (8)                     self.match_animation = match_animation
SNGT_QHENOMENOLOGY_30: (8)                     self.mismatch_animation = mismatch_animation
SNGT_QHENOMENOLOGY_31: (8)                     self.anim_config = dict(**kwargs)
SNGT_QHENOMENOLOGY_32: (8)                     self.source_pieces =
source.family_members_with_points()
SNGT_QHENOMENOLOGY_33: (8)                     self.target_pieces =
target.family_members_with_points()
SNGT_QHENOMENOLOGY_34: (8)                     self.anims = []
SNGT_QHENOMENOLOGY_35: (8)                     for pair in matched_pairs:
SNGT_QHENOMENOLOGY_36: (12)                        self.add_transform(*pair)
SNGT_QHENOMENOLOGY_37: (8)                     for pair in
self.find_pairs_with_matching_shapes(self.source_pieces, self.target_pieces):
SNGT_QHENOMENOLOGY_38: (12)                        self.add_transform(*pair)
SNGT_QHENOMENOLOGY_39: (8)                     for source_piece in self.source_pieces:
SNGT_QHENOMENOLOGY_40: (12)                        if any([source_piece in
anim.mobject.get_family() for anim in self.anims]):
SNGT_QHENOMENOLOGY_41: (16)                            continue
SNGT_QHENOMENOLOGY_42: (12)                        self.anims.append(FadeOutToPoint(
SNGT_QHENOMENOLOGY_43: (16)                            source_piece, target.get_center(),
SNGT_QHENOMENOLOGY_44: (16)                            **self.anim_config
SNGT_QHENOMENOLOGY_45: (12)                        ))
SNGT_QHENOMENOLOGY_46: (8)                     for target_piece in self.target_pieces:
SNGT_QHENOMENOLOGY_47: (12)                        if any([target_piece in
anim.mobject.get_family() for anim in self.anims]):
SNGT_QHENOMENOLOGY_48: (16)                            continue
SNGT_QHENOMENOLOGY_49: (12)                        self.anims.append(FadeInFromPoint(
SNGT_QHENOMENOLOGY_50: (16)                            target_piece, source.get_center(),
SNGT_QHENOMENOLOGY_51: (16)                            **self.anim_config
SNGT_QHENOMENOLOGY_52: (12)                        ))
SNGT_QHENOMENOLOGY_53: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_54: (12)                        *self.anims,
SNGT_QHENOMENOLOGY_55: (12)                        run_time=run_time,
SNGT_QHENOMENOLOGY_56: (12)                        lag_ratio=lag_ratio,
SNGT_QHENOMENOLOGY_57: (8)                     )
SNGT_QHENOMENOLOGY_58: (4)                 def add_transform(
```

```
SNGT_QHENOMENOLOGY_59: (8)                      self,
SNGT_QHENOMENOLOGY_60: (8)                      source: Mobject,
SNGT_QHENOMENOLOGY_61: (8)                      target: Mobject,
SNGT_QHENOMENOLOGY_62: (4)                  ):
SNGT_QHENOMENOLOGY_63: (8)                      new_source_pieces =
source.family_members_with_points()
SNGT_QHENOMENOLOGY_64: (8)                      new_target_pieces =
target.family_members_with_points()
SNGT_QHENOMENOLOGY_65: (8)                      if len(new_source_pieces) == 0 or
len(new_target_pieces) == 0:
SNGT_QHENOMENOLOGY_66: (12)                         return
SNGT_QHENOMENOLOGY_67: (8)                      source_is_new = all(char in self.source_pieces for
char in new_source_pieces)
SNGT_QHENOMENOLOGY_68: (8)                      target_is_new = all(char in self.target_pieces for
char in new_target_pieces)
SNGT_QHENOMENOLOGY_69: (8)                      if not source_is_new or not target_is_new:
SNGT_QHENOMENOLOGY_70: (12)                         return
SNGT_QHENOMENOLOGY_71: (8)                      transform_type = self.mismatch_animation
SNGT_QHENOMENOLOGY_72: (8)                      if source.has_same_shape_as(target):
SNGT_QHENOMENOLOGY_73: (12)                         transform_type = self.match_animation
SNGT_QHENOMENOLOGY_74: (8)                      self.anims.append(transform_type(source, target,
**self.anim_config))
SNGT_QHENOMENOLOGY_75: (8)                      for char in new_source_pieces:
SNGT_QHENOMENOLOGY_76: (12)                         self.source_pieces.remove(char)
SNGT_QHENOMENOLOGY_77: (8)                      for char in new_target_pieces:
SNGT_QHENOMENOLOGY_78: (12)                         self.target_pieces.remove(char)
SNGT_QHENOMENOLOGY_79: (4)                  def find_pairs_with_matching_shapes(
SNGT_QHENOMENOLOGY_80: (8)                      self,
SNGT_QHENOMENOLOGY_81: (8)                      chars1: list[Mobject],
SNGT_QHENOMENOLOGY_82: (8)                      chars2: list[Mobject]
SNGT_QHENOMENOLOGY_83: (4)                  ) -> list[tuple[Mobject, Mobject]]:
SNGT_QHENOMENOLOGY_84: (8)                      result = []
SNGT_QHENOMENOLOGY_85: (8)                      for char1, char2 in it.product(chars1, chars2):
SNGT_QHENOMENOLOGY_86: (12)                         if char1.has_same_shape_as(char2):
SNGT_QHENOMENOLOGY_87: (16)                             result.append((char1, char2))
SNGT_QHENOMENOLOGY_88: (8)                      return result
SNGT_QHENOMENOLOGY_89: (4)                  def clean_up_from_scene(self, scene: Scene) -> None:
SNGT_QHENOMENOLOGY_90: (8)                      super().clean_up_from_scene(scene)
SNGT_QHENOMENOLOGY_91: (8)                      scene.remove(self.mobject)
SNGT_QHENOMENOLOGY_92: (8)                      scene.add(self.target)
SNGT_QHENOMENOLOGY_93: (0)              class TransformMatchingShapes(TransformMatchingParts):
SNGT_QHENOMENOLOGY_94: (4)                  """Alias for TransformMatchingParts"""
SNGT_QHENOMENOLOGY_95: (4)                  pass
SNGT_QHENOMENOLOGY_96: (0)              class TransformMatchingStrings(TransformMatchingParts):
SNGT_QHENOMENOLOGY_97: (4)                  def __init__(
SNGT_QHENOMENOLOGY_98: (8)                      self,
SNGT_QHENOMENOLOGY_99: (8)                      source: StringMobject,
SNGT_QHENOMENOLOGY_100: (8)                     target: StringMobject,
SNGT_QHENOMENOLOGY_101: (8)                     matched_keys: Iterable[str] = [],
SNGT_QHENOMENOLOGY_102: (8)                     key_map: dict[str, str] = dict(),
SNGT_QHENOMENOLOGY_103: (8)                     matched_pairs: Iterable[tuple[VMobject, VMobject]]
= [],
SNGT_QHENOMENOLOGY_104: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_105: (4)                 ):
SNGT_QHENOMENOLOGY_106: (8)                     matched_pairs = [
SNGT_QHENOMENOLOGY_107: (12)                        *matched_pairs,
SNGT_QHENOMENOLOGY_108: (12)                        *self.matching_blocks(source, target,
matched_keys, key_map),
SNGT_QHENOMENOLOGY_109: (8)                     ]
SNGT_QHENOMENOLOGY_110: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_111: (12)                        source, target,
SNGT_QHENOMENOLOGY_112: (12)                        matched_pairs=matched_pairs,
SNGT_QHENOMENOLOGY_113: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_114: (8)                     )
SNGT_QHENOMENOLOGY_115: (4)                 def matching_blocks(
SNGT_QHENOMENOLOGY_116: (8)                     self,
SNGT_QHENOMENOLOGY_117: (8)                     source: StringMobject,
SNGT_QHENOMENOLOGY_118: (8)                     target: StringMobject,
SNGT_QHENOMENOLOGY_119: (8)                     matched_keys: Iterable[str],
```

```
SNGT_QHENOMENOLOGY_120: (8)                     key_map: dict[str, str]
SNGT_QHENOMENOLOGY_121: (4)                 ) -> list[tuple[VMobject, VMobject]]:
SNGT_QHENOMENOLOGY_122: (8)                     syms1 = source.get_symbol_substrings()
SNGT_QHENOMENOLOGY_123: (8)                     syms2 = target.get_symbol_substrings()
SNGT_QHENOMENOLOGY_124: (8)                     counts1 = list(map(source.substr_to_path_count,
syms1))
SNGT_QHENOMENOLOGY_125: (8)                     counts2 = list(map(target.substr_to_path_count,
syms2))
SNGT_QHENOMENOLOGY_126: (8)                     blocks = [(source[key], target[key]) for key in
matched_keys]
SNGT_QHENOMENOLOGY_127: (8)                     blocks += [(source[key1], target[key2]) for key1,
key2 in key_map.items()]
SNGT_QHENOMENOLOGY_128: (8)                     for sub_source, sub_target in blocks:
SNGT_QHENOMENOLOGY_129: (12)                        for i in range(len(syms1)):
SNGT_QHENOMENOLOGY_130: (16)                            if source[i] in
sub_source.family_members_with_points():
SNGT_QHENOMENOLOGY_131: (20)                                syms1[i] = "Null1"
SNGT_QHENOMENOLOGY_132: (12)                        for j in range(len(syms2)):
SNGT_QHENOMENOLOGY_133: (16)                            if target[j] in
sub_target.family_members_with_points():
SNGT_QHENOMENOLOGY_134: (20)                                syms2[j] = "Null2"
SNGT_QHENOMENOLOGY_135: (8)                     while True:
SNGT_QHENOMENOLOGY_136: (12)                        matcher = SequenceMatcher(None, syms1, syms2)
SNGT_QHENOMENOLOGY_137: (12)                        match = matcher.find_longest_match(0,
len(syms1), 0, len(syms2))
SNGT_QHENOMENOLOGY_138: (12)                        if match.size == 0:
SNGT_QHENOMENOLOGY_139: (16)                            break
SNGT_QHENOMENOLOGY_140: (12)                        i1 = sum(counts1[:match.a])
SNGT_QHENOMENOLOGY_141: (12)                        i2 = sum(counts2[:match.b])
SNGT_QHENOMENOLOGY_142: (12)                        size = sum(counts1[match.a:match.a +
match.size])
SNGT_QHENOMENOLOGY_143: (12)                        blocks.append((source[i1:i1 + size],
target[i2:i2 + size]))
SNGT_QHENOMENOLOGY_144: (12)                        for i in range(match.size):
SNGT_QHENOMENOLOGY_145: (16)                            syms1[match.a + i] = "Null1"
SNGT_QHENOMENOLOGY_146: (16)                            syms2[match.b + i] = "Null2"
SNGT_QHENOMENOLOGY_147: (8)                     return blocks
SNGT_QHENOMENOLOGY_148: (0)             class TransformMatchingTex(TransformMatchingStrings):
SNGT_QHENOMENOLOGY_149: (4)                 """Alias for TransformMatchingStrings"""
SNGT_QHENOMENOLOGY_150: (4)                 pass
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 46 - brace.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import math
SNGT_QHENOMENOLOGY_3: (0)               import copy
SNGT_QHENOMENOLOGY_4: (0)               import numpy as np
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import
DEFAULT_MOBJECT_TO_MOBJECT_BUFF, SMALL_BUFF
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import DOWN, LEFT, ORIGIN, RIGHT,
DL, DR, UL
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import PI
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.animation.composition import AnimationGroup
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.animation.fading import FadeIn
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.animation.growing import GrowFromCenter
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.svg.tex_mobject import TexText
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.mobject.svg.text_mobject import Text
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_18: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_19: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_20: (4)                  from typing import Iterable
```

```
SNGT_QHENOMENOLOGY_21: (4)                 from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_22: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_23: (4)                 from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_24: (0)             class Brace(Tex):
SNGT_QHENOMENOLOGY_25: (4)                 def __init__(
SNGT_QHENOMENOLOGY_26: (8)                     self,
SNGT_QHENOMENOLOGY_27: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_28: (8)                     direction: Vect3 = DOWN,
SNGT_QHENOMENOLOGY_29: (8)                     buff: float = 0.2,
SNGT_QHENOMENOLOGY_30: (8)                     tex_string: str = R"\underbrace{\qquad}",
SNGT_QHENOMENOLOGY_31: (8)                     **kwargs
SNGT_QHENOMENOLOGY_32: (4)                 ):
SNGT_QHENOMENOLOGY_33: (8)                     super().__init__(tex_string, **kwargs)
SNGT_QHENOMENOLOGY_34: (8)                     angle = -math.atan2(*direction[:2]) + PI
SNGT_QHENOMENOLOGY_35: (8)                     mobject.rotate(-angle, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_36: (8)                     left = mobject.get_corner(DL)
SNGT_QHENOMENOLOGY_37: (8)                     right = mobject.get_corner(DR)
SNGT_QHENOMENOLOGY_38: (8)                     target_width = right[0] - left[0]
SNGT_QHENOMENOLOGY_39: (8)                     self.tip_point_index =
np.argmin(self.get_all_points()[:, 1])
SNGT_QHENOMENOLOGY_40: (8)                     self.set_initial_width(target_width)
SNGT_QHENOMENOLOGY_41: (8)                     self.shift(left - self.get_corner(UL) + buff *
DOWN)
SNGT_QHENOMENOLOGY_42: (8)                     for mob in mobject, self:
SNGT_QHENOMENOLOGY_43: (12)                        mob.rotate(angle, about_point=ORIGIN)
SNGT_QHENOMENOLOGY_44: (4)                 def set_initial_width(self, width: float):
SNGT_QHENOMENOLOGY_45: (8)                     width_diff = width - self.get_width()
SNGT_QHENOMENOLOGY_46: (8)                     if width_diff > 0:
SNGT_QHENOMENOLOGY_47: (12)                        for tip, rect, vect in [(self[0], self[1],
RIGHT), (self[5], self[4], LEFT)]:
SNGT_QHENOMENOLOGY_48: (16)                            rect.set_width(
SNGT_QHENOMENOLOGY_49: (20)                                width_diff / 2 + rect.get_width(),
SNGT_QHENOMENOLOGY_50: (20)                                about_edge=vect, stretch=True
SNGT_QHENOMENOLOGY_51: (16)                            )
SNGT_QHENOMENOLOGY_52: (16)                            tip.shift(-width_diff / 2 * vect)
SNGT_QHENOMENOLOGY_53: (8)                     else:
SNGT_QHENOMENOLOGY_54: (12)                        self.set_width(width, stretch=True)
SNGT_QHENOMENOLOGY_55: (8)                     return self
SNGT_QHENOMENOLOGY_56: (4)                 def put_at_tip(
SNGT_QHENOMENOLOGY_57: (8)                     self,
SNGT_QHENOMENOLOGY_58: (8)                     mob: Mobject,
SNGT_QHENOMENOLOGY_59: (8)                     use_next_to: bool = True,
SNGT_QHENOMENOLOGY_60: (8)                     **kwargs
SNGT_QHENOMENOLOGY_61: (4)                 ):
SNGT_QHENOMENOLOGY_62: (8)                     if use_next_to:
SNGT_QHENOMENOLOGY_63: (12)                        mob.next_to(
SNGT_QHENOMENOLOGY_64: (16)                            self.get_tip(),
SNGT_QHENOMENOLOGY_65: (16)                            np.round(self.get_direction()),
SNGT_QHENOMENOLOGY_66: (16)                            **kwargs
SNGT_QHENOMENOLOGY_67: (12)                        )
SNGT_QHENOMENOLOGY_68: (8)                     else:
SNGT_QHENOMENOLOGY_69: (12)                        mob.move_to(self.get_tip())
SNGT_QHENOMENOLOGY_70: (12)                        buff = kwargs.get("buff",
DEFAULT_MOBJECT_TO_MOBJECT_BUFF)
SNGT_QHENOMENOLOGY_71: (12)                        shift_distance = mob.get_width() / 2.0 + buff
SNGT_QHENOMENOLOGY_72: (12)                        mob.shift(self.get_direction() *
shift_distance)
SNGT_QHENOMENOLOGY_73: (8)                     return self
SNGT_QHENOMENOLOGY_74: (4)                 def get_text(self, text: str, **kwargs) -> Text:
SNGT_QHENOMENOLOGY_75: (8)                     buff = kwargs.pop("buff", SMALL_BUFF)
SNGT_QHENOMENOLOGY_76: (8)                     text_mob = Text(text, **kwargs)
SNGT_QHENOMENOLOGY_77: (8)                     self.put_at_tip(text_mob, buff=buff)
SNGT_QHENOMENOLOGY_78: (8)                     return text_mob
SNGT_QHENOMENOLOGY_79: (4)                 def get_tex(self, *tex: str, **kwargs) -> Tex:
SNGT_QHENOMENOLOGY_80: (8)                     buff = kwargs.pop("buff", SMALL_BUFF)
SNGT_QHENOMENOLOGY_81: (8)                     tex_mob = Tex(*tex, **kwargs)
SNGT_QHENOMENOLOGY_82: (8)                     self.put_at_tip(tex_mob, buff=buff)
SNGT_QHENOMENOLOGY_83: (8)                     return tex_mob
SNGT_QHENOMENOLOGY_84: (4)                 def get_tip(self) -> np.ndarray:
```

```
SNGT_QHENOMENOLOGY_85: (8)                      return self.get_all_points()[self.tip_point_index]
SNGT_QHENOMENOLOGY_86: (4)                  def get_direction(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_87: (8)                      vect = self.get_tip() - self.get_center()
SNGT_QHENOMENOLOGY_88: (8)                      return vect / get_norm(vect)
SNGT_QHENOMENOLOGY_89: (0)              class BraceLabel(VMobject):
SNGT_QHENOMENOLOGY_90: (4)                  label_constructor: type = Tex
SNGT_QHENOMENOLOGY_91: (4)                  def __init__(
SNGT_QHENOMENOLOGY_92: (8)                      self,
SNGT_QHENOMENOLOGY_93: (8)                      obj: VMobject | list[VMobject],
SNGT_QHENOMENOLOGY_94: (8)                      text: str | Iterable[str],
SNGT_QHENOMENOLOGY_95: (8)                      brace_direction: np.ndarray = DOWN,
SNGT_QHENOMENOLOGY_96: (8)                      label_scale: float = 1.0,
SNGT_QHENOMENOLOGY_97: (8)                      label_buff: float =
DEFAULT_MOBJECT_TO_MOBJECT_BUFF,
SNGT_QHENOMENOLOGY_98: (8)                      **kwargs
SNGT_QHENOMENOLOGY_99: (4)                  ) -> None:
SNGT_QHENOMENOLOGY_100: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_101: (8)                     self.brace_direction = brace_direction
SNGT_QHENOMENOLOGY_102: (8)                     self.label_scale = label_scale
SNGT_QHENOMENOLOGY_103: (8)                     self.label_buff = label_buff
SNGT_QHENOMENOLOGY_104: (8)                     if isinstance(obj, list):
SNGT_QHENOMENOLOGY_105: (12)                        obj = VGroup(*obj)
SNGT_QHENOMENOLOGY_106: (8)                     self.brace = Brace(obj, brace_direction, **kwargs)
SNGT_QHENOMENOLOGY_107: (8)                     self.label = self.label_constructor(*listify(text),
**kwargs)
SNGT_QHENOMENOLOGY_108: (8)                     self.label.scale(self.label_scale)
SNGT_QHENOMENOLOGY_109: (8)                     self.brace.put_at_tip(self.label,
buff=self.label_buff)
SNGT_QHENOMENOLOGY_110: (8)                     self.set_submobjects([self.brace, self.label])
SNGT_QHENOMENOLOGY_111: (4)                 def creation_anim(
SNGT_QHENOMENOLOGY_112: (8)                     self,
SNGT_QHENOMENOLOGY_113: (8)                     label_anim: Animation = FadeIn,
SNGT_QHENOMENOLOGY_114: (8)                     brace_anim: Animation = GrowFromCenter
SNGT_QHENOMENOLOGY_115: (4)                 ) -> AnimationGroup:
SNGT_QHENOMENOLOGY_116: (8)                     return AnimationGroup(brace_anim(self.brace),
label_anim(self.label))
SNGT_QHENOMENOLOGY_117: (4)                 def shift_brace(self, obj: VMobject | list[VMobject],
**kwargs):
SNGT_QHENOMENOLOGY_118: (8)                     if isinstance(obj, list):
SNGT_QHENOMENOLOGY_119: (12)                        obj = VMobject(*obj)
SNGT_QHENOMENOLOGY_120: (8)                     self.brace = Brace(obj, self.brace_direction,
**kwargs)
SNGT_QHENOMENOLOGY_121: (8)                     self.brace.put_at_tip(self.label)
SNGT_QHENOMENOLOGY_122: (8)                     self.submobjects[0] = self.brace
SNGT_QHENOMENOLOGY_123: (8)                     return self
SNGT_QHENOMENOLOGY_124: (4)                 def change_label(self, *text: str, **kwargs):
SNGT_QHENOMENOLOGY_125: (8)                     self.label = self.label_constructor(*text,
**kwargs)
SNGT_QHENOMENOLOGY_126: (8)                     if self.label_scale != 1:
SNGT_QHENOMENOLOGY_127: (12)                        self.label.scale(self.label_scale)
SNGT_QHENOMENOLOGY_128: (8)                     self.brace.put_at_tip(self.label)
SNGT_QHENOMENOLOGY_129: (8)                     self.submobjects[1] = self.label
SNGT_QHENOMENOLOGY_130: (8)                     return self
SNGT_QHENOMENOLOGY_131: (4)                 def change_brace_label(self, obj: VMobject |
list[VMobject], *text: str):
SNGT_QHENOMENOLOGY_132: (8)                     self.shift_brace(obj)
SNGT_QHENOMENOLOGY_133: (8)                     self.change_label(*text)
SNGT_QHENOMENOLOGY_134: (8)                     return self
SNGT_QHENOMENOLOGY_135: (4)                 def copy(self):
SNGT_QHENOMENOLOGY_136: (8)                     copy_mobject = copy.copy(self)
SNGT_QHENOMENOLOGY_137: (8)                     copy_mobject.brace = self.brace.copy()
SNGT_QHENOMENOLOGY_138: (8)                     copy_mobject.label = self.label.copy()
SNGT_QHENOMENOLOGY_139: (8)                     copy_mobject.set_submobjects([copy_mobject.brace,
copy_mobject.label])
SNGT_QHENOMENOLOGY_140: (8)                     return copy_mobject
SNGT_QHENOMENOLOGY_141: (0)             class BraceText(BraceLabel):
SNGT_QHENOMENOLOGY_142: (4)                 label_constructor: type = TexText
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 47 - surface.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import moderngl
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import GREY
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import OUT
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.utils.bezier import integer_interpolate
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.images import
get_full_raster_image_path
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.space_ops import normalize_along_axis
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.space_ops import cross
SNGT_QHENOMENOLOGY_14: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_15: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_16: (4)                  from typing import Callable, Iterable, Sequence, Tuple
SNGT_QHENOMENOLOGY_17: (4)                  from manimlib.camera.camera import Camera
SNGT_QHENOMENOLOGY_18: (4)                  from manimlib.typing import ManimColor, Vect3,
Vect3Array, Self
SNGT_QHENOMENOLOGY_19: (0)              class Surface(Mobject):
SNGT_QHENOMENOLOGY_20: (4)                  render_primitive: int = moderngl.TRIANGLES
SNGT_QHENOMENOLOGY_21: (4)                  shader_folder: str = "surface"
SNGT_QHENOMENOLOGY_22: (4)                  data_dtype: np.dtype = np.dtype([
SNGT_QHENOMENOLOGY_23: (8)                      ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_24: (8)                      ('du_point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_25: (8)                      ('dv_point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_26: (8)                      ('rgba', np.float32, (4,)),
SNGT_QHENOMENOLOGY_27: (4)                  ])
SNGT_QHENOMENOLOGY_28: (4)                  pointlike_data_keys = ['point', 'du_point', 'dv_point']
SNGT_QHENOMENOLOGY_29: (4)                  def __init__(
SNGT_QHENOMENOLOGY_30: (8)                      self,
SNGT_QHENOMENOLOGY_31: (8)                      color: ManimColor = GREY,
SNGT_QHENOMENOLOGY_32: (8)                      shading: Tuple[float, float, float] = (0.3, 0.2,
0.4),
SNGT_QHENOMENOLOGY_33: (8)                      depth_test: bool = True,
SNGT_QHENOMENOLOGY_34: (8)                      u_range: Tuple[float, float] = (0.0, 1.0),
SNGT_QHENOMENOLOGY_35: (8)                      v_range: Tuple[float, float] = (0.0, 1.0),
SNGT_QHENOMENOLOGY_36: (8)                      resolution: Tuple[int, int] = (101, 101),
SNGT_QHENOMENOLOGY_37: (8)                      prefered_creation_axis: int = 1,
SNGT_QHENOMENOLOGY_38: (8)                      epsilon: float = 1e-4,
SNGT_QHENOMENOLOGY_39: (8)                      **kwargs
SNGT_QHENOMENOLOGY_40: (4)                  ):
SNGT_QHENOMENOLOGY_41: (8)                      self.u_range = u_range
SNGT_QHENOMENOLOGY_42: (8)                      self.v_range = v_range
SNGT_QHENOMENOLOGY_43: (8)                      self.resolution = resolution
SNGT_QHENOMENOLOGY_44: (8)                      self.prefered_creation_axis =
prefered_creation_axis
SNGT_QHENOMENOLOGY_45: (8)                      self.epsilon = epsilon
SNGT_QHENOMENOLOGY_46: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_47: (12)                         **kwargs,
SNGT_QHENOMENOLOGY_48: (12)                         color=color,
SNGT_QHENOMENOLOGY_49: (12)                         shading=shading,
SNGT_QHENOMENOLOGY_50: (12)                         depth_test=depth_test,
SNGT_QHENOMENOLOGY_51: (8)                      )
SNGT_QHENOMENOLOGY_52: (8)                      self.compute_triangle_indices()
SNGT_QHENOMENOLOGY_53: (4)                  def uv_func(self, u: float, v: float) -> tuple[float,
float, float]:
SNGT_QHENOMENOLOGY_54: (8)                      return (u, v, 0.0)
SNGT_QHENOMENOLOGY_55: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_56: (4)                  def init_points(self):
SNGT_QHENOMENOLOGY_57: (8)                      dim = self.dim
SNGT_QHENOMENOLOGY_58: (8)                      nu, nv = self.resolution
SNGT_QHENOMENOLOGY_59: (8)                      u_range = np.linspace(*self.u_range, nu)
SNGT_QHENOMENOLOGY_60: (8)                      v_range = np.linspace(*self.v_range, nv)
```

```
SNGT_QHENOMENOLOGY_61: (8)                      uv_grid = np.array([[[u, v] for v in v_range] for u
in u_range])
SNGT_QHENOMENOLOGY_62: (8)                      uv_plus_du = uv_grid.copy()
SNGT_QHENOMENOLOGY_63: (8)                      uv_plus_du[:, :, 0] += self.epsilon
SNGT_QHENOMENOLOGY_64: (8)                      uv_plus_dv = uv_grid.copy()
SNGT_QHENOMENOLOGY_65: (8)                      uv_plus_dv[:, :, 1] += self.epsilon
SNGT_QHENOMENOLOGY_66: (8)                      points, du_points, dv_points = [
SNGT_QHENOMENOLOGY_67: (12)                         np.apply_along_axis(
SNGT_QHENOMENOLOGY_68: (16)                             lambda p: self.uv_func(*p), 2, grid
SNGT_QHENOMENOLOGY_69: (12)                         ).reshape((nu * nv, dim))
SNGT_QHENOMENOLOGY_70: (12)                         for grid in (uv_grid, uv_plus_du, uv_plus_dv)
SNGT_QHENOMENOLOGY_71: (8)                      ]
SNGT_QHENOMENOLOGY_72: (8)                      self.set_points(points)
SNGT_QHENOMENOLOGY_73: (8)                      self.data['du_point'][:] = du_points
SNGT_QHENOMENOLOGY_74: (8)                      self.data['dv_point'][:] = dv_points
SNGT_QHENOMENOLOGY_75: (4)                  def apply_points_function(self, *args, **kwargs) ->
Self:
SNGT_QHENOMENOLOGY_76: (8)                      super().apply_points_function(*args, **kwargs)
SNGT_QHENOMENOLOGY_77: (8)                      self.get_unit_normals()
SNGT_QHENOMENOLOGY_78: (8)                      return self
SNGT_QHENOMENOLOGY_79: (4)                  def compute_triangle_indices(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_80: (8)                      nu, nv = self.resolution
SNGT_QHENOMENOLOGY_81: (8)                      if nu == 0 or nv == 0:
SNGT_QHENOMENOLOGY_82: (12)                         self.triangle_indices = np.zeros(0, dtype=int)
SNGT_QHENOMENOLOGY_83: (12)                         return self.triangle_indices
SNGT_QHENOMENOLOGY_84: (8)                      index_grid = np.arange(nu * nv).reshape((nu, nv))
SNGT_QHENOMENOLOGY_85: (8)                      indices = np.zeros(6 * (nu - 1) * (nv - 1),
dtype=int)
SNGT_QHENOMENOLOGY_86: (8)                      indices[0::6] = index_grid[:-1, :-1].flatten()  #
Top left
SNGT_QHENOMENOLOGY_87: (8)                      indices[1::6] = index_grid[+1:, :-1].flatten()  #
Bottom left
SNGT_QHENOMENOLOGY_88: (8)                      indices[2::6] = index_grid[:-1, +1:].flatten()  #
Top right
SNGT_QHENOMENOLOGY_89: (8)                      indices[3::6] = index_grid[:-1, +1:].flatten()  #
Top right
SNGT_QHENOMENOLOGY_90: (8)                      indices[4::6] = index_grid[+1:, :-1].flatten()  #
Bottom left
SNGT_QHENOMENOLOGY_91: (8)                      indices[5::6] = index_grid[+1:, +1:].flatten()  #
Bottom right
SNGT_QHENOMENOLOGY_92: (8)                      self.triangle_indices = indices
SNGT_QHENOMENOLOGY_93: (8)                      return self.triangle_indices
SNGT_QHENOMENOLOGY_94: (4)                  def get_triangle_indices(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_95: (8)                      return self.triangle_indices
SNGT_QHENOMENOLOGY_96: (4)                  def get_unit_normals(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_97: (8)                      points = self.get_points()
SNGT_QHENOMENOLOGY_98: (8)                      crosses = cross(
SNGT_QHENOMENOLOGY_99: (12)                         self.data['du_point'] - points,
SNGT_QHENOMENOLOGY_100: (12)                        self.data['dv_point'] - points,
SNGT_QHENOMENOLOGY_101: (8)                     )
SNGT_QHENOMENOLOGY_102: (8)                     return normalize_along_axis(crosses, 1)
SNGT_QHENOMENOLOGY_103: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_104: (4)                 def pointwise_become_partial(
SNGT_QHENOMENOLOGY_105: (8)                     self,
SNGT_QHENOMENOLOGY_106: (8)                     smobject: "Surface",
SNGT_QHENOMENOLOGY_107: (8)                     a: float,
SNGT_QHENOMENOLOGY_108: (8)                     b: float,
SNGT_QHENOMENOLOGY_109: (8)                     axis: int | None = None
SNGT_QHENOMENOLOGY_110: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_111: (8)                     assert isinstance(smobject, Surface)
SNGT_QHENOMENOLOGY_112: (8)                     if axis is None:
SNGT_QHENOMENOLOGY_113: (12)                        axis = self.prefered_creation_axis
SNGT_QHENOMENOLOGY_114: (8)                     if a <= 0 and b >= 1:
SNGT_QHENOMENOLOGY_115: (12)                        self.match_points(smobject)
SNGT_QHENOMENOLOGY_116: (12)                        return self
SNGT_QHENOMENOLOGY_117: (8)                     nu, nv = smobject.resolution
SNGT_QHENOMENOLOGY_118: (8)                     self.data['point'][:] =
self.get_partial_points_array(
SNGT_QHENOMENOLOGY_119: (12)                        smobject.data['point'], a, b,
```

```
SNGT_QHENOMENOLOGY_120: (12)                        (nu, nv, 3),
SNGT_QHENOMENOLOGY_121: (12)                        axis=axis
SNGT_QHENOMENOLOGY_122: (8)                     )
SNGT_QHENOMENOLOGY_123: (8)                     return self
SNGT_QHENOMENOLOGY_124: (4)                 def get_partial_points_array(
SNGT_QHENOMENOLOGY_125: (8)                     self,
SNGT_QHENOMENOLOGY_126: (8)                     points: Vect3Array,
SNGT_QHENOMENOLOGY_127: (8)                     a: float,
SNGT_QHENOMENOLOGY_128: (8)                     b: float,
SNGT_QHENOMENOLOGY_129: (8)                     resolution: Sequence[int],
SNGT_QHENOMENOLOGY_130: (8)                     axis: int
SNGT_QHENOMENOLOGY_131: (4)                 ) -> Vect3Array:
SNGT_QHENOMENOLOGY_132: (8)                     if len(points) == 0:
SNGT_QHENOMENOLOGY_133: (12)                        return points
SNGT_QHENOMENOLOGY_134: (8)                     nu, nv = resolution[:2]
SNGT_QHENOMENOLOGY_135: (8)                     points = points.reshape(resolution).copy()
SNGT_QHENOMENOLOGY_136: (8)                     max_index = resolution[axis] - 1
SNGT_QHENOMENOLOGY_137: (8)                     lower_index, lower_residue = integer_interpolate(0,
max_index, a)
SNGT_QHENOMENOLOGY_138: (8)                     upper_index, upper_residue = integer_interpolate(0,
max_index, b)
SNGT_QHENOMENOLOGY_139: (8)                     if axis == 0:
SNGT_QHENOMENOLOGY_140: (12)                        points[:lower_index] = interpolate(
SNGT_QHENOMENOLOGY_141: (16)                            points[lower_index],
SNGT_QHENOMENOLOGY_142: (16)                            points[lower_index + 1],
SNGT_QHENOMENOLOGY_143: (16)                            lower_residue
SNGT_QHENOMENOLOGY_144: (12)                        )
SNGT_QHENOMENOLOGY_145: (12)                        points[upper_index + 1:] = interpolate(
SNGT_QHENOMENOLOGY_146: (16)                            points[upper_index],
SNGT_QHENOMENOLOGY_147: (16)                            points[upper_index + 1],
SNGT_QHENOMENOLOGY_148: (16)                            upper_residue
SNGT_QHENOMENOLOGY_149: (12)                        )
SNGT_QHENOMENOLOGY_150: (8)                     else:
SNGT_QHENOMENOLOGY_151: (12)                        shape = (nu, 1, resolution[2])
SNGT_QHENOMENOLOGY_152: (12)                        points[:, :lower_index] = interpolate(
SNGT_QHENOMENOLOGY_153: (16)                            points[:, lower_index],
SNGT_QHENOMENOLOGY_154: (16)                            points[:, lower_index + 1],
SNGT_QHENOMENOLOGY_155: (16)                            lower_residue
SNGT_QHENOMENOLOGY_156: (12)                        ).reshape(shape)
SNGT_QHENOMENOLOGY_157: (12)                        points[:, upper_index + 1:] = interpolate(
SNGT_QHENOMENOLOGY_158: (16)                            points[:, upper_index],
SNGT_QHENOMENOLOGY_159: (16)                            points[:, upper_index + 1],
SNGT_QHENOMENOLOGY_160: (16)                            upper_residue
SNGT_QHENOMENOLOGY_161: (12)                        ).reshape(shape)
SNGT_QHENOMENOLOGY_162: (8)                     return points.reshape((nu * nv, *resolution[2:]))
SNGT_QHENOMENOLOGY_163: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_164: (4)                 def sort_faces_back_to_front(self, vect: Vect3 = OUT) -
> Self:
SNGT_QHENOMENOLOGY_165: (8)                     tri_is = self.triangle_indices
SNGT_QHENOMENOLOGY_166: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_167: (8)                     dots = (points[tri_is[::3]] * vect).sum(1)
SNGT_QHENOMENOLOGY_168: (8)                     indices = np.argsort(dots)
SNGT_QHENOMENOLOGY_169: (8)                     for k in range(3):
SNGT_QHENOMENOLOGY_170: (12)                        tri_is[k::3] = tri_is[k::3][indices]
SNGT_QHENOMENOLOGY_171: (8)                     return self
SNGT_QHENOMENOLOGY_172: (4)                 def always_sort_to_camera(self, camera: Camera) ->
Self:
SNGT_QHENOMENOLOGY_173: (8)                     def updater(surface: Surface):
SNGT_QHENOMENOLOGY_174: (12)                        vect = camera.get_location() -
surface.get_center()
SNGT_QHENOMENOLOGY_175: (12)                        surface.sort_faces_back_to_front(vect)
SNGT_QHENOMENOLOGY_176: (8)                     self.add_updater(updater)
SNGT_QHENOMENOLOGY_177: (8)                     return self
SNGT_QHENOMENOLOGY_178: (4)                 def get_shader_vert_indices(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_179: (8)                     return self.get_triangle_indices()
SNGT_QHENOMENOLOGY_180: (0)             class ParametricSurface(Surface):
SNGT_QHENOMENOLOGY_181: (4)                 def __init__(
SNGT_QHENOMENOLOGY_182: (8)                     self,
SNGT_QHENOMENOLOGY_183: (8)                     uv_func: Callable[[float, float], Iterable[float]],
```

```
SNGT_QHENOMENOLOGY_184: (8)                     u_range: tuple[float, float] = (0, 1),
SNGT_QHENOMENOLOGY_185: (8)                     v_range: tuple[float, float] = (0, 1),
SNGT_QHENOMENOLOGY_186: (8)                     **kwargs
SNGT_QHENOMENOLOGY_187: (4)                 ):
SNGT_QHENOMENOLOGY_188: (8)                     self.passed_uv_func = uv_func
SNGT_QHENOMENOLOGY_189: (8)                     super().__init__(u_range=u_range, v_range=v_range,
**kwargs)
SNGT_QHENOMENOLOGY_190: (4)                 def uv_func(self, u, v):
SNGT_QHENOMENOLOGY_191: (8)                     return self.passed_uv_func(u, v)
SNGT_QHENOMENOLOGY_192: (0)             class SGroup(Surface):
SNGT_QHENOMENOLOGY_193: (4)                 def __init__(
SNGT_QHENOMENOLOGY_194: (8)                     self,
SNGT_QHENOMENOLOGY_195: (8)                     *parametric_surfaces: Surface,
SNGT_QHENOMENOLOGY_196: (8)                     **kwargs
SNGT_QHENOMENOLOGY_197: (4)                 ):
SNGT_QHENOMENOLOGY_198: (8)                     super().__init__(resolution=(0, 0), **kwargs)
SNGT_QHENOMENOLOGY_199: (8)                     self.add(*parametric_surfaces)
SNGT_QHENOMENOLOGY_200: (4)                 def init_points(self):
SNGT_QHENOMENOLOGY_201: (8)                     pass  # Needed?
SNGT_QHENOMENOLOGY_202: (0)             class TexturedSurface(Surface):
SNGT_QHENOMENOLOGY_203: (4)                 shader_folder: str = "textured_surface"
SNGT_QHENOMENOLOGY_204: (4)                 data_dtype: Sequence[Tuple[str, type, Tuple[int]]] = [
SNGT_QHENOMENOLOGY_205: (8)                     ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_206: (8)                     ('du_point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_207: (8)                     ('dv_point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_208: (8)                     ('im_coords', np.float32, (2,)),
SNGT_QHENOMENOLOGY_209: (8)                     ('opacity', np.float32, (1,)),
SNGT_QHENOMENOLOGY_210: (4)                 ]
SNGT_QHENOMENOLOGY_211: (4)                 def __init__(
SNGT_QHENOMENOLOGY_212: (8)                     self,
SNGT_QHENOMENOLOGY_213: (8)                     uv_surface: Surface,
SNGT_QHENOMENOLOGY_214: (8)                     image_file: str,
SNGT_QHENOMENOLOGY_215: (8)                     dark_image_file: str | None = None,
SNGT_QHENOMENOLOGY_216: (8)                     **kwargs
SNGT_QHENOMENOLOGY_217: (4)                 ):
SNGT_QHENOMENOLOGY_218: (8)                     if not isinstance(uv_surface, Surface):
SNGT_QHENOMENOLOGY_219: (12)                        raise Exception("uv_surface must be of type
Surface")
SNGT_QHENOMENOLOGY_220: (8)                     if dark_image_file is None:
SNGT_QHENOMENOLOGY_221: (12)                        dark_image_file = image_file
SNGT_QHENOMENOLOGY_222: (12)                        self.num_textures = 1
SNGT_QHENOMENOLOGY_223: (8)                     else:
SNGT_QHENOMENOLOGY_224: (12)                        self.num_textures = 2
SNGT_QHENOMENOLOGY_225: (8)                     texture_paths = {
SNGT_QHENOMENOLOGY_226: (12)                        "LightTexture":
get_full_raster_image_path(image_file),
SNGT_QHENOMENOLOGY_227: (12)                        "DarkTexture":
get_full_raster_image_path(dark_image_file),
SNGT_QHENOMENOLOGY_228: (8)                     }
SNGT_QHENOMENOLOGY_229: (8)                     self.uv_surface = uv_surface
SNGT_QHENOMENOLOGY_230: (8)                     self.uv_func = uv_surface.uv_func
SNGT_QHENOMENOLOGY_231: (8)                     self.u_range: Tuple[float, float] =
uv_surface.u_range
SNGT_QHENOMENOLOGY_232: (8)                     self.v_range: Tuple[float, float] =
uv_surface.v_range
SNGT_QHENOMENOLOGY_233: (8)                     self.resolution: Tuple[int, int] =
uv_surface.resolution
SNGT_QHENOMENOLOGY_234: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_235: (12)                        texture_paths=texture_paths,
SNGT_QHENOMENOLOGY_236: (12)                        shading=tuple(uv_surface.shading),
SNGT_QHENOMENOLOGY_237: (12)                        **kwargs
SNGT_QHENOMENOLOGY_238: (8)                     )
SNGT_QHENOMENOLOGY_239: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_240: (4)                 def init_points(self):
SNGT_QHENOMENOLOGY_241: (8)                     surf = self.uv_surface
SNGT_QHENOMENOLOGY_242: (8)                     nu, nv = surf.resolution
SNGT_QHENOMENOLOGY_243: (8)                     self.resize_points(surf.get_num_points())
SNGT_QHENOMENOLOGY_244: (8)                     self.resolution = surf.resolution
SNGT_QHENOMENOLOGY_245: (8)                     self.data['point'][:] = surf.data['point']
```

```
SNGT_QHENOMENOLOGY_246: (8)                     self.data['du_point'][:] = surf.data['du_point']
SNGT_QHENOMENOLOGY_247: (8)                     self.data['dv_point'][:] = surf.data['dv_point']
SNGT_QHENOMENOLOGY_248: (8)                     self.data['opacity'][:, 0] = surf.data["rgba"][:,
3]
SNGT_QHENOMENOLOGY_249: (8)                     self.data["im_coords"] = np.array([
SNGT_QHENOMENOLOGY_250: (12)                        [u, v]
SNGT_QHENOMENOLOGY_251: (12)                        for u in np.linspace(0, 1, nu)
SNGT_QHENOMENOLOGY_252: (12)                        for v in np.linspace(1, 0, nv)  # Reverse y-
direction
SNGT_QHENOMENOLOGY_253: (8)                     ])
SNGT_QHENOMENOLOGY_254: (4)                 def init_uniforms(self):
SNGT_QHENOMENOLOGY_255: (8)                     super().init_uniforms()
SNGT_QHENOMENOLOGY_256: (8)                     self.uniforms["num_textures"] = self.num_textures
SNGT_QHENOMENOLOGY_257: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_258: (4)                 def set_opacity(self, opacity: float | Iterable[float])
-> Self:
SNGT_QHENOMENOLOGY_259: (8)                     op_arr = np.array(listify(opacity))
SNGT_QHENOMENOLOGY_260: (8)                     self.data["opacity"][:, 0] =
resize_with_interpolation(op_arr, len(self.data))
SNGT_QHENOMENOLOGY_261: (8)                     return self
SNGT_QHENOMENOLOGY_262: (4)                 def set_color(
SNGT_QHENOMENOLOGY_263: (8)                     self,
SNGT_QHENOMENOLOGY_264: (8)                     color: ManimColor | Iterable[ManimColor] | None,
SNGT_QHENOMENOLOGY_265: (8)                     opacity: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_266: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_267: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_268: (8)                     if opacity is not None:
SNGT_QHENOMENOLOGY_269: (12)                        self.set_opacity(opacity)
SNGT_QHENOMENOLOGY_270: (8)                     return self
SNGT_QHENOMENOLOGY_271: (4)                 def pointwise_become_partial(
SNGT_QHENOMENOLOGY_272: (8)                     self,
SNGT_QHENOMENOLOGY_273: (8)                     tsmobject: "TexturedSurface",
SNGT_QHENOMENOLOGY_274: (8)                     a: float,
SNGT_QHENOMENOLOGY_275: (8)                     b: float,
SNGT_QHENOMENOLOGY_276: (8)                     axis: int = 1
SNGT_QHENOMENOLOGY_277: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_278: (8)                     super().pointwise_become_partial(tsmobject, a, b,
axis)
SNGT_QHENOMENOLOGY_279: (8)                     im_coords = self.data["im_coords"]
SNGT_QHENOMENOLOGY_280: (8)                     im_coords[:] = tsmobject.data["im_coords"]
SNGT_QHENOMENOLOGY_281: (8)                     if a <= 0 and b >= 1:
SNGT_QHENOMENOLOGY_282: (12)                        return self
SNGT_QHENOMENOLOGY_283: (8)                     nu, nv = tsmobject.resolution
SNGT_QHENOMENOLOGY_284: (8)                     im_coords[:] = self.get_partial_points_array(
SNGT_QHENOMENOLOGY_285: (12)                        im_coords, a, b, (nu, nv, 2), axis
SNGT_QHENOMENOLOGY_286: (8)                     )
SNGT_QHENOMENOLOGY_287: (8)                     return self
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 48 - drawings.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               import itertools as it
SNGT_QHENOMENOLOGY_4: (0)               import random
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.animation.composition import AnimationGroup
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.animation.rotation import Rotating
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import BLACK
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.constants import BLUE_A
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.constants import BLUE_B
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.constants import BLUE_C
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.constants import BLUE_D
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.constants import DOWN
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.constants import DOWN
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.constants import FRAME_WIDTH
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.constants import GREEN
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.constants import GREEN_SCREEN
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.constants import GREEN_E
```

```
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.constants import GREY
SNGT_QHENOMENOLOGY_19: (0)              from manimlib.constants import GREY_A
SNGT_QHENOMENOLOGY_20: (0)              from manimlib.constants import GREY_B
SNGT_QHENOMENOLOGY_21: (0)              from manimlib.constants import GREY_E
SNGT_QHENOMENOLOGY_22: (0)              from manimlib.constants import LEFT
SNGT_QHENOMENOLOGY_23: (0)              from manimlib.constants import LEFT
SNGT_QHENOMENOLOGY_24: (0)              from manimlib.constants import MED_LARGE_BUFF
SNGT_QHENOMENOLOGY_25: (0)              from manimlib.constants import MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_26: (0)              from manimlib.constants import ORIGIN
SNGT_QHENOMENOLOGY_27: (0)              from manimlib.constants import OUT
SNGT_QHENOMENOLOGY_28: (0)              from manimlib.constants import PI
SNGT_QHENOMENOLOGY_29: (0)              from manimlib.constants import RED
SNGT_QHENOMENOLOGY_30: (0)              from manimlib.constants import RED_E
SNGT_QHENOMENOLOGY_31: (0)              from manimlib.constants import RIGHT
SNGT_QHENOMENOLOGY_32: (0)              from manimlib.constants import SMALL_BUFF
SNGT_QHENOMENOLOGY_33: (0)              from manimlib.constants import SMALL_BUFF
SNGT_QHENOMENOLOGY_34: (0)              from manimlib.constants import UP
SNGT_QHENOMENOLOGY_35: (0)              from manimlib.constants import UL
SNGT_QHENOMENOLOGY_36: (0)              from manimlib.constants import UR
SNGT_QHENOMENOLOGY_37: (0)              from manimlib.constants import DL
SNGT_QHENOMENOLOGY_38: (0)              from manimlib.constants import DR
SNGT_QHENOMENOLOGY_39: (0)              from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_40: (0)              from manimlib.constants import YELLOW
SNGT_QHENOMENOLOGY_41: (0)              from manimlib.constants import TAU
SNGT_QHENOMENOLOGY_42: (0)              from manimlib.mobject.boolean_ops import Difference
SNGT_QHENOMENOLOGY_43: (0)              from manimlib.mobject.boolean_ops import Union
SNGT_QHENOMENOLOGY_44: (0)              from manimlib.mobject.geometry import Arc
SNGT_QHENOMENOLOGY_45: (0)              from manimlib.mobject.geometry import Circle
SNGT_QHENOMENOLOGY_46: (0)              from manimlib.mobject.geometry import Dot
SNGT_QHENOMENOLOGY_47: (0)              from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_48: (0)              from manimlib.mobject.geometry import Polygon
SNGT_QHENOMENOLOGY_49: (0)              from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_50: (0)              from manimlib.mobject.geometry import Square
SNGT_QHENOMENOLOGY_51: (0)              from manimlib.mobject.geometry import AnnularSector
SNGT_QHENOMENOLOGY_52: (0)              from manimlib.mobject.numbers import Integer
SNGT_QHENOMENOLOGY_53: (0)              from manimlib.mobject.shape_matchers import
SurroundingRectangle
SNGT_QHENOMENOLOGY_54: (0)              from manimlib.mobject.svg.svg_mobject import SVGMobject
SNGT_QHENOMENOLOGY_55: (0)              from manimlib.mobject.svg.special_tex import
TexTextFromPresetString
SNGT_QHENOMENOLOGY_56: (0)              from manimlib.mobject.three_dimensions import Prismify
SNGT_QHENOMENOLOGY_57: (0)              from manimlib.mobject.three_dimensions import VCube
SNGT_QHENOMENOLOGY_58: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_59: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_60: (0)              from manimlib.mobject.svg.text_mobject import Text
SNGT_QHENOMENOLOGY_61: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_62: (0)              from manimlib.utils.iterables import adjacent_pairs
SNGT_QHENOMENOLOGY_63: (0)              from manimlib.utils.rate_functions import linear
SNGT_QHENOMENOLOGY_64: (0)              from manimlib.utils.space_ops import angle_of_vector
SNGT_QHENOMENOLOGY_65: (0)              from manimlib.utils.space_ops import compass_directions
SNGT_QHENOMENOLOGY_66: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_67: (0)              from manimlib.utils.space_ops import midpoint
SNGT_QHENOMENOLOGY_68: (0)              from manimlib.utils.space_ops import rotate_vector
SNGT_QHENOMENOLOGY_69: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_70: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_71: (4)                  from typing import Tuple, Sequence, Callable
SNGT_QHENOMENOLOGY_72: (4)                  from manimlib.typing import ManimColor, Vect3
SNGT_QHENOMENOLOGY_73: (0)              class Checkmark(TexTextFromPresetString):
SNGT_QHENOMENOLOGY_74: (4)                  tex: str = R"\ding{51}"
SNGT_QHENOMENOLOGY_75: (4)                  default_color: ManimColor = GREEN
SNGT_QHENOMENOLOGY_76: (0)              class Exmark(TexTextFromPresetString):
SNGT_QHENOMENOLOGY_77: (4)                  tex: str = R"\ding{55}"
SNGT_QHENOMENOLOGY_78: (4)                  default_color: ManimColor = RED
SNGT_QHENOMENOLOGY_79: (0)              class Lightbulb(SVGMobject):
SNGT_QHENOMENOLOGY_80: (4)                  file_name = "lightbulb"
SNGT_QHENOMENOLOGY_81: (4)                  def __init__(
SNGT_QHENOMENOLOGY_82: (8)                      self,
```

```
SNGT_QHENOMENOLOGY_83: (8)                      height: float = 1.0,
SNGT_QHENOMENOLOGY_84: (8)                      color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_85: (8)                      stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_86: (8)                      fill_opacity: float = 0.0,
SNGT_QHENOMENOLOGY_87: (8)                      **kwargs
SNGT_QHENOMENOLOGY_88: (4)                  ):
SNGT_QHENOMENOLOGY_89: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_90: (12)                         height=height,
SNGT_QHENOMENOLOGY_91: (12)                         color=color,
SNGT_QHENOMENOLOGY_92: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_93: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_94: (12)                         **kwargs
SNGT_QHENOMENOLOGY_95: (8)                      )
SNGT_QHENOMENOLOGY_96: (8)                      self.insert_n_curves(25)
SNGT_QHENOMENOLOGY_97: (0)              class Speedometer(VMobject):
SNGT_QHENOMENOLOGY_98: (4)                  def __init__(
SNGT_QHENOMENOLOGY_99: (8)                      self,
SNGT_QHENOMENOLOGY_100: (8)                     arc_angle: float = 4 * PI / 3,
SNGT_QHENOMENOLOGY_101: (8)                     num_ticks: int = 8,
SNGT_QHENOMENOLOGY_102: (8)                     tick_length: float = 0.2,
SNGT_QHENOMENOLOGY_103: (8)                     needle_width: float = 0.1,
SNGT_QHENOMENOLOGY_104: (8)                     needle_height: float = 0.8,
SNGT_QHENOMENOLOGY_105: (8)                     needle_color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_106: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_107: (4)                 ):
SNGT_QHENOMENOLOGY_108: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_109: (8)                     self.arc_angle = arc_angle
SNGT_QHENOMENOLOGY_110: (8)                     self.num_ticks = num_ticks
SNGT_QHENOMENOLOGY_111: (8)                     self.tick_length = tick_length
SNGT_QHENOMENOLOGY_112: (8)                     self.needle_width = needle_width
SNGT_QHENOMENOLOGY_113: (8)                     self.needle_height = needle_height
SNGT_QHENOMENOLOGY_114: (8)                     self.needle_color = needle_color
SNGT_QHENOMENOLOGY_115: (8)                     start_angle = PI / 2 + arc_angle / 2
SNGT_QHENOMENOLOGY_116: (8)                     end_angle = PI / 2 - arc_angle / 2
SNGT_QHENOMENOLOGY_117: (8)                     self.arc = Arc(
SNGT_QHENOMENOLOGY_118: (12)                        start_angle=start_angle,
SNGT_QHENOMENOLOGY_119: (12)                        angle=-self.arc_angle
SNGT_QHENOMENOLOGY_120: (8)                     )
SNGT_QHENOMENOLOGY_121: (8)                     self.add(self.arc)
SNGT_QHENOMENOLOGY_122: (8)                     tick_angle_range = np.linspace(start_angle,
end_angle, num_ticks)
SNGT_QHENOMENOLOGY_123: (8)                     for index, angle in enumerate(tick_angle_range):
SNGT_QHENOMENOLOGY_124: (12)                        vect = rotate_vector(RIGHT, angle)
SNGT_QHENOMENOLOGY_125: (12)                        tick = Line((1 - tick_length) * vect, vect)
SNGT_QHENOMENOLOGY_126: (12)                        label = Integer(10 * index)
SNGT_QHENOMENOLOGY_127: (12)                        label.set_height(tick_length)
SNGT_QHENOMENOLOGY_128: (12)                        label.shift((1 + tick_length) * vect)
SNGT_QHENOMENOLOGY_129: (12)                        self.add(tick, label)
SNGT_QHENOMENOLOGY_130: (8)                     needle = Polygon(
SNGT_QHENOMENOLOGY_131: (12)                        LEFT, UP, RIGHT,
SNGT_QHENOMENOLOGY_132: (12)                        stroke_width=0,
SNGT_QHENOMENOLOGY_133: (12)                        fill_opacity=1,
SNGT_QHENOMENOLOGY_134: (12)                        fill_color=self.needle_color
SNGT_QHENOMENOLOGY_135: (8)                     )
SNGT_QHENOMENOLOGY_136: (8)                     needle.stretch_to_fit_width(needle_width)
SNGT_QHENOMENOLOGY_137: (8)                     needle.stretch_to_fit_height(needle_height)
SNGT_QHENOMENOLOGY_138: (8)                     needle.rotate(start_angle - np.pi / 2,
about_point=ORIGIN)
SNGT_QHENOMENOLOGY_139: (8)                     self.add(needle)
SNGT_QHENOMENOLOGY_140: (8)                     self.needle = needle
SNGT_QHENOMENOLOGY_141: (8)                     self.center_offset = self.get_center()
SNGT_QHENOMENOLOGY_142: (4)                 def get_center(self):
SNGT_QHENOMENOLOGY_143: (8)                     result = VMobject.get_center(self)
SNGT_QHENOMENOLOGY_144: (8)                     if hasattr(self, "center_offset"):
SNGT_QHENOMENOLOGY_145: (12)                        result -= self.center_offset
SNGT_QHENOMENOLOGY_146: (8)                     return result
SNGT_QHENOMENOLOGY_147: (4)                 def get_needle_tip(self):
SNGT_QHENOMENOLOGY_148: (8)                     return self.needle.get_anchors()[1]
SNGT_QHENOMENOLOGY_149: (4)                 def get_needle_angle(self):
```

```
SNGT_QHENOMENOLOGY_150: (8)                     return angle_of_vector(
SNGT_QHENOMENOLOGY_151: (12)                        self.get_needle_tip() - self.get_center()
SNGT_QHENOMENOLOGY_152: (8)                     )
SNGT_QHENOMENOLOGY_153: (4)                 def rotate_needle(self, angle):
SNGT_QHENOMENOLOGY_154: (8)                     self.needle.rotate(angle,
about_point=self.arc.get_arc_center())
SNGT_QHENOMENOLOGY_155: (8)                     return self
SNGT_QHENOMENOLOGY_156: (4)                 def move_needle_to_velocity(self, velocity):
SNGT_QHENOMENOLOGY_157: (8)                     max_velocity = 10 * (self.num_ticks - 1)
SNGT_QHENOMENOLOGY_158: (8)                     proportion = float(velocity) / max_velocity
SNGT_QHENOMENOLOGY_159: (8)                     start_angle = np.pi / 2 + self.arc_angle / 2
SNGT_QHENOMENOLOGY_160: (8)                     target_angle = start_angle - self.arc_angle *
proportion
SNGT_QHENOMENOLOGY_161: (8)                     self.rotate_needle(target_angle -
self.get_needle_angle())
SNGT_QHENOMENOLOGY_162: (8)                     return self
SNGT_QHENOMENOLOGY_163: (0)             class Laptop(VGroup):
SNGT_QHENOMENOLOGY_164: (4)                 def __init__(
SNGT_QHENOMENOLOGY_165: (8)                     self,
SNGT_QHENOMENOLOGY_166: (8)                     width: float = 3,
SNGT_QHENOMENOLOGY_167: (8)                     body_dimensions: Tuple[float, float, float] = (4.0,
3.0, 0.05),
SNGT_QHENOMENOLOGY_168: (8)                     screen_thickness: float = 0.01,
SNGT_QHENOMENOLOGY_169: (8)                     keyboard_width_to_body_width: float = 0.9,
SNGT_QHENOMENOLOGY_170: (8)                     keyboard_height_to_body_height: float = 0.5,
SNGT_QHENOMENOLOGY_171: (8)                     screen_width_to_screen_plate_width: float = 0.9,
SNGT_QHENOMENOLOGY_172: (8)                     key_color_kwargs: dict = dict(
SNGT_QHENOMENOLOGY_173: (12)                        stroke_width=0,
SNGT_QHENOMENOLOGY_174: (12)                        fill_color=BLACK,
SNGT_QHENOMENOLOGY_175: (12)                        fill_opacity=1,
SNGT_QHENOMENOLOGY_176: (8)                     ),
SNGT_QHENOMENOLOGY_177: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_178: (8)                     stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_179: (8)                     body_color: ManimColor = GREY_B,
SNGT_QHENOMENOLOGY_180: (8)                     shaded_body_color: ManimColor = GREY,
SNGT_QHENOMENOLOGY_181: (8)                     open_angle: float = np.pi / 4,
SNGT_QHENOMENOLOGY_182: (8)                     **kwargs
SNGT_QHENOMENOLOGY_183: (4)                 ):
SNGT_QHENOMENOLOGY_184: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_185: (8)                     body = VCube(side_length=1)
SNGT_QHENOMENOLOGY_186: (8)                     for dim, scale_factor in
enumerate(body_dimensions):
SNGT_QHENOMENOLOGY_187: (12)                        body.stretch(scale_factor, dim=dim)
SNGT_QHENOMENOLOGY_188: (8)                     body.set_width(width)
SNGT_QHENOMENOLOGY_189: (8)                     body.set_fill(shaded_body_color, opacity=1)
SNGT_QHENOMENOLOGY_190: (8)                     body.sort(lambda p: p[2])
SNGT_QHENOMENOLOGY_191: (8)                     body[-1].set_fill(body_color)
SNGT_QHENOMENOLOGY_192: (8)                     screen_plate = body.copy()
SNGT_QHENOMENOLOGY_193: (8)                     keyboard = VGroup(*[
SNGT_QHENOMENOLOGY_194: (12)                        VGroup(*[
SNGT_QHENOMENOLOGY_195: (16)                            Square(**key_color_kwargs)
SNGT_QHENOMENOLOGY_196: (16)                            for x in range(12 - y % 2)
SNGT_QHENOMENOLOGY_197: (12)                        ]).arrange(RIGHT, buff=SMALL_BUFF)
SNGT_QHENOMENOLOGY_198: (12)                        for y in range(4)
SNGT_QHENOMENOLOGY_199: (8)                     ]).arrange(DOWN, buff=MED_SMALL_BUFF)
SNGT_QHENOMENOLOGY_200: (8)                     keyboard.stretch_to_fit_width(
SNGT_QHENOMENOLOGY_201: (12)                        keyboard_width_to_body_width *
body.get_width(),
SNGT_QHENOMENOLOGY_202: (8)                     )
SNGT_QHENOMENOLOGY_203: (8)                     keyboard.stretch_to_fit_height(
SNGT_QHENOMENOLOGY_204: (12)                        keyboard_height_to_body_height *
body.get_height(),
SNGT_QHENOMENOLOGY_205: (8)                     )
SNGT_QHENOMENOLOGY_206: (8)                     keyboard.next_to(body, OUT, buff=0.1 * SMALL_BUFF)
SNGT_QHENOMENOLOGY_207: (8)                     keyboard.shift(MED_SMALL_BUFF * UP)
SNGT_QHENOMENOLOGY_208: (8)                     body.add(keyboard)
SNGT_QHENOMENOLOGY_209: (8)                     screen_plate.stretch(screen_thickness /
SNGT_QHENOMENOLOGY_210: (29)                                         body_dimensions[2], dim=2)
SNGT_QHENOMENOLOGY_211: (8)                     screen = Rectangle(
```

```
SNGT_QHENOMENOLOGY_212: (12)                         stroke_width=0,
SNGT_QHENOMENOLOGY_213: (12)                         fill_color=BLACK,
SNGT_QHENOMENOLOGY_214: (12)                         fill_opacity=1,
SNGT_QHENOMENOLOGY_215: (8)                     )
SNGT_QHENOMENOLOGY_216: (8)                     screen.replace(screen_plate, stretch=True)
SNGT_QHENOMENOLOGY_217: (8)                     screen.scale(screen_width_to_screen_plate_width)
SNGT_QHENOMENOLOGY_218: (8)                     screen.next_to(screen_plate, OUT, buff=0.1 *
SMALL_BUFF)
SNGT_QHENOMENOLOGY_219: (8)                     screen_plate.add(screen)
SNGT_QHENOMENOLOGY_220: (8)                     screen_plate.next_to(body, UP, buff=0)
SNGT_QHENOMENOLOGY_221: (8)                     screen_plate.rotate(
SNGT_QHENOMENOLOGY_222: (12)                         open_angle, RIGHT,
SNGT_QHENOMENOLOGY_223: (12)                         about_point=screen_plate.get_bottom()
SNGT_QHENOMENOLOGY_224: (8)                     )
SNGT_QHENOMENOLOGY_225: (8)                     self.screen_plate = screen_plate
SNGT_QHENOMENOLOGY_226: (8)                     self.screen = screen
SNGT_QHENOMENOLOGY_227: (8)                     axis = Line(
SNGT_QHENOMENOLOGY_228: (12)                         body.get_corner(UP + LEFT + OUT),
SNGT_QHENOMENOLOGY_229: (12)                         body.get_corner(UP + RIGHT + OUT),
SNGT_QHENOMENOLOGY_230: (12)                         color=BLACK,
SNGT_QHENOMENOLOGY_231: (12)                         stroke_width=2
SNGT_QHENOMENOLOGY_232: (8)                     )
SNGT_QHENOMENOLOGY_233: (8)                     self.axis = axis
SNGT_QHENOMENOLOGY_234: (8)                     self.add(body, screen_plate, axis)
SNGT_QHENOMENOLOGY_235: (0)             class VideoIcon(SVGMobject):
SNGT_QHENOMENOLOGY_236: (4)                 file_name: str = "video_icon"
SNGT_QHENOMENOLOGY_237: (4)                 def __init__(
SNGT_QHENOMENOLOGY_238: (8)                     self,
SNGT_QHENOMENOLOGY_239: (8)                     width: float = 1.2,
SNGT_QHENOMENOLOGY_240: (8)                     color=BLUE_A,
SNGT_QHENOMENOLOGY_241: (8)                     **kwargs
SNGT_QHENOMENOLOGY_242: (4)                 ):
SNGT_QHENOMENOLOGY_243: (8)                     super().__init__(color=color, **kwargs)
SNGT_QHENOMENOLOGY_244: (8)                     self.set_width(width)
SNGT_QHENOMENOLOGY_245: (0)             class VideoSeries(VGroup):
SNGT_QHENOMENOLOGY_246: (4)                 def __init__(
SNGT_QHENOMENOLOGY_247: (8)                     self,
SNGT_QHENOMENOLOGY_248: (8)                     num_videos: int = 11,
SNGT_QHENOMENOLOGY_249: (8)                     gradient_colors: Sequence[ManimColor] = [BLUE_B,
BLUE_D],
SNGT_QHENOMENOLOGY_250: (8)                     width: float = FRAME_WIDTH - MED_LARGE_BUFF,
SNGT_QHENOMENOLOGY_251: (8)                     **kwargs
SNGT_QHENOMENOLOGY_252: (4)                 ):
SNGT_QHENOMENOLOGY_253: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_254: (12)                         *(VideoIcon() for x in range(num_videos)),
SNGT_QHENOMENOLOGY_255: (12)                         **kwargs
SNGT_QHENOMENOLOGY_256: (8)                     )
SNGT_QHENOMENOLOGY_257: (8)                     self.arrange(RIGHT)
SNGT_QHENOMENOLOGY_258: (8)                     self.set_width(width)
SNGT_QHENOMENOLOGY_259: (8)                     self.set_color_by_gradient(*gradient_colors)
SNGT_QHENOMENOLOGY_260: (0)             class Clock(VGroup):
SNGT_QHENOMENOLOGY_261: (4)                 def __init__(
SNGT_QHENOMENOLOGY_262: (8)                     self,
SNGT_QHENOMENOLOGY_263: (8)                     stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_264: (8)                     stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_265: (8)                     hour_hand_height: float = 0.3,
SNGT_QHENOMENOLOGY_266: (8)                     minute_hand_height: float = 0.6,
SNGT_QHENOMENOLOGY_267: (8)                     tick_length: float = 0.1,
SNGT_QHENOMENOLOGY_268: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_269: (4)                 ):
SNGT_QHENOMENOLOGY_270: (8)                     style = dict(stroke_color=stroke_color,
stroke_width=stroke_width)
SNGT_QHENOMENOLOGY_271: (8)                     circle = Circle(**style)
SNGT_QHENOMENOLOGY_272: (8)                     ticks = []
SNGT_QHENOMENOLOGY_273: (8)                     for x, point in enumerate(compass_directions(12,
UP)):
SNGT_QHENOMENOLOGY_274: (12)                         length = tick_length
SNGT_QHENOMENOLOGY_275: (12)                         if x % 3 == 0:
SNGT_QHENOMENOLOGY_276: (16)                             length *= 2
```

```
SNGT_QHENOMENOLOGY_277: (12)                         ticks.append(Line(point, (1 - length) * point,
**style))
SNGT_QHENOMENOLOGY_278: (8)                     self.hour_hand = Line(ORIGIN, hour_hand_height *
UP, **style)
SNGT_QHENOMENOLOGY_279: (8)                     self.minute_hand = Line(ORIGIN, minute_hand_height
* UP, **style)
SNGT_QHENOMENOLOGY_280: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_281: (12)                         circle, self.hour_hand, self.minute_hand,
SNGT_QHENOMENOLOGY_282: (12)                         *ticks
SNGT_QHENOMENOLOGY_283: (8)                     )
SNGT_QHENOMENOLOGY_284: (0)             class ClockPassesTime(AnimationGroup):
SNGT_QHENOMENOLOGY_285: (4)                 def __init__(
SNGT_QHENOMENOLOGY_286: (8)                     self,
SNGT_QHENOMENOLOGY_287: (8)                     clock: Clock,
SNGT_QHENOMENOLOGY_288: (8)                     run_time: float = 5.0,
SNGT_QHENOMENOLOGY_289: (8)                     hours_passed: float = 12.0,
SNGT_QHENOMENOLOGY_290: (8)                     rate_func: Callable[[float], float] = linear,
SNGT_QHENOMENOLOGY_291: (8)                     **kwargs
SNGT_QHENOMENOLOGY_292: (4)                 ):
SNGT_QHENOMENOLOGY_293: (8)                     rot_kwargs = dict(
SNGT_QHENOMENOLOGY_294: (12)                         axis=OUT,
SNGT_QHENOMENOLOGY_295: (12)                         about_point=clock.get_center()
SNGT_QHENOMENOLOGY_296: (8)                     )
SNGT_QHENOMENOLOGY_297: (8)                     hour_radians = -hours_passed * 2 * PI / 12
SNGT_QHENOMENOLOGY_298: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_299: (12)                         Rotating(
SNGT_QHENOMENOLOGY_300: (16)                             clock.hour_hand,
SNGT_QHENOMENOLOGY_301: (16)                             angle=hour_radians,
SNGT_QHENOMENOLOGY_302: (16)                             **rot_kwargs
SNGT_QHENOMENOLOGY_303: (12)                         ),
SNGT_QHENOMENOLOGY_304: (12)                         Rotating(
SNGT_QHENOMENOLOGY_305: (16)                             clock.minute_hand,
SNGT_QHENOMENOLOGY_306: (16)                             angle=12 * hour_radians,
SNGT_QHENOMENOLOGY_307: (16)                             **rot_kwargs
SNGT_QHENOMENOLOGY_308: (12)                         ),
SNGT_QHENOMENOLOGY_309: (12)                         **kwargs
SNGT_QHENOMENOLOGY_310: (8)                     )
SNGT_QHENOMENOLOGY_311: (0)             class Bubble(VGroup):
SNGT_QHENOMENOLOGY_312: (4)                 file_name: str = "Bubbles_speech.svg"
SNGT_QHENOMENOLOGY_313: (4)                 bubble_center_adjustment_factor = 0.125
SNGT_QHENOMENOLOGY_314: (4)                 def __init__(
SNGT_QHENOMENOLOGY_315: (8)                     self,
SNGT_QHENOMENOLOGY_316: (8)                     content: str | VMobject | None = None,
SNGT_QHENOMENOLOGY_317: (8)                     buff: float = 1.0,
SNGT_QHENOMENOLOGY_318: (8)                     filler_shape: Tuple[float, float] = (3.0, 2.0),
SNGT_QHENOMENOLOGY_319: (8)                     pin_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_320: (8)                     direction: Vect3 = LEFT,
SNGT_QHENOMENOLOGY_321: (8)                     add_content: bool = True,
SNGT_QHENOMENOLOGY_322: (8)                     fill_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_323: (8)                     fill_opacity: float = 0.8,
SNGT_QHENOMENOLOGY_324: (8)                     stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_325: (8)                     stroke_width: float = 3.0,
SNGT_QHENOMENOLOGY_326: (8)                     **kwargs
SNGT_QHENOMENOLOGY_327: (4)                 ):
SNGT_QHENOMENOLOGY_328: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_329: (8)                     self.direction = direction
SNGT_QHENOMENOLOGY_330: (8)                     if content is None:
SNGT_QHENOMENOLOGY_331: (12)                         content = Rectangle(*filler_shape)
SNGT_QHENOMENOLOGY_332: (12)                         content.set_fill(opacity=0)
SNGT_QHENOMENOLOGY_333: (12)                         content.set_stroke(width=0)
SNGT_QHENOMENOLOGY_334: (8)                     elif isinstance(content, str):
SNGT_QHENOMENOLOGY_335: (12)                         content = Text(content)
SNGT_QHENOMENOLOGY_336: (8)                     self.content = content
SNGT_QHENOMENOLOGY_337: (8)                     self.body = self.get_body(content, direction, buff)
SNGT_QHENOMENOLOGY_338: (8)                     self.body.set_fill(fill_color, fill_opacity)
SNGT_QHENOMENOLOGY_339: (8)                     self.body.set_stroke(stroke_color, stroke_width)
SNGT_QHENOMENOLOGY_340: (8)                     self.add(self.body)
SNGT_QHENOMENOLOGY_341: (8)                     if add_content:
SNGT_QHENOMENOLOGY_342: (12)                         self.add(self.content)
```

```
SNGT_QHENOMENOLOGY_343: (8)                     if pin_point is not None:
SNGT_QHENOMENOLOGY_344: (12)                        self.pin_to(pin_point)
SNGT_QHENOMENOLOGY_345: (4)                 def get_body(self, content: VMobject, direction: Vect3,
buff: float) -> VMobject:
SNGT_QHENOMENOLOGY_346: (8)                     body = SVGMobject(self.file_name)
SNGT_QHENOMENOLOGY_347: (8)                     if direction[0] > 0:
SNGT_QHENOMENOLOGY_348: (12)                        body.flip()
SNGT_QHENOMENOLOGY_349: (8)                     width = content.get_width()
SNGT_QHENOMENOLOGY_350: (8)                     height = content.get_height()
SNGT_QHENOMENOLOGY_351: (8)                     target_width = width + min(buff, height)
SNGT_QHENOMENOLOGY_352: (8)                     target_height = 1.35 * (height + buff)  # Magic
number?
SNGT_QHENOMENOLOGY_353: (8)                     body.set_shape(target_width, target_height)
SNGT_QHENOMENOLOGY_354: (8)                     body.move_to(content)
SNGT_QHENOMENOLOGY_355: (8)                     body.shift(self.bubble_center_adjustment_factor *
body.get_height() * DOWN)
SNGT_QHENOMENOLOGY_356: (8)                     return body
SNGT_QHENOMENOLOGY_357: (4)                 def get_tip(self):
SNGT_QHENOMENOLOGY_358: (8)                     return self.get_corner(DOWN + self.direction)
SNGT_QHENOMENOLOGY_359: (4)                 def get_bubble_center(self):
SNGT_QHENOMENOLOGY_360: (8)                     factor = self.bubble_center_adjustment_factor
SNGT_QHENOMENOLOGY_361: (8)                     return self.get_center() + factor *
self.get_height() * UP
SNGT_QHENOMENOLOGY_362: (4)                 def move_tip_to(self, point):
SNGT_QHENOMENOLOGY_363: (8)                     self.shift(point - self.get_tip())
SNGT_QHENOMENOLOGY_364: (8)                     return self
SNGT_QHENOMENOLOGY_365: (4)                 def flip(self, axis=UP, only_body=True, **kwargs):
SNGT_QHENOMENOLOGY_366: (8)                     super().flip(axis=axis, **kwargs)
SNGT_QHENOMENOLOGY_367: (8)                     if only_body:
SNGT_QHENOMENOLOGY_368: (12)                        self.content.flip(axis=axis)
SNGT_QHENOMENOLOGY_369: (8)                     if abs(axis[1]) > 0:
SNGT_QHENOMENOLOGY_370: (12)                        self.direction = -np.array(self.direction)
SNGT_QHENOMENOLOGY_371: (8)                     return self
SNGT_QHENOMENOLOGY_372: (4)                 def pin_to(self, mobject, auto_flip=False):
SNGT_QHENOMENOLOGY_373: (8)                     mob_center = mobject.get_center()
SNGT_QHENOMENOLOGY_374: (8)                     want_to_flip = np.sign(mob_center[0]) !=
np.sign(self.direction[0])
SNGT_QHENOMENOLOGY_375: (8)                     if want_to_flip and auto_flip:
SNGT_QHENOMENOLOGY_376: (12)                        self.flip()
SNGT_QHENOMENOLOGY_377: (8)                     boundary_point = mobject.get_bounding_box_point(UP
- self.direction)
SNGT_QHENOMENOLOGY_378: (8)                     vector_from_center = 1.0 * (boundary_point -
mob_center)
SNGT_QHENOMENOLOGY_379: (8)                     self.move_tip_to(mob_center + vector_from_center)
SNGT_QHENOMENOLOGY_380: (8)                     return self
SNGT_QHENOMENOLOGY_381: (4)                 def position_mobject_inside(self, mobject,
buff=MED_LARGE_BUFF):
SNGT_QHENOMENOLOGY_382: (8)                     mobject.set_max_width(self.body.get_width() - 2 *
buff)
SNGT_QHENOMENOLOGY_383: (8)                     mobject.set_max_height(self.body.get_height() / 1.5
- 2 * buff)
SNGT_QHENOMENOLOGY_384: (8)                     mobject.shift(self.get_bubble_center() -
mobject.get_center())
SNGT_QHENOMENOLOGY_385: (8)                     return mobject
SNGT_QHENOMENOLOGY_386: (4)                 def add_content(self, mobject):
SNGT_QHENOMENOLOGY_387: (8)                     self.position_mobject_inside(mobject)
SNGT_QHENOMENOLOGY_388: (8)                     self.content = mobject
SNGT_QHENOMENOLOGY_389: (8)                     return self.content
SNGT_QHENOMENOLOGY_390: (4)                 def write(self, text):
SNGT_QHENOMENOLOGY_391: (8)                     self.add_content(Text(text))
SNGT_QHENOMENOLOGY_392: (8)                     return self
SNGT_QHENOMENOLOGY_393: (4)                 def resize_to_content(self, buff=1.0):  # TODO
SNGT_QHENOMENOLOGY_394: (8)                     self.body.match_points(self.get_body(
SNGT_QHENOMENOLOGY_395: (12)                        self.content, self.direction, buff
SNGT_QHENOMENOLOGY_396: (8)                     ))
SNGT_QHENOMENOLOGY_397: (4)                 def clear(self):
SNGT_QHENOMENOLOGY_398: (8)                     self.remove(self.content)
SNGT_QHENOMENOLOGY_399: (8)                     return self
SNGT_QHENOMENOLOGY_400: (0)             class SpeechBubble(Bubble):
```

```
SNGT_QHENOMENOLOGY_401: (4)                 def __init__(
SNGT_QHENOMENOLOGY_402: (8)                     self,
SNGT_QHENOMENOLOGY_403: (8)                     content: str | VMobject | None = None,
SNGT_QHENOMENOLOGY_404: (8)                     buff: float = MED_SMALL_BUFF,
SNGT_QHENOMENOLOGY_405: (8)                     filler_shape: Tuple[float, float] = (2.0, 1.0),
SNGT_QHENOMENOLOGY_406: (8)                     stem_height_to_bubble_height: float = 0.5,
SNGT_QHENOMENOLOGY_407: (8)                     stem_top_x_props: Tuple[float, float] = (0.2, 0.3),
SNGT_QHENOMENOLOGY_408: (8)                     **kwargs
SNGT_QHENOMENOLOGY_409: (4)                 ):
SNGT_QHENOMENOLOGY_410: (8)                     self.stem_height_to_bubble_height =
stem_height_to_bubble_height
SNGT_QHENOMENOLOGY_411: (8)                     self.stem_top_x_props = stem_top_x_props
SNGT_QHENOMENOLOGY_412: (8)                     super().__init__(content, buff, filler_shape,
**kwargs)
SNGT_QHENOMENOLOGY_413: (4)                 def get_body(self, content: VMobject, direction: Vect3,
buff: float) -> VMobject:
SNGT_QHENOMENOLOGY_414: (8)                     rect = SurroundingRectangle(content, buff=buff)
SNGT_QHENOMENOLOGY_415: (8)                     rect.round_corners()
SNGT_QHENOMENOLOGY_416: (8)                     lp = rect.get_corner(DL)
SNGT_QHENOMENOLOGY_417: (8)                     rp = rect.get_corner(DR)
SNGT_QHENOMENOLOGY_418: (8)                     stem_height = self.stem_height_to_bubble_height *
rect.get_height()
SNGT_QHENOMENOLOGY_419: (8)                     low_prop, high_prop = self.stem_top_x_props
SNGT_QHENOMENOLOGY_420: (8)                     triangle = Polygon(
SNGT_QHENOMENOLOGY_421: (12)                        interpolate(lp, rp, low_prop),
SNGT_QHENOMENOLOGY_422: (12)                        interpolate(lp, rp, high_prop),
SNGT_QHENOMENOLOGY_423: (12)                        lp + stem_height * DOWN,
SNGT_QHENOMENOLOGY_424: (8)                     )
SNGT_QHENOMENOLOGY_425: (8)                     result = Union(rect, triangle)
SNGT_QHENOMENOLOGY_426: (8)                     result.insert_n_curves(20)
SNGT_QHENOMENOLOGY_427: (8)                     if direction[0] > 0:
SNGT_QHENOMENOLOGY_428: (12)                        result.flip()
SNGT_QHENOMENOLOGY_429: (8)                     return result
SNGT_QHENOMENOLOGY_430: (0)             class ThoughtBubble(Bubble):
SNGT_QHENOMENOLOGY_431: (4)                 def __init__(
SNGT_QHENOMENOLOGY_432: (8)                     self,
SNGT_QHENOMENOLOGY_433: (8)                     content: str | VMobject | None = None,
SNGT_QHENOMENOLOGY_434: (8)                     buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_435: (8)                     filler_shape: Tuple[float, float] = (2.0, 1.0),
SNGT_QHENOMENOLOGY_436: (8)                     bulge_radius: float = 0.35,
SNGT_QHENOMENOLOGY_437: (8)                     bulge_overlap: float = 0.25,
SNGT_QHENOMENOLOGY_438: (8)                     noise_factor: float = 0.1,
SNGT_QHENOMENOLOGY_439: (8)                     circle_radii: list[float] = [0.1, 0.15, 0.2],
SNGT_QHENOMENOLOGY_440: (8)                     **kwargs
SNGT_QHENOMENOLOGY_441: (4)                 ):
SNGT_QHENOMENOLOGY_442: (8)                     self.bulge_radius = bulge_radius
SNGT_QHENOMENOLOGY_443: (8)                     self.bulge_overlap = bulge_overlap
SNGT_QHENOMENOLOGY_444: (8)                     self.noise_factor = noise_factor
SNGT_QHENOMENOLOGY_445: (8)                     self.circle_radii = circle_radii
SNGT_QHENOMENOLOGY_446: (8)                     super().__init__(content, buff, filler_shape,
**kwargs)
SNGT_QHENOMENOLOGY_447: (4)                 def get_body(self, content: VMobject, direction: Vect3,
buff: float) -> VMobject:
SNGT_QHENOMENOLOGY_448: (8)                     rect = SurroundingRectangle(content, buff)
SNGT_QHENOMENOLOGY_449: (8)                     perimeter = rect.get_arc_length()
SNGT_QHENOMENOLOGY_450: (8)                     radius = self.bulge_radius
SNGT_QHENOMENOLOGY_451: (8)                     step = (1 - self.bulge_overlap) * (2 * radius)
SNGT_QHENOMENOLOGY_452: (8)                     nf = self.noise_factor
SNGT_QHENOMENOLOGY_453: (8)                     corners = [rect.get_corner(v) for v in [DL, UL, UR,
DR]]
SNGT_QHENOMENOLOGY_454: (8)                     points = []
SNGT_QHENOMENOLOGY_455: (8)                     for c1, c2 in adjacent_pairs(corners):
SNGT_QHENOMENOLOGY_456: (12)                        n_alphas = int(get_norm(c1 - c2) / step) + 1
SNGT_QHENOMENOLOGY_457: (12)                        for alpha in np.linspace(0, 1, n_alphas):
SNGT_QHENOMENOLOGY_458: (16)                            points.append(interpolate(
SNGT_QHENOMENOLOGY_459: (20)                                c1, c2, alpha + nf * (step / n_alphas)
* (random.random() - 0.5)
SNGT_QHENOMENOLOGY_460: (16)                            ))
SNGT_QHENOMENOLOGY_461: (8)                     cloud = Union(rect, *(
```

```
SNGT_QHENOMENOLOGY_462: (12)                         Circle(radius=radius * (1 + nf *
random.random())).move_to(point)
SNGT_QHENOMENOLOGY_463: (12)                         for point in points
SNGT_QHENOMENOLOGY_464: (8)                     ))
SNGT_QHENOMENOLOGY_465: (8)                     cloud.set_stroke(WHITE, 2)
SNGT_QHENOMENOLOGY_466: (8)                     circles = VGroup(Circle(radius=radius) for radius
in self.circle_radii)
SNGT_QHENOMENOLOGY_467: (8)                     circ_buff = 0.25 * self.circle_radii[0]
SNGT_QHENOMENOLOGY_468: (8)                     circles.arrange(UR, buff=circ_buff)
SNGT_QHENOMENOLOGY_469: (8)                     circles[1].shift(circ_buff * DR)
SNGT_QHENOMENOLOGY_470: (8)                     circles.next_to(cloud, DOWN, 4 * circ_buff,
aligned_edge=LEFT)
SNGT_QHENOMENOLOGY_471: (8)                     circles.set_stroke(WHITE, 2)
SNGT_QHENOMENOLOGY_472: (8)                     result = VGroup(*circles, cloud)
SNGT_QHENOMENOLOGY_473: (8)                     if direction[0] > 0:
SNGT_QHENOMENOLOGY_474: (12)                         result.flip()
SNGT_QHENOMENOLOGY_475: (8)                     return result
SNGT_QHENOMENOLOGY_476: (0)             class OldSpeechBubble(Bubble):
SNGT_QHENOMENOLOGY_477: (4)                 file_name: str = "Bubbles_speech.svg"
SNGT_QHENOMENOLOGY_478: (0)             class DoubleSpeechBubble(Bubble):
SNGT_QHENOMENOLOGY_479: (4)                 file_name: str = "Bubbles_double_speech.svg"
SNGT_QHENOMENOLOGY_480: (0)             class OldThoughtBubble(Bubble):
SNGT_QHENOMENOLOGY_481: (4)                 file_name: str = "Bubbles_thought.svg"
SNGT_QHENOMENOLOGY_482: (4)                 def get_body(self, content: VMobject, direction: Vect3,
buff: float) -> VMobject:
SNGT_QHENOMENOLOGY_483: (8)                     body = super().get_body(content, direction, buff)
SNGT_QHENOMENOLOGY_484: (8)                     body.sort(lambda p: p[1])
SNGT_QHENOMENOLOGY_485: (8)                     return body
SNGT_QHENOMENOLOGY_486: (4)                 def make_green_screen(self):
SNGT_QHENOMENOLOGY_487: (8)                     self.body[-1].set_fill(GREEN_SCREEN, opacity=1)
SNGT_QHENOMENOLOGY_488: (8)                     return self
SNGT_QHENOMENOLOGY_489: (0)             class VectorizedEarth(SVGMobject):
SNGT_QHENOMENOLOGY_490: (4)                 file_name: str = "earth"
SNGT_QHENOMENOLOGY_491: (4)                 def __init__(
SNGT_QHENOMENOLOGY_492: (8)                     self,
SNGT_QHENOMENOLOGY_493: (8)                     height: float = 2.0,
SNGT_QHENOMENOLOGY_494: (8)                     **kwargs
SNGT_QHENOMENOLOGY_495: (4)                 ):
SNGT_QHENOMENOLOGY_496: (8)                     super().__init__(height=height, **kwargs)
SNGT_QHENOMENOLOGY_497: (8)                     self.insert_n_curves(20)
SNGT_QHENOMENOLOGY_498: (8)                     circle = Circle(
SNGT_QHENOMENOLOGY_499: (12)                         stroke_width=3,
SNGT_QHENOMENOLOGY_500: (12)                         stroke_color=GREEN,
SNGT_QHENOMENOLOGY_501: (12)                         fill_opacity=1,
SNGT_QHENOMENOLOGY_502: (12)                         fill_color=BLUE_C,
SNGT_QHENOMENOLOGY_503: (8)                     )
SNGT_QHENOMENOLOGY_504: (8)                     circle.replace(self)
SNGT_QHENOMENOLOGY_505: (8)                     self.add_to_back(circle)
SNGT_QHENOMENOLOGY_506: (0)             class Piano(VGroup):
SNGT_QHENOMENOLOGY_507: (4)                 def __init__(
SNGT_QHENOMENOLOGY_508: (8)                     self,
SNGT_QHENOMENOLOGY_509: (8)                     n_white_keys = 52,
SNGT_QHENOMENOLOGY_510: (8)                     black_pattern = [0, 2, 3, 5, 6],
SNGT_QHENOMENOLOGY_511: (8)                     white_keys_per_octave = 7,
SNGT_QHENOMENOLOGY_512: (8)                     white_key_dims = (0.15, 1.0),
SNGT_QHENOMENOLOGY_513: (8)                     black_key_dims = (0.1, 0.66),
SNGT_QHENOMENOLOGY_514: (8)                     key_buff = 0.02,
SNGT_QHENOMENOLOGY_515: (8)                     white_key_color = WHITE,
SNGT_QHENOMENOLOGY_516: (8)                     black_key_color = GREY_E,
SNGT_QHENOMENOLOGY_517: (8)                     total_width = 13,
SNGT_QHENOMENOLOGY_518: (8)                     **kwargs
SNGT_QHENOMENOLOGY_519: (4)                 ):
SNGT_QHENOMENOLOGY_520: (8)                     self.n_white_keys = n_white_keys
SNGT_QHENOMENOLOGY_521: (8)                     self.black_pattern = black_pattern
SNGT_QHENOMENOLOGY_522: (8)                     self.white_keys_per_octave = white_keys_per_octave
SNGT_QHENOMENOLOGY_523: (8)                     self.white_key_dims = white_key_dims
SNGT_QHENOMENOLOGY_524: (8)                     self.black_key_dims = black_key_dims
SNGT_QHENOMENOLOGY_525: (8)                     self.key_buff = key_buff
SNGT_QHENOMENOLOGY_526: (8)                     self.white_key_color = white_key_color
```

```
SNGT_QHENOMENOLOGY_527: (8)                     self.black_key_color = black_key_color
SNGT_QHENOMENOLOGY_528: (8)                     self.total_width = total_width
SNGT_QHENOMENOLOGY_529: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_530: (8)                     self.add_white_keys()
SNGT_QHENOMENOLOGY_531: (8)                     self.add_black_keys()
SNGT_QHENOMENOLOGY_532: (8)                     self.sort_keys()
SNGT_QHENOMENOLOGY_533: (8)                     self[:-1].reverse_points()
SNGT_QHENOMENOLOGY_534: (8)                     self.set_width(self.total_width)
SNGT_QHENOMENOLOGY_535: (4)                 def add_white_keys(self):
SNGT_QHENOMENOLOGY_536: (8)                     key = Rectangle(*self.white_key_dims)
SNGT_QHENOMENOLOGY_537: (8)                     key.set_fill(self.white_key_color, 1)
SNGT_QHENOMENOLOGY_538: (8)                     key.set_stroke(width=0)
SNGT_QHENOMENOLOGY_539: (8)                     self.white_keys = key.get_grid(1,
self.n_white_keys, buff=self.key_buff)
SNGT_QHENOMENOLOGY_540: (8)                     self.add(*self.white_keys)
SNGT_QHENOMENOLOGY_541: (4)                 def add_black_keys(self):
SNGT_QHENOMENOLOGY_542: (8)                     key = Rectangle(*self.black_key_dims)
SNGT_QHENOMENOLOGY_543: (8)                     key.set_fill(self.black_key_color, 1)
SNGT_QHENOMENOLOGY_544: (8)                     key.set_stroke(width=0)
SNGT_QHENOMENOLOGY_545: (8)                     self.black_keys = VGroup()
SNGT_QHENOMENOLOGY_546: (8)                     for i in range(len(self.white_keys) - 1):
SNGT_QHENOMENOLOGY_547: (12)                        if i % self.white_keys_per_octave not in
self.black_pattern:
SNGT_QHENOMENOLOGY_548: (16)                            continue
SNGT_QHENOMENOLOGY_549: (12)                        wk1 = self.white_keys[i]
SNGT_QHENOMENOLOGY_550: (12)                        wk2 = self.white_keys[i + 1]
SNGT_QHENOMENOLOGY_551: (12)                        bk = key.copy()
SNGT_QHENOMENOLOGY_552: (12)                        bk.move_to(midpoint(wk1.get_top(),
wk2.get_top()), UP)
SNGT_QHENOMENOLOGY_553: (12)                        big_bk = bk.copy()
SNGT_QHENOMENOLOGY_554: (12)                        big_bk.stretch((bk.get_width() + self.key_buff)
/ bk.get_width(), 0)
SNGT_QHENOMENOLOGY_555: (12)                        big_bk.stretch((bk.get_height() +
self.key_buff) / bk.get_height(), 1)
SNGT_QHENOMENOLOGY_556: (12)                        big_bk.move_to(bk, UP)
SNGT_QHENOMENOLOGY_557: (12)                        for wk in wk1, wk2:
SNGT_QHENOMENOLOGY_558: (16)                            wk.become(Difference(wk,
big_bk).match_style(wk))
SNGT_QHENOMENOLOGY_559: (12)                        self.black_keys.add(bk)
SNGT_QHENOMENOLOGY_560: (8)                     self.add(*self.black_keys)
SNGT_QHENOMENOLOGY_561: (4)                 def sort_keys(self):
SNGT_QHENOMENOLOGY_562: (8)                     self.sort(lambda p: p[0])
SNGT_QHENOMENOLOGY_563: (0)             class Piano3D(VGroup):
SNGT_QHENOMENOLOGY_564: (4)                 def __init__(
SNGT_QHENOMENOLOGY_565: (8)                     self,
SNGT_QHENOMENOLOGY_566: (8)                     shading: Tuple[float, float, float] = (1.0, 0.2,
0.2),
SNGT_QHENOMENOLOGY_567: (8)                     stroke_width: float = 0.25,
SNGT_QHENOMENOLOGY_568: (8)                     stroke_color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_569: (8)                     key_depth: float = 0.1,
SNGT_QHENOMENOLOGY_570: (8)                     black_key_shift: float = 0.05,
SNGT_QHENOMENOLOGY_571: (8)                     piano_2d_config: dict = dict(
SNGT_QHENOMENOLOGY_572: (12)                        white_key_color=GREY_A,
SNGT_QHENOMENOLOGY_573: (12)                        key_buff=0.001
SNGT_QHENOMENOLOGY_574: (8)                     ),
SNGT_QHENOMENOLOGY_575: (8)                     **kwargs
SNGT_QHENOMENOLOGY_576: (4)                 ):
SNGT_QHENOMENOLOGY_577: (8)                     piano_2d = Piano(**piano_2d_config)
SNGT_QHENOMENOLOGY_578: (8)                     super().__init__(*(
SNGT_QHENOMENOLOGY_579: (12)                        Prismify(key, key_depth)
SNGT_QHENOMENOLOGY_580: (12)                        for key in piano_2d
SNGT_QHENOMENOLOGY_581: (8)                     ))
SNGT_QHENOMENOLOGY_582: (8)                     self.set_stroke(stroke_color, stroke_width)
SNGT_QHENOMENOLOGY_583: (8)                     self.set_shading(*shading)
SNGT_QHENOMENOLOGY_584: (8)                     self.apply_depth_test()
SNGT_QHENOMENOLOGY_585: (8)                     for i, key in enumerate(self):
SNGT_QHENOMENOLOGY_586: (12)                        if piano_2d[i] in piano_2d.black_keys:
SNGT_QHENOMENOLOGY_587: (16)                            key.shift(black_key_shift * OUT)
SNGT_QHENOMENOLOGY_588: (16)                            key.set_color(BLACK)
```

```
SNGT_QHENOMENOLOGY_589: (0)             class DieFace(VGroup):
SNGT_QHENOMENOLOGY_590: (4)                 def __init__(
SNGT_QHENOMENOLOGY_591: (8)                     self,
SNGT_QHENOMENOLOGY_592: (8)                     value: int,
SNGT_QHENOMENOLOGY_593: (8)                     side_length: float = 1.0,
SNGT_QHENOMENOLOGY_594: (8)                     corner_radius: float = 0.15,
SNGT_QHENOMENOLOGY_595: (8)                     stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_596: (8)                     stroke_width: float = 2.0,
SNGT_QHENOMENOLOGY_597: (8)                     fill_color: ManimColor = GREY_E,
SNGT_QHENOMENOLOGY_598: (8)                     dot_radius: float = 0.08,
SNGT_QHENOMENOLOGY_599: (8)                     dot_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_600: (8)                     dot_coalesce_factor: float = 0.5
SNGT_QHENOMENOLOGY_601: (4)                 ):
SNGT_QHENOMENOLOGY_602: (8)                     dot = Dot(radius=dot_radius, fill_color=dot_color)
SNGT_QHENOMENOLOGY_603: (8)                     square = Square(
SNGT_QHENOMENOLOGY_604: (12)                        side_length=side_length,
SNGT_QHENOMENOLOGY_605: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_606: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_607: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_608: (12)                        fill_opacity=1.0,
SNGT_QHENOMENOLOGY_609: (8)                     )
SNGT_QHENOMENOLOGY_610: (8)                     square.round_corners(corner_radius)
SNGT_QHENOMENOLOGY_611: (8)                     if not (1 <= value <= 6):
SNGT_QHENOMENOLOGY_612: (12)                        raise Exception("DieFace only accepts integer
inputs between 1 and 6")
SNGT_QHENOMENOLOGY_613: (8)                     edge_group = [
SNGT_QHENOMENOLOGY_614: (12)                        (ORIGIN,),
SNGT_QHENOMENOLOGY_615: (12)                        (UL, DR),
SNGT_QHENOMENOLOGY_616: (12)                        (UL, ORIGIN, DR),
SNGT_QHENOMENOLOGY_617: (12)                        (UL, UR, DL, DR),
SNGT_QHENOMENOLOGY_618: (12)                        (UL, UR, ORIGIN, DL, DR),
SNGT_QHENOMENOLOGY_619: (12)                        (UL, UR, LEFT, RIGHT, DL, DR),
SNGT_QHENOMENOLOGY_620: (8)                     ][value - 1]
SNGT_QHENOMENOLOGY_621: (8)                     arrangement = VGroup(*(
SNGT_QHENOMENOLOGY_622: (12)
dot.copy().move_to(square.get_bounding_box_point(vect))
SNGT_QHENOMENOLOGY_623: (12)                        for vect in edge_group
SNGT_QHENOMENOLOGY_624: (8)                     ))
SNGT_QHENOMENOLOGY_625: (8)
arrangement.space_out_submobjects(dot_coalesce_factor)
SNGT_QHENOMENOLOGY_626: (8)                     super().__init__(square, arrangement)
SNGT_QHENOMENOLOGY_627: (8)                     self.dots = arrangement
SNGT_QHENOMENOLOGY_628: (8)                     self.value = value
SNGT_QHENOMENOLOGY_629: (8)                     self.index = value
SNGT_QHENOMENOLOGY_630: (0)             class Dartboard(VGroup):
SNGT_QHENOMENOLOGY_631: (4)                 radius = 3
SNGT_QHENOMENOLOGY_632: (4)                 n_sectors = 20
SNGT_QHENOMENOLOGY_633: (4)                 def __init__(self, **kwargs):
SNGT_QHENOMENOLOGY_634: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_635: (8)                     n_sectors = self.n_sectors
SNGT_QHENOMENOLOGY_636: (8)                     angle = TAU / n_sectors
SNGT_QHENOMENOLOGY_637: (8)                     segments = VGroup(*[
SNGT_QHENOMENOLOGY_638: (12)                        VGroup(*[
SNGT_QHENOMENOLOGY_639: (16)                            AnnularSector(
SNGT_QHENOMENOLOGY_640: (20)                                inner_radius=in_r,
SNGT_QHENOMENOLOGY_641: (20)                                outer_radius=out_r,
SNGT_QHENOMENOLOGY_642: (20)                                start_angle=n * angle,
SNGT_QHENOMENOLOGY_643: (20)                                angle=angle,
SNGT_QHENOMENOLOGY_644: (20)                                fill_color=color,
SNGT_QHENOMENOLOGY_645: (16)                            )
SNGT_QHENOMENOLOGY_646: (16)                            for n, color in zip(
SNGT_QHENOMENOLOGY_647: (20)                                range(n_sectors),
SNGT_QHENOMENOLOGY_648: (20)                                it.cycle(colors)
SNGT_QHENOMENOLOGY_649: (16)                            )
SNGT_QHENOMENOLOGY_650: (12)                        ])
SNGT_QHENOMENOLOGY_651: (12)                        for colors, in_r, out_r in [
SNGT_QHENOMENOLOGY_652: (16)                            ([GREY_B, GREY_E], 0, 1),
SNGT_QHENOMENOLOGY_653: (16)                            ([GREEN_E, RED_E], 0.5, 0.55),
SNGT_QHENOMENOLOGY_654: (16)                            ([GREEN_E, RED_E], 0.95, 1),
```

```
SNGT_QHENOMENOLOGY_655: (12)                    ]
SNGT_QHENOMENOLOGY_656: (8)                 ])
SNGT_QHENOMENOLOGY_657: (8)                 segments.rotate(-angle / 2)
SNGT_QHENOMENOLOGY_658: (8)                 bullseyes = VGroup(*[
SNGT_QHENOMENOLOGY_659: (12)                    Circle(radius=r)
SNGT_QHENOMENOLOGY_660: (12)                    for r in [0.07, 0.035]
SNGT_QHENOMENOLOGY_661: (8)                 ])
SNGT_QHENOMENOLOGY_662: (8)                 bullseyes.set_fill(opacity=1)
SNGT_QHENOMENOLOGY_663: (8)                 bullseyes.set_stroke(width=0)
SNGT_QHENOMENOLOGY_664: (8)                 bullseyes[0].set_color(GREEN_E)
SNGT_QHENOMENOLOGY_665: (8)                 bullseyes[1].set_color(RED_E)
SNGT_QHENOMENOLOGY_666: (8)                 self.bullseye = bullseyes[1]
SNGT_QHENOMENOLOGY_667: (8)                 self.add(*segments, *bullseyes)
SNGT_QHENOMENOLOGY_668: (8)                 self.scale(self.radius)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 49 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 50 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 51 - dot_cloud.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import moderngl
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import GREY_C, YELLOW
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import ORIGIN, NULL_POINTS
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.types.point_cloud_mobject import
PMobject
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_9: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                  import numpy.typing as npt
SNGT_QHENOMENOLOGY_12: (4)                  from typing import Sequence, Tuple
SNGT_QHENOMENOLOGY_13: (4)                  from manimlib.typing import ManimColor, Vect3,
Vect3Array, Self
SNGT_QHENOMENOLOGY_14: (0)              DEFAULT_DOT_RADIUS = 0.05
SNGT_QHENOMENOLOGY_15: (0)              DEFAULT_GLOW_DOT_RADIUS = 0.2
SNGT_QHENOMENOLOGY_16: (0)              DEFAULT_GRID_HEIGHT = 6
SNGT_QHENOMENOLOGY_17: (0)              DEFAULT_BUFF_RATIO = 0.5
SNGT_QHENOMENOLOGY_18: (0)              class DotCloud(PMobject):
SNGT_QHENOMENOLOGY_19: (4)                  shader_folder: str = "true_dot"
SNGT_QHENOMENOLOGY_20: (4)                  render_primitive: int = moderngl.POINTS
SNGT_QHENOMENOLOGY_21: (4)                  data_dtype: Sequence[Tuple[str, type, Tuple[int]]] = [
SNGT_QHENOMENOLOGY_22: (8)                      ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_23: (8)                      ('radius', np.float32, (1,)),
SNGT_QHENOMENOLOGY_24: (8)                      ('rgba', np.float32, (4,)),
SNGT_QHENOMENOLOGY_25: (4)                  ]
SNGT_QHENOMENOLOGY_26: (4)                  def __init__(
SNGT_QHENOMENOLOGY_27: (8)                      self,
SNGT_QHENOMENOLOGY_28: (8)                      points: Vect3Array = NULL_POINTS,
SNGT_QHENOMENOLOGY_29: (8)                      color: ManimColor = GREY_C,
SNGT_QHENOMENOLOGY_30: (8)                      opacity: float = 1.0,
SNGT_QHENOMENOLOGY_31: (8)                      radius: float = DEFAULT_DOT_RADIUS,
SNGT_QHENOMENOLOGY_32: (8)                      glow_factor: float = 0.0,
SNGT_QHENOMENOLOGY_33: (8)                      anti_alias_width: float = 2.0,
SNGT_QHENOMENOLOGY_34: (8)                      **kwargs
SNGT_QHENOMENOLOGY_35: (4)                  ):
```

```
SNGT_QHENOMENOLOGY_36: (8)                      self.radius = radius
SNGT_QHENOMENOLOGY_37: (8)                      self.glow_factor = glow_factor
SNGT_QHENOMENOLOGY_38: (8)                      self.anti_alias_width = anti_alias_width
SNGT_QHENOMENOLOGY_39: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_40: (12)                         color=color,
SNGT_QHENOMENOLOGY_41: (12)                         opacity=opacity,
SNGT_QHENOMENOLOGY_42: (12)                         **kwargs
SNGT_QHENOMENOLOGY_43: (8)                      )
SNGT_QHENOMENOLOGY_44: (8)                      self.set_radius(self.radius)
SNGT_QHENOMENOLOGY_45: (8)                      if points is not None:
SNGT_QHENOMENOLOGY_46: (12)                         self.set_points(points)
SNGT_QHENOMENOLOGY_47: (4)                  def init_uniforms(self) -> None:
SNGT_QHENOMENOLOGY_48: (8)                      super().init_uniforms()
SNGT_QHENOMENOLOGY_49: (8)                      self.uniforms["glow_factor"] = self.glow_factor
SNGT_QHENOMENOLOGY_50: (8)                      self.uniforms["anti_alias_width"] =
self.anti_alias_width
SNGT_QHENOMENOLOGY_51: (4)                  def to_grid(
SNGT_QHENOMENOLOGY_52: (8)                      self,
SNGT_QHENOMENOLOGY_53: (8)                      n_rows: int,
SNGT_QHENOMENOLOGY_54: (8)                      n_cols: int,
SNGT_QHENOMENOLOGY_55: (8)                      n_layers: int = 1,
SNGT_QHENOMENOLOGY_56: (8)                      buff_ratio: float | None = None,
SNGT_QHENOMENOLOGY_57: (8)                      h_buff_ratio: float = 1.0,
SNGT_QHENOMENOLOGY_58: (8)                      v_buff_ratio: float = 1.0,
SNGT_QHENOMENOLOGY_59: (8)                      d_buff_ratio: float = 1.0,
SNGT_QHENOMENOLOGY_60: (8)                      height: float = DEFAULT_GRID_HEIGHT,
SNGT_QHENOMENOLOGY_61: (4)                  ) -> Self:
SNGT_QHENOMENOLOGY_62: (8)                      n_points = n_rows * n_cols * n_layers
SNGT_QHENOMENOLOGY_63: (8)                      points = np.repeat(range(n_points), 3,
axis=0).reshape((n_points, 3))
SNGT_QHENOMENOLOGY_64: (8)                      points[:, 0] = points[:, 0] % n_cols
SNGT_QHENOMENOLOGY_65: (8)                      points[:, 1] = (points[:, 1] // n_cols) % n_rows
SNGT_QHENOMENOLOGY_66: (8)                      points[:, 2] = points[:, 2] // (n_rows * n_cols)
SNGT_QHENOMENOLOGY_67: (8)                      self.set_points(points.astype(float))
SNGT_QHENOMENOLOGY_68: (8)                      if buff_ratio is not None:
SNGT_QHENOMENOLOGY_69: (12)                         v_buff_ratio = buff_ratio
SNGT_QHENOMENOLOGY_70: (12)                         h_buff_ratio = buff_ratio
SNGT_QHENOMENOLOGY_71: (12)                         d_buff_ratio = buff_ratio
SNGT_QHENOMENOLOGY_72: (8)                      radius = self.get_radius()
SNGT_QHENOMENOLOGY_73: (8)                      ns = [n_cols, n_rows, n_layers]
SNGT_QHENOMENOLOGY_74: (8)                      brs = [h_buff_ratio, v_buff_ratio, d_buff_ratio]
SNGT_QHENOMENOLOGY_75: (8)                      self.set_radius(0)
SNGT_QHENOMENOLOGY_76: (8)                      for n, br, dim in zip(ns, brs, range(3)):
SNGT_QHENOMENOLOGY_77: (12)                         self.rescale_to_fit(2 * radius * (1 + br) * (n
- 1), dim, stretch=True)
SNGT_QHENOMENOLOGY_78: (8)                      self.set_radius(radius)
SNGT_QHENOMENOLOGY_79: (8)                      if height is not None:
SNGT_QHENOMENOLOGY_80: (12)                         self.set_height(height)
SNGT_QHENOMENOLOGY_81: (8)                      self.center()
SNGT_QHENOMENOLOGY_82: (8)                      return self
SNGT_QHENOMENOLOGY_83: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_84: (4)                  def set_radii(self, radii: npt.ArrayLike) -> Self:
SNGT_QHENOMENOLOGY_85: (8)                      n_points = self.get_num_points()
SNGT_QHENOMENOLOGY_86: (8)                      radii = np.array(radii).reshape((len(radii), 1))
SNGT_QHENOMENOLOGY_87: (8)                      self.data["radius"][:] =
resize_with_interpolation(radii, n_points)
SNGT_QHENOMENOLOGY_88: (8)                      self.refresh_bounding_box()
SNGT_QHENOMENOLOGY_89: (8)                      return self
SNGT_QHENOMENOLOGY_90: (4)                  def get_radii(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_91: (8)                      return self.data["radius"]
SNGT_QHENOMENOLOGY_92: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_93: (4)                  def set_radius(self, radius: float) -> Self:
SNGT_QHENOMENOLOGY_94: (8)                      data = self.data if self.get_num_points() > 0 else
self._data_defaults
SNGT_QHENOMENOLOGY_95: (8)                      data["radius"][:] = radius
SNGT_QHENOMENOLOGY_96: (8)                      self.refresh_bounding_box()
SNGT_QHENOMENOLOGY_97: (8)                      return self
SNGT_QHENOMENOLOGY_98: (4)                  def get_radius(self) -> float:
SNGT_QHENOMENOLOGY_99: (8)                      return self.get_radii().max()
```

```
SNGT_QHENOMENOLOGY_100: (4)                 def scale_radii(self, scale_factor: float) -> Self:
SNGT_QHENOMENOLOGY_101: (8)                     self.set_radius(scale_factor * self.get_radii())
SNGT_QHENOMENOLOGY_102: (8)                     return self
SNGT_QHENOMENOLOGY_103: (4)                 def set_glow_factor(self, glow_factor: float) -> Self:
SNGT_QHENOMENOLOGY_104: (8)                     self.uniforms["glow_factor"] = glow_factor
SNGT_QHENOMENOLOGY_105: (8)                     return self
SNGT_QHENOMENOLOGY_106: (4)                 def get_glow_factor(self) -> float:
SNGT_QHENOMENOLOGY_107: (8)                     return self.uniforms["glow_factor"]
SNGT_QHENOMENOLOGY_108: (4)                 def compute_bounding_box(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_109: (8)                     bb = super().compute_bounding_box()
SNGT_QHENOMENOLOGY_110: (8)                     radius = self.get_radius()
SNGT_QHENOMENOLOGY_111: (8)                     bb[0] += np.full((3,), -radius)
SNGT_QHENOMENOLOGY_112: (8)                     bb[2] += np.full((3,), radius)
SNGT_QHENOMENOLOGY_113: (8)                     return bb
SNGT_QHENOMENOLOGY_114: (4)                 def scale(
SNGT_QHENOMENOLOGY_115: (8)                     self,
SNGT_QHENOMENOLOGY_116: (8)                     scale_factor: float | npt.ArrayLike,
SNGT_QHENOMENOLOGY_117: (8)                     scale_radii: bool = True,
SNGT_QHENOMENOLOGY_118: (8)                     **kwargs
SNGT_QHENOMENOLOGY_119: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_120: (8)                     super().scale(scale_factor, **kwargs)
SNGT_QHENOMENOLOGY_121: (8)                     if scale_radii:
SNGT_QHENOMENOLOGY_122: (12)                        self.set_radii(scale_factor * self.get_radii())
SNGT_QHENOMENOLOGY_123: (8)                     return self
SNGT_QHENOMENOLOGY_124: (4)                 def make_3d(
SNGT_QHENOMENOLOGY_125: (8)                     self,
SNGT_QHENOMENOLOGY_126: (8)                     reflectiveness: float = 0.5,
SNGT_QHENOMENOLOGY_127: (8)                     gloss: float = 0.1,
SNGT_QHENOMENOLOGY_128: (8)                     shadow: float = 0.2
SNGT_QHENOMENOLOGY_129: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_130: (8)                     self.set_shading(reflectiveness, gloss, shadow)
SNGT_QHENOMENOLOGY_131: (8)                     self.apply_depth_test()
SNGT_QHENOMENOLOGY_132: (8)                     return self
SNGT_QHENOMENOLOGY_133: (0)             class TrueDot(DotCloud):
SNGT_QHENOMENOLOGY_134: (4)                 def __init__(self, center: Vect3 = ORIGIN, **kwargs):
SNGT_QHENOMENOLOGY_135: (8)                     super().__init__(points=np.array([center]),
**kwargs)
SNGT_QHENOMENOLOGY_136: (0)             class GlowDots(DotCloud):
SNGT_QHENOMENOLOGY_137: (4)                 def __init__(
SNGT_QHENOMENOLOGY_138: (8)                     self,
SNGT_QHENOMENOLOGY_139: (8)                     points: Vect3Array = NULL_POINTS,
SNGT_QHENOMENOLOGY_140: (8)                     color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_141: (8)                     radius: float = DEFAULT_GLOW_DOT_RADIUS,
SNGT_QHENOMENOLOGY_142: (8)                     glow_factor: float = 2.0,
SNGT_QHENOMENOLOGY_143: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_144: (4)                 ):
SNGT_QHENOMENOLOGY_145: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_146: (12)                        points,
SNGT_QHENOMENOLOGY_147: (12)                        color=color,
SNGT_QHENOMENOLOGY_148: (12)                        radius=radius,
SNGT_QHENOMENOLOGY_149: (12)                        glow_factor=glow_factor,
SNGT_QHENOMENOLOGY_150: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_151: (8)                     )
SNGT_QHENOMENOLOGY_152: (0)             class GlowDot(GlowDots):
SNGT_QHENOMENOLOGY_153: (4)                 def __init__(self, center: Vect3 = ORIGIN, **kwargs):
SNGT_QHENOMENOLOGY_154: (8)                     super().__init__(points=np.array([center]),
**kwargs)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 52 - probability.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.constants import BLUE, BLUE_E, GREEN_E,
GREY_B, GREY_D, MAROON_B, YELLOW
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import DOWN, LEFT, RIGHT, UP
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import MED_LARGE_BUFF,
MED_SMALL_BUFF, SMALL_BUFF
```

```
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.svg.brace import Brace
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.svg.tex_mobject import TexText
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.color import color_gradient
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_15: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_16: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_17: (4)                  from typing import Iterable
SNGT_QHENOMENOLOGY_18: (4)                  from manimlib.typing import ManimColor
SNGT_QHENOMENOLOGY_19: (0)              EPSILON = 0.0001
SNGT_QHENOMENOLOGY_20: (0)              class SampleSpace(Rectangle):
SNGT_QHENOMENOLOGY_21: (4)                  def __init__(
SNGT_QHENOMENOLOGY_22: (8)                      self,
SNGT_QHENOMENOLOGY_23: (8)                      width: float = 3,
SNGT_QHENOMENOLOGY_24: (8)                      height: float = 3,
SNGT_QHENOMENOLOGY_25: (8)                      fill_color: ManimColor = GREY_D,
SNGT_QHENOMENOLOGY_26: (8)                      fill_opacity: float = 1,
SNGT_QHENOMENOLOGY_27: (8)                      stroke_width: float = 0.5,
SNGT_QHENOMENOLOGY_28: (8)                      stroke_color: ManimColor = GREY_B,
SNGT_QHENOMENOLOGY_29: (8)                      default_label_scale_val: float = 1,
SNGT_QHENOMENOLOGY_30: (8)                      **kwargs,
SNGT_QHENOMENOLOGY_31: (4)                  ):
SNGT_QHENOMENOLOGY_32: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_33: (12)                         width, height,
SNGT_QHENOMENOLOGY_34: (12)                         fill_color=fill_color,
SNGT_QHENOMENOLOGY_35: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_36: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_37: (12)                         stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_38: (8)                      )
SNGT_QHENOMENOLOGY_39: (8)                      self.default_label_scale_val =
default_label_scale_val
SNGT_QHENOMENOLOGY_40: (4)                  def add_title(
SNGT_QHENOMENOLOGY_41: (8)                      self,
SNGT_QHENOMENOLOGY_42: (8)                      title: str = "Sample space",
SNGT_QHENOMENOLOGY_43: (8)                      buff: float = MED_SMALL_BUFF
SNGT_QHENOMENOLOGY_44: (4)                  ) -> None:
SNGT_QHENOMENOLOGY_45: (8)                      title_mob = TexText(title)
SNGT_QHENOMENOLOGY_46: (8)                      if title_mob.get_width() > self.get_width():
SNGT_QHENOMENOLOGY_47: (12)                         title_mob.set_width(self.get_width())
SNGT_QHENOMENOLOGY_48: (8)                      title_mob.next_to(self, UP, buff=buff)
SNGT_QHENOMENOLOGY_49: (8)                      self.title = title_mob
SNGT_QHENOMENOLOGY_50: (8)                      self.add(title_mob)
SNGT_QHENOMENOLOGY_51: (4)                  def add_label(self, label: str) -> None:
SNGT_QHENOMENOLOGY_52: (8)                      self.label = label
SNGT_QHENOMENOLOGY_53: (4)                  def complete_p_list(self, p_list: list[float]) ->
list[float]:
SNGT_QHENOMENOLOGY_54: (8)                      new_p_list = listify(p_list)
SNGT_QHENOMENOLOGY_55: (8)                      remainder = 1.0 - sum(new_p_list)
SNGT_QHENOMENOLOGY_56: (8)                      if abs(remainder) > EPSILON:
SNGT_QHENOMENOLOGY_57: (12)                         new_p_list.append(remainder)
SNGT_QHENOMENOLOGY_58: (8)                      return new_p_list
SNGT_QHENOMENOLOGY_59: (4)                  def get_division_along_dimension(
SNGT_QHENOMENOLOGY_60: (8)                      self,
SNGT_QHENOMENOLOGY_61: (8)                      p_list: list[float],
SNGT_QHENOMENOLOGY_62: (8)                      dim: int,
SNGT_QHENOMENOLOGY_63: (8)                      colors: Iterable[ManimColor],
SNGT_QHENOMENOLOGY_64: (8)                      vect: np.ndarray
SNGT_QHENOMENOLOGY_65: (4)                  ) -> VGroup:
SNGT_QHENOMENOLOGY_66: (8)                      p_list = self.complete_p_list(p_list)
SNGT_QHENOMENOLOGY_67: (8)                      colors = color_gradient(colors, len(p_list))
SNGT_QHENOMENOLOGY_68: (8)                      last_point = self.get_edge_center(-vect)
SNGT_QHENOMENOLOGY_69: (8)                      parts = VGroup()
SNGT_QHENOMENOLOGY_70: (8)                      for factor, color in zip(p_list, colors):
SNGT_QHENOMENOLOGY_71: (12)                         part = SampleSpace()
```

```
SNGT_QHENOMENOLOGY_72: (12)                         part.set_fill(color, 1)
SNGT_QHENOMENOLOGY_73: (12)                         part.replace(self, stretch=True)
SNGT_QHENOMENOLOGY_74: (12)                         part.stretch(factor, dim)
SNGT_QHENOMENOLOGY_75: (12)                         part.move_to(last_point, -vect)
SNGT_QHENOMENOLOGY_76: (12)                         last_point = part.get_edge_center(vect)
SNGT_QHENOMENOLOGY_77: (12)                         parts.add(part)
SNGT_QHENOMENOLOGY_78: (8)                     return parts
SNGT_QHENOMENOLOGY_79: (4)                 def get_horizontal_division(
SNGT_QHENOMENOLOGY_80: (8)                     self,
SNGT_QHENOMENOLOGY_81: (8)                     p_list: list[float],
SNGT_QHENOMENOLOGY_82: (8)                     colors: Iterable[ManimColor] = [GREEN_E, BLUE_E],
SNGT_QHENOMENOLOGY_83: (8)                     vect: np.ndarray = DOWN
SNGT_QHENOMENOLOGY_84: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_85: (8)                     return self.get_division_along_dimension(p_list, 1,
colors, vect)
SNGT_QHENOMENOLOGY_86: (4)                 def get_vertical_division(
SNGT_QHENOMENOLOGY_87: (8)                     self,
SNGT_QHENOMENOLOGY_88: (8)                     p_list: list[float],
SNGT_QHENOMENOLOGY_89: (8)                     colors: Iterable[ManimColor] = [MAROON_B, YELLOW],
SNGT_QHENOMENOLOGY_90: (8)                     vect: np.ndarray = RIGHT
SNGT_QHENOMENOLOGY_91: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_92: (8)                     return self.get_division_along_dimension(p_list, 0,
colors, vect)
SNGT_QHENOMENOLOGY_93: (4)                 def divide_horizontally(self, *args, **kwargs) -> None:
SNGT_QHENOMENOLOGY_94: (8)                     self.horizontal_parts =
self.get_horizontal_division(*args, **kwargs)
SNGT_QHENOMENOLOGY_95: (8)                     self.add(self.horizontal_parts)
SNGT_QHENOMENOLOGY_96: (4)                 def divide_vertically(self, *args, **kwargs) -> None:
SNGT_QHENOMENOLOGY_97: (8)                     self.vertical_parts =
self.get_vertical_division(*args, **kwargs)
SNGT_QHENOMENOLOGY_98: (8)                     self.add(self.vertical_parts)
SNGT_QHENOMENOLOGY_99: (4)                 def get_subdivision_braces_and_labels(
SNGT_QHENOMENOLOGY_100: (8)                     self,
SNGT_QHENOMENOLOGY_101: (8)                     parts: VGroup,
SNGT_QHENOMENOLOGY_102: (8)                     labels: str,
SNGT_QHENOMENOLOGY_103: (8)                     direction: np.ndarray,
SNGT_QHENOMENOLOGY_104: (8)                     buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_105: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_106: (8)                     label_mobs = VGroup()
SNGT_QHENOMENOLOGY_107: (8)                     braces = VGroup()
SNGT_QHENOMENOLOGY_108: (8)                     for label, part in zip(labels, parts):
SNGT_QHENOMENOLOGY_109: (12)                         brace = Brace(
SNGT_QHENOMENOLOGY_110: (16)                             part, direction,
SNGT_QHENOMENOLOGY_111: (16)                             buff=buff
SNGT_QHENOMENOLOGY_112: (12)                         )
SNGT_QHENOMENOLOGY_113: (12)                         if isinstance(label, Mobject):
SNGT_QHENOMENOLOGY_114: (16)                             label_mob = label
SNGT_QHENOMENOLOGY_115: (12)                         else:
SNGT_QHENOMENOLOGY_116: (16)                             label_mob = Tex(label)
SNGT_QHENOMENOLOGY_117: (16)
label_mob.scale(self.default_label_scale_val)
SNGT_QHENOMENOLOGY_118: (12)                         label_mob.next_to(brace, direction, buff)
SNGT_QHENOMENOLOGY_119: (12)                         braces.add(brace)
SNGT_QHENOMENOLOGY_120: (12)                         label_mobs.add(label_mob)
SNGT_QHENOMENOLOGY_121: (8)                     parts.braces = braces
SNGT_QHENOMENOLOGY_122: (8)                     parts.labels = label_mobs
SNGT_QHENOMENOLOGY_123: (8)                     parts.label_kwargs = {
SNGT_QHENOMENOLOGY_124: (12)                         "labels": label_mobs.copy(),
SNGT_QHENOMENOLOGY_125: (12)                         "direction": direction,
SNGT_QHENOMENOLOGY_126: (12)                         "buff": buff,
SNGT_QHENOMENOLOGY_127: (8)                     }
SNGT_QHENOMENOLOGY_128: (8)                     return VGroup(parts.braces, parts.labels)
SNGT_QHENOMENOLOGY_129: (4)                 def get_side_braces_and_labels(
SNGT_QHENOMENOLOGY_130: (8)                     self,
SNGT_QHENOMENOLOGY_131: (8)                     labels: str,
SNGT_QHENOMENOLOGY_132: (8)                     direction: np.ndarray = LEFT,
SNGT_QHENOMENOLOGY_133: (8)                     **kwargs
SNGT_QHENOMENOLOGY_134: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_135: (8)                     assert hasattr(self, "horizontal_parts")
```

```
SNGT_QHENOMENOLOGY_136: (8)                     parts = self.horizontal_parts
SNGT_QHENOMENOLOGY_137: (8)                     return
self.get_subdivision_braces_and_labels(parts, labels, direction, **kwargs)
SNGT_QHENOMENOLOGY_138: (4)                 def get_top_braces_and_labels(
SNGT_QHENOMENOLOGY_139: (8)                     self,
SNGT_QHENOMENOLOGY_140: (8)                     labels: str,
SNGT_QHENOMENOLOGY_141: (8)                     **kwargs
SNGT_QHENOMENOLOGY_142: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_143: (8)                     assert hasattr(self, "vertical_parts")
SNGT_QHENOMENOLOGY_144: (8)                     parts = self.vertical_parts
SNGT_QHENOMENOLOGY_145: (8)                     return
self.get_subdivision_braces_and_labels(parts, labels, UP, **kwargs)
SNGT_QHENOMENOLOGY_146: (4)                 def get_bottom_braces_and_labels(
SNGT_QHENOMENOLOGY_147: (8)                     self,
SNGT_QHENOMENOLOGY_148: (8)                     labels: str,
SNGT_QHENOMENOLOGY_149: (8)                     **kwargs
SNGT_QHENOMENOLOGY_150: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_151: (8)                     assert hasattr(self, "vertical_parts")
SNGT_QHENOMENOLOGY_152: (8)                     parts = self.vertical_parts
SNGT_QHENOMENOLOGY_153: (8)                     return
self.get_subdivision_braces_and_labels(parts, labels, DOWN, **kwargs)
SNGT_QHENOMENOLOGY_154: (4)                 def add_braces_and_labels(self) -> None:
SNGT_QHENOMENOLOGY_155: (8)                     for attr in "horizontal_parts", "vertical_parts":
SNGT_QHENOMENOLOGY_156: (12)                        if not hasattr(self, attr):
SNGT_QHENOMENOLOGY_157: (16)                            continue
SNGT_QHENOMENOLOGY_158: (12)                        parts = getattr(self, attr)
SNGT_QHENOMENOLOGY_159: (12)                        for subattr in "braces", "labels":
SNGT_QHENOMENOLOGY_160: (16)                            if hasattr(parts, subattr):
SNGT_QHENOMENOLOGY_161: (20)                                self.add(getattr(parts, subattr))
SNGT_QHENOMENOLOGY_162: (4)                 def __getitem__(self, index: int | slice) -> VGroup:
SNGT_QHENOMENOLOGY_163: (8)                     if hasattr(self, "horizontal_parts"):
SNGT_QHENOMENOLOGY_164: (12)                        return self.horizontal_parts[index]
SNGT_QHENOMENOLOGY_165: (8)                     elif hasattr(self, "vertical_parts"):
SNGT_QHENOMENOLOGY_166: (12)                        return self.vertical_parts[index]
SNGT_QHENOMENOLOGY_167: (8)                     return self.split()[index]
SNGT_QHENOMENOLOGY_168: (0)             class BarChart(VGroup):
SNGT_QHENOMENOLOGY_169: (4)                 def __init__(
SNGT_QHENOMENOLOGY_170: (8)                     self,
SNGT_QHENOMENOLOGY_171: (8)                     values: Iterable[float],
SNGT_QHENOMENOLOGY_172: (8)                     height: float = 4,
SNGT_QHENOMENOLOGY_173: (8)                     width: float = 6,
SNGT_QHENOMENOLOGY_174: (8)                     n_ticks: int = 4,
SNGT_QHENOMENOLOGY_175: (8)                     include_x_ticks: bool = False,
SNGT_QHENOMENOLOGY_176: (8)                     tick_width: float = 0.2,
SNGT_QHENOMENOLOGY_177: (8)                     tick_height: float = 0.15,
SNGT_QHENOMENOLOGY_178: (8)                     label_y_axis: bool = True,
SNGT_QHENOMENOLOGY_179: (8)                     y_axis_label_height: float = 0.25,
SNGT_QHENOMENOLOGY_180: (8)                     max_value: float = 1,
SNGT_QHENOMENOLOGY_181: (8)                     bar_colors: list[ManimColor] = [BLUE, YELLOW],
SNGT_QHENOMENOLOGY_182: (8)                     bar_fill_opacity: float = 0.8,
SNGT_QHENOMENOLOGY_183: (8)                     bar_stroke_width: float = 3,
SNGT_QHENOMENOLOGY_184: (8)                     bar_names: list[str] = [],
SNGT_QHENOMENOLOGY_185: (8)                     bar_label_scale_val: float = 0.75,
SNGT_QHENOMENOLOGY_186: (8)                     **kwargs
SNGT_QHENOMENOLOGY_187: (4)                 ):
SNGT_QHENOMENOLOGY_188: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_189: (8)                     self.height = height
SNGT_QHENOMENOLOGY_190: (8)                     self.width = width
SNGT_QHENOMENOLOGY_191: (8)                     self.n_ticks = n_ticks
SNGT_QHENOMENOLOGY_192: (8)                     self.include_x_ticks = include_x_ticks
SNGT_QHENOMENOLOGY_193: (8)                     self.tick_width = tick_width
SNGT_QHENOMENOLOGY_194: (8)                     self.tick_height = tick_height
SNGT_QHENOMENOLOGY_195: (8)                     self.label_y_axis = label_y_axis
SNGT_QHENOMENOLOGY_196: (8)                     self.y_axis_label_height = y_axis_label_height
SNGT_QHENOMENOLOGY_197: (8)                     self.max_value = max_value
SNGT_QHENOMENOLOGY_198: (8)                     self.bar_colors = bar_colors
SNGT_QHENOMENOLOGY_199: (8)                     self.bar_fill_opacity = bar_fill_opacity
SNGT_QHENOMENOLOGY_200: (8)                     self.bar_stroke_width = bar_stroke_width
SNGT_QHENOMENOLOGY_201: (8)                     self.bar_names = bar_names
```

```
SNGT_QHENOMENOLOGY_202: (8)                     self.bar_label_scale_val = bar_label_scale_val
SNGT_QHENOMENOLOGY_203: (8)                     if self.max_value is None:
SNGT_QHENOMENOLOGY_204: (12)                        self.max_value = max(values)
SNGT_QHENOMENOLOGY_205: (8)                     self.n_ticks_x = len(values)
SNGT_QHENOMENOLOGY_206: (8)                     self.add_axes()
SNGT_QHENOMENOLOGY_207: (8)                     self.add_bars(values)
SNGT_QHENOMENOLOGY_208: (8)                     self.center()
SNGT_QHENOMENOLOGY_209: (4)                 def add_axes(self) -> None:
SNGT_QHENOMENOLOGY_210: (8)                     x_axis = Line(self.tick_width * LEFT / 2,
self.width * RIGHT)
SNGT_QHENOMENOLOGY_211: (8)                     y_axis = Line(MED_LARGE_BUFF * DOWN, self.height *
UP)
SNGT_QHENOMENOLOGY_212: (8)                     y_ticks = VGroup()
SNGT_QHENOMENOLOGY_213: (8)                     heights = np.linspace(0, self.height, self.n_ticks
+ 1)
SNGT_QHENOMENOLOGY_214: (8)                     values = np.linspace(0, self.max_value,
self.n_ticks + 1)
SNGT_QHENOMENOLOGY_215: (8)                     for y, value in zip(heights, values):
SNGT_QHENOMENOLOGY_216: (12)                        y_tick = Line(LEFT, RIGHT)
SNGT_QHENOMENOLOGY_217: (12)                        y_tick.set_width(self.tick_width)
SNGT_QHENOMENOLOGY_218: (12)                        y_tick.move_to(y * UP)
SNGT_QHENOMENOLOGY_219: (12)                        y_ticks.add(y_tick)
SNGT_QHENOMENOLOGY_220: (8)                     y_axis.add(y_ticks)
SNGT_QHENOMENOLOGY_221: (8)                     if self.include_x_ticks == True:
SNGT_QHENOMENOLOGY_222: (12)                        x_ticks = VGroup()
SNGT_QHENOMENOLOGY_223: (12)                        widths = np.linspace(0, self.width,
self.n_ticks_x + 1)
SNGT_QHENOMENOLOGY_224: (12)                        label_values = np.linspace(0,
len(self.bar_names), self.n_ticks_x + 1)
SNGT_QHENOMENOLOGY_225: (12)                        for x, value in zip(widths, label_values):
SNGT_QHENOMENOLOGY_226: (16)                            x_tick = Line(UP, DOWN)
SNGT_QHENOMENOLOGY_227: (16)                            x_tick.set_height(self.tick_height)
SNGT_QHENOMENOLOGY_228: (16)                            x_tick.move_to(x * RIGHT)
SNGT_QHENOMENOLOGY_229: (16)                            x_ticks.add(x_tick)
SNGT_QHENOMENOLOGY_230: (12)                        x_axis.add(x_ticks)
SNGT_QHENOMENOLOGY_231: (8)                     self.add(x_axis, y_axis)
SNGT_QHENOMENOLOGY_232: (8)                     self.x_axis, self.y_axis = x_axis, y_axis
SNGT_QHENOMENOLOGY_233: (8)                     if self.label_y_axis:
SNGT_QHENOMENOLOGY_234: (12)                        labels = VGroup()
SNGT_QHENOMENOLOGY_235: (12)                        for y_tick, value in zip(y_ticks, values):
SNGT_QHENOMENOLOGY_236: (16)                            label = Tex(str(np.round(value, 2)))
SNGT_QHENOMENOLOGY_237: (16)                            label.set_height(self.y_axis_label_height)
SNGT_QHENOMENOLOGY_238: (16)                            label.next_to(y_tick, LEFT, SMALL_BUFF)
SNGT_QHENOMENOLOGY_239: (16)                            labels.add(label)
SNGT_QHENOMENOLOGY_240: (12)                        self.y_axis_labels = labels
SNGT_QHENOMENOLOGY_241: (12)                        self.add(labels)
SNGT_QHENOMENOLOGY_242: (4)                 def add_bars(self, values: Iterable[float]) -> None:
SNGT_QHENOMENOLOGY_243: (8)                     buff = float(self.width) / (2 * len(values))
SNGT_QHENOMENOLOGY_244: (8)                     bars = VGroup()
SNGT_QHENOMENOLOGY_245: (8)                     for i, value in enumerate(values):
SNGT_QHENOMENOLOGY_246: (12)                        bar = Rectangle(
SNGT_QHENOMENOLOGY_247: (16)                            height=(value / self.max_value) *
self.height,
SNGT_QHENOMENOLOGY_248: (16)                            width=buff,
SNGT_QHENOMENOLOGY_249: (16)                            stroke_width=self.bar_stroke_width,
SNGT_QHENOMENOLOGY_250: (16)                            fill_opacity=self.bar_fill_opacity,
SNGT_QHENOMENOLOGY_251: (12)                        )
SNGT_QHENOMENOLOGY_252: (12)                        bar.move_to((2 * i + 0.5) * buff * RIGHT, DOWN
+ LEFT * 5)
SNGT_QHENOMENOLOGY_253: (12)                        bars.add(bar)
SNGT_QHENOMENOLOGY_254: (8)                     bars.set_color_by_gradient(*self.bar_colors)
SNGT_QHENOMENOLOGY_255: (8)                     bar_labels = VGroup()
SNGT_QHENOMENOLOGY_256: (8)                     for bar, name in zip(bars, self.bar_names):
SNGT_QHENOMENOLOGY_257: (12)                        label = Tex(str(name))
SNGT_QHENOMENOLOGY_258: (12)                        label.scale(self.bar_label_scale_val)
SNGT_QHENOMENOLOGY_259: (12)                        label.next_to(bar, DOWN, SMALL_BUFF)
SNGT_QHENOMENOLOGY_260: (12)                        bar_labels.add(label)
SNGT_QHENOMENOLOGY_261: (8)                     self.add(bars, bar_labels)
SNGT_QHENOMENOLOGY_262: (8)                     self.bars = bars
```

```
SNGT_QHENOMENOLOGY_263: (8)                     self.bar_labels = bar_labels
SNGT_QHENOMENOLOGY_264: (4)                 def change_bar_values(self, values: Iterable[float]) ->
None:
SNGT_QHENOMENOLOGY_265: (8)                     for bar, value in zip(self.bars, values):
SNGT_QHENOMENOLOGY_266: (12)                        bar_bottom = bar.get_bottom()
SNGT_QHENOMENOLOGY_267: (12)                        bar.stretch_to_fit_height(
SNGT_QHENOMENOLOGY_268: (16)                            (value / self.max_value) * self.height
SNGT_QHENOMENOLOGY_269: (12)                        )
SNGT_QHENOMENOLOGY_270: (12)                        bar.move_to(bar_bottom, DOWN)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 53 - special_tex.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.constants import MED_SMALL_BUFF, WHITE,
GREY_C
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.constants import DOWN, LEFT, RIGHT, UP
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import FRAME_WIDTH
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import MED_LARGE_BUFF, SMALL_BUFF
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.svg.tex_mobject import TexText
SNGT_QHENOMENOLOGY_9: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                  from manimlib.typing import ManimColor, Vect3
SNGT_QHENOMENOLOGY_12: (0)              class BulletedList(VGroup):
SNGT_QHENOMENOLOGY_13: (4)                  def __init__(
SNGT_QHENOMENOLOGY_14: (8)                      self,
SNGT_QHENOMENOLOGY_15: (8)                      *items: str,
SNGT_QHENOMENOLOGY_16: (8)                      buff: float = MED_LARGE_BUFF,
SNGT_QHENOMENOLOGY_17: (8)                      aligned_edge: Vect3 = LEFT,
SNGT_QHENOMENOLOGY_18: (8)                      **kwargs
SNGT_QHENOMENOLOGY_19: (4)                  ):
SNGT_QHENOMENOLOGY_20: (8)                      labelled_content = [R"\item " + item for item in
items]
SNGT_QHENOMENOLOGY_21: (8)                      tex_string = "\n".join([
SNGT_QHENOMENOLOGY_22: (12)                         R"\begin{itemize}",
SNGT_QHENOMENOLOGY_23: (12)                         *labelled_content,
SNGT_QHENOMENOLOGY_24: (12)                         R"\end{itemize}"
SNGT_QHENOMENOLOGY_25: (8)                      ])
SNGT_QHENOMENOLOGY_26: (8)                      tex_text = TexText(tex_string,
isolate=labelled_content, **kwargs)
SNGT_QHENOMENOLOGY_27: (8)                      lines = (tex_text.select_part(part) for part in
labelled_content)
SNGT_QHENOMENOLOGY_28: (8)                      super().__init__(*lines)
SNGT_QHENOMENOLOGY_29: (8)                      self.arrange(DOWN, buff=buff,
aligned_edge=aligned_edge)
SNGT_QHENOMENOLOGY_30: (4)                  def fade_all_but(self, index: int, opacity: float =
0.25) -> None:
SNGT_QHENOMENOLOGY_31: (8)                      for i, part in enumerate(self.submobjects):
SNGT_QHENOMENOLOGY_32: (12)                         part.set_fill(opacity=(1.0 if i == index else
opacity))
SNGT_QHENOMENOLOGY_33: (0)              class TexTextFromPresetString(TexText):
SNGT_QHENOMENOLOGY_34: (4)                  tex: str = ""
SNGT_QHENOMENOLOGY_35: (4)                  default_color: ManimColor = WHITE
SNGT_QHENOMENOLOGY_36: (4)                  def __init__(self, **kwargs):
SNGT_QHENOMENOLOGY_37: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_38: (12)                         self.tex,
SNGT_QHENOMENOLOGY_39: (12)                         color=kwargs.pop("color", self.default_color),
SNGT_QHENOMENOLOGY_40: (12)                         **kwargs
SNGT_QHENOMENOLOGY_41: (8)                      )
SNGT_QHENOMENOLOGY_42: (0)              class Title(TexText):
SNGT_QHENOMENOLOGY_43: (4)                  def __init__(
SNGT_QHENOMENOLOGY_44: (8)                      self,
SNGT_QHENOMENOLOGY_45: (8)                      *text_parts: str,
SNGT_QHENOMENOLOGY_46: (8)                      font_size: int = 72,
SNGT_QHENOMENOLOGY_47: (8)                      include_underline: bool = True,
```

```
SNGT_QHENOMENOLOGY_48: (8)                      underline_width: float = FRAME_WIDTH - 2,
SNGT_QHENOMENOLOGY_49: (8)                      match_underline_width_to_text: bool = False,
SNGT_QHENOMENOLOGY_50: (8)                      underline_buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_51: (8)                      underline_style: dict = dict(stroke_width=2,
stroke_color=GREY_C),
SNGT_QHENOMENOLOGY_52: (8)                      **kwargs
SNGT_QHENOMENOLOGY_53: (4)                  ):
SNGT_QHENOMENOLOGY_54: (8)                      super().__init__(*text_parts, font_size=font_size,
**kwargs)
SNGT_QHENOMENOLOGY_55: (8)                      self.to_edge(UP, buff=MED_SMALL_BUFF)
SNGT_QHENOMENOLOGY_56: (8)                      if include_underline:
SNGT_QHENOMENOLOGY_57: (12)                         underline = Line(LEFT, RIGHT,
**underline_style)
SNGT_QHENOMENOLOGY_58: (12)                         underline.next_to(self, DOWN,
buff=underline_buff)
SNGT_QHENOMENOLOGY_59: (12)                         if match_underline_width_to_text:
SNGT_QHENOMENOLOGY_60: (16)                             underline.match_width(self)
SNGT_QHENOMENOLOGY_61: (12)                         else:
SNGT_QHENOMENOLOGY_62: (16)                             underline.set_width(underline_width)
SNGT_QHENOMENOLOGY_63: (12)                         self.add(underline)
SNGT_QHENOMENOLOGY_64: (12)                         self.underline = underline
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 54 - svg_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from xml.etree import ElementTree as ET
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               import svgelements as se
SNGT_QHENOMENOLOGY_5: (0)               import io
SNGT_QHENOMENOLOGY_6: (0)               from pathlib import Path
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import RIGHT
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.logger import log
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.geometry import Circle
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.geometry import Polygon
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.geometry import Polyline
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.mobject.geometry import RoundedRectangle
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.images import
get_full_vector_image_path
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.iterables import hash_obj
SNGT_QHENOMENOLOGY_18: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_19: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_20: (4)                  from manimlib.typing import ManimColor, Vect3Array
SNGT_QHENOMENOLOGY_21: (0)              SVG_HASH_TO_MOB_MAP: dict[int, list[VMobject]] = {}
SNGT_QHENOMENOLOGY_22: (0)              PATH_TO_POINTS: dict[str, Vect3Array] = {}
SNGT_QHENOMENOLOGY_23: (0)              def _convert_point_to_3d(x: float, y: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_24: (4)                  return np.array([x, y, 0.0])
SNGT_QHENOMENOLOGY_25: (0)              class SVGMobject(VMobject):
SNGT_QHENOMENOLOGY_26: (4)                  file_name: str = ""
SNGT_QHENOMENOLOGY_27: (4)                  height: float | None = 2.0
SNGT_QHENOMENOLOGY_28: (4)                  width: float | None = None
SNGT_QHENOMENOLOGY_29: (4)                  def __init__(
SNGT_QHENOMENOLOGY_30: (8)                      self,
SNGT_QHENOMENOLOGY_31: (8)                      file_name: str = "",
SNGT_QHENOMENOLOGY_32: (8)                      svg_string: str = "",
SNGT_QHENOMENOLOGY_33: (8)                      should_center: bool = True,
SNGT_QHENOMENOLOGY_34: (8)                      height: float | None = None,
SNGT_QHENOMENOLOGY_35: (8)                      width: float | None = None,
SNGT_QHENOMENOLOGY_36: (8)                      color: ManimColor = None,
SNGT_QHENOMENOLOGY_37: (8)                      fill_color: ManimColor = None,
SNGT_QHENOMENOLOGY_38: (8)                      fill_opacity: float | None = None,
SNGT_QHENOMENOLOGY_39: (8)                      stroke_width: float | None = 0.0,
SNGT_QHENOMENOLOGY_40: (8)                      stroke_color: ManimColor = None,
SNGT_QHENOMENOLOGY_41: (8)                      stroke_opacity: float | None = None,
```

```
SNGT_QHENOMENOLOGY_42: (8)                      svg_default: dict = dict(
SNGT_QHENOMENOLOGY_43: (12)                         color=None,
SNGT_QHENOMENOLOGY_44: (12)                         opacity=None,
SNGT_QHENOMENOLOGY_45: (12)                         fill_color=None,
SNGT_QHENOMENOLOGY_46: (12)                         fill_opacity=None,
SNGT_QHENOMENOLOGY_47: (12)                         stroke_width=None,
SNGT_QHENOMENOLOGY_48: (12)                         stroke_color=None,
SNGT_QHENOMENOLOGY_49: (12)                         stroke_opacity=None,
SNGT_QHENOMENOLOGY_50: (8)                      ),
SNGT_QHENOMENOLOGY_51: (8)                      path_string_config: dict = dict(),
SNGT_QHENOMENOLOGY_52: (8)                      **kwargs
SNGT_QHENOMENOLOGY_53: (4)                  ):
SNGT_QHENOMENOLOGY_54: (8)                      if svg_string != "":
SNGT_QHENOMENOLOGY_55: (12)                         self.svg_string = svg_string
SNGT_QHENOMENOLOGY_56: (8)                      elif file_name != "":
SNGT_QHENOMENOLOGY_57: (12)                         self.svg_string =
self.file_name_to_svg_string(file_name)
SNGT_QHENOMENOLOGY_58: (8)                      elif self.file_name != "":
SNGT_QHENOMENOLOGY_59: (12)                         self.file_name_to_svg_string(self.file_name)
SNGT_QHENOMENOLOGY_60: (8)                      else:
SNGT_QHENOMENOLOGY_61: (12)                         raise Exception("Must specify either a
file_name or svg_string SVGMobject")
SNGT_QHENOMENOLOGY_62: (8)                      self.svg_default = dict(svg_default)
SNGT_QHENOMENOLOGY_63: (8)                      self.path_string_config = dict(path_string_config)
SNGT_QHENOMENOLOGY_64: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_65: (8)                      self.init_svg_mobject()
SNGT_QHENOMENOLOGY_66: (8)                      self.ensure_positive_orientation()
SNGT_QHENOMENOLOGY_67: (8)                      self.set_style(
SNGT_QHENOMENOLOGY_68: (12)                         fill_color=color or fill_color,
SNGT_QHENOMENOLOGY_69: (12)                         fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_70: (12)                         stroke_color=color or stroke_color,
SNGT_QHENOMENOLOGY_71: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_72: (12)                         stroke_opacity=stroke_opacity,
SNGT_QHENOMENOLOGY_73: (8)                      )
SNGT_QHENOMENOLOGY_74: (8)                      height = height or self.height
SNGT_QHENOMENOLOGY_75: (8)                      width = width or self.width
SNGT_QHENOMENOLOGY_76: (8)                      if should_center:
SNGT_QHENOMENOLOGY_77: (12)                         self.center()
SNGT_QHENOMENOLOGY_78: (8)                      if height is not None:
SNGT_QHENOMENOLOGY_79: (12)                         self.set_height(height)
SNGT_QHENOMENOLOGY_80: (8)                      if width is not None:
SNGT_QHENOMENOLOGY_81: (12)                         self.set_width(width)
SNGT_QHENOMENOLOGY_82: (4)                  def init_svg_mobject(self) -> None:
SNGT_QHENOMENOLOGY_83: (8)                      hash_val = hash_obj(self.hash_seed)
SNGT_QHENOMENOLOGY_84: (8)                      if hash_val in SVG_HASH_TO_MOB_MAP:
SNGT_QHENOMENOLOGY_85: (12)                         submobs = [sm.copy() for sm in
SVG_HASH_TO_MOB_MAP[hash_val]]
SNGT_QHENOMENOLOGY_86: (8)                      else:
SNGT_QHENOMENOLOGY_87: (12)                         submobs =
self.mobjects_from_svg_string(self.svg_string)
SNGT_QHENOMENOLOGY_88: (12)                         SVG_HASH_TO_MOB_MAP[hash_val] = [sm.copy() for
sm in submobs]
SNGT_QHENOMENOLOGY_89: (8)                      self.add(*submobs)
SNGT_QHENOMENOLOGY_90: (8)                      self.flip(RIGHT)  # Flip y
SNGT_QHENOMENOLOGY_91: (4)                  @property
SNGT_QHENOMENOLOGY_92: (4)                  def hash_seed(self) -> tuple:
SNGT_QHENOMENOLOGY_93: (8)                      return (
SNGT_QHENOMENOLOGY_94: (12)                         self.__class__.__name__,
SNGT_QHENOMENOLOGY_95: (12)                         self.svg_default,
SNGT_QHENOMENOLOGY_96: (12)                         self.path_string_config,
SNGT_QHENOMENOLOGY_97: (12)                         self.svg_string
SNGT_QHENOMENOLOGY_98: (8)                      )
SNGT_QHENOMENOLOGY_99: (4)                  def mobjects_from_svg_string(self, svg_string: str) ->
list[VMobject]:
SNGT_QHENOMENOLOGY_100: (8)                     element_tree =
ET.ElementTree(ET.fromstring(svg_string))
SNGT_QHENOMENOLOGY_101: (8)                     new_tree = self.modify_xml_tree(element_tree)
SNGT_QHENOMENOLOGY_102: (8)                     data_stream = io.BytesIO()
SNGT_QHENOMENOLOGY_103: (8)                     new_tree.write(data_stream)
```

```
SNGT_QHENOMENOLOGY_104: (8)                     data_stream.seek(0)
SNGT_QHENOMENOLOGY_105: (8)                     svg = se.SVG.parse(data_stream)
SNGT_QHENOMENOLOGY_106: (8)                     data_stream.close()
SNGT_QHENOMENOLOGY_107: (8)                     return self.mobjects_from_svg(svg)
SNGT_QHENOMENOLOGY_108: (4)                 def file_name_to_svg_string(self, file_name: str) ->
str:
SNGT_QHENOMENOLOGY_109: (8)                     return
Path(get_full_vector_image_path(file_name)).read_text()
SNGT_QHENOMENOLOGY_110: (4)                 def modify_xml_tree(self, element_tree: ET.ElementTree)
-> ET.ElementTree:
SNGT_QHENOMENOLOGY_111: (8)                     config_style_attrs =
self.generate_config_style_dict()
SNGT_QHENOMENOLOGY_112: (8)                     style_keys = (
SNGT_QHENOMENOLOGY_113: (12)                        "fill",
SNGT_QHENOMENOLOGY_114: (12)                        "fill-opacity",
SNGT_QHENOMENOLOGY_115: (12)                        "stroke",
SNGT_QHENOMENOLOGY_116: (12)                        "stroke-opacity",
SNGT_QHENOMENOLOGY_117: (12)                        "stroke-width",
SNGT_QHENOMENOLOGY_118: (12)                        "style"
SNGT_QHENOMENOLOGY_119: (8)                     )
SNGT_QHENOMENOLOGY_120: (8)                     root = element_tree.getroot()
SNGT_QHENOMENOLOGY_121: (8)                     style_attrs = {
SNGT_QHENOMENOLOGY_122: (12)                        k: v
SNGT_QHENOMENOLOGY_123: (12)                        for k, v in root.attrib.items()
SNGT_QHENOMENOLOGY_124: (12)                        if k in style_keys
SNGT_QHENOMENOLOGY_125: (8)                     }
SNGT_QHENOMENOLOGY_126: (8)                     SVG_XMLNS = "{http://www.w3.org/2000/svg}"
SNGT_QHENOMENOLOGY_127: (8)                     new_root = ET.Element("svg")
SNGT_QHENOMENOLOGY_128: (8)                     config_style_node = ET.SubElement(new_root, f"
{SVG_XMLNS}g", config_style_attrs)
SNGT_QHENOMENOLOGY_129: (8)                     root_style_node = ET.SubElement(config_style_node,
f"{SVG_XMLNS}g", style_attrs)
SNGT_QHENOMENOLOGY_130: (8)                     root_style_node.extend(root)
SNGT_QHENOMENOLOGY_131: (8)                     return ET.ElementTree(new_root)
SNGT_QHENOMENOLOGY_132: (4)                 def generate_config_style_dict(self) -> dict[str, str]:
SNGT_QHENOMENOLOGY_133: (8)                     keys_converting_dict = {
SNGT_QHENOMENOLOGY_134: (12)                        "fill": ("color", "fill_color"),
SNGT_QHENOMENOLOGY_135: (12)                        "fill-opacity": ("opacity", "fill_opacity"),
SNGT_QHENOMENOLOGY_136: (12)                        "stroke": ("color", "stroke_color"),
SNGT_QHENOMENOLOGY_137: (12)                        "stroke-opacity": ("opacity",
"stroke_opacity"),
SNGT_QHENOMENOLOGY_138: (12)                        "stroke-width": ("stroke_width",)
SNGT_QHENOMENOLOGY_139: (8)                     }
SNGT_QHENOMENOLOGY_140: (8)                     svg_default_dict = self.svg_default
SNGT_QHENOMENOLOGY_141: (8)                     result = {}
SNGT_QHENOMENOLOGY_142: (8)                     for svg_key, style_keys in
keys_converting_dict.items():
SNGT_QHENOMENOLOGY_143: (12)                        for style_key in style_keys:
SNGT_QHENOMENOLOGY_144: (16)                            if svg_default_dict[style_key] is None:
SNGT_QHENOMENOLOGY_145: (20)                                continue
SNGT_QHENOMENOLOGY_146: (16)                            result[svg_key] =
str(svg_default_dict[style_key])
SNGT_QHENOMENOLOGY_147: (8)                     return result
SNGT_QHENOMENOLOGY_148: (4)                 def mobjects_from_svg(self, svg: se.SVG) ->
list[VMobject]:
SNGT_QHENOMENOLOGY_149: (8)                     result = []
SNGT_QHENOMENOLOGY_150: (8)                     for shape in svg.elements():
SNGT_QHENOMENOLOGY_151: (12)                        if isinstance(shape, (se.Group, se.Use)):
SNGT_QHENOMENOLOGY_152: (16)                            continue
SNGT_QHENOMENOLOGY_153: (12)                        elif isinstance(shape, se.Path):
SNGT_QHENOMENOLOGY_154: (16)                            mob = self.path_to_mobject(shape)
SNGT_QHENOMENOLOGY_155: (12)                        elif isinstance(shape, se.SimpleLine):
SNGT_QHENOMENOLOGY_156: (16)                            mob = self.line_to_mobject(shape)
SNGT_QHENOMENOLOGY_157: (12)                        elif isinstance(shape, se.Rect):
SNGT_QHENOMENOLOGY_158: (16)                            mob = self.rect_to_mobject(shape)
SNGT_QHENOMENOLOGY_159: (12)                        elif isinstance(shape, (se.Circle,
se.Ellipse)):
SNGT_QHENOMENOLOGY_160: (16)                            mob = self.ellipse_to_mobject(shape)
SNGT_QHENOMENOLOGY_161: (12)                        elif isinstance(shape, se.Polygon):
```

```
SNGT_QHENOMENOLOGY_162: (16)                            mob = self.polygon_to_mobject(shape)
SNGT_QHENOMENOLOGY_163: (12)                        elif isinstance(shape, se.Polyline):
SNGT_QHENOMENOLOGY_164: (16)                            mob = self.polyline_to_mobject(shape)
SNGT_QHENOMENOLOGY_165: (12)                        elif type(shape) == se.SVGElement:
SNGT_QHENOMENOLOGY_166: (16)                            continue
SNGT_QHENOMENOLOGY_167: (12)                        else:
SNGT_QHENOMENOLOGY_168: (16)                            log.warning("Unsupported element type: %s",
type(shape))
SNGT_QHENOMENOLOGY_169: (16)                            continue
SNGT_QHENOMENOLOGY_170: (12)                        if not mob.has_points():
SNGT_QHENOMENOLOGY_171: (16)                            continue
SNGT_QHENOMENOLOGY_172: (12)                        if isinstance(shape, se.GraphicObject):
SNGT_QHENOMENOLOGY_173: (16)                            self.apply_style_to_mobject(mob, shape)
SNGT_QHENOMENOLOGY_174: (12)                        if isinstance(shape, se.Transformable) and
shape.apply:
SNGT_QHENOMENOLOGY_175: (16)                            self.handle_transform(mob, shape.transform)
SNGT_QHENOMENOLOGY_176: (12)                        result.append(mob)
SNGT_QHENOMENOLOGY_177: (8)                 return result
SNGT_QHENOMENOLOGY_178: (4)             @staticmethod
SNGT_QHENOMENOLOGY_179: (4)             def handle_transform(mob: VMobject, matrix: se.Matrix)
-> VMobject:
SNGT_QHENOMENOLOGY_180: (8)                 mat = np.array([
SNGT_QHENOMENOLOGY_181: (12)                    [matrix.a, matrix.c],
SNGT_QHENOMENOLOGY_182: (12)                    [matrix.b, matrix.d]
SNGT_QHENOMENOLOGY_183: (8)                 ])
SNGT_QHENOMENOLOGY_184: (8)                 vec = np.array([matrix.e, matrix.f, 0.0])
SNGT_QHENOMENOLOGY_185: (8)                 mob.apply_matrix(mat)
SNGT_QHENOMENOLOGY_186: (8)                 mob.shift(vec)
SNGT_QHENOMENOLOGY_187: (8)                 return mob
SNGT_QHENOMENOLOGY_188: (4)             @staticmethod
SNGT_QHENOMENOLOGY_189: (4)             def apply_style_to_mobject(
SNGT_QHENOMENOLOGY_190: (8)                 mob: VMobject,
SNGT_QHENOMENOLOGY_191: (8)                 shape: se.GraphicObject
SNGT_QHENOMENOLOGY_192: (4)             ) -> VMobject:
SNGT_QHENOMENOLOGY_193: (8)                 mob.set_style(
SNGT_QHENOMENOLOGY_194: (12)                    stroke_width=shape.stroke_width,
SNGT_QHENOMENOLOGY_195: (12)                    stroke_color=shape.stroke.hexrgb,
SNGT_QHENOMENOLOGY_196: (12)                    stroke_opacity=shape.stroke.opacity,
SNGT_QHENOMENOLOGY_197: (12)                    fill_color=shape.fill.hexrgb,
SNGT_QHENOMENOLOGY_198: (12)                    fill_opacity=shape.fill.opacity
SNGT_QHENOMENOLOGY_199: (8)                 )
SNGT_QHENOMENOLOGY_200: (8)                 return mob
SNGT_QHENOMENOLOGY_201: (4)             def path_to_mobject(self, path: se.Path) ->
VMobjectFromSVGPath:
SNGT_QHENOMENOLOGY_202: (8)                 return VMobjectFromSVGPath(path,
**self.path_string_config)
SNGT_QHENOMENOLOGY_203: (4)             def line_to_mobject(self, line: se.SimpleLine) -> Line:
SNGT_QHENOMENOLOGY_204: (8)                 return Line(
SNGT_QHENOMENOLOGY_205: (12)                    start=_convert_point_to_3d(line.x1, line.y1),
SNGT_QHENOMENOLOGY_206: (12)                    end=_convert_point_to_3d(line.x2, line.y2)
SNGT_QHENOMENOLOGY_207: (8)                 )
SNGT_QHENOMENOLOGY_208: (4)             def rect_to_mobject(self, rect: se.Rect) -> Rectangle:
SNGT_QHENOMENOLOGY_209: (8)                 if rect.rx == 0 or rect.ry == 0:
SNGT_QHENOMENOLOGY_210: (12)                    mob = Rectangle(
SNGT_QHENOMENOLOGY_211: (16)                        width=rect.width,
SNGT_QHENOMENOLOGY_212: (16)                        height=rect.height,
SNGT_QHENOMENOLOGY_213: (12)                    )
SNGT_QHENOMENOLOGY_214: (8)                 else:
SNGT_QHENOMENOLOGY_215: (12)                    mob = RoundedRectangle(
SNGT_QHENOMENOLOGY_216: (16)                        width=rect.width,
SNGT_QHENOMENOLOGY_217: (16)                        height=rect.height * rect.rx / rect.ry,
SNGT_QHENOMENOLOGY_218: (16)                        corner_radius=rect.rx
SNGT_QHENOMENOLOGY_219: (12)                    )
SNGT_QHENOMENOLOGY_220: (12)                    mob.stretch_to_fit_height(rect.height)
SNGT_QHENOMENOLOGY_221: (8)                 mob.shift(_convert_point_to_3d(
SNGT_QHENOMENOLOGY_222: (12)                    rect.x + rect.width / 2,
SNGT_QHENOMENOLOGY_223: (12)                    rect.y + rect.height / 2
SNGT_QHENOMENOLOGY_224: (8)                 ))
SNGT_QHENOMENOLOGY_225: (8)                 return mob
```

```
SNGT_QHENOMENOLOGY_226: (4)                 def ellipse_to_mobject(self, ellipse: se.Circle |
se.Ellipse) -> Circle:
SNGT_QHENOMENOLOGY_227: (8)                     mob = Circle(radius=ellipse.rx)
SNGT_QHENOMENOLOGY_228: (8)                     mob.stretch_to_fit_height(2 * ellipse.ry)
SNGT_QHENOMENOLOGY_229: (8)                     mob.shift(_convert_point_to_3d(
SNGT_QHENOMENOLOGY_230: (12)                        ellipse.cx, ellipse.cy
SNGT_QHENOMENOLOGY_231: (8)                     ))
SNGT_QHENOMENOLOGY_232: (8)                     return mob
SNGT_QHENOMENOLOGY_233: (4)                 def polygon_to_mobject(self, polygon: se.Polygon) ->
Polygon:
SNGT_QHENOMENOLOGY_234: (8)                     points = [
SNGT_QHENOMENOLOGY_235: (12)                        _convert_point_to_3d(*point)
SNGT_QHENOMENOLOGY_236: (12)                        for point in polygon
SNGT_QHENOMENOLOGY_237: (8)                     ]
SNGT_QHENOMENOLOGY_238: (8)                     return Polygon(*points)
SNGT_QHENOMENOLOGY_239: (4)                 def polyline_to_mobject(self, polyline: se.Polyline) ->
Polyline:
SNGT_QHENOMENOLOGY_240: (8)                     points = [
SNGT_QHENOMENOLOGY_241: (12)                        _convert_point_to_3d(*point)
SNGT_QHENOMENOLOGY_242: (12)                        for point in polyline
SNGT_QHENOMENOLOGY_243: (8)                     ]
SNGT_QHENOMENOLOGY_244: (8)                     return Polyline(*points)
SNGT_QHENOMENOLOGY_245: (4)                 def text_to_mobject(self, text: se.Text):
SNGT_QHENOMENOLOGY_246: (8)                     pass
SNGT_QHENOMENOLOGY_247: (0)             class VMobjectFromSVGPath(VMobject):
SNGT_QHENOMENOLOGY_248: (4)                 def __init__(
SNGT_QHENOMENOLOGY_249: (8)                     self,
SNGT_QHENOMENOLOGY_250: (8)                     path_obj: se.Path,
SNGT_QHENOMENOLOGY_251: (8)                     **kwargs
SNGT_QHENOMENOLOGY_252: (4)                 ):
SNGT_QHENOMENOLOGY_253: (8)                     path_obj.approximate_arcs_with_quads()
SNGT_QHENOMENOLOGY_254: (8)                     self.path_obj = path_obj
SNGT_QHENOMENOLOGY_255: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_256: (4)                 def init_points(self) -> None:
SNGT_QHENOMENOLOGY_257: (8)                     path_string = self.path_obj.d()
SNGT_QHENOMENOLOGY_258: (8)                     if path_string not in PATH_TO_POINTS:
SNGT_QHENOMENOLOGY_259: (12)                        self.handle_commands()
SNGT_QHENOMENOLOGY_260: (12)                        PATH_TO_POINTS[path_string] =
self.get_points().copy()
SNGT_QHENOMENOLOGY_261: (8)                     else:
SNGT_QHENOMENOLOGY_262: (12)                        points = PATH_TO_POINTS[path_string]
SNGT_QHENOMENOLOGY_263: (12)                        self.set_points(points)
SNGT_QHENOMENOLOGY_264: (4)                 def handle_commands(self) -> None:
SNGT_QHENOMENOLOGY_265: (8)                     segment_class_to_func_map = {
SNGT_QHENOMENOLOGY_266: (12)                        se.Move: (self.start_new_path, ("end",)),
SNGT_QHENOMENOLOGY_267: (12)                        se.Close: (self.close_path, ()),
SNGT_QHENOMENOLOGY_268: (12)                        se.Line: (lambda p: self.add_line_to(p,
allow_null_line=False), ("end",)),
SNGT_QHENOMENOLOGY_269: (12)                        se.QuadraticBezier: (lambda c, e:
self.add_quadratic_bezier_curve_to(c, e, allow_null_curve=False), ("control", "end")),
SNGT_QHENOMENOLOGY_270: (12)                        se.CubicBezier:
(self.add_cubic_bezier_curve_to, ("control1", "control2", "end"))
SNGT_QHENOMENOLOGY_271: (8)                     }
SNGT_QHENOMENOLOGY_272: (8)                     for segment in self.path_obj:
SNGT_QHENOMENOLOGY_273: (12)                        segment_class = segment.__class__
SNGT_QHENOMENOLOGY_274: (12)                        func, attr_names =
segment_class_to_func_map[segment_class]
SNGT_QHENOMENOLOGY_275: (12)                        points = [
SNGT_QHENOMENOLOGY_276: (16)
_convert_point_to_3d(*segment.__getattribute__(attr_name))
SNGT_QHENOMENOLOGY_277: (16)                            for attr_name in attr_names
SNGT_QHENOMENOLOGY_278: (12)                        ]
SNGT_QHENOMENOLOGY_279: (12)                        func(*points)
SNGT_QHENOMENOLOGY_280: (8)                     if self.has_new_path_started():
SNGT_QHENOMENOLOGY_281: (12)                        self.resize_points(self.get_num_points() - 2)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 55 - tex_mobject.py:
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import re
SNGT_QHENOMENOLOGY_3: (0)              from pathlib import Path
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.mobject.svg.string_mobject import
StringMobject
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.utils.color import color_to_hex
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.utils.color import hex_to_int
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.utils.tex_file_writing import latex_to_svg
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.utils.tex import num_tex_symbols
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.logger import log
SNGT_QHENOMENOLOGY_12: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_13: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_14: (4)                 from manimlib.typing import ManimColor, Span, Selector,
Self
SNGT_QHENOMENOLOGY_15: (0)             SCALE_FACTOR_PER_FONT_POINT = 0.001
SNGT_QHENOMENOLOGY_16: (0)             class Tex(StringMobject):
SNGT_QHENOMENOLOGY_17: (4)                 tex_environment: str = "align*"
SNGT_QHENOMENOLOGY_18: (4)                 def __init__(
SNGT_QHENOMENOLOGY_19: (8)                     self,
SNGT_QHENOMENOLOGY_20: (8)                     *tex_strings: str,
SNGT_QHENOMENOLOGY_21: (8)                     font_size: int = 48,
SNGT_QHENOMENOLOGY_22: (8)                     alignment: str = R"\centering",
SNGT_QHENOMENOLOGY_23: (8)                     template: str = "",
SNGT_QHENOMENOLOGY_24: (8)                     additional_preamble: str = "",
SNGT_QHENOMENOLOGY_25: (8)                     tex_to_color_map: dict = dict(),
SNGT_QHENOMENOLOGY_26: (8)                     t2c: dict = dict(),
SNGT_QHENOMENOLOGY_27: (8)                     isolate: Selector = [],
SNGT_QHENOMENOLOGY_28: (8)                     use_labelled_svg: bool = True,
SNGT_QHENOMENOLOGY_29: (8)                     **kwargs
SNGT_QHENOMENOLOGY_30: (4)                 ):
SNGT_QHENOMENOLOGY_31: (8)                     if len(tex_strings) > 1:
SNGT_QHENOMENOLOGY_32: (12)                        if isinstance(isolate, (str, re.Pattern,
tuple)):
SNGT_QHENOMENOLOGY_33: (16)                            isolate = [isolate]
SNGT_QHENOMENOLOGY_34: (12)                        isolate = [*isolate, *tex_strings]
SNGT_QHENOMENOLOGY_35: (8)                     tex_string = (" ".join(tex_strings)).strip()
SNGT_QHENOMENOLOGY_36: (8)                     if not tex_string.strip():
SNGT_QHENOMENOLOGY_37: (12)                        tex_string = R"\\"
SNGT_QHENOMENOLOGY_38: (8)                     self.font_size = font_size
SNGT_QHENOMENOLOGY_39: (8)                     self.tex_string = tex_string
SNGT_QHENOMENOLOGY_40: (8)                     self.alignment = alignment
SNGT_QHENOMENOLOGY_41: (8)                     self.template = template
SNGT_QHENOMENOLOGY_42: (8)                     self.additional_preamble = additional_preamble
SNGT_QHENOMENOLOGY_43: (8)                     self.tex_to_color_map = dict(**t2c,
**tex_to_color_map)
SNGT_QHENOMENOLOGY_44: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_45: (12)                        tex_string,
SNGT_QHENOMENOLOGY_46: (12)                        use_labelled_svg=use_labelled_svg,
SNGT_QHENOMENOLOGY_47: (12)                        isolate=isolate,
SNGT_QHENOMENOLOGY_48: (12)                        **kwargs
SNGT_QHENOMENOLOGY_49: (8)                     )
SNGT_QHENOMENOLOGY_50: (8)
self.set_color_by_tex_to_color_map(self.tex_to_color_map)
SNGT_QHENOMENOLOGY_51: (8)                     self.scale(SCALE_FACTOR_PER_FONT_POINT * font_size)
SNGT_QHENOMENOLOGY_52: (4)                 def get_svg_string_by_content(self, content: str) ->
str:
SNGT_QHENOMENOLOGY_53: (8)                     return latex_to_svg(content, self.template,
self.additional_preamble, short_tex=self.tex_string)
SNGT_QHENOMENOLOGY_54: (4)                 def _handle_scale_side_effects(self, scale_factor:
float) -> Self:
SNGT_QHENOMENOLOGY_55: (8)                     self.font_size *= scale_factor
SNGT_QHENOMENOLOGY_56: (8)                     return self
SNGT_QHENOMENOLOGY_57: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_58: (4)                 def get_command_matches(string: str) -> list[re.Match]:
```

```
SNGT_QHENOMENOLOGY_59: (8)                    pattern = re.compile(r"""
SNGT_QHENOMENOLOGY_60: (12)                       (?P<command>\\(?:[a-zA-Z]+|.))
SNGT_QHENOMENOLOGY_61: (12)                       |(?P<open>{+)
SNGT_QHENOMENOLOGY_62: (12)                       |(?P<close>}+)
SNGT_QHENOMENOLOGY_63: (8)                    """, flags=re.X | re.S)
SNGT_QHENOMENOLOGY_64: (8)                    result = []
SNGT_QHENOMENOLOGY_65: (8)                    open_stack = []
SNGT_QHENOMENOLOGY_66: (8)                    for match_obj in pattern.finditer(string):
SNGT_QHENOMENOLOGY_67: (12)                       if match_obj.group("open"):
SNGT_QHENOMENOLOGY_68: (16)                           open_stack.append((match_obj.span(),
len(result)))
SNGT_QHENOMENOLOGY_69: (12)                       elif match_obj.group("close"):
SNGT_QHENOMENOLOGY_70: (16)                           close_start, close_end = match_obj.span()
SNGT_QHENOMENOLOGY_71: (16)                           while True:
SNGT_QHENOMENOLOGY_72: (20)                               if not open_stack:
SNGT_QHENOMENOLOGY_73: (24)                                   raise ValueError("Missing '{'
inserted")
SNGT_QHENOMENOLOGY_74: (20)                               (open_start, open_end), index =
open_stack.pop()
SNGT_QHENOMENOLOGY_75: (20)                               n = min(open_end - open_start,
close_end - close_start)
SNGT_QHENOMENOLOGY_76: (20)                               result.insert(index, pattern.fullmatch(
SNGT_QHENOMENOLOGY_77: (24)                                   string, pos=open_end - n,
endpos=open_end
SNGT_QHENOMENOLOGY_78: (20)                               ))
SNGT_QHENOMENOLOGY_79: (20)                               result.append(pattern.fullmatch(
SNGT_QHENOMENOLOGY_80: (24)                                   string, pos=close_start,
endpos=close_start + n
SNGT_QHENOMENOLOGY_81: (20)                               ))
SNGT_QHENOMENOLOGY_82: (20)                               close_start += n
SNGT_QHENOMENOLOGY_83: (20)                               if close_start < close_end:
SNGT_QHENOMENOLOGY_84: (24)                                   continue
SNGT_QHENOMENOLOGY_85: (20)                               open_end -= n
SNGT_QHENOMENOLOGY_86: (20)                               if open_start < open_end:
SNGT_QHENOMENOLOGY_87: (24)                                   open_stack.append(((open_start,
open_end), index))
SNGT_QHENOMENOLOGY_88: (20)                               break
SNGT_QHENOMENOLOGY_89: (12)                       else:
SNGT_QHENOMENOLOGY_90: (16)                           result.append(match_obj)
SNGT_QHENOMENOLOGY_91: (8)                    if open_stack:
SNGT_QHENOMENOLOGY_92: (12)                       raise ValueError("Missing '}' inserted")
SNGT_QHENOMENOLOGY_93: (8)                    return result
SNGT_QHENOMENOLOGY_94: (4)                @staticmethod
SNGT_QHENOMENOLOGY_95: (4)                def get_command_flag(match_obj: re.Match) -> int:
SNGT_QHENOMENOLOGY_96: (8)                    if match_obj.group("open"):
SNGT_QHENOMENOLOGY_97: (12)                       return 1
SNGT_QHENOMENOLOGY_98: (8)                    if match_obj.group("close"):
SNGT_QHENOMENOLOGY_99: (12)                       return -1
SNGT_QHENOMENOLOGY_100: (8)                   return 0
SNGT_QHENOMENOLOGY_101: (4)               @staticmethod
SNGT_QHENOMENOLOGY_102: (4)               def replace_for_content(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_103: (8)                   return match_obj.group()
SNGT_QHENOMENOLOGY_104: (4)               @staticmethod
SNGT_QHENOMENOLOGY_105: (4)               def replace_for_matching(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_106: (8)                   if match_obj.group("command"):
SNGT_QHENOMENOLOGY_107: (12)                      return match_obj.group()
SNGT_QHENOMENOLOGY_108: (8)                   return ""
SNGT_QHENOMENOLOGY_109: (4)               @staticmethod
SNGT_QHENOMENOLOGY_110: (4)               def get_attr_dict_from_command_pair(
SNGT_QHENOMENOLOGY_111: (8)                   open_command: re.Match, close_command: re.Match
SNGT_QHENOMENOLOGY_112: (4)               ) -> dict[str, str] | None:
SNGT_QHENOMENOLOGY_113: (8)                   if len(open_command.group()) >= 2:
SNGT_QHENOMENOLOGY_114: (12)                      return {}
SNGT_QHENOMENOLOGY_115: (8)                   return None
SNGT_QHENOMENOLOGY_116: (4)               def get_configured_items(self) -> list[tuple[Span,
dict[str, str]]]:
SNGT_QHENOMENOLOGY_117: (8)                   return [
SNGT_QHENOMENOLOGY_118: (12)                      (span, {})
SNGT_QHENOMENOLOGY_119: (12)                      for selector in self.tex_to_color_map
```

```
SNGT_QHENOMENOLOGY_120: (12)                        for span in
self.find_spans_by_selector(selector)
SNGT_QHENOMENOLOGY_121: (8)                    ]
SNGT_QHENOMENOLOGY_122: (4)                @staticmethod
SNGT_QHENOMENOLOGY_123: (4)                def get_color_command(rgb_hex: str) -> str:
SNGT_QHENOMENOLOGY_124: (8)                    rgb = hex_to_int(rgb_hex)
SNGT_QHENOMENOLOGY_125: (8)                    rg, b = divmod(rgb, 256)
SNGT_QHENOMENOLOGY_126: (8)                    r, g = divmod(rg, 256)
SNGT_QHENOMENOLOGY_127: (8)                    return f"\\color[RGB]{{{r}, {g}, {b}}}"
SNGT_QHENOMENOLOGY_128: (4)                @staticmethod
SNGT_QHENOMENOLOGY_129: (4)                def get_command_string(
SNGT_QHENOMENOLOGY_130: (8)                    attr_dict: dict[str, str], is_end: bool, label_hex:
str | None
SNGT_QHENOMENOLOGY_131: (4)                ) -> str:
SNGT_QHENOMENOLOGY_132: (8)                    if label_hex is None:
SNGT_QHENOMENOLOGY_133: (12)                       return ""
SNGT_QHENOMENOLOGY_134: (8)                    if is_end:
SNGT_QHENOMENOLOGY_135: (12)                       return "}}"
SNGT_QHENOMENOLOGY_136: (8)                    return "{{" + Tex.get_color_command(label_hex)
SNGT_QHENOMENOLOGY_137: (4)                def get_content_prefix_and_suffix(
SNGT_QHENOMENOLOGY_138: (8)                    self, is_labelled: bool
SNGT_QHENOMENOLOGY_139: (4)                ) -> tuple[str, str]:
SNGT_QHENOMENOLOGY_140: (8)                    prefix_lines = []
SNGT_QHENOMENOLOGY_141: (8)                    suffix_lines = []
SNGT_QHENOMENOLOGY_142: (8)                    if not is_labelled:
SNGT_QHENOMENOLOGY_143: (12)                       prefix_lines.append(self.get_color_command(
SNGT_QHENOMENOLOGY_144: (16)                           color_to_hex(self.base_color)
SNGT_QHENOMENOLOGY_145: (12)                       ))
SNGT_QHENOMENOLOGY_146: (8)                    if self.alignment:
SNGT_QHENOMENOLOGY_147: (12)                       prefix_lines.append(self.alignment)
SNGT_QHENOMENOLOGY_148: (8)                    if self.tex_environment:
SNGT_QHENOMENOLOGY_149: (12)
prefix_lines.append(f"\\begin{{{self.tex_environment}}}")
SNGT_QHENOMENOLOGY_150: (12)
suffix_lines.append(f"\\end{{{self.tex_environment}}}")
SNGT_QHENOMENOLOGY_151: (8)                    return (
SNGT_QHENOMENOLOGY_152: (12)                       "".join([line + "\n" for line in
prefix_lines]),
SNGT_QHENOMENOLOGY_153: (12)                       "".join(["\n" + line for line in suffix_lines])
SNGT_QHENOMENOLOGY_154: (8)                    )
SNGT_QHENOMENOLOGY_155: (4)                def get_parts_by_tex(self, selector: Selector) ->
VGroup:
SNGT_QHENOMENOLOGY_156: (8)                    return self.select_parts(selector)
SNGT_QHENOMENOLOGY_157: (4)                def get_part_by_tex(self, selector: Selector, index:
int = 0) -> VMobject:
SNGT_QHENOMENOLOGY_158: (8)                    return self.select_part(selector, index)
SNGT_QHENOMENOLOGY_159: (4)                def set_color_by_tex(self, selector: Selector, color:
ManimColor):
SNGT_QHENOMENOLOGY_160: (8)                    return self.set_parts_color(selector, color)
SNGT_QHENOMENOLOGY_161: (4)                def set_color_by_tex_to_color_map(
SNGT_QHENOMENOLOGY_162: (8)                    self, color_map: dict[Selector, ManimColor]
SNGT_QHENOMENOLOGY_163: (4)                ):
SNGT_QHENOMENOLOGY_164: (8)                    return self.set_parts_color_by_dict(color_map)
SNGT_QHENOMENOLOGY_165: (4)                def get_tex(self) -> str:
SNGT_QHENOMENOLOGY_166: (8)                    return self.get_string()
SNGT_QHENOMENOLOGY_167: (4)                def substr_to_path_count(self, substr: str) -> int:
SNGT_QHENOMENOLOGY_168: (8)                    tex = self.get_tex()
SNGT_QHENOMENOLOGY_169: (8)                    if len(self) != num_tex_symbols(tex):
SNGT_QHENOMENOLOGY_170: (12)                       log.warning(f"Estimated size of {tex} does not
match true size")
SNGT_QHENOMENOLOGY_171: (8)                    return num_tex_symbols(substr)
SNGT_QHENOMENOLOGY_172: (4)                def get_symbol_substrings(self):
SNGT_QHENOMENOLOGY_173: (8)                    pattern = "|".join((
SNGT_QHENOMENOLOGY_174: (12)                       r"\\[a-zA-Z]+",
SNGT_QHENOMENOLOGY_175: (12)                       r"[^\^\{\}\s\_\$\\\&]",
SNGT_QHENOMENOLOGY_176: (8)                    ))
SNGT_QHENOMENOLOGY_177: (8)                    return re.findall(pattern, self.string)
SNGT_QHENOMENOLOGY_178: (4)                def make_number_changeable(
SNGT_QHENOMENOLOGY_179: (8)                    self,
```

```
SNGT_QHENOMENOLOGY_180: (8)                     value: float | int | str,
SNGT_QHENOMENOLOGY_181: (8)                     index: int = 0,
SNGT_QHENOMENOLOGY_182: (8)                     replace_all: bool = False,
SNGT_QHENOMENOLOGY_183: (8)                     **config,
SNGT_QHENOMENOLOGY_184: (4)                 ) -> VMobject:
SNGT_QHENOMENOLOGY_185: (8)                     substr = str(value)
SNGT_QHENOMENOLOGY_186: (8)                     parts = self.select_parts(substr)
SNGT_QHENOMENOLOGY_187: (8)                     if len(parts) == 0:
SNGT_QHENOMENOLOGY_188: (12)                        log.warning(f"{value} not found in
Tex.make_number_changeable call")
SNGT_QHENOMENOLOGY_189: (12)                        return VMobject()
SNGT_QHENOMENOLOGY_190: (8)                     if index > len(parts) - 1:
SNGT_QHENOMENOLOGY_191: (12)                        log.warning(f"Requested {index}th occurance of
{value}, but only {len(parts)} exist")
SNGT_QHENOMENOLOGY_192: (12)                        return VMobject()
SNGT_QHENOMENOLOGY_193: (8)                     if not replace_all:
SNGT_QHENOMENOLOGY_194: (12)                        parts = [parts[index]]
SNGT_QHENOMENOLOGY_195: (8)                     from manimlib.mobject.numbers import DecimalNumber
SNGT_QHENOMENOLOGY_196: (8)                     decimal_mobs = []
SNGT_QHENOMENOLOGY_197: (8)                     for part in parts:
SNGT_QHENOMENOLOGY_198: (12)                        if "." in substr:
SNGT_QHENOMENOLOGY_199: (16)                            num_decimal_places = len(substr.split(".")
[1])
SNGT_QHENOMENOLOGY_200: (12)                        else:
SNGT_QHENOMENOLOGY_201: (16)                            num_decimal_places = 0
SNGT_QHENOMENOLOGY_202: (12)                        decimal_mob = DecimalNumber(
SNGT_QHENOMENOLOGY_203: (16)                            float(value),
SNGT_QHENOMENOLOGY_204: (16)                            num_decimal_places=num_decimal_places,
SNGT_QHENOMENOLOGY_205: (16)                            **config,
SNGT_QHENOMENOLOGY_206: (12)                        )
SNGT_QHENOMENOLOGY_207: (12)                        decimal_mob.replace(part)
SNGT_QHENOMENOLOGY_208: (12)                        decimal_mob.match_style(part)
SNGT_QHENOMENOLOGY_209: (12)                        if len(part) > 1:
SNGT_QHENOMENOLOGY_210: (16)                            self.remove(*part[1:])
SNGT_QHENOMENOLOGY_211: (12)
self.replace_submobject(self.submobjects.index(part[0]), decimal_mob)
SNGT_QHENOMENOLOGY_212: (12)                        decimal_mobs.append(decimal_mob)
SNGT_QHENOMENOLOGY_213: (12)                        self.string = self.string.replace(substr,
R"\decimalmob", 1)
SNGT_QHENOMENOLOGY_214: (8)                     if replace_all:
SNGT_QHENOMENOLOGY_215: (12)                        return VGroup(*decimal_mobs)
SNGT_QHENOMENOLOGY_216: (8)                     return decimal_mobs[index]
SNGT_QHENOMENOLOGY_217: (0)             class TexText(Tex):
SNGT_QHENOMENOLOGY_218: (4)                 tex_environment: str = ""
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 56 - vector_field.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import itertools as it
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from scipy.integrate import solve_ivp
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import FRAME_HEIGHT, FRAME_WIDTH
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.animation.indication import VShowPassingFlash
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.utils.bezier import inverse_interpolate
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.color import get_colormap_list
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.color import get_color_map
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.iterables import cartesian_product
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.utils.rate_functions import linear
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_17: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_18: (0)              if TYPE_CHECKING:
```

```
SNGT_QHENOMENOLOGY_19: (4)                 from typing import Callable, Iterable, Sequence,
TypeVar, Tuple, Optional
SNGT_QHENOMENOLOGY_20: (4)                 from manimlib.typing import ManimColor, Vect3, VectN,
VectArray, Vect3Array, Vect4Array
SNGT_QHENOMENOLOGY_21: (4)                 from manimlib.mobject.coordinate_systems import
CoordinateSystem
SNGT_QHENOMENOLOGY_22: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_23: (4)                 T = TypeVar("T")
SNGT_QHENOMENOLOGY_24: (0)             def get_vectorized_rgb_gradient_function(
SNGT_QHENOMENOLOGY_25: (4)                 min_value: T,
SNGT_QHENOMENOLOGY_26: (4)                 max_value: T,
SNGT_QHENOMENOLOGY_27: (4)                 color_map: str
SNGT_QHENOMENOLOGY_28: (0)             ) -> Callable[[VectN], Vect3Array]:
SNGT_QHENOMENOLOGY_29: (4)                 rgbs = np.array(get_colormap_list(color_map))
SNGT_QHENOMENOLOGY_30: (4)                 def func(values):
SNGT_QHENOMENOLOGY_31: (8)                     alphas = inverse_interpolate(
SNGT_QHENOMENOLOGY_32: (12)                        min_value, max_value, np.array(values)
SNGT_QHENOMENOLOGY_33: (8)                     )
SNGT_QHENOMENOLOGY_34: (8)                     alphas = np.clip(alphas, 0, 1)
SNGT_QHENOMENOLOGY_35: (8)                     scaled_alphas = alphas * (len(rgbs) - 1)
SNGT_QHENOMENOLOGY_36: (8)                     indices = scaled_alphas.astype(int)
SNGT_QHENOMENOLOGY_37: (8)                     next_indices = np.clip(indices + 1, 0, len(rgbs) -
1)
SNGT_QHENOMENOLOGY_38: (8)                     inter_alphas = scaled_alphas % 1
SNGT_QHENOMENOLOGY_39: (8)                     inter_alphas =
inter_alphas.repeat(3).reshape((len(indices), 3))
SNGT_QHENOMENOLOGY_40: (8)                     result = interpolate(rgbs[indices],
rgbs[next_indices], inter_alphas)
SNGT_QHENOMENOLOGY_41: (8)                     return result
SNGT_QHENOMENOLOGY_42: (4)                 return func
SNGT_QHENOMENOLOGY_43: (0)             def get_rgb_gradient_function(
SNGT_QHENOMENOLOGY_44: (4)                 min_value: T,
SNGT_QHENOMENOLOGY_45: (4)                 max_value: T,
SNGT_QHENOMENOLOGY_46: (4)                 color_map: str
SNGT_QHENOMENOLOGY_47: (0)             ) -> Callable[[float], Vect3]:
SNGT_QHENOMENOLOGY_48: (4)                 vectorized_func =
get_vectorized_rgb_gradient_function(min_value, max_value, color_map)
SNGT_QHENOMENOLOGY_49: (4)                 return lambda value: vectorized_func(np.array([value]))
[0]
SNGT_QHENOMENOLOGY_50: (0)             def ode_solution_points(function, state0, time, dt=0.01):
SNGT_QHENOMENOLOGY_51: (4)                 solution = solve_ivp(
SNGT_QHENOMENOLOGY_52: (8)                     lambda t, state: function(state),
SNGT_QHENOMENOLOGY_53: (8)                     t_span=(0, time),
SNGT_QHENOMENOLOGY_54: (8)                     y0=state0,
SNGT_QHENOMENOLOGY_55: (8)                     t_eval=np.arange(0, time, dt)
SNGT_QHENOMENOLOGY_56: (4)                 )
SNGT_QHENOMENOLOGY_57: (4)                 return solution.y.T
SNGT_QHENOMENOLOGY_58: (0)             def move_along_vector_field(
SNGT_QHENOMENOLOGY_59: (4)                 mobject: Mobject,
SNGT_QHENOMENOLOGY_60: (4)                 func: Callable[[Vect3], Vect3]
SNGT_QHENOMENOLOGY_61: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_62: (4)                 mobject.add_updater(
SNGT_QHENOMENOLOGY_63: (8)                     lambda m, dt: m.shift(
SNGT_QHENOMENOLOGY_64: (12)                        func(m.get_center()) * dt
SNGT_QHENOMENOLOGY_65: (8)                     )
SNGT_QHENOMENOLOGY_66: (4)                 )
SNGT_QHENOMENOLOGY_67: (4)                 return mobject
SNGT_QHENOMENOLOGY_68: (0)             def move_submobjects_along_vector_field(
SNGT_QHENOMENOLOGY_69: (4)                 mobject: Mobject,
SNGT_QHENOMENOLOGY_70: (4)                 func: Callable[[Vect3], Vect3]
SNGT_QHENOMENOLOGY_71: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_72: (4)                 def apply_nudge(mob, dt):
SNGT_QHENOMENOLOGY_73: (8)                     for submob in mob:
SNGT_QHENOMENOLOGY_74: (12)                        x, y = submob.get_center()[:2]
SNGT_QHENOMENOLOGY_75: (12)                        if abs(x) < FRAME_WIDTH and abs(y) <
FRAME_HEIGHT:
SNGT_QHENOMENOLOGY_76: (16)                            submob.shift(func(submob.get_center()) *
dt)
SNGT_QHENOMENOLOGY_77: (4)                 mobject.add_updater(apply_nudge)
```

```
SNGT_QHENOMENOLOGY_78: (4)                 return mobject
SNGT_QHENOMENOLOGY_79: (0)             def move_points_along_vector_field(
SNGT_QHENOMENOLOGY_80: (4)                 mobject: Mobject,
SNGT_QHENOMENOLOGY_81: (4)                 func: Callable[[float, float], Iterable[float]],
SNGT_QHENOMENOLOGY_82: (4)                 coordinate_system: CoordinateSystem
SNGT_QHENOMENOLOGY_83: (0)             ) -> Mobject:
SNGT_QHENOMENOLOGY_84: (4)                 cs = coordinate_system
SNGT_QHENOMENOLOGY_85: (4)                 origin = cs.get_origin()
SNGT_QHENOMENOLOGY_86: (4)                 def apply_nudge(mob, dt):
SNGT_QHENOMENOLOGY_87: (8)                     mob.apply_function(
SNGT_QHENOMENOLOGY_88: (12)                        lambda p: p + (cs.c2p(*func(*cs.p2c(p))) -
origin) * dt
SNGT_QHENOMENOLOGY_89: (8)                     )
SNGT_QHENOMENOLOGY_90: (4)                 mobject.add_updater(apply_nudge)
SNGT_QHENOMENOLOGY_91: (4)                 return mobject
SNGT_QHENOMENOLOGY_92: (0)             def get_sample_coords(
SNGT_QHENOMENOLOGY_93: (4)                 coordinate_system: CoordinateSystem,
SNGT_QHENOMENOLOGY_94: (4)                 density: float = 1.0
SNGT_QHENOMENOLOGY_95: (0)             ) -> it.product[tuple[Vect3, ...]]:
SNGT_QHENOMENOLOGY_96: (4)                 ranges = []
SNGT_QHENOMENOLOGY_97: (4)                 for range_args in coordinate_system.get_all_ranges():
SNGT_QHENOMENOLOGY_98: (8)                     _min, _max, step = range_args
SNGT_QHENOMENOLOGY_99: (8)                     step /= density
SNGT_QHENOMENOLOGY_100: (8)                    ranges.append(np.arange(_min, _max + step, step))
SNGT_QHENOMENOLOGY_101: (4)                return np.array(list(it.product(*ranges)))
SNGT_QHENOMENOLOGY_102: (0)            def vectorize(pointwise_function: Callable[[Tuple],
Tuple]):
SNGT_QHENOMENOLOGY_103: (4)                def v_func(coords_array: VectArray) -> VectArray:
SNGT_QHENOMENOLOGY_104: (8)                    return np.array([pointwise_function(*coords) for
coords in coords_array])
SNGT_QHENOMENOLOGY_105: (4)                return v_func
SNGT_QHENOMENOLOGY_106: (0)            class VectorField(VMobject):
SNGT_QHENOMENOLOGY_107: (4)                def __init__(
SNGT_QHENOMENOLOGY_108: (8)                    self,
SNGT_QHENOMENOLOGY_109: (8)                    func: Callable[[VectArray], VectArray],
SNGT_QHENOMENOLOGY_110: (8)                    coordinate_system: CoordinateSystem,
SNGT_QHENOMENOLOGY_111: (8)                    density: float = 2.0,
SNGT_QHENOMENOLOGY_112: (8)                    magnitude_range: Optional[Tuple[float, float]] =
None,
SNGT_QHENOMENOLOGY_113: (8)                    color: Optional[ManimColor] = None,
SNGT_QHENOMENOLOGY_114: (8)                    color_map_name: Optional[str] = "3b1b_colormap",
SNGT_QHENOMENOLOGY_115: (8)                    color_map: Optional[Callable[[Sequence[float]],
Vect4Array]] = None,
SNGT_QHENOMENOLOGY_116: (8)                    stroke_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_117: (8)                    stroke_width: float = 3,
SNGT_QHENOMENOLOGY_118: (8)                    tip_width_ratio: float = 4,
SNGT_QHENOMENOLOGY_119: (8)                    tip_len_to_width: float = 0.01,
SNGT_QHENOMENOLOGY_120: (8)                    max_vect_len: float | None = None,
SNGT_QHENOMENOLOGY_121: (8)                    max_vect_len_to_step_size: float = 0.8,
SNGT_QHENOMENOLOGY_122: (8)                    flat_stroke: bool = False,
SNGT_QHENOMENOLOGY_123: (8)                    norm_to_opacity_func=None,  # TODO, check on this
SNGT_QHENOMENOLOGY_124: (8)                    **kwargs
SNGT_QHENOMENOLOGY_125: (4)                ):
SNGT_QHENOMENOLOGY_126: (8)                    self.func = func
SNGT_QHENOMENOLOGY_127: (8)                    self.coordinate_system = coordinate_system
SNGT_QHENOMENOLOGY_128: (8)                    self.stroke_width = stroke_width
SNGT_QHENOMENOLOGY_129: (8)                    self.tip_width_ratio = tip_width_ratio
SNGT_QHENOMENOLOGY_130: (8)                    self.tip_len_to_width = tip_len_to_width
SNGT_QHENOMENOLOGY_131: (8)                    self.norm_to_opacity_func = norm_to_opacity_func
SNGT_QHENOMENOLOGY_132: (8)                    self.sample_coords =
get_sample_coords(coordinate_system, density)
SNGT_QHENOMENOLOGY_133: (8)                    self.update_sample_points()
SNGT_QHENOMENOLOGY_134: (8)                    if max_vect_len is None:
SNGT_QHENOMENOLOGY_135: (12)                       step_size = get_norm(self.sample_points[1] -
self.sample_points[0])
SNGT_QHENOMENOLOGY_136: (12)                       self.max_displayed_vect_len =
max_vect_len_to_step_size * step_size
SNGT_QHENOMENOLOGY_137: (8)                    else:
SNGT_QHENOMENOLOGY_138: (12)                       self.max_displayed_vect_len = max_vect_len *
```

```
coordinate_system.get_x_unit_size()
SNGT_QHENOMENOLOGY_139: (8)                     if magnitude_range is None:
SNGT_QHENOMENOLOGY_140: (12)                        max_value = max(map(get_norm,
func(self.sample_coords)))
SNGT_QHENOMENOLOGY_141: (12)                        magnitude_range = (0, max_value)
SNGT_QHENOMENOLOGY_142: (8)                     self.magnitude_range = magnitude_range
SNGT_QHENOMENOLOGY_143: (8)                     if color is not None:
SNGT_QHENOMENOLOGY_144: (12)                        self.color_map = None
SNGT_QHENOMENOLOGY_145: (8)                     else:
SNGT_QHENOMENOLOGY_146: (12)                        self.color_map = color_map or
get_color_map(color_map_name)
SNGT_QHENOMENOLOGY_147: (8)
self.init_base_stroke_width_array(len(self.sample_coords))
SNGT_QHENOMENOLOGY_148: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_149: (12)                        stroke_opacity=stroke_opacity,
SNGT_QHENOMENOLOGY_150: (12)                        flat_stroke=flat_stroke,
SNGT_QHENOMENOLOGY_151: (12)                        **kwargs
SNGT_QHENOMENOLOGY_152: (8)                     )
SNGT_QHENOMENOLOGY_153: (8)                     self.set_stroke(color, stroke_width)
SNGT_QHENOMENOLOGY_154: (8)                     self.update_vectors()
SNGT_QHENOMENOLOGY_155: (4)                 def init_points(self):
SNGT_QHENOMENOLOGY_156: (8)                     n_samples = len(self.sample_coords)
SNGT_QHENOMENOLOGY_157: (8)                     self.set_points(np.zeros((8 * n_samples - 1, 3)))
SNGT_QHENOMENOLOGY_158: (8)                     self.set_joint_type('no_joint')
SNGT_QHENOMENOLOGY_159: (4)                 def get_sample_points(
SNGT_QHENOMENOLOGY_160: (8)                     self,
SNGT_QHENOMENOLOGY_161: (8)                     center: np.ndarray,
SNGT_QHENOMENOLOGY_162: (8)                     width: float,
SNGT_QHENOMENOLOGY_163: (8)                     height: float,
SNGT_QHENOMENOLOGY_164: (8)                     depth: float,
SNGT_QHENOMENOLOGY_165: (8)                     x_density: float,
SNGT_QHENOMENOLOGY_166: (8)                     y_density: float,
SNGT_QHENOMENOLOGY_167: (8)                     z_density: float
SNGT_QHENOMENOLOGY_168: (4)                 ) -> np.ndarray:
SNGT_QHENOMENOLOGY_169: (8)                     to_corner = np.array([width / 2, height / 2, depth
/ 2])
SNGT_QHENOMENOLOGY_170: (8)                     spacings = 1.0 / np.array([x_density, y_density,
z_density])
SNGT_QHENOMENOLOGY_171: (8)                     to_corner = spacings * (to_corner /
spacings).astype(int)
SNGT_QHENOMENOLOGY_172: (8)                     lower_corner = center - to_corner
SNGT_QHENOMENOLOGY_173: (8)                     upper_corner = center + to_corner + spacings
SNGT_QHENOMENOLOGY_174: (8)                     return cartesian_product(*(
SNGT_QHENOMENOLOGY_175: (12)                        np.arange(low, high, space)
SNGT_QHENOMENOLOGY_176: (12)                        for low, high, space in zip(lower_corner,
upper_corner, spacings)
SNGT_QHENOMENOLOGY_177: (8)                     ))
SNGT_QHENOMENOLOGY_178: (4)                 def init_base_stroke_width_array(self,
n_sample_points):
SNGT_QHENOMENOLOGY_179: (8)                     arr = np.ones(8 * n_sample_points - 1)
SNGT_QHENOMENOLOGY_180: (8)                     arr[4::8] = self.tip_width_ratio
SNGT_QHENOMENOLOGY_181: (8)                     arr[5::8] = self.tip_width_ratio * 0.5
SNGT_QHENOMENOLOGY_182: (8)                     arr[6::8] = 0
SNGT_QHENOMENOLOGY_183: (8)                     arr[7::8] = 0
SNGT_QHENOMENOLOGY_184: (8)                     self.base_stroke_width_array = arr
SNGT_QHENOMENOLOGY_185: (4)                 def set_sample_coords(self, sample_coords: VectArray):
SNGT_QHENOMENOLOGY_186: (8)                     self.sample_coords = sample_coords
SNGT_QHENOMENOLOGY_187: (8)                     return self
SNGT_QHENOMENOLOGY_188: (4)                 def set_stroke(self, color=None, width=None,
opacity=None, behind=None, flat=None, recurse=True):
SNGT_QHENOMENOLOGY_189: (8)                     super().set_stroke(color, None, opacity, behind,
flat, recurse)
SNGT_QHENOMENOLOGY_190: (8)                     if width is not None:
SNGT_QHENOMENOLOGY_191: (12)                        self.set_stroke_width(float(width))
SNGT_QHENOMENOLOGY_192: (8)                     return self
SNGT_QHENOMENOLOGY_193: (4)                 def set_stroke_width(self, width: float):
SNGT_QHENOMENOLOGY_194: (8)                     if self.get_num_points() > 0:
SNGT_QHENOMENOLOGY_195: (12)                        self.get_stroke_widths()[:] = width *
self.base_stroke_width_array
```

```
SNGT_QHENOMENOLOGY_196: (12)                     self.stroke_width = width
SNGT_QHENOMENOLOGY_197: (8)                 return self
SNGT_QHENOMENOLOGY_198: (4)             def update_sample_points(self):
SNGT_QHENOMENOLOGY_199: (8)                 self.sample_points =
self.coordinate_system.c2p(*self.sample_coords.T)
SNGT_QHENOMENOLOGY_200: (4)             def update_vectors(self):
SNGT_QHENOMENOLOGY_201: (8)                 tip_width = self.tip_width_ratio *
self.stroke_width
SNGT_QHENOMENOLOGY_202: (8)                 tip_len = self.tip_len_to_width * tip_width
SNGT_QHENOMENOLOGY_203: (8)                 outputs = self.func(self.sample_coords)
SNGT_QHENOMENOLOGY_204: (8)                 output_norms = np.linalg.norm(outputs, axis=1)[:,
np.newaxis]
SNGT_QHENOMENOLOGY_205: (8)                 out_vects = self.coordinate_system.c2p(*outputs.T)
- self.coordinate_system.get_origin()
SNGT_QHENOMENOLOGY_206: (8)                 out_vect_norms = np.linalg.norm(out_vects, axis=1)
[:, np.newaxis]
SNGT_QHENOMENOLOGY_207: (8)                 unit_outputs = np.zeros_like(out_vects)
SNGT_QHENOMENOLOGY_208: (8)                 np.true_divide(out_vects, out_vect_norms,
out=unit_outputs, where=(out_vect_norms > 0))
SNGT_QHENOMENOLOGY_209: (8)                 max_len = self.max_displayed_vect_len
SNGT_QHENOMENOLOGY_210: (8)                 if max_len < np.inf:
SNGT_QHENOMENOLOGY_211: (12)                     drawn_norms = max_len * np.tanh(out_vect_norms
/ max_len)
SNGT_QHENOMENOLOGY_212: (8)                 else:
SNGT_QHENOMENOLOGY_213: (12)                     drawn_norms = out_vect_norms
SNGT_QHENOMENOLOGY_214: (8)                 dist_to_head_base = np.clip(drawn_norms - tip_len,
0, np.inf)  # Mixing units!
SNGT_QHENOMENOLOGY_215: (8)                 points = self.get_points()
SNGT_QHENOMENOLOGY_216: (8)                 points[0::8] = self.sample_points
SNGT_QHENOMENOLOGY_217: (8)                 points[2::8] = self.sample_points +
dist_to_head_base * unit_outputs
SNGT_QHENOMENOLOGY_218: (8)                 points[4::8] = points[2::8]
SNGT_QHENOMENOLOGY_219: (8)                 points[6::8] = self.sample_points + drawn_norms *
unit_outputs
SNGT_QHENOMENOLOGY_220: (8)                 for i in (1, 3, 5):
SNGT_QHENOMENOLOGY_221: (12)                     points[i::8] = 0.5 * (points[i - 1::8] +
points[i + 1::8])
SNGT_QHENOMENOLOGY_222: (8)                 points[7::8] = points[6:-1:8]
SNGT_QHENOMENOLOGY_223: (8)                 width_arr = self.stroke_width *
self.base_stroke_width_array
SNGT_QHENOMENOLOGY_224: (8)                 width_scalars = np.clip(drawn_norms / tip_len, 0,
1)
SNGT_QHENOMENOLOGY_225: (8)                 width_scalars = np.repeat(width_scalars, 8)[:-1]
SNGT_QHENOMENOLOGY_226: (8)                 self.get_stroke_widths()[:] = width_scalars *
width_arr
SNGT_QHENOMENOLOGY_227: (8)                 if self.color_map is not None:
SNGT_QHENOMENOLOGY_228: (12)                     self.get_stroke_colors()  # Ensures the array
is updated to appropriate length
SNGT_QHENOMENOLOGY_229: (12)                     low, high = self.magnitude_range
SNGT_QHENOMENOLOGY_230: (12)                     self.data['stroke_rgba'][:, :3] =
self.color_map(
SNGT_QHENOMENOLOGY_231: (16)                         inverse_interpolate(low, high,
np.repeat(output_norms, 8)[:-1])
SNGT_QHENOMENOLOGY_232: (12)                     )[:, :3]
SNGT_QHENOMENOLOGY_233: (8)                 if self.norm_to_opacity_func is not None:
SNGT_QHENOMENOLOGY_234: (12)                     self.get_stroke_opacities()[:] =
self.norm_to_opacity_func(
SNGT_QHENOMENOLOGY_235: (16)                         np.repeat(output_norms, 8)[:-1]
SNGT_QHENOMENOLOGY_236: (12)                     )
SNGT_QHENOMENOLOGY_237: (8)                 self.note_changed_data()
SNGT_QHENOMENOLOGY_238: (8)                 return self
SNGT_QHENOMENOLOGY_239: (0)         class TimeVaryingVectorField(VectorField):
SNGT_QHENOMENOLOGY_240: (4)             def __init__(
SNGT_QHENOMENOLOGY_241: (8)                 self,
SNGT_QHENOMENOLOGY_242: (8)                 time_func: Callable[[VectArray, float], VectArray],
SNGT_QHENOMENOLOGY_243: (8)                 coordinate_system: CoordinateSystem,
SNGT_QHENOMENOLOGY_244: (8)                 **kwargs
SNGT_QHENOMENOLOGY_245: (4)             ):
SNGT_QHENOMENOLOGY_246: (8)                 self.time = 0
```

```
SNGT_QHENOMENOLOGY_247: (8)                     def func(coords):
SNGT_QHENOMENOLOGY_248: (12)                         return time_func(coords, self.time)
SNGT_QHENOMENOLOGY_249: (8)                     super().__init__(func, coordinate_system, **kwargs)
SNGT_QHENOMENOLOGY_250: (8)                     self.add_updater(lambda m, dt:
m.increment_time(dt))
SNGT_QHENOMENOLOGY_251: (8)                     self.always.update_vectors()
SNGT_QHENOMENOLOGY_252: (4)                 def increment_time(self, dt):
SNGT_QHENOMENOLOGY_253: (8)                     self.time += dt
SNGT_QHENOMENOLOGY_254: (0)             class StreamLines(VGroup):
SNGT_QHENOMENOLOGY_255: (4)                 def __init__(
SNGT_QHENOMENOLOGY_256: (8)                     self,
SNGT_QHENOMENOLOGY_257: (8)                     func: Callable[[VectArray], VectArray],
SNGT_QHENOMENOLOGY_258: (8)                     coordinate_system: CoordinateSystem,
SNGT_QHENOMENOLOGY_259: (8)                     density: float = 1.0,
SNGT_QHENOMENOLOGY_260: (8)                     n_repeats: int = 1,
SNGT_QHENOMENOLOGY_261: (8)                     noise_factor: float | None = None,
SNGT_QHENOMENOLOGY_262: (8)                     solution_time: float = 3,
SNGT_QHENOMENOLOGY_263: (8)                     dt: float = 0.05,
SNGT_QHENOMENOLOGY_264: (8)                     arc_len: float = 3,
SNGT_QHENOMENOLOGY_265: (8)                     max_time_steps: int = 200,
SNGT_QHENOMENOLOGY_266: (8)                     n_samples_per_line: int = 10,
SNGT_QHENOMENOLOGY_267: (8)                     cutoff_norm: float = 15,
SNGT_QHENOMENOLOGY_268: (8)                     stroke_width: float = 1.0,
SNGT_QHENOMENOLOGY_269: (8)                     stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_270: (8)                     stroke_opacity: float = 1,
SNGT_QHENOMENOLOGY_271: (8)                     color_by_magnitude: bool = True,
SNGT_QHENOMENOLOGY_272: (8)                     magnitude_range: Tuple[float, float] = (0, 2.0),
SNGT_QHENOMENOLOGY_273: (8)                     taper_stroke_width: bool = False,
SNGT_QHENOMENOLOGY_274: (8)                     color_map: str = "3b1b_colormap",
SNGT_QHENOMENOLOGY_275: (8)                     **kwargs
SNGT_QHENOMENOLOGY_276: (4)                 ):
SNGT_QHENOMENOLOGY_277: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_278: (8)                     self.func = func
SNGT_QHENOMENOLOGY_279: (8)                     self.coordinate_system = coordinate_system
SNGT_QHENOMENOLOGY_280: (8)                     self.density = density
SNGT_QHENOMENOLOGY_281: (8)                     self.n_repeats = n_repeats
SNGT_QHENOMENOLOGY_282: (8)                     self.noise_factor = noise_factor
SNGT_QHENOMENOLOGY_283: (8)                     self.solution_time = solution_time
SNGT_QHENOMENOLOGY_284: (8)                     self.dt = dt
SNGT_QHENOMENOLOGY_285: (8)                     self.arc_len = arc_len
SNGT_QHENOMENOLOGY_286: (8)                     self.max_time_steps = max_time_steps
SNGT_QHENOMENOLOGY_287: (8)                     self.n_samples_per_line = n_samples_per_line
SNGT_QHENOMENOLOGY_288: (8)                     self.cutoff_norm = cutoff_norm
SNGT_QHENOMENOLOGY_289: (8)                     self.stroke_width = stroke_width
SNGT_QHENOMENOLOGY_290: (8)                     self.stroke_color = stroke_color
SNGT_QHENOMENOLOGY_291: (8)                     self.stroke_opacity = stroke_opacity
SNGT_QHENOMENOLOGY_292: (8)                     self.color_by_magnitude = color_by_magnitude
SNGT_QHENOMENOLOGY_293: (8)                     self.magnitude_range = magnitude_range
SNGT_QHENOMENOLOGY_294: (8)                     self.taper_stroke_width = taper_stroke_width
SNGT_QHENOMENOLOGY_295: (8)                     self.color_map = color_map
SNGT_QHENOMENOLOGY_296: (8)                     self.draw_lines()
SNGT_QHENOMENOLOGY_297: (8)                     self.init_style()
SNGT_QHENOMENOLOGY_298: (4)                 def point_func(self, points: Vect3Array) -> Vect3:
SNGT_QHENOMENOLOGY_299: (8)                     in_coords =
np.array(self.coordinate_system.p2c(points)).T
SNGT_QHENOMENOLOGY_300: (8)                     out_coords = self.func(in_coords)
SNGT_QHENOMENOLOGY_301: (8)                     origin = self.coordinate_system.get_origin()
SNGT_QHENOMENOLOGY_302: (8)                     return self.coordinate_system.c2p(*out_coords.T) -
origin
SNGT_QHENOMENOLOGY_303: (4)                 def draw_lines(self) -> None:
SNGT_QHENOMENOLOGY_304: (8)                     lines = []
SNGT_QHENOMENOLOGY_305: (8)                     origin = self.coordinate_system.get_origin()
SNGT_QHENOMENOLOGY_306: (8)                     lines = []
SNGT_QHENOMENOLOGY_307: (8)                     for coords in self.get_sample_coords():
SNGT_QHENOMENOLOGY_308: (12)                         solution_coords =
ode_solution_points(self.func, coords, self.solution_time, self.dt)
SNGT_QHENOMENOLOGY_309: (12)                         line = VMobject()
SNGT_QHENOMENOLOGY_310: (12)
line.set_points_smoothly(self.coordinate_system.c2p(*solution_coords.T))
```

```
SNGT_QHENOMENOLOGY_311: (12)                         line.virtual_time = self.solution_time
SNGT_QHENOMENOLOGY_312: (12)                         lines.append(line)
SNGT_QHENOMENOLOGY_313: (8)                     self.set_submobjects(lines)
SNGT_QHENOMENOLOGY_314: (4)                 def get_sample_coords(self):
SNGT_QHENOMENOLOGY_315: (8)                     cs = self.coordinate_system
SNGT_QHENOMENOLOGY_316: (8)                     sample_coords = get_sample_coords(cs, self.density)
SNGT_QHENOMENOLOGY_317: (8)                     noise_factor = self.noise_factor
SNGT_QHENOMENOLOGY_318: (8)                     if noise_factor is None:
SNGT_QHENOMENOLOGY_319: (12)                         noise_factor = (cs.get_x_unit_size() /
self.density) * 0.5
SNGT_QHENOMENOLOGY_320: (8)                     return np.array([
SNGT_QHENOMENOLOGY_321: (12)                         coords + noise_factor *
np.random.random(coords.shape)
SNGT_QHENOMENOLOGY_322: (12)                         for n in range(self.n_repeats)
SNGT_QHENOMENOLOGY_323: (12)                         for coords in sample_coords
SNGT_QHENOMENOLOGY_324: (8)                     ])
SNGT_QHENOMENOLOGY_325: (4)                 def init_style(self) -> None:
SNGT_QHENOMENOLOGY_326: (8)                     if self.color_by_magnitude:
SNGT_QHENOMENOLOGY_327: (12)                         values_to_rgbs =
get_vectorized_rgb_gradient_function(
SNGT_QHENOMENOLOGY_328: (16)                             *self.magnitude_range, self.color_map,
SNGT_QHENOMENOLOGY_329: (12)                         )
SNGT_QHENOMENOLOGY_330: (12)                         cs = self.coordinate_system
SNGT_QHENOMENOLOGY_331: (12)                         for line in self.submobjects:
SNGT_QHENOMENOLOGY_332: (16)                             norms = [
SNGT_QHENOMENOLOGY_333: (20)                                 get_norm(self.func(*cs.p2c(point)))
SNGT_QHENOMENOLOGY_334: (20)                                 for point in line.get_points()
SNGT_QHENOMENOLOGY_335: (16)                             ]
SNGT_QHENOMENOLOGY_336: (16)                             rgbs = values_to_rgbs(norms)
SNGT_QHENOMENOLOGY_337: (16)                             rgbas = np.zeros((len(rgbs), 4))
SNGT_QHENOMENOLOGY_338: (16)                             rgbas[:, :3] = rgbs
SNGT_QHENOMENOLOGY_339: (16)                             rgbas[:, 3] = self.stroke_opacity
SNGT_QHENOMENOLOGY_340: (16)                             line.set_rgba_array(rgbas, "stroke_rgba")
SNGT_QHENOMENOLOGY_341: (8)                     else:
SNGT_QHENOMENOLOGY_342: (12)                         self.set_stroke(self.stroke_color,
opacity=self.stroke_opacity)
SNGT_QHENOMENOLOGY_343: (8)                     if self.taper_stroke_width:
SNGT_QHENOMENOLOGY_344: (12)                         width = [0, self.stroke_width, 0]
SNGT_QHENOMENOLOGY_345: (8)                     else:
SNGT_QHENOMENOLOGY_346: (12)                         width = self.stroke_width
SNGT_QHENOMENOLOGY_347: (8)                     self.set_stroke(width=width)
SNGT_QHENOMENOLOGY_348: (0)             class AnimatedStreamLines(VGroup):
SNGT_QHENOMENOLOGY_349: (4)                 def __init__(
SNGT_QHENOMENOLOGY_350: (8)                     self,
SNGT_QHENOMENOLOGY_351: (8)                     stream_lines: StreamLines,
SNGT_QHENOMENOLOGY_352: (8)                     lag_range: float = 4,
SNGT_QHENOMENOLOGY_353: (8)                     rate_multiple: float = 1.0,
SNGT_QHENOMENOLOGY_354: (8)                     line_anim_config: dict = dict(
SNGT_QHENOMENOLOGY_355: (12)                         rate_func=linear,
SNGT_QHENOMENOLOGY_356: (12)                         time_width=1.0,
SNGT_QHENOMENOLOGY_357: (8)                     ),
SNGT_QHENOMENOLOGY_358: (8)                     **kwargs
SNGT_QHENOMENOLOGY_359: (4)                 ):
SNGT_QHENOMENOLOGY_360: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_361: (8)                     self.stream_lines = stream_lines
SNGT_QHENOMENOLOGY_362: (8)                     for line in stream_lines:
SNGT_QHENOMENOLOGY_363: (12)                         line.anim = VShowPassingFlash(
SNGT_QHENOMENOLOGY_364: (16)                             line,
SNGT_QHENOMENOLOGY_365: (16)                             run_time=line.virtual_time / rate_multiple,
SNGT_QHENOMENOLOGY_366: (16)                             **line_anim_config,
SNGT_QHENOMENOLOGY_367: (12)                         )
SNGT_QHENOMENOLOGY_368: (12)                         line.anim.begin()
SNGT_QHENOMENOLOGY_369: (12)                         line.time = -lag_range * np.random.random()
SNGT_QHENOMENOLOGY_370: (12)                         self.add(line.anim.mobject)
SNGT_QHENOMENOLOGY_371: (8)                     self.add_updater(lambda m, dt: m.update(dt))
SNGT_QHENOMENOLOGY_372: (4)                 def update(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_373: (8)                     stream_lines = self.stream_lines
SNGT_QHENOMENOLOGY_374: (8)                     for line in stream_lines:
SNGT_QHENOMENOLOGY_375: (12)                         line.time += dt
```

```
SNGT_QHENOMENOLOGY_376: (12)                        adjusted_time = max(line.time, 0) %
line.anim.run_time
SNGT_QHENOMENOLOGY_377: (12)                        line.anim.update(adjusted_time /
line.anim.run_time)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 57 - text_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from contextlib import contextmanager
SNGT_QHENOMENOLOGY_3: (0)               import os
SNGT_QHENOMENOLOGY_4: (0)               from pathlib import Path
SNGT_QHENOMENOLOGY_5: (0)               import re
SNGT_QHENOMENOLOGY_6: (0)               import tempfile
SNGT_QHENOMENOLOGY_7: (0)               from functools import lru_cache
SNGT_QHENOMENOLOGY_8: (0)               import manimpango
SNGT_QHENOMENOLOGY_9: (0)               import pygments
SNGT_QHENOMENOLOGY_10: (0)              import pygments.formatters
SNGT_QHENOMENOLOGY_11: (0)              import pygments.lexers
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.constants import DEFAULT_PIXEL_WIDTH,
FRAME_WIDTH
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.constants import NORMAL
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.logger import log
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.mobject.svg.string_mobject import
StringMobject
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.cache import cache_on_disk
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.utils.color import color_to_hex
SNGT_QHENOMENOLOGY_19: (0)              from manimlib.utils.color import int_to_hex
SNGT_QHENOMENOLOGY_20: (0)              from manimlib.utils.simple_functions import hash_string
SNGT_QHENOMENOLOGY_21: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_22: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_23: (4)                  from typing import Iterable
SNGT_QHENOMENOLOGY_24: (4)                  from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_25: (4)                  from manimlib.typing import ManimColor, Span, Selector
SNGT_QHENOMENOLOGY_26: (0)              TEXT_MOB_SCALE_FACTOR = 0.0076
SNGT_QHENOMENOLOGY_27: (0)              DEFAULT_LINE_SPACING_SCALE = 0.6
SNGT_QHENOMENOLOGY_28: (0)              DEFAULT_CANVAS_WIDTH = 16384
SNGT_QHENOMENOLOGY_29: (0)              DEFAULT_CANVAS_HEIGHT = 16384
SNGT_QHENOMENOLOGY_30: (0)              class _Alignment:
SNGT_QHENOMENOLOGY_31: (4)                  VAL_DICT = {
SNGT_QHENOMENOLOGY_32: (8)                      "LEFT": 0,
SNGT_QHENOMENOLOGY_33: (8)                      "CENTER": 1,
SNGT_QHENOMENOLOGY_34: (8)                      "RIGHT": 2
SNGT_QHENOMENOLOGY_35: (4)                  }
SNGT_QHENOMENOLOGY_36: (4)                  def __init__(self, s: str):
SNGT_QHENOMENOLOGY_37: (8)                      self.value = _Alignment.VAL_DICT[s.upper()]
SNGT_QHENOMENOLOGY_38: (0)              @lru_cache(maxsize=128)
SNGT_QHENOMENOLOGY_39: (0)              @cache_on_disk
SNGT_QHENOMENOLOGY_40: (0)              def markup_to_svg(
SNGT_QHENOMENOLOGY_41: (4)                  markup_str: str,
SNGT_QHENOMENOLOGY_42: (4)                  justify: bool = False,
SNGT_QHENOMENOLOGY_43: (4)                  indent: float = 0,
SNGT_QHENOMENOLOGY_44: (4)                  alignment: str = "CENTER",
SNGT_QHENOMENOLOGY_45: (4)                  line_width: float | None = None,
SNGT_QHENOMENOLOGY_46: (0)              ) -> str:
SNGT_QHENOMENOLOGY_47: (4)                  validate_error =
manimpango.MarkupUtils.validate(markup_str)
SNGT_QHENOMENOLOGY_48: (4)                  if validate_error:
SNGT_QHENOMENOLOGY_49: (8)                      raise ValueError(
SNGT_QHENOMENOLOGY_50: (12)                         f"Invalid markup string \"{markup_str}\"\n" + \
SNGT_QHENOMENOLOGY_51: (12)                         f"{validate_error}"
SNGT_QHENOMENOLOGY_52: (8)                      )
SNGT_QHENOMENOLOGY_53: (4)                  alignment = _Alignment(alignment)
SNGT_QHENOMENOLOGY_54: (4)                  if line_width is None:
SNGT_QHENOMENOLOGY_55: (8)                      pango_width = -1
SNGT_QHENOMENOLOGY_56: (4)                  else:
```

```
SNGT_QHENOMENOLOGY_57: (8)                     pango_width = line_width / FRAME_WIDTH *
DEFAULT_PIXEL_WIDTH
SNGT_QHENOMENOLOGY_58: (4)                 temp_file = Path(tempfile.gettempdir(),
hash_string(markup_str)).with_suffix(".svg")
SNGT_QHENOMENOLOGY_59: (4)                 manimpango.MarkupUtils.text2svg(
SNGT_QHENOMENOLOGY_60: (8)                     text=markup_str,
SNGT_QHENOMENOLOGY_61: (8)                     font="",                      # Already handled
SNGT_QHENOMENOLOGY_62: (8)                     slant="NORMAL",               # Already handled
SNGT_QHENOMENOLOGY_63: (8)                     weight="NORMAL",              # Already handled
SNGT_QHENOMENOLOGY_64: (8)                     size=1,                       # Already handled
SNGT_QHENOMENOLOGY_65: (8)                     _=0,                          # Empty parameter
SNGT_QHENOMENOLOGY_66: (8)                     disable_liga=False,
SNGT_QHENOMENOLOGY_67: (8)                     file_name=str(temp_file),
SNGT_QHENOMENOLOGY_68: (8)                     START_X=0,
SNGT_QHENOMENOLOGY_69: (8)                     START_Y=0,
SNGT_QHENOMENOLOGY_70: (8)                     width=DEFAULT_CANVAS_WIDTH,
SNGT_QHENOMENOLOGY_71: (8)                     height=DEFAULT_CANVAS_HEIGHT,
SNGT_QHENOMENOLOGY_72: (8)                     justify=justify,
SNGT_QHENOMENOLOGY_73: (8)                     indent=indent,
SNGT_QHENOMENOLOGY_74: (8)                     line_spacing=None,            # Already handled
SNGT_QHENOMENOLOGY_75: (8)                     alignment=alignment,
SNGT_QHENOMENOLOGY_76: (8)                     pango_width=pango_width
SNGT_QHENOMENOLOGY_77: (4)                 )
SNGT_QHENOMENOLOGY_78: (4)                 result = temp_file.read_text()
SNGT_QHENOMENOLOGY_79: (4)                 os.remove(temp_file)
SNGT_QHENOMENOLOGY_80: (4)                 return result
SNGT_QHENOMENOLOGY_81: (0)             class MarkupText(StringMobject):
SNGT_QHENOMENOLOGY_82: (4)                 MARKUP_TAGS = {
SNGT_QHENOMENOLOGY_83: (8)                     "b": {"font_weight": "bold"},
SNGT_QHENOMENOLOGY_84: (8)                     "big": {"font_size": "larger"},
SNGT_QHENOMENOLOGY_85: (8)                     "i": {"font_style": "italic"},
SNGT_QHENOMENOLOGY_86: (8)                     "s": {"strikethrough": "true"},
SNGT_QHENOMENOLOGY_87: (8)                     "sub": {"baseline_shift": "subscript",
"font_scale": "subscript"},
SNGT_QHENOMENOLOGY_88: (8)                     "sup": {"baseline_shift": "superscript",
"font_scale": "superscript"},
SNGT_QHENOMENOLOGY_89: (8)                     "small": {"font_size": "smaller"},
SNGT_QHENOMENOLOGY_90: (8)                     "tt": {"font_family": "monospace"},
SNGT_QHENOMENOLOGY_91: (8)                     "u": {"underline": "single"},
SNGT_QHENOMENOLOGY_92: (4)                 }
SNGT_QHENOMENOLOGY_93: (4)                 MARKUP_ENTITY_DICT = {
SNGT_QHENOMENOLOGY_94: (8)                     "<": "&lt;",
SNGT_QHENOMENOLOGY_95: (8)                     ">": "&gt;",
SNGT_QHENOMENOLOGY_96: (8)                     "&": "&amp;",
SNGT_QHENOMENOLOGY_97: (8)                     "\"": "&quot;",
SNGT_QHENOMENOLOGY_98: (8)                     "'": "&apos;"
SNGT_QHENOMENOLOGY_99: (4)                 }
SNGT_QHENOMENOLOGY_100: (4)                def __init__(
SNGT_QHENOMENOLOGY_101: (8)                    self,
SNGT_QHENOMENOLOGY_102: (8)                    text: str,
SNGT_QHENOMENOLOGY_103: (8)                    font_size: int = 48,
SNGT_QHENOMENOLOGY_104: (8)                    height: float | None = None,
SNGT_QHENOMENOLOGY_105: (8)                    justify: bool = False,
SNGT_QHENOMENOLOGY_106: (8)                    indent: float = 0,
SNGT_QHENOMENOLOGY_107: (8)                    alignment: str = "",
SNGT_QHENOMENOLOGY_108: (8)                    line_width: float | None = None,
SNGT_QHENOMENOLOGY_109: (8)                    font: str = "",
SNGT_QHENOMENOLOGY_110: (8)                    slant: str = NORMAL,
SNGT_QHENOMENOLOGY_111: (8)                    weight: str = NORMAL,
SNGT_QHENOMENOLOGY_112: (8)                    gradient: Iterable[ManimColor] | None = None,
SNGT_QHENOMENOLOGY_113: (8)                    line_spacing_height: float | None = None,
SNGT_QHENOMENOLOGY_114: (8)                    text2color: dict = {},
SNGT_QHENOMENOLOGY_115: (8)                    text2font: dict = {},
SNGT_QHENOMENOLOGY_116: (8)                    text2gradient: dict = {},
SNGT_QHENOMENOLOGY_117: (8)                    text2slant: dict = {},
SNGT_QHENOMENOLOGY_118: (8)                    text2weight: dict = {},
SNGT_QHENOMENOLOGY_119: (8)                    lsh: float | None = None,  # Overrides
line_spacing_height
SNGT_QHENOMENOLOGY_120: (8)                    t2c: dict = {},  # Overrides text2color if nonempty
```

```
SNGT_QHENOMENOLOGY_121: (8)                     t2f: dict = {},  # Overrides text2font if nonempty
SNGT_QHENOMENOLOGY_122: (8)                     t2g: dict = {},  # Overrides text2gradient if
nonempty
SNGT_QHENOMENOLOGY_123: (8)                     t2s: dict = {},  # Overrides text2slant if nonempty
SNGT_QHENOMENOLOGY_124: (8)                     t2w: dict = {},  # Overrides text2weight if
nonempty
SNGT_QHENOMENOLOGY_125: (8)                     global_config: dict = {},
SNGT_QHENOMENOLOGY_126: (8)                     local_configs: dict = {},
SNGT_QHENOMENOLOGY_127: (8)                     disable_ligatures: bool = True,
SNGT_QHENOMENOLOGY_128: (8)                     isolate: Selector = re.compile(r"\w+", re.U),
SNGT_QHENOMENOLOGY_129: (8)                     **kwargs
SNGT_QHENOMENOLOGY_130: (4)                 ):
SNGT_QHENOMENOLOGY_131: (8)                     text_config = manim_config.text
SNGT_QHENOMENOLOGY_132: (8)                     self.text = text
SNGT_QHENOMENOLOGY_133: (8)                     self.font_size = font_size
SNGT_QHENOMENOLOGY_134: (8)                     self.justify = justify
SNGT_QHENOMENOLOGY_135: (8)                     self.indent = indent
SNGT_QHENOMENOLOGY_136: (8)                     self.alignment = alignment or text_config.alignment
SNGT_QHENOMENOLOGY_137: (8)                     self.line_width = line_width
SNGT_QHENOMENOLOGY_138: (8)                     self.font = font or text_config.font
SNGT_QHENOMENOLOGY_139: (8)                     self.slant = slant
SNGT_QHENOMENOLOGY_140: (8)                     self.weight = weight
SNGT_QHENOMENOLOGY_141: (8)                     self.lsh = line_spacing_height or lsh
SNGT_QHENOMENOLOGY_142: (8)                     self.t2c = text2color or t2c
SNGT_QHENOMENOLOGY_143: (8)                     self.t2f = text2font or t2f
SNGT_QHENOMENOLOGY_144: (8)                     self.t2g = text2gradient or t2g
SNGT_QHENOMENOLOGY_145: (8)                     self.t2s = text2slant or t2s
SNGT_QHENOMENOLOGY_146: (8)                     self.t2w = text2weight or t2w
SNGT_QHENOMENOLOGY_147: (8)                     self.global_config = global_config
SNGT_QHENOMENOLOGY_148: (8)                     self.local_configs = local_configs
SNGT_QHENOMENOLOGY_149: (8)                     self.disable_ligatures = disable_ligatures
SNGT_QHENOMENOLOGY_150: (8)                     self.isolate = isolate
SNGT_QHENOMENOLOGY_151: (8)                     if not isinstance(self, Text):
SNGT_QHENOMENOLOGY_152: (12)                        self.validate_markup_string(text)
SNGT_QHENOMENOLOGY_153: (8)                     super().__init__(text, height=height, **kwargs)
SNGT_QHENOMENOLOGY_154: (8)                     if self.t2g:
SNGT_QHENOMENOLOGY_155: (12)                        log.warning("""
SNGT_QHENOMENOLOGY_156: (16)                            Manim currently cannot parse gradient from
svg.
SNGT_QHENOMENOLOGY_157: (16)                            Please set gradient via
`set_color_by_gradient`.
SNGT_QHENOMENOLOGY_158: (12)                        """)
SNGT_QHENOMENOLOGY_159: (8)                     if gradient:
SNGT_QHENOMENOLOGY_160: (12)                        self.set_color_by_gradient(*gradient)
SNGT_QHENOMENOLOGY_161: (8)                     if self.t2c:
SNGT_QHENOMENOLOGY_162: (12)                        self.set_color_by_text_to_color_map(self.t2c)
SNGT_QHENOMENOLOGY_163: (8)                     if height is None:
SNGT_QHENOMENOLOGY_164: (12)                        self.scale(TEXT_MOB_SCALE_FACTOR)
SNGT_QHENOMENOLOGY_165: (4)                 def get_svg_string_by_content(self, content: str) ->
str:
SNGT_QHENOMENOLOGY_166: (8)                     self.content = content
SNGT_QHENOMENOLOGY_167: (8)                     return markup_to_svg(
SNGT_QHENOMENOLOGY_168: (12)                        content,
SNGT_QHENOMENOLOGY_169: (12)                        justify=self.justify,
SNGT_QHENOMENOLOGY_170: (12)                        indent=self.indent,
SNGT_QHENOMENOLOGY_171: (12)                        alignment=self.alignment,
SNGT_QHENOMENOLOGY_172: (12)                        line_width=self.line_width
SNGT_QHENOMENOLOGY_173: (8)                     )
SNGT_QHENOMENOLOGY_174: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_175: (4)                 def escape_markup_char(substr: str) -> str:
SNGT_QHENOMENOLOGY_176: (8)                     return MarkupText.MARKUP_ENTITY_DICT.get(substr,
substr)
SNGT_QHENOMENOLOGY_177: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_178: (4)                 def unescape_markup_char(substr: str) -> str:
SNGT_QHENOMENOLOGY_179: (8)                     return {
SNGT_QHENOMENOLOGY_180: (12)                        v: k
SNGT_QHENOMENOLOGY_181: (12)                        for k, v in
MarkupText.MARKUP_ENTITY_DICT.items()
SNGT_QHENOMENOLOGY_182: (8)                     }.get(substr, substr)
```

```
SNGT_QHENOMENOLOGY_183: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_184: (4)                 def get_command_matches(string: str) -> list[re.Match]:
SNGT_QHENOMENOLOGY_185: (8)                     pattern = re.compile(r"""
SNGT_QHENOMENOLOGY_186: (12)                        (?P<tag>
SNGT_QHENOMENOLOGY_187: (16)                            <
SNGT_QHENOMENOLOGY_188: (16)                            (?P<close_slash>/)?
SNGT_QHENOMENOLOGY_189: (16)                            (?P<tag_name>\w+)\s*
SNGT_QHENOMENOLOGY_190: (16)                            (?P<attr_list>(?:\w+\s*\=\s*(?P<quot>
["']).*?(?P=quot)\s*)*)
SNGT_QHENOMENOLOGY_191: (16)                            (?P<elision_slash>/)?
SNGT_QHENOMENOLOGY_192: (16)                            >
SNGT_QHENOMENOLOGY_193: (12)                        )
SNGT_QHENOMENOLOGY_194: (12)                        |(?P<passthrough>
SNGT_QHENOMENOLOGY_195: (16)                            <\?.*?\?>|<!--.*?-->|<!\[CDATA\[.*?\]\]>|
<!DOCTYPE.*?>
SNGT_QHENOMENOLOGY_196: (12)                        )
SNGT_QHENOMENOLOGY_197: (12)                        |(?P<entity>&(?P<unicode>\#(?P<hex>x)?)?(?
P<content>.*?);)
SNGT_QHENOMENOLOGY_198: (12)                        |(?P<char>[>"'])
SNGT_QHENOMENOLOGY_199: (8)                     """, flags=re.X | re.S)
SNGT_QHENOMENOLOGY_200: (8)                     return list(pattern.finditer(string))
SNGT_QHENOMENOLOGY_201: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_202: (4)                 def get_command_flag(match_obj: re.Match) -> int:
SNGT_QHENOMENOLOGY_203: (8)                     if match_obj.group("tag"):
SNGT_QHENOMENOLOGY_204: (12)                        if match_obj.group("close_slash"):
SNGT_QHENOMENOLOGY_205: (16)                            return -1
SNGT_QHENOMENOLOGY_206: (12)                        if not match_obj.group("elision_slash"):
SNGT_QHENOMENOLOGY_207: (16)                            return 1
SNGT_QHENOMENOLOGY_208: (8)                     return 0
SNGT_QHENOMENOLOGY_209: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_210: (4)                 def replace_for_content(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_211: (8)                     if match_obj.group("tag"):
SNGT_QHENOMENOLOGY_212: (12)                        return ""
SNGT_QHENOMENOLOGY_213: (8)                     if match_obj.group("char"):
SNGT_QHENOMENOLOGY_214: (12)                        return
MarkupText.escape_markup_char(match_obj.group("char"))
SNGT_QHENOMENOLOGY_215: (8)                     return match_obj.group()
SNGT_QHENOMENOLOGY_216: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_217: (4)                 def replace_for_matching(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_218: (8)                     if match_obj.group("tag") or
match_obj.group("passthrough"):
SNGT_QHENOMENOLOGY_219: (12)                        return ""
SNGT_QHENOMENOLOGY_220: (8)                     if match_obj.group("entity"):
SNGT_QHENOMENOLOGY_221: (12)                        if match_obj.group("unicode"):
SNGT_QHENOMENOLOGY_222: (16)                            base = 10
SNGT_QHENOMENOLOGY_223: (16)                            if match_obj.group("hex"):
SNGT_QHENOMENOLOGY_224: (20)                                base = 16
SNGT_QHENOMENOLOGY_225: (16)                            return chr(int(match_obj.group("content"),
base))
SNGT_QHENOMENOLOGY_226: (12)                        return
MarkupText.unescape_markup_char(match_obj.group("entity"))
SNGT_QHENOMENOLOGY_227: (8)                     return match_obj.group()
SNGT_QHENOMENOLOGY_228: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_229: (4)                 def get_attr_dict_from_command_pair(
SNGT_QHENOMENOLOGY_230: (8)                     open_command: re.Match, close_command: re.Match
SNGT_QHENOMENOLOGY_231: (4)                 ) -> dict[str, str] | None:
SNGT_QHENOMENOLOGY_232: (8)                     pattern = r"""
SNGT_QHENOMENOLOGY_233: (12)                        (?P<attr_name>\w+)
SNGT_QHENOMENOLOGY_234: (12)                        \s*\=\s*
SNGT_QHENOMENOLOGY_235: (12)                        (?P<quot>["'])(?P<attr_val>.*?)(?P=quot)
SNGT_QHENOMENOLOGY_236: (8)                     """
SNGT_QHENOMENOLOGY_237: (8)                     tag_name = open_command.group("tag_name")
SNGT_QHENOMENOLOGY_238: (8)                     if tag_name == "span":
SNGT_QHENOMENOLOGY_239: (12)                        return {
SNGT_QHENOMENOLOGY_240: (16)                            match_obj.group("attr_name"):
match_obj.group("attr_val")
SNGT_QHENOMENOLOGY_241: (16)                            for match_obj in re.finditer(
SNGT_QHENOMENOLOGY_242: (20)                                pattern,
open_command.group("attr_list"), re.S | re.X
```

```
SNGT_QHENOMENOLOGY_243: (16)                         )
SNGT_QHENOMENOLOGY_244: (12)                     }
SNGT_QHENOMENOLOGY_245: (8)                 return MarkupText.MARKUP_TAGS.get(tag_name, {})
SNGT_QHENOMENOLOGY_246: (4)             def get_configured_items(self) -> list[tuple[Span,
dict[str, str]]]:
SNGT_QHENOMENOLOGY_247: (8)                 return [
SNGT_QHENOMENOLOGY_248: (12)                     *(
SNGT_QHENOMENOLOGY_249: (16)                         (span, {key: val})
SNGT_QHENOMENOLOGY_250: (16)                         for t2x_dict, key in (
SNGT_QHENOMENOLOGY_251: (20)                             (self.t2c, "foreground"),
SNGT_QHENOMENOLOGY_252: (20)                             (self.t2f, "font_family"),
SNGT_QHENOMENOLOGY_253: (20)                             (self.t2s, "font_style"),
SNGT_QHENOMENOLOGY_254: (20)                             (self.t2w, "font_weight")
SNGT_QHENOMENOLOGY_255: (16)                         )
SNGT_QHENOMENOLOGY_256: (16)                         for selector, val in t2x_dict.items()
SNGT_QHENOMENOLOGY_257: (16)                         for span in
self.find_spans_by_selector(selector)
SNGT_QHENOMENOLOGY_258: (12)                     ),
SNGT_QHENOMENOLOGY_259: (12)                     *(
SNGT_QHENOMENOLOGY_260: (16)                         (span, local_config)
SNGT_QHENOMENOLOGY_261: (16)                         for selector, local_config in
self.local_configs.items()
SNGT_QHENOMENOLOGY_262: (16)                         for span in
self.find_spans_by_selector(selector)
SNGT_QHENOMENOLOGY_263: (12)                     )
SNGT_QHENOMENOLOGY_264: (8)                 ]
SNGT_QHENOMENOLOGY_265: (4)             @staticmethod
SNGT_QHENOMENOLOGY_266: (4)             def get_command_string(
SNGT_QHENOMENOLOGY_267: (8)                 attr_dict: dict[str, str], is_end: bool, label_hex:
str | None
SNGT_QHENOMENOLOGY_268: (4)             ) -> str:
SNGT_QHENOMENOLOGY_269: (8)                 if is_end:
SNGT_QHENOMENOLOGY_270: (12)                     return "</span>"
SNGT_QHENOMENOLOGY_271: (8)                 if label_hex is not None:
SNGT_QHENOMENOLOGY_272: (12)                     converted_attr_dict = {"foreground": label_hex}
SNGT_QHENOMENOLOGY_273: (12)                     for key, val in attr_dict.items():
SNGT_QHENOMENOLOGY_274: (16)                         if key in (
SNGT_QHENOMENOLOGY_275: (20)                             "background", "bgcolor",
SNGT_QHENOMENOLOGY_276: (20)                             "underline_color", "overline_color",
"strikethrough_color"
SNGT_QHENOMENOLOGY_277: (16)                         ):
SNGT_QHENOMENOLOGY_278: (20)                             converted_attr_dict[key] = "black"
SNGT_QHENOMENOLOGY_279: (16)                         elif key not in ("foreground", "fgcolor",
"color"):
SNGT_QHENOMENOLOGY_280: (20)                             converted_attr_dict[key] = val
SNGT_QHENOMENOLOGY_281: (8)                 else:
SNGT_QHENOMENOLOGY_282: (12)                     converted_attr_dict = attr_dict.copy()
SNGT_QHENOMENOLOGY_283: (8)                 attrs_str = " ".join([
SNGT_QHENOMENOLOGY_284: (12)                     f"{key}='{val}'"
SNGT_QHENOMENOLOGY_285: (12)                     for key, val in converted_attr_dict.items()
SNGT_QHENOMENOLOGY_286: (8)                 ])
SNGT_QHENOMENOLOGY_287: (8)                 return f"<span {attrs_str}>"
SNGT_QHENOMENOLOGY_288: (4)             def get_content_prefix_and_suffix(
SNGT_QHENOMENOLOGY_289: (8)                 self, is_labelled: bool
SNGT_QHENOMENOLOGY_290: (4)             ) -> tuple[str, str]:
SNGT_QHENOMENOLOGY_291: (8)                 global_attr_dict = {
SNGT_QHENOMENOLOGY_292: (12)                     "foreground": color_to_hex(self.base_color),
SNGT_QHENOMENOLOGY_293: (12)                     "font_family": self.font,
SNGT_QHENOMENOLOGY_294: (12)                     "font_style": self.slant,
SNGT_QHENOMENOLOGY_295: (12)                     "font_weight": self.weight,
SNGT_QHENOMENOLOGY_296: (12)                     "font_size": str(round(self.font_size * 1024)),
SNGT_QHENOMENOLOGY_297: (8)                 }
SNGT_QHENOMENOLOGY_298: (8)                 pango_version = manimpango.pango_version()
SNGT_QHENOMENOLOGY_299: (8)                 if tuple(map(int, pango_version.split("."))) < (1,
50):
SNGT_QHENOMENOLOGY_300: (12)                     if self.lsh is not None:
SNGT_QHENOMENOLOGY_301: (16)                         log.warning(
SNGT_QHENOMENOLOGY_302: (20)                             "Pango version %s found (< 1.50), "
SNGT_QHENOMENOLOGY_303: (20)                             "unable to set `line_height`
```

```
attribute",
SNGT_QHENOMENOLOGY_304: (20)                                pango_version
SNGT_QHENOMENOLOGY_305: (16)                            )
SNGT_QHENOMENOLOGY_306: (8)                         else:
SNGT_QHENOMENOLOGY_307: (12)                            line_spacing_scale = self.lsh or
DEFAULT_LINE_SPACING_SCALE
SNGT_QHENOMENOLOGY_308: (12)                            global_attr_dict["line_height"] = str(
SNGT_QHENOMENOLOGY_309: (16)                                ((line_spacing_scale) + 1) * 0.6
SNGT_QHENOMENOLOGY_310: (12)                            )
SNGT_QHENOMENOLOGY_311: (8)                         if self.disable_ligatures:
SNGT_QHENOMENOLOGY_312: (12)                            global_attr_dict["font_features"] =
"liga=0,dlig=0,clig=0,hlig=0"
SNGT_QHENOMENOLOGY_313: (8)                         global_attr_dict.update(self.global_config)
SNGT_QHENOMENOLOGY_314: (8)                         return tuple(
SNGT_QHENOMENOLOGY_315: (12)                            self.get_command_string(
SNGT_QHENOMENOLOGY_316: (16)                                global_attr_dict,
SNGT_QHENOMENOLOGY_317: (16)                                is_end=is_end,
SNGT_QHENOMENOLOGY_318: (16)                                label_hex=int_to_hex(0) if is_labelled else
None
SNGT_QHENOMENOLOGY_319: (12)                            )
SNGT_QHENOMENOLOGY_320: (12)                            for is_end in (False, True)
SNGT_QHENOMENOLOGY_321: (8)                         )
SNGT_QHENOMENOLOGY_322: (4)                     def get_parts_by_text(self, selector: Selector) ->
VGroup:
SNGT_QHENOMENOLOGY_323: (8)                         return self.select_parts(selector)
SNGT_QHENOMENOLOGY_324: (4)                     def get_part_by_text(self, selector: Selector,
**kwargs) -> VGroup:
SNGT_QHENOMENOLOGY_325: (8)                         return self.select_part(selector, **kwargs)
SNGT_QHENOMENOLOGY_326: (4)                     def set_color_by_text(self, selector: Selector, color:
ManimColor):
SNGT_QHENOMENOLOGY_327: (8)                         return self.set_parts_color(selector, color)
SNGT_QHENOMENOLOGY_328: (4)                     def set_color_by_text_to_color_map(
SNGT_QHENOMENOLOGY_329: (8)                         self, color_map: dict[Selector, ManimColor]
SNGT_QHENOMENOLOGY_330: (4)                     ):
SNGT_QHENOMENOLOGY_331: (8)                         return self.set_parts_color_by_dict(color_map)
SNGT_QHENOMENOLOGY_332: (4)                     def get_text(self) -> str:
SNGT_QHENOMENOLOGY_333: (8)                         return self.get_string()
SNGT_QHENOMENOLOGY_334: (0)                 class Text(MarkupText):
SNGT_QHENOMENOLOGY_335: (4)                     def __init__(
SNGT_QHENOMENOLOGY_336: (8)                         self,
SNGT_QHENOMENOLOGY_337: (8)                         text: str,
SNGT_QHENOMENOLOGY_338: (8)                         isolate: Selector = (re.compile(r"\w+", re.U),
re.compile(r"\S+", re.U)),
SNGT_QHENOMENOLOGY_339: (8)                         use_labelled_svg: bool = True,
SNGT_QHENOMENOLOGY_340: (8)                         path_string_config: dict = dict(
SNGT_QHENOMENOLOGY_341: (12)                            use_simple_quadratic_approx=True,
SNGT_QHENOMENOLOGY_342: (8)                         ),
SNGT_QHENOMENOLOGY_343: (8)                         **kwargs
SNGT_QHENOMENOLOGY_344: (4)                     ):
SNGT_QHENOMENOLOGY_345: (8)                         super().__init__(
SNGT_QHENOMENOLOGY_346: (12)                            text,
SNGT_QHENOMENOLOGY_347: (12)                            isolate=isolate,
SNGT_QHENOMENOLOGY_348: (12)                            use_labelled_svg=use_labelled_svg,
SNGT_QHENOMENOLOGY_349: (12)                            path_string_config=path_string_config,
SNGT_QHENOMENOLOGY_350: (12)                            **kwargs
SNGT_QHENOMENOLOGY_351: (8)                         )
SNGT_QHENOMENOLOGY_352: (4)                     @staticmethod
SNGT_QHENOMENOLOGY_353: (4)                     def get_command_matches(string: str) -> list[re.Match]:
SNGT_QHENOMENOLOGY_354: (8)                         pattern = re.compile(r"""[<>&"']""")
SNGT_QHENOMENOLOGY_355: (8)                         return list(pattern.finditer(string))
SNGT_QHENOMENOLOGY_356: (4)                     @staticmethod
SNGT_QHENOMENOLOGY_357: (4)                     def get_command_flag(match_obj: re.Match) -> int:
SNGT_QHENOMENOLOGY_358: (8)                         return 0
SNGT_QHENOMENOLOGY_359: (4)                     @staticmethod
SNGT_QHENOMENOLOGY_360: (4)                     def replace_for_content(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_361: (8)                         return Text.escape_markup_char(match_obj.group())
SNGT_QHENOMENOLOGY_362: (4)                     @staticmethod
SNGT_QHENOMENOLOGY_363: (4)                     def replace_for_matching(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_364: (8)                         return match_obj.group()
```

```
SNGT_QHENOMENOLOGY_365: (0)            class Code(MarkupText):
SNGT_QHENOMENOLOGY_366: (4)                def __init__(
SNGT_QHENOMENOLOGY_367: (8)                    self,
SNGT_QHENOMENOLOGY_368: (8)                    code: str,
SNGT_QHENOMENOLOGY_369: (8)                    font: str = "Consolas",
SNGT_QHENOMENOLOGY_370: (8)                    font_size: int = 24,
SNGT_QHENOMENOLOGY_371: (8)                    lsh: float = 1.0,
SNGT_QHENOMENOLOGY_372: (8)                    fill_color: ManimColor = None,
SNGT_QHENOMENOLOGY_373: (8)                    stroke_color: ManimColor = None,
SNGT_QHENOMENOLOGY_374: (8)                    language: str = "python",
SNGT_QHENOMENOLOGY_375: (8)                    code_style: str = "monokai",
SNGT_QHENOMENOLOGY_376: (8)                    **kwargs
SNGT_QHENOMENOLOGY_377: (4)                ):
SNGT_QHENOMENOLOGY_378: (8)                    lexer = pygments.lexers.get_lexer_by_name(language)
SNGT_QHENOMENOLOGY_379: (8)                    formatter =
pygments.formatters.PangoMarkupFormatter(
SNGT_QHENOMENOLOGY_380: (12)                       style=code_style
SNGT_QHENOMENOLOGY_381: (8)                    )
SNGT_QHENOMENOLOGY_382: (8)                    markup = pygments.highlight(code, lexer, formatter)
SNGT_QHENOMENOLOGY_383: (8)                    markup = re.sub(r"</?tt>", "", markup)
SNGT_QHENOMENOLOGY_384: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_385: (12)                       markup,
SNGT_QHENOMENOLOGY_386: (12)                       font=font,
SNGT_QHENOMENOLOGY_387: (12)                       font_size=font_size,
SNGT_QHENOMENOLOGY_388: (12)                       lsh=lsh,
SNGT_QHENOMENOLOGY_389: (12)                       stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_390: (12)                       fill_color=fill_color,
SNGT_QHENOMENOLOGY_391: (12)                       **kwargs
SNGT_QHENOMENOLOGY_392: (8)                    )
SNGT_QHENOMENOLOGY_393: (0)            @contextmanager
SNGT_QHENOMENOLOGY_394: (0)            def register_font(font_file: str | Path):
SNGT_QHENOMENOLOGY_395: (4)                """Temporarily add a font file to Pango's search path.
SNGT_QHENOMENOLOGY_396: (4)                This searches for the font_file at various places. The
order it searches it described below.
SNGT_QHENOMENOLOGY_397: (4)                1. Absolute path.
SNGT_QHENOMENOLOGY_398: (4)                2. Downloads dir.
SNGT_QHENOMENOLOGY_399: (4)                Parameters
SNGT_QHENOMENOLOGY_400: (4)                ----------
SNGT_QHENOMENOLOGY_401: (4)                font_file :
SNGT_QHENOMENOLOGY_402: (8)                    The font file to add.
SNGT_QHENOMENOLOGY_403: (4)                Examples
SNGT_QHENOMENOLOGY_404: (4)                --------
SNGT_QHENOMENOLOGY_405: (4)                Use ``with register_font(...)`` to add a font file to
search
SNGT_QHENOMENOLOGY_406: (4)                path.
SNGT_QHENOMENOLOGY_407: (4)                .. code-block:: python
SNGT_QHENOMENOLOGY_408: (8)                    with register_font("path/to/font_file.ttf"):
SNGT_QHENOMENOLOGY_409: (11)                      a = Text("Hello", font="Custom Font Name")
SNGT_QHENOMENOLOGY_410: (4)                Raises
SNGT_QHENOMENOLOGY_411: (4)                ------
SNGT_QHENOMENOLOGY_412: (4)                FileNotFoundError:
SNGT_QHENOMENOLOGY_413: (8)                    If the font doesn't exists.
SNGT_QHENOMENOLOGY_414: (4)                AttributeError:
SNGT_QHENOMENOLOGY_415: (8)                    If this method is used on macOS.
SNGT_QHENOMENOLOGY_416: (4)                Notes
SNGT_QHENOMENOLOGY_417: (4)                -----
SNGT_QHENOMENOLOGY_418: (4)                This method of adding font files also works with
:class:`CairoText`.
SNGT_QHENOMENOLOGY_419: (4)                .. important ::
SNGT_QHENOMENOLOGY_420: (8)                    This method is available for macOS for
``ManimPango>=v0.2.3``. Using this
SNGT_QHENOMENOLOGY_421: (8)                    method with previous releases will raise an
:class:`AttributeError` on macOS.
SNGT_QHENOMENOLOGY_422: (4)                """
SNGT_QHENOMENOLOGY_423: (4)                file_path = Path(font_file).resolve()
SNGT_QHENOMENOLOGY_424: (4)                if not file_path.exists():
SNGT_QHENOMENOLOGY_425: (8)                    error = f"Can't find {font_file}."
SNGT_QHENOMENOLOGY_426: (8)                    raise FileNotFoundError(error)
SNGT_QHENOMENOLOGY_427: (4)                try:
```

```
SNGT_QHENOMENOLOGY_428: (8)                     assert manimpango.register_font(str(file_path))
SNGT_QHENOMENOLOGY_429: (8)                     yield
SNGT_QHENOMENOLOGY_430: (4)                 finally:
SNGT_QHENOMENOLOGY_431: (8)                     manimpango.unregister_font(str(file_path))
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 58 - value_tracker.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_5: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_6: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_7: (4)                   from manimlib.typing import Self
SNGT_QHENOMENOLOGY_8: (0)               class ValueTracker(Mobject):
SNGT_QHENOMENOLOGY_9: (4)                   """
SNGT_QHENOMENOLOGY_10: (4)                  Not meant to be displayed.  Instead the position
encodes some
SNGT_QHENOMENOLOGY_11: (4)                  number, often one which another animation or
continual_animation
SNGT_QHENOMENOLOGY_12: (4)                  uses for its update function, and by treating it as a
mobject it can
SNGT_QHENOMENOLOGY_13: (4)                  still be animated and manipulated just like anything
else.
SNGT_QHENOMENOLOGY_14: (4)                  """
SNGT_QHENOMENOLOGY_15: (4)                  value_type: type = np.float64
SNGT_QHENOMENOLOGY_16: (4)                  def __init__(
SNGT_QHENOMENOLOGY_17: (8)                      self,
SNGT_QHENOMENOLOGY_18: (8)                      value: float | complex | np.ndarray = 0,
SNGT_QHENOMENOLOGY_19: (8)                      **kwargs
SNGT_QHENOMENOLOGY_20: (4)                  ):
SNGT_QHENOMENOLOGY_21: (8)                      self.value = value
SNGT_QHENOMENOLOGY_22: (8)                      super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_23: (4)                  def init_uniforms(self) -> None:
SNGT_QHENOMENOLOGY_24: (8)                      super().init_uniforms()
SNGT_QHENOMENOLOGY_25: (8)                      self.uniforms["value"] = np.array(
SNGT_QHENOMENOLOGY_26: (12)                         listify(self.value),
SNGT_QHENOMENOLOGY_27: (12)                         dtype=self.value_type,
SNGT_QHENOMENOLOGY_28: (8)                      )
SNGT_QHENOMENOLOGY_29: (4)                  def get_value(self) -> float | complex | np.ndarray:
SNGT_QHENOMENOLOGY_30: (8)                      result = self.uniforms["value"]
SNGT_QHENOMENOLOGY_31: (8)                      if len(result) == 1:
SNGT_QHENOMENOLOGY_32: (12)                         return result[0]
SNGT_QHENOMENOLOGY_33: (8)                      return result
SNGT_QHENOMENOLOGY_34: (4)                  def set_value(self, value: float | complex |
np.ndarray) -> Self:
SNGT_QHENOMENOLOGY_35: (8)                      self.uniforms["value"][:] = value
SNGT_QHENOMENOLOGY_36: (8)                      return self
SNGT_QHENOMENOLOGY_37: (4)                  def increment_value(self, d_value: float | complex) ->
None:
SNGT_QHENOMENOLOGY_38: (8)                      self.set_value(self.get_value() + d_value)
SNGT_QHENOMENOLOGY_39: (0)              class ExponentialValueTracker(ValueTracker):
SNGT_QHENOMENOLOGY_40: (4)                  """
SNGT_QHENOMENOLOGY_41: (4)                  Operates just like ValueTracker, except it encodes the
value as the
SNGT_QHENOMENOLOGY_42: (4)                  exponential of a position coordinate, which changes how
interpolation
SNGT_QHENOMENOLOGY_43: (4)                  behaves
SNGT_QHENOMENOLOGY_44: (4)                  """
SNGT_QHENOMENOLOGY_45: (4)                  def get_value(self) -> float | complex:
SNGT_QHENOMENOLOGY_46: (8)                      return np.exp(ValueTracker.get_value(self))
SNGT_QHENOMENOLOGY_47: (4)                  def set_value(self, value: float | complex):
SNGT_QHENOMENOLOGY_48: (8)                      return ValueTracker.set_value(self, np.log(value))
SNGT_QHENOMENOLOGY_49: (0)              class ComplexValueTracker(ValueTracker):
SNGT_QHENOMENOLOGY_50: (4)                  value_type: type = np.complex128
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 59 - image_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               import moderngl
SNGT_QHENOMENOLOGY_4: (0)               from PIL import Image
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import DL, DR, UL, UR
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.utils.bezier import inverse_interpolate
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.utils.images import
get_full_raster_image_path
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.utils.iterables import listify
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_11: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_12: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_13: (4)                  from typing import Sequence, Tuple
SNGT_QHENOMENOLOGY_14: (4)                  from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_15: (0)              class ImageMobject(Mobject):
SNGT_QHENOMENOLOGY_16: (4)                  shader_folder: str = "image"
SNGT_QHENOMENOLOGY_17: (4)                  data_dtype: Sequence[Tuple[str, type, Tuple[int]]] = [
SNGT_QHENOMENOLOGY_18: (8)                      ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_19: (8)                      ('im_coords', np.float32, (2,)),
SNGT_QHENOMENOLOGY_20: (8)                      ('opacity', np.float32, (1,)),
SNGT_QHENOMENOLOGY_21: (4)                  ]
SNGT_QHENOMENOLOGY_22: (4)                  render_primitive: int = moderngl.TRIANGLES
SNGT_QHENOMENOLOGY_23: (4)                  def __init__(
SNGT_QHENOMENOLOGY_24: (8)                      self,
SNGT_QHENOMENOLOGY_25: (8)                      filename: str,
SNGT_QHENOMENOLOGY_26: (8)                      height: float = 4.0,
SNGT_QHENOMENOLOGY_27: (8)                      **kwargs
SNGT_QHENOMENOLOGY_28: (4)                  ):
SNGT_QHENOMENOLOGY_29: (8)                      self.height = height
SNGT_QHENOMENOLOGY_30: (8)                      self.image_path =
get_full_raster_image_path(filename)
SNGT_QHENOMENOLOGY_31: (8)                      self.image = Image.open(self.image_path)
SNGT_QHENOMENOLOGY_32: (8)                      super().__init__(texture_paths={"Texture":
self.image_path}, **kwargs)
SNGT_QHENOMENOLOGY_33: (4)                  def init_data(self) -> None:
SNGT_QHENOMENOLOGY_34: (8)                      super().init_data(length=6)
SNGT_QHENOMENOLOGY_35: (8)                      self.data["point"][:] = [UL, DL, UR, DR, UR, DL]
SNGT_QHENOMENOLOGY_36: (8)                      self.data["im_coords"][:] = [(0, 0), (0, 1), (1,
0), (1, 1), (1, 0), (0, 1)]
SNGT_QHENOMENOLOGY_37: (8)                      self.data["opacity"][:] = self.opacity
SNGT_QHENOMENOLOGY_38: (4)                  def init_points(self) -> None:
SNGT_QHENOMENOLOGY_39: (8)                      size = self.image.size
SNGT_QHENOMENOLOGY_40: (8)                      self.set_width(2 * size[0] / size[1], stretch=True)
SNGT_QHENOMENOLOGY_41: (8)                      self.set_height(self.height)
SNGT_QHENOMENOLOGY_42: (4)                  @Mobject.affects_data
SNGT_QHENOMENOLOGY_43: (4)                  def set_opacity(self, opacity: float, recurse: bool =
True):
SNGT_QHENOMENOLOGY_44: (8)                      self.data["opacity"][:, 0] =
resize_with_interpolation(
SNGT_QHENOMENOLOGY_45: (12)                         np.array(listify(opacity)),
SNGT_QHENOMENOLOGY_46: (12)                         self.get_num_points()
SNGT_QHENOMENOLOGY_47: (8)                      )
SNGT_QHENOMENOLOGY_48: (8)                      return self
SNGT_QHENOMENOLOGY_49: (4)                  def set_color(self, color, opacity=None, recurse=None):
SNGT_QHENOMENOLOGY_50: (8)                      return self
SNGT_QHENOMENOLOGY_51: (4)                  def point_to_rgb(self, point: Vect3) -> Vect3:
SNGT_QHENOMENOLOGY_52: (8)                      x0, y0 = self.get_corner(UL)[:2]
SNGT_QHENOMENOLOGY_53: (8)                      x1, y1 = self.get_corner(DR)[:2]
SNGT_QHENOMENOLOGY_54: (8)                      x_alpha = inverse_interpolate(x0, x1, point[0])
SNGT_QHENOMENOLOGY_55: (8)                      y_alpha = inverse_interpolate(y0, y1, point[1])
SNGT_QHENOMENOLOGY_56: (8)                      if not (0 <= x_alpha <= 1) and (0 <= y_alpha <= 1):
SNGT_QHENOMENOLOGY_57: (12)                         raise Exception("Cannot sample color from
outside an image")
SNGT_QHENOMENOLOGY_58: (8)                      pw, ph = self.image.size
```

```
SNGT_QHENOMENOLOGY_59: (8)                    rgb = self.image.getpixel((
SNGT_QHENOMENOLOGY_60: (12)                       int((pw - 1) * x_alpha),
SNGT_QHENOMENOLOGY_61: (12)                       int((ph - 1) * y_alpha),
SNGT_QHENOMENOLOGY_62: (8)                    ))[:3]
SNGT_QHENOMENOLOGY_63: (8)                    return np.array(rgb) / 255
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 60 - shape_matchers.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from colour import Color
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.constants import BLACK, RED, YELLOW, WHITE
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.constants import DL, DOWN, DR, LEFT, RIGHT,
UL, UR
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.constants import SMALL_BUFF
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_10: (0)            from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_11: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_12: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_13: (4)                from typing import Sequence
SNGT_QHENOMENOLOGY_14: (4)                from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_15: (4)                from manimlib.typing import ManimColor, Self
SNGT_QHENOMENOLOGY_16: (0)            class SurroundingRectangle(Rectangle):
SNGT_QHENOMENOLOGY_17: (4)                def __init__(
SNGT_QHENOMENOLOGY_18: (8)                    self,
SNGT_QHENOMENOLOGY_19: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_20: (8)                    buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_21: (8)                    color: ManimColor = YELLOW,
SNGT_QHENOMENOLOGY_22: (8)                    **kwargs
SNGT_QHENOMENOLOGY_23: (4)                ):
SNGT_QHENOMENOLOGY_24: (8)                    super().__init__(color=color, **kwargs)
SNGT_QHENOMENOLOGY_25: (8)                    self.buff = buff
SNGT_QHENOMENOLOGY_26: (8)                    self.surround(mobject)
SNGT_QHENOMENOLOGY_27: (8)                    if mobject.is_fixed_in_frame():
SNGT_QHENOMENOLOGY_28: (12)                       self.fix_in_frame()
SNGT_QHENOMENOLOGY_29: (4)                def surround(self, mobject, buff=None) -> Self:
SNGT_QHENOMENOLOGY_30: (8)                    self.mobject = mobject
SNGT_QHENOMENOLOGY_31: (8)                    self.buff = buff if buff is not None else self.buff
SNGT_QHENOMENOLOGY_32: (8)                    super().surround(mobject, self.buff)
SNGT_QHENOMENOLOGY_33: (8)                    return self
SNGT_QHENOMENOLOGY_34: (4)                def set_buff(self, buff) -> Self:
SNGT_QHENOMENOLOGY_35: (8)                    self.buff = buff
SNGT_QHENOMENOLOGY_36: (8)                    self.surround(self.mobject)
SNGT_QHENOMENOLOGY_37: (8)                    return self
SNGT_QHENOMENOLOGY_38: (0)            class BackgroundRectangle(SurroundingRectangle):
SNGT_QHENOMENOLOGY_39: (4)                def __init__(
SNGT_QHENOMENOLOGY_40: (8)                    self,
SNGT_QHENOMENOLOGY_41: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_42: (8)                    color: ManimColor = None,
SNGT_QHENOMENOLOGY_43: (8)                    stroke_width: float = 0,
SNGT_QHENOMENOLOGY_44: (8)                    stroke_opacity: float = 0,
SNGT_QHENOMENOLOGY_45: (8)                    fill_opacity: float = 0.75,
SNGT_QHENOMENOLOGY_46: (8)                    buff: float = 0,
SNGT_QHENOMENOLOGY_47: (8)                    **kwargs
SNGT_QHENOMENOLOGY_48: (4)                ):
SNGT_QHENOMENOLOGY_49: (8)                    if color is None:
SNGT_QHENOMENOLOGY_50: (12)                       color = manim_config.camera.background_color
SNGT_QHENOMENOLOGY_51: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_52: (12)                       mobject,
SNGT_QHENOMENOLOGY_53: (12)                       color=color,
SNGT_QHENOMENOLOGY_54: (12)                       stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_55: (12)                       stroke_opacity=stroke_opacity,
SNGT_QHENOMENOLOGY_56: (12)                       fill_opacity=fill_opacity,
```

```
SNGT_QHENOMENOLOGY_57: (12)                         buff=buff,
SNGT_QHENOMENOLOGY_58: (12)                         **kwargs
SNGT_QHENOMENOLOGY_59: (8)                     )
SNGT_QHENOMENOLOGY_60: (8)                     self.original_fill_opacity = fill_opacity
SNGT_QHENOMENOLOGY_61: (4)                 def pointwise_become_partial(self, mobject: Mobject, a:
float, b: float) -> Self:
SNGT_QHENOMENOLOGY_62: (8)                     self.set_fill(opacity=b *
self.original_fill_opacity)
SNGT_QHENOMENOLOGY_63: (8)                     return self
SNGT_QHENOMENOLOGY_64: (4)                 def set_style(
SNGT_QHENOMENOLOGY_65: (8)                     self,
SNGT_QHENOMENOLOGY_66: (8)                     stroke_color: ManimColor | None = None,
SNGT_QHENOMENOLOGY_67: (8)                     stroke_width: float | None = None,
SNGT_QHENOMENOLOGY_68: (8)                     fill_color: ManimColor | None = None,
SNGT_QHENOMENOLOGY_69: (8)                     fill_opacity: float | None = None,
SNGT_QHENOMENOLOGY_70: (8)                     family: bool = True
SNGT_QHENOMENOLOGY_71: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_72: (8)                     VMobject.set_style(
SNGT_QHENOMENOLOGY_73: (12)                         self,
SNGT_QHENOMENOLOGY_74: (12)                         stroke_color=BLACK,
SNGT_QHENOMENOLOGY_75: (12)                         stroke_width=0,
SNGT_QHENOMENOLOGY_76: (12)                         fill_color=BLACK,
SNGT_QHENOMENOLOGY_77: (12)                         fill_opacity=fill_opacity
SNGT_QHENOMENOLOGY_78: (8)                     )
SNGT_QHENOMENOLOGY_79: (8)                     return self
SNGT_QHENOMENOLOGY_80: (4)                 def get_fill_color(self) -> Color:
SNGT_QHENOMENOLOGY_81: (8)                     return Color(self.color)
SNGT_QHENOMENOLOGY_82: (0)             class Cross(VGroup):
SNGT_QHENOMENOLOGY_83: (4)                 def __init__(
SNGT_QHENOMENOLOGY_84: (8)                     self,
SNGT_QHENOMENOLOGY_85: (8)                     mobject: Mobject,
SNGT_QHENOMENOLOGY_86: (8)                     stroke_color: ManimColor = RED,
SNGT_QHENOMENOLOGY_87: (8)                     stroke_width: float | Sequence[float] = [0, 6, 0],
SNGT_QHENOMENOLOGY_88: (8)                     **kwargs
SNGT_QHENOMENOLOGY_89: (4)                 ):
SNGT_QHENOMENOLOGY_90: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_91: (12)                         Line(UL, DR),
SNGT_QHENOMENOLOGY_92: (12)                         Line(UR, DL),
SNGT_QHENOMENOLOGY_93: (8)                     )
SNGT_QHENOMENOLOGY_94: (8)                     self.insert_n_curves(20)
SNGT_QHENOMENOLOGY_95: (8)                     self.replace(mobject, stretch=True)
SNGT_QHENOMENOLOGY_96: (8)                     self.set_stroke(stroke_color, width=stroke_width)
SNGT_QHENOMENOLOGY_97: (0)             class Underline(Line):
SNGT_QHENOMENOLOGY_98: (4)                 def __init__(
SNGT_QHENOMENOLOGY_99: (8)                     self,
SNGT_QHENOMENOLOGY_100: (8)                    mobject: Mobject,
SNGT_QHENOMENOLOGY_101: (8)                    buff: float = SMALL_BUFF,
SNGT_QHENOMENOLOGY_102: (8)                    stroke_color=WHITE,
SNGT_QHENOMENOLOGY_103: (8)                    stroke_width: float | Sequence[float] = [0, 3, 3,
0],
SNGT_QHENOMENOLOGY_104: (8)                    stretch_factor=1.2,
SNGT_QHENOMENOLOGY_105: (8)                    **kwargs
SNGT_QHENOMENOLOGY_106: (4)                ):
SNGT_QHENOMENOLOGY_107: (8)                    super().__init__(LEFT, RIGHT, **kwargs)
SNGT_QHENOMENOLOGY_108: (8)                    if not isinstance(stroke_width, (float, int)):
SNGT_QHENOMENOLOGY_109: (12)                       self.insert_n_curves(len(stroke_width) - 2)
SNGT_QHENOMENOLOGY_110: (8)                    self.set_stroke(stroke_color, stroke_width)
SNGT_QHENOMENOLOGY_111: (8)                    self.set_width(mobject.get_width() *
stretch_factor)
SNGT_QHENOMENOLOGY_112: (8)                    self.next_to(mobject, DOWN, buff=buff)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 61 - string_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from abc import ABC, abstractmethod
SNGT_QHENOMENOLOGY_3: (0)              import itertools as it
SNGT_QHENOMENOLOGY_4: (0)              import re
```

```
SNGT_QHENOMENOLOGY_5: (0)               from scipy.optimize import linear_sum_assignment
SNGT_QHENOMENOLOGY_6: (0)               from scipy.spatial.distance import cdist
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.logger import log
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.svg.svg_mobject import SVGMobject
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.color import color_to_hex
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.color import hex_to_int
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.color import int_to_hex
SNGT_QHENOMENOLOGY_15: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_16: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_17: (4)                  from typing import Callable
SNGT_QHENOMENOLOGY_18: (4)                  from manimlib.typing import ManimColor, Span, Selector
SNGT_QHENOMENOLOGY_19: (0)              class StringMobject(SVGMobject, ABC):
SNGT_QHENOMENOLOGY_20: (4)                  """
SNGT_QHENOMENOLOGY_21: (4)                  An abstract base class for `Tex` and `MarkupText`
SNGT_QHENOMENOLOGY_22: (4)                  This class aims to optimize the logic of "slicing
submobjects
SNGT_QHENOMENOLOGY_23: (4)                  via substrings". This could be much clearer and more
user-friendly
SNGT_QHENOMENOLOGY_24: (4)                  than slicing through numerical indices explicitly.
SNGT_QHENOMENOLOGY_25: (4)                  Users are expected to specify substrings in `isolate`
parameter
SNGT_QHENOMENOLOGY_26: (4)                  if they want to do anything with their corresponding
submobjects.
SNGT_QHENOMENOLOGY_27: (4)                  `isolate` parameter can be either a string, a
`re.Pattern` object,
SNGT_QHENOMENOLOGY_28: (4)                  or a 2-tuple containing integers or None, or a
collection of the above.
SNGT_QHENOMENOLOGY_29: (4)                  Note, substrings specified cannot *partly* overlap with
each other.
SNGT_QHENOMENOLOGY_30: (4)                  Each instance of `StringMobject` may generate 2 svg
files.
SNGT_QHENOMENOLOGY_31: (4)                  The additional one is generated with some color
commands inserted,
SNGT_QHENOMENOLOGY_32: (4)                  so that each submobject of the original `SVGMobject`
will be labelled
SNGT_QHENOMENOLOGY_33: (4)                  by the color of its paired submobject from the
additional `SVGMobject`.
SNGT_QHENOMENOLOGY_34: (4)                  """
SNGT_QHENOMENOLOGY_35: (4)                  height = None
SNGT_QHENOMENOLOGY_36: (4)                  def __init__(
SNGT_QHENOMENOLOGY_37: (8)                      self,
SNGT_QHENOMENOLOGY_38: (8)                      string: str,
SNGT_QHENOMENOLOGY_39: (8)                      fill_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_40: (8)                      fill_border_width: float = 0.5,
SNGT_QHENOMENOLOGY_41: (8)                      stroke_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_42: (8)                      stroke_width: float = 0,
SNGT_QHENOMENOLOGY_43: (8)                      base_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_44: (8)                      isolate: Selector = (),
SNGT_QHENOMENOLOGY_45: (8)                      protect: Selector = (),
SNGT_QHENOMENOLOGY_46: (8)                      use_labelled_svg: bool = False,
SNGT_QHENOMENOLOGY_47: (8)                      **kwargs
SNGT_QHENOMENOLOGY_48: (4)                  ):
SNGT_QHENOMENOLOGY_49: (8)                      self.string = string
SNGT_QHENOMENOLOGY_50: (8)                      self.base_color = base_color or WHITE
SNGT_QHENOMENOLOGY_51: (8)                      self.isolate = isolate
SNGT_QHENOMENOLOGY_52: (8)                      self.protect = protect
SNGT_QHENOMENOLOGY_53: (8)                      self.use_labelled_svg = use_labelled_svg
SNGT_QHENOMENOLOGY_54: (8)                      self.parse()
SNGT_QHENOMENOLOGY_55: (8)                      svg_string = self.get_svg_string()
SNGT_QHENOMENOLOGY_56: (8)                      super().__init__(svg_string=svg_string, **kwargs)
SNGT_QHENOMENOLOGY_57: (8)                      self.set_stroke(stroke_color, stroke_width)
SNGT_QHENOMENOLOGY_58: (8)                      self.set_fill(fill_color,
border_width=fill_border_width)
SNGT_QHENOMENOLOGY_59: (8)                      self.labels = [submob.label for submob in
```

```
self.submobjects]
SNGT_QHENOMENOLOGY_60: (4)                 def get_svg_string(self, is_labelled: bool = False) ->
str:
SNGT_QHENOMENOLOGY_61: (8)                     content = self.get_content(is_labelled or
self.use_labelled_svg)
SNGT_QHENOMENOLOGY_62: (8)                     return self.get_svg_string_by_content(content)
SNGT_QHENOMENOLOGY_63: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_64: (4)                 def get_svg_string_by_content(self, content: str) ->
str:
SNGT_QHENOMENOLOGY_65: (8)                     return ""
SNGT_QHENOMENOLOGY_66: (4)                 def assign_labels_by_color(self, mobjects:
list[VMobject]) -> None:
SNGT_QHENOMENOLOGY_67: (8)                     """
SNGT_QHENOMENOLOGY_68: (8)                     Assuming each mobject in the list `mobjects` has a
fill color
SNGT_QHENOMENOLOGY_69: (8)                     meant to represent a numerical label, this assigns
those
SNGT_QHENOMENOLOGY_70: (8)                     those numerical labels to each mobject as an
attribute
SNGT_QHENOMENOLOGY_71: (8)                     """
SNGT_QHENOMENOLOGY_72: (8)                     labels_count = len(self.labelled_spans)
SNGT_QHENOMENOLOGY_73: (8)                     if labels_count == 1:
SNGT_QHENOMENOLOGY_74: (12)                        for mob in mobjects:
SNGT_QHENOMENOLOGY_75: (16)                            mob.label = 0
SNGT_QHENOMENOLOGY_76: (12)                        return
SNGT_QHENOMENOLOGY_77: (8)                     unrecognizable_colors = []
SNGT_QHENOMENOLOGY_78: (8)                     for mob in mobjects:
SNGT_QHENOMENOLOGY_79: (12)                        label =
hex_to_int(color_to_hex(mob.get_fill_color()))
SNGT_QHENOMENOLOGY_80: (12)                        if label >= labels_count:
SNGT_QHENOMENOLOGY_81: (16)                            unrecognizable_colors.append(label)
SNGT_QHENOMENOLOGY_82: (16)                            label = 0
SNGT_QHENOMENOLOGY_83: (12)                        mob.label = label
SNGT_QHENOMENOLOGY_84: (8)                     if unrecognizable_colors:
SNGT_QHENOMENOLOGY_85: (12)                        log.warning(
SNGT_QHENOMENOLOGY_86: (16)                            "Unrecognizable color labels detected (%s).
" + \
SNGT_QHENOMENOLOGY_87: (16)                            "The result could be unexpected.",
SNGT_QHENOMENOLOGY_88: (16)                            ", ".join(
SNGT_QHENOMENOLOGY_89: (20)                                int_to_hex(color)
SNGT_QHENOMENOLOGY_90: (20)                                for color in unrecognizable_colors
SNGT_QHENOMENOLOGY_91: (16)                            )
SNGT_QHENOMENOLOGY_92: (12)                        )
SNGT_QHENOMENOLOGY_93: (4)                 def mobjects_from_svg_string(self, svg_string: str) ->
list[VMobject]:
SNGT_QHENOMENOLOGY_94: (8)                     submobs =
super().mobjects_from_svg_string(svg_string)
SNGT_QHENOMENOLOGY_95: (8)                     if self.use_labelled_svg:
SNGT_QHENOMENOLOGY_96: (12)                        self.assign_labels_by_color(submobs)
SNGT_QHENOMENOLOGY_97: (12)                        return submobs
SNGT_QHENOMENOLOGY_98: (8)                     unlabelled_submobs = submobs
SNGT_QHENOMENOLOGY_99: (8)                     labelled_content =
self.get_content(is_labelled=True)
SNGT_QHENOMENOLOGY_100: (8)                    labelled_file =
self.get_file_path_by_content(labelled_content)
SNGT_QHENOMENOLOGY_101: (8)                    labelled_submobs =
super().mobjects_from_file(labelled_file)
SNGT_QHENOMENOLOGY_102: (8)                    self.labelled_submobs = labelled_submobs
SNGT_QHENOMENOLOGY_103: (8)                    self.unlabelled_submobs = unlabelled_submobs
SNGT_QHENOMENOLOGY_104: (8)                    self.assign_labels_by_color(labelled_submobs)
SNGT_QHENOMENOLOGY_105: (8)
self.rearrange_submobjects_by_positions(labelled_submobs, unlabelled_submobs)
SNGT_QHENOMENOLOGY_106: (8)                    for usm, lsm in zip(unlabelled_submobs,
labelled_submobs):
SNGT_QHENOMENOLOGY_107: (12)                       usm.label = lsm.label
SNGT_QHENOMENOLOGY_108: (8)                    if len(unlabelled_submobs) !=
len(labelled_submobs):
SNGT_QHENOMENOLOGY_109: (12)                       log.warning(
SNGT_QHENOMENOLOGY_110: (16)                           "Cannot align submobjects of the labelled
```

```
svg " + \
SNGT_QHENOMENOLOGY_111: (16)                            "to the original svg. Skip the labelling
process."
SNGT_QHENOMENOLOGY_112: (12)                        )
SNGT_QHENOMENOLOGY_113: (12)                        for usm in unlabelled_submobs:
SNGT_QHENOMENOLOGY_114: (16)                            usm.label = 0
SNGT_QHENOMENOLOGY_115: (12)                        return unlabelled_submobs
SNGT_QHENOMENOLOGY_116: (8)                     return unlabelled_submobs
SNGT_QHENOMENOLOGY_117: (4)                 def rearrange_submobjects_by_positions(
SNGT_QHENOMENOLOGY_118: (8)                     self, labelled_submobs: list[VMobject],
unlabelled_submobs: list[VMobject],
SNGT_QHENOMENOLOGY_119: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_120: (8)                     """
SNGT_QHENOMENOLOGY_121: (8)                     Rearrange `labeleled_submobjects` so that each
submobject
SNGT_QHENOMENOLOGY_122: (8)                     is labelled by the nearest one of
`unlabelled_submobs`.
SNGT_QHENOMENOLOGY_123: (8)                     The correctness cannot be ensured, since the svg
may
SNGT_QHENOMENOLOGY_124: (8)                     change significantly after inserting color
commands.
SNGT_QHENOMENOLOGY_125: (8)                     """
SNGT_QHENOMENOLOGY_126: (8)                     if len(labelled_submobs) == 0:
SNGT_QHENOMENOLOGY_127: (12)                        return
SNGT_QHENOMENOLOGY_128: (8)                     labelled_svg = VGroup(*labelled_submobs)
SNGT_QHENOMENOLOGY_129: (8)                     labelled_svg.replace(VGroup(*unlabelled_submobs))
SNGT_QHENOMENOLOGY_130: (8)                     distance_matrix = cdist(
SNGT_QHENOMENOLOGY_131: (12)                        [submob.get_center() for submob in
unlabelled_submobs],
SNGT_QHENOMENOLOGY_132: (12)                        [submob.get_center() for submob in
labelled_submobs]
SNGT_QHENOMENOLOGY_133: (8)                     )
SNGT_QHENOMENOLOGY_134: (8)                     _, indices = linear_sum_assignment(distance_matrix)
SNGT_QHENOMENOLOGY_135: (8)                     labelled_submobs[:] = [labelled_submobs[index] for
index in indices]
SNGT_QHENOMENOLOGY_136: (4)                 def find_spans_by_selector(self, selector: Selector) ->
list[Span]:
SNGT_QHENOMENOLOGY_137: (8)                     def find_spans_by_single_selector(sel):
SNGT_QHENOMENOLOGY_138: (12)                        if isinstance(sel, str):
SNGT_QHENOMENOLOGY_139: (16)                            return [
SNGT_QHENOMENOLOGY_140: (20)                                match_obj.span()
SNGT_QHENOMENOLOGY_141: (20)                                for match_obj in
re.finditer(re.escape(sel), self.string)
SNGT_QHENOMENOLOGY_142: (16)                            ]
SNGT_QHENOMENOLOGY_143: (12)                        if isinstance(sel, re.Pattern):
SNGT_QHENOMENOLOGY_144: (16)                            return [
SNGT_QHENOMENOLOGY_145: (20)                                match_obj.span()
SNGT_QHENOMENOLOGY_146: (20)                                for match_obj in
sel.finditer(self.string)
SNGT_QHENOMENOLOGY_147: (16)                            ]
SNGT_QHENOMENOLOGY_148: (12)                        if isinstance(sel, tuple) and len(sel) == 2 and
all(
SNGT_QHENOMENOLOGY_149: (16)                            isinstance(index, int) or index is None
SNGT_QHENOMENOLOGY_150: (16)                            for index in sel
SNGT_QHENOMENOLOGY_151: (12)                        ):
SNGT_QHENOMENOLOGY_152: (16)                            l = len(self.string)
SNGT_QHENOMENOLOGY_153: (16)                            span = tuple(
SNGT_QHENOMENOLOGY_154: (20)                                default_index if index is None else
SNGT_QHENOMENOLOGY_155: (20)                                min(index, l) if index >= 0 else
max(index + l, 0)
SNGT_QHENOMENOLOGY_156: (20)                                for index, default_index in zip(sel,
(0, l))
SNGT_QHENOMENOLOGY_157: (16)                            )
SNGT_QHENOMENOLOGY_158: (16)                            return [span]
SNGT_QHENOMENOLOGY_159: (12)                        return None
SNGT_QHENOMENOLOGY_160: (8)                     result = find_spans_by_single_selector(selector)
SNGT_QHENOMENOLOGY_161: (8)                     if result is None:
SNGT_QHENOMENOLOGY_162: (12)                        result = []
SNGT_QHENOMENOLOGY_163: (12)                        for sel in selector:
```

```
SNGT_QHENOMENOLOGY_164: (16)                            spans = find_spans_by_single_selector(sel)
SNGT_QHENOMENOLOGY_165: (16)                            if spans is None:
SNGT_QHENOMENOLOGY_166: (20)                                raise TypeError(f"Invalid selector:
'{sel}'")
SNGT_QHENOMENOLOGY_167: (16)                            result.extend(spans)
SNGT_QHENOMENOLOGY_168: (8)                     return list(filter(lambda span: span[0] <= span[1],
result))
SNGT_QHENOMENOLOGY_169: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_170: (4)                 def span_contains(span_0: Span, span_1: Span) -> bool:
SNGT_QHENOMENOLOGY_171: (8)                     return span_0[0] <= span_1[0] and span_0[1] >=
span_1[1]
SNGT_QHENOMENOLOGY_172: (4)                 def parse(self) -> None:
SNGT_QHENOMENOLOGY_173: (8)                     def get_substr(span: Span) -> str:
SNGT_QHENOMENOLOGY_174: (12)                        return self.string[slice(*span)]
SNGT_QHENOMENOLOGY_175: (8)                     configured_items = self.get_configured_items()
SNGT_QHENOMENOLOGY_176: (8)                     isolated_spans =
self.find_spans_by_selector(self.isolate)
SNGT_QHENOMENOLOGY_177: (8)                     protected_spans =
self.find_spans_by_selector(self.protect)
SNGT_QHENOMENOLOGY_178: (8)                     command_matches =
self.get_command_matches(self.string)
SNGT_QHENOMENOLOGY_179: (8)                     def get_key(category, i, flag):
SNGT_QHENOMENOLOGY_180: (12)                        def get_span_by_category(category, i):
SNGT_QHENOMENOLOGY_181: (16)                            if category == 0:
SNGT_QHENOMENOLOGY_182: (20)                                return configured_items[i][0]
SNGT_QHENOMENOLOGY_183: (16)                            if category == 1:
SNGT_QHENOMENOLOGY_184: (20)                                return isolated_spans[i]
SNGT_QHENOMENOLOGY_185: (16)                            if category == 2:
SNGT_QHENOMENOLOGY_186: (20)                                return protected_spans[i]
SNGT_QHENOMENOLOGY_187: (16)                            return command_matches[i].span()
SNGT_QHENOMENOLOGY_188: (12)                        index, paired_index =
get_span_by_category(category, i)[::flag]
SNGT_QHENOMENOLOGY_189: (12)                        return (
SNGT_QHENOMENOLOGY_190: (16)                            index,
SNGT_QHENOMENOLOGY_191: (16)                            flag * (2 if index != paired_index else
-1),
SNGT_QHENOMENOLOGY_192: (16)                            -paired_index,
SNGT_QHENOMENOLOGY_193: (16)                            flag * category,
SNGT_QHENOMENOLOGY_194: (16)                            flag * i
SNGT_QHENOMENOLOGY_195: (12)                        )
SNGT_QHENOMENOLOGY_196: (8)                     index_items = sorted([
SNGT_QHENOMENOLOGY_197: (12)                        (category, i, flag)
SNGT_QHENOMENOLOGY_198: (12)                        for category, item_length in enumerate((
SNGT_QHENOMENOLOGY_199: (16)                            len(configured_items),
SNGT_QHENOMENOLOGY_200: (16)                            len(isolated_spans),
SNGT_QHENOMENOLOGY_201: (16)                            len(protected_spans),
SNGT_QHENOMENOLOGY_202: (16)                            len(command_matches)
SNGT_QHENOMENOLOGY_203: (12)                        ))
SNGT_QHENOMENOLOGY_204: (12)                        for i in range(item_length)
SNGT_QHENOMENOLOGY_205: (12)                        for flag in (1, -1)
SNGT_QHENOMENOLOGY_206: (8)                     ], key=lambda t: get_key(*t))
SNGT_QHENOMENOLOGY_207: (8)                     inserted_items = []
SNGT_QHENOMENOLOGY_208: (8)                     labelled_items = []
SNGT_QHENOMENOLOGY_209: (8)                     overlapping_spans = []
SNGT_QHENOMENOLOGY_210: (8)                     level_mismatched_spans = []
SNGT_QHENOMENOLOGY_211: (8)                     label = 1
SNGT_QHENOMENOLOGY_212: (8)                     protect_level = 0
SNGT_QHENOMENOLOGY_213: (8)                     bracket_stack = [0]
SNGT_QHENOMENOLOGY_214: (8)                     bracket_count = 0
SNGT_QHENOMENOLOGY_215: (8)                     open_command_stack = []
SNGT_QHENOMENOLOGY_216: (8)                     open_stack = []
SNGT_QHENOMENOLOGY_217: (8)                     for category, i, flag in index_items:
SNGT_QHENOMENOLOGY_218: (12)                        if category >= 2:
SNGT_QHENOMENOLOGY_219: (16)                            protect_level += flag
SNGT_QHENOMENOLOGY_220: (16)                            if flag == 1 or category == 2:
SNGT_QHENOMENOLOGY_221: (20)                                continue
SNGT_QHENOMENOLOGY_222: (16)                            inserted_items.append((i, 0))
SNGT_QHENOMENOLOGY_223: (16)                            command_match = command_matches[i]
SNGT_QHENOMENOLOGY_224: (16)                            command_flag =
```

```
self.get_command_flag(command_match)
SNGT_QHENOMENOLOGY_225: (16)                            if command_flag == 1:
SNGT_QHENOMENOLOGY_226: (20)                                bracket_count += 1
SNGT_QHENOMENOLOGY_227: (20)                                bracket_stack.append(bracket_count)
SNGT_QHENOMENOLOGY_228: (20)
open_command_stack.append((len(inserted_items), i))
SNGT_QHENOMENOLOGY_229: (20)                                continue
SNGT_QHENOMENOLOGY_230: (16)                            if command_flag == 0:
SNGT_QHENOMENOLOGY_231: (20)                                continue
SNGT_QHENOMENOLOGY_232: (16)                            pos, i_ = open_command_stack.pop()
SNGT_QHENOMENOLOGY_233: (16)                            bracket_stack.pop()
SNGT_QHENOMENOLOGY_234: (16)                            open_command_match = command_matches[i_]
SNGT_QHENOMENOLOGY_235: (16)                            attr_dict =
self.get_attr_dict_from_command_pair(
SNGT_QHENOMENOLOGY_236: (20)                                open_command_match, command_match
SNGT_QHENOMENOLOGY_237: (16)                            )
SNGT_QHENOMENOLOGY_238: (16)                            if attr_dict is None:
SNGT_QHENOMENOLOGY_239: (20)                                continue
SNGT_QHENOMENOLOGY_240: (16)                            span = (open_command_match.end(),
command_match.start())
SNGT_QHENOMENOLOGY_241: (16)                            labelled_items.append((span, attr_dict))
SNGT_QHENOMENOLOGY_242: (16)                            inserted_items.insert(pos, (label, 1))
SNGT_QHENOMENOLOGY_243: (16)                            inserted_items.insert(-1, (label, -1))
SNGT_QHENOMENOLOGY_244: (16)                            label += 1
SNGT_QHENOMENOLOGY_245: (16)                            continue
SNGT_QHENOMENOLOGY_246: (12)                        if flag == 1:
SNGT_QHENOMENOLOGY_247: (16)                            open_stack.append((
SNGT_QHENOMENOLOGY_248: (20)                                len(inserted_items), category, i,
SNGT_QHENOMENOLOGY_249: (20)                                protect_level, bracket_stack.copy()
SNGT_QHENOMENOLOGY_250: (16)                            ))
SNGT_QHENOMENOLOGY_251: (16)                            continue
SNGT_QHENOMENOLOGY_252: (12)                        span, attr_dict = configured_items[i] \
SNGT_QHENOMENOLOGY_253: (16)                            if category == 0 else (isolated_spans[i],
{})
SNGT_QHENOMENOLOGY_254: (12)                        pos, category_, i_, protect_level_,
bracket_stack_ \
SNGT_QHENOMENOLOGY_255: (16)                            = open_stack.pop()
SNGT_QHENOMENOLOGY_256: (12)                        if category_ != category or i_ != i:
SNGT_QHENOMENOLOGY_257: (16)                            overlapping_spans.append(span)
SNGT_QHENOMENOLOGY_258: (16)                            continue
SNGT_QHENOMENOLOGY_259: (12)                        if protect_level_ or protect_level:
SNGT_QHENOMENOLOGY_260: (16)                            continue
SNGT_QHENOMENOLOGY_261: (12)                        if bracket_stack_ != bracket_stack:
SNGT_QHENOMENOLOGY_262: (16)                            level_mismatched_spans.append(span)
SNGT_QHENOMENOLOGY_263: (16)                            continue
SNGT_QHENOMENOLOGY_264: (12)                        labelled_items.append((span, attr_dict))
SNGT_QHENOMENOLOGY_265: (12)                        inserted_items.insert(pos, (label, 1))
SNGT_QHENOMENOLOGY_266: (12)                        inserted_items.append((label, -1))
SNGT_QHENOMENOLOGY_267: (12)                        label += 1
SNGT_QHENOMENOLOGY_268: (8)                     labelled_items.insert(0, ((0, len(self.string)),
{}))
SNGT_QHENOMENOLOGY_269: (8)                     inserted_items.insert(0, (0, 1))
SNGT_QHENOMENOLOGY_270: (8)                     inserted_items.append((0, -1))
SNGT_QHENOMENOLOGY_271: (8)                     if overlapping_spans:
SNGT_QHENOMENOLOGY_272: (12)                        log.warning(
SNGT_QHENOMENOLOGY_273: (16)                            "Partly overlapping substrings detected:
%s",
SNGT_QHENOMENOLOGY_274: (16)                            ", ".join(
SNGT_QHENOMENOLOGY_275: (20)                                f"'{get_substr(span)}'"
SNGT_QHENOMENOLOGY_276: (20)                                for span in overlapping_spans
SNGT_QHENOMENOLOGY_277: (16)                            )
SNGT_QHENOMENOLOGY_278: (12)                        )
SNGT_QHENOMENOLOGY_279: (8)                     if level_mismatched_spans:
SNGT_QHENOMENOLOGY_280: (12)                        log.warning(
SNGT_QHENOMENOLOGY_281: (16)                            "Cannot handle substrings: %s",
SNGT_QHENOMENOLOGY_282: (16)                            ", ".join(
SNGT_QHENOMENOLOGY_283: (20)                                f"'{get_substr(span)}'"
SNGT_QHENOMENOLOGY_284: (20)                                for span in level_mismatched_spans
SNGT_QHENOMENOLOGY_285: (16)                            )
```

```
SNGT_QHENOMENOLOGY_286: (12)                     )
SNGT_QHENOMENOLOGY_287: (8)                  def reconstruct_string(
SNGT_QHENOMENOLOGY_288: (12)                     start_item: tuple[int, int],
SNGT_QHENOMENOLOGY_289: (12)                     end_item: tuple[int, int],
SNGT_QHENOMENOLOGY_290: (12)                     command_replace_func: Callable[[re.Match],
str],
SNGT_QHENOMENOLOGY_291: (12)                     command_insert_func: Callable[[int, int,
dict[str, str]], str]
SNGT_QHENOMENOLOGY_292: (8)                  ) -> str:
SNGT_QHENOMENOLOGY_293: (12)                     def get_edge_item(i: int, flag: int) ->
tuple[Span, str]:
SNGT_QHENOMENOLOGY_294: (16)                         if flag == 0:
SNGT_QHENOMENOLOGY_295: (20)                             match_obj = command_matches[i]
SNGT_QHENOMENOLOGY_296: (20)                             return (
SNGT_QHENOMENOLOGY_297: (24)                                 match_obj.span(),
SNGT_QHENOMENOLOGY_298: (24)                                 command_replace_func(match_obj)
SNGT_QHENOMENOLOGY_299: (20)                             )
SNGT_QHENOMENOLOGY_300: (16)                         span, attr_dict = labelled_items[i]
SNGT_QHENOMENOLOGY_301: (16)                         index = span[flag < 0]
SNGT_QHENOMENOLOGY_302: (16)                         return (
SNGT_QHENOMENOLOGY_303: (20)                             (index, index),
SNGT_QHENOMENOLOGY_304: (20)                             command_insert_func(i, flag, attr_dict)
SNGT_QHENOMENOLOGY_305: (16)                         )
SNGT_QHENOMENOLOGY_306: (12)                     items = [
SNGT_QHENOMENOLOGY_307: (16)                         get_edge_item(i, flag)
SNGT_QHENOMENOLOGY_308: (16)                         for i, flag in inserted_items[slice(
SNGT_QHENOMENOLOGY_309: (20)                             inserted_items.index(start_item),
SNGT_QHENOMENOLOGY_310: (20)                             inserted_items.index(end_item) + 1
SNGT_QHENOMENOLOGY_311: (16)                         )]
SNGT_QHENOMENOLOGY_312: (12)                     ]
SNGT_QHENOMENOLOGY_313: (12)                     pieces = [
SNGT_QHENOMENOLOGY_314: (16)                         get_substr((start, end))
SNGT_QHENOMENOLOGY_315: (16)                         for start, end in zip(
SNGT_QHENOMENOLOGY_316: (20)                             [interval_end for (_, interval_end), _
in items[:-1]],
SNGT_QHENOMENOLOGY_317: (20)                             [interval_start for (interval_start,
_), _ in items[1:]]
SNGT_QHENOMENOLOGY_318: (16)                         )
SNGT_QHENOMENOLOGY_319: (12)                     ]
SNGT_QHENOMENOLOGY_320: (12)                     interval_pieces = [piece for _, piece in
items[1:-1]]
SNGT_QHENOMENOLOGY_321: (12)                     return "".join(it.chain(*zip(pieces,
(*interval_pieces, ""))))
SNGT_QHENOMENOLOGY_322: (8)                  self.labelled_spans = [span for span, _ in
labelled_items]
SNGT_QHENOMENOLOGY_323: (8)                  self.reconstruct_string = reconstruct_string
SNGT_QHENOMENOLOGY_324: (4)              def get_content(self, is_labelled: bool) -> str:
SNGT_QHENOMENOLOGY_325: (8)                  content = self.reconstruct_string(
SNGT_QHENOMENOLOGY_326: (12)                     (0, 1), (0, -1),
SNGT_QHENOMENOLOGY_327: (12)                     self.replace_for_content,
SNGT_QHENOMENOLOGY_328: (12)                     lambda label, flag, attr_dict:
self.get_command_string(
SNGT_QHENOMENOLOGY_329: (16)                         attr_dict,
SNGT_QHENOMENOLOGY_330: (16)                         is_end=flag < 0,
SNGT_QHENOMENOLOGY_331: (16)                         label_hex=int_to_hex(label) if is_labelled
else None
SNGT_QHENOMENOLOGY_332: (12)                     )
SNGT_QHENOMENOLOGY_333: (8)                  )
SNGT_QHENOMENOLOGY_334: (8)                  prefix, suffix =
self.get_content_prefix_and_suffix(
SNGT_QHENOMENOLOGY_335: (12)                     is_labelled=is_labelled
SNGT_QHENOMENOLOGY_336: (8)                  )
SNGT_QHENOMENOLOGY_337: (8)                  return "".join((prefix, content, suffix))
SNGT_QHENOMENOLOGY_338: (4)              @staticmethod
SNGT_QHENOMENOLOGY_339: (4)              @abstractmethod
SNGT_QHENOMENOLOGY_340: (4)              def get_command_matches(string: str) -> list[re.Match]:
SNGT_QHENOMENOLOGY_341: (8)                  return []
SNGT_QHENOMENOLOGY_342: (4)              @staticmethod
SNGT_QHENOMENOLOGY_343: (4)              @abstractmethod
```

```
SNGT_QHENOMENOLOGY_344: (4)                 def get_command_flag(match_obj: re.Match) -> int:
SNGT_QHENOMENOLOGY_345: (8)                     return 0
SNGT_QHENOMENOLOGY_346: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_347: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_348: (4)                 def replace_for_content(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_349: (8)                     return ""
SNGT_QHENOMENOLOGY_350: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_351: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_352: (4)                 def replace_for_matching(match_obj: re.Match) -> str:
SNGT_QHENOMENOLOGY_353: (8)                     return ""
SNGT_QHENOMENOLOGY_354: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_355: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_356: (4)                 def get_attr_dict_from_command_pair(
SNGT_QHENOMENOLOGY_357: (8)                     open_command: re.Match, close_command: re.Match,
SNGT_QHENOMENOLOGY_358: (4)                 ) -> dict[str, str] | None:
SNGT_QHENOMENOLOGY_359: (8)                     return None
SNGT_QHENOMENOLOGY_360: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_361: (4)                 def get_configured_items(self) -> list[tuple[Span,
dict[str, str]]]:
SNGT_QHENOMENOLOGY_362: (8)                     return []
SNGT_QHENOMENOLOGY_363: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_364: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_365: (4)                 def get_command_string(
SNGT_QHENOMENOLOGY_366: (8)                     attr_dict: dict[str, str], is_end: bool, label_hex:
str | None
SNGT_QHENOMENOLOGY_367: (4)                 ) -> str:
SNGT_QHENOMENOLOGY_368: (8)                     return ""
SNGT_QHENOMENOLOGY_369: (4)                 @abstractmethod
SNGT_QHENOMENOLOGY_370: (4)                 def get_content_prefix_and_suffix(
SNGT_QHENOMENOLOGY_371: (8)                     self, is_labelled: bool
SNGT_QHENOMENOLOGY_372: (4)                 ) -> tuple[str, str]:
SNGT_QHENOMENOLOGY_373: (8)                     return "", ""
SNGT_QHENOMENOLOGY_374: (4)                 def get_submob_indices_list_by_span(
SNGT_QHENOMENOLOGY_375: (8)                     self, arbitrary_span: Span
SNGT_QHENOMENOLOGY_376: (4)                 ) -> list[int]:
SNGT_QHENOMENOLOGY_377: (8)                     return [
SNGT_QHENOMENOLOGY_378: (12)                        submob_index
SNGT_QHENOMENOLOGY_379: (12)                        for submob_index, label in
enumerate(self.labels)
SNGT_QHENOMENOLOGY_380: (12)                        if self.span_contains(arbitrary_span,
self.labelled_spans[label])
SNGT_QHENOMENOLOGY_381: (8)                     ]
SNGT_QHENOMENOLOGY_382: (4)                 def get_specified_part_items(self) -> list[tuple[str,
list[int]]]:
SNGT_QHENOMENOLOGY_383: (8)                     return [
SNGT_QHENOMENOLOGY_384: (12)                        (
SNGT_QHENOMENOLOGY_385: (16)                            self.string[slice(*span)],
SNGT_QHENOMENOLOGY_386: (16)                            self.get_submob_indices_list_by_span(span)
SNGT_QHENOMENOLOGY_387: (12)                        )
SNGT_QHENOMENOLOGY_388: (12)                        for span in self.labelled_spans[1:]
SNGT_QHENOMENOLOGY_389: (8)                     ]
SNGT_QHENOMENOLOGY_390: (4)                 def get_specified_substrings(self) -> list[str]:
SNGT_QHENOMENOLOGY_391: (8)                     substrs = [
SNGT_QHENOMENOLOGY_392: (12)                        self.string[slice(*span)]
SNGT_QHENOMENOLOGY_393: (12)                        for span in self.labelled_spans[1:]
SNGT_QHENOMENOLOGY_394: (8)                     ]
SNGT_QHENOMENOLOGY_395: (8)                     return list(dict.fromkeys(substrs).keys())
SNGT_QHENOMENOLOGY_396: (4)                 def get_group_part_items(self) -> list[tuple[str,
list[int]]]:
SNGT_QHENOMENOLOGY_397: (8)                     if not self.labels:
SNGT_QHENOMENOLOGY_398: (12)                        return []
SNGT_QHENOMENOLOGY_399: (8)                     def get_neighbouring_pairs(vals):
SNGT_QHENOMENOLOGY_400: (12)                        return list(zip(vals[:-1], vals[1:]))
SNGT_QHENOMENOLOGY_401: (8)                     range_lens, group_labels = zip(*(
SNGT_QHENOMENOLOGY_402: (12)                        (len(list(grouper)), val)
SNGT_QHENOMENOLOGY_403: (12)                        for val, grouper in it.groupby(self.labels)
SNGT_QHENOMENOLOGY_404: (8)                     ))
SNGT_QHENOMENOLOGY_405: (8)                     submob_indices_lists = [
SNGT_QHENOMENOLOGY_406: (12)                        list(range(*submob_range))
```

```
SNGT_QHENOMENOLOGY_407: (12)                     for submob_range in get_neighbouring_pairs(
SNGT_QHENOMENOLOGY_408: (16)                         [0, *it.accumulate(range_lens)]
SNGT_QHENOMENOLOGY_409: (12)                     )
SNGT_QHENOMENOLOGY_410: (8)                 ]
SNGT_QHENOMENOLOGY_411: (8)                 labelled_spans = self.labelled_spans
SNGT_QHENOMENOLOGY_412: (8)                 start_items = [
SNGT_QHENOMENOLOGY_413: (12)                     (group_labels[0], 1),
SNGT_QHENOMENOLOGY_414: (12)                     *(
SNGT_QHENOMENOLOGY_415: (16)                         (curr_label, 1)
SNGT_QHENOMENOLOGY_416: (16)                         if self.span_contains(
SNGT_QHENOMENOLOGY_417: (20)                             labelled_spans[prev_label],
labelled_spans[curr_label]
SNGT_QHENOMENOLOGY_418: (16)                         )
SNGT_QHENOMENOLOGY_419: (16)                         else (prev_label, -1)
SNGT_QHENOMENOLOGY_420: (16)                         for prev_label, curr_label in
get_neighbouring_pairs(
SNGT_QHENOMENOLOGY_421: (20)                             group_labels
SNGT_QHENOMENOLOGY_422: (16)                         )
SNGT_QHENOMENOLOGY_423: (12)                     )
SNGT_QHENOMENOLOGY_424: (8)                 ]
SNGT_QHENOMENOLOGY_425: (8)                 end_items = [
SNGT_QHENOMENOLOGY_426: (12)                     *(
SNGT_QHENOMENOLOGY_427: (16)                         (curr_label, -1)
SNGT_QHENOMENOLOGY_428: (16)                         if self.span_contains(
SNGT_QHENOMENOLOGY_429: (20)                             labelled_spans[next_label],
labelled_spans[curr_label]
SNGT_QHENOMENOLOGY_430: (16)                         )
SNGT_QHENOMENOLOGY_431: (16)                         else (next_label, 1)
SNGT_QHENOMENOLOGY_432: (16)                         for curr_label, next_label in
get_neighbouring_pairs(
SNGT_QHENOMENOLOGY_433: (20)                             group_labels
SNGT_QHENOMENOLOGY_434: (16)                         )
SNGT_QHENOMENOLOGY_435: (12)                     ),
SNGT_QHENOMENOLOGY_436: (12)                     (group_labels[-1], -1)
SNGT_QHENOMENOLOGY_437: (8)                 ]
SNGT_QHENOMENOLOGY_438: (8)                 group_substrs = [
SNGT_QHENOMENOLOGY_439: (12)                     re.sub(r"\s+", "", self.reconstruct_string(
SNGT_QHENOMENOLOGY_440: (16)                         start_item, end_item,
SNGT_QHENOMENOLOGY_441: (16)                         self.replace_for_matching,
SNGT_QHENOMENOLOGY_442: (16)                         lambda label, flag, attr_dict: ""
SNGT_QHENOMENOLOGY_443: (12)                     ))
SNGT_QHENOMENOLOGY_444: (12)                     for start_item, end_item in zip(start_items,
end_items)
SNGT_QHENOMENOLOGY_445: (8)                 ]
SNGT_QHENOMENOLOGY_446: (8)                 return list(zip(group_substrs,
submob_indices_lists))
SNGT_QHENOMENOLOGY_447: (4)             def get_submob_indices_lists_by_selector(
SNGT_QHENOMENOLOGY_448: (8)                 self, selector: Selector
SNGT_QHENOMENOLOGY_449: (4)             ) -> list[list[int]]:
SNGT_QHENOMENOLOGY_450: (8)                 return list(filter(
SNGT_QHENOMENOLOGY_451: (12)                     lambda indices_list: indices_list,
SNGT_QHENOMENOLOGY_452: (12)                     [
SNGT_QHENOMENOLOGY_453: (16)                         self.get_submob_indices_list_by_span(span)
SNGT_QHENOMENOLOGY_454: (16)                         for span in
self.find_spans_by_selector(selector)
SNGT_QHENOMENOLOGY_455: (12)                     ]
SNGT_QHENOMENOLOGY_456: (8)                 ))
SNGT_QHENOMENOLOGY_457: (4)             def build_parts_from_indices_lists(
SNGT_QHENOMENOLOGY_458: (8)                 self, indices_lists: list[list[int]]
SNGT_QHENOMENOLOGY_459: (4)             ) -> VGroup:
SNGT_QHENOMENOLOGY_460: (8)                 return VGroup(*(
SNGT_QHENOMENOLOGY_461: (12)                     VGroup(*(
SNGT_QHENOMENOLOGY_462: (16)                         self.submobjects[submob_index]
SNGT_QHENOMENOLOGY_463: (16)                         for submob_index in indices_list
SNGT_QHENOMENOLOGY_464: (12)                     ))
SNGT_QHENOMENOLOGY_465: (12)                     for indices_list in indices_lists
SNGT_QHENOMENOLOGY_466: (8)                 ))
SNGT_QHENOMENOLOGY_467: (4)             def build_groups(self) -> VGroup:
SNGT_QHENOMENOLOGY_468: (8)                 return self.build_parts_from_indices_lists([
```

```
SNGT_QHENOMENOLOGY_469: (12)                         indices_list
SNGT_QHENOMENOLOGY_470: (12)                         for _, indices_list in
self.get_group_part_items()
SNGT_QHENOMENOLOGY_471: (8)                     ])
SNGT_QHENOMENOLOGY_472: (4)                 def select_parts(self, selector: Selector) -> VGroup:
SNGT_QHENOMENOLOGY_473: (8)                     specified_substrings =
self.get_specified_substrings()
SNGT_QHENOMENOLOGY_474: (8)                     if isinstance(selector, (str, re.Pattern)) and
selector not in specified_substrings:
SNGT_QHENOMENOLOGY_475: (12)                         return
self.select_unisolated_substring(selector)
SNGT_QHENOMENOLOGY_476: (8)                     indices_list =
self.get_submob_indices_lists_by_selector(selector)
SNGT_QHENOMENOLOGY_477: (8)                     return
self.build_parts_from_indices_lists(indices_list)
SNGT_QHENOMENOLOGY_478: (4)                 def __getitem__(self, value: int | slice | Selector) ->
VMobject:
SNGT_QHENOMENOLOGY_479: (8)                     if isinstance(value, (int, slice)):
SNGT_QHENOMENOLOGY_480: (12)                         return super().__getitem__(value)
SNGT_QHENOMENOLOGY_481: (8)                     return self.select_parts(value)
SNGT_QHENOMENOLOGY_482: (4)                 def select_part(self, selector: Selector, index: int =
0) -> VMobject:
SNGT_QHENOMENOLOGY_483: (8)                     return self.select_parts(selector)[index]
SNGT_QHENOMENOLOGY_484: (4)                 def substr_to_path_count(self, substr: str) -> int:
SNGT_QHENOMENOLOGY_485: (8)                     return len(re.sub(r"\s", "", substr))
SNGT_QHENOMENOLOGY_486: (4)                 def get_symbol_substrings(self):
SNGT_QHENOMENOLOGY_487: (8)                     return list(re.sub(r"\s", "", self.string))
SNGT_QHENOMENOLOGY_488: (4)                 def select_unisolated_substring(self, pattern: str |
re.Pattern) -> VGroup:
SNGT_QHENOMENOLOGY_489: (8)                     if isinstance(pattern, str):
SNGT_QHENOMENOLOGY_490: (12)                         pattern = re.compile(re.escape(pattern))
SNGT_QHENOMENOLOGY_491: (8)                     result = []
SNGT_QHENOMENOLOGY_492: (8)                     for match in re.finditer(pattern, self.string):
SNGT_QHENOMENOLOGY_493: (12)                         index = match.start()
SNGT_QHENOMENOLOGY_494: (12)                         start =
self.substr_to_path_count(self.string[:index])
SNGT_QHENOMENOLOGY_495: (12)                         substr = match.group()
SNGT_QHENOMENOLOGY_496: (12)                         end = start + self.substr_to_path_count(substr)
SNGT_QHENOMENOLOGY_497: (12)                         result.append(self[start:end])
SNGT_QHENOMENOLOGY_498: (8)                     return VGroup(*result)
SNGT_QHENOMENOLOGY_499: (4)                 def set_parts_color(self, selector: Selector, color:
ManimColor):
SNGT_QHENOMENOLOGY_500: (8)                     self.select_parts(selector).set_color(color)
SNGT_QHENOMENOLOGY_501: (8)                     return self
SNGT_QHENOMENOLOGY_502: (4)                 def set_parts_color_by_dict(self, color_map:
dict[Selector, ManimColor]):
SNGT_QHENOMENOLOGY_503: (8)                     for selector, color in color_map.items():
SNGT_QHENOMENOLOGY_504: (12)                         self.set_parts_color(selector, color)
SNGT_QHENOMENOLOGY_505: (8)                     return self
SNGT_QHENOMENOLOGY_506: (4)                 def get_string(self) -> str:
SNGT_QHENOMENOLOGY_507: (8)                     return self.string
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 62 - old_tex_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from functools import reduce
SNGT_QHENOMENOLOGY_3: (0)             import operator as op
SNGT_QHENOMENOLOGY_4: (0)             import re
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.constants import BLACK, WHITE
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.mobject.svg.svg_mobject import SVGMobject
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.utils.tex_file_writing import latex_to_svg
SNGT_QHENOMENOLOGY_9: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_10: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_11: (4)                from typing import Iterable, List, Dict
SNGT_QHENOMENOLOGY_12: (4)                from manimlib.typing import ManimColor
```

```
SNGT_QHENOMENOLOGY_13: (0)             SCALE_FACTOR_PER_FONT_POINT = 0.001
SNGT_QHENOMENOLOGY_14: (0)             class SingleStringTex(SVGMobject):
SNGT_QHENOMENOLOGY_15: (4)                 height: float | None = None
SNGT_QHENOMENOLOGY_16: (4)                 def __init__(
SNGT_QHENOMENOLOGY_17: (8)                     self,
SNGT_QHENOMENOLOGY_18: (8)                     tex_string: str,
SNGT_QHENOMENOLOGY_19: (8)                     height: float | None = None,
SNGT_QHENOMENOLOGY_20: (8)                     fill_color: ManimColor = WHITE,
SNGT_QHENOMENOLOGY_21: (8)                     fill_opacity: float = 1.0,
SNGT_QHENOMENOLOGY_22: (8)                     stroke_width: float = 0,
SNGT_QHENOMENOLOGY_23: (8)                     svg_default: dict = dict(fill_color=WHITE),
SNGT_QHENOMENOLOGY_24: (8)                     path_string_config: dict = dict(),
SNGT_QHENOMENOLOGY_25: (8)                     font_size: int = 48,
SNGT_QHENOMENOLOGY_26: (8)                     alignment: str = R"\centering",
SNGT_QHENOMENOLOGY_27: (8)                     math_mode: bool = True,
SNGT_QHENOMENOLOGY_28: (8)                     organize_left_to_right: bool = False,
SNGT_QHENOMENOLOGY_29: (8)                     template: str = "",
SNGT_QHENOMENOLOGY_30: (8)                     additional_preamble: str = "",
SNGT_QHENOMENOLOGY_31: (8)                     **kwargs
SNGT_QHENOMENOLOGY_32: (4)                 ):
SNGT_QHENOMENOLOGY_33: (8)                     self.tex_string = tex_string
SNGT_QHENOMENOLOGY_34: (8)                     self.svg_default = dict(svg_default)
SNGT_QHENOMENOLOGY_35: (8)                     self.path_string_config = dict(path_string_config)
SNGT_QHENOMENOLOGY_36: (8)                     self.font_size = font_size
SNGT_QHENOMENOLOGY_37: (8)                     self.alignment = alignment
SNGT_QHENOMENOLOGY_38: (8)                     self.math_mode = math_mode
SNGT_QHENOMENOLOGY_39: (8)                     self.organize_left_to_right =
organize_left_to_right
SNGT_QHENOMENOLOGY_40: (8)                     self.template = template
SNGT_QHENOMENOLOGY_41: (8)                     self.additional_preamble = additional_preamble
SNGT_QHENOMENOLOGY_42: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_43: (12)                        height=height,
SNGT_QHENOMENOLOGY_44: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_45: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_46: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_47: (12)                        path_string_config=path_string_config,
SNGT_QHENOMENOLOGY_48: (12)                        **kwargs
SNGT_QHENOMENOLOGY_49: (8)                     )
SNGT_QHENOMENOLOGY_50: (8)                     if self.height is None:
SNGT_QHENOMENOLOGY_51: (12)                        self.scale(SCALE_FACTOR_PER_FONT_POINT *
self.font_size)
SNGT_QHENOMENOLOGY_52: (8)                     if self.organize_left_to_right:
SNGT_QHENOMENOLOGY_53: (12)                        self.organize_submobjects_left_to_right()
SNGT_QHENOMENOLOGY_54: (4)                 @property
SNGT_QHENOMENOLOGY_55: (4)                 def hash_seed(self) -> tuple:
SNGT_QHENOMENOLOGY_56: (8)                     return (
SNGT_QHENOMENOLOGY_57: (12)                        self.__class__.__name__,
SNGT_QHENOMENOLOGY_58: (12)                        self.svg_default,
SNGT_QHENOMENOLOGY_59: (12)                        self.path_string_config,
SNGT_QHENOMENOLOGY_60: (12)                        self.tex_string,
SNGT_QHENOMENOLOGY_61: (12)                        self.alignment,
SNGT_QHENOMENOLOGY_62: (12)                        self.math_mode,
SNGT_QHENOMENOLOGY_63: (12)                        self.template,
SNGT_QHENOMENOLOGY_64: (12)                        self.additional_preamble
SNGT_QHENOMENOLOGY_65: (8)                     )
SNGT_QHENOMENOLOGY_66: (4)                 def get_svg_string_by_content(self, content: str) ->
str:
SNGT_QHENOMENOLOGY_67: (8)                     return latex_to_svg(content, self.template,
self.additional_preamble)
SNGT_QHENOMENOLOGY_68: (4)                 def get_tex_file_body(self, tex_string: str) -> str:
SNGT_QHENOMENOLOGY_69: (8)                     new_tex = self.get_modified_expression(tex_string)
SNGT_QHENOMENOLOGY_70: (8)                     if self.math_mode:
SNGT_QHENOMENOLOGY_71: (12)                        new_tex = "\\begin{align*}\n" + new_tex +
"\n\\end{align*}"
SNGT_QHENOMENOLOGY_72: (8)                     return self.alignment + "\n" + new_tex
SNGT_QHENOMENOLOGY_73: (4)                 def get_modified_expression(self, tex_string: str) ->
str:
SNGT_QHENOMENOLOGY_74: (8)                     return
self.modify_special_strings(tex_string.strip())
```

```
SNGT_QHENOMENOLOGY_75: (4)                 def modify_special_strings(self, tex: str) -> str:
SNGT_QHENOMENOLOGY_76: (8)                     tex = tex.strip()
SNGT_QHENOMENOLOGY_77: (8)                     should_add_filler = reduce(op.or_, [
SNGT_QHENOMENOLOGY_78: (12)                        tex == "\\over",
SNGT_QHENOMENOLOGY_79: (12)                        tex == "\\overline",
SNGT_QHENOMENOLOGY_80: (12)                        tex == "\\sqrt",
SNGT_QHENOMENOLOGY_81: (12)                        tex == "\\sqrt{",
SNGT_QHENOMENOLOGY_82: (12)                        tex.endswith("_"),
SNGT_QHENOMENOLOGY_83: (12)                        tex.endswith("^"),
SNGT_QHENOMENOLOGY_84: (12)                        tex.endswith("dot"),
SNGT_QHENOMENOLOGY_85: (8)                     ])
SNGT_QHENOMENOLOGY_86: (8)                     if should_add_filler:
SNGT_QHENOMENOLOGY_87: (12)                        filler = "{\\quad}"
SNGT_QHENOMENOLOGY_88: (12)                        tex += filler
SNGT_QHENOMENOLOGY_89: (8)                     should_add_double_filler = reduce(op.or_, [
SNGT_QHENOMENOLOGY_90: (12)                        tex == "\\overset",
SNGT_QHENOMENOLOGY_91: (8)                     ])
SNGT_QHENOMENOLOGY_92: (8)                     if should_add_double_filler:
SNGT_QHENOMENOLOGY_93: (12)                        filler = "{\\quad}{\\quad}"
SNGT_QHENOMENOLOGY_94: (12)                        tex += filler
SNGT_QHENOMENOLOGY_95: (8)                     if tex == "\\substack":
SNGT_QHENOMENOLOGY_96: (12)                        tex = "\\quad"
SNGT_QHENOMENOLOGY_97: (8)                     if tex == "":
SNGT_QHENOMENOLOGY_98: (12)                        tex = "\\quad"
SNGT_QHENOMENOLOGY_99: (8)                     if tex.startswith("\\\\"):
SNGT_QHENOMENOLOGY_100: (12)                       tex = tex.replace("\\\\", "\\quad\\\\")
SNGT_QHENOMENOLOGY_101: (8)                    tex = self.balance_braces(tex)
SNGT_QHENOMENOLOGY_102: (8)                    num_lefts, num_rights = [
SNGT_QHENOMENOLOGY_103: (12)                       len([
SNGT_QHENOMENOLOGY_104: (16)                           s for s in tex.split(substr)[1:]
SNGT_QHENOMENOLOGY_105: (16)                           if s and s[0] in "(){}[]|.\\"
SNGT_QHENOMENOLOGY_106: (12)                       ])
SNGT_QHENOMENOLOGY_107: (12)                       for substr in ("\\left", "\\right")
SNGT_QHENOMENOLOGY_108: (8)                    ]
SNGT_QHENOMENOLOGY_109: (8)                    if num_lefts != num_rights:
SNGT_QHENOMENOLOGY_110: (12)                       tex = tex.replace("\\left", "\\big")
SNGT_QHENOMENOLOGY_111: (12)                       tex = tex.replace("\\right", "\\big")
SNGT_QHENOMENOLOGY_112: (8)                    for context in ["array"]:
SNGT_QHENOMENOLOGY_113: (12)                       begin_in = ("\\begin{%s}" % context) in tex
SNGT_QHENOMENOLOGY_114: (12)                       end_in = ("\\end{%s}" % context) in tex
SNGT_QHENOMENOLOGY_115: (12)                       if begin_in ^ end_in:
SNGT_QHENOMENOLOGY_116: (16)                           tex = ""
SNGT_QHENOMENOLOGY_117: (8)                    return tex
SNGT_QHENOMENOLOGY_118: (4)                def balance_braces(self, tex: str) -> str:
SNGT_QHENOMENOLOGY_119: (8)                    """
SNGT_QHENOMENOLOGY_120: (8)                    Makes Tex resiliant to unmatched braces
SNGT_QHENOMENOLOGY_121: (8)                    """
SNGT_QHENOMENOLOGY_122: (8)                    num_unclosed_brackets = 0
SNGT_QHENOMENOLOGY_123: (8)                    for i in range(len(tex)):
SNGT_QHENOMENOLOGY_124: (12)                       if i > 0 and tex[i - 1] == "\\":
SNGT_QHENOMENOLOGY_125: (16)                           continue
SNGT_QHENOMENOLOGY_126: (12)                       char = tex[i]
SNGT_QHENOMENOLOGY_127: (12)                       if char == "{":
SNGT_QHENOMENOLOGY_128: (16)                           num_unclosed_brackets += 1
SNGT_QHENOMENOLOGY_129: (12)                       elif char == "}":
SNGT_QHENOMENOLOGY_130: (16)                           if num_unclosed_brackets == 0:
SNGT_QHENOMENOLOGY_131: (20)                               tex = "{" + tex
SNGT_QHENOMENOLOGY_132: (16)                           else:
SNGT_QHENOMENOLOGY_133: (20)                               num_unclosed_brackets -= 1
SNGT_QHENOMENOLOGY_134: (8)                    tex += num_unclosed_brackets * "}"
SNGT_QHENOMENOLOGY_135: (8)                    return tex
SNGT_QHENOMENOLOGY_136: (4)                def get_tex(self) -> str:
SNGT_QHENOMENOLOGY_137: (8)                    return self.tex_string
SNGT_QHENOMENOLOGY_138: (4)                def organize_submobjects_left_to_right(self):
SNGT_QHENOMENOLOGY_139: (8)                    self.sort(lambda p: p[0])
SNGT_QHENOMENOLOGY_140: (8)                    return self
SNGT_QHENOMENOLOGY_141: (0)            class OldTex(SingleStringTex):
SNGT_QHENOMENOLOGY_142: (4)                def __init__(
SNGT_QHENOMENOLOGY_143: (8)                    self,
```

```
SNGT_QHENOMENOLOGY_144: (8)                     *tex_strings: str,
SNGT_QHENOMENOLOGY_145: (8)                     arg_separator: str = "",
SNGT_QHENOMENOLOGY_146: (8)                     isolate: List[str] = [],
SNGT_QHENOMENOLOGY_147: (8)                     tex_to_color_map: Dict[str, ManimColor] = {},
SNGT_QHENOMENOLOGY_148: (8)                     **kwargs
SNGT_QHENOMENOLOGY_149: (4)                 ):
SNGT_QHENOMENOLOGY_150: (8)                     self.tex_strings = self.break_up_tex_strings(
SNGT_QHENOMENOLOGY_151: (12)                        tex_strings,
SNGT_QHENOMENOLOGY_152: (12)                        substrings_to_isolate=[*isolate,
*tex_to_color_map.keys()]
SNGT_QHENOMENOLOGY_153: (8)                     )
SNGT_QHENOMENOLOGY_154: (8)                     full_string = arg_separator.join(self.tex_strings)
SNGT_QHENOMENOLOGY_155: (8)                     super().__init__(full_string, **kwargs)
SNGT_QHENOMENOLOGY_156: (8)                     self.break_up_by_substrings(self.tex_strings)
SNGT_QHENOMENOLOGY_157: (8)
self.set_color_by_tex_to_color_map(tex_to_color_map)
SNGT_QHENOMENOLOGY_158: (8)                     if self.organize_left_to_right:
SNGT_QHENOMENOLOGY_159: (12)                        self.organize_submobjects_left_to_right()
SNGT_QHENOMENOLOGY_160: (4)                 def break_up_tex_strings(self, tex_strings:
Iterable[str], substrings_to_isolate: List[str] = []) -> Iterable[str]:
SNGT_QHENOMENOLOGY_161: (8)                     if len(substrings_to_isolate) == 0:
SNGT_QHENOMENOLOGY_162: (12)                        return tex_strings
SNGT_QHENOMENOLOGY_163: (8)                     patterns = (
SNGT_QHENOMENOLOGY_164: (12)                        "({})".format(re.escape(ss))
SNGT_QHENOMENOLOGY_165: (12)                        for ss in substrings_to_isolate
SNGT_QHENOMENOLOGY_166: (8)                     )
SNGT_QHENOMENOLOGY_167: (8)                     pattern = "|".join(patterns)
SNGT_QHENOMENOLOGY_168: (8)                     pieces = []
SNGT_QHENOMENOLOGY_169: (8)                     for s in tex_strings:
SNGT_QHENOMENOLOGY_170: (12)                        if pattern:
SNGT_QHENOMENOLOGY_171: (16)                            pieces.extend(re.split(pattern, s))
SNGT_QHENOMENOLOGY_172: (12)                        else:
SNGT_QHENOMENOLOGY_173: (16)                            pieces.append(s)
SNGT_QHENOMENOLOGY_174: (8)                     return list(filter(lambda s: s, pieces))
SNGT_QHENOMENOLOGY_175: (4)                 def break_up_by_substrings(self, tex_strings:
Iterable[str]):
SNGT_QHENOMENOLOGY_176: (8)                     """
SNGT_QHENOMENOLOGY_177: (8)                     Reorganize existing submojects one layer
SNGT_QHENOMENOLOGY_178: (8)                     deeper based on the structure of tex_strings (as a
list
SNGT_QHENOMENOLOGY_179: (8)                     of tex_strings)
SNGT_QHENOMENOLOGY_180: (8)                     """
SNGT_QHENOMENOLOGY_181: (8)                     if len(list(tex_strings)) == 1:
SNGT_QHENOMENOLOGY_182: (12)                        submob = self.copy()
SNGT_QHENOMENOLOGY_183: (12)                        self.set_submobjects([submob])
SNGT_QHENOMENOLOGY_184: (12)                        return self
SNGT_QHENOMENOLOGY_185: (8)                     new_submobjects = []
SNGT_QHENOMENOLOGY_186: (8)                     curr_index = 0
SNGT_QHENOMENOLOGY_187: (8)                     for tex_string in tex_strings:
SNGT_QHENOMENOLOGY_188: (12)                        tex_string = tex_string.strip()
SNGT_QHENOMENOLOGY_189: (12)                        if len(tex_string) == 0:
SNGT_QHENOMENOLOGY_190: (16)                            continue
SNGT_QHENOMENOLOGY_191: (12)                        sub_tex_mob = SingleStringTex(tex_string,
math_mode=self.math_mode)
SNGT_QHENOMENOLOGY_192: (12)                        num_submobs = len(sub_tex_mob)
SNGT_QHENOMENOLOGY_193: (12)                        if num_submobs == 0:
SNGT_QHENOMENOLOGY_194: (16)                            continue
SNGT_QHENOMENOLOGY_195: (12)                        new_index = curr_index + num_submobs
SNGT_QHENOMENOLOGY_196: (12)
sub_tex_mob.set_submobjects(self.submobjects[curr_index:new_index])
SNGT_QHENOMENOLOGY_197: (12)                        new_submobjects.append(sub_tex_mob)
SNGT_QHENOMENOLOGY_198: (12)                        curr_index = new_index
SNGT_QHENOMENOLOGY_199: (8)                     self.set_submobjects(new_submobjects)
SNGT_QHENOMENOLOGY_200: (8)                     return self
SNGT_QHENOMENOLOGY_201: (4)                 def get_parts_by_tex(
SNGT_QHENOMENOLOGY_202: (8)                     self,
SNGT_QHENOMENOLOGY_203: (8)                     tex: str,
SNGT_QHENOMENOLOGY_204: (8)                     substring: bool = True,
SNGT_QHENOMENOLOGY_205: (8)                     case_sensitive: bool = True
```

```
SNGT_QHENOMENOLOGY_206: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_207: (8)                     def test(tex1, tex2):
SNGT_QHENOMENOLOGY_208: (12)                        if not case_sensitive:
SNGT_QHENOMENOLOGY_209: (16)                            tex1 = tex1.lower()
SNGT_QHENOMENOLOGY_210: (16)                            tex2 = tex2.lower()
SNGT_QHENOMENOLOGY_211: (12)                        if substring:
SNGT_QHENOMENOLOGY_212: (16)                            return tex1 in tex2
SNGT_QHENOMENOLOGY_213: (12)                        else:
SNGT_QHENOMENOLOGY_214: (16)                            return tex1 == tex2
SNGT_QHENOMENOLOGY_215: (8)                     return VGroup(*filter(
SNGT_QHENOMENOLOGY_216: (12)                        lambda m: isinstance(m, SingleStringTex) and
test(tex, m.get_tex()),
SNGT_QHENOMENOLOGY_217: (12)                        self.submobjects
SNGT_QHENOMENOLOGY_218: (8)                     ))
SNGT_QHENOMENOLOGY_219: (4)                 def get_part_by_tex(self, tex: str, **kwargs) ->
SingleStringTex | None:
SNGT_QHENOMENOLOGY_220: (8)                     all_parts = self.get_parts_by_tex(tex, **kwargs)
SNGT_QHENOMENOLOGY_221: (8)                     return all_parts[0] if all_parts else None
SNGT_QHENOMENOLOGY_222: (4)                 def set_color_by_tex(self, tex: str, color: ManimColor,
**kwargs):
SNGT_QHENOMENOLOGY_223: (8)                     self.get_parts_by_tex(tex,
**kwargs).set_color(color)
SNGT_QHENOMENOLOGY_224: (8)                     return self
SNGT_QHENOMENOLOGY_225: (4)                 def set_color_by_tex_to_color_map(
SNGT_QHENOMENOLOGY_226: (8)                     self,
SNGT_QHENOMENOLOGY_227: (8)                     tex_to_color_map: dict[str, ManimColor],
SNGT_QHENOMENOLOGY_228: (8)                     **kwargs
SNGT_QHENOMENOLOGY_229: (4)                 ):
SNGT_QHENOMENOLOGY_230: (8)                     for tex, color in list(tex_to_color_map.items()):
SNGT_QHENOMENOLOGY_231: (12)                        self.set_color_by_tex(tex, color, **kwargs)
SNGT_QHENOMENOLOGY_232: (8)                     return self
SNGT_QHENOMENOLOGY_233: (4)                 def index_of_part(self, part: SingleStringTex, start:
int = 0) -> int:
SNGT_QHENOMENOLOGY_234: (8)                     return self.submobjects.index(part, start)
SNGT_QHENOMENOLOGY_235: (4)                 def index_of_part_by_tex(self, tex: str, start: int =
0, **kwargs) -> int:
SNGT_QHENOMENOLOGY_236: (8)                     part = self.get_part_by_tex(tex, **kwargs)
SNGT_QHENOMENOLOGY_237: (8)                     return self.index_of_part(part, start)
SNGT_QHENOMENOLOGY_238: (4)                 def slice_by_tex(
SNGT_QHENOMENOLOGY_239: (8)                     self,
SNGT_QHENOMENOLOGY_240: (8)                     start_tex: str | None = None,
SNGT_QHENOMENOLOGY_241: (8)                     stop_tex: str | None = None,
SNGT_QHENOMENOLOGY_242: (8)                     **kwargs
SNGT_QHENOMENOLOGY_243: (4)                 ) -> VGroup:
SNGT_QHENOMENOLOGY_244: (8)                     if start_tex is None:
SNGT_QHENOMENOLOGY_245: (12)                        start_index = 0
SNGT_QHENOMENOLOGY_246: (8)                     else:
SNGT_QHENOMENOLOGY_247: (12)                        start_index =
self.index_of_part_by_tex(start_tex, **kwargs)
SNGT_QHENOMENOLOGY_248: (8)                     if stop_tex is None:
SNGT_QHENOMENOLOGY_249: (12)                        return self[start_index:]
SNGT_QHENOMENOLOGY_250: (8)                     else:
SNGT_QHENOMENOLOGY_251: (12)                        stop_index =
self.index_of_part_by_tex(stop_tex, start=start_index, **kwargs)
SNGT_QHENOMENOLOGY_252: (12)                        return self[start_index:stop_index]
SNGT_QHENOMENOLOGY_253: (4)                 def sort_alphabetically(self) -> None:
SNGT_QHENOMENOLOGY_254: (8)                     self.submobjects.sort(key=lambda m: m.get_tex())
SNGT_QHENOMENOLOGY_255: (4)                 def set_bstroke(self, color: ManimColor = BLACK, width:
float = 4):
SNGT_QHENOMENOLOGY_256: (8)                     self.set_stroke(color, width, background=True)
SNGT_QHENOMENOLOGY_257: (8)                     return self
SNGT_QHENOMENOLOGY_258: (0)             class OldTexText(OldTex):
SNGT_QHENOMENOLOGY_259: (4)                 def __init__(
SNGT_QHENOMENOLOGY_260: (8)                     self,
SNGT_QHENOMENOLOGY_261: (8)                     *tex_strings: str,
SNGT_QHENOMENOLOGY_262: (8)                     math_mode: bool = False,
SNGT_QHENOMENOLOGY_263: (8)                     arg_separator: str = "",
SNGT_QHENOMENOLOGY_264: (8)                     **kwargs
SNGT_QHENOMENOLOGY_265: (4)                 ):
```

```
SNGT_QHENOMENOLOGY_266: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_267: (12)                        *tex_strings,
SNGT_QHENOMENOLOGY_268: (12)                        math_mode=math_mode,
SNGT_QHENOMENOLOGY_269: (12)                        arg_separator=arg_separator,
SNGT_QHENOMENOLOGY_270: (12)                        **kwargs
SNGT_QHENOMENOLOGY_271: (8)                     )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 63 - three_dimensions.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import math
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import BLUE, BLUE_D, BLUE_E,
GREY_A, BLACK
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import IN, ORIGIN, OUT, RIGHT
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import PI, TAU
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.mobject.types.surface import SGroup
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.mobject.types.surface import Surface
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.geometry import Polygon
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.mobject.geometry import Square
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.utils.iterables import adjacent_pairs
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.space_ops import compass_directions
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.utils.space_ops import z_to_vector
SNGT_QHENOMENOLOGY_19: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_20: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_21: (4)                  from typing import Tuple, TypeVar
SNGT_QHENOMENOLOGY_22: (4)                  from manimlib.typing import ManimColor, Vect3, Sequence
SNGT_QHENOMENOLOGY_23: (4)                  T = TypeVar("T", bound=Mobject)
SNGT_QHENOMENOLOGY_24: (0)              class SurfaceMesh(VGroup):
SNGT_QHENOMENOLOGY_25: (4)                  def __init__(
SNGT_QHENOMENOLOGY_26: (8)                      self,
SNGT_QHENOMENOLOGY_27: (8)                      uv_surface: Surface,
SNGT_QHENOMENOLOGY_28: (8)                      resolution: Tuple[int, int] = (21, 11),
SNGT_QHENOMENOLOGY_29: (8)                      stroke_width: float = 1,
SNGT_QHENOMENOLOGY_30: (8)                      stroke_color: ManimColor = GREY_A,
SNGT_QHENOMENOLOGY_31: (8)                      normal_nudge: float = 1e-2,
SNGT_QHENOMENOLOGY_32: (8)                      depth_test: bool = True,
SNGT_QHENOMENOLOGY_33: (8)                      joint_type: str = 'no_joint',
SNGT_QHENOMENOLOGY_34: (8)                      **kwargs
SNGT_QHENOMENOLOGY_35: (4)                  ):
SNGT_QHENOMENOLOGY_36: (8)                      self.uv_surface = uv_surface
SNGT_QHENOMENOLOGY_37: (8)                      self.resolution = resolution
SNGT_QHENOMENOLOGY_38: (8)                      self.normal_nudge = normal_nudge
SNGT_QHENOMENOLOGY_39: (8)                      super().__init__(
SNGT_QHENOMENOLOGY_40: (12)                         stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_41: (12)                         stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_42: (12)                         depth_test=depth_test,
SNGT_QHENOMENOLOGY_43: (12)                         joint_type=joint_type,
SNGT_QHENOMENOLOGY_44: (12)                         **kwargs
SNGT_QHENOMENOLOGY_45: (8)                      )
SNGT_QHENOMENOLOGY_46: (4)                  def init_points(self) -> None:
SNGT_QHENOMENOLOGY_47: (8)                      uv_surface = self.uv_surface
SNGT_QHENOMENOLOGY_48: (8)                      full_nu, full_nv = uv_surface.resolution
SNGT_QHENOMENOLOGY_49: (8)                      part_nu, part_nv = self.resolution
SNGT_QHENOMENOLOGY_50: (8)                      u_indices = np.linspace(0, full_nu - 1, part_nu)
SNGT_QHENOMENOLOGY_51: (8)                      v_indices = np.linspace(0, full_nv - 1, part_nv)
SNGT_QHENOMENOLOGY_52: (8)                      points = uv_surface.get_points()
SNGT_QHENOMENOLOGY_53: (8)                      normals = uv_surface.get_unit_normals()
SNGT_QHENOMENOLOGY_54: (8)                      nudge = self.normal_nudge
SNGT_QHENOMENOLOGY_55: (8)                      nudged_points = points + nudge * normals
```

```
SNGT_QHENOMENOLOGY_56: (8)                    for ui in u_indices:
SNGT_QHENOMENOLOGY_57: (12)                       path = VMobject()
SNGT_QHENOMENOLOGY_58: (12)                       low_ui = full_nv * int(math.floor(ui))
SNGT_QHENOMENOLOGY_59: (12)                       high_ui = full_nv * int(math.ceil(ui))
SNGT_QHENOMENOLOGY_60: (12)                       path.set_points_smoothly(interpolate(
SNGT_QHENOMENOLOGY_61: (16)                           nudged_points[low_ui:low_ui + full_nv],
SNGT_QHENOMENOLOGY_62: (16)                           nudged_points[high_ui:high_ui + full_nv],
SNGT_QHENOMENOLOGY_63: (16)                           ui % 1
SNGT_QHENOMENOLOGY_64: (12)                       ))
SNGT_QHENOMENOLOGY_65: (12)                       self.add(path)
SNGT_QHENOMENOLOGY_66: (8)                    for vi in v_indices:
SNGT_QHENOMENOLOGY_67: (12)                       path = VMobject()
SNGT_QHENOMENOLOGY_68: (12)                       path.set_points_smoothly(interpolate(
SNGT_QHENOMENOLOGY_69: (16)
nudged_points[int(math.floor(vi))::full_nv],
SNGT_QHENOMENOLOGY_70: (16)                           nudged_points[int(math.ceil(vi))::full_nv],
SNGT_QHENOMENOLOGY_71: (16)                           vi % 1
SNGT_QHENOMENOLOGY_72: (12)                       ))
SNGT_QHENOMENOLOGY_73: (12)                       self.add(path)
SNGT_QHENOMENOLOGY_74: (0)            class Sphere(Surface):
SNGT_QHENOMENOLOGY_75: (4)                def __init__(
SNGT_QHENOMENOLOGY_76: (8)                    self,
SNGT_QHENOMENOLOGY_77: (8)                    u_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_78: (8)                    v_range: Tuple[float, float] = (0, PI),
SNGT_QHENOMENOLOGY_79: (8)                    resolution: Tuple[int, int] = (101, 51),
SNGT_QHENOMENOLOGY_80: (8)                    radius: float = 1.0,
SNGT_QHENOMENOLOGY_81: (8)                    **kwargs,
SNGT_QHENOMENOLOGY_82: (4)                ):
SNGT_QHENOMENOLOGY_83: (8)                    self.radius = radius
SNGT_QHENOMENOLOGY_84: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_85: (12)                       u_range=u_range,
SNGT_QHENOMENOLOGY_86: (12)                       v_range=v_range,
SNGT_QHENOMENOLOGY_87: (12)                       resolution=resolution,
SNGT_QHENOMENOLOGY_88: (12)                       **kwargs
SNGT_QHENOMENOLOGY_89: (8)                    )
SNGT_QHENOMENOLOGY_90: (4)                def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_91: (8)                    return self.radius * np.array([
SNGT_QHENOMENOLOGY_92: (12)                       math.cos(u) * math.sin(v),
SNGT_QHENOMENOLOGY_93: (12)                       math.sin(u) * math.sin(v),
SNGT_QHENOMENOLOGY_94: (12)                       -math.cos(v)
SNGT_QHENOMENOLOGY_95: (8)                    ])
SNGT_QHENOMENOLOGY_96: (0)            class Torus(Surface):
SNGT_QHENOMENOLOGY_97: (4)                def __init__(
SNGT_QHENOMENOLOGY_98: (8)                    self,
SNGT_QHENOMENOLOGY_99: (8)                    u_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_100: (8)                   v_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_101: (8)                   r1: float = 3.0,
SNGT_QHENOMENOLOGY_102: (8)                   r2: float = 1.0,
SNGT_QHENOMENOLOGY_103: (8)                   **kwargs,
SNGT_QHENOMENOLOGY_104: (4)               ):
SNGT_QHENOMENOLOGY_105: (8)                   self.r1 = r1
SNGT_QHENOMENOLOGY_106: (8)                   self.r2 = r2
SNGT_QHENOMENOLOGY_107: (8)                   super().__init__(
SNGT_QHENOMENOLOGY_108: (12)                      u_range=u_range,
SNGT_QHENOMENOLOGY_109: (12)                      v_range=v_range,
SNGT_QHENOMENOLOGY_110: (12)                      **kwargs,
SNGT_QHENOMENOLOGY_111: (8)                   )
SNGT_QHENOMENOLOGY_112: (4)               def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_113: (8)                   P = np.array([math.cos(u), math.sin(u), 0])
SNGT_QHENOMENOLOGY_114: (8)                   return (self.r1 - self.r2 * math.cos(v)) * P -
self.r2 * math.sin(v) * OUT
SNGT_QHENOMENOLOGY_115: (0)           class Cylinder(Surface):
SNGT_QHENOMENOLOGY_116: (4)               def __init__(
SNGT_QHENOMENOLOGY_117: (8)                   self,
SNGT_QHENOMENOLOGY_118: (8)                   u_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_119: (8)                   v_range: Tuple[float, float] = (-1, 1),
SNGT_QHENOMENOLOGY_120: (8)                   resolution: Tuple[int, int] = (101, 11),
SNGT_QHENOMENOLOGY_121: (8)                   height: float = 2,
SNGT_QHENOMENOLOGY_122: (8)                   radius: float = 1,
```

```
SNGT_QHENOMENOLOGY_123: (8)                     axis: Vect3 = OUT,
SNGT_QHENOMENOLOGY_124: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_125: (4)                 ):
SNGT_QHENOMENOLOGY_126: (8)                     self.height = height
SNGT_QHENOMENOLOGY_127: (8)                     self.radius = radius
SNGT_QHENOMENOLOGY_128: (8)                     self.axis = axis
SNGT_QHENOMENOLOGY_129: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_130: (12)                        u_range=u_range,
SNGT_QHENOMENOLOGY_131: (12)                        v_range=v_range,
SNGT_QHENOMENOLOGY_132: (12)                        resolution=resolution,
SNGT_QHENOMENOLOGY_133: (12)                        **kwargs
SNGT_QHENOMENOLOGY_134: (8)                     )
SNGT_QHENOMENOLOGY_135: (4)                 def init_points(self):
SNGT_QHENOMENOLOGY_136: (8)                     super().init_points()
SNGT_QHENOMENOLOGY_137: (8)                     self.scale(self.radius)
SNGT_QHENOMENOLOGY_138: (8)                     self.set_depth(self.height, stretch=True)
SNGT_QHENOMENOLOGY_139: (8)                     self.apply_matrix(z_to_vector(self.axis))
SNGT_QHENOMENOLOGY_140: (4)                 def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_141: (8)                     return np.array([np.cos(u), np.sin(u), v])
SNGT_QHENOMENOLOGY_142: (0)             class Cone(Cylinder):
SNGT_QHENOMENOLOGY_143: (4)                 def __init__(
SNGT_QHENOMENOLOGY_144: (8)                     self,
SNGT_QHENOMENOLOGY_145: (8)                     u_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_146: (8)                     v_range: Tuple[float, float] = (0, 1),
SNGT_QHENOMENOLOGY_147: (8)                     *args,
SNGT_QHENOMENOLOGY_148: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_149: (4)                 ):
SNGT_QHENOMENOLOGY_150: (8)                     super().__init__(u_range=u_range, v_range=v_range,
*args, **kwargs)
SNGT_QHENOMENOLOGY_151: (4)                 def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_152: (8)                     return np.array([(1 - v) * np.cos(u), (1 - v) *
np.sin(u), v])
SNGT_QHENOMENOLOGY_153: (0)             class Line3D(Cylinder):
SNGT_QHENOMENOLOGY_154: (4)                 def __init__(
SNGT_QHENOMENOLOGY_155: (8)                     self,
SNGT_QHENOMENOLOGY_156: (8)                     start: Vect3,
SNGT_QHENOMENOLOGY_157: (8)                     end: Vect3,
SNGT_QHENOMENOLOGY_158: (8)                     width: float = 0.05,
SNGT_QHENOMENOLOGY_159: (8)                     resolution: Tuple[int, int] = (21, 25),
SNGT_QHENOMENOLOGY_160: (8)                     **kwargs
SNGT_QHENOMENOLOGY_161: (4)                 ):
SNGT_QHENOMENOLOGY_162: (8)                     axis = end - start
SNGT_QHENOMENOLOGY_163: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_164: (12)                        height=get_norm(axis),
SNGT_QHENOMENOLOGY_165: (12)                        radius=width / 2,
SNGT_QHENOMENOLOGY_166: (12)                        axis=axis,
SNGT_QHENOMENOLOGY_167: (12)                        resolution=resolution,
SNGT_QHENOMENOLOGY_168: (12)                        **kwargs
SNGT_QHENOMENOLOGY_169: (8)                     )
SNGT_QHENOMENOLOGY_170: (8)                     self.shift((start + end) / 2)
SNGT_QHENOMENOLOGY_171: (0)             class Disk3D(Surface):
SNGT_QHENOMENOLOGY_172: (4)                 def __init__(
SNGT_QHENOMENOLOGY_173: (8)                     self,
SNGT_QHENOMENOLOGY_174: (8)                     radius: float = 1,
SNGT_QHENOMENOLOGY_175: (8)                     u_range: Tuple[float, float] = (0, 1),
SNGT_QHENOMENOLOGY_176: (8)                     v_range: Tuple[float, float] = (0, TAU),
SNGT_QHENOMENOLOGY_177: (8)                     resolution: Tuple[int, int] = (2, 100),
SNGT_QHENOMENOLOGY_178: (8)                     **kwargs
SNGT_QHENOMENOLOGY_179: (4)                 ):
SNGT_QHENOMENOLOGY_180: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_181: (12)                        u_range=u_range,
SNGT_QHENOMENOLOGY_182: (12)                        v_range=v_range,
SNGT_QHENOMENOLOGY_183: (12)                        resolution=resolution,
SNGT_QHENOMENOLOGY_184: (12)                        **kwargs,
SNGT_QHENOMENOLOGY_185: (8)                     )
SNGT_QHENOMENOLOGY_186: (8)                     self.scale(radius)
SNGT_QHENOMENOLOGY_187: (4)                 def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_188: (8)                     return np.array([
SNGT_QHENOMENOLOGY_189: (12)                        u * math.cos(v),
```

```
SNGT_QHENOMENOLOGY_190: (12)                         u * math.sin(v),
SNGT_QHENOMENOLOGY_191: (12)                         0
SNGT_QHENOMENOLOGY_192: (8)                     ])
SNGT_QHENOMENOLOGY_193: (0)             class Square3D(Surface):
SNGT_QHENOMENOLOGY_194: (4)                 def __init__(
SNGT_QHENOMENOLOGY_195: (8)                     self,
SNGT_QHENOMENOLOGY_196: (8)                     side_length: float = 2.0,
SNGT_QHENOMENOLOGY_197: (8)                     u_range: Tuple[float, float] = (-1, 1),
SNGT_QHENOMENOLOGY_198: (8)                     v_range: Tuple[float, float] = (-1, 1),
SNGT_QHENOMENOLOGY_199: (8)                     resolution: Tuple[int, int] = (2, 2),
SNGT_QHENOMENOLOGY_200: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_201: (4)                 ):
SNGT_QHENOMENOLOGY_202: (8)                     super().__init__(
SNGT_QHENOMENOLOGY_203: (12)                         u_range=u_range,
SNGT_QHENOMENOLOGY_204: (12)                         v_range=v_range,
SNGT_QHENOMENOLOGY_205: (12)                         resolution=resolution,
SNGT_QHENOMENOLOGY_206: (12)                         **kwargs
SNGT_QHENOMENOLOGY_207: (8)                     )
SNGT_QHENOMENOLOGY_208: (8)                     self.scale(side_length / 2)
SNGT_QHENOMENOLOGY_209: (4)                 def uv_func(self, u: float, v: float) -> np.ndarray:
SNGT_QHENOMENOLOGY_210: (8)                     return np.array([u, v, 0])
SNGT_QHENOMENOLOGY_211: (0)             def square_to_cube_faces(square: T) -> list[T]:
SNGT_QHENOMENOLOGY_212: (4)                 radius = square.get_height() / 2
SNGT_QHENOMENOLOGY_213: (4)                 square.move_to(radius * OUT)
SNGT_QHENOMENOLOGY_214: (4)                 result = [square.copy()]
SNGT_QHENOMENOLOGY_215: (4)                 result.extend([
SNGT_QHENOMENOLOGY_216: (8)                     square.copy().rotate(PI / 2, axis=vect,
about_point=ORIGIN)
SNGT_QHENOMENOLOGY_217: (8)                     for vect in compass_directions(4)
SNGT_QHENOMENOLOGY_218: (4)                 ])
SNGT_QHENOMENOLOGY_219: (4)                 result.append(square.copy().rotate(PI, RIGHT,
about_point=ORIGIN))
SNGT_QHENOMENOLOGY_220: (4)                 return result
SNGT_QHENOMENOLOGY_221: (0)             class Cube(SGroup):
SNGT_QHENOMENOLOGY_222: (4)                 def __init__(
SNGT_QHENOMENOLOGY_223: (8)                     self,
SNGT_QHENOMENOLOGY_224: (8)                     color: ManimColor = BLUE,
SNGT_QHENOMENOLOGY_225: (8)                     opacity: float = 1,
SNGT_QHENOMENOLOGY_226: (8)                     shading: Tuple[float, float, float] = (0.1, 0.5,
0.1),
SNGT_QHENOMENOLOGY_227: (8)                     square_resolution: Tuple[int, int] = (2, 2),
SNGT_QHENOMENOLOGY_228: (8)                     side_length: float = 2,
SNGT_QHENOMENOLOGY_229: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_230: (4)                 ):
SNGT_QHENOMENOLOGY_231: (8)                     face = Square3D(
SNGT_QHENOMENOLOGY_232: (12)                         resolution=square_resolution,
SNGT_QHENOMENOLOGY_233: (12)                         side_length=side_length,
SNGT_QHENOMENOLOGY_234: (12)                         color=color,
SNGT_QHENOMENOLOGY_235: (12)                         opacity=opacity,
SNGT_QHENOMENOLOGY_236: (12)                         shading=shading,
SNGT_QHENOMENOLOGY_237: (8)                     )
SNGT_QHENOMENOLOGY_238: (8)                     super().__init__(*square_to_cube_faces(face),
**kwargs)
SNGT_QHENOMENOLOGY_239: (0)             class Prism(Cube):
SNGT_QHENOMENOLOGY_240: (4)                 def __init__(
SNGT_QHENOMENOLOGY_241: (8)                     self,
SNGT_QHENOMENOLOGY_242: (8)                     width: float = 3.0,
SNGT_QHENOMENOLOGY_243: (8)                     height: float = 2.0,
SNGT_QHENOMENOLOGY_244: (8)                     depth: float = 1.0,
SNGT_QHENOMENOLOGY_245: (8)                     **kwargs
SNGT_QHENOMENOLOGY_246: (4)                 ):
SNGT_QHENOMENOLOGY_247: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_248: (8)                     for dim, value in enumerate([width, height,
depth]):
SNGT_QHENOMENOLOGY_249: (12)                         self.rescale_to_fit(value, dim, stretch=True)
SNGT_QHENOMENOLOGY_250: (0)             class VGroup3D(VGroup):
SNGT_QHENOMENOLOGY_251: (4)                 def __init__(
SNGT_QHENOMENOLOGY_252: (8)                     self,
SNGT_QHENOMENOLOGY_253: (8)                     *vmobjects: VMobject,
```

```
SNGT_QHENOMENOLOGY_254: (8)                     depth_test: bool = True,
SNGT_QHENOMENOLOGY_255: (8)                     shading: Tuple[float, float, float] = (0.2, 0.2,
0.2),
SNGT_QHENOMENOLOGY_256: (8)                     joint_type: str = "no_joint",
SNGT_QHENOMENOLOGY_257: (8)                     **kwargs
SNGT_QHENOMENOLOGY_258: (4)                 ):
SNGT_QHENOMENOLOGY_259: (8)                     super().__init__(*vmobjects, **kwargs)
SNGT_QHENOMENOLOGY_260: (8)                     self.set_shading(*shading)
SNGT_QHENOMENOLOGY_261: (8)                     self.set_joint_type(joint_type)
SNGT_QHENOMENOLOGY_262: (8)                     if depth_test:
SNGT_QHENOMENOLOGY_263: (12)                        self.apply_depth_test()
SNGT_QHENOMENOLOGY_264: (0)             class VCube(VGroup3D):
SNGT_QHENOMENOLOGY_265: (4)                 def __init__(
SNGT_QHENOMENOLOGY_266: (8)                     self,
SNGT_QHENOMENOLOGY_267: (8)                     side_length: float = 2.0,
SNGT_QHENOMENOLOGY_268: (8)                     fill_color: ManimColor = BLUE_D,
SNGT_QHENOMENOLOGY_269: (8)                     fill_opacity: float = 1,
SNGT_QHENOMENOLOGY_270: (8)                     stroke_width: float = 0,
SNGT_QHENOMENOLOGY_271: (8)                     **kwargs
SNGT_QHENOMENOLOGY_272: (4)                 ):
SNGT_QHENOMENOLOGY_273: (8)                     style = dict(
SNGT_QHENOMENOLOGY_274: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_275: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_276: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_277: (12)                        **kwargs
SNGT_QHENOMENOLOGY_278: (8)                     )
SNGT_QHENOMENOLOGY_279: (8)                     face = Square(side_length=side_length, **style)
SNGT_QHENOMENOLOGY_280: (8)                     super().__init__(*square_to_cube_faces(face),
**style)
SNGT_QHENOMENOLOGY_281: (0)             class VPrism(VCube):
SNGT_QHENOMENOLOGY_282: (4)                 def __init__(
SNGT_QHENOMENOLOGY_283: (8)                     self,
SNGT_QHENOMENOLOGY_284: (8)                     width: float = 3.0,
SNGT_QHENOMENOLOGY_285: (8)                     height: float = 2.0,
SNGT_QHENOMENOLOGY_286: (8)                     depth: float = 1.0,
SNGT_QHENOMENOLOGY_287: (8)                     **kwargs
SNGT_QHENOMENOLOGY_288: (4)                 ):
SNGT_QHENOMENOLOGY_289: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_290: (8)                     for dim, value in enumerate([width, height,
depth]):
SNGT_QHENOMENOLOGY_291: (12)                        self.rescale_to_fit(value, dim, stretch=True)
SNGT_QHENOMENOLOGY_292: (0)             class Dodecahedron(VGroup3D):
SNGT_QHENOMENOLOGY_293: (4)                 def __init__(
SNGT_QHENOMENOLOGY_294: (8)                     self,
SNGT_QHENOMENOLOGY_295: (8)                     fill_color: ManimColor = BLUE_E,
SNGT_QHENOMENOLOGY_296: (8)                     fill_opacity: float = 1,
SNGT_QHENOMENOLOGY_297: (8)                     stroke_color: ManimColor = BLUE_E,
SNGT_QHENOMENOLOGY_298: (8)                     stroke_width: float = 1,
SNGT_QHENOMENOLOGY_299: (8)                     shading: Tuple[float, float, float] = (0.2, 0.2,
0.2),
SNGT_QHENOMENOLOGY_300: (8)                     **kwargs,
SNGT_QHENOMENOLOGY_301: (4)                 ):
SNGT_QHENOMENOLOGY_302: (8)                     style = dict(
SNGT_QHENOMENOLOGY_303: (12)                        fill_color=fill_color,
SNGT_QHENOMENOLOGY_304: (12)                        fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_305: (12)                        stroke_color=stroke_color,
SNGT_QHENOMENOLOGY_306: (12)                        stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_307: (12)                        shading=shading,
SNGT_QHENOMENOLOGY_308: (12)                        **kwargs
SNGT_QHENOMENOLOGY_309: (8)                     )
SNGT_QHENOMENOLOGY_310: (8)                     phi = (1 + math.sqrt(5)) / 2
SNGT_QHENOMENOLOGY_311: (8)                     x, y, z = np.identity(3)
SNGT_QHENOMENOLOGY_312: (8)                     pentagon1 = Polygon(
SNGT_QHENOMENOLOGY_313: (12)                        np.array([phi, 1 / phi, 0]),
SNGT_QHENOMENOLOGY_314: (12)                        np.array([1, 1, 1]),
SNGT_QHENOMENOLOGY_315: (12)                        np.array([1 / phi, 0, phi]),
SNGT_QHENOMENOLOGY_316: (12)                        np.array([1, -1, 1]),
SNGT_QHENOMENOLOGY_317: (12)                        np.array([phi, -1 / phi, 0]),
SNGT_QHENOMENOLOGY_318: (12)                        **style
```

```
SNGT_QHENOMENOLOGY_319: (8)                     )
SNGT_QHENOMENOLOGY_320: (8)                     pentagon2 = pentagon1.copy().stretch(-1, 2,
about_point=ORIGIN)
SNGT_QHENOMENOLOGY_321: (8)                     pentagon2.reverse_points()
SNGT_QHENOMENOLOGY_322: (8)                     x_pair = VGroup(pentagon1, pentagon2)
SNGT_QHENOMENOLOGY_323: (8)                     z_pair = x_pair.copy().apply_matrix(np.array([z, -
x, -y]).T)
SNGT_QHENOMENOLOGY_324: (8)                     y_pair = x_pair.copy().apply_matrix(np.array([y, z,
x]).T)
SNGT_QHENOMENOLOGY_325: (8)                     pentagons = [*x_pair, *y_pair, *z_pair]
SNGT_QHENOMENOLOGY_326: (8)                     for pentagon in list(pentagons):
SNGT_QHENOMENOLOGY_327: (12)                        pc = pentagon.copy()
SNGT_QHENOMENOLOGY_328: (12)                        pc.apply_function(lambda p: -p)
SNGT_QHENOMENOLOGY_329: (12)                        pc.reverse_points()
SNGT_QHENOMENOLOGY_330: (12)                        pentagons.append(pc)
SNGT_QHENOMENOLOGY_331: (8)                     super().__init__(*pentagons, **style)
SNGT_QHENOMENOLOGY_332: (0)             class Prismify(VGroup3D):
SNGT_QHENOMENOLOGY_333: (4)                 def __init__(self, vmobject, depth=1.0, direction=IN,
**kwargs):
SNGT_QHENOMENOLOGY_334: (8)                     vect = depth * direction
SNGT_QHENOMENOLOGY_335: (8)                     pieces = [vmobject.copy()]
SNGT_QHENOMENOLOGY_336: (8)                     points = vmobject.get_anchors()
SNGT_QHENOMENOLOGY_337: (8)                     for p1, p2 in adjacent_pairs(points):
SNGT_QHENOMENOLOGY_338: (12)                        wall = VMobject()
SNGT_QHENOMENOLOGY_339: (12)                        wall.match_style(vmobject)
SNGT_QHENOMENOLOGY_340: (12)                        wall.set_points_as_corners([p1, p2, p2 + vect,
p1 + vect])
SNGT_QHENOMENOLOGY_341: (12)                        pieces.append(wall)
SNGT_QHENOMENOLOGY_342: (8)                     top = vmobject.copy()
SNGT_QHENOMENOLOGY_343: (8)                     top.shift(vect)
SNGT_QHENOMENOLOGY_344: (8)                     top.reverse_points()
SNGT_QHENOMENOLOGY_345: (8)                     pieces.append(top)
SNGT_QHENOMENOLOGY_346: (8)                     super().__init__(*pieces, **kwargs)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 64 - vectorized_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from functools import wraps
SNGT_QHENOMENOLOGY_3: (0)               import numpy as np
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.constants import GREY_A, GREY_C, GREY_E
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.constants import BLACK
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.constants import DEFAULT_STROKE_WIDTH
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.constants import ORIGIN, OUT
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.constants import PI
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.constants import TAU
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.mobject.mobject import Point
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.bezier import bezier
SNGT_QHENOMENOLOGY_15: (0)              from manimlib.utils.bezier import
get_quadratic_approximation_of_cubic
SNGT_QHENOMENOLOGY_16: (0)              from manimlib.utils.bezier import
approx_smooth_quadratic_bezier_handles
SNGT_QHENOMENOLOGY_17: (0)              from manimlib.utils.bezier import smooth_quadratic_path
SNGT_QHENOMENOLOGY_18: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_19: (0)              from manimlib.utils.bezier import integer_interpolate
SNGT_QHENOMENOLOGY_20: (0)              from manimlib.utils.bezier import inverse_interpolate
SNGT_QHENOMENOLOGY_21: (0)              from manimlib.utils.bezier import find_intersection
SNGT_QHENOMENOLOGY_22: (0)              from manimlib.utils.bezier import outer_interpolate
SNGT_QHENOMENOLOGY_23: (0)              from manimlib.utils.bezier import
partial_quadratic_bezier_points
SNGT_QHENOMENOLOGY_24: (0)              from manimlib.utils.bezier import
quadratic_bezier_points_for_arc
SNGT_QHENOMENOLOGY_25: (0)              from manimlib.utils.color import color_gradient
SNGT_QHENOMENOLOGY_26: (0)              from manimlib.utils.color import rgb_to_hex
SNGT_QHENOMENOLOGY_27: (0)              from manimlib.utils.iterables import make_even
```

```
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.iterables import resize_array
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_30: (0)             from manimlib.utils.iterables import
resize_preserving_order
SNGT_QHENOMENOLOGY_31: (0)             from manimlib.utils.space_ops import angle_between_vectors
SNGT_QHENOMENOLOGY_32: (0)             from manimlib.utils.space_ops import cross2d
SNGT_QHENOMENOLOGY_33: (0)             from manimlib.utils.space_ops import earclip_triangulation
SNGT_QHENOMENOLOGY_34: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_35: (0)             from manimlib.utils.space_ops import get_unit_normal
SNGT_QHENOMENOLOGY_36: (0)             from manimlib.utils.space_ops import line_intersects_path
SNGT_QHENOMENOLOGY_37: (0)             from manimlib.utils.space_ops import midpoint
SNGT_QHENOMENOLOGY_38: (0)             from manimlib.utils.space_ops import
rotation_between_vectors
SNGT_QHENOMENOLOGY_39: (0)             from manimlib.utils.space_ops import
rotation_matrix_transpose
SNGT_QHENOMENOLOGY_40: (0)             from manimlib.utils.space_ops import poly_line_length
SNGT_QHENOMENOLOGY_41: (0)             from manimlib.utils.space_ops import z_to_vector
SNGT_QHENOMENOLOGY_42: (0)             from manimlib.shader_wrapper import VShaderWrapper
SNGT_QHENOMENOLOGY_43: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_44: (0)             from typing import Generic, TypeVar, Iterable
SNGT_QHENOMENOLOGY_45: (0)             SubVmobjectType = TypeVar('SubVmobjectType',
bound='VMobject')
SNGT_QHENOMENOLOGY_46: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_47: (4)                 from typing import Callable, Tuple, Any, Optional
SNGT_QHENOMENOLOGY_48: (4)                 from manimlib.typing import ManimColor, Vect3, Vect4,
Vect3Array, Self
SNGT_QHENOMENOLOGY_49: (4)                 from moderngl.context import Context
SNGT_QHENOMENOLOGY_50: (0)             DEFAULT_STROKE_COLOR = GREY_A
SNGT_QHENOMENOLOGY_51: (0)             DEFAULT_FILL_COLOR = GREY_C
SNGT_QHENOMENOLOGY_52: (0)             class VMobject(Mobject):
SNGT_QHENOMENOLOGY_53: (4)                 data_dtype: np.dtype = np.dtype([
SNGT_QHENOMENOLOGY_54: (8)                     ('point', np.float32, (3,)),
SNGT_QHENOMENOLOGY_55: (8)                     ('stroke_rgba', np.float32, (4,)),
SNGT_QHENOMENOLOGY_56: (8)                     ('stroke_width', np.float32, (1,)),
SNGT_QHENOMENOLOGY_57: (8)                     ('joint_angle', np.float32, (1,)),
SNGT_QHENOMENOLOGY_58: (8)                     ('fill_rgba', np.float32, (4,)),
SNGT_QHENOMENOLOGY_59: (8)                     ('base_normal', np.float32, (3,)),  # Base points
and unit normal vectors are interleaved in this array
SNGT_QHENOMENOLOGY_60: (8)                     ('fill_border_width', np.float32, (1,)),
SNGT_QHENOMENOLOGY_61: (4)                 ])
SNGT_QHENOMENOLOGY_62: (4)                 pre_function_handle_to_anchor_scale_factor: float =
0.01
SNGT_QHENOMENOLOGY_63: (4)                 make_smooth_after_applying_functions: bool = False
SNGT_QHENOMENOLOGY_64: (4)                 tolerance_for_point_equality: float = 1e-8
SNGT_QHENOMENOLOGY_65: (4)                 joint_type_map: dict = {
SNGT_QHENOMENOLOGY_66: (8)                     "no_joint": 0,
SNGT_QHENOMENOLOGY_67: (8)                     "auto": 1,
SNGT_QHENOMENOLOGY_68: (8)                     "bevel": 2,
SNGT_QHENOMENOLOGY_69: (8)                     "miter": 3,
SNGT_QHENOMENOLOGY_70: (4)                 }
SNGT_QHENOMENOLOGY_71: (4)                 def __init__(
SNGT_QHENOMENOLOGY_72: (8)                     self,
SNGT_QHENOMENOLOGY_73: (8)                     color: ManimColor = None,  # If set, this will
override stroke_color and fill_color
SNGT_QHENOMENOLOGY_74: (8)                     fill_color: ManimColor = None,
SNGT_QHENOMENOLOGY_75: (8)                     fill_opacity: float | Iterable[float] | None = 0.0,
SNGT_QHENOMENOLOGY_76: (8)                     stroke_color: ManimColor = None,
SNGT_QHENOMENOLOGY_77: (8)                     stroke_opacity: float | Iterable[float] | None =
1.0,
SNGT_QHENOMENOLOGY_78: (8)                     stroke_width: float | Iterable[float] | None =
DEFAULT_STROKE_WIDTH,
SNGT_QHENOMENOLOGY_79: (8)                     stroke_behind: bool = False,
SNGT_QHENOMENOLOGY_80: (8)                     background_image_file: str | None = None,
SNGT_QHENOMENOLOGY_81: (8)                     long_lines: bool = False,
SNGT_QHENOMENOLOGY_82: (8)                     joint_type: str = "auto",
SNGT_QHENOMENOLOGY_83: (8)                     flat_stroke: bool = False,
SNGT_QHENOMENOLOGY_84: (8)                     scale_stroke_with_zoom: bool = False,
SNGT_QHENOMENOLOGY_85: (8)                     use_simple_quadratic_approx: bool = False,
```

```
SNGT_QHENOMENOLOGY_86: (8)                      anti_alias_width: float = 1.5,
SNGT_QHENOMENOLOGY_87: (8)                      fill_border_width: float = 0.0,
SNGT_QHENOMENOLOGY_88: (8)                      **kwargs
SNGT_QHENOMENOLOGY_89: (4)                  ):
SNGT_QHENOMENOLOGY_90: (8)                      self.fill_color = fill_color or color or
DEFAULT_FILL_COLOR
SNGT_QHENOMENOLOGY_91: (8)                      self.fill_opacity = fill_opacity
SNGT_QHENOMENOLOGY_92: (8)                      self.stroke_color = stroke_color or color or
DEFAULT_STROKE_COLOR
SNGT_QHENOMENOLOGY_93: (8)                      self.stroke_opacity = stroke_opacity
SNGT_QHENOMENOLOGY_94: (8)                      self.stroke_width = stroke_width
SNGT_QHENOMENOLOGY_95: (8)                      self.stroke_behind = stroke_behind
SNGT_QHENOMENOLOGY_96: (8)                      self.background_image_file = background_image_file
SNGT_QHENOMENOLOGY_97: (8)                      self.long_lines = long_lines
SNGT_QHENOMENOLOGY_98: (8)                      self.joint_type = joint_type
SNGT_QHENOMENOLOGY_99: (8)                      self.flat_stroke = flat_stroke
SNGT_QHENOMENOLOGY_100: (8)                     self.scale_stroke_with_zoom =
scale_stroke_with_zoom
SNGT_QHENOMENOLOGY_101: (8)                     self.use_simple_quadratic_approx =
use_simple_quadratic_approx
SNGT_QHENOMENOLOGY_102: (8)                     self.anti_alias_width = anti_alias_width
SNGT_QHENOMENOLOGY_103: (8)                     self.fill_border_width = fill_border_width
SNGT_QHENOMENOLOGY_104: (8)                     self.needs_new_joint_angles = True
SNGT_QHENOMENOLOGY_105: (8)                     self.needs_new_unit_normal = True
SNGT_QHENOMENOLOGY_106: (8)                     self.subpath_end_indices = None
SNGT_QHENOMENOLOGY_107: (8)                     self.outer_vert_indices = np.zeros(0, dtype=int)
SNGT_QHENOMENOLOGY_108: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_109: (4)                 def get_group_class(self):
SNGT_QHENOMENOLOGY_110: (8)                     return VGroup
SNGT_QHENOMENOLOGY_111: (4)                 def init_uniforms(self):
SNGT_QHENOMENOLOGY_112: (8)                     super().init_uniforms()
SNGT_QHENOMENOLOGY_113: (8)                     self.uniforms.update(
SNGT_QHENOMENOLOGY_114: (12)                        anti_alias_width=self.anti_alias_width,
SNGT_QHENOMENOLOGY_115: (12)
joint_type=self.joint_type_map[self.joint_type],
SNGT_QHENOMENOLOGY_116: (12)                        flat_stroke=float(self.flat_stroke),
SNGT_QHENOMENOLOGY_117: (12)
scale_stroke_with_zoom=float(self.scale_stroke_with_zoom)
SNGT_QHENOMENOLOGY_118: (8)                     )
SNGT_QHENOMENOLOGY_119: (4)                 def add(self, *vmobjects: VMobject) -> Self:
SNGT_QHENOMENOLOGY_120: (8)                     if not all((isinstance(m, VMobject) for m in
vmobjects)):
SNGT_QHENOMENOLOGY_121: (12)                        raise Exception("All submobjects must be of
type VMobject")
SNGT_QHENOMENOLOGY_122: (8)                     return super().add(*vmobjects)
SNGT_QHENOMENOLOGY_123: (4)                 def init_colors(self):
SNGT_QHENOMENOLOGY_124: (8)                     self.set_stroke(
SNGT_QHENOMENOLOGY_125: (12)                        color=self.stroke_color,
SNGT_QHENOMENOLOGY_126: (12)                        width=self.stroke_width,
SNGT_QHENOMENOLOGY_127: (12)                        opacity=self.stroke_opacity,
SNGT_QHENOMENOLOGY_128: (12)                        behind=self.stroke_behind,
SNGT_QHENOMENOLOGY_129: (8)                     )
SNGT_QHENOMENOLOGY_130: (8)                     self.set_fill(
SNGT_QHENOMENOLOGY_131: (12)                        color=self.fill_color,
SNGT_QHENOMENOLOGY_132: (12)                        opacity=self.fill_opacity,
SNGT_QHENOMENOLOGY_133: (12)                        border_width=self.fill_border_width,
SNGT_QHENOMENOLOGY_134: (8)                     )
SNGT_QHENOMENOLOGY_135: (8)                     self.set_shading(*self.shading)
SNGT_QHENOMENOLOGY_136: (8)                     self.set_flat_stroke(self.flat_stroke)
SNGT_QHENOMENOLOGY_137: (8)                     self.color = self.get_color()
SNGT_QHENOMENOLOGY_138: (8)                     return self
SNGT_QHENOMENOLOGY_139: (4)                 def set_fill(
SNGT_QHENOMENOLOGY_140: (8)                     self,
SNGT_QHENOMENOLOGY_141: (8)                     color: ManimColor | Iterable[ManimColor] = None,
SNGT_QHENOMENOLOGY_142: (8)                     opacity: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_143: (8)                     border_width: float | None = None,
SNGT_QHENOMENOLOGY_144: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_145: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_146: (8)                     self.set_rgba_array_by_color(color, opacity,
```

```
'fill_rgba', recurse)
                    if border_width is not None:
                        self.border_width = border_width
                        for mob in self.get_family(recurse):
                            data = mob.data if mob.has_points() > 0
else mob._data_defaults
                            data["fill_border_width"] = border_width
                    return self
                def set_stroke(
                    self,
                    color: ManimColor | Iterable[ManimColor] = None,
                    width: float | Iterable[float] | None = None,
                    opacity: float | Iterable[float] | None = None,
                    behind: bool | None = None,
                    flat: bool | None = None,
                    recurse: bool = True
                ) -> Self:
                    self.set_rgba_array_by_color(color, opacity,
'stroke_rgba', recurse)
                    if width is not None:
                        for mob in self.get_family(recurse):
                            data = mob.data if mob.get_num_points() > 0
else mob._data_defaults
                            if isinstance(width, (float, int)):
                                data['stroke_width'][:, 0] = width
                            else:
                                data['stroke_width'][:, 0] =
resize_with_interpolation(
                                    np.array(width), len(data)
                                ).flatten()
                    if behind is not None:
                        for mob in self.get_family(recurse):
                            if mob.stroke_behind != behind:
                                mob.stroke_behind = behind
                                mob.refresh_shader_wrapper_id()
                    if flat is not None:
                        self.set_flat_stroke(flat)
                    return self
                def set_backstroke(
                    self,
                    color: ManimColor | Iterable[ManimColor] = BLACK,
                    width: float | Iterable[float] = 3,
                ) -> Self:
                    self.set_stroke(color, width, behind=True)
                    return self
                @Mobject.affects_family_data
                def set_style(
                    self,
                    fill_color: ManimColor | Iterable[ManimColor] |
None = None,
                    fill_opacity: float | Iterable[float] | None =
None,
                    fill_rgba: Vect4 | None = None,
                    fill_border_width: float | None = None,
                    stroke_color: ManimColor | Iterable[ManimColor] |
None = None,
                    stroke_opacity: float | Iterable[float] | None =
None,
                    stroke_rgba: Vect4 | None = None,
                    stroke_width: float | Iterable[float] | None =
None,
                    stroke_behind: bool | None = None,
                    flat_stroke: Optional[bool] = None,
                    shading: Tuple[float, float, float] | None = None,
                    recurse: bool = True
                ) -> Self:
                    for mob in self.get_family(recurse):
                        if fill_rgba is not None:
                            mob.data['fill_rgba'][:] =
```

```
resize_with_interpolation(fill_rgba, len(mob.data['fill_rgba']))
SNGT_QHENOMENOLOGY_206: (12)                        else:
SNGT_QHENOMENOLOGY_207: (16)                            mob.set_fill(
SNGT_QHENOMENOLOGY_208: (20)                                color=fill_color,
SNGT_QHENOMENOLOGY_209: (20)                                opacity=fill_opacity,
SNGT_QHENOMENOLOGY_210: (20)                                border_width=fill_border_width,
SNGT_QHENOMENOLOGY_211: (20)                                recurse=False
SNGT_QHENOMENOLOGY_212: (16)                            )
SNGT_QHENOMENOLOGY_213: (12)                        if stroke_rgba is not None:
SNGT_QHENOMENOLOGY_214: (16)                            mob.data['stroke_rgba'][:] =
resize_with_interpolation(stroke_rgba, len(mob.data['stroke_rgba']))
SNGT_QHENOMENOLOGY_215: (16)                            mob.set_stroke(
SNGT_QHENOMENOLOGY_216: (20)                                width=stroke_width,
SNGT_QHENOMENOLOGY_217: (20)                                behind=stroke_behind,
SNGT_QHENOMENOLOGY_218: (20)                                flat=flat_stroke,
SNGT_QHENOMENOLOGY_219: (20)                                recurse=False,
SNGT_QHENOMENOLOGY_220: (16)                            )
SNGT_QHENOMENOLOGY_221: (12)                        else:
SNGT_QHENOMENOLOGY_222: (16)                            mob.set_stroke(
SNGT_QHENOMENOLOGY_223: (20)                                color=stroke_color,
SNGT_QHENOMENOLOGY_224: (20)                                width=stroke_width,
SNGT_QHENOMENOLOGY_225: (20)                                opacity=stroke_opacity,
SNGT_QHENOMENOLOGY_226: (20)                                flat=flat_stroke,
SNGT_QHENOMENOLOGY_227: (20)                                behind=stroke_behind,
SNGT_QHENOMENOLOGY_228: (20)                                recurse=False,
SNGT_QHENOMENOLOGY_229: (16)                            )
SNGT_QHENOMENOLOGY_230: (12)                        if shading is not None:
SNGT_QHENOMENOLOGY_231: (16)                            mob.set_shading(*shading, recurse=False)
SNGT_QHENOMENOLOGY_232: (8)                 return self
SNGT_QHENOMENOLOGY_233: (4)             def get_style(self) -> dict[str, Any]:
SNGT_QHENOMENOLOGY_234: (8)                 data = self.data if self.get_num_points() > 0 else
self._data_defaults
SNGT_QHENOMENOLOGY_235: (8)                 return {
SNGT_QHENOMENOLOGY_236: (12)                    "fill_rgba": data['fill_rgba'].copy(),
SNGT_QHENOMENOLOGY_237: (12)                    "fill_border_width":
data['fill_border_width'].copy(),
SNGT_QHENOMENOLOGY_238: (12)                    "stroke_rgba": data['stroke_rgba'].copy(),
SNGT_QHENOMENOLOGY_239: (12)                    "stroke_width": data['stroke_width'].copy(),
SNGT_QHENOMENOLOGY_240: (12)                    "stroke_behind": self.stroke_behind,
SNGT_QHENOMENOLOGY_241: (12)                    "flat_stroke": self.get_flat_stroke(),
SNGT_QHENOMENOLOGY_242: (12)                    "shading": self.get_shading(),
SNGT_QHENOMENOLOGY_243: (8)                 }
SNGT_QHENOMENOLOGY_244: (4)             def match_style(self, vmobject: VMobject, recurse: bool
= True) -> Self:
SNGT_QHENOMENOLOGY_245: (8)                 self.set_style(**vmobject.get_style(),
recurse=False)
SNGT_QHENOMENOLOGY_246: (8)                 if recurse:
SNGT_QHENOMENOLOGY_247: (12)                    submobs1, submobs2 = self.submobjects,
vmobject.submobjects
SNGT_QHENOMENOLOGY_248: (12)                    if len(submobs1) == 0:
SNGT_QHENOMENOLOGY_249: (16)                        return self
SNGT_QHENOMENOLOGY_250: (12)                    elif len(submobs2) == 0:
SNGT_QHENOMENOLOGY_251: (16)                        submobs2 = [vmobject]
SNGT_QHENOMENOLOGY_252: (12)                    for sm1, sm2 in zip(*make_even(submobs1,
submobs2)):
SNGT_QHENOMENOLOGY_253: (16)                        sm1.match_style(sm2)
SNGT_QHENOMENOLOGY_254: (8)                 return self
SNGT_QHENOMENOLOGY_255: (4)             def set_color(
SNGT_QHENOMENOLOGY_256: (8)                 self,
SNGT_QHENOMENOLOGY_257: (8)                 color: ManimColor | Iterable[ManimColor] | None,
SNGT_QHENOMENOLOGY_258: (8)                 opacity: float | Iterable[float] | None = None,
SNGT_QHENOMENOLOGY_259: (8)                 recurse: bool = True
SNGT_QHENOMENOLOGY_260: (4)             ) -> Self:
SNGT_QHENOMENOLOGY_261: (8)                 self.set_fill(color, opacity=opacity,
recurse=recurse)
SNGT_QHENOMENOLOGY_262: (8)                 self.set_stroke(color, opacity=opacity,
recurse=recurse)
SNGT_QHENOMENOLOGY_263: (8)                 return self
SNGT_QHENOMENOLOGY_264: (4)             def set_opacity(
```

```
SNGT_QHENOMENOLOGY_265: (8)                     self,
SNGT_QHENOMENOLOGY_266: (8)                     opacity: float | Iterable[float] | None,
SNGT_QHENOMENOLOGY_267: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_268: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_269: (8)                     self.set_fill(opacity=opacity, recurse=recurse)
SNGT_QHENOMENOLOGY_270: (8)                     self.set_stroke(opacity=opacity, recurse=recurse)
SNGT_QHENOMENOLOGY_271: (8)                     return self
SNGT_QHENOMENOLOGY_272: (4)                 def set_anti_alias_width(self, anti_alias_width: float,
recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_273: (8)                     self.set_uniform(recurse,
anti_alias_width=anti_alias_width)
SNGT_QHENOMENOLOGY_274: (8)                     return self
SNGT_QHENOMENOLOGY_275: (4)                 def fade(self, darkness: float = 0.5, recurse: bool =
True) -> Self:
SNGT_QHENOMENOLOGY_276: (8)                     mobs = self.get_family() if recurse else [self]
SNGT_QHENOMENOLOGY_277: (8)                     for mob in mobs:
SNGT_QHENOMENOLOGY_278: (12)                        factor = 1.0 - darkness
SNGT_QHENOMENOLOGY_279: (12)                        mob.set_fill(
SNGT_QHENOMENOLOGY_280: (16)                            opacity=factor * mob.get_fill_opacity(),
SNGT_QHENOMENOLOGY_281: (16)                            recurse=False,
SNGT_QHENOMENOLOGY_282: (12)                        )
SNGT_QHENOMENOLOGY_283: (12)                        mob.set_stroke(
SNGT_QHENOMENOLOGY_284: (16)                            opacity=factor * mob.get_stroke_opacity(),
SNGT_QHENOMENOLOGY_285: (16)                            recurse=False,
SNGT_QHENOMENOLOGY_286: (12)                        )
SNGT_QHENOMENOLOGY_287: (8)                     return self
SNGT_QHENOMENOLOGY_288: (4)                 def get_fill_colors(self) -> list[str]:
SNGT_QHENOMENOLOGY_289: (8)                     return [
SNGT_QHENOMENOLOGY_290: (12)                        rgb_to_hex(rgba[:3])
SNGT_QHENOMENOLOGY_291: (12)                        for rgba in self.data['fill_rgba']
SNGT_QHENOMENOLOGY_292: (8)                     ]
SNGT_QHENOMENOLOGY_293: (4)                 def get_fill_opacities(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_294: (8)                     return self.data['fill_rgba'][:, 3]
SNGT_QHENOMENOLOGY_295: (4)                 def get_stroke_colors(self) -> list[str]:
SNGT_QHENOMENOLOGY_296: (8)                     return [
SNGT_QHENOMENOLOGY_297: (12)                        rgb_to_hex(rgba[:3])
SNGT_QHENOMENOLOGY_298: (12)                        for rgba in self.data['stroke_rgba']
SNGT_QHENOMENOLOGY_299: (8)                     ]
SNGT_QHENOMENOLOGY_300: (4)                 def get_stroke_opacities(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_301: (8)                     return self.data['stroke_rgba'][:, 3]
SNGT_QHENOMENOLOGY_302: (4)                 def get_stroke_widths(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_303: (8)                     return self.data['stroke_width'][:, 0]
SNGT_QHENOMENOLOGY_304: (4)                 def get_fill_color(self) -> str:
SNGT_QHENOMENOLOGY_305: (8)                     """
SNGT_QHENOMENOLOGY_306: (8)                     If there are multiple colors (for gradient)
SNGT_QHENOMENOLOGY_307: (8)                     this returns the first one
SNGT_QHENOMENOLOGY_308: (8)                     """
SNGT_QHENOMENOLOGY_309: (8)                     data = self.data if self.has_points() else
self._data_defaults
SNGT_QHENOMENOLOGY_310: (8)                     return rgb_to_hex(data["fill_rgba"][0, :3])
SNGT_QHENOMENOLOGY_311: (4)                 def get_fill_opacity(self) -> float:
SNGT_QHENOMENOLOGY_312: (8)                     """
SNGT_QHENOMENOLOGY_313: (8)                     If there are multiple opacities, this returns the
SNGT_QHENOMENOLOGY_314: (8)                     first
SNGT_QHENOMENOLOGY_315: (8)                     """
SNGT_QHENOMENOLOGY_316: (8)                     data = self.data if self.has_points() else
self._data_defaults
SNGT_QHENOMENOLOGY_317: (8)                     return data["fill_rgba"][0, 3]
SNGT_QHENOMENOLOGY_318: (4)                 def get_stroke_color(self) -> str:
SNGT_QHENOMENOLOGY_319: (8)                     data = self.data if self.has_points() else
self._data_defaults
SNGT_QHENOMENOLOGY_320: (8)                     return rgb_to_hex(data["stroke_rgba"][0, :3])
SNGT_QHENOMENOLOGY_321: (4)                 def get_stroke_width(self) -> float:
SNGT_QHENOMENOLOGY_322: (8)                     data = self.data if self.has_points() else
self._data_defaults
SNGT_QHENOMENOLOGY_323: (8)                     return data["stroke_width"][0, 0]
SNGT_QHENOMENOLOGY_324: (4)                 def get_stroke_opacity(self) -> float:
SNGT_QHENOMENOLOGY_325: (8)                     data = self.data if self.has_points() else
self._data_defaults
```

```
SNGT_QHENOMENOLOGY_326: (8)                     return data["stroke_rgba"][0, 3]
SNGT_QHENOMENOLOGY_327: (4)                 def get_color(self) -> str:
SNGT_QHENOMENOLOGY_328: (8)                     if self.has_fill():
SNGT_QHENOMENOLOGY_329: (12)                        return self.get_fill_color()
SNGT_QHENOMENOLOGY_330: (8)                     return self.get_stroke_color()
SNGT_QHENOMENOLOGY_331: (4)                 def get_anti_alias_width(self):
SNGT_QHENOMENOLOGY_332: (8)                     return self.uniforms["anti_alias_width"]
SNGT_QHENOMENOLOGY_333: (4)                 def has_stroke(self) -> bool:
SNGT_QHENOMENOLOGY_334: (8)                     data = self.data if len(self.data) > 0 else
self._data_defaults
SNGT_QHENOMENOLOGY_335: (8)                     return any(data['stroke_width']) and
any(data['stroke_rgba'][:, 3])
SNGT_QHENOMENOLOGY_336: (4)                 def has_fill(self) -> bool:
SNGT_QHENOMENOLOGY_337: (8)                     data = self.data if len(self.data) > 0 else
self._data_defaults
SNGT_QHENOMENOLOGY_338: (8)                     return any(data['fill_rgba'][:, 3])
SNGT_QHENOMENOLOGY_339: (4)                 def get_opacity(self) -> float:
SNGT_QHENOMENOLOGY_340: (8)                     if self.has_fill():
SNGT_QHENOMENOLOGY_341: (12)                        return self.get_fill_opacity()
SNGT_QHENOMENOLOGY_342: (8)                     return self.get_stroke_opacity()
SNGT_QHENOMENOLOGY_343: (4)                 def set_flat_stroke(self, flat_stroke: bool = True,
recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_344: (8)                     self.set_uniform(recurse,
flat_stroke=float(flat_stroke))
SNGT_QHENOMENOLOGY_345: (8)                     return self
SNGT_QHENOMENOLOGY_346: (4)                 def get_flat_stroke(self) -> bool:
SNGT_QHENOMENOLOGY_347: (8)                     return self.uniforms["flat_stroke"] == 1.0
SNGT_QHENOMENOLOGY_348: (4)                 def set_scale_stroke_with_zoom(self,
scale_stroke_with_zoom: bool = True, recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_349: (8)                     self.set_uniform(recurse,
scale_stroke_with_zoom=float(scale_stroke_with_zoom))
SNGT_QHENOMENOLOGY_350: (8)                     pass
SNGT_QHENOMENOLOGY_351: (4)                 def get_scale_stroke_with_zoom(self) -> bool:
SNGT_QHENOMENOLOGY_352: (8)                     return self.uniforms["flat_stroke"] == 1.0
SNGT_QHENOMENOLOGY_353: (4)                 def set_joint_type(self, joint_type: str, recurse: bool
= True) -> Self:
SNGT_QHENOMENOLOGY_354: (8)                     for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_355: (12)                        mob.uniforms["joint_type"] =
self.joint_type_map[joint_type]
SNGT_QHENOMENOLOGY_356: (8)                     return self
SNGT_QHENOMENOLOGY_357: (4)                 def get_joint_type(self) -> float:
SNGT_QHENOMENOLOGY_358: (8)                     return self.uniforms["joint_type"]
SNGT_QHENOMENOLOGY_359: (4)                 def apply_depth_test(
SNGT_QHENOMENOLOGY_360: (8)                     self,
SNGT_QHENOMENOLOGY_361: (8)                     anti_alias_width: float = 0,
SNGT_QHENOMENOLOGY_362: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_363: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_364: (8)                     super().apply_depth_test(recurse)
SNGT_QHENOMENOLOGY_365: (8)                     self.set_anti_alias_width(anti_alias_width)
SNGT_QHENOMENOLOGY_366: (8)                     return self
SNGT_QHENOMENOLOGY_367: (4)                 def deactivate_depth_test(
SNGT_QHENOMENOLOGY_368: (8)                     self,
SNGT_QHENOMENOLOGY_369: (8)                     anti_alias_width: float = 1.0,
SNGT_QHENOMENOLOGY_370: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_371: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_372: (8)                     super().deactivate_depth_test(recurse)
SNGT_QHENOMENOLOGY_373: (8)                     self.set_anti_alias_width(anti_alias_width)
SNGT_QHENOMENOLOGY_374: (8)                     return self
SNGT_QHENOMENOLOGY_375: (4)                 def use_winding_fill(self, value: bool = True, recurse:
bool = True) -> Self:
SNGT_QHENOMENOLOGY_376: (8)                     return self
SNGT_QHENOMENOLOGY_377: (4)                 def set_anchors_and_handles(
SNGT_QHENOMENOLOGY_378: (8)                     self,
SNGT_QHENOMENOLOGY_379: (8)                     anchors: Vect3Array,
SNGT_QHENOMENOLOGY_380: (8)                     handles: Vect3Array,
SNGT_QHENOMENOLOGY_381: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_382: (8)                     if len(anchors) == 0:
SNGT_QHENOMENOLOGY_383: (12)                        self.clear_points()
SNGT_QHENOMENOLOGY_384: (12)                        return self
```

```
SNGT_QHENOMENOLOGY_385: (8)                     assert len(anchors) == len(handles) + 1
SNGT_QHENOMENOLOGY_386: (8)                     points = resize_array(self.get_points(), 2 *
len(anchors) - 1)
SNGT_QHENOMENOLOGY_387: (8)                     points[0::2] = anchors
SNGT_QHENOMENOLOGY_388: (8)                     points[1::2] = handles
SNGT_QHENOMENOLOGY_389: (8)                     self.set_points(points)
SNGT_QHENOMENOLOGY_390: (8)                     return self
SNGT_QHENOMENOLOGY_391: (4)                 def start_new_path(self, point: Vect3) -> Self:
SNGT_QHENOMENOLOGY_392: (8)                     if self.has_points():
SNGT_QHENOMENOLOGY_393: (12)                        self.append_points([self.get_last_point(),
point])
SNGT_QHENOMENOLOGY_394: (8)                     else:
SNGT_QHENOMENOLOGY_395: (12)                        self.set_points([point])
SNGT_QHENOMENOLOGY_396: (8)                     return self
SNGT_QHENOMENOLOGY_397: (4)                 def add_cubic_bezier_curve(
SNGT_QHENOMENOLOGY_398: (8)                     self,
SNGT_QHENOMENOLOGY_399: (8)                     anchor1: Vect3,
SNGT_QHENOMENOLOGY_400: (8)                     handle1: Vect3,
SNGT_QHENOMENOLOGY_401: (8)                     handle2: Vect3,
SNGT_QHENOMENOLOGY_402: (8)                     anchor2: Vect3
SNGT_QHENOMENOLOGY_403: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_404: (8)                     self.start_new_path(anchor1)
SNGT_QHENOMENOLOGY_405: (8)                     self.add_cubic_bezier_curve_to(handle1, handle2,
anchor2)
SNGT_QHENOMENOLOGY_406: (8)                     return self
SNGT_QHENOMENOLOGY_407: (4)                 def add_cubic_bezier_curve_to(
SNGT_QHENOMENOLOGY_408: (8)                     self,
SNGT_QHENOMENOLOGY_409: (8)                     handle1: Vect3,
SNGT_QHENOMENOLOGY_410: (8)                     handle2: Vect3,
SNGT_QHENOMENOLOGY_411: (8)                     anchor: Vect3,
SNGT_QHENOMENOLOGY_412: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_413: (8)                     """
SNGT_QHENOMENOLOGY_414: (8)                     Add cubic bezier curve to the path.
SNGT_QHENOMENOLOGY_415: (8)                     """
SNGT_QHENOMENOLOGY_416: (8)                     self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_417: (8)                     last = self.get_last_point()
SNGT_QHENOMENOLOGY_418: (8)                     v1 = handle1 - last
SNGT_QHENOMENOLOGY_419: (8)                     v2 = anchor - handle2
SNGT_QHENOMENOLOGY_420: (8)                     angle = angle_between_vectors(v1, v2)
SNGT_QHENOMENOLOGY_421: (8)                     if self.use_simple_quadratic_approx and angle < 45
* DEG:
SNGT_QHENOMENOLOGY_422: (12)                        quad_approx = [last, find_intersection(last,
v1, anchor, -v2), anchor]
SNGT_QHENOMENOLOGY_423: (8)                     else:
SNGT_QHENOMENOLOGY_424: (12)                        quad_approx =
get_quadratic_approximation_of_cubic(
SNGT_QHENOMENOLOGY_425: (16)                            last, handle1, handle2, anchor
SNGT_QHENOMENOLOGY_426: (12)                        )
SNGT_QHENOMENOLOGY_427: (8)                     if self.consider_points_equal(quad_approx[1],
last):
SNGT_QHENOMENOLOGY_428: (12)                        quad_approx[1] = midpoint(*quad_approx[1:3])
SNGT_QHENOMENOLOGY_429: (8)                     self.append_points(quad_approx[1:])
SNGT_QHENOMENOLOGY_430: (8)                     return self
SNGT_QHENOMENOLOGY_431: (4)                 def add_quadratic_bezier_curve_to(self, handle: Vect3,
anchor: Vect3, allow_null_curve=True) -> Self:
SNGT_QHENOMENOLOGY_432: (8)                     self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_433: (8)                     last_point = self.get_last_point()
SNGT_QHENOMENOLOGY_434: (8)                     if not allow_null_curve and
self.consider_points_equal(last_point, anchor):
SNGT_QHENOMENOLOGY_435: (12)                        return self
SNGT_QHENOMENOLOGY_436: (8)                     if self.consider_points_equal(handle, last_point):
SNGT_QHENOMENOLOGY_437: (12)                        handle = midpoint(handle, anchor)
SNGT_QHENOMENOLOGY_438: (8)                     self.append_points([handle, anchor])
SNGT_QHENOMENOLOGY_439: (8)                     return self
SNGT_QHENOMENOLOGY_440: (4)                 def add_line_to(self, point: Vect3, allow_null_line:
bool = True) -> Self:
SNGT_QHENOMENOLOGY_441: (8)                     self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_442: (8)                     last_point = self.get_last_point()
SNGT_QHENOMENOLOGY_443: (8)                     if not allow_null_line and
```

```
self.consider_points_equal(last_point, point):
SNGT_QHENOMENOLOGY_444: (12)                         return self
SNGT_QHENOMENOLOGY_445: (8)                     alphas = np.linspace(0, 1, 5 if self.long_lines
else 3)
SNGT_QHENOMENOLOGY_446: (8)                     self.append_points(outer_interpolate(last_point,
point, alphas[1:]))
SNGT_QHENOMENOLOGY_447: (8)                     return self
SNGT_QHENOMENOLOGY_448: (4)                 def add_smooth_curve_to(self, point: Vect3) -> Self:
SNGT_QHENOMENOLOGY_449: (8)                     if self.has_new_path_started():
SNGT_QHENOMENOLOGY_450: (12)                         self.add_line_to(point)
SNGT_QHENOMENOLOGY_451: (8)                     else:
SNGT_QHENOMENOLOGY_452: (12)                         self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_453: (12)                         new_handle =
self.get_reflection_of_last_handle()
SNGT_QHENOMENOLOGY_454: (12)                         self.add_quadratic_bezier_curve_to(new_handle,
point)
SNGT_QHENOMENOLOGY_455: (8)                     return self
SNGT_QHENOMENOLOGY_456: (4)                 def add_smooth_cubic_curve_to(self, handle: Vect3,
point: Vect3) -> Self:
SNGT_QHENOMENOLOGY_457: (8)                     self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_458: (8)                     if self.get_num_points() == 1:
SNGT_QHENOMENOLOGY_459: (12)                         new_handle = handle
SNGT_QHENOMENOLOGY_460: (8)                     else:
SNGT_QHENOMENOLOGY_461: (12)                         new_handle =
self.get_reflection_of_last_handle()
SNGT_QHENOMENOLOGY_462: (8)                     self.add_cubic_bezier_curve_to(new_handle, handle,
point)
SNGT_QHENOMENOLOGY_463: (8)                     return self
SNGT_QHENOMENOLOGY_464: (4)                 def add_arc_to(self, point: Vect3, angle: float,
n_components: int | None = None, threshold: float = 1e-3) -> Self:
SNGT_QHENOMENOLOGY_465: (8)                     self.throw_error_if_no_points()
SNGT_QHENOMENOLOGY_466: (8)                     if abs(angle) < threshold:
SNGT_QHENOMENOLOGY_467: (12)                         self.add_line_to(point)
SNGT_QHENOMENOLOGY_468: (12)                         return self
SNGT_QHENOMENOLOGY_469: (8)                     if n_components is None:
SNGT_QHENOMENOLOGY_470: (12)                         n_components = int(np.ceil(8 * abs(angle) /
TAU))
SNGT_QHENOMENOLOGY_471: (8)                     arc_points = quadratic_bezier_points_for_arc(angle,
n_components)
SNGT_QHENOMENOLOGY_472: (8)                     target_vect = point - self.get_end()
SNGT_QHENOMENOLOGY_473: (8)                     curr_vect = arc_points[-1] - arc_points[0]
SNGT_QHENOMENOLOGY_474: (8)                     arc_points = arc_points @
rotation_between_vectors(curr_vect, target_vect).T
SNGT_QHENOMENOLOGY_475: (8)                     arc_points *= get_norm(target_vect) /
get_norm(curr_vect)
SNGT_QHENOMENOLOGY_476: (8)                     arc_points += (self.get_end() - arc_points[0])
SNGT_QHENOMENOLOGY_477: (8)                     self.append_points(arc_points[1:])
SNGT_QHENOMENOLOGY_478: (8)                     return self
SNGT_QHENOMENOLOGY_479: (4)                 def has_new_path_started(self) -> bool:
SNGT_QHENOMENOLOGY_480: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_481: (8)                     if len(points) == 0:
SNGT_QHENOMENOLOGY_482: (12)                         return False
SNGT_QHENOMENOLOGY_483: (8)                     elif len(points) == 1:
SNGT_QHENOMENOLOGY_484: (12)                         return True
SNGT_QHENOMENOLOGY_485: (8)                     return self.consider_points_equal(points[-3],
points[-2])
SNGT_QHENOMENOLOGY_486: (4)                 def get_last_point(self) -> Vect3:
SNGT_QHENOMENOLOGY_487: (8)                     return self.get_points()[-1]
SNGT_QHENOMENOLOGY_488: (4)                 def get_reflection_of_last_handle(self) -> Vect3:
SNGT_QHENOMENOLOGY_489: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_490: (8)                     return 2 * points[-1] - points[-2]
SNGT_QHENOMENOLOGY_491: (4)                 def close_path(self, smooth: bool = False) -> Self:
SNGT_QHENOMENOLOGY_492: (8)                     if self.is_closed():
SNGT_QHENOMENOLOGY_493: (12)                         return self
SNGT_QHENOMENOLOGY_494: (8)                     ends = self.get_subpath_end_indices()
SNGT_QHENOMENOLOGY_495: (8)                     last_path_start = self.get_points()[0 if len(ends)
== 1 else ends[-2] + 2]
SNGT_QHENOMENOLOGY_496: (8)                     if smooth:
SNGT_QHENOMENOLOGY_497: (12)                         self.add_smooth_curve_to(last_path_start)
```

```
SNGT_QHENOMENOLOGY_498: (8)                     else:
SNGT_QHENOMENOLOGY_499: (12)                        self.add_line_to(last_path_start)
SNGT_QHENOMENOLOGY_500: (8)                     return self
SNGT_QHENOMENOLOGY_501: (4)                 def is_closed(self) -> bool:
SNGT_QHENOMENOLOGY_502: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_503: (8)                     ends = self.get_subpath_end_indices()
SNGT_QHENOMENOLOGY_504: (8)                     last_path_start = points[0 if len(ends) == 1 else
ends[-2] + 2]
SNGT_QHENOMENOLOGY_505: (8)                     return self.consider_points_equal(last_path_start,
points[-1])
SNGT_QHENOMENOLOGY_506: (4)                 def subdivide_curves_by_condition(
SNGT_QHENOMENOLOGY_507: (8)                     self,
SNGT_QHENOMENOLOGY_508: (8)                     tuple_to_subdivisions: Callable,
SNGT_QHENOMENOLOGY_509: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_510: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_511: (8)                     for vmob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_512: (12)                        if not vmob.has_points():
SNGT_QHENOMENOLOGY_513: (16)                            continue
SNGT_QHENOMENOLOGY_514: (12)                        new_points = [vmob.get_points()[0]]
SNGT_QHENOMENOLOGY_515: (12)                        for tup in vmob.get_bezier_tuples():
SNGT_QHENOMENOLOGY_516: (16)                            n_divisions = tuple_to_subdivisions(*tup)
SNGT_QHENOMENOLOGY_517: (16)                            if n_divisions > 0:
SNGT_QHENOMENOLOGY_518: (20)                                alphas = np.linspace(0, 1, n_divisions
+ 2)
SNGT_QHENOMENOLOGY_519: (20)                                new_points.extend([
SNGT_QHENOMENOLOGY_520: (24)
partial_quadratic_bezier_points(tup, a1, a2)[1:]
SNGT_QHENOMENOLOGY_521: (24)                                    for a1, a2 in zip(alphas,
alphas[1:])
SNGT_QHENOMENOLOGY_522: (20)                                ])
SNGT_QHENOMENOLOGY_523: (16)                            else:
SNGT_QHENOMENOLOGY_524: (20)                                new_points.append(tup[1:])
SNGT_QHENOMENOLOGY_525: (12)                        vmob.set_points(np.vstack(new_points))
SNGT_QHENOMENOLOGY_526: (8)                     return self
SNGT_QHENOMENOLOGY_527: (4)                 def subdivide_sharp_curves(
SNGT_QHENOMENOLOGY_528: (8)                     self,
SNGT_QHENOMENOLOGY_529: (8)                     angle_threshold: float = 30 * DEG,
SNGT_QHENOMENOLOGY_530: (8)                     recurse: bool = True
SNGT_QHENOMENOLOGY_531: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_532: (8)                     def tuple_to_subdivisions(b0, b1, b2):
SNGT_QHENOMENOLOGY_533: (12)                        angle = angle_between_vectors(b1 - b0, b2 - b1)
SNGT_QHENOMENOLOGY_534: (12)                        return int(angle / angle_threshold)
SNGT_QHENOMENOLOGY_535: (8)
self.subdivide_curves_by_condition(tuple_to_subdivisions, recurse)
SNGT_QHENOMENOLOGY_536: (8)                     return self
SNGT_QHENOMENOLOGY_537: (4)                 def subdivide_intersections(self, recurse: bool = True,
n_subdivisions: int = 1) -> Self:
SNGT_QHENOMENOLOGY_538: (8)                     path = self.get_anchors()
SNGT_QHENOMENOLOGY_539: (8)                     def tuple_to_subdivisions(b0, b1, b2):
SNGT_QHENOMENOLOGY_540: (12)                        if line_intersects_path(b0, b1, path):
SNGT_QHENOMENOLOGY_541: (16)                            return n_subdivisions
SNGT_QHENOMENOLOGY_542: (12)                        return 0
SNGT_QHENOMENOLOGY_543: (8)
self.subdivide_curves_by_condition(tuple_to_subdivisions, recurse)
SNGT_QHENOMENOLOGY_544: (8)                     return self
SNGT_QHENOMENOLOGY_545: (4)                 def add_points_as_corners(self, points:
Iterable[Vect3]) -> Self:
SNGT_QHENOMENOLOGY_546: (8)                     for point in points:
SNGT_QHENOMENOLOGY_547: (12)                        self.add_line_to(point)
SNGT_QHENOMENOLOGY_548: (8)                     return self
SNGT_QHENOMENOLOGY_549: (4)                 def set_points_as_corners(self, points:
Iterable[Vect3]) -> Self:
SNGT_QHENOMENOLOGY_550: (8)                     anchors = np.array(points)
SNGT_QHENOMENOLOGY_551: (8)                     handles = 0.5 * (anchors[:-1] + anchors[1:])
SNGT_QHENOMENOLOGY_552: (8)                     self.set_anchors_and_handles(anchors, handles)
SNGT_QHENOMENOLOGY_553: (8)                     return self
SNGT_QHENOMENOLOGY_554: (4)                 def set_points_smoothly(
SNGT_QHENOMENOLOGY_555: (8)                     self,
SNGT_QHENOMENOLOGY_556: (8)                     points: Iterable[Vect3],
```

```
SNGT_QHENOMENOLOGY_557: (8)                     approx: bool = True
SNGT_QHENOMENOLOGY_558: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_559: (8)                     self.set_points_as_corners(points)
SNGT_QHENOMENOLOGY_560: (8)                     self.make_smooth(approx=approx)
SNGT_QHENOMENOLOGY_561: (8)                     return self
SNGT_QHENOMENOLOGY_562: (4)                 def is_smooth(self, angle_tol=1 * DEG) -> bool:
SNGT_QHENOMENOLOGY_563: (8)                     angles = np.abs(self.get_joint_angles()[0::2])
SNGT_QHENOMENOLOGY_564: (8)                     return (angles < angle_tol).all()
SNGT_QHENOMENOLOGY_565: (4)                 def change_anchor_mode(self, mode: str) -> Self:
SNGT_QHENOMENOLOGY_566: (8)                     assert mode in ("jagged", "approx_smooth",
"true_smooth")
SNGT_QHENOMENOLOGY_567: (8)                     if self.get_num_points() == 0:
SNGT_QHENOMENOLOGY_568: (12)                        return self
SNGT_QHENOMENOLOGY_569: (8)                     subpaths = self.get_subpaths()
SNGT_QHENOMENOLOGY_570: (8)                     self.clear_points()
SNGT_QHENOMENOLOGY_571: (8)                     for subpath in subpaths:
SNGT_QHENOMENOLOGY_572: (12)                        anchors = subpath[::2]
SNGT_QHENOMENOLOGY_573: (12)                        new_subpath = np.array(subpath)
SNGT_QHENOMENOLOGY_574: (12)                        if mode == "jagged":
SNGT_QHENOMENOLOGY_575: (16)                            new_subpath[1::2] = 0.5 * (anchors[:-1] +
anchors[1:])
SNGT_QHENOMENOLOGY_576: (12)                        elif mode == "approx_smooth":
SNGT_QHENOMENOLOGY_577: (16)                            new_subpath[1::2] =
approx_smooth_quadratic_bezier_handles(anchors)
SNGT_QHENOMENOLOGY_578: (12)                        elif mode == "true_smooth":
SNGT_QHENOMENOLOGY_579: (16)                            new_subpath =
smooth_quadratic_path(anchors)
SNGT_QHENOMENOLOGY_580: (12)                        a0 = new_subpath[0:-1:2]
SNGT_QHENOMENOLOGY_581: (12)                        h = new_subpath[1::2]
SNGT_QHENOMENOLOGY_582: (12)                        a1 = new_subpath[2::2]
SNGT_QHENOMENOLOGY_583: (12)                        false_ends = np.equal(a0, h).all(1)
SNGT_QHENOMENOLOGY_584: (12)                        h[false_ends] = 0.5 * (a0[false_ends] +
a1[false_ends])
SNGT_QHENOMENOLOGY_585: (12)                        self.add_subpath(new_subpath)
SNGT_QHENOMENOLOGY_586: (8)                     return self
SNGT_QHENOMENOLOGY_587: (4)                 def make_smooth(self, approx=True, recurse=True) ->
Self:
SNGT_QHENOMENOLOGY_588: (8)                     """
SNGT_QHENOMENOLOGY_589: (8)                     Edits the path so as to pass smoothly through all
SNGT_QHENOMENOLOGY_590: (8)                     the current anchor points.
SNGT_QHENOMENOLOGY_591: (8)                     If approx is False, this may increase the total
SNGT_QHENOMENOLOGY_592: (8)                     number of points.
SNGT_QHENOMENOLOGY_593: (8)                     """
SNGT_QHENOMENOLOGY_594: (8)                     mode = "approx_smooth" if approx else "true_smooth"
SNGT_QHENOMENOLOGY_595: (8)                     for submob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_596: (12)                        if submob.is_smooth():
SNGT_QHENOMENOLOGY_597: (16)                            continue
SNGT_QHENOMENOLOGY_598: (12)                        submob.change_anchor_mode(mode)
SNGT_QHENOMENOLOGY_599: (8)                     return self
SNGT_QHENOMENOLOGY_600: (4)                 def make_approximately_smooth(self, recurse=True) ->
Self:
SNGT_QHENOMENOLOGY_601: (8)                     self.make_smooth(approx=True, recurse=recurse)
SNGT_QHENOMENOLOGY_602: (8)                     return self
SNGT_QHENOMENOLOGY_603: (4)                 def make_jagged(self, recurse=True) -> Self:
SNGT_QHENOMENOLOGY_604: (8)                     for submob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_605: (12)                        submob.change_anchor_mode("jagged")
SNGT_QHENOMENOLOGY_606: (8)                     return self
SNGT_QHENOMENOLOGY_607: (4)                 def add_subpath(self, points: Vect3Array) -> Self:
SNGT_QHENOMENOLOGY_608: (8)                     assert len(points) % 2 == 1 or len(points) == 0
SNGT_QHENOMENOLOGY_609: (8)                     if not self.has_points():
SNGT_QHENOMENOLOGY_610: (12)                        self.set_points(points)
SNGT_QHENOMENOLOGY_611: (12)                        return self
SNGT_QHENOMENOLOGY_612: (8)                     if not self.consider_points_equal(points[0],
self.get_points()[-1]):
SNGT_QHENOMENOLOGY_613: (12)                        self.start_new_path(points[0])
SNGT_QHENOMENOLOGY_614: (8)                     self.append_points(points[1:])
SNGT_QHENOMENOLOGY_615: (8)                     return self
SNGT_QHENOMENOLOGY_616: (4)                 def append_vectorized_mobject(self, vmobject: VMobject)
-> Self:
```

```
SNGT_QHENOMENOLOGY_617: (8)                     self.add_subpath(vmobject.get_points())
SNGT_QHENOMENOLOGY_618: (8)                     n = vmobject.get_num_points()
SNGT_QHENOMENOLOGY_619: (8)                     self.data[-n:] = vmobject.data
SNGT_QHENOMENOLOGY_620: (8)                     return self
SNGT_QHENOMENOLOGY_621: (4)                 def consider_points_equal(self, p0: Vect3, p1: Vect3) -
> bool:
SNGT_QHENOMENOLOGY_622: (8)                     return all(abs(p1 - p0) <
self.tolerance_for_point_equality)
SNGT_QHENOMENOLOGY_623: (4)                 def get_bezier_tuples_from_points(self, points:
Vect3Array) -> Iterable[Vect3Array]:
SNGT_QHENOMENOLOGY_624: (8)                     n_curves = (len(points) - 1) // 2
SNGT_QHENOMENOLOGY_625: (8)                     return (points[2 * i:2 * i + 3] for i in
range(n_curves))
SNGT_QHENOMENOLOGY_626: (4)                 def get_bezier_tuples(self) -> Iterable[Vect3Array]:
SNGT_QHENOMENOLOGY_627: (8)                     return
self.get_bezier_tuples_from_points(self.get_points())
SNGT_QHENOMENOLOGY_628: (4)                 def get_subpath_end_indices_from_points(self, points:
Vect3Array) -> np.ndarray:
SNGT_QHENOMENOLOGY_629: (8)                     atol = 1e-4  # TODO, this is too unsystematic
SNGT_QHENOMENOLOGY_630: (8)                     a0, h, a1 = points[0:-1:2], points[1::2],
points[2::2]
SNGT_QHENOMENOLOGY_631: (8)                     is_end = (a0 == h).all(1) & (abs(h - a1) >
atol).any(1)
SNGT_QHENOMENOLOGY_632: (8)                     end_indices = (2 * n for n, end in
enumerate(is_end) if end)
SNGT_QHENOMENOLOGY_633: (8)                     return np.array([*end_indices, len(points) - 1])
SNGT_QHENOMENOLOGY_634: (4)                 def get_subpath_end_indices(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_635: (8)                     if self.subpath_end_indices is None:
SNGT_QHENOMENOLOGY_636: (12)                        self.subpath_end_indices =
self.get_subpath_end_indices_from_points(self.get_points())
SNGT_QHENOMENOLOGY_637: (8)                     return self.subpath_end_indices
SNGT_QHENOMENOLOGY_638: (4)                 def get_subpaths_from_points(self, points: Vect3Array)
-> list[Vect3Array]:
SNGT_QHENOMENOLOGY_639: (8)                     if len(points) == 0:
SNGT_QHENOMENOLOGY_640: (12)                        return []
SNGT_QHENOMENOLOGY_641: (8)                     end_indices =
self.get_subpath_end_indices_from_points(points)
SNGT_QHENOMENOLOGY_642: (8)                     start_indices = [0, *(end_indices[:-1] + 2)]
SNGT_QHENOMENOLOGY_643: (8)                     return [points[i1:i2 + 1] for i1, i2 in
zip(start_indices, end_indices)]
SNGT_QHENOMENOLOGY_644: (4)                 def get_subpaths(self) -> list[Vect3Array]:
SNGT_QHENOMENOLOGY_645: (8)                     return
self.get_subpaths_from_points(self.get_points())
SNGT_QHENOMENOLOGY_646: (4)                 def get_nth_curve_points(self, n: int) -> Vect3Array:
SNGT_QHENOMENOLOGY_647: (8)                     assert n < self.get_num_curves()
SNGT_QHENOMENOLOGY_648: (8)                     return self.get_points()[2 * n:2 * n + 3]
SNGT_QHENOMENOLOGY_649: (4)                 def get_nth_curve_function(self, n: int) ->
Callable[[float], Vect3]:
SNGT_QHENOMENOLOGY_650: (8)                     return bezier(self.get_nth_curve_points(n))
SNGT_QHENOMENOLOGY_651: (4)                 def get_num_curves(self) -> int:
SNGT_QHENOMENOLOGY_652: (8)                     return self.get_num_points() // 2
SNGT_QHENOMENOLOGY_653: (4)                 def quick_point_from_proportion(self, alpha: float) ->
Vect3:
SNGT_QHENOMENOLOGY_654: (8)                     num_curves = self.get_num_curves()
SNGT_QHENOMENOLOGY_655: (8)                     if num_curves == 0:
SNGT_QHENOMENOLOGY_656: (12)                        return self.get_center()
SNGT_QHENOMENOLOGY_657: (8)                     n, residue = integer_interpolate(0, num_curves,
alpha)
SNGT_QHENOMENOLOGY_658: (8)                     curve_func = self.get_nth_curve_function(n)
SNGT_QHENOMENOLOGY_659: (8)                     return curve_func(residue)
SNGT_QHENOMENOLOGY_660: (4)                 def curve_and_prop_of_partial_point(self, alpha) ->
Tuple[int, float]:
SNGT_QHENOMENOLOGY_661: (8)                     """
SNGT_QHENOMENOLOGY_662: (8)                     If you want a point a proportion alpha along the
curve, this
SNGT_QHENOMENOLOGY_663: (8)                     gives you the index of the appropriate bezier
curve, together
SNGT_QHENOMENOLOGY_664: (8)                     with the proportion along that curve you'd need to
travel
```

```
SNGT_QHENOMENOLOGY_665: (8)                     """
SNGT_QHENOMENOLOGY_666: (8)                     if alpha == 0:
SNGT_QHENOMENOLOGY_667: (12)                        return (0, 0.0)
SNGT_QHENOMENOLOGY_668: (8)                     partials: list[float] = [0]
SNGT_QHENOMENOLOGY_669: (8)                     for tup in self.get_bezier_tuples():
SNGT_QHENOMENOLOGY_670: (12)                        if self.consider_points_equal(tup[0], tup[1]):
SNGT_QHENOMENOLOGY_671: (16)                            arclen = 0
SNGT_QHENOMENOLOGY_672: (12)                        else:
SNGT_QHENOMENOLOGY_673: (16)                            arclen = get_norm(tup[2] - tup[0])
SNGT_QHENOMENOLOGY_674: (12)                        partials.append(partials[-1] + arclen)
SNGT_QHENOMENOLOGY_675: (8)                     full = partials[-1]
SNGT_QHENOMENOLOGY_676: (8)                     if full == 0:
SNGT_QHENOMENOLOGY_677: (12)                        return len(partials), 1.0
SNGT_QHENOMENOLOGY_678: (8)                     index = next(
SNGT_QHENOMENOLOGY_679: (12)                        (i for i, x in enumerate(partials) if x >= full
* alpha),
SNGT_QHENOMENOLOGY_680: (12)                        len(partials) - 1  # Default
SNGT_QHENOMENOLOGY_681: (8)                     )
SNGT_QHENOMENOLOGY_682: (8)                     residue = float(inverse_interpolate(
SNGT_QHENOMENOLOGY_683: (12)                        partials[index - 1] / full, partials[index] /
full, alpha
SNGT_QHENOMENOLOGY_684: (8)                     ))
SNGT_QHENOMENOLOGY_685: (8)                     return index - 1, residue
SNGT_QHENOMENOLOGY_686: (4)                 def point_from_proportion(self, alpha: float) -> Vect3:
SNGT_QHENOMENOLOGY_687: (8)                     if alpha <= 0:
SNGT_QHENOMENOLOGY_688: (12)                        return self.get_start()
SNGT_QHENOMENOLOGY_689: (8)                     elif alpha >= 1:
SNGT_QHENOMENOLOGY_690: (12)                        return self.get_end()
SNGT_QHENOMENOLOGY_691: (8)                     if self.get_num_points() == 0:
SNGT_QHENOMENOLOGY_692: (12)                        return self.get_center()
SNGT_QHENOMENOLOGY_693: (8)                     index, residue =
self.curve_and_prop_of_partial_point(alpha)
SNGT_QHENOMENOLOGY_694: (8)                     return self.get_nth_curve_function(index)(residue)
SNGT_QHENOMENOLOGY_695: (4)                 def get_anchors_and_handles(self) -> list[Vect3]:
SNGT_QHENOMENOLOGY_696: (8)                     """
SNGT_QHENOMENOLOGY_697: (8)                     returns anchors1, handles, anchors2,
SNGT_QHENOMENOLOGY_698: (8)                     where (anchors1[i], handles[i], anchors2[i])
SNGT_QHENOMENOLOGY_699: (8)                     will be three points defining a quadratic bezier
curve
SNGT_QHENOMENOLOGY_700: (8)                     for any i in range(0, len(anchors1))
SNGT_QHENOMENOLOGY_701: (8)                     """
SNGT_QHENOMENOLOGY_702: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_703: (8)                     return [points[0:-1:2], points[1::2], points[2::2]]
SNGT_QHENOMENOLOGY_704: (4)                 def get_start_anchors(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_705: (8)                     return self.get_points()[0:-1:2]
SNGT_QHENOMENOLOGY_706: (4)                 def get_end_anchors(self) -> Vect3:
SNGT_QHENOMENOLOGY_707: (8)                     return self.get_points()[2::2]
SNGT_QHENOMENOLOGY_708: (4)                 def get_anchors(self) -> Vect3Array:
SNGT_QHENOMENOLOGY_709: (8)                     return self.get_points()[::2]
SNGT_QHENOMENOLOGY_710: (4)                 def get_points_without_null_curves(self, atol: float =
1e-9) -> Vect3Array:
SNGT_QHENOMENOLOGY_711: (8)                     new_points = [self.get_points()[0]]
SNGT_QHENOMENOLOGY_712: (8)                     for tup in self.get_bezier_tuples():
SNGT_QHENOMENOLOGY_713: (12)                        if get_norm(tup[1] - tup[0]) > atol or
get_norm(tup[2] - tup[0]) > atol:
SNGT_QHENOMENOLOGY_714: (16)                            new_points.append(tup[1:])
SNGT_QHENOMENOLOGY_715: (8)                     return np.vstack(new_points)
SNGT_QHENOMENOLOGY_716: (4)                 def get_arc_length(self, n_sample_points: int | None =
None) -> float:
SNGT_QHENOMENOLOGY_717: (8)                     if n_sample_points is not None:
SNGT_QHENOMENOLOGY_718: (12)                        points = np.array([
SNGT_QHENOMENOLOGY_719: (16)                            self.quick_point_from_proportion(a)
SNGT_QHENOMENOLOGY_720: (16)                            for a in np.linspace(0, 1, n_sample_points)
SNGT_QHENOMENOLOGY_721: (12)                        ])
SNGT_QHENOMENOLOGY_722: (12)                        return poly_line_length(points)
SNGT_QHENOMENOLOGY_723: (8)                     points = self.get_points()
SNGT_QHENOMENOLOGY_724: (8)                     inner_len = poly_line_length(points[::2])
SNGT_QHENOMENOLOGY_725: (8)                     outer_len = poly_line_length(points)
SNGT_QHENOMENOLOGY_726: (8)                     return interpolate(inner_len, outer_len, 1 / 3)
```

```
SNGT_QHENOMENOLOGY_727: (4)                 def get_area_vector(self) -> Vect3:
SNGT_QHENOMENOLOGY_728: (8)                     if not self.has_points():
SNGT_QHENOMENOLOGY_729: (12)                        return np.zeros(3)
SNGT_QHENOMENOLOGY_730: (8)                     p0 = self.get_anchors()
SNGT_QHENOMENOLOGY_731: (8)                     p1 = np.vstack([p0[1:], p0[0]])
SNGT_QHENOMENOLOGY_732: (8)                     sums = p0 + p1
SNGT_QHENOMENOLOGY_733: (8)                     diffs = p1 - p0
SNGT_QHENOMENOLOGY_734: (8)                     return 0.5 * np.array([
SNGT_QHENOMENOLOGY_735: (12)                        (sums[:, 1] * diffs[:, 2]).sum(),  # Add up (y0
+ y1)*(z1 - z0)
SNGT_QHENOMENOLOGY_736: (12)                        (sums[:, 2] * diffs[:, 0]).sum(),  # Add up (z0
+ z1)*(x1 - x0)
SNGT_QHENOMENOLOGY_737: (12)                        (sums[:, 0] * diffs[:, 1]).sum(),  # Add up (x0
+ x1)*(y1 - y0)
SNGT_QHENOMENOLOGY_738: (8)                     ])
SNGT_QHENOMENOLOGY_739: (4)                 def get_unit_normal(self, refresh: bool = False) ->
Vect3:
SNGT_QHENOMENOLOGY_740: (8)                     if self.get_num_points() < 3:
SNGT_QHENOMENOLOGY_741: (12)                        return OUT
SNGT_QHENOMENOLOGY_742: (8)                     if not self.needs_new_unit_normal and not refresh:
SNGT_QHENOMENOLOGY_743: (12)                        return self.data["base_normal"][1, :]
SNGT_QHENOMENOLOGY_744: (8)                     area_vect = self.get_area_vector()
SNGT_QHENOMENOLOGY_745: (8)                     area = get_norm(area_vect)
SNGT_QHENOMENOLOGY_746: (8)                     if area > 0:
SNGT_QHENOMENOLOGY_747: (12)                        normal = area_vect / area
SNGT_QHENOMENOLOGY_748: (8)                     else:
SNGT_QHENOMENOLOGY_749: (12)                        p = self.get_points()
SNGT_QHENOMENOLOGY_750: (12)                        normal = get_unit_normal(p[1] - p[0], p[2] -
p[1])
SNGT_QHENOMENOLOGY_751: (8)                     self.data["base_normal"][1::2] = normal
SNGT_QHENOMENOLOGY_752: (8)                     self.needs_new_unit_normal = False
SNGT_QHENOMENOLOGY_753: (8)                     return normal
SNGT_QHENOMENOLOGY_754: (4)                 def refresh_unit_normal(self) -> Self:
SNGT_QHENOMENOLOGY_755: (8)                     self.needs_new_unit_normal = True
SNGT_QHENOMENOLOGY_756: (8)                     return self
SNGT_QHENOMENOLOGY_757: (4)                 def rotate(
SNGT_QHENOMENOLOGY_758: (8)                     self,
SNGT_QHENOMENOLOGY_759: (8)                     angle: float,
SNGT_QHENOMENOLOGY_760: (8)                     axis: Vect3 = OUT,
SNGT_QHENOMENOLOGY_761: (8)                     about_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_762: (8)                     **kwargs
SNGT_QHENOMENOLOGY_763: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_764: (8)                     super().rotate(angle, axis, about_point, **kwargs)
SNGT_QHENOMENOLOGY_765: (8)                     for mob in self.get_family():
SNGT_QHENOMENOLOGY_766: (12)                        mob.refresh_unit_normal()
SNGT_QHENOMENOLOGY_767: (8)                     return self
SNGT_QHENOMENOLOGY_768: (4)                 def ensure_positive_orientation(self, recurse=True) ->
Self:
SNGT_QHENOMENOLOGY_769: (8)                     for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_770: (12)                        if mob.get_unit_normal()[2] < 0:
SNGT_QHENOMENOLOGY_771: (16)                            mob.reverse_points()
SNGT_QHENOMENOLOGY_772: (8)                     return self
SNGT_QHENOMENOLOGY_773: (4)                 def align_points(self, vmobject: VMobject) -> Self:
SNGT_QHENOMENOLOGY_774: (8)                     if self.get_num_points() ==
len(vmobject.get_points()):
SNGT_QHENOMENOLOGY_775: (12)                        for mob in [self, vmobject]:
SNGT_QHENOMENOLOGY_776: (16)                            mob.get_joint_angles()
SNGT_QHENOMENOLOGY_777: (12)                        return self
SNGT_QHENOMENOLOGY_778: (8)                     for mob in self, vmobject:
SNGT_QHENOMENOLOGY_779: (12)                        if not mob.has_points():
SNGT_QHENOMENOLOGY_780: (16)                            mob.start_new_path(mob.get_center())
SNGT_QHENOMENOLOGY_781: (8)                     subpaths1 = self.get_subpaths()
SNGT_QHENOMENOLOGY_782: (8)                     subpaths2 = vmobject.get_subpaths()
SNGT_QHENOMENOLOGY_783: (8)                     for subpaths in [subpaths1, subpaths2]:
SNGT_QHENOMENOLOGY_784: (12)                        subpaths.sort(key=lambda sp: -sum(
SNGT_QHENOMENOLOGY_785: (16)                            get_norm(p2 - p1)
SNGT_QHENOMENOLOGY_786: (16)                            for p1, p2 in zip(sp, sp[1:])
SNGT_QHENOMENOLOGY_787: (12)                        ))
SNGT_QHENOMENOLOGY_788: (8)                     n_subpaths = max(len(subpaths1), len(subpaths2))
```

```
SNGT_QHENOMENOLOGY_789: (8)                     new_subpaths1 = []
SNGT_QHENOMENOLOGY_790: (8)                     new_subpaths2 = []
SNGT_QHENOMENOLOGY_791: (8)                     def get_nth_subpath(path_list, n):
SNGT_QHENOMENOLOGY_792: (12)                        if n >= len(path_list):
SNGT_QHENOMENOLOGY_793: (16)                            return np.vstack([path_list[0][:-1],
path_list[0][::-1]])
SNGT_QHENOMENOLOGY_794: (12)                        return path_list[n]
SNGT_QHENOMENOLOGY_795: (8)                     for n in range(n_subpaths):
SNGT_QHENOMENOLOGY_796: (12)                        sp1 = get_nth_subpath(subpaths1, n)
SNGT_QHENOMENOLOGY_797: (12)                        sp2 = get_nth_subpath(subpaths2, n)
SNGT_QHENOMENOLOGY_798: (12)                        diff1 = max(0, (len(sp2) - len(sp1)) // 2)
SNGT_QHENOMENOLOGY_799: (12)                        diff2 = max(0, (len(sp1) - len(sp2)) // 2)
SNGT_QHENOMENOLOGY_800: (12)                        sp1 = self.insert_n_curves_to_point_list(diff1,
sp1)
SNGT_QHENOMENOLOGY_801: (12)                        sp2 = self.insert_n_curves_to_point_list(diff2,
sp2)
SNGT_QHENOMENOLOGY_802: (12)                        if n > 0:
SNGT_QHENOMENOLOGY_803: (16)                            new_subpaths1.append(new_subpaths1[-1][-1])
SNGT_QHENOMENOLOGY_804: (16)                            new_subpaths2.append(new_subpaths2[-1][-1])
SNGT_QHENOMENOLOGY_805: (12)                        new_subpaths1.append(sp1)
SNGT_QHENOMENOLOGY_806: (12)                        new_subpaths2.append(sp2)
SNGT_QHENOMENOLOGY_807: (8)                     for mob, paths in [(self, new_subpaths1),
(vmobject, new_subpaths2)]:
SNGT_QHENOMENOLOGY_808: (12)                        new_points = np.vstack(paths)
SNGT_QHENOMENOLOGY_809: (12)                        mob.resize_points(len(new_points),
resize_func=resize_preserving_order)
SNGT_QHENOMENOLOGY_810: (12)                        mob.set_points(new_points)
SNGT_QHENOMENOLOGY_811: (12)                        mob.get_joint_angles()
SNGT_QHENOMENOLOGY_812: (8)                     return self
SNGT_QHENOMENOLOGY_813: (4)                 def insert_n_curves(self, n: int, recurse: bool = True)
-> Self:
SNGT_QHENOMENOLOGY_814: (8)                     for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_815: (12)                        if mob.get_num_curves() > 0:
SNGT_QHENOMENOLOGY_816: (16)                            new_points =
mob.insert_n_curves_to_point_list(n, mob.get_points())
SNGT_QHENOMENOLOGY_817: (16)                            mob.set_points(new_points)
SNGT_QHENOMENOLOGY_818: (8)                     return self
SNGT_QHENOMENOLOGY_819: (4)                 def insert_n_curves_to_point_list(self, n: int, points:
Vect3Array) -> Vect3Array:
SNGT_QHENOMENOLOGY_820: (8)                     if len(points) == 1:
SNGT_QHENOMENOLOGY_821: (12)                        return np.repeat(points, 2 * n + 1, 0)
SNGT_QHENOMENOLOGY_822: (8)                     bezier_tuples =
list(self.get_bezier_tuples_from_points(points))
SNGT_QHENOMENOLOGY_823: (8)                     atol = self.tolerance_for_point_equality
SNGT_QHENOMENOLOGY_824: (8)                     norms = [
SNGT_QHENOMENOLOGY_825: (12)                        0 if get_norm(tup[1] - tup[0]) < atol else
get_norm(tup[2] - tup[0])
SNGT_QHENOMENOLOGY_826: (12)                        for tup in bezier_tuples
SNGT_QHENOMENOLOGY_827: (8)                     ]
SNGT_QHENOMENOLOGY_828: (8)                     ipc = np.zeros(len(bezier_tuples), dtype=int)
SNGT_QHENOMENOLOGY_829: (8)                     for _ in range(n):
SNGT_QHENOMENOLOGY_830: (12)                        index = np.argmax(norms)
SNGT_QHENOMENOLOGY_831: (12)                        ipc[index] += 1
SNGT_QHENOMENOLOGY_832: (12)                        norms[index] *= ipc[index] / (ipc[index] + 1)
SNGT_QHENOMENOLOGY_833: (8)                     new_points = [points[0]]
SNGT_QHENOMENOLOGY_834: (8)                     for tup, n_inserts in zip(bezier_tuples, ipc):
SNGT_QHENOMENOLOGY_835: (12)                        alphas = np.linspace(0, 1, n_inserts + 2)
SNGT_QHENOMENOLOGY_836: (12)                        for a1, a2 in zip(alphas, alphas[1:]):
SNGT_QHENOMENOLOGY_837: (16)
new_points.extend(partial_quadratic_bezier_points(tup, a1, a2)[1:])
SNGT_QHENOMENOLOGY_838: (8)                     return np.vstack(new_points)
SNGT_QHENOMENOLOGY_839: (4)                 def pointwise_become_partial(self, vmobject: VMobject,
a: float, b: float) -> Self:
SNGT_QHENOMENOLOGY_840: (8)                     assert isinstance(vmobject, VMobject)
SNGT_QHENOMENOLOGY_841: (8)                     vm_points = vmobject.get_points()
SNGT_QHENOMENOLOGY_842: (8)                     self.data["joint_angle"] =
vmobject.data["joint_angle"]
SNGT_QHENOMENOLOGY_843: (8)                     if a <= 0 and b >= 1:
SNGT_QHENOMENOLOGY_844: (12)                        self.set_points(vm_points, refresh=False)
```

```
SNGT_QHENOMENOLOGY_845: (12)                     return self
SNGT_QHENOMENOLOGY_846: (8)                 num_curves = vmobject.get_num_curves()
SNGT_QHENOMENOLOGY_847: (8)                 lower_index, lower_residue = integer_interpolate(0,
num_curves, a)
SNGT_QHENOMENOLOGY_848: (8)                 upper_index, upper_residue = integer_interpolate(0,
num_curves, b)
SNGT_QHENOMENOLOGY_849: (8)                 i1 = 2 * lower_index
SNGT_QHENOMENOLOGY_850: (8)                 i2 = 2 * lower_index + 3
SNGT_QHENOMENOLOGY_851: (8)                 i3 = 2 * upper_index
SNGT_QHENOMENOLOGY_852: (8)                 i4 = 2 * upper_index + 3
SNGT_QHENOMENOLOGY_853: (8)                 new_points = vm_points.copy()
SNGT_QHENOMENOLOGY_854: (8)                 if num_curves == 0:
SNGT_QHENOMENOLOGY_855: (12)                     new_points[:] = 0
SNGT_QHENOMENOLOGY_856: (12)                     return self
SNGT_QHENOMENOLOGY_857: (8)                 if lower_index == upper_index:
SNGT_QHENOMENOLOGY_858: (12)                     tup =
partial_quadratic_bezier_points(vm_points[i1:i2], lower_residue, upper_residue)
SNGT_QHENOMENOLOGY_859: (12)                     new_points[:i1] = tup[0]
SNGT_QHENOMENOLOGY_860: (12)                     new_points[i1:i4] = tup
SNGT_QHENOMENOLOGY_861: (12)                     new_points[i4:] = tup[2]
SNGT_QHENOMENOLOGY_862: (8)                 else:
SNGT_QHENOMENOLOGY_863: (12)                     low_tup =
partial_quadratic_bezier_points(vm_points[i1:i2], lower_residue, 1)
SNGT_QHENOMENOLOGY_864: (12)                     high_tup =
partial_quadratic_bezier_points(vm_points[i3:i4], 0, upper_residue)
SNGT_QHENOMENOLOGY_865: (12)                     new_points[0:i1] = low_tup[0]
SNGT_QHENOMENOLOGY_866: (12)                     new_points[i1:i2] = low_tup
SNGT_QHENOMENOLOGY_867: (12)                     new_points[i3:i4] = high_tup
SNGT_QHENOMENOLOGY_868: (12)                     new_points[i4:] = high_tup[2]
SNGT_QHENOMENOLOGY_869: (8)                 self.data["joint_angle"][:i1] = 0
SNGT_QHENOMENOLOGY_870: (8)                 self.data["joint_angle"][i4:] = 0
SNGT_QHENOMENOLOGY_871: (8)                 self.set_points(new_points, refresh=False)
SNGT_QHENOMENOLOGY_872: (8)                 return self
SNGT_QHENOMENOLOGY_873: (4)             def get_subcurve(self, a: float, b: float) -> Self:
SNGT_QHENOMENOLOGY_874: (8)                 vmob = self.copy()
SNGT_QHENOMENOLOGY_875: (8)                 vmob.pointwise_become_partial(self, a, b)
SNGT_QHENOMENOLOGY_876: (8)                 return vmob
SNGT_QHENOMENOLOGY_877: (4)             def get_outer_vert_indices(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_878: (8)                 """
SNGT_QHENOMENOLOGY_879: (8)                 Returns the pattern (0, 1, 2, 2, 3, 4, 4, 5, 6,
...)
SNGT_QHENOMENOLOGY_880: (8)                 """
SNGT_QHENOMENOLOGY_881: (8)                 n_curves = self.get_num_curves()
SNGT_QHENOMENOLOGY_882: (8)                 if len(self.outer_vert_indices) != 3 * n_curves:
SNGT_QHENOMENOLOGY_883: (12)                     self.outer_vert_indices = (np.arange(1, 3 *
n_curves + 1) * 2) // 3
SNGT_QHENOMENOLOGY_884: (8)                 return self.outer_vert_indices
SNGT_QHENOMENOLOGY_885: (4)             def get_triangulation(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_886: (8)                 points = self.get_points()
SNGT_QHENOMENOLOGY_887: (8)                 if len(points) <= 1:
SNGT_QHENOMENOLOGY_888: (12)                     return np.zeros(0, dtype='i4')
SNGT_QHENOMENOLOGY_889: (8)                 normal_vector = self.get_unit_normal()
SNGT_QHENOMENOLOGY_890: (8)                 if not np.isclose(normal_vector, OUT).all():
SNGT_QHENOMENOLOGY_891: (12)                     points = np.dot(points,
z_to_vector(normal_vector))
SNGT_QHENOMENOLOGY_892: (8)                 v01s = points[1::2] - points[0:-1:2]
SNGT_QHENOMENOLOGY_893: (8)                 v12s = points[2::2] - points[1::2]
SNGT_QHENOMENOLOGY_894: (8)                 curve_orientations = np.sign(cross2d(v01s, v12s))
SNGT_QHENOMENOLOGY_895: (8)                 concave_parts = curve_orientations < 0
SNGT_QHENOMENOLOGY_896: (8)                 indices = np.arange(len(points), dtype=int)
SNGT_QHENOMENOLOGY_897: (8)                 inner_vert_indices = np.hstack([
SNGT_QHENOMENOLOGY_898: (12)                     indices[0::2],
SNGT_QHENOMENOLOGY_899: (12)                     indices[1::2][concave_parts],
SNGT_QHENOMENOLOGY_900: (8)                 ])
SNGT_QHENOMENOLOGY_901: (8)                 inner_vert_indices.sort()
SNGT_QHENOMENOLOGY_902: (8)                 end_indices = self.get_subpath_end_indices()
SNGT_QHENOMENOLOGY_903: (8)                 counts = np.arange(1, len(inner_vert_indices) + 1)
SNGT_QHENOMENOLOGY_904: (8)                 rings = counts[inner_vert_indices % 2 == 0]
[end_indices // 2]
```

```
SNGT_QHENOMENOLOGY_905: (8)                     inner_verts = points[inner_vert_indices]
SNGT_QHENOMENOLOGY_906: (8)                     inner_tri_indices = inner_vert_indices[
SNGT_QHENOMENOLOGY_907: (12)                        earclip_triangulation(inner_verts, rings)
SNGT_QHENOMENOLOGY_908: (8)                     ]
SNGT_QHENOMENOLOGY_909: (8)                     iti = inner_tri_indices
SNGT_QHENOMENOLOGY_910: (8)                     null1 = (iti[0::3] + 1 == iti[1::3]) & (iti[0::3] +
2 == iti[2::3])
SNGT_QHENOMENOLOGY_911: (8)                     null2 = (iti[0::3] - 1 == iti[1::3]) & (iti[0::3] -
2 == iti[2::3])
SNGT_QHENOMENOLOGY_912: (8)                     inner_tri_indices = iti[~(null1 | null2).repeat(3)]
SNGT_QHENOMENOLOGY_913: (8)                     ovi = self.get_outer_vert_indices()
SNGT_QHENOMENOLOGY_914: (8)                     tri_indices = np.hstack([ovi, inner_tri_indices])
SNGT_QHENOMENOLOGY_915: (8)                     return tri_indices
SNGT_QHENOMENOLOGY_916: (4)                 def refresh_joint_angles(self) -> Self:
SNGT_QHENOMENOLOGY_917: (8)                     for mob in self.get_family():
SNGT_QHENOMENOLOGY_918: (12)                        mob.needs_new_joint_angles = True
SNGT_QHENOMENOLOGY_919: (8)                     return self
SNGT_QHENOMENOLOGY_920: (4)                 def get_joint_angles(self, refresh: bool = False) ->
np.ndarray:
SNGT_QHENOMENOLOGY_921: (8)                     """
SNGT_QHENOMENOLOGY_922: (8)                     The 'joint product' is a 4-vector holding the cross
and dot
SNGT_QHENOMENOLOGY_923: (8)                     product between tangent vectors at a joint
SNGT_QHENOMENOLOGY_924: (8)                     """
SNGT_QHENOMENOLOGY_925: (8)                     if not self.needs_new_joint_angles and not refresh:
SNGT_QHENOMENOLOGY_926: (12)                        return self.data["joint_angle"][:, 0]
SNGT_QHENOMENOLOGY_927: (8)                     if "joint_angle" in self.locked_data_keys:
SNGT_QHENOMENOLOGY_928: (12)                        return self.data["joint_angle"][:, 0]
SNGT_QHENOMENOLOGY_929: (8)                     self.needs_new_joint_angles = False
SNGT_QHENOMENOLOGY_930: (8)                     self._data_has_changed = True
SNGT_QHENOMENOLOGY_931: (8)                     points = self.get_points() @
rotation_between_vectors(OUT, self.get_unit_normal())
SNGT_QHENOMENOLOGY_932: (8)                     if len(points) < 3:
SNGT_QHENOMENOLOGY_933: (12)                        return self.data["joint_angle"][:, 0]
SNGT_QHENOMENOLOGY_934: (8)                     a0, h, a1 = points[0:-1:2], points[1::2],
points[2::2]
SNGT_QHENOMENOLOGY_935: (8)                     a0_to_h = h - a0
SNGT_QHENOMENOLOGY_936: (8)                     h_to_a1 = a1 - h
SNGT_QHENOMENOLOGY_937: (8)                     v_in = np.zeros(points.shape)
SNGT_QHENOMENOLOGY_938: (8)                     v_out = np.zeros(points.shape)
SNGT_QHENOMENOLOGY_939: (8)                     v_in[1::2] = a0_to_h
SNGT_QHENOMENOLOGY_940: (8)                     v_in[2::2] = h_to_a1
SNGT_QHENOMENOLOGY_941: (8)                     v_out[0:-1:2] = a0_to_h
SNGT_QHENOMENOLOGY_942: (8)                     v_out[1::2] = h_to_a1
SNGT_QHENOMENOLOGY_943: (8)                     ends = self.get_subpath_end_indices()
SNGT_QHENOMENOLOGY_944: (8)                     starts = [0, *(e + 2 for e in ends[:-1])]
SNGT_QHENOMENOLOGY_945: (8)                     for start, end in zip(starts, ends):
SNGT_QHENOMENOLOGY_946: (12)                        if start == end:
SNGT_QHENOMENOLOGY_947: (16)                            continue
SNGT_QHENOMENOLOGY_948: (12)                        if (points[start] == points[end]).all():
SNGT_QHENOMENOLOGY_949: (16)                            v_in[start] = v_out[end - 1]
SNGT_QHENOMENOLOGY_950: (16)                            v_out[end] = v_in[start + 1]
SNGT_QHENOMENOLOGY_951: (12)                        else:
SNGT_QHENOMENOLOGY_952: (16)                            v_in[start] = v_out[start]
SNGT_QHENOMENOLOGY_953: (16)                            v_out[end] = v_in[end]
SNGT_QHENOMENOLOGY_954: (8)                     angles_in = np.arctan2(v_in[:, 1], v_in[:, 0])
SNGT_QHENOMENOLOGY_955: (8)                     angles_out = np.arctan2(v_out[:, 1], v_out[:, 0])
SNGT_QHENOMENOLOGY_956: (8)                     angle_diffs = angles_out - angles_in
SNGT_QHENOMENOLOGY_957: (8)                     angle_diffs[angle_diffs < -PI] += TAU
SNGT_QHENOMENOLOGY_958: (8)                     angle_diffs[angle_diffs > PI] -= TAU
SNGT_QHENOMENOLOGY_959: (8)                     self.data["joint_angle"][:, 0] = angle_diffs
SNGT_QHENOMENOLOGY_960: (8)                     return self.data["joint_angle"][:, 0]
SNGT_QHENOMENOLOGY_961: (4)                 def lock_matching_data(self, vmobject1: VMobject,
vmobject2: VMobject) -> Self:
SNGT_QHENOMENOLOGY_962: (8)                     for mob in [self, vmobject1, vmobject2]:
SNGT_QHENOMENOLOGY_963: (12)                        mob.get_joint_angles()
SNGT_QHENOMENOLOGY_964: (8)                     super().lock_matching_data(vmobject1, vmobject2)
SNGT_QHENOMENOLOGY_965: (8)                     return self
SNGT_QHENOMENOLOGY_966: (4)                 def triggers_refresh(func: Callable):
```

```
SNGT_QHENOMENOLOGY_967: (8)                     @wraps(func)
SNGT_QHENOMENOLOGY_968: (8)                     def wrapper(self, *args, refresh=True, **kwargs):
SNGT_QHENOMENOLOGY_969: (12)                        func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_970: (12)                        if refresh:
SNGT_QHENOMENOLOGY_971: (16)                            self.subpath_end_indices = None
SNGT_QHENOMENOLOGY_972: (16)                            self.refresh_joint_angles()
SNGT_QHENOMENOLOGY_973: (16)                            self.refresh_unit_normal()
SNGT_QHENOMENOLOGY_974: (12)                        return self
SNGT_QHENOMENOLOGY_975: (8)                     return wrapper
SNGT_QHENOMENOLOGY_976: (4)                 @triggers_refresh
SNGT_QHENOMENOLOGY_977: (4)                 def set_points(self, points: Vect3Array) -> Self:
SNGT_QHENOMENOLOGY_978: (8)                     assert len(points) == 0 or len(points) % 2 == 1
SNGT_QHENOMENOLOGY_979: (8)                     return super().set_points(points)
SNGT_QHENOMENOLOGY_980: (4)                 @triggers_refresh
SNGT_QHENOMENOLOGY_981: (4)                 def append_points(self, points: Vect3Array) -> Self:
SNGT_QHENOMENOLOGY_982: (8)                     assert len(points) % 2 == 0
SNGT_QHENOMENOLOGY_983: (8)                     return super().append_points(points)
SNGT_QHENOMENOLOGY_984: (4)                 def reverse_points(self, recurse: bool = True) -> Self:
SNGT_QHENOMENOLOGY_985: (8)                     for mob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_986: (12)                        if not mob.has_points():
SNGT_QHENOMENOLOGY_987: (16)                            continue
SNGT_QHENOMENOLOGY_988: (12)                        inner_ends = mob.get_subpath_end_indices()[:-1]
SNGT_QHENOMENOLOGY_989: (12)                        mob.data["point"][inner_ends + 1] =
mob.data["point"][inner_ends + 2]
SNGT_QHENOMENOLOGY_990: (12)                        mob.data["base_normal"][1::2] *= -1  # Invert
normal vector
SNGT_QHENOMENOLOGY_991: (12)                        self.subpath_end_indices = None
SNGT_QHENOMENOLOGY_992: (8)                     return super().reverse_points()
SNGT_QHENOMENOLOGY_993: (4)                 @triggers_refresh
SNGT_QHENOMENOLOGY_994: (4)                 def set_data(self, data: np.ndarray) -> Self:
SNGT_QHENOMENOLOGY_995: (8)                     return super().set_data(data)
SNGT_QHENOMENOLOGY_996: (4)                 @triggers_refresh
SNGT_QHENOMENOLOGY_997: (4)                 def apply_function(
SNGT_QHENOMENOLOGY_998: (8)                     self,
SNGT_QHENOMENOLOGY_999: (8)                     function: Callable[[Vect3], Vect3],
SNGT_QHENOMENOLOGY_1000: (8)                    make_smooth: bool = False,
SNGT_QHENOMENOLOGY_1001: (8)                    **kwargs
SNGT_QHENOMENOLOGY_1002: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1003: (8)                    super().apply_function(function, **kwargs)
SNGT_QHENOMENOLOGY_1004: (8)                    if self.make_smooth_after_applying_functions or
make_smooth:
SNGT_QHENOMENOLOGY_1005: (12)                       self.make_smooth(approx=True)
SNGT_QHENOMENOLOGY_1006: (8)                    return self
SNGT_QHENOMENOLOGY_1007: (4)                @triggers_refresh
SNGT_QHENOMENOLOGY_1008: (4)                def stretch(self, *args, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_1009: (8)                    return super().stretch(*args, **kwargs)
SNGT_QHENOMENOLOGY_1010: (4)                @triggers_refresh
SNGT_QHENOMENOLOGY_1011: (4)                def apply_matrix(self, *args, **kwargs) -> Self:
SNGT_QHENOMENOLOGY_1012: (8)                    return super().apply_matrix(*args, **kwargs)
SNGT_QHENOMENOLOGY_1013: (4)                def rotate(
SNGT_QHENOMENOLOGY_1014: (8)                    self,
SNGT_QHENOMENOLOGY_1015: (8)                    angle: float,
SNGT_QHENOMENOLOGY_1016: (8)                    axis: Vect3 = OUT,
SNGT_QHENOMENOLOGY_1017: (8)                    about_point: Vect3 | None = None,
SNGT_QHENOMENOLOGY_1018: (8)                    **kwargs
SNGT_QHENOMENOLOGY_1019: (4)                ) -> Self:
SNGT_QHENOMENOLOGY_1020: (8)                    rot_matrix_T = rotation_matrix_transpose(angle,
axis)
SNGT_QHENOMENOLOGY_1021: (8)                    self.apply_points_function(
SNGT_QHENOMENOLOGY_1022: (12)                       lambda points: np.dot(points, rot_matrix_T),
SNGT_QHENOMENOLOGY_1023: (12)                       about_point,
SNGT_QHENOMENOLOGY_1024: (12)                       **kwargs
SNGT_QHENOMENOLOGY_1025: (8)                    )
SNGT_QHENOMENOLOGY_1026: (8)                    for mob in self.get_family():
SNGT_QHENOMENOLOGY_1027: (12)                       mob.get_unit_normal(refresh=True)
SNGT_QHENOMENOLOGY_1028: (8)                    return self
SNGT_QHENOMENOLOGY_1029: (4)                def set_animating_status(self, is_animating: bool,
recurse: bool = True):
SNGT_QHENOMENOLOGY_1030: (8)                    super().set_animating_status(is_animating, recurse)
```

```
SNGT_QHENOMENOLOGY_1031: (8)                    for submob in self.get_family(recurse):
SNGT_QHENOMENOLOGY_1032: (12)                       submob.get_joint_angles(refresh=True)
SNGT_QHENOMENOLOGY_1033: (8)                    return self
SNGT_QHENOMENOLOGY_1034: (4)                def init_shader_wrapper(self, ctx: Context):
SNGT_QHENOMENOLOGY_1035: (8)                    self.shader_wrapper = VShaderWrapper(
SNGT_QHENOMENOLOGY_1036: (12)                       ctx=ctx,
SNGT_QHENOMENOLOGY_1037: (12)                       vert_data=self.data,
SNGT_QHENOMENOLOGY_1038: (12)                       mobject_uniforms=self.uniforms,
SNGT_QHENOMENOLOGY_1039: (12)
code_replacements=self.shader_code_replacements,
SNGT_QHENOMENOLOGY_1040: (12)                       stroke_behind=self.stroke_behind,
SNGT_QHENOMENOLOGY_1041: (12)                       depth_test=self.depth_test
SNGT_QHENOMENOLOGY_1042: (8)                    )
SNGT_QHENOMENOLOGY_1043: (4)                def refresh_shader_wrapper_id(self):
SNGT_QHENOMENOLOGY_1044: (8)                    for submob in self.get_family():
SNGT_QHENOMENOLOGY_1045: (12)                       if submob.shader_wrapper is not None:
SNGT_QHENOMENOLOGY_1046: (16)                           submob.shader_wrapper.stroke_behind =
submob.stroke_behind
SNGT_QHENOMENOLOGY_1047: (8)                    super().refresh_shader_wrapper_id()
SNGT_QHENOMENOLOGY_1048: (8)                    return self
SNGT_QHENOMENOLOGY_1049: (4)                def get_shader_data(self) -> np.ndarray:
SNGT_QHENOMENOLOGY_1050: (8)                    self.get_joint_angles()
SNGT_QHENOMENOLOGY_1051: (8)                    self.data["base_normal"][0::2] = self.data["point"]
[0]
SNGT_QHENOMENOLOGY_1052: (8)                    return super().get_shader_data()
SNGT_QHENOMENOLOGY_1053: (4)                def get_shader_vert_indices(self) ->
Optional[np.ndarray]:
SNGT_QHENOMENOLOGY_1054: (8)                    return self.get_outer_vert_indices()
SNGT_QHENOMENOLOGY_1055: (0)            class VGroup(Group, VMobject, Generic[SubVmobjectType]):
SNGT_QHENOMENOLOGY_1056: (4)                def __init__(self, *vmobjects: SubVmobjectType |
Iterable[SubVmobjectType], **kwargs):
SNGT_QHENOMENOLOGY_1057: (8)                    super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_1058: (8)                    if any(isinstance(vmob, Mobject) and not
isinstance(vmob, VMobject) for vmob in vmobjects):
SNGT_QHENOMENOLOGY_1059: (12)                       raise Exception("Only VMobjects can be passed
into VGroup")
SNGT_QHENOMENOLOGY_1060: (8)                    self._ingest_args(*vmobjects)
SNGT_QHENOMENOLOGY_1061: (8)                    if self.submobjects:
SNGT_QHENOMENOLOGY_1062: (12)
self.uniforms.update(self.submobjects[0].uniforms)
SNGT_QHENOMENOLOGY_1063: (4)                def __add__(self, other: VMobject) -> Self:
SNGT_QHENOMENOLOGY_1064: (8)                    assert isinstance(other, VMobject)
SNGT_QHENOMENOLOGY_1065: (8)                    return self.add(other)
SNGT_QHENOMENOLOGY_1066: (4)                def __getitem__(self, index) -> SubVmobjectType:
SNGT_QHENOMENOLOGY_1067: (8)                    return super().__getitem__(index)
SNGT_QHENOMENOLOGY_1068: (0)            class VectorizedPoint(Point, VMobject):
SNGT_QHENOMENOLOGY_1069: (4)                def __init__(
SNGT_QHENOMENOLOGY_1070: (8)                    self,
SNGT_QHENOMENOLOGY_1071: (8)                    location: np.ndarray = ORIGIN,
SNGT_QHENOMENOLOGY_1072: (8)                    color: ManimColor = BLACK,
SNGT_QHENOMENOLOGY_1073: (8)                    fill_opacity: float = 0.0,
SNGT_QHENOMENOLOGY_1074: (8)                    stroke_width: float = 0.0,
SNGT_QHENOMENOLOGY_1075: (8)                    **kwargs
SNGT_QHENOMENOLOGY_1076: (4)                ):
SNGT_QHENOMENOLOGY_1077: (8)                    Point.__init__(self, location, **kwargs)
SNGT_QHENOMENOLOGY_1078: (8)                    VMobject.__init__(
SNGT_QHENOMENOLOGY_1079: (12)                       self,
SNGT_QHENOMENOLOGY_1080: (12)                       color=color,
SNGT_QHENOMENOLOGY_1081: (12)                       fill_opacity=fill_opacity,
SNGT_QHENOMENOLOGY_1082: (12)                       stroke_width=stroke_width,
SNGT_QHENOMENOLOGY_1083: (12)                       **kwargs
SNGT_QHENOMENOLOGY_1084: (8)                    )
SNGT_QHENOMENOLOGY_1085: (8)                    self.set_points(np.array([location]))
SNGT_QHENOMENOLOGY_1086: (0)            class CurvesAsSubmobjects(VGroup):
SNGT_QHENOMENOLOGY_1087: (4)                def __init__(self, vmobject: VMobject, **kwargs):
SNGT_QHENOMENOLOGY_1088: (8)                    super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_1089: (8)                    for tup in vmobject.get_bezier_tuples():
SNGT_QHENOMENOLOGY_1090: (12)                       part = VMobject()
SNGT_QHENOMENOLOGY_1091: (12)                       part.set_points(tup)
```

```
SNGT_QHENOMENOLOGY_1092: (12)                         part.match_style(vmobject)
SNGT_QHENOMENOLOGY_1093: (12)                         self.add(part)
SNGT_QHENOMENOLOGY_1094: (0)             class DashedVMobject(VMobject):
SNGT_QHENOMENOLOGY_1095: (4)                 def __init__(
SNGT_QHENOMENOLOGY_1096: (8)                     self,
SNGT_QHENOMENOLOGY_1097: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_1098: (8)                     num_dashes: int = 15,
SNGT_QHENOMENOLOGY_1099: (8)                     positive_space_ratio: float = 0.5,
SNGT_QHENOMENOLOGY_1100: (8)                     **kwargs
SNGT_QHENOMENOLOGY_1101: (4)                 ):
SNGT_QHENOMENOLOGY_1102: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_1103: (8)                     if num_dashes > 0:
SNGT_QHENOMENOLOGY_1104: (12)                         alphas = np.linspace(0, 1, num_dashes + 1)
SNGT_QHENOMENOLOGY_1105: (12)                         full_d_alpha = (1.0 / num_dashes)
SNGT_QHENOMENOLOGY_1106: (12)                         partial_d_alpha = full_d_alpha *
positive_space_ratio
SNGT_QHENOMENOLOGY_1107: (12)                         alphas /= (1 - full_d_alpha + partial_d_alpha)
SNGT_QHENOMENOLOGY_1108: (12)                         self.add(*[
SNGT_QHENOMENOLOGY_1109: (16)                             vmobject.get_subcurve(alpha, alpha +
partial_d_alpha)
SNGT_QHENOMENOLOGY_1110: (16)                             for alpha in alphas[:-1]
SNGT_QHENOMENOLOGY_1111: (12)                         ])
SNGT_QHENOMENOLOGY_1112: (8)                     self.match_style(vmobject, recurse=False)
SNGT_QHENOMENOLOGY_1113: (0)             class VHighlight(VGroup):
SNGT_QHENOMENOLOGY_1114: (4)                 def __init__(
SNGT_QHENOMENOLOGY_1115: (8)                     self,
SNGT_QHENOMENOLOGY_1116: (8)                     vmobject: VMobject,
SNGT_QHENOMENOLOGY_1117: (8)                     n_layers: int = 5,
SNGT_QHENOMENOLOGY_1118: (8)                     color_bounds: Tuple[ManimColor] = (GREY_C, GREY_E),
SNGT_QHENOMENOLOGY_1119: (8)                     max_stroke_addition: float = 5.0,
SNGT_QHENOMENOLOGY_1120: (4)                 ):
SNGT_QHENOMENOLOGY_1121: (8)                     outline = vmobject.replicate(n_layers)
SNGT_QHENOMENOLOGY_1122: (8)                     outline.set_fill(opacity=0)
SNGT_QHENOMENOLOGY_1123: (8)                     added_widths = np.linspace(0, max_stroke_addition,
n_layers + 1)[1:]
SNGT_QHENOMENOLOGY_1124: (8)                     colors = color_gradient(color_bounds, n_layers)
SNGT_QHENOMENOLOGY_1125: (8)                     for part, added_width, color in
zip(reversed(outline), added_widths, colors):
SNGT_QHENOMENOLOGY_1126: (12)                         for sm in part.family_members_with_points():
SNGT_QHENOMENOLOGY_1127: (16)                             sm.set_stroke(
SNGT_QHENOMENOLOGY_1128: (20)                                 width=sm.get_stroke_width() +
added_width,
SNGT_QHENOMENOLOGY_1129: (20)                                 color=color,
SNGT_QHENOMENOLOGY_1130: (16)                             )
SNGT_QHENOMENOLOGY_1131: (8)                     super().__init__(*outline)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 65 - point_cloud_mobject.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import numpy as np
SNGT_QHENOMENOLOGY_3: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_4: (0)             from manimlib.utils.color import color_gradient
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.utils.color import color_to_rgba
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_7: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_10: (4)                from manimlib.typing import ManimColor, Vect3,
Vect3Array, Vect4Array, Self
SNGT_QHENOMENOLOGY_11: (0)            class PMobject(Mobject):
SNGT_QHENOMENOLOGY_12: (4)                def set_points(self, points: Vect3Array):
SNGT_QHENOMENOLOGY_13: (8)                    if len(points) == 0:
SNGT_QHENOMENOLOGY_14: (12)                       points = np.zeros((0, 3))
SNGT_QHENOMENOLOGY_15: (8)                    super().set_points(points)
SNGT_QHENOMENOLOGY_16: (8)                    self.resize_points(len(points))
SNGT_QHENOMENOLOGY_17: (8)                    return self
```

```
SNGT_QHENOMENOLOGY_18: (4)                 def add_points(
SNGT_QHENOMENOLOGY_19: (8)                     self,
SNGT_QHENOMENOLOGY_20: (8)                     points: Vect3Array,
SNGT_QHENOMENOLOGY_21: (8)                     rgbas: Vect4Array | None = None,
SNGT_QHENOMENOLOGY_22: (8)                     color: ManimColor | None = None,
SNGT_QHENOMENOLOGY_23: (8)                     opacity: float | None = None
SNGT_QHENOMENOLOGY_24: (4)                 ) -> Self:
SNGT_QHENOMENOLOGY_25: (8)                     """
SNGT_QHENOMENOLOGY_26: (8)                     points must be a Nx3 numpy array, as must rgbas if
it is not None
SNGT_QHENOMENOLOGY_27: (8)                     """
SNGT_QHENOMENOLOGY_28: (8)                     self.append_points(points)
SNGT_QHENOMENOLOGY_29: (8)                     if color is not None:
SNGT_QHENOMENOLOGY_30: (12)                        if opacity is None:
SNGT_QHENOMENOLOGY_31: (16)                            opacity = self.data["rgba"][-1, 3]
SNGT_QHENOMENOLOGY_32: (12)                        rgbas = np.repeat(
SNGT_QHENOMENOLOGY_33: (16)                            [color_to_rgba(color, opacity)],
SNGT_QHENOMENOLOGY_34: (16)                            len(points),
SNGT_QHENOMENOLOGY_35: (16)                            axis=0
SNGT_QHENOMENOLOGY_36: (12)                        )
SNGT_QHENOMENOLOGY_37: (8)                     if rgbas is not None:
SNGT_QHENOMENOLOGY_38: (12)                        self.data["rgba"][-len(rgbas):] = rgbas
SNGT_QHENOMENOLOGY_39: (8)                     return self
SNGT_QHENOMENOLOGY_40: (4)                 def add_point(self, point: Vect3, rgba=None,
color=None, opacity=None) -> Self:
SNGT_QHENOMENOLOGY_41: (8)                     rgbas = None if rgba is None else [rgba]
SNGT_QHENOMENOLOGY_42: (8)                     self.add_points([point], rgbas, color, opacity)
SNGT_QHENOMENOLOGY_43: (8)                     return self
SNGT_QHENOMENOLOGY_44: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_45: (4)                 def set_color_by_gradient(self, *colors: ManimColor) ->
Self:
SNGT_QHENOMENOLOGY_46: (8)                     self.data["rgba"][:] = np.array(list(map(
SNGT_QHENOMENOLOGY_47: (12)                        color_to_rgba,
SNGT_QHENOMENOLOGY_48: (12)                        color_gradient(colors, self.get_num_points())
SNGT_QHENOMENOLOGY_49: (8)                     )))
SNGT_QHENOMENOLOGY_50: (8)                     return self
SNGT_QHENOMENOLOGY_51: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_52: (4)                 def match_colors(self, pmobject: PMobject) -> Self:
SNGT_QHENOMENOLOGY_53: (8)                     self.data["rgba"][:] = resize_with_interpolation(
SNGT_QHENOMENOLOGY_54: (12)                        pmobject.data["rgba"], self.get_num_points()
SNGT_QHENOMENOLOGY_55: (8)                     )
SNGT_QHENOMENOLOGY_56: (8)                     return self
SNGT_QHENOMENOLOGY_57: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_58: (4)                 def filter_out(self, condition: Callable[[np.ndarray],
bool]) -> Self:
SNGT_QHENOMENOLOGY_59: (8)                     for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_60: (12)                        mob.data =
mob.data[~np.apply_along_axis(condition, 1, mob.get_points())]
SNGT_QHENOMENOLOGY_61: (8)                     return self
SNGT_QHENOMENOLOGY_62: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_63: (4)                 def sort_points(self, function: Callable[[Vect3], None]
= lambda p: p[0]) -> Self:
SNGT_QHENOMENOLOGY_64: (8)                     """
SNGT_QHENOMENOLOGY_65: (8)                     function is any map from R^3 to R
SNGT_QHENOMENOLOGY_66: (8)                     """
SNGT_QHENOMENOLOGY_67: (8)                     for mob in self.family_members_with_points():
SNGT_QHENOMENOLOGY_68: (12)                        indices = np.argsort(
SNGT_QHENOMENOLOGY_69: (16)                            np.apply_along_axis(function, 1,
mob.get_points())
SNGT_QHENOMENOLOGY_70: (12)                        )
SNGT_QHENOMENOLOGY_71: (12)                        mob.data[:] = mob.data[indices]
SNGT_QHENOMENOLOGY_72: (8)                     return self
SNGT_QHENOMENOLOGY_73: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_74: (4)                 def ingest_submobjects(self) -> Self:
SNGT_QHENOMENOLOGY_75: (8)                     self.data = np.vstack([
SNGT_QHENOMENOLOGY_76: (12)                        sm.data for sm in self.get_family()
SNGT_QHENOMENOLOGY_77: (8)                     ])
SNGT_QHENOMENOLOGY_78: (8)                     return self
SNGT_QHENOMENOLOGY_79: (4)                 def point_from_proportion(self, alpha: float) ->
```

```
np.ndarray:
SNGT_QHENOMENOLOGY_80: (8)                     index = alpha * (self.get_num_points() - 1)
SNGT_QHENOMENOLOGY_81: (8)                     return self.get_points()[int(index)]
SNGT_QHENOMENOLOGY_82: (4)                 @Mobject.affects_data
SNGT_QHENOMENOLOGY_83: (4)                 def pointwise_become_partial(self, pmobject: PMobject,
a: float, b: float) -> Self:
SNGT_QHENOMENOLOGY_84: (8)                     lower_index = int(a * pmobject.get_num_points())
SNGT_QHENOMENOLOGY_85: (8)                     upper_index = int(b * pmobject.get_num_points())
SNGT_QHENOMENOLOGY_86: (8)                     self.data =
pmobject.data[lower_index:upper_index].copy()
SNGT_QHENOMENOLOGY_87: (8)                     return self
SNGT_QHENOMENOLOGY_88: (0)             class PGroup(PMobject):
SNGT_QHENOMENOLOGY_89: (4)                 def __init__(self, *pmobs: PMobject, **kwargs):
SNGT_QHENOMENOLOGY_90: (8)                     if not all([isinstance(m, PMobject) for m in
pmobs]):
SNGT_QHENOMENOLOGY_91: (12)                        raise Exception("All submobjects must be of
type PMobject")
SNGT_QHENOMENOLOGY_92: (8)                     super().__init__(**kwargs)
SNGT_QHENOMENOLOGY_93: (8)                     self.add(*pmobs)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 66 - scene.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from collections import OrderedDict
SNGT_QHENOMENOLOGY_3: (0)              import platform
SNGT_QHENOMENOLOGY_4: (0)              import random
SNGT_QHENOMENOLOGY_5: (0)              import time
SNGT_QHENOMENOLOGY_6: (0)              from functools import wraps
SNGT_QHENOMENOLOGY_7: (0)              from contextlib import contextmanager
SNGT_QHENOMENOLOGY_8: (0)              from contextlib import ExitStack
SNGT_QHENOMENOLOGY_9: (0)              from pyglet.window import key as PygletWindowKeys
SNGT_QHENOMENOLOGY_10: (0)             import numpy as np
SNGT_QHENOMENOLOGY_11: (0)             from tqdm.auto import tqdm as ProgressDisplay
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.animation.animation import prepare_animation
SNGT_QHENOMENOLOGY_13: (0)             from manimlib.camera.camera import Camera
SNGT_QHENOMENOLOGY_14: (0)             from manimlib.camera.camera_frame import CameraFrame
SNGT_QHENOMENOLOGY_15: (0)             from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_16: (0)             from manimlib.event_handler import EVENT_DISPATCHER
SNGT_QHENOMENOLOGY_17: (0)             from manimlib.event_handler.event_type import EventType
SNGT_QHENOMENOLOGY_18: (0)             from manimlib.logger import log
SNGT_QHENOMENOLOGY_19: (0)             from manimlib.mobject.mobject import _AnimationBuilder
SNGT_QHENOMENOLOGY_20: (0)             from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_22: (0)             from manimlib.mobject.mobject import Point
SNGT_QHENOMENOLOGY_23: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_24: (0)             from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_25: (0)             from manimlib.scene.scene_embed import
InteractiveSceneEmbed
SNGT_QHENOMENOLOGY_26: (0)             from manimlib.scene.scene_embed import CheckpointManager
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.scene.scene_file_writer import
SceneFileWriter
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.dict_ops import merge_dicts_recursively
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.utils.family_ops import
extract_mobject_family_members
SNGT_QHENOMENOLOGY_30: (0)             from manimlib.utils.family_ops import
recursive_mobject_remove
SNGT_QHENOMENOLOGY_31: (0)             from manimlib.utils.iterables import batch_by_property
SNGT_QHENOMENOLOGY_32: (0)             from manimlib.window import Window
SNGT_QHENOMENOLOGY_33: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_34: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_35: (4)                 from typing import Callable, Iterable, TypeVar,
Optional
SNGT_QHENOMENOLOGY_36: (4)                 from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_37: (4)                 T = TypeVar('T')
SNGT_QHENOMENOLOGY_38: (4)                 from PIL.Image import Image
```

```
SNGT_QHENOMENOLOGY_39: (4)                 from manimlib.animation.animation import Animation
SNGT_QHENOMENOLOGY_40: (0)             class Scene(object):
SNGT_QHENOMENOLOGY_41: (4)                 random_seed: int = 0
SNGT_QHENOMENOLOGY_42: (4)                 pan_sensitivity: float = 0.5
SNGT_QHENOMENOLOGY_43: (4)                 scroll_sensitivity: float = 20
SNGT_QHENOMENOLOGY_44: (4)                 drag_to_pan: bool = True
SNGT_QHENOMENOLOGY_45: (4)                 max_num_saved_states: int = 50
SNGT_QHENOMENOLOGY_46: (4)                 default_camera_config: dict = dict()
SNGT_QHENOMENOLOGY_47: (4)                 default_file_writer_config: dict = dict()
SNGT_QHENOMENOLOGY_48: (4)                 samples = 0
SNGT_QHENOMENOLOGY_49: (4)                 default_frame_orientation = (0, 0)
SNGT_QHENOMENOLOGY_50: (4)                 def __init__(
SNGT_QHENOMENOLOGY_51: (8)                     self,
SNGT_QHENOMENOLOGY_52: (8)                     window: Optional[Window] = None,
SNGT_QHENOMENOLOGY_53: (8)                     camera_config: dict = dict(),
SNGT_QHENOMENOLOGY_54: (8)                     file_writer_config: dict = dict(),
SNGT_QHENOMENOLOGY_55: (8)                     skip_animations: bool = False,
SNGT_QHENOMENOLOGY_56: (8)                     always_update_mobjects: bool = False,
SNGT_QHENOMENOLOGY_57: (8)                     start_at_animation_number: int | None = None,
SNGT_QHENOMENOLOGY_58: (8)                     end_at_animation_number: int | None = None,
SNGT_QHENOMENOLOGY_59: (8)                     show_animation_progress: bool = False,
SNGT_QHENOMENOLOGY_60: (8)                     leave_progress_bars: bool = False,
SNGT_QHENOMENOLOGY_61: (8)                     preview_while_skipping: bool = True,
SNGT_QHENOMENOLOGY_62: (8)                     presenter_mode: bool = False,
SNGT_QHENOMENOLOGY_63: (8)                     default_wait_time: float = 1.0,
SNGT_QHENOMENOLOGY_64: (4)                 ):
SNGT_QHENOMENOLOGY_65: (8)                     self.skip_animations = skip_animations
SNGT_QHENOMENOLOGY_66: (8)                     self.always_update_mobjects =
always_update_mobjects
SNGT_QHENOMENOLOGY_67: (8)                     self.start_at_animation_number =
start_at_animation_number
SNGT_QHENOMENOLOGY_68: (8)                     self.end_at_animation_number =
end_at_animation_number
SNGT_QHENOMENOLOGY_69: (8)                     self.show_animation_progress =
show_animation_progress
SNGT_QHENOMENOLOGY_70: (8)                     self.leave_progress_bars = leave_progress_bars
SNGT_QHENOMENOLOGY_71: (8)                     self.preview_while_skipping =
preview_while_skipping
SNGT_QHENOMENOLOGY_72: (8)                     self.presenter_mode = presenter_mode
SNGT_QHENOMENOLOGY_73: (8)                     self.default_wait_time = default_wait_time
SNGT_QHENOMENOLOGY_74: (8)                     self.camera_config = merge_dicts_recursively(
SNGT_QHENOMENOLOGY_75: (12)                        manim_config.camera,         # Global default
SNGT_QHENOMENOLOGY_76: (12)                        self.default_camera_config,  # Updated
configuration that subclasses may specify
SNGT_QHENOMENOLOGY_77: (12)                        camera_config,               # Updated
configuration from instantiation
SNGT_QHENOMENOLOGY_78: (8)                     )
SNGT_QHENOMENOLOGY_79: (8)                     self.file_writer_config = merge_dicts_recursively(
SNGT_QHENOMENOLOGY_80: (12)                        manim_config.file_writer,
SNGT_QHENOMENOLOGY_81: (12)                        self.default_file_writer_config,
SNGT_QHENOMENOLOGY_82: (12)                        file_writer_config,
SNGT_QHENOMENOLOGY_83: (8)                     )
SNGT_QHENOMENOLOGY_84: (8)                     self.window = window
SNGT_QHENOMENOLOGY_85: (8)                     if self.window:
SNGT_QHENOMENOLOGY_86: (12)                        self.window.init_for_scene(self)
SNGT_QHENOMENOLOGY_87: (12)                        self.camera_config["fps"] = 30
SNGT_QHENOMENOLOGY_88: (8)                     self.camera: Camera = Camera(
SNGT_QHENOMENOLOGY_89: (12)                        window=self.window,
SNGT_QHENOMENOLOGY_90: (12)                        samples=self.samples,
SNGT_QHENOMENOLOGY_91: (12)                        **self.camera_config
SNGT_QHENOMENOLOGY_92: (8)                     )
SNGT_QHENOMENOLOGY_93: (8)                     self.frame: CameraFrame = self.camera.frame
SNGT_QHENOMENOLOGY_94: (8)
self.frame.reorient(*self.default_frame_orientation)
SNGT_QHENOMENOLOGY_95: (8)                     self.frame.make_orientation_default()
SNGT_QHENOMENOLOGY_96: (8)                     self.file_writer = SceneFileWriter(self,
**self.file_writer_config)
SNGT_QHENOMENOLOGY_97: (8)                     self.mobjects: list[Mobject] = [self.camera.frame]
SNGT_QHENOMENOLOGY_98: (8)                     self.render_groups: list[Mobject] = []
```

```
SNGT_QHENOMENOLOGY_99: (8)                      self.id_to_mobject_map: dict[int, Mobject] = dict()
SNGT_QHENOMENOLOGY_100: (8)                     self.num_plays: int = 0
SNGT_QHENOMENOLOGY_101: (8)                     self.time: float = 0
SNGT_QHENOMENOLOGY_102: (8)                     self.skip_time: float = 0
SNGT_QHENOMENOLOGY_103: (8)                     self.original_skipping_status: bool =
self.skip_animations
SNGT_QHENOMENOLOGY_104: (8)                     self.undo_stack = []
SNGT_QHENOMENOLOGY_105: (8)                     self.redo_stack = []
SNGT_QHENOMENOLOGY_106: (8)                     if self.start_at_animation_number is not None:
SNGT_QHENOMENOLOGY_107: (12)                        self.skip_animations = True
SNGT_QHENOMENOLOGY_108: (8)                     if self.file_writer.has_progress_display():
SNGT_QHENOMENOLOGY_109: (12)                        self.show_animation_progress = False
SNGT_QHENOMENOLOGY_110: (8)                     self.mouse_point = Point()
SNGT_QHENOMENOLOGY_111: (8)                     self.mouse_drag_point = Point()
SNGT_QHENOMENOLOGY_112: (8)                     self.hold_on_wait = self.presenter_mode
SNGT_QHENOMENOLOGY_113: (8)                     self.quit_interaction = False
SNGT_QHENOMENOLOGY_114: (8)                     if self.random_seed is not None:
SNGT_QHENOMENOLOGY_115: (12)                        random.seed(self.random_seed)
SNGT_QHENOMENOLOGY_116: (12)                        np.random.seed(self.random_seed)
SNGT_QHENOMENOLOGY_117: (4)                 def __str__(self) -> str:
SNGT_QHENOMENOLOGY_118: (8)                     return self.__class__.__name__
SNGT_QHENOMENOLOGY_119: (4)                 def get_window(self) -> Window | None:
SNGT_QHENOMENOLOGY_120: (8)                     return self.window
SNGT_QHENOMENOLOGY_121: (4)                 def run(self) -> None:
SNGT_QHENOMENOLOGY_122: (8)                     self.virtual_animation_start_time: float = 0
SNGT_QHENOMENOLOGY_123: (8)                     self.real_animation_start_time: float = time.time()
SNGT_QHENOMENOLOGY_124: (8)                     self.file_writer.begin()
SNGT_QHENOMENOLOGY_125: (8)                     self.setup()
SNGT_QHENOMENOLOGY_126: (8)                     try:
SNGT_QHENOMENOLOGY_127: (12)                        self.construct()
SNGT_QHENOMENOLOGY_128: (12)                        self.interact()
SNGT_QHENOMENOLOGY_129: (8)                     except EndScene:
SNGT_QHENOMENOLOGY_130: (12)                        pass
SNGT_QHENOMENOLOGY_131: (8)                     except KeyboardInterrupt:
SNGT_QHENOMENOLOGY_132: (12)                        print("", end="\r")
SNGT_QHENOMENOLOGY_133: (12)                        self.file_writer.ended_with_interrupt = True
SNGT_QHENOMENOLOGY_134: (8)                     self.tear_down()
SNGT_QHENOMENOLOGY_135: (4)                 def setup(self) -> None:
SNGT_QHENOMENOLOGY_136: (8)                     """
SNGT_QHENOMENOLOGY_137: (8)                     This is meant to be implement by any scenes which
SNGT_QHENOMENOLOGY_138: (8)                     are comonly subclassed, and have some common setup
SNGT_QHENOMENOLOGY_139: (8)                     involved before the construct method is called.
SNGT_QHENOMENOLOGY_140: (8)                     """
SNGT_QHENOMENOLOGY_141: (8)                     pass
SNGT_QHENOMENOLOGY_142: (4)                 def construct(self) -> None:
SNGT_QHENOMENOLOGY_143: (8)                     pass
SNGT_QHENOMENOLOGY_144: (4)                 def tear_down(self) -> None:
SNGT_QHENOMENOLOGY_145: (8)                     self.stop_skipping()
SNGT_QHENOMENOLOGY_146: (8)                     self.file_writer.finish()
SNGT_QHENOMENOLOGY_147: (8)                     if self.window:
SNGT_QHENOMENOLOGY_148: (12)                        self.window.destroy()
SNGT_QHENOMENOLOGY_149: (12)                        self.window = None
SNGT_QHENOMENOLOGY_150: (4)                 def interact(self) -> None:
SNGT_QHENOMENOLOGY_151: (8)                     """
SNGT_QHENOMENOLOGY_152: (8)                     If there is a window, enter a loop
SNGT_QHENOMENOLOGY_153: (8)                     which updates the frame while under
SNGT_QHENOMENOLOGY_154: (8)                     the hood calling the pyglet event loop
SNGT_QHENOMENOLOGY_155: (8)                     """
SNGT_QHENOMENOLOGY_156: (8)                     if self.window is None:
SNGT_QHENOMENOLOGY_157: (12)                        return
SNGT_QHENOMENOLOGY_158: (8)                     log.info(
SNGT_QHENOMENOLOGY_159: (12)                        "\nTips: Using the keys `d`, `f`, or `z` " +
SNGT_QHENOMENOLOGY_160: (12)                        "you can interact with the scene. " +
SNGT_QHENOMENOLOGY_161: (12)                        "Press `command + q` or `esc` to quit"
SNGT_QHENOMENOLOGY_162: (8)                     )
SNGT_QHENOMENOLOGY_163: (8)                     self.skip_animations = False
SNGT_QHENOMENOLOGY_164: (8)                     while not self.is_window_closing():
SNGT_QHENOMENOLOGY_165: (12)                        self.update_frame(1 / self.camera.fps)
SNGT_QHENOMENOLOGY_166: (4)                 def embed(
```

```
SNGT_QHENOMENOLOGY_167: (8)                     self,
SNGT_QHENOMENOLOGY_168: (8)                     close_scene_on_exit: bool = True,
SNGT_QHENOMENOLOGY_169: (8)                     show_animation_progress: bool = False,
SNGT_QHENOMENOLOGY_170: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_171: (8)                     if not self.window:
SNGT_QHENOMENOLOGY_172: (12)                        return
SNGT_QHENOMENOLOGY_173: (8)                     self.show_animation_progress =
show_animation_progress
SNGT_QHENOMENOLOGY_174: (8)                     self.stop_skipping()
SNGT_QHENOMENOLOGY_175: (8)                     self.update_frame(force_draw=True)
SNGT_QHENOMENOLOGY_176: (8)                     InteractiveSceneEmbed(self).launch()
SNGT_QHENOMENOLOGY_177: (8)                     if close_scene_on_exit:
SNGT_QHENOMENOLOGY_178: (12)                        raise EndScene()
SNGT_QHENOMENOLOGY_179: (4)                 def get_image(self) -> Image:
SNGT_QHENOMENOLOGY_180: (8)                     if self.window is not None:
SNGT_QHENOMENOLOGY_181: (12)                        self.camera.use_window_fbo(False)
SNGT_QHENOMENOLOGY_182: (12)                        self.camera.capture(*self.render_groups)
SNGT_QHENOMENOLOGY_183: (8)                     image = self.camera.get_image()
SNGT_QHENOMENOLOGY_184: (8)                     if self.window is not None:
SNGT_QHENOMENOLOGY_185: (12)                        self.camera.use_window_fbo(True)
SNGT_QHENOMENOLOGY_186: (8)                     return image
SNGT_QHENOMENOLOGY_187: (4)                 def show(self) -> None:
SNGT_QHENOMENOLOGY_188: (8)                     self.update_frame(force_draw=True)
SNGT_QHENOMENOLOGY_189: (8)                     self.get_image().show()
SNGT_QHENOMENOLOGY_190: (4)                 def update_frame(self, dt: float = 0, force_draw: bool
= False) -> None:
SNGT_QHENOMENOLOGY_191: (8)                     self.increment_time(dt)
SNGT_QHENOMENOLOGY_192: (8)                     self.update_mobjects(dt)
SNGT_QHENOMENOLOGY_193: (8)                     if self.skip_animations and not force_draw:
SNGT_QHENOMENOLOGY_194: (12)                        return
SNGT_QHENOMENOLOGY_195: (8)                     if self.is_window_closing():
SNGT_QHENOMENOLOGY_196: (12)                        raise EndScene()
SNGT_QHENOMENOLOGY_197: (8)                     if self.window and dt == 0 and not
self.window.has_undrawn_event() and not force_draw:
SNGT_QHENOMENOLOGY_198: (12)                        self.window._window.dispatch_events()
SNGT_QHENOMENOLOGY_199: (12)                        return
SNGT_QHENOMENOLOGY_200: (8)                     self.camera.capture(*self.render_groups)
SNGT_QHENOMENOLOGY_201: (8)                     if self.window and not self.skip_animations:
SNGT_QHENOMENOLOGY_202: (12)                        vt = self.time -
self.virtual_animation_start_time
SNGT_QHENOMENOLOGY_203: (12)                        rt = time.time() -
self.real_animation_start_time
SNGT_QHENOMENOLOGY_204: (12)                        time.sleep(max(vt - rt, 0))
SNGT_QHENOMENOLOGY_205: (4)                 def emit_frame(self) -> None:
SNGT_QHENOMENOLOGY_206: (8)                     if not self.skip_animations:
SNGT_QHENOMENOLOGY_207: (12)                        self.file_writer.write_frame(self.camera)
SNGT_QHENOMENOLOGY_208: (4)                 def update_mobjects(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_209: (8)                     for mobject in self.mobjects:
SNGT_QHENOMENOLOGY_210: (12)                        mobject.update(dt)
SNGT_QHENOMENOLOGY_211: (4)                 def should_update_mobjects(self) -> bool:
SNGT_QHENOMENOLOGY_212: (8)                     return self.always_update_mobjects or any(
SNGT_QHENOMENOLOGY_213: (12)                        mob.has_updaters() for mob in self.mobjects
SNGT_QHENOMENOLOGY_214: (8)                     )
SNGT_QHENOMENOLOGY_215: (4)                 def get_time(self) -> float:
SNGT_QHENOMENOLOGY_216: (8)                     return self.time
SNGT_QHENOMENOLOGY_217: (4)                 def increment_time(self, dt: float) -> None:
SNGT_QHENOMENOLOGY_218: (8)                     self.time += dt
SNGT_QHENOMENOLOGY_219: (4)                 def get_top_level_mobjects(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_220: (8)                     mobjects = self.get_mobjects()
SNGT_QHENOMENOLOGY_221: (8)                     families = [m.get_family() for m in mobjects]
SNGT_QHENOMENOLOGY_222: (8)                     def is_top_level(mobject):
SNGT_QHENOMENOLOGY_223: (12)                        num_families = sum([
SNGT_QHENOMENOLOGY_224: (16)                            (mobject in family)
SNGT_QHENOMENOLOGY_225: (16)                            for family in families
SNGT_QHENOMENOLOGY_226: (12)                        ])
SNGT_QHENOMENOLOGY_227: (12)                        return num_families == 1
SNGT_QHENOMENOLOGY_228: (8)                     return list(filter(is_top_level, mobjects))
SNGT_QHENOMENOLOGY_229: (4)                 def get_mobject_family_members(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_230: (8)                     return
```

```
extract_mobject_family_members(self.mobjects)
SNGT_QHENOMENOLOGY_231: (4)                def assemble_render_groups(self):
SNGT_QHENOMENOLOGY_232: (8)                    """
SNGT_QHENOMENOLOGY_233: (8)                    Rendering can be more efficient when mobjects of
the
SNGT_QHENOMENOLOGY_234: (8)                    same type are grouped together, so this function
creates
SNGT_QHENOMENOLOGY_235: (8)                    Groups of all clusters of adjacent Mobjects in the
scene
SNGT_QHENOMENOLOGY_236: (8)                    """
SNGT_QHENOMENOLOGY_237: (8)                    batches = batch_by_property(
SNGT_QHENOMENOLOGY_238: (12)                       self.mobjects,
SNGT_QHENOMENOLOGY_239: (12)                       lambda m: str(type(m)) +
str(m.get_shader_wrapper(self.camera.ctx).get_id()) + str(m.z_index)
SNGT_QHENOMENOLOGY_240: (8)                    )
SNGT_QHENOMENOLOGY_241: (8)                    for group in self.render_groups:
SNGT_QHENOMENOLOGY_242: (12)                       group.clear()
SNGT_QHENOMENOLOGY_243: (8)                    self.render_groups = [
SNGT_QHENOMENOLOGY_244: (12)                       batch[0].get_group_class()(*batch)
SNGT_QHENOMENOLOGY_245: (12)                       for batch, key in batches
SNGT_QHENOMENOLOGY_246: (8)                    ]
SNGT_QHENOMENOLOGY_247: (4)                @staticmethod
SNGT_QHENOMENOLOGY_248: (4)                def affects_mobject_list(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_249: (8)                    @wraps(func)
SNGT_QHENOMENOLOGY_250: (8)                    def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_251: (12)                       func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_252: (12)                       self.assemble_render_groups()
SNGT_QHENOMENOLOGY_253: (12)                       return self
SNGT_QHENOMENOLOGY_254: (8)                    return wrapper
SNGT_QHENOMENOLOGY_255: (4)                @affects_mobject_list
SNGT_QHENOMENOLOGY_256: (4)                def add(self, *new_mobjects: Mobject):
SNGT_QHENOMENOLOGY_257: (8)                    """
SNGT_QHENOMENOLOGY_258: (8)                    Mobjects will be displayed, from background to
SNGT_QHENOMENOLOGY_259: (8)                    foreground in the order with which they are added.
SNGT_QHENOMENOLOGY_260: (8)                    """
SNGT_QHENOMENOLOGY_261: (8)                    self.remove(*new_mobjects)
SNGT_QHENOMENOLOGY_262: (8)                    self.mobjects += new_mobjects
SNGT_QHENOMENOLOGY_263: (8)                    id_to_scene_order = {id(m): idx for idx, m in
enumerate(self.mobjects)}
SNGT_QHENOMENOLOGY_264: (8)                    self.mobjects.sort(key=lambda m: (m.z_index,
id_to_scene_order[id(m)]))
SNGT_QHENOMENOLOGY_265: (8)                    self.id_to_mobject_map.update({
SNGT_QHENOMENOLOGY_266: (12)                       id(sm): sm
SNGT_QHENOMENOLOGY_267: (12)                       for m in new_mobjects
SNGT_QHENOMENOLOGY_268: (12)                       for sm in m.get_family()
SNGT_QHENOMENOLOGY_269: (8)                    })
SNGT_QHENOMENOLOGY_270: (8)                    return self
SNGT_QHENOMENOLOGY_271: (4)                def add_mobjects_among(self, values: Iterable):
SNGT_QHENOMENOLOGY_272: (8)                    """
SNGT_QHENOMENOLOGY_273: (8)                    This is meant mostly for quick prototyping,
SNGT_QHENOMENOLOGY_274: (8)                    e.g. to add all mobjects defined up to a point,
SNGT_QHENOMENOLOGY_275: (8)                    call self.add_mobjects_among(locals().values())
SNGT_QHENOMENOLOGY_276: (8)                    """
SNGT_QHENOMENOLOGY_277: (8)                    self.add(*filter(
SNGT_QHENOMENOLOGY_278: (12)                       lambda m: isinstance(m, Mobject),
SNGT_QHENOMENOLOGY_279: (12)                       values
SNGT_QHENOMENOLOGY_280: (8)                    ))
SNGT_QHENOMENOLOGY_281: (8)                    return self
SNGT_QHENOMENOLOGY_282: (4)                @affects_mobject_list
SNGT_QHENOMENOLOGY_283: (4)                def replace(self, mobject: Mobject, *replacements:
Mobject):
SNGT_QHENOMENOLOGY_284: (8)                    if mobject in self.mobjects:
SNGT_QHENOMENOLOGY_285: (12)                       index = self.mobjects.index(mobject)
SNGT_QHENOMENOLOGY_286: (12)                       self.mobjects = [
SNGT_QHENOMENOLOGY_287: (16)                           *self.mobjects[:index],
SNGT_QHENOMENOLOGY_288: (16)                           *replacements,
SNGT_QHENOMENOLOGY_289: (16)                           *self.mobjects[index + 1:]
SNGT_QHENOMENOLOGY_290: (12)                       ]
```

```
SNGT_QHENOMENOLOGY_291: (8)                     return self
SNGT_QHENOMENOLOGY_292: (4)                 @affects_mobject_list
SNGT_QHENOMENOLOGY_293: (4)                 def remove(self, *mobjects_to_remove: Mobject):
SNGT_QHENOMENOLOGY_294: (8)                     """
SNGT_QHENOMENOLOGY_295: (8)                     Removes anything in mobjects from scenes mobject
list, but in the event that one
SNGT_QHENOMENOLOGY_296: (8)                     of the items to be removed is a member of the
family of an item in mobject_list,
SNGT_QHENOMENOLOGY_297: (8)                     the other family members are added back into the
list.
SNGT_QHENOMENOLOGY_298: (8)                     For example, if the scene includes Group(m1, m2,
m3), and we call scene.remove(m1),
SNGT_QHENOMENOLOGY_299: (8)                     the desired behavior is for the scene to then
include m2 and m3 (ungrouped).
SNGT_QHENOMENOLOGY_300: (8)                     """
SNGT_QHENOMENOLOGY_301: (8)                     to_remove =
set(extract_mobject_family_members(mobjects_to_remove))
SNGT_QHENOMENOLOGY_302: (8)                     new_mobjects, _ =
recursive_mobject_remove(self.mobjects, to_remove)
SNGT_QHENOMENOLOGY_303: (8)                     self.mobjects = new_mobjects
SNGT_QHENOMENOLOGY_304: (4)                 def bring_to_front(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_305: (8)                     self.add(*mobjects)
SNGT_QHENOMENOLOGY_306: (8)                     return self
SNGT_QHENOMENOLOGY_307: (4)                 @affects_mobject_list
SNGT_QHENOMENOLOGY_308: (4)                 def bring_to_back(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_309: (8)                     self.remove(*mobjects)
SNGT_QHENOMENOLOGY_310: (8)                     self.mobjects = list(mobjects) + self.mobjects
SNGT_QHENOMENOLOGY_311: (8)                     return self
SNGT_QHENOMENOLOGY_312: (4)                 @affects_mobject_list
SNGT_QHENOMENOLOGY_313: (4)                 def clear(self):
SNGT_QHENOMENOLOGY_314: (8)                     self.mobjects = []
SNGT_QHENOMENOLOGY_315: (8)                     return self
SNGT_QHENOMENOLOGY_316: (4)                 def get_mobjects(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_317: (8)                     return list(self.mobjects)
SNGT_QHENOMENOLOGY_318: (4)                 def get_mobject_copies(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_319: (8)                     return [m.copy() for m in self.mobjects]
SNGT_QHENOMENOLOGY_320: (4)                 def point_to_mobject(
SNGT_QHENOMENOLOGY_321: (8)                     self,
SNGT_QHENOMENOLOGY_322: (8)                     point: np.ndarray,
SNGT_QHENOMENOLOGY_323: (8)                     search_set: Iterable[Mobject] | None = None,
SNGT_QHENOMENOLOGY_324: (8)                     buff: float = 0
SNGT_QHENOMENOLOGY_325: (4)                 ) -> Mobject | None:
SNGT_QHENOMENOLOGY_326: (8)                     """
SNGT_QHENOMENOLOGY_327: (8)                     E.g. if clicking on the scene, this returns the top
layer mobject
SNGT_QHENOMENOLOGY_328: (8)                     under a given point
SNGT_QHENOMENOLOGY_329: (8)                     """
SNGT_QHENOMENOLOGY_330: (8)                     if search_set is None:
SNGT_QHENOMENOLOGY_331: (12)                        search_set = self.mobjects
SNGT_QHENOMENOLOGY_332: (8)                     for mobject in reversed(search_set):
SNGT_QHENOMENOLOGY_333: (12)                        if mobject.is_point_touching(point, buff=buff):
SNGT_QHENOMENOLOGY_334: (16)                            return mobject
SNGT_QHENOMENOLOGY_335: (8)                     return None
SNGT_QHENOMENOLOGY_336: (4)                 def get_group(self, *mobjects):
SNGT_QHENOMENOLOGY_337: (8)                     if all(isinstance(m, VMobject) for m in mobjects):
SNGT_QHENOMENOLOGY_338: (12)                        return VGroup(*mobjects)
SNGT_QHENOMENOLOGY_339: (8)                     else:
SNGT_QHENOMENOLOGY_340: (12)                        return Group(*mobjects)
SNGT_QHENOMENOLOGY_341: (4)                 def id_to_mobject(self, id_value):
SNGT_QHENOMENOLOGY_342: (8)                     return self.id_to_mobject_map[id_value]
SNGT_QHENOMENOLOGY_343: (4)                 def ids_to_group(self, *id_values):
SNGT_QHENOMENOLOGY_344: (8)                     return self.get_group(*filter(
SNGT_QHENOMENOLOGY_345: (12)                        lambda x: x is not None,
SNGT_QHENOMENOLOGY_346: (12)                        map(self.id_to_mobject, id_values)
SNGT_QHENOMENOLOGY_347: (8)                     ))
SNGT_QHENOMENOLOGY_348: (4)                 def i2g(self, *id_values):
SNGT_QHENOMENOLOGY_349: (8)                     return self.ids_to_group(*id_values)
SNGT_QHENOMENOLOGY_350: (4)                 def i2m(self, id_value):
SNGT_QHENOMENOLOGY_351: (8)                     return self.id_to_mobject(id_value)
```

```
SNGT_QHENOMENOLOGY_352: (4)                def update_skipping_status(self) -> None:
SNGT_QHENOMENOLOGY_353: (8)                    if self.start_at_animation_number is not None:
SNGT_QHENOMENOLOGY_354: (12)                       if self.num_plays ==
self.start_at_animation_number:
SNGT_QHENOMENOLOGY_355: (16)                           self.skip_time = self.time
SNGT_QHENOMENOLOGY_356: (16)                           if not self.original_skipping_status:
SNGT_QHENOMENOLOGY_357: (20)                               self.stop_skipping()
SNGT_QHENOMENOLOGY_358: (8)                    if self.end_at_animation_number is not None:
SNGT_QHENOMENOLOGY_359: (12)                       if self.num_plays >=
self.end_at_animation_number:
SNGT_QHENOMENOLOGY_360: (16)                           raise EndScene()
SNGT_QHENOMENOLOGY_361: (4)                def stop_skipping(self) -> None:
SNGT_QHENOMENOLOGY_362: (8)                    self.virtual_animation_start_time = self.time
SNGT_QHENOMENOLOGY_363: (8)                    self.real_animation_start_time = time.time()
SNGT_QHENOMENOLOGY_364: (8)                    self.skip_animations = False
SNGT_QHENOMENOLOGY_365: (4)                def get_time_progression(
SNGT_QHENOMENOLOGY_366: (8)                    self,
SNGT_QHENOMENOLOGY_367: (8)                    run_time: float,
SNGT_QHENOMENOLOGY_368: (8)                    n_iterations: int | None = None,
SNGT_QHENOMENOLOGY_369: (8)                    desc: str = "",
SNGT_QHENOMENOLOGY_370: (8)                    override_skip_animations: bool = False
SNGT_QHENOMENOLOGY_371: (4)                ) -> list[float] | np.ndarray | ProgressDisplay:
SNGT_QHENOMENOLOGY_372: (8)                    if self.skip_animations and not
override_skip_animations:
SNGT_QHENOMENOLOGY_373: (12)                       return [run_time]
SNGT_QHENOMENOLOGY_374: (8)                    times = np.arange(0, run_time, 1 / self.camera.fps)
+ 1 / self.camera.fps
SNGT_QHENOMENOLOGY_375: (8)
self.file_writer.set_progress_display_description(sub_desc=desc)
SNGT_QHENOMENOLOGY_376: (8)                    if self.show_animation_progress:
SNGT_QHENOMENOLOGY_377: (12)                       return ProgressDisplay(
SNGT_QHENOMENOLOGY_378: (16)                           times,
SNGT_QHENOMENOLOGY_379: (16)                           total=n_iterations,
SNGT_QHENOMENOLOGY_380: (16)                           leave=self.leave_progress_bars,
SNGT_QHENOMENOLOGY_381: (16)                           ascii=True if platform.system() ==
'Windows' else None,
SNGT_QHENOMENOLOGY_382: (16)                           desc=desc,
SNGT_QHENOMENOLOGY_383: (16)                           bar_format="{l_bar} {n_fmt:3}/{total_fmt:3}
{rate_fmt}{postfix}",
SNGT_QHENOMENOLOGY_384: (12)                       )
SNGT_QHENOMENOLOGY_385: (8)                    else:
SNGT_QHENOMENOLOGY_386: (12)                       return times
SNGT_QHENOMENOLOGY_387: (4)                def get_run_time(self, animations: Iterable[Animation])
-> float:
SNGT_QHENOMENOLOGY_388: (8)                    return np.max([animation.get_run_time() for
animation in animations])
SNGT_QHENOMENOLOGY_389: (4)                def get_animation_time_progression(
SNGT_QHENOMENOLOGY_390: (8)                    self,
SNGT_QHENOMENOLOGY_391: (8)                    animations: Iterable[Animation]
SNGT_QHENOMENOLOGY_392: (4)                ) -> list[float] | np.ndarray | ProgressDisplay:
SNGT_QHENOMENOLOGY_393: (8)                    animations = list(animations)
SNGT_QHENOMENOLOGY_394: (8)                    run_time = self.get_run_time(animations)
SNGT_QHENOMENOLOGY_395: (8)                    description = f"{self.num_plays} {animations[0]}"
SNGT_QHENOMENOLOGY_396: (8)                    if len(animations) > 1:
SNGT_QHENOMENOLOGY_397: (12)                       description += ", etc."
SNGT_QHENOMENOLOGY_398: (8)                    time_progression =
self.get_time_progression(run_time, desc=description)
SNGT_QHENOMENOLOGY_399: (8)                    return time_progression
SNGT_QHENOMENOLOGY_400: (4)                def get_wait_time_progression(
SNGT_QHENOMENOLOGY_401: (8)                    self,
SNGT_QHENOMENOLOGY_402: (8)                    duration: float,
SNGT_QHENOMENOLOGY_403: (8)                    stop_condition: Callable[[], bool] | None = None
SNGT_QHENOMENOLOGY_404: (4)                ) -> list[float] | np.ndarray | ProgressDisplay:
SNGT_QHENOMENOLOGY_405: (8)                    kw = {"desc": f"{self.num_plays} Waiting"}
SNGT_QHENOMENOLOGY_406: (8)                    if stop_condition is not None:
SNGT_QHENOMENOLOGY_407: (12)                       kw["n_iterations"] = -1  # So it doesn't show %
progress
SNGT_QHENOMENOLOGY_408: (12)                       kw["override_skip_animations"] = True
SNGT_QHENOMENOLOGY_409: (8)                    return self.get_time_progression(duration, **kw)
```

```
SNGT_QHENOMENOLOGY_410: (4)                 def pre_play(self):
SNGT_QHENOMENOLOGY_411: (8)                     if self.presenter_mode and self.num_plays == 0:
SNGT_QHENOMENOLOGY_412: (12)                        self.hold_loop()
SNGT_QHENOMENOLOGY_413: (8)                     self.update_skipping_status()
SNGT_QHENOMENOLOGY_414: (8)                     if not self.skip_animations:
SNGT_QHENOMENOLOGY_415: (12)                        self.file_writer.begin_animation()
SNGT_QHENOMENOLOGY_416: (8)                     if self.window:
SNGT_QHENOMENOLOGY_417: (12)                        self.virtual_animation_start_time = self.time
SNGT_QHENOMENOLOGY_418: (12)                        self.real_animation_start_time = time.time()
SNGT_QHENOMENOLOGY_419: (4)                 def post_play(self):
SNGT_QHENOMENOLOGY_420: (8)                     if not self.skip_animations:
SNGT_QHENOMENOLOGY_421: (12)                        self.file_writer.end_animation()
SNGT_QHENOMENOLOGY_422: (8)                     if self.preview_while_skipping and
self.skip_animations and self.window is not None:
SNGT_QHENOMENOLOGY_423: (12)                        self.update_frame(dt=0, force_draw=True)
SNGT_QHENOMENOLOGY_424: (8)                     self.num_plays += 1
SNGT_QHENOMENOLOGY_425: (4)                 def begin_animations(self, animations:
Iterable[Animation]) -> None:
SNGT_QHENOMENOLOGY_426: (8)                     all_mobjects =
set(self.get_mobject_family_members())
SNGT_QHENOMENOLOGY_427: (8)                     for animation in animations:
SNGT_QHENOMENOLOGY_428: (12)                        animation.begin()
SNGT_QHENOMENOLOGY_429: (12)                        if animation.mobject not in all_mobjects:
SNGT_QHENOMENOLOGY_430: (16)                            self.add(animation.mobject)
SNGT_QHENOMENOLOGY_431: (16)                            all_mobjects =
all_mobjects.union(animation.mobject.get_family())
SNGT_QHENOMENOLOGY_432: (4)                 def progress_through_animations(self, animations:
Iterable[Animation]) -> None:
SNGT_QHENOMENOLOGY_433: (8)                     last_t = 0
SNGT_QHENOMENOLOGY_434: (8)                     for t in
self.get_animation_time_progression(animations):
SNGT_QHENOMENOLOGY_435: (12)                        dt = t - last_t
SNGT_QHENOMENOLOGY_436: (12)                        last_t = t
SNGT_QHENOMENOLOGY_437: (12)                        for animation in animations:
SNGT_QHENOMENOLOGY_438: (16)                            animation.update_mobjects(dt)
SNGT_QHENOMENOLOGY_439: (16)                            alpha = t / animation.run_time
SNGT_QHENOMENOLOGY_440: (16)                            animation.interpolate(alpha)
SNGT_QHENOMENOLOGY_441: (12)                        self.update_frame(dt)
SNGT_QHENOMENOLOGY_442: (12)                        self.emit_frame()
SNGT_QHENOMENOLOGY_443: (4)                 def finish_animations(self, animations:
Iterable[Animation]) -> None:
SNGT_QHENOMENOLOGY_444: (8)                     for animation in animations:
SNGT_QHENOMENOLOGY_445: (12)                        animation.finish()
SNGT_QHENOMENOLOGY_446: (12)                        animation.clean_up_from_scene(self)
SNGT_QHENOMENOLOGY_447: (8)                     if self.skip_animations:
SNGT_QHENOMENOLOGY_448: (12)
self.update_mobjects(self.get_run_time(animations))
SNGT_QHENOMENOLOGY_449: (8)                     else:
SNGT_QHENOMENOLOGY_450: (12)                        self.update_mobjects(0)
SNGT_QHENOMENOLOGY_451: (4)                 @affects_mobject_list
SNGT_QHENOMENOLOGY_452: (4)                 def play(
SNGT_QHENOMENOLOGY_453: (8)                     self,
SNGT_QHENOMENOLOGY_454: (8)                     *proto_animations: Animation | _AnimationBuilder,
SNGT_QHENOMENOLOGY_455: (8)                     run_time: float | None = None,
SNGT_QHENOMENOLOGY_456: (8)                     rate_func: Callable[[float], float] | None = None,
SNGT_QHENOMENOLOGY_457: (8)                     lag_ratio: float | None = None,
SNGT_QHENOMENOLOGY_458: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_459: (8)                     if len(proto_animations) == 0:
SNGT_QHENOMENOLOGY_460: (12)                        log.warning("Called Scene.play with no
animations")
SNGT_QHENOMENOLOGY_461: (12)                        return
SNGT_QHENOMENOLOGY_462: (8)                     animations = list(map(prepare_animation,
proto_animations))
SNGT_QHENOMENOLOGY_463: (8)                     for anim in animations:
SNGT_QHENOMENOLOGY_464: (12)                        anim.update_rate_info(run_time, rate_func,
lag_ratio)
SNGT_QHENOMENOLOGY_465: (8)                     self.pre_play()
SNGT_QHENOMENOLOGY_466: (8)                     self.begin_animations(animations)
SNGT_QHENOMENOLOGY_467: (8)                     self.progress_through_animations(animations)
```

```
SNGT_QHENOMENOLOGY_468: (8)                     self.finish_animations(animations)
SNGT_QHENOMENOLOGY_469: (8)                     self.post_play()
SNGT_QHENOMENOLOGY_470: (4)                 def wait(
SNGT_QHENOMENOLOGY_471: (8)                     self,
SNGT_QHENOMENOLOGY_472: (8)                     duration: Optional[float] = None,
SNGT_QHENOMENOLOGY_473: (8)                     stop_condition: Callable[[], bool] = None,
SNGT_QHENOMENOLOGY_474: (8)                     note: str = None,
SNGT_QHENOMENOLOGY_475: (8)                     ignore_presenter_mode: bool = False
SNGT_QHENOMENOLOGY_476: (4)                 ):
SNGT_QHENOMENOLOGY_477: (8)                     if duration is None:
SNGT_QHENOMENOLOGY_478: (12)                        duration = self.default_wait_time
SNGT_QHENOMENOLOGY_479: (8)                     self.pre_play()
SNGT_QHENOMENOLOGY_480: (8)                     self.update_mobjects(dt=0)  # Any problems with
this?
SNGT_QHENOMENOLOGY_481: (8)                     if self.presenter_mode and not self.skip_animations
and not ignore_presenter_mode:
SNGT_QHENOMENOLOGY_482: (12)                        if note:
SNGT_QHENOMENOLOGY_483: (16)                            log.info(note)
SNGT_QHENOMENOLOGY_484: (12)                        self.hold_loop()
SNGT_QHENOMENOLOGY_485: (8)                     else:
SNGT_QHENOMENOLOGY_486: (12)                        time_progression =
self.get_wait_time_progression(duration, stop_condition)
SNGT_QHENOMENOLOGY_487: (12)                        last_t = 0
SNGT_QHENOMENOLOGY_488: (12)                        for t in time_progression:
SNGT_QHENOMENOLOGY_489: (16)                            dt = t - last_t
SNGT_QHENOMENOLOGY_490: (16)                            last_t = t
SNGT_QHENOMENOLOGY_491: (16)                            self.update_frame(dt)
SNGT_QHENOMENOLOGY_492: (16)                            self.emit_frame()
SNGT_QHENOMENOLOGY_493: (16)                            if stop_condition is not None and
stop_condition():
SNGT_QHENOMENOLOGY_494: (20)                                break
SNGT_QHENOMENOLOGY_495: (8)                     self.post_play()
SNGT_QHENOMENOLOGY_496: (4)                 def hold_loop(self):
SNGT_QHENOMENOLOGY_497: (8)                     while self.hold_on_wait:
SNGT_QHENOMENOLOGY_498: (12)                        self.update_frame(dt=1 / self.camera.fps)
SNGT_QHENOMENOLOGY_499: (8)                     self.hold_on_wait = True
SNGT_QHENOMENOLOGY_500: (4)                 def wait_until(
SNGT_QHENOMENOLOGY_501: (8)                     self,
SNGT_QHENOMENOLOGY_502: (8)                     stop_condition: Callable[[], bool],
SNGT_QHENOMENOLOGY_503: (8)                     max_time: float = 60
SNGT_QHENOMENOLOGY_504: (4)                 ):
SNGT_QHENOMENOLOGY_505: (8)                     self.wait(max_time, stop_condition=stop_condition)
SNGT_QHENOMENOLOGY_506: (4)                 def force_skipping(self):
SNGT_QHENOMENOLOGY_507: (8)                     self.original_skipping_status =
self.skip_animations
SNGT_QHENOMENOLOGY_508: (8)                     self.skip_animations = True
SNGT_QHENOMENOLOGY_509: (8)                     return self
SNGT_QHENOMENOLOGY_510: (4)                 def revert_to_original_skipping_status(self):
SNGT_QHENOMENOLOGY_511: (8)                     if hasattr(self, "original_skipping_status"):
SNGT_QHENOMENOLOGY_512: (12)                        self.skip_animations =
self.original_skipping_status
SNGT_QHENOMENOLOGY_513: (8)                     return self
SNGT_QHENOMENOLOGY_514: (4)                 def add_sound(
SNGT_QHENOMENOLOGY_515: (8)                     self,
SNGT_QHENOMENOLOGY_516: (8)                     sound_file: str,
SNGT_QHENOMENOLOGY_517: (8)                     time_offset: float = 0,
SNGT_QHENOMENOLOGY_518: (8)                     gain: float | None = None,
SNGT_QHENOMENOLOGY_519: (8)                     gain_to_background: float | None = None
SNGT_QHENOMENOLOGY_520: (4)                 ):
SNGT_QHENOMENOLOGY_521: (8)                     if self.skip_animations:
SNGT_QHENOMENOLOGY_522: (12)                        return
SNGT_QHENOMENOLOGY_523: (8)                     time = self.get_time() + time_offset
SNGT_QHENOMENOLOGY_524: (8)                     self.file_writer.add_sound(sound_file, time, gain,
gain_to_background)
SNGT_QHENOMENOLOGY_525: (4)                 def get_state(self) -> SceneState:
SNGT_QHENOMENOLOGY_526: (8)                     return SceneState(self)
SNGT_QHENOMENOLOGY_527: (4)                 @affects_mobject_list
SNGT_QHENOMENOLOGY_528: (4)                 def restore_state(self, scene_state: SceneState):
SNGT_QHENOMENOLOGY_529: (8)                     scene_state.restore_scene(self)
```

```
SNGT_QHENOMENOLOGY_530: (4)                def save_state(self) -> None:
SNGT_QHENOMENOLOGY_531: (8)                    state = self.get_state()
SNGT_QHENOMENOLOGY_532: (8)                    if self.undo_stack and
state.mobjects_match(self.undo_stack[-1]):
SNGT_QHENOMENOLOGY_533: (12)                       return
SNGT_QHENOMENOLOGY_534: (8)                    self.redo_stack = []
SNGT_QHENOMENOLOGY_535: (8)                    self.undo_stack.append(state)
SNGT_QHENOMENOLOGY_536: (8)                    if len(self.undo_stack) >
self.max_num_saved_states:
SNGT_QHENOMENOLOGY_537: (12)                       self.undo_stack.pop(0)
SNGT_QHENOMENOLOGY_538: (4)                def undo(self):
SNGT_QHENOMENOLOGY_539: (8)                    if self.undo_stack:
SNGT_QHENOMENOLOGY_540: (12)                       self.redo_stack.append(self.get_state())
SNGT_QHENOMENOLOGY_541: (12)                       self.restore_state(self.undo_stack.pop())
SNGT_QHENOMENOLOGY_542: (4)                def redo(self):
SNGT_QHENOMENOLOGY_543: (8)                    if self.redo_stack:
SNGT_QHENOMENOLOGY_544: (12)                       self.undo_stack.append(self.get_state())
SNGT_QHENOMENOLOGY_545: (12)                       self.restore_state(self.redo_stack.pop())
SNGT_QHENOMENOLOGY_546: (4)                @contextmanager
SNGT_QHENOMENOLOGY_547: (4)                def temp_skip(self):
SNGT_QHENOMENOLOGY_548: (8)                    prev_status = self.skip_animations
SNGT_QHENOMENOLOGY_549: (8)                    self.skip_animations = True
SNGT_QHENOMENOLOGY_550: (8)                    try:
SNGT_QHENOMENOLOGY_551: (12)                       yield
SNGT_QHENOMENOLOGY_552: (8)                    finally:
SNGT_QHENOMENOLOGY_553: (12)                       if not prev_status:
SNGT_QHENOMENOLOGY_554: (16)                           self.stop_skipping()
SNGT_QHENOMENOLOGY_555: (4)                @contextmanager
SNGT_QHENOMENOLOGY_556: (4)                def temp_progress_bar(self):
SNGT_QHENOMENOLOGY_557: (8)                    prev_progress = self.show_animation_progress
SNGT_QHENOMENOLOGY_558: (8)                    self.show_animation_progress = True
SNGT_QHENOMENOLOGY_559: (8)                    try:
SNGT_QHENOMENOLOGY_560: (12)                       yield
SNGT_QHENOMENOLOGY_561: (8)                    finally:
SNGT_QHENOMENOLOGY_562: (12)                       self.show_animation_progress = prev_progress
SNGT_QHENOMENOLOGY_563: (4)                @contextmanager
SNGT_QHENOMENOLOGY_564: (4)                def temp_record(self):
SNGT_QHENOMENOLOGY_565: (8)                    self.camera.use_window_fbo(False)
SNGT_QHENOMENOLOGY_566: (8)                    self.file_writer.begin_insert()
SNGT_QHENOMENOLOGY_567: (8)                    try:
SNGT_QHENOMENOLOGY_568: (12)                       yield
SNGT_QHENOMENOLOGY_569: (8)                    finally:
SNGT_QHENOMENOLOGY_570: (12)                       self.file_writer.end_insert()
SNGT_QHENOMENOLOGY_571: (12)                       self.camera.use_window_fbo(True)
SNGT_QHENOMENOLOGY_572: (4)                def temp_config_change(self, skip=False, record=False,
progress_bar=False):
SNGT_QHENOMENOLOGY_573: (8)                    stack = ExitStack()
SNGT_QHENOMENOLOGY_574: (8)                    if skip:
SNGT_QHENOMENOLOGY_575: (12)                       stack.enter_context(self.temp_skip())
SNGT_QHENOMENOLOGY_576: (8)                    if record:
SNGT_QHENOMENOLOGY_577: (12)                       stack.enter_context(self.temp_record())
SNGT_QHENOMENOLOGY_578: (8)                    if progress_bar:
SNGT_QHENOMENOLOGY_579: (12)                       stack.enter_context(self.temp_progress_bar())
SNGT_QHENOMENOLOGY_580: (8)                    return stack
SNGT_QHENOMENOLOGY_581: (4)                def is_window_closing(self):
SNGT_QHENOMENOLOGY_582: (8)                    return self.window and (self.window.is_closing or
self.quit_interaction)
SNGT_QHENOMENOLOGY_583: (4)                def set_floor_plane(self, plane: str = "xy"):
SNGT_QHENOMENOLOGY_584: (8)                    if plane == "xy":
SNGT_QHENOMENOLOGY_585: (12)                       self.frame.set_euler_axes("zxz")
SNGT_QHENOMENOLOGY_586: (8)                    elif plane == "xz":
SNGT_QHENOMENOLOGY_587: (12)                       self.frame.set_euler_axes("zxy")
SNGT_QHENOMENOLOGY_588: (8)                    else:
SNGT_QHENOMENOLOGY_589: (12)                       raise Exception("Only `xz` and `xy` are valid
floor planes")
SNGT_QHENOMENOLOGY_590: (4)                def on_mouse_motion(
SNGT_QHENOMENOLOGY_591: (8)                    self,
SNGT_QHENOMENOLOGY_592: (8)                    point: Vect3,
SNGT_QHENOMENOLOGY_593: (8)                    d_point: Vect3
```

```
SNGT_QHENOMENOLOGY_594: (4)                ) -> None:
SNGT_QHENOMENOLOGY_595: (8)                    assert self.window is not None
SNGT_QHENOMENOLOGY_596: (8)                    self.mouse_point.move_to(point)
SNGT_QHENOMENOLOGY_597: (8)                    event_data = {"point": point, "d_point": d_point}
SNGT_QHENOMENOLOGY_598: (8)                    propagate_event =
EVENT_DISPATCHER.dispatch(EventType.MouseMotionEvent, **event_data)
SNGT_QHENOMENOLOGY_599: (8)                    if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_600: (12)                       return
SNGT_QHENOMENOLOGY_601: (8)                    frame = self.camera.frame
SNGT_QHENOMENOLOGY_602: (8)                    if
self.window.is_key_pressed(ord(manim_config.key_bindings.pan_3d)):
SNGT_QHENOMENOLOGY_603: (12)                       ff_d_point =
frame.to_fixed_frame_point(d_point, relative=True)
SNGT_QHENOMENOLOGY_604: (12)                       ff_d_point *= self.pan_sensitivity
SNGT_QHENOMENOLOGY_605: (12)                       frame.increment_theta(-ff_d_point[0])
SNGT_QHENOMENOLOGY_606: (12)                       frame.increment_phi(ff_d_point[1])
SNGT_QHENOMENOLOGY_607: (8)                    elif
self.window.is_key_pressed(ord(manim_config.key_bindings.pan)):
SNGT_QHENOMENOLOGY_608: (12)                       frame.shift(-d_point)
SNGT_QHENOMENOLOGY_609: (4)                def on_mouse_drag(
SNGT_QHENOMENOLOGY_610: (8)                    self,
SNGT_QHENOMENOLOGY_611: (8)                    point: Vect3,
SNGT_QHENOMENOLOGY_612: (8)                    d_point: Vect3,
SNGT_QHENOMENOLOGY_613: (8)                    buttons: int,
SNGT_QHENOMENOLOGY_614: (8)                    modifiers: int
SNGT_QHENOMENOLOGY_615: (4)                ) -> None:
SNGT_QHENOMENOLOGY_616: (8)                    self.mouse_drag_point.move_to(point)
SNGT_QHENOMENOLOGY_617: (8)                    if self.drag_to_pan:
SNGT_QHENOMENOLOGY_618: (12)                       self.frame.shift(-d_point)
SNGT_QHENOMENOLOGY_619: (8)                    event_data = {"point": point, "d_point": d_point,
"buttons": buttons, "modifiers": modifiers}
SNGT_QHENOMENOLOGY_620: (8)                    propagate_event =
EVENT_DISPATCHER.dispatch(EventType.MouseDragEvent, **event_data)
SNGT_QHENOMENOLOGY_621: (8)                    if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_622: (12)                       return
SNGT_QHENOMENOLOGY_623: (4)                def on_mouse_press(
SNGT_QHENOMENOLOGY_624: (8)                    self,
SNGT_QHENOMENOLOGY_625: (8)                    point: Vect3,
SNGT_QHENOMENOLOGY_626: (8)                    button: int,
SNGT_QHENOMENOLOGY_627: (8)                    mods: int
SNGT_QHENOMENOLOGY_628: (4)                ) -> None:
SNGT_QHENOMENOLOGY_629: (8)                    self.mouse_drag_point.move_to(point)
SNGT_QHENOMENOLOGY_630: (8)                    event_data = {"point": point, "button": button,
"mods": mods}
SNGT_QHENOMENOLOGY_631: (8)                    propagate_event =
EVENT_DISPATCHER.dispatch(EventType.MousePressEvent, **event_data)
SNGT_QHENOMENOLOGY_632: (8)                    if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_633: (12)                       return
SNGT_QHENOMENOLOGY_634: (4)                def on_mouse_release(
SNGT_QHENOMENOLOGY_635: (8)                    self,
SNGT_QHENOMENOLOGY_636: (8)                    point: Vect3,
SNGT_QHENOMENOLOGY_637: (8)                    button: int,
SNGT_QHENOMENOLOGY_638: (8)                    mods: int
SNGT_QHENOMENOLOGY_639: (4)                ) -> None:
SNGT_QHENOMENOLOGY_640: (8)                    event_data = {"point": point, "button": button,
"mods": mods}
SNGT_QHENOMENOLOGY_641: (8)                    propagate_event =
EVENT_DISPATCHER.dispatch(EventType.MouseReleaseEvent, **event_data)
SNGT_QHENOMENOLOGY_642: (8)                    if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_643: (12)                       return
SNGT_QHENOMENOLOGY_644: (4)                def on_mouse_scroll(
SNGT_QHENOMENOLOGY_645: (8)                    self,
SNGT_QHENOMENOLOGY_646: (8)                    point: Vect3,
SNGT_QHENOMENOLOGY_647: (8)                    offset: Vect3,
SNGT_QHENOMENOLOGY_648: (8)                    x_pixel_offset: float,
```

```
SNGT_QHENOMENOLOGY_649: (8)                     y_pixel_offset: float
SNGT_QHENOMENOLOGY_650: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_651: (8)                     event_data = {"point": point, "offset": offset}
SNGT_QHENOMENOLOGY_652: (8)                     propagate_event =
EVENT_DISPATCHER.dispatch(EventType.MouseScrollEvent, **event_data)
SNGT_QHENOMENOLOGY_653: (8)                     if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_654: (12)                        return
SNGT_QHENOMENOLOGY_655: (8)                     rel_offset = y_pixel_offset /
self.camera.get_pixel_height()
SNGT_QHENOMENOLOGY_656: (8)                     self.frame.scale(
SNGT_QHENOMENOLOGY_657: (12)                        1 - self.scroll_sensitivity * rel_offset,
SNGT_QHENOMENOLOGY_658: (12)                        about_point=point
SNGT_QHENOMENOLOGY_659: (8)                     )
SNGT_QHENOMENOLOGY_660: (4)                 def on_key_release(
SNGT_QHENOMENOLOGY_661: (8)                     self,
SNGT_QHENOMENOLOGY_662: (8)                     symbol: int,
SNGT_QHENOMENOLOGY_663: (8)                     modifiers: int
SNGT_QHENOMENOLOGY_664: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_665: (8)                     event_data = {"symbol": symbol, "modifiers":
modifiers}
SNGT_QHENOMENOLOGY_666: (8)                     propagate_event =
EVENT_DISPATCHER.dispatch(EventType.KeyReleaseEvent, **event_data)
SNGT_QHENOMENOLOGY_667: (8)                     if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_668: (12)                        return
SNGT_QHENOMENOLOGY_669: (4)                 def on_key_press(
SNGT_QHENOMENOLOGY_670: (8)                     self,
SNGT_QHENOMENOLOGY_671: (8)                     symbol: int,
SNGT_QHENOMENOLOGY_672: (8)                     modifiers: int
SNGT_QHENOMENOLOGY_673: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_674: (8)                     try:
SNGT_QHENOMENOLOGY_675: (12)                        char = chr(symbol)
SNGT_QHENOMENOLOGY_676: (8)                     except OverflowError:
SNGT_QHENOMENOLOGY_677: (12)                        log.warning("The value of the pressed key is
too large.")
SNGT_QHENOMENOLOGY_678: (12)                        return
SNGT_QHENOMENOLOGY_679: (8)                     event_data = {"symbol": symbol, "modifiers":
modifiers}
SNGT_QHENOMENOLOGY_680: (8)                     propagate_event =
EVENT_DISPATCHER.dispatch(EventType.KeyPressEvent, **event_data)
SNGT_QHENOMENOLOGY_681: (8)                     if propagate_event is not None and propagate_event
is False:
SNGT_QHENOMENOLOGY_682: (12)                        return
SNGT_QHENOMENOLOGY_683: (8)                     if char == manim_config.key_bindings.reset:
SNGT_QHENOMENOLOGY_684: (12)
self.play(self.camera.frame.animate.to_default_state())
SNGT_QHENOMENOLOGY_685: (8)                     elif char == "z" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_686: (12)                        self.undo()
SNGT_QHENOMENOLOGY_687: (8)                     elif char == "z" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL | PygletWindowKeys.MOD_SHIFT)):
SNGT_QHENOMENOLOGY_688: (12)                        self.redo()
SNGT_QHENOMENOLOGY_689: (8)                     elif char == manim_config.key_bindings.quit and
(modifiers & (PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_690: (12)                        self.quit_interaction = True
SNGT_QHENOMENOLOGY_691: (8)                     elif char == " " or symbol ==
PygletWindowKeys.RIGHT:
SNGT_QHENOMENOLOGY_692: (12)                        self.hold_on_wait = False
SNGT_QHENOMENOLOGY_693: (4)                 def on_resize(self, width: int, height: int) -> None:
SNGT_QHENOMENOLOGY_694: (8)                     pass
SNGT_QHENOMENOLOGY_695: (4)                 def on_show(self) -> None:
SNGT_QHENOMENOLOGY_696: (8)                     pass
SNGT_QHENOMENOLOGY_697: (4)                 def on_hide(self) -> None:
SNGT_QHENOMENOLOGY_698: (8)                     pass
SNGT_QHENOMENOLOGY_699: (4)                 def on_close(self) -> None:
SNGT_QHENOMENOLOGY_700: (8)                     pass
SNGT_QHENOMENOLOGY_701: (4)                 def focus(self) -> None:
SNGT_QHENOMENOLOGY_702: (8)                     """
```

```
SNGT_QHENOMENOLOGY_703: (8)                     Puts focus on the ManimGL window.
SNGT_QHENOMENOLOGY_704: (8)                     """
SNGT_QHENOMENOLOGY_705: (8)                     if not self.window:
SNGT_QHENOMENOLOGY_706: (12)                        return
SNGT_QHENOMENOLOGY_707: (8)                     self.window.focus()
SNGT_QHENOMENOLOGY_708: (0)             class SceneState():
SNGT_QHENOMENOLOGY_709: (4)                 def __init__(self, scene: Scene, ignore: list[Mobject]
| None = None):
SNGT_QHENOMENOLOGY_710: (8)                     self.time = scene.time
SNGT_QHENOMENOLOGY_711: (8)                     self.num_plays = scene.num_plays
SNGT_QHENOMENOLOGY_712: (8)                     self.mobjects_to_copies =
OrderedDict.fromkeys(scene.mobjects)
SNGT_QHENOMENOLOGY_713: (8)                     if ignore:
SNGT_QHENOMENOLOGY_714: (12)                        for mob in ignore:
SNGT_QHENOMENOLOGY_715: (16)                            self.mobjects_to_copies.pop(mob, None)
SNGT_QHENOMENOLOGY_716: (8)                     last_m2c = scene.undo_stack[-1].mobjects_to_copies
if scene.undo_stack else dict()
SNGT_QHENOMENOLOGY_717: (8)                     for mob in self.mobjects_to_copies:
SNGT_QHENOMENOLOGY_718: (12)                        if mob in last_m2c and
last_m2c[mob].looks_identical(mob):
SNGT_QHENOMENOLOGY_719: (16)                            self.mobjects_to_copies[mob] =
last_m2c[mob]
SNGT_QHENOMENOLOGY_720: (12)                        else:
SNGT_QHENOMENOLOGY_721: (16)                            self.mobjects_to_copies[mob] = mob.copy()
SNGT_QHENOMENOLOGY_722: (4)                 def __eq__(self, state: SceneState):
SNGT_QHENOMENOLOGY_723: (8)                     return all((
SNGT_QHENOMENOLOGY_724: (12)                        self.time == state.time,
SNGT_QHENOMENOLOGY_725: (12)                        self.num_plays == state.num_plays,
SNGT_QHENOMENOLOGY_726: (12)                        self.mobjects_to_copies ==
state.mobjects_to_copies
SNGT_QHENOMENOLOGY_727: (8)                     ))
SNGT_QHENOMENOLOGY_728: (4)                 def mobjects_match(self, state: SceneState):
SNGT_QHENOMENOLOGY_729: (8)                     return self.mobjects_to_copies ==
state.mobjects_to_copies
SNGT_QHENOMENOLOGY_730: (4)                 def n_changes(self, state: SceneState):
SNGT_QHENOMENOLOGY_731: (8)                     m2c = state.mobjects_to_copies
SNGT_QHENOMENOLOGY_732: (8)                     return sum(
SNGT_QHENOMENOLOGY_733: (12)                        1 - int(mob in m2c and
mob.looks_identical(m2c[mob]))
SNGT_QHENOMENOLOGY_734: (12)                        for mob in self.mobjects_to_copies
SNGT_QHENOMENOLOGY_735: (8)                     )
SNGT_QHENOMENOLOGY_736: (4)                 def restore_scene(self, scene: Scene):
SNGT_QHENOMENOLOGY_737: (8)                     scene.time = self.time
SNGT_QHENOMENOLOGY_738: (8)                     scene.num_plays = self.num_plays
SNGT_QHENOMENOLOGY_739: (8)                     scene.mobjects = [
SNGT_QHENOMENOLOGY_740: (12)                        mob.become(mob_copy)
SNGT_QHENOMENOLOGY_741: (12)                        for mob, mob_copy in
self.mobjects_to_copies.items()
SNGT_QHENOMENOLOGY_742: (8)                     ]
SNGT_QHENOMENOLOGY_743: (0)             class EndScene(Exception):
SNGT_QHENOMENOLOGY_744: (4)                 pass
SNGT_QHENOMENOLOGY_745: (0)             class ThreeDScene(Scene):
SNGT_QHENOMENOLOGY_746: (4)                 samples = 4
SNGT_QHENOMENOLOGY_747: (4)                 default_frame_orientation = (-30, 70)
SNGT_QHENOMENOLOGY_748: (4)                 always_depth_test = True
SNGT_QHENOMENOLOGY_749: (4)                 def add(self, *mobjects: Mobject, set_depth_test: bool
= True, perp_stroke: bool = True):
SNGT_QHENOMENOLOGY_750: (8)                     for mob in mobjects:
SNGT_QHENOMENOLOGY_751: (12)                        if set_depth_test and not
mob.is_fixed_in_frame() and self.always_depth_test:
SNGT_QHENOMENOLOGY_752: (16)                            mob.apply_depth_test()
SNGT_QHENOMENOLOGY_753: (12)                        if isinstance(mob, VMobject) and
mob.has_stroke() and perp_stroke:
SNGT_QHENOMENOLOGY_754: (16)                            mob.set_flat_stroke(False)
SNGT_QHENOMENOLOGY_755: (8)                     super().add(*mobjects)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 67 - __init__.py:
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 68 - scene_embed.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import inspect
SNGT_QHENOMENOLOGY_3: (0)              import pyperclip
SNGT_QHENOMENOLOGY_4: (0)              import traceback
SNGT_QHENOMENOLOGY_5: (0)              from IPython.terminal import pt_inputhooks
SNGT_QHENOMENOLOGY_6: (0)              from IPython.terminal.embed import InteractiveShellEmbed
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.animation.fading import VFadeInThenOut
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.constants import RED
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.mobject.frame import FullScreenRectangle
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.module_loader import ModuleLoader
SNGT_QHENOMENOLOGY_13: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_14: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_15: (4)                 from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_16: (0)             class InteractiveSceneEmbed:
SNGT_QHENOMENOLOGY_17: (4)                 def __init__(self, scene: Scene):
SNGT_QHENOMENOLOGY_18: (8)                     self.scene = scene
SNGT_QHENOMENOLOGY_19: (8)                     self.checkpoint_manager = CheckpointManager()
SNGT_QHENOMENOLOGY_20: (8)                     self.shell =
self.get_ipython_shell_for_embedded_scene()
SNGT_QHENOMENOLOGY_21: (8)                     self.enable_gui()
SNGT_QHENOMENOLOGY_22: (8)                     self.ensure_frame_update_post_cell()
SNGT_QHENOMENOLOGY_23: (8)                     self.ensure_flash_on_error()
SNGT_QHENOMENOLOGY_24: (8)                     if manim_config.embed.autoreload:
SNGT_QHENOMENOLOGY_25: (12)                        self.auto_reload()
SNGT_QHENOMENOLOGY_26: (4)                 def launch(self):
SNGT_QHENOMENOLOGY_27: (8)                     self.shell()
SNGT_QHENOMENOLOGY_28: (4)                 def get_ipython_shell_for_embedded_scene(self) ->
InteractiveShellEmbed:
SNGT_QHENOMENOLOGY_29: (8)                     """
SNGT_QHENOMENOLOGY_30: (8)                     Create embedded IPython terminal configured to have
access to
SNGT_QHENOMENOLOGY_31: (8)                     the local namespace of the caller
SNGT_QHENOMENOLOGY_32: (8)                     """
SNGT_QHENOMENOLOGY_33: (8)                     caller_frame =
inspect.currentframe().f_back.f_back.f_back
SNGT_QHENOMENOLOGY_34: (8)                     module =
ModuleLoader.get_module(caller_frame.f_globals["__file__"])
SNGT_QHENOMENOLOGY_35: (8)                     module.__dict__.update(caller_frame.f_locals)
SNGT_QHENOMENOLOGY_36: (8)                     module.__dict__.update(self.get_shortcuts())
SNGT_QHENOMENOLOGY_37: (8)                     exception_mode = manim_config.embed.exception_mode
SNGT_QHENOMENOLOGY_38: (8)                     return InteractiveShellEmbed(
SNGT_QHENOMENOLOGY_39: (12)                        user_module=module,
SNGT_QHENOMENOLOGY_40: (12)                        display_banner=False,
SNGT_QHENOMENOLOGY_41: (12)                        xmode=exception_mode
SNGT_QHENOMENOLOGY_42: (8)                     )
SNGT_QHENOMENOLOGY_43: (4)                 def get_shortcuts(self):
SNGT_QHENOMENOLOGY_44: (8)                     """
SNGT_QHENOMENOLOGY_45: (8)                     A few custom shortcuts useful to have in the
interactive shell namespace
SNGT_QHENOMENOLOGY_46: (8)                     """
SNGT_QHENOMENOLOGY_47: (8)                     scene = self.scene
SNGT_QHENOMENOLOGY_48: (8)                     return dict(
SNGT_QHENOMENOLOGY_49: (12)                        play=scene.play,
SNGT_QHENOMENOLOGY_50: (12)                        wait=scene.wait,
SNGT_QHENOMENOLOGY_51: (12)                        add=scene.add,
SNGT_QHENOMENOLOGY_52: (12)                        remove=scene.remove,
SNGT_QHENOMENOLOGY_53: (12)                        clear=scene.clear,
SNGT_QHENOMENOLOGY_54: (12)                        focus=scene.focus,
SNGT_QHENOMENOLOGY_55: (12)                        save_state=scene.save_state,
SNGT_QHENOMENOLOGY_56: (12)                        undo=scene.undo,
```

```
SNGT_QHENOMENOLOGY_57: (12)                         redo=scene.redo,
SNGT_QHENOMENOLOGY_58: (12)                         i2g=scene.i2g,
SNGT_QHENOMENOLOGY_59: (12)                         i2m=scene.i2m,
SNGT_QHENOMENOLOGY_60: (12)                         checkpoint_paste=self.checkpoint_paste,
SNGT_QHENOMENOLOGY_61: (12)
clear_checkpoints=self.checkpoint_manager.clear_checkpoints,
SNGT_QHENOMENOLOGY_62: (12)                         reload=self.reload_scene  # Defined below
SNGT_QHENOMENOLOGY_63: (8)                     )
SNGT_QHENOMENOLOGY_64: (4)                 def enable_gui(self):
SNGT_QHENOMENOLOGY_65: (8)                     """Enables gui interactions during the embed"""
SNGT_QHENOMENOLOGY_66: (8)                     def inputhook(context):
SNGT_QHENOMENOLOGY_67: (12)                         while not context.input_is_ready():
SNGT_QHENOMENOLOGY_68: (16)                             if not self.scene.is_window_closing():
SNGT_QHENOMENOLOGY_69: (20)                                 self.scene.update_frame(dt=0)
SNGT_QHENOMENOLOGY_70: (12)                         if self.scene.is_window_closing():
SNGT_QHENOMENOLOGY_71: (16)                             self.shell.ask_exit()
SNGT_QHENOMENOLOGY_72: (8)                     pt_inputhooks.register("manim", inputhook)
SNGT_QHENOMENOLOGY_73: (8)                     self.shell.enable_gui("manim")
SNGT_QHENOMENOLOGY_74: (4)                 def ensure_frame_update_post_cell(self):
SNGT_QHENOMENOLOGY_75: (8)                     """Ensure the scene updates its frame after each
ipython cell"""
SNGT_QHENOMENOLOGY_76: (8)                     def post_cell_func(*args, **kwargs):
SNGT_QHENOMENOLOGY_77: (12)                         if not self.scene.is_window_closing():
SNGT_QHENOMENOLOGY_78: (16)                             self.scene.update_frame(dt=0,
force_draw=True)
SNGT_QHENOMENOLOGY_79: (8)                     self.shell.events.register("post_run_cell",
post_cell_func)
SNGT_QHENOMENOLOGY_80: (4)                 def ensure_flash_on_error(self):
SNGT_QHENOMENOLOGY_81: (8)                     """Flash border, and potentially play sound, on
exceptions"""
SNGT_QHENOMENOLOGY_82: (8)                     def custom_exc(shell, etype, evalue, tb,
tb_offset=None):
SNGT_QHENOMENOLOGY_83: (12)                         shell.showtraceback((etype, evalue, tb),
tb_offset=tb_offset)
SNGT_QHENOMENOLOGY_84: (12)                         rect = FullScreenRectangle().set_stroke(RED,
30).set_fill(opacity=0)
SNGT_QHENOMENOLOGY_85: (12)                         rect.fix_in_frame()
SNGT_QHENOMENOLOGY_86: (12)                         self.scene.play(VFadeInThenOut(rect,
run_time=0.5))
SNGT_QHENOMENOLOGY_87: (8)                     self.shell.set_custom_exc((Exception,), custom_exc)
SNGT_QHENOMENOLOGY_88: (4)                 def reload_scene(self, embed_line: int | None = None) -
> None:
SNGT_QHENOMENOLOGY_89: (8)                     """
SNGT_QHENOMENOLOGY_90: (8)                     Reloads the scene just like the `manimgl` command
would do with the
SNGT_QHENOMENOLOGY_91: (8)                     same arguments that were provided for the initial
startup. This allows
SNGT_QHENOMENOLOGY_92: (8)                     for quick iteration during scene development since
we don't have to exit
SNGT_QHENOMENOLOGY_93: (8)                     the IPython kernel and re-run the `manimgl` command
again. The GUI stays
SNGT_QHENOMENOLOGY_94: (8)                     open during the reload.
SNGT_QHENOMENOLOGY_95: (8)                     If `embed_line` is provided, the scene will be
reloaded at that line
SNGT_QHENOMENOLOGY_96: (8)                     number. This corresponds to the `linemarker` param
of the
SNGT_QHENOMENOLOGY_97: (8)                     `extract_scene.insert_embed_line_to_module()`
method.
SNGT_QHENOMENOLOGY_98: (8)                     Before reload, the scene is cleared and the entire
state is reset, such
SNGT_QHENOMENOLOGY_99: (8)                     that we can start from a clean slate. This is taken
care of by the
SNGT_QHENOMENOLOGY_100: (8)                    run_scenes function in __main__.py, which will
catch the error raised by the
SNGT_QHENOMENOLOGY_101: (8)                    `exit_raise` magic command that we invoke here.
SNGT_QHENOMENOLOGY_102: (8)                    Note that we cannot define a custom exception class
for this error,
SNGT_QHENOMENOLOGY_103: (8)                    since the IPython kernel will swallow any
exception. While we can catch
```

```
SNGT_QHENOMENOLOGY_104: (8)                     such an exception in our custom exception handler
registered with the
SNGT_QHENOMENOLOGY_105: (8)                     `set_custom_exc` method, we cannot break out of the
IPython shell by
SNGT_QHENOMENOLOGY_106: (8)                     this means.
SNGT_QHENOMENOLOGY_107: (8)                     """
SNGT_QHENOMENOLOGY_108: (8)                     run_config = manim_config.run
SNGT_QHENOMENOLOGY_109: (8)                     run_config.is_reload = True
SNGT_QHENOMENOLOGY_110: (8)                     if embed_line:
SNGT_QHENOMENOLOGY_111: (12)                        run_config.embed_line = embed_line
SNGT_QHENOMENOLOGY_112: (8)                     print("Reloading...")
SNGT_QHENOMENOLOGY_113: (8)                     self.shell.run_line_magic("exit_raise", "")
SNGT_QHENOMENOLOGY_114: (4)                 def auto_reload(self):
SNGT_QHENOMENOLOGY_115: (8)                     """Enables reload the shell's module before all
calls"""
SNGT_QHENOMENOLOGY_116: (8)                     def pre_cell_func(*args, **kwargs):
SNGT_QHENOMENOLOGY_117: (12)                        new_mod =
ModuleLoader.get_module(self.shell.user_module.__file__, is_during_reload=True)
SNGT_QHENOMENOLOGY_118: (12)                        self.shell.user_ns.update(vars(new_mod))
SNGT_QHENOMENOLOGY_119: (8)                     self.shell.events.register("pre_run_cell",
pre_cell_func)
SNGT_QHENOMENOLOGY_120: (4)                 def checkpoint_paste(
SNGT_QHENOMENOLOGY_121: (8)                     self,
SNGT_QHENOMENOLOGY_122: (8)                     skip: bool = False,
SNGT_QHENOMENOLOGY_123: (8)                     record: bool = False,
SNGT_QHENOMENOLOGY_124: (8)                     progress_bar: bool = True
SNGT_QHENOMENOLOGY_125: (4)                 ):
SNGT_QHENOMENOLOGY_126: (8)                     with self.scene.temp_config_change(skip, record,
progress_bar):
SNGT_QHENOMENOLOGY_127: (12)
self.checkpoint_manager.checkpoint_paste(self.shell, self.scene)
SNGT_QHENOMENOLOGY_128: (0)             class CheckpointManager:
SNGT_QHENOMENOLOGY_129: (4)                 def __init__(self):
SNGT_QHENOMENOLOGY_130: (8)                     self.checkpoint_states: dict[str,
list[tuple[Mobject, Mobject]]] = dict()
SNGT_QHENOMENOLOGY_131: (4)                 def checkpoint_paste(self, shell, scene):
SNGT_QHENOMENOLOGY_132: (8)                     """
SNGT_QHENOMENOLOGY_133: (8)                     Used during interactive development to run (or re-
run)
SNGT_QHENOMENOLOGY_134: (8)                     a block of scene code.
SNGT_QHENOMENOLOGY_135: (8)                     If the copied selection starts with a comment, this
will
SNGT_QHENOMENOLOGY_136: (8)                     revert to the state of the scene the first time
this function
SNGT_QHENOMENOLOGY_137: (8)                     was called on a block of code starting with that
comment.
SNGT_QHENOMENOLOGY_138: (8)                     """
SNGT_QHENOMENOLOGY_139: (8)                     code_string = pyperclip.paste()
SNGT_QHENOMENOLOGY_140: (8)                     checkpoint_key =
self.get_leading_comment(code_string)
SNGT_QHENOMENOLOGY_141: (8)                     self.handle_checkpoint_key(scene, checkpoint_key)
SNGT_QHENOMENOLOGY_142: (8)                     shell.run_cell(code_string)
SNGT_QHENOMENOLOGY_143: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_144: (4)                 def get_leading_comment(code_string: str) -> str:
SNGT_QHENOMENOLOGY_145: (8)                     leading_line = code_string.partition("\n")
[0].lstrip()
SNGT_QHENOMENOLOGY_146: (8)                     if leading_line.startswith("#"):
SNGT_QHENOMENOLOGY_147: (12)                        return leading_line
SNGT_QHENOMENOLOGY_148: (8)                     return ""
SNGT_QHENOMENOLOGY_149: (4)                 def handle_checkpoint_key(self, scene, key: str):
SNGT_QHENOMENOLOGY_150: (8)                     if not key:
SNGT_QHENOMENOLOGY_151: (12)                        return
SNGT_QHENOMENOLOGY_152: (8)                     elif key in self.checkpoint_states:
SNGT_QHENOMENOLOGY_153: (12)
scene.restore_state(self.checkpoint_states[key])
SNGT_QHENOMENOLOGY_154: (12)                        all_keys = list(self.checkpoint_states.keys())
SNGT_QHENOMENOLOGY_155: (12)                        index = all_keys.index(key)
SNGT_QHENOMENOLOGY_156: (12)                        for later_key in all_keys[index + 1:]:
SNGT_QHENOMENOLOGY_157: (16)                            self.checkpoint_states.pop(later_key)
```

```
SNGT_QHENOMENOLOGY_158: (8)                     else:
SNGT_QHENOMENOLOGY_159: (12)                         self.checkpoint_states[key] = scene.get_state()
SNGT_QHENOMENOLOGY_160: (4)                 def clear_checkpoints(self):
SNGT_QHENOMENOLOGY_161: (8)                     self.checkpoint_states = dict()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 69 - module_loader.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import builtins
SNGT_QHENOMENOLOGY_3: (0)               import importlib
SNGT_QHENOMENOLOGY_4: (0)               import os
SNGT_QHENOMENOLOGY_5: (0)               import sys
SNGT_QHENOMENOLOGY_6: (0)               import sysconfig
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.logger import log
SNGT_QHENOMENOLOGY_9: (0)               Module = importlib.util.types.ModuleType
SNGT_QHENOMENOLOGY_10: (0)              class ModuleLoader:
SNGT_QHENOMENOLOGY_11: (4)                  """
SNGT_QHENOMENOLOGY_12: (4)                  Utility class to load a module from a file and handle
its imports.
SNGT_QHENOMENOLOGY_13: (4)                  Most parts of this class are only needed for the reload
functionality,
SNGT_QHENOMENOLOGY_14: (4)                  while the `get_module` method is the main entry point
to import a module.
SNGT_QHENOMENOLOGY_15: (4)                  """
SNGT_QHENOMENOLOGY_16: (4)                  @staticmethod
SNGT_QHENOMENOLOGY_17: (4)                  def get_module(file_name: str | None,
is_during_reload=False) -> Module | None:
SNGT_QHENOMENOLOGY_18: (8)                      """
SNGT_QHENOMENOLOGY_19: (8)                      Imports a module from a file and returns it.
SNGT_QHENOMENOLOGY_20: (8)                      During reload (when the user calls `reload()` in
the IPython shell), we
SNGT_QHENOMENOLOGY_21: (8)                      also track the imported modules and reload them as
well (they would be
SNGT_QHENOMENOLOGY_22: (8)                      cached otherwise). See the reload_manager where the
reload parameter is set.
SNGT_QHENOMENOLOGY_23: (8)                      Note that `exec_module()` is called twice when
reloading a module:
SNGT_QHENOMENOLOGY_24: (8)                      1. In exec_module_and_track_imports to track the
imports
SNGT_QHENOMENOLOGY_25: (8)                      2. Here to actually execute the module again with
the respective
SNGT_QHENOMENOLOGY_26: (11)                        imported modules reloaded.
SNGT_QHENOMENOLOGY_27: (8)                      """
SNGT_QHENOMENOLOGY_28: (8)                      if file_name is None:
SNGT_QHENOMENOLOGY_29: (12)                         return None
SNGT_QHENOMENOLOGY_30: (8)                      module_name = file_name.replace(os.sep,
".").replace(".py", "")
SNGT_QHENOMENOLOGY_31: (8)                      spec =
importlib.util.spec_from_file_location(module_name, file_name)
SNGT_QHENOMENOLOGY_32: (8)                      module = importlib.util.module_from_spec(spec)
SNGT_QHENOMENOLOGY_33: (8)                      if is_during_reload:
SNGT_QHENOMENOLOGY_34: (12)                         imported_modules =
ModuleLoader._exec_module_and_track_imports(spec, module)
SNGT_QHENOMENOLOGY_35: (12)                         reloaded_modules_tracker = set()
SNGT_QHENOMENOLOGY_36: (12)                         ModuleLoader._reload_modules(imported_modules,
reloaded_modules_tracker)
SNGT_QHENOMENOLOGY_37: (8)                      spec.loader.exec_module(module)
SNGT_QHENOMENOLOGY_38: (8)                      return module
SNGT_QHENOMENOLOGY_39: (4)                  @staticmethod
SNGT_QHENOMENOLOGY_40: (4)                  def _exec_module_and_track_imports(spec, module:
Module) -> set[str]:
SNGT_QHENOMENOLOGY_41: (8)                      """
SNGT_QHENOMENOLOGY_42: (8)                      Executes the given module (imports it) and returns
all the modules that
SNGT_QHENOMENOLOGY_43: (8)                      are imported during its execution.
SNGT_QHENOMENOLOGY_44: (8)                      This is achieved by replacing the __import__
```

```
function with a custom one
SNGT_QHENOMENOLOGY_45: (8)                    that tracks the imported modules. At the end, the
original __import__
SNGT_QHENOMENOLOGY_46: (8)                    built-in function is restored.
SNGT_QHENOMENOLOGY_47: (8)                    """
SNGT_QHENOMENOLOGY_48: (8)                    imported_modules: set[str] = set()
SNGT_QHENOMENOLOGY_49: (8)                    original_import = builtins.__import__
SNGT_QHENOMENOLOGY_50: (8)                    def tracked_import(name, globals=None, locals=None,
fromlist=(), level=0):
SNGT_QHENOMENOLOGY_51: (12)                       """
SNGT_QHENOMENOLOGY_52: (12)                       Custom __import__ function that does exactly
the same as the original
SNGT_QHENOMENOLOGY_53: (12)                       one, but also tracks the imported modules by
means of adding their
SNGT_QHENOMENOLOGY_54: (12)                       names to a set.
SNGT_QHENOMENOLOGY_55: (12)                       """
SNGT_QHENOMENOLOGY_56: (12)                       result = original_import(name, globals, locals,
fromlist, level)
SNGT_QHENOMENOLOGY_57: (12)                       imported_modules.add(name)
SNGT_QHENOMENOLOGY_58: (12)                       return result
SNGT_QHENOMENOLOGY_59: (8)                    builtins.__import__ = tracked_import
SNGT_QHENOMENOLOGY_60: (8)                    try:
SNGT_QHENOMENOLOGY_61: (12)                       module_name = module.__name__
SNGT_QHENOMENOLOGY_62: (12)                       log.debug('Reloading module "%s"', module_name)
SNGT_QHENOMENOLOGY_63: (12)                       spec.loader.exec_module(module)
SNGT_QHENOMENOLOGY_64: (8)                    finally:
SNGT_QHENOMENOLOGY_65: (12)                       builtins.__import__ = original_import
SNGT_QHENOMENOLOGY_66: (8)                    return imported_modules
SNGT_QHENOMENOLOGY_67: (4)                @staticmethod
SNGT_QHENOMENOLOGY_68: (4)                def _reload_modules(modules: set[str],
reloaded_modules_tracker: set[str]):
SNGT_QHENOMENOLOGY_69: (8)                    """
SNGT_QHENOMENOLOGY_70: (8)                    Out of the given modules, reloads the ones that
were not already imported.
SNGT_QHENOMENOLOGY_71: (8)                    We skip modules that are not user-defined (see
`is_user_defined_module()`).
SNGT_QHENOMENOLOGY_72: (8)                    """
SNGT_QHENOMENOLOGY_73: (8)                    for mod in modules:
SNGT_QHENOMENOLOGY_74: (12)                       if mod in reloaded_modules_tracker:
SNGT_QHENOMENOLOGY_75: (16)                           continue
SNGT_QHENOMENOLOGY_76: (12)                       if not
ModuleLoader._is_user_defined_module(mod):
SNGT_QHENOMENOLOGY_77: (16)                           continue
SNGT_QHENOMENOLOGY_78: (12)                       module = sys.modules[mod]
SNGT_QHENOMENOLOGY_79: (12)                       ModuleLoader._deep_reload(module,
reloaded_modules_tracker)
SNGT_QHENOMENOLOGY_80: (12)                       reloaded_modules_tracker.add(mod)
SNGT_QHENOMENOLOGY_81: (4)                @staticmethod
SNGT_QHENOMENOLOGY_82: (4)                def _is_user_defined_module(mod: str) -> bool:
SNGT_QHENOMENOLOGY_83: (8)                    """
SNGT_QHENOMENOLOGY_84: (8)                    Returns whether the given module is user-defined or
not.
SNGT_QHENOMENOLOGY_85: (8)                    A module is considered user-defined if
SNGT_QHENOMENOLOGY_86: (8)                    - it is not part of the standard library
SNGT_QHENOMENOLOGY_87: (8)                    - AND it is not an external library (site-packages
or dist-packages)
SNGT_QHENOMENOLOGY_88: (8)                    """
SNGT_QHENOMENOLOGY_89: (8)                    if mod not in sys.modules:
SNGT_QHENOMENOLOGY_90: (12)                       return False
SNGT_QHENOMENOLOGY_91: (8)                    if mod in sys.builtin_module_names:
SNGT_QHENOMENOLOGY_92: (12)                       return False
SNGT_QHENOMENOLOGY_93: (8)                    module = sys.modules[mod]
SNGT_QHENOMENOLOGY_94: (8)                    module_path = getattr(module, "__file__", None)
SNGT_QHENOMENOLOGY_95: (8)                    if module_path is None:
SNGT_QHENOMENOLOGY_96: (12)                       return False
SNGT_QHENOMENOLOGY_97: (8)                    module_path = os.path.abspath(module_path)
SNGT_QHENOMENOLOGY_98: (8)                    if "site-packages" in module_path or "dist-
packages" in module_path:
SNGT_QHENOMENOLOGY_99: (12)                       return False
```

```
SNGT_QHENOMENOLOGY_100: (8)                     standard_lib_path = sysconfig.get_path("stdlib")
SNGT_QHENOMENOLOGY_101: (8)                     if module_path.startswith(standard_lib_path):
SNGT_QHENOMENOLOGY_102: (12)                        return False
SNGT_QHENOMENOLOGY_103: (8)                     return True
SNGT_QHENOMENOLOGY_104: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_105: (4)                 def _deep_reload(module: Module,
reloaded_modules_tracker: set[str]):
SNGT_QHENOMENOLOGY_106: (8)                     """
SNGT_QHENOMENOLOGY_107: (8)                     Recursively reloads modules imported by the given
module.
SNGT_QHENOMENOLOGY_108: (8)                     Only user-defined modules are reloaded, see
`is_user_defined_module()`.
SNGT_QHENOMENOLOGY_109: (8)                     """
SNGT_QHENOMENOLOGY_110: (8)                     ignore_manimlib_modules =
manim_config.ignore_manimlib_modules_on_reload
SNGT_QHENOMENOLOGY_111: (8)                     if ignore_manimlib_modules and
module.__name__.startswith("manimlib"):
SNGT_QHENOMENOLOGY_112: (12)                        return
SNGT_QHENOMENOLOGY_113: (8)                     if module.__name__.startswith("manimlib.config"):
SNGT_QHENOMENOLOGY_114: (12)                        return
SNGT_QHENOMENOLOGY_115: (8)                     if not hasattr(module, "__dict__"):
SNGT_QHENOMENOLOGY_116: (12)                        return
SNGT_QHENOMENOLOGY_117: (8)                     if module.__name__ in reloaded_modules_tracker:
SNGT_QHENOMENOLOGY_118: (12)                        return
SNGT_QHENOMENOLOGY_119: (8)                     reloaded_modules_tracker.add(module.__name__)
SNGT_QHENOMENOLOGY_120: (8)                     for _attr_name, attr_value in
module.__dict__.items():
SNGT_QHENOMENOLOGY_121: (12)                        if isinstance(attr_value, Module):
SNGT_QHENOMENOLOGY_122: (16)                            if
ModuleLoader._is_user_defined_module(attr_value.__name__):
SNGT_QHENOMENOLOGY_123: (20)                                ModuleLoader._deep_reload(attr_value,
reloaded_modules_tracker)
SNGT_QHENOMENOLOGY_124: (12)                        elif hasattr(attr_value, "__module__"):
SNGT_QHENOMENOLOGY_125: (16)                            attr_module_name = attr_value.__module__
SNGT_QHENOMENOLOGY_126: (16)                            if
ModuleLoader._is_user_defined_module(attr_module_name):
SNGT_QHENOMENOLOGY_127: (20)                                attr_module =
sys.modules[attr_module_name]
SNGT_QHENOMENOLOGY_128: (20)                                ModuleLoader._deep_reload(attr_module,
reloaded_modules_tracker)
SNGT_QHENOMENOLOGY_129: (8)                     log.debug('Reloading module "%s"', module.__name__)
SNGT_QHENOMENOLOGY_130: (8)                     importlib.reload(module)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 70 - shader_wrapper.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import copy
SNGT_QHENOMENOLOGY_3: (0)               import os
SNGT_QHENOMENOLOGY_4: (0)               import re
SNGT_QHENOMENOLOGY_5: (0)               import OpenGL.GL as gl
SNGT_QHENOMENOLOGY_6: (0)               import moderngl
SNGT_QHENOMENOLOGY_7: (0)               import numpy as np
SNGT_QHENOMENOLOGY_8: (0)               from functools import lru_cache
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.config import parse_cli
SNGT_QHENOMENOLOGY_10: (0)              from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.utils.shaders import
get_shader_code_from_file
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.utils.shaders import get_shader_program
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.shaders import image_path_to_texture
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.shaders import set_program_uniform
SNGT_QHENOMENOLOGY_15: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_16: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_17: (4)                  from typing import Optional, Tuple, Iterable
SNGT_QHENOMENOLOGY_18: (4)                  from manimlib.typing import UniformDict
SNGT_QHENOMENOLOGY_19: (4)                  from moderngl.vertex_array import VertexArray
SNGT_QHENOMENOLOGY_20: (4)                  from moderngl.framebuffer import Framebuffer
SNGT_QHENOMENOLOGY_21: (0)              class ShaderWrapper(object):
```

```
SNGT_QHENOMENOLOGY_22: (4)                 def __init__(
SNGT_QHENOMENOLOGY_23: (8)                     self,
SNGT_QHENOMENOLOGY_24: (8)                     ctx: moderngl.context.Context,
SNGT_QHENOMENOLOGY_25: (8)                     vert_data: np.ndarray,
SNGT_QHENOMENOLOGY_26: (8)                     shader_folder: Optional[str] = None,
SNGT_QHENOMENOLOGY_27: (8)                     mobject_uniforms: Optional[UniformDict] = None,  #
A dictionary mapping names of uniform variables
SNGT_QHENOMENOLOGY_28: (8)                     texture_paths: Optional[dict[str, str]] = None,  #
A dictionary mapping names to filepaths for textures.
SNGT_QHENOMENOLOGY_29: (8)                     depth_test: bool = False,
SNGT_QHENOMENOLOGY_30: (8)                     render_primitive: int = moderngl.TRIANGLE_STRIP,
SNGT_QHENOMENOLOGY_31: (8)                     code_replacements: dict[str, str] = dict(),
SNGT_QHENOMENOLOGY_32: (4)                 ):
SNGT_QHENOMENOLOGY_33: (8)                     self.ctx = ctx
SNGT_QHENOMENOLOGY_34: (8)                     self.vert_data = vert_data
SNGT_QHENOMENOLOGY_35: (8)                     self.vert_attributes = vert_data.dtype.names
SNGT_QHENOMENOLOGY_36: (8)                     self.shader_folder = shader_folder
SNGT_QHENOMENOLOGY_37: (8)                     self.depth_test = depth_test
SNGT_QHENOMENOLOGY_38: (8)                     self.render_primitive = render_primitive
SNGT_QHENOMENOLOGY_39: (8)                     self.texture_paths = texture_paths or dict()
SNGT_QHENOMENOLOGY_40: (8)                     self.program_uniform_mirror: UniformDict = dict()
SNGT_QHENOMENOLOGY_41: (8)                     self.bind_to_mobject_uniforms(mobject_uniforms or
dict())
SNGT_QHENOMENOLOGY_42: (8)                     self.init_program_code()
SNGT_QHENOMENOLOGY_43: (8)                     for old, new in code_replacements.items():
SNGT_QHENOMENOLOGY_44: (12)                        self.replace_code(old, new)
SNGT_QHENOMENOLOGY_45: (8)                     self.init_program()
SNGT_QHENOMENOLOGY_46: (8)                     self.init_textures()
SNGT_QHENOMENOLOGY_47: (8)                     self.init_vertex_objects()
SNGT_QHENOMENOLOGY_48: (8)                     self.refresh_id()
SNGT_QHENOMENOLOGY_49: (4)                 def __deepcopy__(self, memo):
SNGT_QHENOMENOLOGY_50: (8)                     return None
SNGT_QHENOMENOLOGY_51: (4)                 def init_program_code(self) -> None:
SNGT_QHENOMENOLOGY_52: (8)                     def get_code(name: str) -> str | None:
SNGT_QHENOMENOLOGY_53: (12)                        return get_shader_code_from_file(
SNGT_QHENOMENOLOGY_54: (16)                            os.path.join(self.shader_folder, f"
{name}.glsl")
SNGT_QHENOMENOLOGY_55: (12)                        )
SNGT_QHENOMENOLOGY_56: (8)                     self.program_code: dict[str, str | None] = {
SNGT_QHENOMENOLOGY_57: (12)                        "vertex_shader": get_code("vert"),
SNGT_QHENOMENOLOGY_58: (12)                        "geometry_shader": get_code("geom"),
SNGT_QHENOMENOLOGY_59: (12)                        "fragment_shader": get_code("frag"),
SNGT_QHENOMENOLOGY_60: (8)                     }
SNGT_QHENOMENOLOGY_61: (4)                 def init_program(self):
SNGT_QHENOMENOLOGY_62: (8)                     if not self.shader_folder:
SNGT_QHENOMENOLOGY_63: (12)                        self.program = None
SNGT_QHENOMENOLOGY_64: (12)                        self.vert_format = None
SNGT_QHENOMENOLOGY_65: (12)                        self.programs = []
SNGT_QHENOMENOLOGY_66: (12)                        return
SNGT_QHENOMENOLOGY_67: (8)                     self.program = get_shader_program(self.ctx,
**self.program_code)
SNGT_QHENOMENOLOGY_68: (8)                     self.vert_format =
moderngl.detect_format(self.program, self.vert_attributes)
SNGT_QHENOMENOLOGY_69: (8)                     self.programs = [self.program]
SNGT_QHENOMENOLOGY_70: (4)                 def init_textures(self):
SNGT_QHENOMENOLOGY_71: (8)                     self.texture_names_to_ids = dict()
SNGT_QHENOMENOLOGY_72: (8)                     self.textures = []
SNGT_QHENOMENOLOGY_73: (8)                     for name, path in self.texture_paths.items():
SNGT_QHENOMENOLOGY_74: (12)                        self.add_texture(name,
image_path_to_texture(path, self.ctx))
SNGT_QHENOMENOLOGY_75: (4)                 def init_vertex_objects(self):
SNGT_QHENOMENOLOGY_76: (8)                     self.vbo = None
SNGT_QHENOMENOLOGY_77: (8)                     self.vaos = []
SNGT_QHENOMENOLOGY_78: (4)                 def add_texture(self, name: str, texture:
moderngl.Texture):
SNGT_QHENOMENOLOGY_79: (8)                     max_units =
self.ctx.info['GL_MAX_TEXTURE_IMAGE_UNITS']
SNGT_QHENOMENOLOGY_80: (8)                     if len(self.textures) >= max_units:
SNGT_QHENOMENOLOGY_81: (12)                        raise ValueError(f"Unable to use more than
```

```
{max_units} textures for a program")
SNGT_QHENOMENOLOGY_82: (8)                      self.texture_names_to_ids[name] =
len(self.textures)
SNGT_QHENOMENOLOGY_83: (8)                      self.textures.append(texture)
SNGT_QHENOMENOLOGY_84: (4)                  def bind_to_mobject_uniforms(self, mobject_uniforms:
UniformDict):
SNGT_QHENOMENOLOGY_85: (8)                      self.mobject_uniforms = mobject_uniforms
SNGT_QHENOMENOLOGY_86: (4)                  def get_id(self) -> int:
SNGT_QHENOMENOLOGY_87: (8)                      return self.id
SNGT_QHENOMENOLOGY_88: (4)                  def refresh_id(self) -> None:
SNGT_QHENOMENOLOGY_89: (8)                      self.id = hash("".join(map(str, [
SNGT_QHENOMENOLOGY_90: (12)                         "".join(map(str, self.program_code.values())),
SNGT_QHENOMENOLOGY_91: (12)                         self.mobject_uniforms,
SNGT_QHENOMENOLOGY_92: (12)                         self.depth_test,
SNGT_QHENOMENOLOGY_93: (12)                         self.render_primitive,
SNGT_QHENOMENOLOGY_94: (12)                         self.texture_paths,
SNGT_QHENOMENOLOGY_95: (8)                      ])))
SNGT_QHENOMENOLOGY_96: (4)                  def replace_code(self, old: str, new: str) -> None:
SNGT_QHENOMENOLOGY_97: (8)                      code_map = self.program_code
SNGT_QHENOMENOLOGY_98: (8)                      for name in code_map:
SNGT_QHENOMENOLOGY_99: (12)                         if code_map[name] is None:
SNGT_QHENOMENOLOGY_100: (16)                            continue
SNGT_QHENOMENOLOGY_101: (12)                        code_map[name] = re.sub(old, new,
code_map[name])
SNGT_QHENOMENOLOGY_102: (8)                     self.init_program()
SNGT_QHENOMENOLOGY_103: (8)                     self.refresh_id()
SNGT_QHENOMENOLOGY_104: (4)                 def use_clip_plane(self):
SNGT_QHENOMENOLOGY_105: (8)                     if "clip_plane" not in self.mobject_uniforms:
SNGT_QHENOMENOLOGY_106: (12)                        return False
SNGT_QHENOMENOLOGY_107: (8)                     return any(self.mobject_uniforms["clip_plane"])
SNGT_QHENOMENOLOGY_108: (4)                 def set_ctx_depth_test(self, enable: bool = True) ->
None:
SNGT_QHENOMENOLOGY_109: (8)                     if enable:
SNGT_QHENOMENOLOGY_110: (12)                        self.ctx.enable(moderngl.DEPTH_TEST)
SNGT_QHENOMENOLOGY_111: (8)                     else:
SNGT_QHENOMENOLOGY_112: (12)                        self.ctx.disable(moderngl.DEPTH_TEST)
SNGT_QHENOMENOLOGY_113: (4)                 def set_ctx_clip_plane(self, enable: bool = True) ->
None:
SNGT_QHENOMENOLOGY_114: (8)                     if enable:
SNGT_QHENOMENOLOGY_115: (12)                        gl.glEnable(gl.GL_CLIP_DISTANCE0)
SNGT_QHENOMENOLOGY_116: (4)                 def read_in(self, data_list: Iterable[np.ndarray]):
SNGT_QHENOMENOLOGY_117: (8)                     total_len = sum(map(len, data_list))
SNGT_QHENOMENOLOGY_118: (8)                     if total_len == 0:
SNGT_QHENOMENOLOGY_119: (12)                        if self.vbo is not None:
SNGT_QHENOMENOLOGY_120: (16)                            self.vbo.clear()
SNGT_QHENOMENOLOGY_121: (12)                        return
SNGT_QHENOMENOLOGY_122: (8)                     if len(self.vert_data) != total_len:
SNGT_QHENOMENOLOGY_123: (12)                        self.vert_data = np.concatenate(data_list)
SNGT_QHENOMENOLOGY_124: (8)                     else:
SNGT_QHENOMENOLOGY_125: (12)                        np.concatenate(data_list, out=self.vert_data)
SNGT_QHENOMENOLOGY_126: (8)                     total_size = self.vert_data.itemsize * total_len
SNGT_QHENOMENOLOGY_127: (8)                     if self.vbo is not None and self.vbo.size !=
total_size:
SNGT_QHENOMENOLOGY_128: (12)                        self.release()  # This sets vbo to be None
SNGT_QHENOMENOLOGY_129: (8)                     if self.vbo is None:
SNGT_QHENOMENOLOGY_130: (12)                        self.vbo = self.ctx.buffer(self.vert_data)
SNGT_QHENOMENOLOGY_131: (12)                        self.generate_vaos()
SNGT_QHENOMENOLOGY_132: (8)                     else:
SNGT_QHENOMENOLOGY_133: (12)                        self.vbo.write(self.vert_data)
SNGT_QHENOMENOLOGY_134: (4)                 def generate_vaos(self):
SNGT_QHENOMENOLOGY_135: (8)                     self.vaos = [
SNGT_QHENOMENOLOGY_136: (12)                        self.ctx.vertex_array(
SNGT_QHENOMENOLOGY_137: (16)                            program=program,
SNGT_QHENOMENOLOGY_138: (16)                            content=[(self.vbo, self.vert_format,
*self.vert_attributes)],
SNGT_QHENOMENOLOGY_139: (16)                            mode=self.render_primitive,
SNGT_QHENOMENOLOGY_140: (12)                        )
SNGT_QHENOMENOLOGY_141: (12)                        for program in self.programs
SNGT_QHENOMENOLOGY_142: (8)                     ]
```

```
SNGT_QHENOMENOLOGY_143: (4)                def pre_render(self):
SNGT_QHENOMENOLOGY_144: (8)                    self.set_ctx_depth_test(self.depth_test)
SNGT_QHENOMENOLOGY_145: (8)                    self.set_ctx_clip_plane(self.use_clip_plane())
SNGT_QHENOMENOLOGY_146: (8)                    for tid, texture in enumerate(self.textures):
SNGT_QHENOMENOLOGY_147: (12)                       texture.use(tid)
SNGT_QHENOMENOLOGY_148: (4)                def render(self):
SNGT_QHENOMENOLOGY_149: (8)                    for vao in self.vaos:
SNGT_QHENOMENOLOGY_150: (12)                       vao.render()
SNGT_QHENOMENOLOGY_151: (4)                def update_program_uniforms(self, camera_uniforms:
UniformDict):
SNGT_QHENOMENOLOGY_152: (8)                    for program in self.programs:
SNGT_QHENOMENOLOGY_153: (12)                       if program is None:
SNGT_QHENOMENOLOGY_154: (16)                           continue
SNGT_QHENOMENOLOGY_155: (12)                       for uniforms in [self.mobject_uniforms,
camera_uniforms, self.texture_names_to_ids]:
SNGT_QHENOMENOLOGY_156: (16)                           for name, value in uniforms.items():
SNGT_QHENOMENOLOGY_157: (20)                               set_program_uniform(program, name,
value)
SNGT_QHENOMENOLOGY_158: (4)                def release(self):
SNGT_QHENOMENOLOGY_159: (8)                    for obj in (self.vbo, *self.vaos):
SNGT_QHENOMENOLOGY_160: (12)                       if obj is not None:
SNGT_QHENOMENOLOGY_161: (16)                           obj.release()
SNGT_QHENOMENOLOGY_162: (8)                    self.init_vertex_objects()
SNGT_QHENOMENOLOGY_163: (4)                def release_textures(self):
SNGT_QHENOMENOLOGY_164: (8)                    for texture in self.textures:
SNGT_QHENOMENOLOGY_165: (12)                       texture.release()
SNGT_QHENOMENOLOGY_166: (12)                       del texture
SNGT_QHENOMENOLOGY_167: (8)                    self.textures = []
SNGT_QHENOMENOLOGY_168: (8)                    self.texture_names_to_ids = dict()
SNGT_QHENOMENOLOGY_169: (0)            class VShaderWrapper(ShaderWrapper):
SNGT_QHENOMENOLOGY_170: (4)                def __init__(
SNGT_QHENOMENOLOGY_171: (8)                    self,
SNGT_QHENOMENOLOGY_172: (8)                    ctx: moderngl.context.Context,
SNGT_QHENOMENOLOGY_173: (8)                    vert_data: np.ndarray,
SNGT_QHENOMENOLOGY_174: (8)                    shader_folder: Optional[str] = None,
SNGT_QHENOMENOLOGY_175: (8)                    mobject_uniforms: Optional[UniformDict] = None,  #
A dictionary mapping names of uniform variables
SNGT_QHENOMENOLOGY_176: (8)                    texture_paths: Optional[dict[str, str]] = None,  #
A dictionary mapping names to filepaths for textures.
SNGT_QHENOMENOLOGY_177: (8)                    depth_test: bool = False,
SNGT_QHENOMENOLOGY_178: (8)                    render_primitive: int = moderngl.TRIANGLES,
SNGT_QHENOMENOLOGY_179: (8)                    code_replacements: dict[str, str] = dict(),
SNGT_QHENOMENOLOGY_180: (8)                    stroke_behind: bool = False,
SNGT_QHENOMENOLOGY_181: (4)                ):
SNGT_QHENOMENOLOGY_182: (8)                    self.stroke_behind = stroke_behind
SNGT_QHENOMENOLOGY_183: (8)                    super().__init__(
SNGT_QHENOMENOLOGY_184: (12)                       ctx=ctx,
SNGT_QHENOMENOLOGY_185: (12)                       vert_data=vert_data,
SNGT_QHENOMENOLOGY_186: (12)                       shader_folder=shader_folder,
SNGT_QHENOMENOLOGY_187: (12)                       mobject_uniforms=mobject_uniforms,
SNGT_QHENOMENOLOGY_188: (12)                       texture_paths=texture_paths,
SNGT_QHENOMENOLOGY_189: (12)                       depth_test=depth_test,
SNGT_QHENOMENOLOGY_190: (12)                       render_primitive=render_primitive,
SNGT_QHENOMENOLOGY_191: (12)                       code_replacements=code_replacements,
SNGT_QHENOMENOLOGY_192: (8)                    )
SNGT_QHENOMENOLOGY_193: (8)                    self.fill_canvas =
VShaderWrapper.get_fill_canvas(self.ctx)
SNGT_QHENOMENOLOGY_194: (8)                    self.add_texture('Texture',
self.fill_canvas[0].color_attachments[0])
SNGT_QHENOMENOLOGY_195: (8)                    self.add_texture('DepthTexture',
self.fill_canvas[2].color_attachments[0])
SNGT_QHENOMENOLOGY_196: (4)                def init_program_code(self) -> None:
SNGT_QHENOMENOLOGY_197: (8)                    self.program_code = {
SNGT_QHENOMENOLOGY_198: (12)                       f"{vtype}_{name}": get_shader_code_from_file(
SNGT_QHENOMENOLOGY_199: (16)                           os.path.join("quadratic_bezier", f"
{vtype}", f"{name}.glsl")
SNGT_QHENOMENOLOGY_200: (12)                       )
SNGT_QHENOMENOLOGY_201: (12)                       for vtype in ["stroke", "fill", "depth"]
SNGT_QHENOMENOLOGY_202: (12)                       for name in ["vert", "geom", "frag"]
```

```
SNGT_QHENOMENOLOGY_203: (8)                     }
SNGT_QHENOMENOLOGY_204: (4)                 def init_program(self):
SNGT_QHENOMENOLOGY_205: (8)                     self.stroke_program = get_shader_program(
SNGT_QHENOMENOLOGY_206: (12)                        self.ctx,
SNGT_QHENOMENOLOGY_207: (12)                        vertex_shader=self.program_code["stroke_vert"],
SNGT_QHENOMENOLOGY_208: (12)
geometry_shader=self.program_code["stroke_geom"],
SNGT_QHENOMENOLOGY_209: (12)
fragment_shader=self.program_code["stroke_frag"],
SNGT_QHENOMENOLOGY_210: (8)                     )
SNGT_QHENOMENOLOGY_211: (8)                     self.fill_program = get_shader_program(
SNGT_QHENOMENOLOGY_212: (12)                        self.ctx,
SNGT_QHENOMENOLOGY_213: (12)                        vertex_shader=self.program_code["fill_vert"],
SNGT_QHENOMENOLOGY_214: (12)                        geometry_shader=self.program_code["fill_geom"],
SNGT_QHENOMENOLOGY_215: (12)                        fragment_shader=self.program_code["fill_frag"],
SNGT_QHENOMENOLOGY_216: (8)                     )
SNGT_QHENOMENOLOGY_217: (8)                     self.fill_border_program = get_shader_program(
SNGT_QHENOMENOLOGY_218: (12)                        self.ctx,
SNGT_QHENOMENOLOGY_219: (12)                        vertex_shader=self.program_code["stroke_vert"],
SNGT_QHENOMENOLOGY_220: (12)
geometry_shader=self.program_code["stroke_geom"],
SNGT_QHENOMENOLOGY_221: (12)
fragment_shader=self.program_code["stroke_frag"].replace(
SNGT_QHENOMENOLOGY_222: (16)                            "// MODIFY FRAG COLOR",
SNGT_QHENOMENOLOGY_223: (16)                            "frag_color.a *= 0.95; frag_color.rgb *=
frag_color.a;",
SNGT_QHENOMENOLOGY_224: (12)                        )
SNGT_QHENOMENOLOGY_225: (8)                     )
SNGT_QHENOMENOLOGY_226: (8)                     self.fill_depth_program = get_shader_program(
SNGT_QHENOMENOLOGY_227: (12)                        self.ctx,
SNGT_QHENOMENOLOGY_228: (12)                        vertex_shader=self.program_code["depth_vert"],
SNGT_QHENOMENOLOGY_229: (12)
geometry_shader=self.program_code["depth_geom"],
SNGT_QHENOMENOLOGY_230: (12)
fragment_shader=self.program_code["depth_frag"],
SNGT_QHENOMENOLOGY_231: (8)                     )
SNGT_QHENOMENOLOGY_232: (8)                     self.programs = [self.stroke_program,
self.fill_program, self.fill_border_program, self.fill_depth_program]
SNGT_QHENOMENOLOGY_233: (8)                     self.stroke_vert_format = '3f 4f 1f 1f 16x 3f 4x'
SNGT_QHENOMENOLOGY_234: (8)                     self.stroke_vert_attributes = ['point',
'stroke_rgba', 'stroke_width', 'joint_angle', 'unit_normal']
SNGT_QHENOMENOLOGY_235: (8)                     self.fill_vert_format = '3f 24x 4f 3f 4x'
SNGT_QHENOMENOLOGY_236: (8)                     self.fill_vert_attributes = ['point', 'fill_rgba',
'base_normal']
SNGT_QHENOMENOLOGY_237: (8)                     self.fill_border_vert_format = '3f 20x 1f 4f 3f 1f'
SNGT_QHENOMENOLOGY_238: (8)                     self.fill_border_vert_attributes = ['point',
'joint_angle', 'stroke_rgba', 'unit_normal', 'stroke_width']
SNGT_QHENOMENOLOGY_239: (8)                     self.fill_depth_vert_format = '3f 40x 3f 4x'
SNGT_QHENOMENOLOGY_240: (8)                     self.fill_depth_vert_attributes = ['point',
'base_normal']
SNGT_QHENOMENOLOGY_241: (4)                 def init_vertex_objects(self):
SNGT_QHENOMENOLOGY_242: (8)                     self.vbo = None
SNGT_QHENOMENOLOGY_243: (8)                     self.stroke_vao = None
SNGT_QHENOMENOLOGY_244: (8)                     self.fill_vao = None
SNGT_QHENOMENOLOGY_245: (8)                     self.fill_border_vao = None
SNGT_QHENOMENOLOGY_246: (8)                     self.vaos = []
SNGT_QHENOMENOLOGY_247: (4)                 def generate_vaos(self):
SNGT_QHENOMENOLOGY_248: (8)                     self.stroke_vao = self.ctx.vertex_array(
SNGT_QHENOMENOLOGY_249: (12)                        program=self.stroke_program,
SNGT_QHENOMENOLOGY_250: (12)                        content=[(self.vbo, self.stroke_vert_format,
*self.stroke_vert_attributes)],
SNGT_QHENOMENOLOGY_251: (12)                        mode=self.render_primitive,
SNGT_QHENOMENOLOGY_252: (8)                     )
SNGT_QHENOMENOLOGY_253: (8)                     self.fill_vao = self.ctx.vertex_array(
SNGT_QHENOMENOLOGY_254: (12)                        program=self.fill_program,
SNGT_QHENOMENOLOGY_255: (12)                        content=[(self.vbo, self.fill_vert_format,
*self.fill_vert_attributes)],
SNGT_QHENOMENOLOGY_256: (12)                        mode=self.render_primitive,
SNGT_QHENOMENOLOGY_257: (8)                     )
```

```
SNGT_QHENOMENOLOGY_258: (8)                     self.fill_border_vao = self.ctx.vertex_array(
SNGT_QHENOMENOLOGY_259: (12)                         program=self.fill_border_program,
SNGT_QHENOMENOLOGY_260: (12)                         content=[(self.vbo,
self.fill_border_vert_format, *self.fill_border_vert_attributes)],
SNGT_QHENOMENOLOGY_261: (12)                         mode=self.render_primitive,
SNGT_QHENOMENOLOGY_262: (8)                     )
SNGT_QHENOMENOLOGY_263: (8)                     self.fill_depth_vao = self.ctx.vertex_array(
SNGT_QHENOMENOLOGY_264: (12)                         program=self.fill_depth_program,
SNGT_QHENOMENOLOGY_265: (12)                         content=[(self.vbo,
self.fill_depth_vert_format, *self.fill_depth_vert_attributes)],
SNGT_QHENOMENOLOGY_266: (12)                         mode=self.render_primitive,
SNGT_QHENOMENOLOGY_267: (8)                     )
SNGT_QHENOMENOLOGY_268: (8)                     self.vaos = [self.stroke_vao, self.fill_vao,
self.fill_border_vao, self.fill_depth_vao]
SNGT_QHENOMENOLOGY_269: (4)                 def set_backstroke(self, value: bool = True):
SNGT_QHENOMENOLOGY_270: (8)                     self.stroke_behind = value
SNGT_QHENOMENOLOGY_271: (4)                 def refresh_id(self):
SNGT_QHENOMENOLOGY_272: (8)                     super().refresh_id()
SNGT_QHENOMENOLOGY_273: (8)                     self.id = hash(str(self.id) +
str(self.stroke_behind))
SNGT_QHENOMENOLOGY_274: (4)                 def render_stroke(self):
SNGT_QHENOMENOLOGY_275: (8)                     if self.stroke_vao is None:
SNGT_QHENOMENOLOGY_276: (12)                         return
SNGT_QHENOMENOLOGY_277: (8)                     self.stroke_vao.render()
SNGT_QHENOMENOLOGY_278: (4)                 def render_fill(self):
SNGT_QHENOMENOLOGY_279: (8)                     if self.fill_vao is None:
SNGT_QHENOMENOLOGY_280: (12)                         return
SNGT_QHENOMENOLOGY_281: (8)                     original_fbo = self.ctx.fbo
SNGT_QHENOMENOLOGY_282: (8)                     fill_tx_fbo, fill_tx_vao, depth_tx_fbo =
self.fill_canvas
SNGT_QHENOMENOLOGY_283: (8)                     fill_tx_fbo.clear()
SNGT_QHENOMENOLOGY_284: (8)                     fill_tx_fbo.use()
SNGT_QHENOMENOLOGY_285: (8)                     apply_depth_test =
bool(gl.glGetBooleanv(gl.GL_DEPTH_TEST))
SNGT_QHENOMENOLOGY_286: (8)                     self.ctx.disable(moderngl.DEPTH_TEST)
SNGT_QHENOMENOLOGY_287: (8)                     gl.glBlendFuncSeparate(
SNGT_QHENOMENOLOGY_288: (12)                         gl.GL_SRC_ALPHA, gl.GL_ONE_MINUS_SRC_ALPHA,
SNGT_QHENOMENOLOGY_289: (12)                         gl.GL_ONE_MINUS_DST_ALPHA, gl.GL_ONE
SNGT_QHENOMENOLOGY_290: (8)                     )
SNGT_QHENOMENOLOGY_291: (8)                     self.fill_vao.render()
SNGT_QHENOMENOLOGY_292: (8)                     if apply_depth_test:
SNGT_QHENOMENOLOGY_293: (12)                         self.ctx.enable(moderngl.DEPTH_TEST)
SNGT_QHENOMENOLOGY_294: (12)                         depth_tx_fbo.clear(1.0)
SNGT_QHENOMENOLOGY_295: (12)                         depth_tx_fbo.use()
SNGT_QHENOMENOLOGY_296: (12)                         gl.glBlendFunc(gl.GL_ONE, gl.GL_ONE)
SNGT_QHENOMENOLOGY_297: (12)                         gl.glBlendEquation(gl.GL_MIN)
SNGT_QHENOMENOLOGY_298: (12)                         self.fill_depth_vao.render()
SNGT_QHENOMENOLOGY_299: (8)                     gl.glBlendFunc(gl.GL_ONE, gl.GL_ONE)
SNGT_QHENOMENOLOGY_300: (8)                     gl.glBlendEquation(gl.GL_MAX)
SNGT_QHENOMENOLOGY_301: (8)                     self.fill_border_vao.render()
SNGT_QHENOMENOLOGY_302: (8)                     original_fbo.use()
SNGT_QHENOMENOLOGY_303: (8)                     gl.glBlendFunc(gl.GL_ONE,
gl.GL_ONE_MINUS_SRC_ALPHA)
SNGT_QHENOMENOLOGY_304: (8)                     gl.glBlendEquation(gl.GL_FUNC_ADD)
SNGT_QHENOMENOLOGY_305: (8)                     fill_tx_vao.render()
SNGT_QHENOMENOLOGY_306: (8)                     gl.glBlendFunc(gl.GL_SRC_ALPHA,
gl.GL_ONE_MINUS_SRC_ALPHA)
SNGT_QHENOMENOLOGY_307: (4)                 @lru_cache
SNGT_QHENOMENOLOGY_308: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_309: (4)                 def get_fill_canvas(ctx: moderngl.Context) ->
Tuple[Framebuffer, VertexArray, Framebuffer]:
SNGT_QHENOMENOLOGY_310: (8)                     """
SNGT_QHENOMENOLOGY_311: (8)                     Because VMobjects with fill are rendered in a funny
way, using
SNGT_QHENOMENOLOGY_312: (8)                     alpha blending to effectively compute the winding
number around
SNGT_QHENOMENOLOGY_313: (8)                     each pixel, they need to be rendered to a separate
texture, which
SNGT_QHENOMENOLOGY_314: (8)                     is then composited onto the ordinary frame buffer.
```

```
SNGT_QHENOMENOLOGY_315: (8)                     This returns a texture, loaded into a frame buffer,
and a vao
SNGT_QHENOMENOLOGY_316: (8)                     which can display that texture as a simple quad
onto a screen,
SNGT_QHENOMENOLOGY_317: (8)                     along with the rgb value which is meant to be
discarded.
SNGT_QHENOMENOLOGY_318: (8)                     """
SNGT_QHENOMENOLOGY_319: (8)                     size = manim_config.camera.resolution
SNGT_QHENOMENOLOGY_320: (8)                     double_size = (2 * size[0], 2 * size[1])
SNGT_QHENOMENOLOGY_321: (8)                     fill_texture = ctx.texture(size=double_size,
components=4, dtype='f2')
SNGT_QHENOMENOLOGY_322: (8)                     depth_texture = ctx.texture(size=size,
components=1, dtype='f4')
SNGT_QHENOMENOLOGY_323: (8)                     fill_texture_fbo = ctx.framebuffer(fill_texture)
SNGT_QHENOMENOLOGY_324: (8)                     depth_texture_fbo = ctx.framebuffer(depth_texture)
SNGT_QHENOMENOLOGY_325: (8)                     simple_vert = '''
SNGT_QHENOMENOLOGY_326: (12)                        in vec2 texcoord;
SNGT_QHENOMENOLOGY_327: (12)                        out vec2 uv;
SNGT_QHENOMENOLOGY_328: (12)                        void main() {
SNGT_QHENOMENOLOGY_329: (16)                            gl_Position = vec4((2.0 * texcoord - 1.0),
0.0, 1.0);
SNGT_QHENOMENOLOGY_330: (16)                            uv = texcoord;
SNGT_QHENOMENOLOGY_331: (12)                        }
SNGT_QHENOMENOLOGY_332: (8)                     '''
SNGT_QHENOMENOLOGY_333: (8)                     alpha_adjust_frag = '''
SNGT_QHENOMENOLOGY_334: (12)                        uniform sampler2D Texture;
SNGT_QHENOMENOLOGY_335: (12)                        uniform sampler2D DepthTexture;
SNGT_QHENOMENOLOGY_336: (12)                        in vec2 uv;
SNGT_QHENOMENOLOGY_337: (12)                        out vec4 color;
SNGT_QHENOMENOLOGY_338: (12)                        void main() {
SNGT_QHENOMENOLOGY_339: (16)                            color = texture(Texture, uv);
SNGT_QHENOMENOLOGY_340: (16)                            if(color.a == 0) discard;
SNGT_QHENOMENOLOGY_341: (16)                            if(color.a < 0){
SNGT_QHENOMENOLOGY_342: (20)                                color.a = -color.a / (1.0 - color.a);
SNGT_QHENOMENOLOGY_343: (20)                                color.rgb *= (color.a - 1);
SNGT_QHENOMENOLOGY_344: (16)                            }
SNGT_QHENOMENOLOGY_345: (16)                            // Counteract scaling in fill frag
SNGT_QHENOMENOLOGY_346: (16)                            color *= 1.06;
SNGT_QHENOMENOLOGY_347: (16)                            gl_FragDepth = texture(DepthTexture, uv)
[0];
SNGT_QHENOMENOLOGY_348: (12)                        }
SNGT_QHENOMENOLOGY_349: (8)                     '''
SNGT_QHENOMENOLOGY_350: (8)                     fill_program = ctx.program(
SNGT_QHENOMENOLOGY_351: (12)                        vertex_shader=simple_vert,
SNGT_QHENOMENOLOGY_352: (12)                        fragment_shader=alpha_adjust_frag,
SNGT_QHENOMENOLOGY_353: (8)                     )
SNGT_QHENOMENOLOGY_354: (8)                     verts = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
SNGT_QHENOMENOLOGY_355: (8)                     simple_vbo =
ctx.buffer(verts.astype('f4').tobytes())
SNGT_QHENOMENOLOGY_356: (8)                     fill_texture_vao = ctx.simple_vertex_array(
SNGT_QHENOMENOLOGY_357: (12)                        fill_program, simple_vbo, 'texcoord',
SNGT_QHENOMENOLOGY_358: (12)                        mode=moderngl.TRIANGLE_STRIP
SNGT_QHENOMENOLOGY_359: (8)                     )
SNGT_QHENOMENOLOGY_360: (8)                     return (fill_texture_fbo, fill_texture_vao,
depth_texture_fbo)
SNGT_QHENOMENOLOGY_361: (4)                 def render(self):
SNGT_QHENOMENOLOGY_362: (8)                     if self.stroke_behind:
SNGT_QHENOMENOLOGY_363: (12)                        self.render_stroke()
SNGT_QHENOMENOLOGY_364: (12)                        self.render_fill()
SNGT_QHENOMENOLOGY_365: (8)                     else:
SNGT_QHENOMENOLOGY_366: (12)                        self.render_fill()
SNGT_QHENOMENOLOGY_367: (12)                        self.render_stroke()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 71 - interactive_scene.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import itertools as it
```

```
SNGT_QHENOMENOLOGY_3: (0)              import numpy as np
SNGT_QHENOMENOLOGY_4: (0)              import pyperclip
SNGT_QHENOMENOLOGY_5: (0)              from IPython.core.getipython import get_ipython
SNGT_QHENOMENOLOGY_6: (0)              from pyglet.window import key as PygletWindowKeys
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.animation.fading import FadeIn
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.constants import DL, DOWN, DR, LEFT, ORIGIN,
RIGHT, UL, UP, UR
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.constants import FRAME_WIDTH, FRAME_HEIGHT,
SMALL_BUFF
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.constants import PI
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.constants import DEG
SNGT_QHENOMENOLOGY_13: (0)             from manimlib.constants import MANIM_COLORS, WHITE, GREY_A,
GREY_C
SNGT_QHENOMENOLOGY_14: (0)             from manimlib.mobject.geometry import Line
SNGT_QHENOMENOLOGY_15: (0)             from manimlib.mobject.geometry import Rectangle
SNGT_QHENOMENOLOGY_16: (0)             from manimlib.mobject.geometry import Square
SNGT_QHENOMENOLOGY_17: (0)             from manimlib.mobject.mobject import Group
SNGT_QHENOMENOLOGY_18: (0)             from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_19: (0)             from manimlib.mobject.numbers import DecimalNumber
SNGT_QHENOMENOLOGY_20: (0)             from manimlib.mobject.svg.tex_mobject import Tex
SNGT_QHENOMENOLOGY_21: (0)             from manimlib.mobject.svg.text_mobject import Text
SNGT_QHENOMENOLOGY_22: (0)             from manimlib.mobject.types.dot_cloud import DotCloud
SNGT_QHENOMENOLOGY_23: (0)             from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_24: (0)             from manimlib.mobject.types.vectorized_mobject import
VHighlight
SNGT_QHENOMENOLOGY_25: (0)             from manimlib.mobject.types.vectorized_mobject import
VMobject
SNGT_QHENOMENOLOGY_26: (0)             from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_27: (0)             from manimlib.scene.scene import SceneState
SNGT_QHENOMENOLOGY_28: (0)             from manimlib.utils.family_ops import
extract_mobject_family_members
SNGT_QHENOMENOLOGY_29: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_30: (0)             from manimlib.utils.tex_file_writing import LatexError
SNGT_QHENOMENOLOGY_31: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_32: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_33: (4)                 from manimlib.typing import Vect3
SNGT_QHENOMENOLOGY_34: (0)             SELECT_KEY = manim_config.key_bindings.select
SNGT_QHENOMENOLOGY_35: (0)             UNSELECT_KEY = manim_config.key_bindings.unselect
SNGT_QHENOMENOLOGY_36: (0)             GRAB_KEY = manim_config.key_bindings.grab
SNGT_QHENOMENOLOGY_37: (0)             X_GRAB_KEY = manim_config.key_bindings.x_grab
SNGT_QHENOMENOLOGY_38: (0)             Y_GRAB_KEY = manim_config.key_bindings.y_grab
SNGT_QHENOMENOLOGY_39: (0)             GRAB_KEYS = [GRAB_KEY, X_GRAB_KEY, Y_GRAB_KEY]
SNGT_QHENOMENOLOGY_40: (0)             RESIZE_KEY = manim_config.key_bindings.resize  # TODO
SNGT_QHENOMENOLOGY_41: (0)             COLOR_KEY = manim_config.key_bindings.color
SNGT_QHENOMENOLOGY_42: (0)             INFORMATION_KEY = manim_config.key_bindings.information
SNGT_QHENOMENOLOGY_43: (0)             CURSOR_KEY = manim_config.key_bindings.cursor
SNGT_QHENOMENOLOGY_44: (0)             ARROW_SYMBOLS: list[int] = [
SNGT_QHENOMENOLOGY_45: (4)                 PygletWindowKeys.LEFT,
SNGT_QHENOMENOLOGY_46: (4)                 PygletWindowKeys.UP,
SNGT_QHENOMENOLOGY_47: (4)                 PygletWindowKeys.RIGHT,
SNGT_QHENOMENOLOGY_48: (4)                 PygletWindowKeys.DOWN,
SNGT_QHENOMENOLOGY_49: (0)             ]
SNGT_QHENOMENOLOGY_50: (0)             ALL_MODIFIERS = PygletWindowKeys.MOD_CTRL |
PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_SHIFT
SNGT_QHENOMENOLOGY_51: (0)             class InteractiveScene(Scene):
SNGT_QHENOMENOLOGY_52: (4)                 """
SNGT_QHENOMENOLOGY_53: (4)                 To select mobjects on screen, hold ctrl and move the
mouse to highlight a region,
SNGT_QHENOMENOLOGY_54: (4)                 or just tap ctrl to select the mobject under the
cursor.
SNGT_QHENOMENOLOGY_55: (4)                 Pressing command + t will toggle between modes where
you either select top level
SNGT_QHENOMENOLOGY_56: (4)                 mobjects part of the scene, or low level pieces.
SNGT_QHENOMENOLOGY_57: (4)                 Hold 'g' to grab the selection and move it around
SNGT_QHENOMENOLOGY_58: (4)                 Hold 'h' to drag it constrained in the horizontal
direction
SNGT_QHENOMENOLOGY_59: (4)                 Hold 'v' to drag it constrained in the vertical
```

```
direction
SNGT_QHENOMENOLOGY_60: (4)                Hold 't' to resize selection, adding 'shift' to resize
with respect to a corner
SNGT_QHENOMENOLOGY_61: (4)                Command + 'c' copies the ids of selections to clipboard
SNGT_QHENOMENOLOGY_62: (4)                Command + 'v' will paste either:
SNGT_QHENOMENOLOGY_63: (8)                    - The copied mobject
SNGT_QHENOMENOLOGY_64: (8)                    - A Tex mobject based on copied LaTeX
SNGT_QHENOMENOLOGY_65: (8)                    - A Text mobject based on copied Text
SNGT_QHENOMENOLOGY_66: (4)                Command + 'z' restores selection back to its original
state
SNGT_QHENOMENOLOGY_67: (4)                Command + 's' saves the selected mobjects to file
SNGT_QHENOMENOLOGY_68: (4)                """
SNGT_QHENOMENOLOGY_69: (4)                corner_dot_config = dict(
SNGT_QHENOMENOLOGY_70: (8)                    color=WHITE,
SNGT_QHENOMENOLOGY_71: (8)                    radius=0.05,
SNGT_QHENOMENOLOGY_72: (8)                    glow_factor=2.0,
SNGT_QHENOMENOLOGY_73: (4)                )
SNGT_QHENOMENOLOGY_74: (4)                selection_rectangle_stroke_color = WHITE
SNGT_QHENOMENOLOGY_75: (4)                selection_rectangle_stroke_width = 1.0
SNGT_QHENOMENOLOGY_76: (4)                palette_colors = MANIM_COLORS
SNGT_QHENOMENOLOGY_77: (4)                selection_nudge_size = 0.05
SNGT_QHENOMENOLOGY_78: (4)                cursor_location_config = dict(
SNGT_QHENOMENOLOGY_79: (8)                    font_size=24,
SNGT_QHENOMENOLOGY_80: (8)                    fill_color=GREY_C,
SNGT_QHENOMENOLOGY_81: (8)                    num_decimal_places=3,
SNGT_QHENOMENOLOGY_82: (4)                )
SNGT_QHENOMENOLOGY_83: (4)                time_label_config = dict(
SNGT_QHENOMENOLOGY_84: (8)                    font_size=24,
SNGT_QHENOMENOLOGY_85: (8)                    fill_color=GREY_C,
SNGT_QHENOMENOLOGY_86: (8)                    num_decimal_places=1,
SNGT_QHENOMENOLOGY_87: (4)                )
SNGT_QHENOMENOLOGY_88: (4)                crosshair_width = 0.2
SNGT_QHENOMENOLOGY_89: (4)                crosshair_style = dict(
SNGT_QHENOMENOLOGY_90: (8)                    stroke_color=GREY_A,
SNGT_QHENOMENOLOGY_91: (8)                    stroke_width=[3, 0, 3],
SNGT_QHENOMENOLOGY_92: (4)                )
SNGT_QHENOMENOLOGY_93: (4)                def setup(self):
SNGT_QHENOMENOLOGY_94: (8)                    self.selection = Group()
SNGT_QHENOMENOLOGY_95: (8)                    self.selection_highlight =
self.get_selection_highlight()
SNGT_QHENOMENOLOGY_96: (8)                    self.selection_rectangle =
self.get_selection_rectangle()
SNGT_QHENOMENOLOGY_97: (8)                    self.crosshair = self.get_crosshair()
SNGT_QHENOMENOLOGY_98: (8)                    self.information_label =
self.get_information_label()
SNGT_QHENOMENOLOGY_99: (8)                    self.color_palette = self.get_color_palette()
SNGT_QHENOMENOLOGY_100: (8)                   self.unselectables = [
SNGT_QHENOMENOLOGY_101: (12)                      self.selection,
SNGT_QHENOMENOLOGY_102: (12)                      self.selection_highlight,
SNGT_QHENOMENOLOGY_103: (12)                      self.selection_rectangle,
SNGT_QHENOMENOLOGY_104: (12)                      self.crosshair,
SNGT_QHENOMENOLOGY_105: (12)                      self.information_label,
SNGT_QHENOMENOLOGY_106: (12)                      self.camera.frame
SNGT_QHENOMENOLOGY_107: (8)                   ]
SNGT_QHENOMENOLOGY_108: (8)                   self.select_top_level_mobs = True
SNGT_QHENOMENOLOGY_109: (8)                   self.regenerate_selection_search_set()
SNGT_QHENOMENOLOGY_110: (8)                   self.is_selecting = False
SNGT_QHENOMENOLOGY_111: (8)                   self.is_grabbing = False
SNGT_QHENOMENOLOGY_112: (8)                   self.add(self.selection_highlight)
SNGT_QHENOMENOLOGY_113: (4)               def get_selection_rectangle(self):
SNGT_QHENOMENOLOGY_114: (8)                   rect = Rectangle(
SNGT_QHENOMENOLOGY_115: (12)
stroke_color=self.selection_rectangle_stroke_color,
SNGT_QHENOMENOLOGY_116: (12)
stroke_width=self.selection_rectangle_stroke_width,
SNGT_QHENOMENOLOGY_117: (8)                   )
SNGT_QHENOMENOLOGY_118: (8)                   rect.fix_in_frame()
SNGT_QHENOMENOLOGY_119: (8)                   rect.fixed_corner = ORIGIN
SNGT_QHENOMENOLOGY_120: (8)                   rect.add_updater(self.update_selection_rectangle)
```

```
SNGT_QHENOMENOLOGY_121: (8)                     return rect
SNGT_QHENOMENOLOGY_122: (4)                 def update_selection_rectangle(self, rect: Rectangle):
SNGT_QHENOMENOLOGY_123: (8)                     p1 = rect.fixed_corner
SNGT_QHENOMENOLOGY_124: (8)                     p2 =
self.frame.to_fixed_frame_point(self.mouse_point.get_center())
SNGT_QHENOMENOLOGY_125: (8)                     rect.set_points_as_corners([
SNGT_QHENOMENOLOGY_126: (12)                        p1, np.array([p2[0], p1[1], 0]),
SNGT_QHENOMENOLOGY_127: (12)                        p2, np.array([p1[0], p2[1], 0]),
SNGT_QHENOMENOLOGY_128: (12)                        p1,
SNGT_QHENOMENOLOGY_129: (8)                     ])
SNGT_QHENOMENOLOGY_130: (8)                     return rect
SNGT_QHENOMENOLOGY_131: (4)                 def get_selection_highlight(self):
SNGT_QHENOMENOLOGY_132: (8)                     result = Group()
SNGT_QHENOMENOLOGY_133: (8)                     result.tracked_mobjects = []
SNGT_QHENOMENOLOGY_134: (8)                     result.add_updater(self.update_selection_highlight)
SNGT_QHENOMENOLOGY_135: (8)                     return result
SNGT_QHENOMENOLOGY_136: (4)                 def update_selection_highlight(self, highlight:
Mobject):
SNGT_QHENOMENOLOGY_137: (8)                     if set(highlight.tracked_mobjects) ==
set(self.selection):
SNGT_QHENOMENOLOGY_138: (12)                        return
SNGT_QHENOMENOLOGY_139: (8)                     highlight.tracked_mobjects = list(self.selection)
SNGT_QHENOMENOLOGY_140: (8)                     highlight.set_submobjects([
SNGT_QHENOMENOLOGY_141: (12)                        self.get_highlight(mob)
SNGT_QHENOMENOLOGY_142: (12)                        for mob in self.selection
SNGT_QHENOMENOLOGY_143: (8)                     ])
SNGT_QHENOMENOLOGY_144: (8)                     try:
SNGT_QHENOMENOLOGY_145: (12)                        index = min((
SNGT_QHENOMENOLOGY_146: (16)                            i for i, mob in enumerate(self.mobjects)
SNGT_QHENOMENOLOGY_147: (16)                            for sm in self.selection
SNGT_QHENOMENOLOGY_148: (16)                            if sm in mob.get_family()
SNGT_QHENOMENOLOGY_149: (12)                        ))
SNGT_QHENOMENOLOGY_150: (12)                        self.mobjects.remove(highlight)
SNGT_QHENOMENOLOGY_151: (12)                        self.mobjects.insert(index - 1, highlight)
SNGT_QHENOMENOLOGY_152: (8)                     except ValueError:
SNGT_QHENOMENOLOGY_153: (12)                        pass
SNGT_QHENOMENOLOGY_154: (4)                 def get_crosshair(self):
SNGT_QHENOMENOLOGY_155: (8)                     lines = VMobject().replicate(2)
SNGT_QHENOMENOLOGY_156: (8)                     lines[0].set_points([LEFT, ORIGIN, RIGHT])
SNGT_QHENOMENOLOGY_157: (8)                     lines[1].set_points([UP, ORIGIN, DOWN])
SNGT_QHENOMENOLOGY_158: (8)                     crosshair = VGroup(*lines)
SNGT_QHENOMENOLOGY_159: (8)                     crosshair.set_width(self.crosshair_width)
SNGT_QHENOMENOLOGY_160: (8)                     crosshair.set_style(**self.crosshair_style)
SNGT_QHENOMENOLOGY_161: (8)                     crosshair.set_animating_status(True)
SNGT_QHENOMENOLOGY_162: (8)                     crosshair.fix_in_frame()
SNGT_QHENOMENOLOGY_163: (8)                     return crosshair
SNGT_QHENOMENOLOGY_164: (4)                 def get_color_palette(self):
SNGT_QHENOMENOLOGY_165: (8)                     palette = VGroup(*(
SNGT_QHENOMENOLOGY_166: (12)                        Square(fill_color=color, fill_opacity=1,
side_length=1)
SNGT_QHENOMENOLOGY_167: (12)                        for color in self.palette_colors
SNGT_QHENOMENOLOGY_168: (8)                     ))
SNGT_QHENOMENOLOGY_169: (8)                     palette.set_stroke(width=0)
SNGT_QHENOMENOLOGY_170: (8)                     palette.arrange(RIGHT, buff=0.5)
SNGT_QHENOMENOLOGY_171: (8)                     palette.set_width(FRAME_WIDTH - 0.5)
SNGT_QHENOMENOLOGY_172: (8)                     palette.to_edge(DOWN, buff=SMALL_BUFF)
SNGT_QHENOMENOLOGY_173: (8)                     palette.fix_in_frame()
SNGT_QHENOMENOLOGY_174: (8)                     return palette
SNGT_QHENOMENOLOGY_175: (4)                 def get_information_label(self):
SNGT_QHENOMENOLOGY_176: (8)                     loc_label = VGroup(*(
SNGT_QHENOMENOLOGY_177: (12)                        DecimalNumber(**self.cursor_location_config)
SNGT_QHENOMENOLOGY_178: (12)                        for n in range(3)
SNGT_QHENOMENOLOGY_179: (8)                     ))
SNGT_QHENOMENOLOGY_180: (8)                     def update_coords(loc_label):
SNGT_QHENOMENOLOGY_181: (12)                        for mob, coord in zip(loc_label,
self.mouse_point.get_location()):
SNGT_QHENOMENOLOGY_182: (16)                            mob.set_value(coord)
SNGT_QHENOMENOLOGY_183: (12)                        loc_label.arrange(RIGHT,
buff=loc_label.get_height())
```

```
SNGT_QHENOMENOLOGY_184: (12)                        loc_label.to_corner(DR, buff=SMALL_BUFF)
SNGT_QHENOMENOLOGY_185: (12)                        loc_label.fix_in_frame()
SNGT_QHENOMENOLOGY_186: (12)                        return loc_label
SNGT_QHENOMENOLOGY_187: (8)                     loc_label.add_updater(update_coords)
SNGT_QHENOMENOLOGY_188: (8)                     time_label = DecimalNumber(0,
**self.time_label_config)
SNGT_QHENOMENOLOGY_189: (8)                     time_label.to_corner(DL, buff=SMALL_BUFF)
SNGT_QHENOMENOLOGY_190: (8)                     time_label.fix_in_frame()
SNGT_QHENOMENOLOGY_191: (8)                     time_label.add_updater(lambda m, dt:
m.increment_value(dt))
SNGT_QHENOMENOLOGY_192: (8)                     return VGroup(loc_label, time_label)
SNGT_QHENOMENOLOGY_193: (4)                 def get_state(self):
SNGT_QHENOMENOLOGY_194: (8)                     return SceneState(self, ignore=[
SNGT_QHENOMENOLOGY_195: (12)                        self.selection_highlight,
SNGT_QHENOMENOLOGY_196: (12)                        self.selection_rectangle,
SNGT_QHENOMENOLOGY_197: (12)                        self.crosshair,
SNGT_QHENOMENOLOGY_198: (8)                     ])
SNGT_QHENOMENOLOGY_199: (4)                 def restore_state(self, scene_state: SceneState):
SNGT_QHENOMENOLOGY_200: (8)                     super().restore_state(scene_state)
SNGT_QHENOMENOLOGY_201: (8)                     self.mobjects.insert(0, self.selection_highlight)
SNGT_QHENOMENOLOGY_202: (4)                 def add(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_203: (8)                     super().add(*mobjects)
SNGT_QHENOMENOLOGY_204: (8)                     self.regenerate_selection_search_set()
SNGT_QHENOMENOLOGY_205: (4)                 def remove(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_206: (8)                     super().remove(*mobjects)
SNGT_QHENOMENOLOGY_207: (8)                     self.regenerate_selection_search_set()
SNGT_QHENOMENOLOGY_208: (4)                 def toggle_selection_mode(self):
SNGT_QHENOMENOLOGY_209: (8)                     self.select_top_level_mobs = not
self.select_top_level_mobs
SNGT_QHENOMENOLOGY_210: (8)                     self.refresh_selection_scope()
SNGT_QHENOMENOLOGY_211: (8)                     self.regenerate_selection_search_set()
SNGT_QHENOMENOLOGY_212: (4)                 def get_selection_search_set(self) -> list[Mobject]:
SNGT_QHENOMENOLOGY_213: (8)                     return self.selection_search_set
SNGT_QHENOMENOLOGY_214: (4)                 def regenerate_selection_search_set(self):
SNGT_QHENOMENOLOGY_215: (8)                     selectable = list(filter(
SNGT_QHENOMENOLOGY_216: (12)                        lambda m: m not in self.unselectables,
SNGT_QHENOMENOLOGY_217: (12)                        self.mobjects
SNGT_QHENOMENOLOGY_218: (8)                     ))
SNGT_QHENOMENOLOGY_219: (8)                     if self.select_top_level_mobs:
SNGT_QHENOMENOLOGY_220: (12)                        self.selection_search_set = selectable
SNGT_QHENOMENOLOGY_221: (8)                     else:
SNGT_QHENOMENOLOGY_222: (12)                        self.selection_search_set = [
SNGT_QHENOMENOLOGY_223: (16)                            submob
SNGT_QHENOMENOLOGY_224: (16)                            for mob in selectable
SNGT_QHENOMENOLOGY_225: (16)                            for submob in
mob.family_members_with_points()
SNGT_QHENOMENOLOGY_226: (12)                        ]
SNGT_QHENOMENOLOGY_227: (4)                 def refresh_selection_scope(self):
SNGT_QHENOMENOLOGY_228: (8)                     curr = list(self.selection)
SNGT_QHENOMENOLOGY_229: (8)                     if self.select_top_level_mobs:
SNGT_QHENOMENOLOGY_230: (12)                        self.selection.set_submobjects([
SNGT_QHENOMENOLOGY_231: (16)                            mob
SNGT_QHENOMENOLOGY_232: (16)                            for mob in self.mobjects
SNGT_QHENOMENOLOGY_233: (16)                            if any(sm in mob.get_family() for sm in
curr)
SNGT_QHENOMENOLOGY_234: (12)                        ])
SNGT_QHENOMENOLOGY_235: (12)
self.selection.refresh_bounding_box(recurse_down=True)
SNGT_QHENOMENOLOGY_236: (8)                     else:
SNGT_QHENOMENOLOGY_237: (12)                        self.selection.set_submobjects(
SNGT_QHENOMENOLOGY_238: (16)                            extract_mobject_family_members(
SNGT_QHENOMENOLOGY_239: (20)                                curr, exclude_pointless=True,
SNGT_QHENOMENOLOGY_240: (16)                            )
SNGT_QHENOMENOLOGY_241: (12)                        )
SNGT_QHENOMENOLOGY_242: (4)                 def get_corner_dots(self, mobject: Mobject) -> Mobject:
SNGT_QHENOMENOLOGY_243: (8)                     dots = DotCloud(**self.corner_dot_config)
SNGT_QHENOMENOLOGY_244: (8)                     radius = float(self.corner_dot_config["radius"])
SNGT_QHENOMENOLOGY_245: (8)                     if mobject.get_depth() < 1e-2:
SNGT_QHENOMENOLOGY_246: (12)                        vects = [DL, UL, UR, DR]
```

```
SNGT_QHENOMENOLOGY_247: (8)                     else:
SNGT_QHENOMENOLOGY_248: (12)                         vects = np.array(list(it.product(*3 * [[-1,
1]])))
SNGT_QHENOMENOLOGY_249: (8)                     dots.add_updater(lambda d: d.set_points([
SNGT_QHENOMENOLOGY_250: (12)                         mobject.get_corner(v) + v * radius
SNGT_QHENOMENOLOGY_251: (12)                         for v in vects
SNGT_QHENOMENOLOGY_252: (8)                     ]))
SNGT_QHENOMENOLOGY_253: (8)                     return dots
SNGT_QHENOMENOLOGY_254: (4)                 def get_highlight(self, mobject: Mobject) -> Mobject:
SNGT_QHENOMENOLOGY_255: (8)                     if isinstance(mobject, VMobject) and
mobject.has_points() and not self.select_top_level_mobs:
SNGT_QHENOMENOLOGY_256: (12)                         length = max([mobject.get_height(),
mobject.get_width()])
SNGT_QHENOMENOLOGY_257: (12)                         result = VHighlight(
SNGT_QHENOMENOLOGY_258: (16)                             mobject,
SNGT_QHENOMENOLOGY_259: (16)                             max_stroke_addition=min([50 * length, 10]),
SNGT_QHENOMENOLOGY_260: (12)                         )
SNGT_QHENOMENOLOGY_261: (12)                         result.add_updater(lambda m: m.replace(mobject,
stretch=True))
SNGT_QHENOMENOLOGY_262: (12)                         return result
SNGT_QHENOMENOLOGY_263: (8)                     elif isinstance(mobject, DotCloud):
SNGT_QHENOMENOLOGY_264: (12)                         return Mobject()
SNGT_QHENOMENOLOGY_265: (8)                     else:
SNGT_QHENOMENOLOGY_266: (12)                         return self.get_corner_dots(mobject)
SNGT_QHENOMENOLOGY_267: (4)                 def add_to_selection(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_268: (8)                     mobs = list(filter(
SNGT_QHENOMENOLOGY_269: (12)                         lambda m: m not in self.unselectables and m not
in self.selection,
SNGT_QHENOMENOLOGY_270: (12)                         mobjects
SNGT_QHENOMENOLOGY_271: (8)                     ))
SNGT_QHENOMENOLOGY_272: (8)                     if len(mobs) == 0:
SNGT_QHENOMENOLOGY_273: (12)                         return
SNGT_QHENOMENOLOGY_274: (8)                     self.selection.add(*mobs)
SNGT_QHENOMENOLOGY_275: (8)                     for mob in mobs:
SNGT_QHENOMENOLOGY_276: (12)                         mob.set_animating_status(True)
SNGT_QHENOMENOLOGY_277: (4)                 def toggle_from_selection(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_278: (8)                     for mob in mobjects:
SNGT_QHENOMENOLOGY_279: (12)                         if mob in self.selection:
SNGT_QHENOMENOLOGY_280: (16)                             self.selection.remove(mob)
SNGT_QHENOMENOLOGY_281: (16)                             mob.set_animating_status(False)
SNGT_QHENOMENOLOGY_282: (16)                             mob.refresh_bounding_box()
SNGT_QHENOMENOLOGY_283: (12)                         else:
SNGT_QHENOMENOLOGY_284: (16)                             self.add_to_selection(mob)
SNGT_QHENOMENOLOGY_285: (4)                 def clear_selection(self):
SNGT_QHENOMENOLOGY_286: (8)                     for mob in self.selection:
SNGT_QHENOMENOLOGY_287: (12)                         mob.set_animating_status(False)
SNGT_QHENOMENOLOGY_288: (12)                         mob.refresh_bounding_box()
SNGT_QHENOMENOLOGY_289: (8)                     self.selection.set_submobjects([])
SNGT_QHENOMENOLOGY_290: (4)                 def disable_interaction(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_291: (8)                     for mob in mobjects:
SNGT_QHENOMENOLOGY_292: (12)                         for sm in mob.get_family():
SNGT_QHENOMENOLOGY_293: (16)                             self.unselectables.append(sm)
SNGT_QHENOMENOLOGY_294: (8)                     self.regenerate_selection_search_set()
SNGT_QHENOMENOLOGY_295: (4)                 def enable_interaction(self, *mobjects: Mobject):
SNGT_QHENOMENOLOGY_296: (8)                     for mob in mobjects:
SNGT_QHENOMENOLOGY_297: (12)                         for sm in mob.get_family():
SNGT_QHENOMENOLOGY_298: (16)                             if sm in self.unselectables:
SNGT_QHENOMENOLOGY_299: (20)                                 self.unselectables.remove(sm)
SNGT_QHENOMENOLOGY_300: (4)                 def copy_selection(self):
SNGT_QHENOMENOLOGY_301: (8)                     names = []
SNGT_QHENOMENOLOGY_302: (8)                     shell = get_ipython()
SNGT_QHENOMENOLOGY_303: (8)                     for mob in self.selection:
SNGT_QHENOMENOLOGY_304: (12)                         name = str(id(mob))
SNGT_QHENOMENOLOGY_305: (12)                         if shell is None:
SNGT_QHENOMENOLOGY_306: (16)                             continue
SNGT_QHENOMENOLOGY_307: (12)                         for key, value in shell.user_ns.items():
SNGT_QHENOMENOLOGY_308: (16)                             if mob is value:
SNGT_QHENOMENOLOGY_309: (20)                                 name = key
SNGT_QHENOMENOLOGY_310: (12)                         names.append(name)
```

```
SNGT_QHENOMENOLOGY_311: (8)                     pyperclip.copy(", ".join(names))
SNGT_QHENOMENOLOGY_312: (4)                 def paste_selection(self):
SNGT_QHENOMENOLOGY_313: (8)                     clipboard_str = pyperclip.paste()
SNGT_QHENOMENOLOGY_314: (8)                     try:
SNGT_QHENOMENOLOGY_315: (12)                        ids = map(int, clipboard_str.split(","))
SNGT_QHENOMENOLOGY_316: (12)                        mobs = map(self.id_to_mobject, ids)
SNGT_QHENOMENOLOGY_317: (12)                        mob_copies = [m.copy() for m in mobs if m is
not None]
SNGT_QHENOMENOLOGY_318: (12)                        self.clear_selection()
SNGT_QHENOMENOLOGY_319: (12)                        self.play(*(
SNGT_QHENOMENOLOGY_320: (16)                            FadeIn(mc, run_time=0.5, scale=1.5)
SNGT_QHENOMENOLOGY_321: (16)                            for mc in mob_copies
SNGT_QHENOMENOLOGY_322: (12)                        ))
SNGT_QHENOMENOLOGY_323: (12)                        self.add_to_selection(*mob_copies)
SNGT_QHENOMENOLOGY_324: (12)                        return
SNGT_QHENOMENOLOGY_325: (8)                     except ValueError:
SNGT_QHENOMENOLOGY_326: (12)                        pass
SNGT_QHENOMENOLOGY_327: (8)                     if set("\\^=+").intersection(clipboard_str):  #
Proxy to text for LaTeX
SNGT_QHENOMENOLOGY_328: (12)                        try:
SNGT_QHENOMENOLOGY_329: (16)                            new_mob = Tex(clipboard_str)
SNGT_QHENOMENOLOGY_330: (12)                        except LatexError:
SNGT_QHENOMENOLOGY_331: (16)                            return
SNGT_QHENOMENOLOGY_332: (8)                     else:
SNGT_QHENOMENOLOGY_333: (12)                        new_mob = Text(clipboard_str)
SNGT_QHENOMENOLOGY_334: (8)                     self.clear_selection()
SNGT_QHENOMENOLOGY_335: (8)                     self.add(new_mob)
SNGT_QHENOMENOLOGY_336: (8)                     self.add_to_selection(new_mob)
SNGT_QHENOMENOLOGY_337: (4)                 def delete_selection(self):
SNGT_QHENOMENOLOGY_338: (8)                     self.remove(*self.selection)
SNGT_QHENOMENOLOGY_339: (8)                     self.clear_selection()
SNGT_QHENOMENOLOGY_340: (4)                 def enable_selection(self):
SNGT_QHENOMENOLOGY_341: (8)                     self.is_selecting = True
SNGT_QHENOMENOLOGY_342: (8)                     self.add(self.selection_rectangle)
SNGT_QHENOMENOLOGY_343: (8)                     self.selection_rectangle.fixed_corner =
self.frame.to_fixed_frame_point(
SNGT_QHENOMENOLOGY_344: (12)                        self.mouse_point.get_center()
SNGT_QHENOMENOLOGY_345: (8)                     )
SNGT_QHENOMENOLOGY_346: (4)                 def gather_new_selection(self):
SNGT_QHENOMENOLOGY_347: (8)                     self.is_selecting = False
SNGT_QHENOMENOLOGY_348: (8)                     if self.selection_rectangle in self.mobjects:
SNGT_QHENOMENOLOGY_349: (12)                        self.remove(self.selection_rectangle)
SNGT_QHENOMENOLOGY_350: (12)                        additions = []
SNGT_QHENOMENOLOGY_351: (12)                        for mob in
reversed(self.get_selection_search_set()):
SNGT_QHENOMENOLOGY_352: (16)                            if
self.selection_rectangle.is_touching(mob):
SNGT_QHENOMENOLOGY_353: (20)                                additions.append(mob)
SNGT_QHENOMENOLOGY_354: (20)                                if
self.selection_rectangle.get_arc_length() < 1e-2:
SNGT_QHENOMENOLOGY_355: (24)                                    break
SNGT_QHENOMENOLOGY_356: (12)                        self.toggle_from_selection(*additions)
SNGT_QHENOMENOLOGY_357: (4)                 def prepare_grab(self):
SNGT_QHENOMENOLOGY_358: (8)                     mp = self.mouse_point.get_center()
SNGT_QHENOMENOLOGY_359: (8)                     self.mouse_to_selection = mp -
self.selection.get_center()
SNGT_QHENOMENOLOGY_360: (8)                     self.is_grabbing = True
SNGT_QHENOMENOLOGY_361: (4)                 def prepare_resizing(self, about_corner=False):
SNGT_QHENOMENOLOGY_362: (8)                     center = self.selection.get_center()
SNGT_QHENOMENOLOGY_363: (8)                     mp = self.mouse_point.get_center()
SNGT_QHENOMENOLOGY_364: (8)                     if about_corner:
SNGT_QHENOMENOLOGY_365: (12)                        self.scale_about_point =
self.selection.get_corner(center - mp)
SNGT_QHENOMENOLOGY_366: (8)                     else:
SNGT_QHENOMENOLOGY_367: (12)                        self.scale_about_point = center
SNGT_QHENOMENOLOGY_368: (8)                     self.scale_ref_vect = mp - self.scale_about_point
SNGT_QHENOMENOLOGY_369: (8)                     self.scale_ref_width = self.selection.get_width()
SNGT_QHENOMENOLOGY_370: (8)                     self.scale_ref_height = self.selection.get_height()
SNGT_QHENOMENOLOGY_371: (4)                 def toggle_color_palette(self):
```

```
SNGT_QHENOMENOLOGY_372: (8)                     if len(self.selection) == 0:
SNGT_QHENOMENOLOGY_373: (12)                         return
SNGT_QHENOMENOLOGY_374: (8)                     if self.color_palette not in self.mobjects:
SNGT_QHENOMENOLOGY_375: (12)                         self.save_state()
SNGT_QHENOMENOLOGY_376: (12)                         self.add(self.color_palette)
SNGT_QHENOMENOLOGY_377: (8)                     else:
SNGT_QHENOMENOLOGY_378: (12)                         self.remove(self.color_palette)
SNGT_QHENOMENOLOGY_379: (4)                 def display_information(self, show=True):
SNGT_QHENOMENOLOGY_380: (8)                     if show:
SNGT_QHENOMENOLOGY_381: (12)                         self.add(self.information_label)
SNGT_QHENOMENOLOGY_382: (8)                     else:
SNGT_QHENOMENOLOGY_383: (12)                         self.remove(self.information_label)
SNGT_QHENOMENOLOGY_384: (4)                 def group_selection(self):
SNGT_QHENOMENOLOGY_385: (8)                     group = self.get_group(*self.selection)
SNGT_QHENOMENOLOGY_386: (8)                     self.add(group)
SNGT_QHENOMENOLOGY_387: (8)                     self.clear_selection()
SNGT_QHENOMENOLOGY_388: (8)                     self.add_to_selection(group)
SNGT_QHENOMENOLOGY_389: (4)                 def ungroup_selection(self):
SNGT_QHENOMENOLOGY_390: (8)                     pieces = []
SNGT_QHENOMENOLOGY_391: (8)                     for mob in list(self.selection):
SNGT_QHENOMENOLOGY_392: (12)                         self.remove(mob)
SNGT_QHENOMENOLOGY_393: (12)                         pieces.extend(list(mob))
SNGT_QHENOMENOLOGY_394: (8)                     self.clear_selection()
SNGT_QHENOMENOLOGY_395: (8)                     self.add(*pieces)
SNGT_QHENOMENOLOGY_396: (8)                     self.add_to_selection(*pieces)
SNGT_QHENOMENOLOGY_397: (4)                 def nudge_selection(self, vect: np.ndarray, large: bool
= False):
SNGT_QHENOMENOLOGY_398: (8)                     nudge = self.selection_nudge_size
SNGT_QHENOMENOLOGY_399: (8)                     if large:
SNGT_QHENOMENOLOGY_400: (12)                         nudge *= 10
SNGT_QHENOMENOLOGY_401: (8)                     self.selection.shift(nudge * vect)
SNGT_QHENOMENOLOGY_402: (4)                 def on_key_press(self, symbol: int, modifiers: int) ->
None:
SNGT_QHENOMENOLOGY_403: (8)                     super().on_key_press(symbol, modifiers)
SNGT_QHENOMENOLOGY_404: (8)                     char = chr(symbol)
SNGT_QHENOMENOLOGY_405: (8)                     if char == SELECT_KEY and (modifiers &
ALL_MODIFIERS) == 0:
SNGT_QHENOMENOLOGY_406: (12)                         self.enable_selection()
SNGT_QHENOMENOLOGY_407: (8)                     if char == UNSELECT_KEY:
SNGT_QHENOMENOLOGY_408: (12)                         self.clear_selection()
SNGT_QHENOMENOLOGY_409: (8)                     elif char in GRAB_KEYS and (modifiers &
ALL_MODIFIERS) == 0:
SNGT_QHENOMENOLOGY_410: (12)                         self.prepare_grab()
SNGT_QHENOMENOLOGY_411: (8)                     elif char == RESIZE_KEY and (modifiers &
PygletWindowKeys.MOD_SHIFT):
SNGT_QHENOMENOLOGY_412: (12)                         self.prepare_resizing(about_corner=((modifiers
& PygletWindowKeys.MOD_SHIFT) > 0))
SNGT_QHENOMENOLOGY_413: (8)                     elif symbol == PygletWindowKeys.LSHIFT:
SNGT_QHENOMENOLOGY_414: (12)                         if self.window.is_key_pressed(ord("t")):
SNGT_QHENOMENOLOGY_415: (16)                             self.prepare_resizing(about_corner=True)
SNGT_QHENOMENOLOGY_416: (8)                     elif char == COLOR_KEY and (modifiers &
ALL_MODIFIERS) == 0:
SNGT_QHENOMENOLOGY_417: (12)                         self.toggle_color_palette()
SNGT_QHENOMENOLOGY_418: (8)                     elif char == INFORMATION_KEY and (modifiers &
ALL_MODIFIERS) == 0:
SNGT_QHENOMENOLOGY_419: (12)                         self.display_information()
SNGT_QHENOMENOLOGY_420: (8)                     elif char == "c" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_421: (12)                         self.copy_selection()
SNGT_QHENOMENOLOGY_422: (8)                     elif char == "v" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_423: (12)                         self.paste_selection()
SNGT_QHENOMENOLOGY_424: (8)                     elif char == "x" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_425: (12)                         self.copy_selection()
SNGT_QHENOMENOLOGY_426: (12)                         self.delete_selection()
SNGT_QHENOMENOLOGY_427: (8)                     elif symbol == PygletWindowKeys.BACKSPACE:
SNGT_QHENOMENOLOGY_428: (12)                         self.delete_selection()
SNGT_QHENOMENOLOGY_429: (8)                     elif char == "a" and (modifiers &
```

```
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_430: (12)                         self.clear_selection()
SNGT_QHENOMENOLOGY_431: (12)                         self.add_to_selection(*self.mobjects)
SNGT_QHENOMENOLOGY_432: (8)                     elif char == "g" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_433: (12)                         self.group_selection()
SNGT_QHENOMENOLOGY_434: (8)                     elif char == "g" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL | PygletWindowKeys.MOD_SHIFT)):
SNGT_QHENOMENOLOGY_435: (12)                         self.ungroup_selection()
SNGT_QHENOMENOLOGY_436: (8)                     elif char == "t" and (modifiers &
(PygletWindowKeys.MOD_COMMAND | PygletWindowKeys.MOD_CTRL)):
SNGT_QHENOMENOLOGY_437: (12)                         self.toggle_selection_mode()
SNGT_QHENOMENOLOGY_438: (8)                     elif char == "d" and (modifiers &
PygletWindowKeys.MOD_SHIFT):
SNGT_QHENOMENOLOGY_439: (12)                         self.copy_frame_positioning()
SNGT_QHENOMENOLOGY_440: (8)                     elif char == "c" and (modifiers &
PygletWindowKeys.MOD_SHIFT):
SNGT_QHENOMENOLOGY_441: (12)                         self.copy_cursor_position()
SNGT_QHENOMENOLOGY_442: (8)                     elif symbol in ARROW_SYMBOLS:
SNGT_QHENOMENOLOGY_443: (12)                         self.nudge_selection(
SNGT_QHENOMENOLOGY_444: (16)                             vect=[LEFT, UP, RIGHT, DOWN]
[ARROW_SYMBOLS.index(symbol)],
SNGT_QHENOMENOLOGY_445: (16)                             large=(modifiers &
PygletWindowKeys.MOD_SHIFT),
SNGT_QHENOMENOLOGY_446: (12)                         )
SNGT_QHENOMENOLOGY_447: (8)                     if char == CURSOR_KEY:
SNGT_QHENOMENOLOGY_448: (12)                         if self.crosshair in self.mobjects:
SNGT_QHENOMENOLOGY_449: (16)                             self.remove(self.crosshair)
SNGT_QHENOMENOLOGY_450: (12)                         else:
SNGT_QHENOMENOLOGY_451: (16)                             self.add(self.crosshair)
SNGT_QHENOMENOLOGY_452: (8)                     if char == SELECT_KEY:
SNGT_QHENOMENOLOGY_453: (12)                         self.add(self.crosshair)
SNGT_QHENOMENOLOGY_454: (8)                     if char in [GRAB_KEY, X_GRAB_KEY, Y_GRAB_KEY,
RESIZE_KEY]:
SNGT_QHENOMENOLOGY_455: (12)                         self.save_state()
SNGT_QHENOMENOLOGY_456: (4)                 def on_key_release(self, symbol: int, modifiers: int) -
> None:
SNGT_QHENOMENOLOGY_457: (8)                     super().on_key_release(symbol, modifiers)
SNGT_QHENOMENOLOGY_458: (8)                     if chr(symbol) == SELECT_KEY:
SNGT_QHENOMENOLOGY_459: (12)                         self.gather_new_selection()
SNGT_QHENOMENOLOGY_460: (8)                     if chr(symbol) in GRAB_KEYS:
SNGT_QHENOMENOLOGY_461: (12)                         self.is_grabbing = False
SNGT_QHENOMENOLOGY_462: (8)                     elif chr(symbol) == INFORMATION_KEY:
SNGT_QHENOMENOLOGY_463: (12)                         self.display_information(False)
SNGT_QHENOMENOLOGY_464: (8)                     elif symbol == PygletWindowKeys.LSHIFT and
self.window.is_key_pressed(ord(RESIZE_KEY)):
SNGT_QHENOMENOLOGY_465: (12)                         self.prepare_resizing(about_corner=False)
SNGT_QHENOMENOLOGY_466: (4)                 def handle_grabbing(self, point: Vect3):
SNGT_QHENOMENOLOGY_467: (8)                     diff = point - self.mouse_to_selection
SNGT_QHENOMENOLOGY_468: (8)                     if self.window.is_key_pressed(ord(GRAB_KEY)):
SNGT_QHENOMENOLOGY_469: (12)                         self.selection.move_to(diff)
SNGT_QHENOMENOLOGY_470: (8)                     elif self.window.is_key_pressed(ord(X_GRAB_KEY)):
SNGT_QHENOMENOLOGY_471: (12)                         self.selection.set_x(diff[0])
SNGT_QHENOMENOLOGY_472: (8)                     elif self.window.is_key_pressed(ord(Y_GRAB_KEY)):
SNGT_QHENOMENOLOGY_473: (12)                         self.selection.set_y(diff[1])
SNGT_QHENOMENOLOGY_474: (4)                 def handle_resizing(self, point: Vect3):
SNGT_QHENOMENOLOGY_475: (8)                     if not hasattr(self, "scale_about_point"):
SNGT_QHENOMENOLOGY_476: (12)                         return
SNGT_QHENOMENOLOGY_477: (8)                     vect = point - self.scale_about_point
SNGT_QHENOMENOLOGY_478: (8)                     if
self.window.is_key_pressed(PygletWindowKeys.LCTRL):
SNGT_QHENOMENOLOGY_479: (12)                         for i in (0, 1):
SNGT_QHENOMENOLOGY_480: (16)                             scalar = vect[i] / self.scale_ref_vect[i]
SNGT_QHENOMENOLOGY_481: (16)                             self.selection.rescale_to_fit(
SNGT_QHENOMENOLOGY_482: (20)                                 scalar * [self.scale_ref_width,
self.scale_ref_height][i],
SNGT_QHENOMENOLOGY_483: (20)                                 dim=i,
SNGT_QHENOMENOLOGY_484: (20)                                 about_point=self.scale_about_point,
SNGT_QHENOMENOLOGY_485: (20)                                 stretch=True,
```

```
SNGT_QHENOMENOLOGY_486: (16)                            )
SNGT_QHENOMENOLOGY_487: (8)                     else:
SNGT_QHENOMENOLOGY_488: (12)                        scalar = get_norm(vect) /
get_norm(self.scale_ref_vect)
SNGT_QHENOMENOLOGY_489: (12)                        self.selection.set_width(
SNGT_QHENOMENOLOGY_490: (16)                            scalar * self.scale_ref_width,
SNGT_QHENOMENOLOGY_491: (16)                            about_point=self.scale_about_point
SNGT_QHENOMENOLOGY_492: (12)                        )
SNGT_QHENOMENOLOGY_493: (4)                 def handle_sweeping_selection(self, point: Vect3):
SNGT_QHENOMENOLOGY_494: (8)                     mob = self.point_to_mobject(
SNGT_QHENOMENOLOGY_495: (12)                        point,
SNGT_QHENOMENOLOGY_496: (12)                        search_set=self.get_selection_search_set(),
SNGT_QHENOMENOLOGY_497: (12)                        buff=SMALL_BUFF
SNGT_QHENOMENOLOGY_498: (8)                     )
SNGT_QHENOMENOLOGY_499: (8)                     if mob is not None:
SNGT_QHENOMENOLOGY_500: (12)                        self.add_to_selection(mob)
SNGT_QHENOMENOLOGY_501: (4)                 def choose_color(self, point: Vect3):
SNGT_QHENOMENOLOGY_502: (8)                     to_search = [
SNGT_QHENOMENOLOGY_503: (12)                        sm
SNGT_QHENOMENOLOGY_504: (12)                        for mobject in self.mobjects
SNGT_QHENOMENOLOGY_505: (12)                        for sm in mobject.family_members_with_points()
SNGT_QHENOMENOLOGY_506: (12)                        if mobject not in self.unselectables
SNGT_QHENOMENOLOGY_507: (8)                     ]
SNGT_QHENOMENOLOGY_508: (8)                     mob = self.point_to_mobject(point, to_search)
SNGT_QHENOMENOLOGY_509: (8)                     if mob is not None:
SNGT_QHENOMENOLOGY_510: (12)                        self.selection.set_color(mob.get_color())
SNGT_QHENOMENOLOGY_511: (8)                     self.remove(self.color_palette)
SNGT_QHENOMENOLOGY_512: (4)                 def on_mouse_motion(self, point: Vect3, d_point: Vect3)
-> None:
SNGT_QHENOMENOLOGY_513: (8)                     super().on_mouse_motion(point, d_point)
SNGT_QHENOMENOLOGY_514: (8)
self.crosshair.move_to(self.frame.to_fixed_frame_point(point))
SNGT_QHENOMENOLOGY_515: (8)                     if self.is_grabbing:
SNGT_QHENOMENOLOGY_516: (12)                        self.handle_grabbing(point)
SNGT_QHENOMENOLOGY_517: (8)                     elif self.window.is_key_pressed(ord(RESIZE_KEY)):
SNGT_QHENOMENOLOGY_518: (12)                        self.handle_resizing(point)
SNGT_QHENOMENOLOGY_519: (8)                     elif self.window.is_key_pressed(ord(SELECT_KEY))
and self.window.is_key_pressed(PygletWindowKeys.LSHIFT):
SNGT_QHENOMENOLOGY_520: (12)                        self.handle_sweeping_selection(point)
SNGT_QHENOMENOLOGY_521: (4)                 def on_mouse_drag(
SNGT_QHENOMENOLOGY_522: (8)                     self,
SNGT_QHENOMENOLOGY_523: (8)                     point: Vect3,
SNGT_QHENOMENOLOGY_524: (8)                     d_point: Vect3,
SNGT_QHENOMENOLOGY_525: (8)                     buttons: int,
SNGT_QHENOMENOLOGY_526: (8)                     modifiers: int
SNGT_QHENOMENOLOGY_527: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_528: (8)                     super().on_mouse_drag(point, d_point, buttons,
modifiers)
SNGT_QHENOMENOLOGY_529: (8)
self.crosshair.move_to(self.frame.to_fixed_frame_point(point))
SNGT_QHENOMENOLOGY_530: (4)                 def on_mouse_release(self, point: Vect3, button: int,
mods: int) -> None:
SNGT_QHENOMENOLOGY_531: (8)                     super().on_mouse_release(point, button, mods)
SNGT_QHENOMENOLOGY_532: (8)                     if self.color_palette in self.mobjects:
SNGT_QHENOMENOLOGY_533: (12)                        self.choose_color(point)
SNGT_QHENOMENOLOGY_534: (8)                     else:
SNGT_QHENOMENOLOGY_535: (12)                        self.clear_selection()
SNGT_QHENOMENOLOGY_536: (4)                 def copy_frame_positioning(self):
SNGT_QHENOMENOLOGY_537: (8)                     frame = self.frame
SNGT_QHENOMENOLOGY_538: (8)                     center = frame.get_center()
SNGT_QHENOMENOLOGY_539: (8)                     height = frame.get_height()
SNGT_QHENOMENOLOGY_540: (8)                     angles = frame.get_euler_angles()
SNGT_QHENOMENOLOGY_541: (8)                     call = f"reorient("
SNGT_QHENOMENOLOGY_542: (8)                     theta, phi, gamma = (angles / DEG).astype(int)
SNGT_QHENOMENOLOGY_543: (8)                     call += f"{theta}, {phi}, {gamma}"
SNGT_QHENOMENOLOGY_544: (8)                     if any(center != 0):
SNGT_QHENOMENOLOGY_545: (12)                        call += f", {tuple(np.round(center, 2))}"
SNGT_QHENOMENOLOGY_546: (8)                     if height != FRAME_HEIGHT:
SNGT_QHENOMENOLOGY_547: (12)                        call += ", {:.2f}".format(height)
```

```
SNGT_QHENOMENOLOGY_548: (8)                     call += ")"
SNGT_QHENOMENOLOGY_549: (8)                     pyperclip.copy(call)
SNGT_QHENOMENOLOGY_550: (4)                 def copy_cursor_position(self):
SNGT_QHENOMENOLOGY_551: (8)
pyperclip.copy(str(tuple(self.mouse_point.get_center().round(2))))
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 72 - scene_file_writer.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import os
SNGT_QHENOMENOLOGY_3: (0)               import platform
SNGT_QHENOMENOLOGY_4: (0)               import shutil
SNGT_QHENOMENOLOGY_5: (0)               import subprocess as sp
SNGT_QHENOMENOLOGY_6: (0)               import sys
SNGT_QHENOMENOLOGY_7: (0)               import numpy as np
SNGT_QHENOMENOLOGY_8: (0)               from pydub import AudioSegment
SNGT_QHENOMENOLOGY_9: (0)               from tqdm.auto import tqdm as ProgressDisplay
SNGT_QHENOMENOLOGY_10: (0)              from pathlib import Path
SNGT_QHENOMENOLOGY_11: (0)              from manimlib.logger import log
SNGT_QHENOMENOLOGY_12: (0)              from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_13: (0)              from manimlib.utils.file_ops import guarantee_existence
SNGT_QHENOMENOLOGY_14: (0)              from manimlib.utils.sounds import get_full_sound_file_path
SNGT_QHENOMENOLOGY_15: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_16: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_17: (4)                  from PIL.Image import Image
SNGT_QHENOMENOLOGY_18: (4)                  from manimlib.camera.camera import Camera
SNGT_QHENOMENOLOGY_19: (4)                  from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_20: (0)              class SceneFileWriter(object):
SNGT_QHENOMENOLOGY_21: (4)                  def __init__(
SNGT_QHENOMENOLOGY_22: (8)                      self,
SNGT_QHENOMENOLOGY_23: (8)                      scene: Scene,
SNGT_QHENOMENOLOGY_24: (8)                      write_to_movie: bool = False,
SNGT_QHENOMENOLOGY_25: (8)                      subdivide_output: bool = False,
SNGT_QHENOMENOLOGY_26: (8)                      png_mode: str = "RGBA",
SNGT_QHENOMENOLOGY_27: (8)                      save_last_frame: bool = False,
SNGT_QHENOMENOLOGY_28: (8)                      movie_file_extension: str = ".mp4",
SNGT_QHENOMENOLOGY_29: (8)                      output_directory: str = ".",
SNGT_QHENOMENOLOGY_30: (8)                      file_name: str | None = None,
SNGT_QHENOMENOLOGY_31: (8)                      open_file_upon_completion: bool = False,
SNGT_QHENOMENOLOGY_32: (8)                      show_file_location_upon_completion: bool = False,
SNGT_QHENOMENOLOGY_33: (8)                      quiet: bool = False,
SNGT_QHENOMENOLOGY_34: (8)                      total_frames: int = 0,
SNGT_QHENOMENOLOGY_35: (8)                      progress_description_len: int = 40,
SNGT_QHENOMENOLOGY_36: (8)                      ffmpeg_bin: str = "ffmpeg",
SNGT_QHENOMENOLOGY_37: (8)                      video_codec: str = "libx264",
SNGT_QHENOMENOLOGY_38: (8)                      pixel_format: str = "yuv420p",
SNGT_QHENOMENOLOGY_39: (8)                      saturation: float = 1.0,
SNGT_QHENOMENOLOGY_40: (8)                      gamma: float = 1.0,
SNGT_QHENOMENOLOGY_41: (4)                  ):
SNGT_QHENOMENOLOGY_42: (8)                      self.scene: Scene = scene
SNGT_QHENOMENOLOGY_43: (8)                      self.write_to_movie = write_to_movie
SNGT_QHENOMENOLOGY_44: (8)                      self.subdivide_output = subdivide_output
SNGT_QHENOMENOLOGY_45: (8)                      self.png_mode = png_mode
SNGT_QHENOMENOLOGY_46: (8)                      self.save_last_frame = save_last_frame
SNGT_QHENOMENOLOGY_47: (8)                      self.movie_file_extension = movie_file_extension
SNGT_QHENOMENOLOGY_48: (8)                      self.output_directory = output_directory
SNGT_QHENOMENOLOGY_49: (8)                      self.file_name = file_name
SNGT_QHENOMENOLOGY_50: (8)                      self.open_file_upon_completion =
open_file_upon_completion
SNGT_QHENOMENOLOGY_51: (8)                      self.show_file_location_upon_completion =
show_file_location_upon_completion
SNGT_QHENOMENOLOGY_52: (8)                      self.quiet = quiet
SNGT_QHENOMENOLOGY_53: (8)                      self.total_frames = total_frames
SNGT_QHENOMENOLOGY_54: (8)                      self.progress_description_len =
progress_description_len
SNGT_QHENOMENOLOGY_55: (8)                      self.ffmpeg_bin = ffmpeg_bin
SNGT_QHENOMENOLOGY_56: (8)                      self.video_codec = video_codec
```

```
SNGT_QHENOMENOLOGY_57: (8)                      self.pixel_format = pixel_format
SNGT_QHENOMENOLOGY_58: (8)                      self.saturation = saturation
SNGT_QHENOMENOLOGY_59: (8)                      self.gamma = gamma
SNGT_QHENOMENOLOGY_60: (8)                      self.writing_process: sp.Popen | None = None
SNGT_QHENOMENOLOGY_61: (8)                      self.progress_display: ProgressDisplay | None =
None
SNGT_QHENOMENOLOGY_62: (8)                      self.ended_with_interrupt: bool = False
SNGT_QHENOMENOLOGY_63: (8)                      self.init_output_directories()
SNGT_QHENOMENOLOGY_64: (8)                      self.init_audio()
SNGT_QHENOMENOLOGY_65: (4)                  def init_output_directories(self) -> None:
SNGT_QHENOMENOLOGY_66: (8)                      if self.save_last_frame:
SNGT_QHENOMENOLOGY_67: (12)                         self.image_file_path =
self.init_image_file_path()
SNGT_QHENOMENOLOGY_68: (8)                      if self.write_to_movie:
SNGT_QHENOMENOLOGY_69: (12)                         self.movie_file_path =
self.init_movie_file_path()
SNGT_QHENOMENOLOGY_70: (8)                      if self.subdivide_output:
SNGT_QHENOMENOLOGY_71: (12)                         self.partial_movie_directory =
self.init_partial_movie_directory()
SNGT_QHENOMENOLOGY_72: (4)                  def init_image_file_path(self) -> Path:
SNGT_QHENOMENOLOGY_73: (8)                      return
self.get_output_file_rootname().with_suffix(".png")
SNGT_QHENOMENOLOGY_74: (4)                  def init_movie_file_path(self) -> Path:
SNGT_QHENOMENOLOGY_75: (8)                      return
self.get_output_file_rootname().with_suffix(self.movie_file_extension)
SNGT_QHENOMENOLOGY_76: (4)                  def init_partial_movie_directory(self):
SNGT_QHENOMENOLOGY_77: (8)                      return
guarantee_existence(self.get_output_file_rootname())
SNGT_QHENOMENOLOGY_78: (4)                  def get_output_file_rootname(self) -> Path:
SNGT_QHENOMENOLOGY_79: (8)                      return Path(
SNGT_QHENOMENOLOGY_80: (12)                         guarantee_existence(self.output_directory),
SNGT_QHENOMENOLOGY_81: (12)                         self.get_output_file_name()
SNGT_QHENOMENOLOGY_82: (8)                      )
SNGT_QHENOMENOLOGY_83: (4)                  def get_output_file_name(self) -> str:
SNGT_QHENOMENOLOGY_84: (8)                      if self.file_name:
SNGT_QHENOMENOLOGY_85: (12)                         return self.file_name
SNGT_QHENOMENOLOGY_86: (8)                      name = str(self.scene)
SNGT_QHENOMENOLOGY_87: (8)                      saan = self.scene.start_at_animation_number
SNGT_QHENOMENOLOGY_88: (8)                      eaan = self.scene.end_at_animation_number
SNGT_QHENOMENOLOGY_89: (8)                      if saan is not None:
SNGT_QHENOMENOLOGY_90: (12)                         name += f"_{saan}"
SNGT_QHENOMENOLOGY_91: (8)                      if eaan is not None:
SNGT_QHENOMENOLOGY_92: (12)                         name += f"_{eaan}"
SNGT_QHENOMENOLOGY_93: (8)                      return name
SNGT_QHENOMENOLOGY_94: (4)                  def get_image_file_path(self) -> str:
SNGT_QHENOMENOLOGY_95: (8)                      return self.image_file_path
SNGT_QHENOMENOLOGY_96: (4)                  def get_next_partial_movie_path(self) -> str:
SNGT_QHENOMENOLOGY_97: (8)                      result = Path(self.partial_movie_directory, f"
{self.scene.num_plays:05}")
SNGT_QHENOMENOLOGY_98: (8)                      return
result.with_suffix(self.movie_file_extension)
SNGT_QHENOMENOLOGY_99: (4)                  def get_movie_file_path(self) -> str:
SNGT_QHENOMENOLOGY_100: (8)                     return self.movie_file_path
SNGT_QHENOMENOLOGY_101: (4)                 def init_audio(self) -> None:
SNGT_QHENOMENOLOGY_102: (8)                     self.includes_sound: bool = False
SNGT_QHENOMENOLOGY_103: (4)                 def create_audio_segment(self) -> None:
SNGT_QHENOMENOLOGY_104: (8)                     self.audio_segment = AudioSegment.silent()
SNGT_QHENOMENOLOGY_105: (4)                 def add_audio_segment(
SNGT_QHENOMENOLOGY_106: (8)                     self,
SNGT_QHENOMENOLOGY_107: (8)                     new_segment: AudioSegment,
SNGT_QHENOMENOLOGY_108: (8)                     time: float | None = None,
SNGT_QHENOMENOLOGY_109: (8)                     gain_to_background: float | None = None
SNGT_QHENOMENOLOGY_110: (4)                 ) -> None:
SNGT_QHENOMENOLOGY_111: (8)                     if not self.includes_sound:
SNGT_QHENOMENOLOGY_112: (12)                        self.includes_sound = True
SNGT_QHENOMENOLOGY_113: (12)                        self.create_audio_segment()
SNGT_QHENOMENOLOGY_114: (8)                     segment = self.audio_segment
SNGT_QHENOMENOLOGY_115: (8)                     curr_end = segment.duration_seconds
SNGT_QHENOMENOLOGY_116: (8)                     if time is None:
```

```
SNGT_QHENOMENOLOGY_117: (12)                     time = curr_end
SNGT_QHENOMENOLOGY_118: (8)                 if time < 0:
SNGT_QHENOMENOLOGY_119: (12)                     raise Exception("Adding sound at timestamp <
0")
SNGT_QHENOMENOLOGY_120: (8)                 new_end = time + new_segment.duration_seconds
SNGT_QHENOMENOLOGY_121: (8)                 diff = new_end - curr_end
SNGT_QHENOMENOLOGY_122: (8)                 if diff > 0:
SNGT_QHENOMENOLOGY_123: (12)                     segment = segment.append(
SNGT_QHENOMENOLOGY_124: (16)                         AudioSegment.silent(int(np.ceil(diff *
1000))),
SNGT_QHENOMENOLOGY_125: (16)                         crossfade=0,
SNGT_QHENOMENOLOGY_126: (12)                     )
SNGT_QHENOMENOLOGY_127: (8)                 self.audio_segment = segment.overlay(
SNGT_QHENOMENOLOGY_128: (12)                     new_segment,
SNGT_QHENOMENOLOGY_129: (12)                     position=int(1000 * time),
SNGT_QHENOMENOLOGY_130: (12)                     gain_during_overlay=gain_to_background,
SNGT_QHENOMENOLOGY_131: (8)                 )
SNGT_QHENOMENOLOGY_132: (4)             def add_sound(
SNGT_QHENOMENOLOGY_133: (8)                 self,
SNGT_QHENOMENOLOGY_134: (8)                 sound_file: str,
SNGT_QHENOMENOLOGY_135: (8)                 time: float | None = None,
SNGT_QHENOMENOLOGY_136: (8)                 gain: float | None = None,
SNGT_QHENOMENOLOGY_137: (8)                 gain_to_background: float | None = None
SNGT_QHENOMENOLOGY_138: (4)             ) -> None:
SNGT_QHENOMENOLOGY_139: (8)                 file_path = get_full_sound_file_path(sound_file)
SNGT_QHENOMENOLOGY_140: (8)                 new_segment = AudioSegment.from_file(file_path)
SNGT_QHENOMENOLOGY_141: (8)                 if gain:
SNGT_QHENOMENOLOGY_142: (12)                     new_segment = new_segment.apply_gain(gain)
SNGT_QHENOMENOLOGY_143: (8)                 self.add_audio_segment(new_segment, time,
gain_to_background)
SNGT_QHENOMENOLOGY_144: (4)             def begin(self) -> None:
SNGT_QHENOMENOLOGY_145: (8)                 if not self.subdivide_output and
self.write_to_movie:
SNGT_QHENOMENOLOGY_146: (12)
self.open_movie_pipe(self.get_movie_file_path())
SNGT_QHENOMENOLOGY_147: (4)             def begin_animation(self) -> None:
SNGT_QHENOMENOLOGY_148: (8)                 if self.subdivide_output and self.write_to_movie:
SNGT_QHENOMENOLOGY_149: (12)
self.open_movie_pipe(self.get_next_partial_movie_path())
SNGT_QHENOMENOLOGY_150: (4)             def end_animation(self) -> None:
SNGT_QHENOMENOLOGY_151: (8)                 if self.subdivide_output and self.write_to_movie:
SNGT_QHENOMENOLOGY_152: (12)                     self.close_movie_pipe()
SNGT_QHENOMENOLOGY_153: (4)             def finish(self) -> None:
SNGT_QHENOMENOLOGY_154: (8)                 if not self.subdivide_output and
self.write_to_movie:
SNGT_QHENOMENOLOGY_155: (12)                     self.close_movie_pipe()
SNGT_QHENOMENOLOGY_156: (12)                     if self.includes_sound:
SNGT_QHENOMENOLOGY_157: (16)                         self.add_sound_to_video()
SNGT_QHENOMENOLOGY_158: (12)
self.print_file_ready_message(self.get_movie_file_path())
SNGT_QHENOMENOLOGY_159: (8)                 if self.save_last_frame:
SNGT_QHENOMENOLOGY_160: (12)                     self.scene.update_frame(force_draw=True)
SNGT_QHENOMENOLOGY_161: (12)                     self.save_final_image(self.scene.get_image())
SNGT_QHENOMENOLOGY_162: (8)                 if self.should_open_file():
SNGT_QHENOMENOLOGY_163: (12)                     self.open_file()
SNGT_QHENOMENOLOGY_164: (4)             def open_movie_pipe(self, file_path: str) -> None:
SNGT_QHENOMENOLOGY_165: (8)                 stem, ext = os.path.splitext(file_path)
SNGT_QHENOMENOLOGY_166: (8)                 self.final_file_path = file_path
SNGT_QHENOMENOLOGY_167: (8)                 self.temp_file_path = stem + "_temp" + ext
SNGT_QHENOMENOLOGY_168: (8)                 fps = self.scene.camera.fps
SNGT_QHENOMENOLOGY_169: (8)                 width, height = self.scene.camera.get_pixel_shape()
SNGT_QHENOMENOLOGY_170: (8)                 vf_arg = 'vflip'
SNGT_QHENOMENOLOGY_171: (8)                 vf_arg += f',eq=saturation={self.saturation}:gamma=
{self.gamma}'
SNGT_QHENOMENOLOGY_172: (8)                 command = [
SNGT_QHENOMENOLOGY_173: (12)                     self.ffmpeg_bin,
SNGT_QHENOMENOLOGY_174: (12)                     '-y',  # overwrite output file if it exists
SNGT_QHENOMENOLOGY_175: (12)                     '-f', 'rawvideo',
SNGT_QHENOMENOLOGY_176: (12)                     '-s', f'{width}x{height}',  # size of one frame
```

```
SNGT_QHENOMENOLOGY_177: (12)                        '-pix_fmt', 'rgba',
SNGT_QHENOMENOLOGY_178: (12)                        '-r', str(fps),  # frames per second
SNGT_QHENOMENOLOGY_179: (12)                        '-i', '-',  # The input comes from a pipe
SNGT_QHENOMENOLOGY_180: (12)                        '-vf', vf_arg,
SNGT_QHENOMENOLOGY_181: (12)                        '-an',  # Tells ffmpeg not to expect any audio
SNGT_QHENOMENOLOGY_182: (12)                        '-loglevel', 'error',
SNGT_QHENOMENOLOGY_183: (8)                    ]
SNGT_QHENOMENOLOGY_184: (8)                    if self.video_codec:
SNGT_QHENOMENOLOGY_185: (12)                        command += ['-vcodec', self.video_codec]
SNGT_QHENOMENOLOGY_186: (8)                    if self.pixel_format:
SNGT_QHENOMENOLOGY_187: (12)                        command += ['-pix_fmt', self.pixel_format]
SNGT_QHENOMENOLOGY_188: (8)                    command += [self.temp_file_path]
SNGT_QHENOMENOLOGY_189: (8)                    self.writing_process = sp.Popen(command,
stdin=sp.PIPE)
SNGT_QHENOMENOLOGY_190: (8)                    if not self.quiet:
SNGT_QHENOMENOLOGY_191: (12)                        self.progress_display = ProgressDisplay(
SNGT_QHENOMENOLOGY_192: (16)                            range(self.total_frames),
SNGT_QHENOMENOLOGY_193: (16)                            leave=False,
SNGT_QHENOMENOLOGY_194: (16)                            ascii=True if platform.system() ==
'Windows' else None,
SNGT_QHENOMENOLOGY_195: (16)                            dynamic_ncols=True,
SNGT_QHENOMENOLOGY_196: (12)                        )
SNGT_QHENOMENOLOGY_197: (12)                        self.set_progress_display_description()
SNGT_QHENOMENOLOGY_198: (4)                def use_fast_encoding(self):
SNGT_QHENOMENOLOGY_199: (8)                    self.video_codec = "libx264rgb"
SNGT_QHENOMENOLOGY_200: (8)                    self.pixel_format = "rgb32"
SNGT_QHENOMENOLOGY_201: (4)                def get_insert_file_path(self, index: int) -> Path:
SNGT_QHENOMENOLOGY_202: (8)                    movie_path = Path(self.get_movie_file_path())
SNGT_QHENOMENOLOGY_203: (8)                    scene_name = movie_path.stem
SNGT_QHENOMENOLOGY_204: (8)                    insert_dir = Path(movie_path.parent, "inserts")
SNGT_QHENOMENOLOGY_205: (8)                    guarantee_existence(insert_dir)
SNGT_QHENOMENOLOGY_206: (8)                    return Path(insert_dir, f"
{scene_name}_{index}").with_suffix(self.movie_file_extension)
SNGT_QHENOMENOLOGY_207: (4)                def begin_insert(self):
SNGT_QHENOMENOLOGY_208: (8)                    self.write_to_movie = True
SNGT_QHENOMENOLOGY_209: (8)                    self.init_output_directories()
SNGT_QHENOMENOLOGY_210: (8)                    index = 0
SNGT_QHENOMENOLOGY_211: (8)                    while (insert_path :=
self.get_insert_file_path(index)).exists():
SNGT_QHENOMENOLOGY_212: (12)                        index += 1
SNGT_QHENOMENOLOGY_213: (8)                    self.inserted_file_path = insert_path
SNGT_QHENOMENOLOGY_214: (8)                    self.open_movie_pipe(self.inserted_file_path)
SNGT_QHENOMENOLOGY_215: (4)                def end_insert(self):
SNGT_QHENOMENOLOGY_216: (8)                    self.close_movie_pipe()
SNGT_QHENOMENOLOGY_217: (8)                    self.write_to_movie = False
SNGT_QHENOMENOLOGY_218: (8)
self.print_file_ready_message(self.inserted_file_path)
SNGT_QHENOMENOLOGY_219: (4)                def has_progress_display(self):
SNGT_QHENOMENOLOGY_220: (8)                    return self.progress_display is not None
SNGT_QHENOMENOLOGY_221: (4)                def set_progress_display_description(self, file: str =
"", sub_desc: str = "") -> None:
SNGT_QHENOMENOLOGY_222: (8)                    if self.progress_display is None:
SNGT_QHENOMENOLOGY_223: (12)                        return
SNGT_QHENOMENOLOGY_224: (8)                    desc_len = self.progress_description_len
SNGT_QHENOMENOLOGY_225: (8)                    if not file:
SNGT_QHENOMENOLOGY_226: (12)                        file =
os.path.split(self.get_movie_file_path())[1]
SNGT_QHENOMENOLOGY_227: (8)                    full_desc = f"{file} {sub_desc}"
SNGT_QHENOMENOLOGY_228: (8)                    if len(full_desc) > desc_len:
SNGT_QHENOMENOLOGY_229: (12)                        full_desc = full_desc[:desc_len - 3] + "..."
SNGT_QHENOMENOLOGY_230: (8)                    else:
SNGT_QHENOMENOLOGY_231: (12)                        full_desc += " " * (desc_len - len(full_desc))
SNGT_QHENOMENOLOGY_232: (8)                    self.progress_display.set_description(full_desc)
SNGT_QHENOMENOLOGY_233: (4)                def write_frame(self, camera: Camera) -> None:
SNGT_QHENOMENOLOGY_234: (8)                    if self.write_to_movie:
SNGT_QHENOMENOLOGY_235: (12)                        raw_bytes = camera.get_raw_fbo_data()
SNGT_QHENOMENOLOGY_236: (12)                        self.writing_process.stdin.write(raw_bytes)
SNGT_QHENOMENOLOGY_237: (12)                        if self.progress_display is not None:
SNGT_QHENOMENOLOGY_238: (16)                            self.progress_display.update()
```

```
SNGT_QHENOMENOLOGY_239: (4)                 def close_movie_pipe(self) -> None:
SNGT_QHENOMENOLOGY_240: (8)                     self.writing_process.stdin.close()
SNGT_QHENOMENOLOGY_241: (8)                     self.writing_process.wait()
SNGT_QHENOMENOLOGY_242: (8)                     self.writing_process.terminate()
SNGT_QHENOMENOLOGY_243: (8)                     if self.progress_display is not None:
SNGT_QHENOMENOLOGY_244: (12)                        self.progress_display.close()
SNGT_QHENOMENOLOGY_245: (8)                     if not self.ended_with_interrupt:
SNGT_QHENOMENOLOGY_246: (12)                        shutil.move(self.temp_file_path,
self.final_file_path)
SNGT_QHENOMENOLOGY_247: (8)                     else:
SNGT_QHENOMENOLOGY_248: (12)                        self.movie_file_path = self.temp_file_path
SNGT_QHENOMENOLOGY_249: (4)                 def add_sound_to_video(self) -> None:
SNGT_QHENOMENOLOGY_250: (8)                     movie_file_path = self.get_movie_file_path()
SNGT_QHENOMENOLOGY_251: (8)                     stem, ext = os.path.splitext(movie_file_path)
SNGT_QHENOMENOLOGY_252: (8)                     sound_file_path = stem + ".wav"
SNGT_QHENOMENOLOGY_253: (8)                     self.add_audio_segment(AudioSegment.silent(0))
SNGT_QHENOMENOLOGY_254: (8)                     self.audio_segment.export(
SNGT_QHENOMENOLOGY_255: (12)                        sound_file_path,
SNGT_QHENOMENOLOGY_256: (12)                        bitrate='312k',
SNGT_QHENOMENOLOGY_257: (8)                     )
SNGT_QHENOMENOLOGY_258: (8)                     temp_file_path = stem + "_temp" + ext
SNGT_QHENOMENOLOGY_259: (8)                     commands = [
SNGT_QHENOMENOLOGY_260: (12)                        self.ffmpeg_bin,
SNGT_QHENOMENOLOGY_261: (12)                        "-i", movie_file_path,
SNGT_QHENOMENOLOGY_262: (12)                        "-i", sound_file_path,
SNGT_QHENOMENOLOGY_263: (12)                        '-y',  # overwrite output file if it exists
SNGT_QHENOMENOLOGY_264: (12)                        "-c:v", "copy",
SNGT_QHENOMENOLOGY_265: (12)                        "-c:a", "aac",
SNGT_QHENOMENOLOGY_266: (12)                        "-b:a", "320k",
SNGT_QHENOMENOLOGY_267: (12)                        "-map", "0:v:0",
SNGT_QHENOMENOLOGY_268: (12)                        "-map", "1:a:0",
SNGT_QHENOMENOLOGY_269: (12)                        '-loglevel', 'error',
SNGT_QHENOMENOLOGY_270: (12)                        temp_file_path,
SNGT_QHENOMENOLOGY_271: (8)                     ]
SNGT_QHENOMENOLOGY_272: (8)                     sp.call(commands)
SNGT_QHENOMENOLOGY_273: (8)                     shutil.move(temp_file_path, movie_file_path)
SNGT_QHENOMENOLOGY_274: (8)                     os.remove(sound_file_path)
SNGT_QHENOMENOLOGY_275: (4)                 def save_final_image(self, image: Image) -> None:
SNGT_QHENOMENOLOGY_276: (8)                     file_path = self.get_image_file_path()
SNGT_QHENOMENOLOGY_277: (8)                     image.save(file_path)
SNGT_QHENOMENOLOGY_278: (8)                     self.print_file_ready_message(file_path)
SNGT_QHENOMENOLOGY_279: (4)                 def print_file_ready_message(self, file_path: str) ->
None:
SNGT_QHENOMENOLOGY_280: (8)                     if not self.quiet:
SNGT_QHENOMENOLOGY_281: (12)                        log.info(f"File ready at {file_path}")
SNGT_QHENOMENOLOGY_282: (4)                 def should_open_file(self) -> bool:
SNGT_QHENOMENOLOGY_283: (8)                     return any([
SNGT_QHENOMENOLOGY_284: (12)                        self.show_file_location_upon_completion,
SNGT_QHENOMENOLOGY_285: (12)                        self.open_file_upon_completion,
SNGT_QHENOMENOLOGY_286: (8)                     ])
SNGT_QHENOMENOLOGY_287: (4)                 def open_file(self) -> None:
SNGT_QHENOMENOLOGY_288: (8)                     if self.quiet:
SNGT_QHENOMENOLOGY_289: (12)                        curr_stdout = sys.stdout
SNGT_QHENOMENOLOGY_290: (12)                        sys.stdout = open(os.devnull, "w")
SNGT_QHENOMENOLOGY_291: (8)                     current_os = platform.system()
SNGT_QHENOMENOLOGY_292: (8)                     file_paths = []
SNGT_QHENOMENOLOGY_293: (8)                     if self.save_last_frame:
SNGT_QHENOMENOLOGY_294: (12)                        file_paths.append(self.get_image_file_path())
SNGT_QHENOMENOLOGY_295: (8)                     if self.write_to_movie:
SNGT_QHENOMENOLOGY_296: (12)                        file_paths.append(self.get_movie_file_path())
SNGT_QHENOMENOLOGY_297: (8)                     for file_path in file_paths:
SNGT_QHENOMENOLOGY_298: (12)                        if current_os == "Windows":
SNGT_QHENOMENOLOGY_299: (16)                            os.startfile(file_path)
SNGT_QHENOMENOLOGY_300: (12)                        else:
SNGT_QHENOMENOLOGY_301: (16)                            commands = []
SNGT_QHENOMENOLOGY_302: (16)                            if current_os == "Linux":
SNGT_QHENOMENOLOGY_303: (20)                                commands.append("xdg-open")
SNGT_QHENOMENOLOGY_304: (16)                            elif current_os.startswith("CYGWIN"):
SNGT_QHENOMENOLOGY_305: (20)                                commands.append("cygstart")
```

```
SNGT_QHENOMENOLOGY_306: (16)                            else:  # Assume macOS
SNGT_QHENOMENOLOGY_307: (20)                                commands.append("open")
SNGT_QHENOMENOLOGY_308: (16)                            if self.show_file_location_upon_completion:
SNGT_QHENOMENOLOGY_309: (20)                                commands.append("-R")
SNGT_QHENOMENOLOGY_310: (16)                            commands.append(file_path)
SNGT_QHENOMENOLOGY_311: (16)                            FNULL = open(os.devnull, 'w')
SNGT_QHENOMENOLOGY_312: (16)                            sp.call(commands, stdout=FNULL,
stderr=sp.STDOUT)
SNGT_QHENOMENOLOGY_313: (16)                            FNULL.close()
SNGT_QHENOMENOLOGY_314: (8)                     if self.quiet:
SNGT_QHENOMENOLOGY_315: (12)                        sys.stdout.close()
SNGT_QHENOMENOLOGY_316: (12)                        sys.stdout = curr_stdout
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 73 - cache.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import os
SNGT_QHENOMENOLOGY_3: (0)               from diskcache import Cache
SNGT_QHENOMENOLOGY_4: (0)               from contextlib import contextmanager
SNGT_QHENOMENOLOGY_5: (0)               from functools import wraps
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.utils.directories import get_cache_dir
SNGT_QHENOMENOLOGY_7: (0)               from manimlib.utils.simple_functions import hash_string
SNGT_QHENOMENOLOGY_8: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_9: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_10: (4)                  T = TypeVar('T')
SNGT_QHENOMENOLOGY_11: (0)              CACHE_SIZE = 1e9  # 1 Gig
SNGT_QHENOMENOLOGY_12: (0)              _cache = Cache(get_cache_dir(), size_limit=CACHE_SIZE)
SNGT_QHENOMENOLOGY_13: (0)              def cache_on_disk(func: Callable[..., T]) -> Callable[...,
T]:
SNGT_QHENOMENOLOGY_14: (4)                  @wraps(func)
SNGT_QHENOMENOLOGY_15: (4)                  def wrapper(*args, **kwargs):
SNGT_QHENOMENOLOGY_16: (8)                      key = hash_string(f"{func.__name__}{args}{kwargs}")
SNGT_QHENOMENOLOGY_17: (8)                      value = _cache.get(key)
SNGT_QHENOMENOLOGY_18: (8)                      if value is None:
SNGT_QHENOMENOLOGY_19: (12)                         value = func(*args, **kwargs)
SNGT_QHENOMENOLOGY_20: (12)                         _cache.set(key, value)
SNGT_QHENOMENOLOGY_21: (8)                      return value
SNGT_QHENOMENOLOGY_22: (4)                  return wrapper
SNGT_QHENOMENOLOGY_23: (0)              def clear_cache():
SNGT_QHENOMENOLOGY_24: (4)                  _cache.clear()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 74 - typing.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_2: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_3: (4)                   from typing import Union, Tuple, Annotated, Literal,
Iterable, Dict
SNGT_QHENOMENOLOGY_4: (4)                   from colour import Color
SNGT_QHENOMENOLOGY_5: (4)                   import numpy as np
SNGT_QHENOMENOLOGY_6: (4)                   import re
SNGT_QHENOMENOLOGY_7: (4)                   try:
SNGT_QHENOMENOLOGY_8: (8)                       from typing import Self
SNGT_QHENOMENOLOGY_9: (4)                   except ImportError:
SNGT_QHENOMENOLOGY_10: (8)                      from typing_extensions import Self
SNGT_QHENOMENOLOGY_11: (4)                  ManimColor = Union[str, Color, None]
SNGT_QHENOMENOLOGY_12: (4)                  RangeSpecifier = Tuple[float, float, float] |
Tuple[float, float]
SNGT_QHENOMENOLOGY_13: (4)                  Span = tuple[int, int]
SNGT_QHENOMENOLOGY_14: (4)                  SingleSelector = Union[
SNGT_QHENOMENOLOGY_15: (8)                      str,
SNGT_QHENOMENOLOGY_16: (8)                      re.Pattern,
SNGT_QHENOMENOLOGY_17: (8)                      tuple[Union[int, None], Union[int, None]],
SNGT_QHENOMENOLOGY_18: (4)                  ]
SNGT_QHENOMENOLOGY_19: (4)                  Selector = Union[SingleSelector,
Iterable[SingleSelector]]
```

```
SNGT_QHENOMENOLOGY_20: (4)                 UniformDict = Dict[str, float | bool | np.ndarray |
tuple]
SNGT_QHENOMENOLOGY_21: (4)                 FloatArray = np.ndarray[int, np.dtype[np.float64]]
SNGT_QHENOMENOLOGY_22: (4)                 Vect2 = Annotated[FloatArray, Literal[2]]
SNGT_QHENOMENOLOGY_23: (4)                 Vect3 = Annotated[FloatArray, Literal[3]]
SNGT_QHENOMENOLOGY_24: (4)                 Vect4 = Annotated[FloatArray, Literal[4]]
SNGT_QHENOMENOLOGY_25: (4)                 VectN = Annotated[FloatArray, Literal["N"]]
SNGT_QHENOMENOLOGY_26: (4)                 Matrix3x3 = Annotated[FloatArray, Literal[3, 3]]
SNGT_QHENOMENOLOGY_27: (4)                 VectArray = Annotated[FloatArray, Literal["N", 1]]
SNGT_QHENOMENOLOGY_28: (4)                 Vect2Array = Annotated[FloatArray, Literal["N", 2]]
SNGT_QHENOMENOLOGY_29: (4)                 Vect3Array = Annotated[FloatArray, Literal["N", 3]]
SNGT_QHENOMENOLOGY_30: (4)                 Vect4Array = Annotated[FloatArray, Literal["N", 4]]
SNGT_QHENOMENOLOGY_31: (4)                 VectNArray = Annotated[FloatArray, Literal["N", "M"]]
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 75 - bezier.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              from scipy import linalg
SNGT_QHENOMENOLOGY_4: (0)              from fontTools.cu2qu.cu2qu import curve_to_quadratic
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.logger import log
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.utils.simple_functions import choose
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.utils.space_ops import cross2d
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.utils.space_ops import cross
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.utils.space_ops import find_intersection
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.utils.space_ops import midpoint
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_12: (0)             from manimlib.utils.space_ops import z_to_vector
SNGT_QHENOMENOLOGY_13: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_14: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_15: (4)                 from typing import Callable, Sequence, TypeVar, Tuple
SNGT_QHENOMENOLOGY_16: (4)                 from manimlib.typing import VectN, FloatArray,
VectNArray, Vect3Array
SNGT_QHENOMENOLOGY_17: (4)                 Scalable = TypeVar("Scalable", float, FloatArray)
SNGT_QHENOMENOLOGY_18: (0)             CLOSED_THRESHOLD = 0.001
SNGT_QHENOMENOLOGY_19: (0)             def bezier(
SNGT_QHENOMENOLOGY_20: (4)                 points: Sequence[float | FloatArray] | VectNArray
SNGT_QHENOMENOLOGY_21: (0)             ) -> Callable[[float], float | FloatArray]:
SNGT_QHENOMENOLOGY_22: (4)                 if len(points) == 0:
SNGT_QHENOMENOLOGY_23: (8)                     raise Exception("bezier cannot be calld on an empty
list")
SNGT_QHENOMENOLOGY_24: (4)                 n = len(points) - 1
SNGT_QHENOMENOLOGY_25: (4)                 def result(t: float) -> float | FloatArray:
SNGT_QHENOMENOLOGY_26: (8)                     return sum(
SNGT_QHENOMENOLOGY_27: (12)                        ((1 - t)**(n - k)) * (t**k) * choose(n, k) *
point
SNGT_QHENOMENOLOGY_28: (12)                        for k, point in enumerate(points)
SNGT_QHENOMENOLOGY_29: (8)                     )
SNGT_QHENOMENOLOGY_30: (4)                 return result
SNGT_QHENOMENOLOGY_31: (0)             def partial_bezier_points(
SNGT_QHENOMENOLOGY_32: (4)                 points: Sequence[Scalable],
SNGT_QHENOMENOLOGY_33: (4)                 a: float,
SNGT_QHENOMENOLOGY_34: (4)                 b: float
SNGT_QHENOMENOLOGY_35: (0)             ) -> list[Scalable]:
SNGT_QHENOMENOLOGY_36: (4)                 """
SNGT_QHENOMENOLOGY_37: (4)                 Given an list of points which define
SNGT_QHENOMENOLOGY_38: (4)                 a bezier curve, and two numbers 0<=a<b<=1,
SNGT_QHENOMENOLOGY_39: (4)                 return an list of the same size, which
SNGT_QHENOMENOLOGY_40: (4)                 describes the portion of the original bezier
SNGT_QHENOMENOLOGY_41: (4)                 curve on the interval [a, b].
SNGT_QHENOMENOLOGY_42: (4)                 This algorithm is pretty nifty, and pretty dense.
SNGT_QHENOMENOLOGY_43: (4)                 """
SNGT_QHENOMENOLOGY_44: (4)                 if a == 1:
SNGT_QHENOMENOLOGY_45: (8)                     return [points[-1]] * len(points)
SNGT_QHENOMENOLOGY_46: (4)                 a_to_1 = [
SNGT_QHENOMENOLOGY_47: (8)                     bezier(points[i:])(a)
SNGT_QHENOMENOLOGY_48: (8)                     for i in range(len(points))
```

```
SNGT_QHENOMENOLOGY_49: (4)                 ]
SNGT_QHENOMENOLOGY_50: (4)                 end_prop = (b - a) / (1. - a)
SNGT_QHENOMENOLOGY_51: (4)                 return [
SNGT_QHENOMENOLOGY_52: (8)                     bezier(a_to_1[:i + 1])(end_prop)
SNGT_QHENOMENOLOGY_53: (8)                     for i in range(len(points))
SNGT_QHENOMENOLOGY_54: (4)                 ]
SNGT_QHENOMENOLOGY_55: (0)             def partial_quadratic_bezier_points(
SNGT_QHENOMENOLOGY_56: (4)                 points: Sequence[VectN] | VectNArray,
SNGT_QHENOMENOLOGY_57: (4)                 a: float,
SNGT_QHENOMENOLOGY_58: (4)                 b: float
SNGT_QHENOMENOLOGY_59: (0)             ) -> list[VectN]:
SNGT_QHENOMENOLOGY_60: (4)                 if a == 1:
SNGT_QHENOMENOLOGY_61: (8)                     return 3 * [points[-1]]
SNGT_QHENOMENOLOGY_62: (4)                 def curve(t):
SNGT_QHENOMENOLOGY_63: (8)                     return points[0] * (1 - t) * (1 - t) + 2 *
points[1] * t * (1 - t) + points[2] * t * t
SNGT_QHENOMENOLOGY_64: (4)                 h0 = curve(a) if a > 0 else points[0]
SNGT_QHENOMENOLOGY_65: (4)                 h2 = curve(b) if b < 1 else points[2]
SNGT_QHENOMENOLOGY_66: (4)                 h1_prime = (1 - a) * points[1] + a * points[2]
SNGT_QHENOMENOLOGY_67: (4)                 end_prop = (b - a) / (1. - a)
SNGT_QHENOMENOLOGY_68: (4)                 h1 = (1 - end_prop) * h0 + end_prop * h1_prime
SNGT_QHENOMENOLOGY_69: (4)                 return [h0, h1, h2]
SNGT_QHENOMENOLOGY_70: (0)             def interpolate(start: Scalable, end: Scalable, alpha:
float | VectN) -> Scalable:
SNGT_QHENOMENOLOGY_71: (4)                 try:
SNGT_QHENOMENOLOGY_72: (8)                     return (1 - alpha) * start + alpha * end
SNGT_QHENOMENOLOGY_73: (4)                 except TypeError:
SNGT_QHENOMENOLOGY_74: (8)                     log.debug(f"`start` parameter with type
`{type(start)}` and dtype `{start.dtype}`")
SNGT_QHENOMENOLOGY_75: (8)                     log.debug(f"`end` parameter with type `{type(end)}`
and dtype `{end.dtype}`")
SNGT_QHENOMENOLOGY_76: (8)                     log.debug(f"`alpha` parameter with value
`{alpha}`")
SNGT_QHENOMENOLOGY_77: (8)                     import sys
SNGT_QHENOMENOLOGY_78: (8)                     sys.exit(2)
SNGT_QHENOMENOLOGY_79: (0)             def outer_interpolate(
SNGT_QHENOMENOLOGY_80: (4)                 start: Scalable,
SNGT_QHENOMENOLOGY_81: (4)                 end: Scalable,
SNGT_QHENOMENOLOGY_82: (4)                 alpha: Scalable,
SNGT_QHENOMENOLOGY_83: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_84: (4)                 result = np.outer(1 - alpha, start) + np.outer(alpha,
end)
SNGT_QHENOMENOLOGY_85: (4)                 return result.reshape((*np.shape(alpha),
*np.shape(start)))
SNGT_QHENOMENOLOGY_86: (0)             def set_array_by_interpolation(
SNGT_QHENOMENOLOGY_87: (4)                 arr: np.ndarray,
SNGT_QHENOMENOLOGY_88: (4)                 arr1: np.ndarray,
SNGT_QHENOMENOLOGY_89: (4)                 arr2: np.ndarray,
SNGT_QHENOMENOLOGY_90: (4)                 alpha: float,
SNGT_QHENOMENOLOGY_91: (4)                 interp_func: Callable[[np.ndarray, np.ndarray, float],
np.ndarray] = interpolate
SNGT_QHENOMENOLOGY_92: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_93: (4)                 arr[:] = interp_func(arr1, arr2, alpha)
SNGT_QHENOMENOLOGY_94: (4)                 return arr
SNGT_QHENOMENOLOGY_95: (0)             def integer_interpolate(
SNGT_QHENOMENOLOGY_96: (4)                 start: int,
SNGT_QHENOMENOLOGY_97: (4)                 end: int,
SNGT_QHENOMENOLOGY_98: (4)                 alpha: float
SNGT_QHENOMENOLOGY_99: (0)             ) -> tuple[int, float]:
SNGT_QHENOMENOLOGY_100: (4)                """
SNGT_QHENOMENOLOGY_101: (4)                alpha is a float between 0 and 1.  This returns
SNGT_QHENOMENOLOGY_102: (4)                an integer between start and end (inclusive)
representing
SNGT_QHENOMENOLOGY_103: (4)                appropriate interpolation between them, along with a
SNGT_QHENOMENOLOGY_104: (4)                "residue" representing a new proportion between the
SNGT_QHENOMENOLOGY_105: (4)                returned integer and the next one of the
SNGT_QHENOMENOLOGY_106: (4)                list.
SNGT_QHENOMENOLOGY_107: (4)                For example, if start=0, end=10, alpha=0.46, This
SNGT_QHENOMENOLOGY_108: (4)                would return (4, 0.6).
```

```
SNGT_QHENOMENOLOGY_109: (4)                 """
SNGT_QHENOMENOLOGY_110: (4)                 if alpha >= 1:
SNGT_QHENOMENOLOGY_111: (8)                     return (end - 1, 1.0)
SNGT_QHENOMENOLOGY_112: (4)                 if alpha <= 0:
SNGT_QHENOMENOLOGY_113: (8)                     return (start, 0)
SNGT_QHENOMENOLOGY_114: (4)                 value = int(interpolate(start, end, alpha))
SNGT_QHENOMENOLOGY_115: (4)                 residue = ((end - start) * alpha) % 1
SNGT_QHENOMENOLOGY_116: (4)                 return (value, residue)
SNGT_QHENOMENOLOGY_117: (0)             def mid(start: Scalable, end: Scalable) -> Scalable:
SNGT_QHENOMENOLOGY_118: (4)                 return (start + end) / 2.0
SNGT_QHENOMENOLOGY_119: (0)             def inverse_interpolate(start: Scalable, end: Scalable,
value: Scalable) -> np.ndarray:
SNGT_QHENOMENOLOGY_120: (4)                 return np.true_divide(value - start, end - start)
SNGT_QHENOMENOLOGY_121: (0)             def match_interpolate(
SNGT_QHENOMENOLOGY_122: (4)                 new_start: Scalable,
SNGT_QHENOMENOLOGY_123: (4)                 new_end: Scalable,
SNGT_QHENOMENOLOGY_124: (4)                 old_start: Scalable,
SNGT_QHENOMENOLOGY_125: (4)                 old_end: Scalable,
SNGT_QHENOMENOLOGY_126: (4)                 old_value: Scalable
SNGT_QHENOMENOLOGY_127: (0)             ) -> Scalable:
SNGT_QHENOMENOLOGY_128: (4)                 return interpolate(
SNGT_QHENOMENOLOGY_129: (8)                     new_start, new_end,
SNGT_QHENOMENOLOGY_130: (8)                     inverse_interpolate(old_start, old_end, old_value)
SNGT_QHENOMENOLOGY_131: (4)                 )
SNGT_QHENOMENOLOGY_132: (0)             def quadratic_bezier_points_for_arc(angle: float,
n_components: int = 8):
SNGT_QHENOMENOLOGY_133: (4)                 n_points = 2 * n_components + 1
SNGT_QHENOMENOLOGY_134: (4)                 angles = np.linspace(0, angle, n_points)
SNGT_QHENOMENOLOGY_135: (4)                 points = np.array([np.cos(angles), np.sin(angles),
np.zeros(n_points)]).T
SNGT_QHENOMENOLOGY_136: (4)                 theta = angle / n_components
SNGT_QHENOMENOLOGY_137: (4)                 points[1::2] /= np.cos(theta / 2)
SNGT_QHENOMENOLOGY_138: (4)                 return points
SNGT_QHENOMENOLOGY_139: (0)             def approx_smooth_quadratic_bezier_handles(
SNGT_QHENOMENOLOGY_140: (4)                 points: FloatArray
SNGT_QHENOMENOLOGY_141: (0)             ) -> FloatArray:
SNGT_QHENOMENOLOGY_142: (4)                 """
SNGT_QHENOMENOLOGY_143: (4)                 Figuring out which bezier curves most smoothly connect
a sequence of points.
SNGT_QHENOMENOLOGY_144: (4)                 Given three successive points, P0, P1 and P2, you can
compute that by defining
SNGT_QHENOMENOLOGY_145: (4)                 h = (1/4) P0 + P1 - (1/4)P2, the bezier curve defined
by (P0, h, P1) will pass
SNGT_QHENOMENOLOGY_146: (4)                 through the point P2.
SNGT_QHENOMENOLOGY_147: (4)                 So for a given set of four successive points, P0, P1,
P2, P3, if we want to add
SNGT_QHENOMENOLOGY_148: (4)                 a handle point h between P1 and P2 so that the
quadratic bezier (P1, h, P2) is
SNGT_QHENOMENOLOGY_149: (4)                 part of a smooth curve passing through all four points,
we calculate one solution
SNGT_QHENOMENOLOGY_150: (4)                 for h that would produce a parbola passing through P3,
call it smooth_to_right, and
SNGT_QHENOMENOLOGY_151: (4)                 another that would produce a parabola passing through
P0, call it smooth_to_left,
SNGT_QHENOMENOLOGY_152: (4)                 and use the midpoint between the two.
SNGT_QHENOMENOLOGY_153: (4)                 """
SNGT_QHENOMENOLOGY_154: (4)                 if len(points) == 1:
SNGT_QHENOMENOLOGY_155: (8)                     return points[0]
SNGT_QHENOMENOLOGY_156: (4)                 elif len(points) == 2:
SNGT_QHENOMENOLOGY_157: (8)                     return midpoint(*points)
SNGT_QHENOMENOLOGY_158: (4)                 smooth_to_right, smooth_to_left = [
SNGT_QHENOMENOLOGY_159: (8)                     0.25 * ps[0:-2] + ps[1:-1] - 0.25 * ps[2:]
SNGT_QHENOMENOLOGY_160: (8)                     for ps in (points, points[::-1])
SNGT_QHENOMENOLOGY_161: (4)                 ]
SNGT_QHENOMENOLOGY_162: (4)                 if np.isclose(points[0], points[-1]).all():
SNGT_QHENOMENOLOGY_163: (8)                     last_str = 0.25 * points[-2] + points[-1] - 0.25 *
points[1]
SNGT_QHENOMENOLOGY_164: (8)                     last_stl = 0.25 * points[1] + points[0] - 0.25 *
points[-2]
```

```
SNGT_QHENOMENOLOGY_165: (4)                 else:
SNGT_QHENOMENOLOGY_166: (8)                     last_str = smooth_to_left[0]
SNGT_QHENOMENOLOGY_167: (8)                     last_stl = smooth_to_right[0]
SNGT_QHENOMENOLOGY_168: (4)                 handles = 0.5 * np.vstack([smooth_to_right,
[last_str]])
SNGT_QHENOMENOLOGY_169: (4)                 handles += 0.5 * np.vstack([last_stl,
smooth_to_left[::-1]])
SNGT_QHENOMENOLOGY_170: (4)                 return handles
SNGT_QHENOMENOLOGY_171: (0)             def smooth_quadratic_path(anchors: Vect3Array) ->
Vect3Array:
SNGT_QHENOMENOLOGY_172: (4)                 """
SNGT_QHENOMENOLOGY_173: (4)                 Returns a path defining a smooth quadratic bezier
spline
SNGT_QHENOMENOLOGY_174: (4)                 through anchors.
SNGT_QHENOMENOLOGY_175: (4)                 """
SNGT_QHENOMENOLOGY_176: (4)                 if len(anchors) < 2:
SNGT_QHENOMENOLOGY_177: (8)                     return anchors
SNGT_QHENOMENOLOGY_178: (4)                 elif len(anchors) == 2:
SNGT_QHENOMENOLOGY_179: (8)                     return np.array([anchors[0], anchors.mean(0),
anchors[1]])
SNGT_QHENOMENOLOGY_180: (4)                 is_flat = (anchors[:, 2] == 0).all()
SNGT_QHENOMENOLOGY_181: (4)                 if not is_flat:
SNGT_QHENOMENOLOGY_182: (8)                     normal = cross(anchors[2] - anchors[1], anchors[1]
- anchors[0])
SNGT_QHENOMENOLOGY_183: (8)                     rot = z_to_vector(normal)
SNGT_QHENOMENOLOGY_184: (8)                     anchors = np.dot(anchors, rot)
SNGT_QHENOMENOLOGY_185: (8)                     shift = anchors[0, 2]
SNGT_QHENOMENOLOGY_186: (8)                     anchors[:, 2] -= shift
SNGT_QHENOMENOLOGY_187: (4)                 h1s, h2s =
get_smooth_cubic_bezier_handle_points(anchors)
SNGT_QHENOMENOLOGY_188: (4)                 quads = [anchors[0, :2]]
SNGT_QHENOMENOLOGY_189: (4)                 for cub_bs in zip(anchors[:-1], h1s, h2s, anchors[1:]):
SNGT_QHENOMENOLOGY_190: (8)                     new_quads = curve_to_quadratic(
SNGT_QHENOMENOLOGY_191: (12)                        [b[:2] for b in cub_bs],
SNGT_QHENOMENOLOGY_192: (12)                        max_err=0.1 * get_norm(cub_bs[3] - cub_bs[0])
SNGT_QHENOMENOLOGY_193: (8)                     )
SNGT_QHENOMENOLOGY_194: (8)                     if new_quads is None or len(new_quads) % 2 == 0:
SNGT_QHENOMENOLOGY_195: (12)                        new_quads =
get_quadratic_approximation_of_cubic(*cub_bs)[:, :2]
SNGT_QHENOMENOLOGY_196: (8)                     quads.extend(new_quads[1:])
SNGT_QHENOMENOLOGY_197: (4)                 new_path = np.zeros((len(quads), 3))
SNGT_QHENOMENOLOGY_198: (4)                 new_path[:, :2] = quads
SNGT_QHENOMENOLOGY_199: (4)                 if not is_flat:
SNGT_QHENOMENOLOGY_200: (8)                     new_path[:, 2] += shift
SNGT_QHENOMENOLOGY_201: (8)                     new_path = np.dot(new_path, rot.T)
SNGT_QHENOMENOLOGY_202: (4)                 return new_path
SNGT_QHENOMENOLOGY_203: (0)             def get_smooth_cubic_bezier_handle_points(
SNGT_QHENOMENOLOGY_204: (4)                 points: Sequence[VectN] | VectNArray
SNGT_QHENOMENOLOGY_205: (0)             ) -> tuple[FloatArray, FloatArray]:
SNGT_QHENOMENOLOGY_206: (4)                 points = np.array(points)
SNGT_QHENOMENOLOGY_207: (4)                 num_handles = len(points) - 1
SNGT_QHENOMENOLOGY_208: (4)                 dim = points.shape[1]
SNGT_QHENOMENOLOGY_209: (4)                 if num_handles < 1:
SNGT_QHENOMENOLOGY_210: (8)                     return np.zeros((0, dim)), np.zeros((0, dim))
SNGT_QHENOMENOLOGY_211: (4)                 l, u = 2, 1
SNGT_QHENOMENOLOGY_212: (4)                 diag = np.zeros((l + u + 1, 2 * num_handles))
SNGT_QHENOMENOLOGY_213: (4)                 diag[0, 1::2] = -1
SNGT_QHENOMENOLOGY_214: (4)                 diag[0, 2::2] = 1
SNGT_QHENOMENOLOGY_215: (4)                 diag[1, 0::2] = 2
SNGT_QHENOMENOLOGY_216: (4)                 diag[1, 1::2] = 1
SNGT_QHENOMENOLOGY_217: (4)                 diag[2, 1:-2:2] = -2
SNGT_QHENOMENOLOGY_218: (4)                 diag[3, 0:-3:2] = 1
SNGT_QHENOMENOLOGY_219: (4)                 diag[2, -2] = -1
SNGT_QHENOMENOLOGY_220: (4)                 diag[1, -1] = 2
SNGT_QHENOMENOLOGY_221: (4)                 b = np.zeros((2 * num_handles, dim))
SNGT_QHENOMENOLOGY_222: (4)                 b[1::2] = 2 * points[1:]
SNGT_QHENOMENOLOGY_223: (4)                 b[0] = points[0]
SNGT_QHENOMENOLOGY_224: (4)                 b[-1] = points[-1]
SNGT_QHENOMENOLOGY_225: (4)                 def solve_func(b):
```

```
SNGT_QHENOMENOLOGY_226: (8)                     return linalg.solve_banded((l, u), diag, b)
SNGT_QHENOMENOLOGY_227: (4)                 use_closed_solve_function = is_closed(points)
SNGT_QHENOMENOLOGY_228: (4)                 if use_closed_solve_function:
SNGT_QHENOMENOLOGY_229: (8)                     matrix = diag_to_matrix((l, u), diag)
SNGT_QHENOMENOLOGY_230: (8)                     matrix[-1, [0, 1, -2, -1]] = [2, -1, 1, -2]
SNGT_QHENOMENOLOGY_231: (8)                     matrix[0, :] = np.zeros(matrix.shape[1])
SNGT_QHENOMENOLOGY_232: (8)                     matrix[0, [0, -1]] = [1, 1]
SNGT_QHENOMENOLOGY_233: (8)                     b[0] = 2 * points[0]
SNGT_QHENOMENOLOGY_234: (8)                     b[-1] = np.zeros(dim)
SNGT_QHENOMENOLOGY_235: (8)                     def closed_curve_solve_func(b):
SNGT_QHENOMENOLOGY_236: (12)                        return linalg.solve(matrix, b)
SNGT_QHENOMENOLOGY_237: (4)                 handle_pairs = np.zeros((2 * num_handles, dim))
SNGT_QHENOMENOLOGY_238: (4)                 for i in range(dim):
SNGT_QHENOMENOLOGY_239: (8)                     if use_closed_solve_function:
SNGT_QHENOMENOLOGY_240: (12)                        handle_pairs[:, i] =
closed_curve_solve_func(b[:, i])
SNGT_QHENOMENOLOGY_241: (8)                     else:
SNGT_QHENOMENOLOGY_242: (12)                        handle_pairs[:, i] = solve_func(b[:, i])
SNGT_QHENOMENOLOGY_243: (4)                 return handle_pairs[0::2], handle_pairs[1::2]
SNGT_QHENOMENOLOGY_244: (0)             def diag_to_matrix(
SNGT_QHENOMENOLOGY_245: (4)                 l_and_u: tuple[int, int],
SNGT_QHENOMENOLOGY_246: (4)                 diag: np.ndarray
SNGT_QHENOMENOLOGY_247: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_248: (4)                 """
SNGT_QHENOMENOLOGY_249: (4)                 Converts array whose rows represent diagonal
SNGT_QHENOMENOLOGY_250: (4)                 entries of a matrix into the matrix itself.
SNGT_QHENOMENOLOGY_251: (4)                 See scipy.linalg.solve_banded
SNGT_QHENOMENOLOGY_252: (4)                 """
SNGT_QHENOMENOLOGY_253: (4)                 l, u = l_and_u
SNGT_QHENOMENOLOGY_254: (4)                 dim = diag.shape[1]
SNGT_QHENOMENOLOGY_255: (4)                 matrix = np.zeros((dim, dim))
SNGT_QHENOMENOLOGY_256: (4)                 for i in range(l + u + 1):
SNGT_QHENOMENOLOGY_257: (8)                     np.fill_diagonal(
SNGT_QHENOMENOLOGY_258: (12)                        matrix[max(0, i - u):, max(0, u - i):],
SNGT_QHENOMENOLOGY_259: (12)                        diag[i, max(0, u - i):]
SNGT_QHENOMENOLOGY_260: (8)                     )
SNGT_QHENOMENOLOGY_261: (4)                 return matrix
SNGT_QHENOMENOLOGY_262: (0)             def is_closed(points: FloatArray) -> bool:
SNGT_QHENOMENOLOGY_263: (4)                 return np.allclose(points[0], points[-1])
SNGT_QHENOMENOLOGY_264: (0)             def get_quadratic_approximation_of_cubic(
SNGT_QHENOMENOLOGY_265: (4)                 a0: FloatArray,
SNGT_QHENOMENOLOGY_266: (4)                 h0: FloatArray,
SNGT_QHENOMENOLOGY_267: (4)                 h1: FloatArray,
SNGT_QHENOMENOLOGY_268: (4)                 a1: FloatArray
SNGT_QHENOMENOLOGY_269: (0)             ) -> FloatArray:
SNGT_QHENOMENOLOGY_270: (4)                 a0 = np.array(a0, ndmin=2)
SNGT_QHENOMENOLOGY_271: (4)                 h0 = np.array(h0, ndmin=2)
SNGT_QHENOMENOLOGY_272: (4)                 h1 = np.array(h1, ndmin=2)
SNGT_QHENOMENOLOGY_273: (4)                 a1 = np.array(a1, ndmin=2)
SNGT_QHENOMENOLOGY_274: (4)                 T0 = h0 - a0
SNGT_QHENOMENOLOGY_275: (4)                 T1 = a1 - h1
SNGT_QHENOMENOLOGY_276: (4)                 has_infl = np.ones(len(a0), dtype=bool)
SNGT_QHENOMENOLOGY_277: (4)                 p = h0 - a0
SNGT_QHENOMENOLOGY_278: (4)                 q = h1 - 2 * h0 + a0
SNGT_QHENOMENOLOGY_279: (4)                 r = a1 - 3 * h1 + 3 * h0 - a0
SNGT_QHENOMENOLOGY_280: (4)                 a = cross2d(q, r)
SNGT_QHENOMENOLOGY_281: (4)                 b = cross2d(p, r)
SNGT_QHENOMENOLOGY_282: (4)                 c = cross2d(p, q)
SNGT_QHENOMENOLOGY_283: (4)                 disc = b * b - 4 * a * c
SNGT_QHENOMENOLOGY_284: (4)                 has_infl &= (disc > 0)
SNGT_QHENOMENOLOGY_285: (4)                 sqrt_disc = np.sqrt(np.abs(disc))
SNGT_QHENOMENOLOGY_286: (4)                 settings = np.seterr(all='ignore')
SNGT_QHENOMENOLOGY_287: (4)                 ti_bounds = []
SNGT_QHENOMENOLOGY_288: (4)                 for sgn in [-1, +1]:
SNGT_QHENOMENOLOGY_289: (8)                     ti = (-b + sgn * sqrt_disc) / (2 * a)
SNGT_QHENOMENOLOGY_290: (8)                     ti[a == 0] = (-c / b)[a == 0]
SNGT_QHENOMENOLOGY_291: (8)                     ti[(a == 0) & (b == 0)] = 0
SNGT_QHENOMENOLOGY_292: (8)                     ti_bounds.append(ti)
SNGT_QHENOMENOLOGY_293: (4)                 ti_min, ti_max = ti_bounds
```

```
SNGT_QHENOMENOLOGY_294: (4)                 np.seterr(**settings)
SNGT_QHENOMENOLOGY_295: (4)                 ti_min_in_range = has_infl & (0 < ti_min) & (ti_min <
1)
SNGT_QHENOMENOLOGY_296: (4)                 ti_max_in_range = has_infl & (0 < ti_max) & (ti_max <
1)
SNGT_QHENOMENOLOGY_297: (4)                 t_mid = 0.5 * np.ones(len(a0))
SNGT_QHENOMENOLOGY_298: (4)                 t_mid[ti_min_in_range] = ti_min[ti_min_in_range]
SNGT_QHENOMENOLOGY_299: (4)                 t_mid[ti_max_in_range] = ti_max[ti_max_in_range]
SNGT_QHENOMENOLOGY_300: (4)                 m, n = a0.shape
SNGT_QHENOMENOLOGY_301: (4)                 t_mid = t_mid.repeat(n).reshape((m, n))
SNGT_QHENOMENOLOGY_302: (4)                 mid = bezier([a0, h0, h1, a1])(t_mid)
SNGT_QHENOMENOLOGY_303: (4)                 Tm = bezier([h0 - a0, h1 - h0, a1 - h1])(t_mid)
SNGT_QHENOMENOLOGY_304: (4)                 i0 = find_intersection(a0, T0, mid, Tm)
SNGT_QHENOMENOLOGY_305: (4)                 i1 = find_intersection(a1, T1, mid, Tm)
SNGT_QHENOMENOLOGY_306: (4)                 m, n = np.shape(a0)
SNGT_QHENOMENOLOGY_307: (4)                 result = np.zeros((5 * m, n))
SNGT_QHENOMENOLOGY_308: (4)                 result[0::5] = a0
SNGT_QHENOMENOLOGY_309: (4)                 result[1::5] = i0
SNGT_QHENOMENOLOGY_310: (4)                 result[2::5] = mid
SNGT_QHENOMENOLOGY_311: (4)                 result[3::5] = i1
SNGT_QHENOMENOLOGY_312: (4)                 result[4::5] = a1
SNGT_QHENOMENOLOGY_313: (4)                 return result
SNGT_QHENOMENOLOGY_314: (0)             def get_smooth_quadratic_bezier_path_through(
SNGT_QHENOMENOLOGY_315: (4)                 points: Sequence[VectN]
SNGT_QHENOMENOLOGY_316: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_317: (4)                 h0, h1 = get_smooth_cubic_bezier_handle_points(points)
SNGT_QHENOMENOLOGY_318: (4)                 a0 = points[:-1]
SNGT_QHENOMENOLOGY_319: (4)                 a1 = points[1:]
SNGT_QHENOMENOLOGY_320: (4)                 return get_quadratic_approximation_of_cubic(a0, h0, h1,
a1)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 76 - __init__.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 77 - tex.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import re
SNGT_QHENOMENOLOGY_3: (0)               from functools import lru_cache
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.utils.tex_to_symbol_count import
TEX_TO_SYMBOL_COUNT
SNGT_QHENOMENOLOGY_5: (0)               @lru_cache
SNGT_QHENOMENOLOGY_6: (0)               def num_tex_symbols(tex: str) -> int:
SNGT_QHENOMENOLOGY_7: (4)                   tex = remove_tex_environments(tex)
SNGT_QHENOMENOLOGY_8: (4)                   commands_pattern = r"""
SNGT_QHENOMENOLOGY_9: (8)                       (?P<sqrt>\\sqrt\[[0-9]+\])|    # Special sqrt with
number
SNGT_QHENOMENOLOGY_10: (8)                      (?P<cmd>\\[a-zA-Z!,-/:;<>]+)   # Regular commands
SNGT_QHENOMENOLOGY_11: (4)                  """
SNGT_QHENOMENOLOGY_12: (4)                  total = 0
SNGT_QHENOMENOLOGY_13: (4)                  pos = 0
SNGT_QHENOMENOLOGY_14: (4)                  for match in re.finditer(commands_pattern, tex,
re.VERBOSE):
SNGT_QHENOMENOLOGY_15: (8)                      total += sum(1 for c in tex[pos:match.start()] if c
not in "^{} \n\t_$\\&")
SNGT_QHENOMENOLOGY_16: (8)                      if match.group("sqrt"):
SNGT_QHENOMENOLOGY_17: (12)                         total += len(match.group()) - 5
SNGT_QHENOMENOLOGY_18: (8)                      else:
SNGT_QHENOMENOLOGY_19: (12)                         total += TEX_TO_SYMBOL_COUNT.get(match.group(),
1)
SNGT_QHENOMENOLOGY_20: (8)                      pos = match.end()
SNGT_QHENOMENOLOGY_21: (4)                  total += sum(1 for c in tex[pos:] if c not in "^{}
\n\t_$\\&")
SNGT_QHENOMENOLOGY_22: (4)                  return total
```

```
SNGT_QHENOMENOLOGY_23: (0)             def remove_tex_environments(tex: str) -> str:
SNGT_QHENOMENOLOGY_24: (4)                 tex = re.sub(r"\\phantom\{[^}]*\}", "", tex)
SNGT_QHENOMENOLOGY_25: (4)                 tex = re.sub(r"\\(begin|end)(\{\w+\})?(\{\w+\})?(\
[\w+\])?", "", tex)
SNGT_QHENOMENOLOGY_26: (4)                 return tex
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 78 - setup.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              import setuptools
SNGT_QHENOMENOLOGY_2: (0)              setuptools.setup()
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 79 - color.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from colour import Color
SNGT_QHENOMENOLOGY_3: (0)              from colour import hex2rgb
SNGT_QHENOMENOLOGY_4: (0)              from colour import rgb2hex
SNGT_QHENOMENOLOGY_5: (0)              import numpy as np
SNGT_QHENOMENOLOGY_6: (0)              import random
SNGT_QHENOMENOLOGY_7: (0)              from matplotlib import pyplot
SNGT_QHENOMENOLOGY_8: (0)              from manimlib.constants import COLORMAP_3B1B
SNGT_QHENOMENOLOGY_9: (0)              from manimlib.constants import WHITE
SNGT_QHENOMENOLOGY_10: (0)             from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_11: (0)             from manimlib.utils.iterables import
resize_with_interpolation
SNGT_QHENOMENOLOGY_12: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_13: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_14: (4)                 from typing import Iterable, Sequence, Callable
SNGT_QHENOMENOLOGY_15: (4)                 from manimlib.typing import ManimColor, Vect3, Vect4,
Vect3Array, Vect4Array, NDArray
SNGT_QHENOMENOLOGY_16: (0)             def color_to_rgb(color: ManimColor) -> Vect3:
SNGT_QHENOMENOLOGY_17: (4)                 if isinstance(color, str):
SNGT_QHENOMENOLOGY_18: (8)                     return hex_to_rgb(color)
SNGT_QHENOMENOLOGY_19: (4)                 elif isinstance(color, Color):
SNGT_QHENOMENOLOGY_20: (8)                     return np.array(color.get_rgb())
SNGT_QHENOMENOLOGY_21: (4)                 else:
SNGT_QHENOMENOLOGY_22: (8)                     raise Exception("Invalid color type")
SNGT_QHENOMENOLOGY_23: (0)             def color_to_rgba(color: ManimColor, alpha: float = 1.0) ->
Vect4:
SNGT_QHENOMENOLOGY_24: (4)                 return np.array([*color_to_rgb(color), alpha])
SNGT_QHENOMENOLOGY_25: (0)             def rgb_to_color(rgb: Vect3 | Sequence[float]) -> Color:
SNGT_QHENOMENOLOGY_26: (4)                 try:
SNGT_QHENOMENOLOGY_27: (8)                     return Color(rgb=tuple(rgb))
SNGT_QHENOMENOLOGY_28: (4)                 except ValueError:
SNGT_QHENOMENOLOGY_29: (8)                     return Color(WHITE)
SNGT_QHENOMENOLOGY_30: (0)             def rgba_to_color(rgba: Vect4) -> Color:
SNGT_QHENOMENOLOGY_31: (4)                 return rgb_to_color(rgba[:3])
SNGT_QHENOMENOLOGY_32: (0)             def rgb_to_hex(rgb: Vect3 | Sequence[float]) -> str:
SNGT_QHENOMENOLOGY_33: (4)                 return rgb2hex(rgb, force_long=True).upper()
SNGT_QHENOMENOLOGY_34: (0)             def hex_to_rgb(hex_code: str) -> Vect3:
SNGT_QHENOMENOLOGY_35: (4)                 return np.array(hex2rgb(hex_code))
SNGT_QHENOMENOLOGY_36: (0)             def invert_color(color: ManimColor) -> Color:
SNGT_QHENOMENOLOGY_37: (4)                 return rgb_to_color(1.0 - color_to_rgb(color))
SNGT_QHENOMENOLOGY_38: (0)             def color_to_int_rgb(color: ManimColor) -> np.ndarray[int,
np.dtype[np.uint8]]:
SNGT_QHENOMENOLOGY_39: (4)                 return (255 * color_to_rgb(color)).astype('uint8')
SNGT_QHENOMENOLOGY_40: (0)             def color_to_int_rgba(color: ManimColor, opacity: float =
1.0) -> np.ndarray[int, np.dtype[np.uint8]]:
SNGT_QHENOMENOLOGY_41: (4)                 alpha = int(255 * opacity)
SNGT_QHENOMENOLOGY_42: (4)                 return np.array([*color_to_int_rgb(color), alpha],
dtype=np.uint8)
SNGT_QHENOMENOLOGY_43: (0)             def color_to_hex(color: ManimColor) -> str:
SNGT_QHENOMENOLOGY_44: (4)                 return Color(color).get_hex_l().upper()
SNGT_QHENOMENOLOGY_45: (0)             def hex_to_int(rgb_hex: str) -> int:
SNGT_QHENOMENOLOGY_46: (4)                 return int(rgb_hex[1:], 16)
```

```
SNGT_QHENOMENOLOGY_47: (0)             def int_to_hex(rgb_int: int) -> str:
SNGT_QHENOMENOLOGY_48: (4)                 return f"#{rgb_int:06x}".upper()
SNGT_QHENOMENOLOGY_49: (0)             def color_gradient(
SNGT_QHENOMENOLOGY_50: (4)                 reference_colors: Iterable[ManimColor],
SNGT_QHENOMENOLOGY_51: (4)                 length_of_output: int
SNGT_QHENOMENOLOGY_52: (0)             ) -> list[Color]:
SNGT_QHENOMENOLOGY_53: (4)                 if length_of_output == 0:
SNGT_QHENOMENOLOGY_54: (8)                     return []
SNGT_QHENOMENOLOGY_55: (4)                 rgbs = list(map(color_to_rgb, reference_colors))
SNGT_QHENOMENOLOGY_56: (4)                 alphas = np.linspace(0, (len(rgbs) - 1),
length_of_output)
SNGT_QHENOMENOLOGY_57: (4)                 floors = alphas.astype('int')
SNGT_QHENOMENOLOGY_58: (4)                 alphas_mod1 = alphas % 1
SNGT_QHENOMENOLOGY_59: (4)                 alphas_mod1[-1] = 1
SNGT_QHENOMENOLOGY_60: (4)                 floors[-1] = len(rgbs) - 2
SNGT_QHENOMENOLOGY_61: (4)                 return [
SNGT_QHENOMENOLOGY_62: (8)                     rgb_to_color(np.sqrt(interpolate(rgbs[i]**2, rgbs[i
+ 1]**2, alpha)))
SNGT_QHENOMENOLOGY_63: (8)                     for i, alpha in zip(floors, alphas_mod1)
SNGT_QHENOMENOLOGY_64: (4)                 ]
SNGT_QHENOMENOLOGY_65: (0)             def interpolate_color(
SNGT_QHENOMENOLOGY_66: (4)                 color1: ManimColor,
SNGT_QHENOMENOLOGY_67: (4)                 color2: ManimColor,
SNGT_QHENOMENOLOGY_68: (4)                 alpha: float
SNGT_QHENOMENOLOGY_69: (0)             ) -> Color:
SNGT_QHENOMENOLOGY_70: (4)                 rgb = np.sqrt(interpolate(color_to_rgb(color1)**2,
color_to_rgb(color2)**2, alpha))
SNGT_QHENOMENOLOGY_71: (4)                 return rgb_to_color(rgb)
SNGT_QHENOMENOLOGY_72: (0)             def interpolate_color_by_hsl(
SNGT_QHENOMENOLOGY_73: (4)                 color1: ManimColor,
SNGT_QHENOMENOLOGY_74: (4)                 color2: ManimColor,
SNGT_QHENOMENOLOGY_75: (4)                 alpha: float
SNGT_QHENOMENOLOGY_76: (0)             ) -> Color:
SNGT_QHENOMENOLOGY_77: (4)                 hsl1 = np.array(Color(color1).get_hsl())
SNGT_QHENOMENOLOGY_78: (4)                 hsl2 = np.array(Color(color2).get_hsl())
SNGT_QHENOMENOLOGY_79: (4)                 return Color(hsl=interpolate(hsl1, hsl2, alpha))
SNGT_QHENOMENOLOGY_80: (0)             def average_color(*colors: ManimColor) -> Color:
SNGT_QHENOMENOLOGY_81: (4)                 rgbs = np.array(list(map(color_to_rgb, colors)))
SNGT_QHENOMENOLOGY_82: (4)                 return rgb_to_color(np.sqrt((rgbs**2).mean(0)))
SNGT_QHENOMENOLOGY_83: (0)             def random_color() -> Color:
SNGT_QHENOMENOLOGY_84: (4)                 return Color(rgb=tuple(np.random.random(3)))
SNGT_QHENOMENOLOGY_85: (0)             def random_bright_color(
SNGT_QHENOMENOLOGY_86: (4)                 hue_range: tuple[float, float] = (0.0, 1.0),
SNGT_QHENOMENOLOGY_87: (4)                 saturation_range: tuple[float, float] = (0.5, 0.8),
SNGT_QHENOMENOLOGY_88: (4)                 luminance_range: tuple[float, float] = (0.5, 1.0),
SNGT_QHENOMENOLOGY_89: (0)             ) -> Color:
SNGT_QHENOMENOLOGY_90: (4)                 return Color(hsl=(
SNGT_QHENOMENOLOGY_91: (8)                     interpolate(*hue_range, random.random()),
SNGT_QHENOMENOLOGY_92: (8)                     interpolate(*saturation_range, random.random()),
SNGT_QHENOMENOLOGY_93: (8)                     interpolate(*luminance_range, random.random()),
SNGT_QHENOMENOLOGY_94: (4)                 ))
SNGT_QHENOMENOLOGY_95: (0)             def get_colormap_from_colors(colors: Iterable[ManimColor])
-> Callable[[Sequence[float]], Vect4Array]:
SNGT_QHENOMENOLOGY_96: (4)                 """
SNGT_QHENOMENOLOGY_97: (4)                 Returns a funciton which takes in values between 0 and
1, and returns
SNGT_QHENOMENOLOGY_98: (4)                 a corresponding list of rgba values
SNGT_QHENOMENOLOGY_99: (4)                 """
SNGT_QHENOMENOLOGY_100: (4)                rgbas = np.array([color_to_rgba(color) for color in
colors])
SNGT_QHENOMENOLOGY_101: (4)                def func(values):
SNGT_QHENOMENOLOGY_102: (8)                    alphas = np.clip(values, 0, 1)
SNGT_QHENOMENOLOGY_103: (8)                    scaled_alphas = alphas * (len(rgbas) - 1)
SNGT_QHENOMENOLOGY_104: (8)                    indices = scaled_alphas.astype(int)
SNGT_QHENOMENOLOGY_105: (8)                    next_indices = np.clip(indices + 1, 0, len(rgbas) -
1)
SNGT_QHENOMENOLOGY_106: (8)                    inter_alphas = scaled_alphas % 1
SNGT_QHENOMENOLOGY_107: (8)                    inter_alphas =
inter_alphas.repeat(4).reshape((len(indices), 4))
```

```
SNGT_QHENOMENOLOGY_108: (8)                     result = interpolate(rgbas[indices],
rgbas[next_indices], inter_alphas)
SNGT_QHENOMENOLOGY_109: (8)                     return result
SNGT_QHENOMENOLOGY_110: (4)                 return func
SNGT_QHENOMENOLOGY_111: (0)             def get_color_map(map_name: str) ->
Callable[[Sequence[float]], Vect4Array]:
SNGT_QHENOMENOLOGY_112: (4)                 if map_name == "3b1b_colormap":
SNGT_QHENOMENOLOGY_113: (8)                     return get_colormap_from_colors(COLORMAP_3B1B)
SNGT_QHENOMENOLOGY_114: (4)                 return pyplot.get_cmap(map_name)
SNGT_QHENOMENOLOGY_115: (0)             def get_colormap_list(
SNGT_QHENOMENOLOGY_116: (4)                 map_name: str = "viridis",
SNGT_QHENOMENOLOGY_117: (4)                 n_colors: int = 9
SNGT_QHENOMENOLOGY_118: (0)             ) -> Vect3Array:
SNGT_QHENOMENOLOGY_119: (4)                 """
SNGT_QHENOMENOLOGY_120: (4)                 Options for map_name:
SNGT_QHENOMENOLOGY_121: (4)                 3b1b_colormap
SNGT_QHENOMENOLOGY_122: (4)                 magma
SNGT_QHENOMENOLOGY_123: (4)                 inferno
SNGT_QHENOMENOLOGY_124: (4)                 plasma
SNGT_QHENOMENOLOGY_125: (4)                 viridis
SNGT_QHENOMENOLOGY_126: (4)                 cividis
SNGT_QHENOMENOLOGY_127: (4)                 twilight
SNGT_QHENOMENOLOGY_128: (4)                 twilight_shifted
SNGT_QHENOMENOLOGY_129: (4)                 turbo
SNGT_QHENOMENOLOGY_130: (4)                 """
SNGT_QHENOMENOLOGY_131: (4)                 from matplotlib.cm import cmaps_listed
SNGT_QHENOMENOLOGY_132: (4)                 if map_name == "3b1b_colormap":
SNGT_QHENOMENOLOGY_133: (8)                     rgbs = np.array([color_to_rgb(color) for color in
COLORMAP_3B1B])
SNGT_QHENOMENOLOGY_134: (4)                 else:
SNGT_QHENOMENOLOGY_135: (8)                     rgbs = cmaps_listed[map_name].colors  # Make more
general?
SNGT_QHENOMENOLOGY_136: (4)                 return resize_with_interpolation(np.array(rgbs),
n_colors)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 80 - debug.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.constants import BLACK
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.logger import log
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.mobject.numbers import Integer
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.mobject.types.vectorized_mobject import
VGroup
SNGT_QHENOMENOLOGY_6: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_7: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_8: (4)                   from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_9: (0)               def print_family(mobject: Mobject, n_tabs: int = 0) ->
None:
SNGT_QHENOMENOLOGY_10: (4)                  """For debugging purposes"""
SNGT_QHENOMENOLOGY_11: (4)                  log.debug("\t" * n_tabs + str(mobject) + " " +
str(id(mobject)))
SNGT_QHENOMENOLOGY_12: (4)                  for submob in mobject.submobjects:
SNGT_QHENOMENOLOGY_13: (8)                      print_family(submob, n_tabs + 1)
SNGT_QHENOMENOLOGY_14: (0)              def index_labels(
SNGT_QHENOMENOLOGY_15: (4)                  mobject: Mobject,
SNGT_QHENOMENOLOGY_16: (4)                  label_height: float = 0.15
SNGT_QHENOMENOLOGY_17: (0)              ) -> VGroup:
SNGT_QHENOMENOLOGY_18: (4)                  labels = VGroup()
SNGT_QHENOMENOLOGY_19: (4)                  for n, submob in enumerate(mobject):
SNGT_QHENOMENOLOGY_20: (8)                      label = Integer(n)
SNGT_QHENOMENOLOGY_21: (8)                      label.set_height(label_height)
SNGT_QHENOMENOLOGY_22: (8)                      label.move_to(submob)
SNGT_QHENOMENOLOGY_23: (8)                      label.set_backstroke(BLACK, 5)
SNGT_QHENOMENOLOGY_24: (8)                      labels.add(label)
SNGT_QHENOMENOLOGY_25: (4)                  return labels
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 81 - paths.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import math
SNGT_QHENOMENOLOGY_3: (0)              import numpy as np
SNGT_QHENOMENOLOGY_4: (0)              from manimlib.constants import OUT
SNGT_QHENOMENOLOGY_5: (0)              from manimlib.utils.bezier import interpolate
SNGT_QHENOMENOLOGY_6: (0)              from manimlib.utils.space_ops import get_norm
SNGT_QHENOMENOLOGY_7: (0)              from manimlib.utils.space_ops import
rotation_matrix_transpose
SNGT_QHENOMENOLOGY_8: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_9: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_10: (4)                 from typing import Callable
SNGT_QHENOMENOLOGY_11: (4)                 from manimlib.typing import Vect3, Vect3Array
SNGT_QHENOMENOLOGY_12: (0)             STRAIGHT_PATH_THRESHOLD = 0.01
SNGT_QHENOMENOLOGY_13: (0)             def straight_path(
SNGT_QHENOMENOLOGY_14: (4)                 start_points: np.ndarray,
SNGT_QHENOMENOLOGY_15: (4)                 end_points: np.ndarray,
SNGT_QHENOMENOLOGY_16: (4)                 alpha: float
SNGT_QHENOMENOLOGY_17: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_18: (4)                 """
SNGT_QHENOMENOLOGY_19: (4)                 Same function as interpolate, but renamed to reflect
SNGT_QHENOMENOLOGY_20: (4)                 intent of being used to determine how a set of points
move
SNGT_QHENOMENOLOGY_21: (4)                 to another set.  For instance, it should be a specific
case
SNGT_QHENOMENOLOGY_22: (4)                 of path_along_arc
SNGT_QHENOMENOLOGY_23: (4)                 """
SNGT_QHENOMENOLOGY_24: (4)                 return interpolate(start_points, end_points, alpha)
SNGT_QHENOMENOLOGY_25: (0)             def path_along_arc(
SNGT_QHENOMENOLOGY_26: (4)                 arc_angle: float,
SNGT_QHENOMENOLOGY_27: (4)                 axis: Vect3 = OUT
SNGT_QHENOMENOLOGY_28: (0)             ) -> Callable[[Vect3Array, Vect3Array, float], Vect3Array]:
SNGT_QHENOMENOLOGY_29: (4)                 """
SNGT_QHENOMENOLOGY_30: (4)                 If vect is vector from start to end, [vect[:,1], -
vect[:,0]] is
SNGT_QHENOMENOLOGY_31: (4)                 perpendicular to vect in the left direction.
SNGT_QHENOMENOLOGY_32: (4)                 """
SNGT_QHENOMENOLOGY_33: (4)                 if abs(arc_angle) < STRAIGHT_PATH_THRESHOLD:
SNGT_QHENOMENOLOGY_34: (8)                     return straight_path
SNGT_QHENOMENOLOGY_35: (4)                 if get_norm(axis) == 0:
SNGT_QHENOMENOLOGY_36: (8)                     axis = OUT
SNGT_QHENOMENOLOGY_37: (4)                 unit_axis = axis / get_norm(axis)
SNGT_QHENOMENOLOGY_38: (4)                 def path(start_points, end_points, alpha):
SNGT_QHENOMENOLOGY_39: (8)                     vects = end_points - start_points
SNGT_QHENOMENOLOGY_40: (8)                     centers = start_points + 0.5 * vects
SNGT_QHENOMENOLOGY_41: (8)                     if arc_angle != np.pi:
SNGT_QHENOMENOLOGY_42: (12)                        centers += np.cross(unit_axis, vects / 2.0) /
math.tan(arc_angle / 2)
SNGT_QHENOMENOLOGY_43: (8)                     rot_matrix_T = rotation_matrix_transpose(alpha *
arc_angle, unit_axis)
SNGT_QHENOMENOLOGY_44: (8)                     return centers + np.dot(start_points - centers,
rot_matrix_T)
SNGT_QHENOMENOLOGY_45: (4)                 return path
SNGT_QHENOMENOLOGY_46: (0)             def clockwise_path() -> Callable[[Vect3Array, Vect3Array,
float], Vect3Array]:
SNGT_QHENOMENOLOGY_47: (4)                 return path_along_arc(-np.pi)
SNGT_QHENOMENOLOGY_48: (0)             def counterclockwise_path() -> Callable[[Vect3Array,
Vect3Array, float], Vect3Array]:
SNGT_QHENOMENOLOGY_49: (4)                 return path_along_arc(np.pi)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 82 - window.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              import moderngl_window as mglw
```

```
SNGT_QHENOMENOLOGY_4: (0)               from moderngl_window.context.pyglet.window import Window as
PygletWindow
SNGT_QHENOMENOLOGY_5: (0)               from moderngl_window.timers.clock import Timer
SNGT_QHENOMENOLOGY_6: (0)               from functools import wraps
SNGT_QHENOMENOLOGY_7: (0)               import screeninfo
SNGT_QHENOMENOLOGY_8: (0)               from manimlib.constants import ASPECT_RATIO
SNGT_QHENOMENOLOGY_9: (0)               from manimlib.constants import FRAME_SHAPE
SNGT_QHENOMENOLOGY_10: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_11: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_12: (4)                  from typing import Callable, TypeVar, Optional
SNGT_QHENOMENOLOGY_13: (4)                  from manimlib.scene.scene import Scene
SNGT_QHENOMENOLOGY_14: (4)                  T = TypeVar("T")
SNGT_QHENOMENOLOGY_15: (0)              class Window(PygletWindow):
SNGT_QHENOMENOLOGY_16: (4)                  fullscreen: bool = False
SNGT_QHENOMENOLOGY_17: (4)                  resizable: bool = True
SNGT_QHENOMENOLOGY_18: (4)                  gl_version: tuple[int, int] = (3, 3)
SNGT_QHENOMENOLOGY_19: (4)                  vsync: bool = True
SNGT_QHENOMENOLOGY_20: (4)                  cursor: bool = True
SNGT_QHENOMENOLOGY_21: (4)                  def __init__(
SNGT_QHENOMENOLOGY_22: (8)                      self,
SNGT_QHENOMENOLOGY_23: (8)                      scene: Optional[Scene] = None,
SNGT_QHENOMENOLOGY_24: (8)                      position_string: str = "UR",
SNGT_QHENOMENOLOGY_25: (8)                      monitor_index: int = 1,
SNGT_QHENOMENOLOGY_26: (8)                      full_screen: bool = False,
SNGT_QHENOMENOLOGY_27: (8)                      size: Optional[tuple[int, int]] = None,
SNGT_QHENOMENOLOGY_28: (8)                      position: Optional[tuple[int, int]] = None,
SNGT_QHENOMENOLOGY_29: (8)                      samples: int = 0
SNGT_QHENOMENOLOGY_30: (4)                  ):
SNGT_QHENOMENOLOGY_31: (8)                      self.scene = scene
SNGT_QHENOMENOLOGY_32: (8)                      self.monitor = self.get_monitor(monitor_index)
SNGT_QHENOMENOLOGY_33: (8)                      self.default_size = size or
self.get_default_size(full_screen)
SNGT_QHENOMENOLOGY_34: (8)                      self.default_position = position or
self.position_from_string(position_string)
SNGT_QHENOMENOLOGY_35: (8)                      self.pressed_keys = set()
SNGT_QHENOMENOLOGY_36: (8)                      super().__init__(samples=samples)
SNGT_QHENOMENOLOGY_37: (8)                      self.to_default_position()
SNGT_QHENOMENOLOGY_38: (8)                      if self.scene:
SNGT_QHENOMENOLOGY_39: (12)                         self.init_for_scene(scene)
SNGT_QHENOMENOLOGY_40: (4)                  def init_for_scene(self, scene: Scene):
SNGT_QHENOMENOLOGY_41: (8)                      """
SNGT_QHENOMENOLOGY_42: (8)                      Resets the state and updates the scene associated
to this window.
SNGT_QHENOMENOLOGY_43: (8)                      This is necessary when we want to reuse an
*existing* window after a
SNGT_QHENOMENOLOGY_44: (8)                      `scene.reload()` was requested, which will create
new scene instances.
SNGT_QHENOMENOLOGY_45: (8)                      """
SNGT_QHENOMENOLOGY_46: (8)                      self.pressed_keys.clear()
SNGT_QHENOMENOLOGY_47: (8)                      self._has_undrawn_event = True
SNGT_QHENOMENOLOGY_48: (8)                      self.scene = scene
SNGT_QHENOMENOLOGY_49: (8)                      self.title = str(scene)
SNGT_QHENOMENOLOGY_50: (8)                      self.init_mgl_context()
SNGT_QHENOMENOLOGY_51: (8)                      self.timer = Timer()
SNGT_QHENOMENOLOGY_52: (8)                      self.config = mglw.WindowConfig(ctx=self.ctx,
wnd=self, timer=self.timer)
SNGT_QHENOMENOLOGY_53: (8)                      mglw.activate_context(window=self, ctx=self.ctx)
SNGT_QHENOMENOLOGY_54: (8)                      self.timer.start()
SNGT_QHENOMENOLOGY_55: (8)                      self.on_resize(*self.size)
SNGT_QHENOMENOLOGY_56: (4)                  def get_monitor(self, index):
SNGT_QHENOMENOLOGY_57: (8)                      try:
SNGT_QHENOMENOLOGY_58: (12)                         monitors = screeninfo.get_monitors()
SNGT_QHENOMENOLOGY_59: (12)                         return monitors[min(index, len(monitors) - 1)]
SNGT_QHENOMENOLOGY_60: (8)                      except screeninfo.ScreenInfoError:
SNGT_QHENOMENOLOGY_61: (12)                         return screeninfo.Monitor(width=1920,
height=1080)
SNGT_QHENOMENOLOGY_62: (4)                  def get_default_size(self, full_screen=False):
SNGT_QHENOMENOLOGY_63: (8)                      width = self.monitor.width // (1 if full_screen
else 2)
```

```
SNGT_QHENOMENOLOGY_64: (8)                      height = int(width // ASPECT_RATIO)
SNGT_QHENOMENOLOGY_65: (8)                      return (width, height)
SNGT_QHENOMENOLOGY_66: (4)                  def position_from_string(self, position_string):
SNGT_QHENOMENOLOGY_67: (8)                      char_to_n = {"L": 0, "U": 0, "O": 1, "R": 2, "D":
2}
SNGT_QHENOMENOLOGY_68: (8)                      size = self.default_size
SNGT_QHENOMENOLOGY_69: (8)                      width_diff = self.monitor.width - size[0]
SNGT_QHENOMENOLOGY_70: (8)                      height_diff = self.monitor.height - size[1]
SNGT_QHENOMENOLOGY_71: (8)                      x_step = char_to_n[position_string[1]] * width_diff
// 2
SNGT_QHENOMENOLOGY_72: (8)                      y_step = char_to_n[position_string[0]] *
height_diff // 2
SNGT_QHENOMENOLOGY_73: (8)                      return (self.monitor.x + x_step, -self.monitor.y +
y_step)
SNGT_QHENOMENOLOGY_74: (4)                  def focus(self):
SNGT_QHENOMENOLOGY_75: (8)                      """
SNGT_QHENOMENOLOGY_76: (8)                      Puts focus on this window by hiding and showing it
again.
SNGT_QHENOMENOLOGY_77: (8)                      Note that the pyglet `activate()` method didn't
work as expected here,
SNGT_QHENOMENOLOGY_78: (8)                      so that's why we have to use this workaround. This
will produce a small
SNGT_QHENOMENOLOGY_79: (8)                      flicker on the window but at least reliably focuses
it. It may also
SNGT_QHENOMENOLOGY_80: (8)                      offset the window position slightly.
SNGT_QHENOMENOLOGY_81: (8)                      """
SNGT_QHENOMENOLOGY_82: (8)                      self._window.set_visible(False)
SNGT_QHENOMENOLOGY_83: (8)                      self._window.set_visible(True)
SNGT_QHENOMENOLOGY_84: (4)                  def to_default_position(self):
SNGT_QHENOMENOLOGY_85: (8)                      self.position = self.default_position
SNGT_QHENOMENOLOGY_86: (8)                      w, h = self.default_size
SNGT_QHENOMENOLOGY_87: (8)                      self.size = (w - 1, h - 1)
SNGT_QHENOMENOLOGY_88: (8)                      self.size = (w, h)
SNGT_QHENOMENOLOGY_89: (4)                  def pixel_coords_to_space_coords(
SNGT_QHENOMENOLOGY_90: (8)                      self,
SNGT_QHENOMENOLOGY_91: (8)                      px: int,
SNGT_QHENOMENOLOGY_92: (8)                      py: int,
SNGT_QHENOMENOLOGY_93: (8)                      relative: bool = False
SNGT_QHENOMENOLOGY_94: (4)                  ) -> np.ndarray:
SNGT_QHENOMENOLOGY_95: (8)                      if self.scene is None or not hasattr(self.scene,
"frame"):
SNGT_QHENOMENOLOGY_96: (12)                         return np.zeros(3)
SNGT_QHENOMENOLOGY_97: (8)                      pixel_shape = np.array(self.size)
SNGT_QHENOMENOLOGY_98: (8)                      fixed_frame_shape = np.array(FRAME_SHAPE)
SNGT_QHENOMENOLOGY_99: (8)                      frame = self.scene.frame
SNGT_QHENOMENOLOGY_100: (8)                     coords = np.zeros(3)
SNGT_QHENOMENOLOGY_101: (8)                     coords[:2] = (fixed_frame_shape / pixel_shape) *
np.array([px, py])
SNGT_QHENOMENOLOGY_102: (8)                     if not relative:
SNGT_QHENOMENOLOGY_103: (12)                        coords[:2] -= 0.5 * fixed_frame_shape
SNGT_QHENOMENOLOGY_104: (8)                     return frame.from_fixed_frame_point(coords,
relative)
SNGT_QHENOMENOLOGY_105: (4)                 def has_undrawn_event(self) -> bool:
SNGT_QHENOMENOLOGY_106: (8)                     return self._has_undrawn_event
SNGT_QHENOMENOLOGY_107: (4)                 def swap_buffers(self):
SNGT_QHENOMENOLOGY_108: (8)                     super().swap_buffers()
SNGT_QHENOMENOLOGY_109: (8)                     self._has_undrawn_event = False
SNGT_QHENOMENOLOGY_110: (4)                 @staticmethod
SNGT_QHENOMENOLOGY_111: (4)                 def note_undrawn_event(func: Callable[..., T]) ->
Callable[..., T]:
SNGT_QHENOMENOLOGY_112: (8)                     @wraps(func)
SNGT_QHENOMENOLOGY_113: (8)                     def wrapper(self, *args, **kwargs):
SNGT_QHENOMENOLOGY_114: (12)                        func(self, *args, **kwargs)
SNGT_QHENOMENOLOGY_115: (12)                        self._has_undrawn_event = True
SNGT_QHENOMENOLOGY_116: (8)                     return wrapper
SNGT_QHENOMENOLOGY_117: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_118: (4)                 def on_mouse_motion(self, x: int, y: int, dx: int, dy:
int) -> None:
SNGT_QHENOMENOLOGY_119: (8)                     super().on_mouse_motion(x, y, dx, dy)
```

```
SNGT_QHENOMENOLOGY_120: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_121: (12)                         return
SNGT_QHENOMENOLOGY_122: (8)                     point = self.pixel_coords_to_space_coords(x, y)
SNGT_QHENOMENOLOGY_123: (8)                     d_point = self.pixel_coords_to_space_coords(dx, dy,
relative=True)
SNGT_QHENOMENOLOGY_124: (8)                     self.scene.on_mouse_motion(point, d_point)
SNGT_QHENOMENOLOGY_125: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_126: (4)                 def on_mouse_drag(self, x: int, y: int, dx: int, dy:
int, buttons: int, modifiers: int) -> None:
SNGT_QHENOMENOLOGY_127: (8)                     super().on_mouse_drag(x, y, dx, dy, buttons,
modifiers)
SNGT_QHENOMENOLOGY_128: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_129: (12)                         return
SNGT_QHENOMENOLOGY_130: (8)                     point = self.pixel_coords_to_space_coords(x, y)
SNGT_QHENOMENOLOGY_131: (8)                     d_point = self.pixel_coords_to_space_coords(dx, dy,
relative=True)
SNGT_QHENOMENOLOGY_132: (8)                     self.scene.on_mouse_drag(point, d_point, buttons,
modifiers)
SNGT_QHENOMENOLOGY_133: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_134: (4)                 def on_mouse_press(self, x: int, y: int, button: int,
mods: int) -> None:
SNGT_QHENOMENOLOGY_135: (8)                     super().on_mouse_press(x, y, button, mods)
SNGT_QHENOMENOLOGY_136: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_137: (12)                         return
SNGT_QHENOMENOLOGY_138: (8)                     point = self.pixel_coords_to_space_coords(x, y)
SNGT_QHENOMENOLOGY_139: (8)                     self.scene.on_mouse_press(point, button, mods)
SNGT_QHENOMENOLOGY_140: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_141: (4)                 def on_mouse_release(self, x: int, y: int, button: int,
mods: int) -> None:
SNGT_QHENOMENOLOGY_142: (8)                     super().on_mouse_release(x, y, button, mods)
SNGT_QHENOMENOLOGY_143: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_144: (12)                         return
SNGT_QHENOMENOLOGY_145: (8)                     point = self.pixel_coords_to_space_coords(x, y)
SNGT_QHENOMENOLOGY_146: (8)                     self.scene.on_mouse_release(point, button, mods)
SNGT_QHENOMENOLOGY_147: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_148: (4)                 def on_mouse_scroll(self, x: int, y: int, x_offset:
float, y_offset: float) -> None:
SNGT_QHENOMENOLOGY_149: (8)                     super().on_mouse_scroll(x, y, x_offset, y_offset)
SNGT_QHENOMENOLOGY_150: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_151: (12)                         return
SNGT_QHENOMENOLOGY_152: (8)                     point = self.pixel_coords_to_space_coords(x, y)
SNGT_QHENOMENOLOGY_153: (8)                     offset =
self.pixel_coords_to_space_coords(x_offset, y_offset, relative=True)
SNGT_QHENOMENOLOGY_154: (8)                     self.scene.on_mouse_scroll(point, offset, x_offset,
y_offset)
SNGT_QHENOMENOLOGY_155: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_156: (4)                 def on_key_press(self, symbol: int, modifiers: int) ->
None:
SNGT_QHENOMENOLOGY_157: (8)                     self.pressed_keys.add(symbol)  # Modifiers?
SNGT_QHENOMENOLOGY_158: (8)                     super().on_key_press(symbol, modifiers)
SNGT_QHENOMENOLOGY_159: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_160: (12)                         return
SNGT_QHENOMENOLOGY_161: (8)                     self.scene.on_key_press(symbol, modifiers)
SNGT_QHENOMENOLOGY_162: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_163: (4)                 def on_key_release(self, symbol: int, modifiers: int) -
> None:
SNGT_QHENOMENOLOGY_164: (8)                     self.pressed_keys.difference_update({symbol})  #
Modifiers?
SNGT_QHENOMENOLOGY_165: (8)                     super().on_key_release(symbol, modifiers)
SNGT_QHENOMENOLOGY_166: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_167: (12)                         return
SNGT_QHENOMENOLOGY_168: (8)                     self.scene.on_key_release(symbol, modifiers)
SNGT_QHENOMENOLOGY_169: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_170: (4)                 def on_resize(self, width: int, height: int) -> None:
SNGT_QHENOMENOLOGY_171: (8)                     super().on_resize(width, height)
SNGT_QHENOMENOLOGY_172: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_173: (12)                         return
SNGT_QHENOMENOLOGY_174: (8)                     self.scene.on_resize(width, height)
SNGT_QHENOMENOLOGY_175: (4)                 @note_undrawn_event
```

```
SNGT_QHENOMENOLOGY_176: (4)                 def on_show(self) -> None:
SNGT_QHENOMENOLOGY_177: (8)                     super().on_show()
SNGT_QHENOMENOLOGY_178: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_179: (12)                        return
SNGT_QHENOMENOLOGY_180: (8)                     self.scene.on_show()
SNGT_QHENOMENOLOGY_181: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_182: (4)                 def on_hide(self) -> None:
SNGT_QHENOMENOLOGY_183: (8)                     super().on_hide()
SNGT_QHENOMENOLOGY_184: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_185: (12)                        return
SNGT_QHENOMENOLOGY_186: (8)                     self.scene.on_hide()
SNGT_QHENOMENOLOGY_187: (4)                 @note_undrawn_event
SNGT_QHENOMENOLOGY_188: (4)                 def on_close(self) -> None:
SNGT_QHENOMENOLOGY_189: (8)                     super().on_close()
SNGT_QHENOMENOLOGY_190: (8)                     if not self.scene:
SNGT_QHENOMENOLOGY_191: (12)                        return
SNGT_QHENOMENOLOGY_192: (8)                     self.scene.on_close()
SNGT_QHENOMENOLOGY_193: (4)                 def is_key_pressed(self, symbol: int) -> bool:
SNGT_QHENOMENOLOGY_194: (8)                     return (symbol in self.pressed_keys)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 83 - images.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               import numpy as np
SNGT_QHENOMENOLOGY_3: (0)               from PIL import Image
SNGT_QHENOMENOLOGY_4: (0)               from manimlib.utils.directories import get_raster_image_dir
SNGT_QHENOMENOLOGY_5: (0)               from manimlib.utils.directories import get_vector_image_dir
SNGT_QHENOMENOLOGY_6: (0)               from manimlib.utils.file_ops import find_file
SNGT_QHENOMENOLOGY_7: (0)               from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)               if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                   from typing import Iterable
SNGT_QHENOMENOLOGY_10: (0)              def get_full_raster_image_path(image_file_name: str) ->
str:
SNGT_QHENOMENOLOGY_11: (4)                  return find_file(
SNGT_QHENOMENOLOGY_12: (8)                      image_file_name,
SNGT_QHENOMENOLOGY_13: (8)                      directories=[get_raster_image_dir()],
SNGT_QHENOMENOLOGY_14: (8)                      extensions=[".jpg", ".jpeg", ".png", ".gif", ""]
SNGT_QHENOMENOLOGY_15: (4)                  )
SNGT_QHENOMENOLOGY_16: (0)              def get_full_vector_image_path(image_file_name: str) ->
str:
SNGT_QHENOMENOLOGY_17: (4)                  return find_file(
SNGT_QHENOMENOLOGY_18: (8)                      image_file_name,
SNGT_QHENOMENOLOGY_19: (8)                      directories=[get_vector_image_dir()],
SNGT_QHENOMENOLOGY_20: (8)                      extensions=[".svg", ".xdv", ""],
SNGT_QHENOMENOLOGY_21: (4)                  )
SNGT_QHENOMENOLOGY_22: (0)              def invert_image(image: Iterable) -> Image.Image:
SNGT_QHENOMENOLOGY_23: (4)                  arr = np.array(image)
SNGT_QHENOMENOLOGY_24: (4)                  arr = (255 * np.ones(arr.shape)).astype(arr.dtype) -
arr
SNGT_QHENOMENOLOGY_25: (4)                  return Image.fromarray(arr)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 84 - sounds.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)               from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)               from manimlib.utils.directories import get_sound_dir
SNGT_QHENOMENOLOGY_3: (0)               from manimlib.utils.file_ops import find_file
SNGT_QHENOMENOLOGY_4: (0)               def get_full_sound_file_path(sound_file_name: str) -> str:
SNGT_QHENOMENOLOGY_5: (4)                   return find_file(
SNGT_QHENOMENOLOGY_6: (8)                       sound_file_name,
SNGT_QHENOMENOLOGY_7: (8)                       directories=[get_sound_dir()],
SNGT_QHENOMENOLOGY_8: (8)                       extensions=[".wav", ".mp3", ""]
SNGT_QHENOMENOLOGY_9: (4)                   )
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
```

SNGT_QHENOMENOLOGY_File 85 - shaders.py:

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import os
SNGT_QHENOMENOLOGY_3: (0)             import re
SNGT_QHENOMENOLOGY_4: (0)             from functools import lru_cache
SNGT_QHENOMENOLOGY_5: (0)             import moderngl
SNGT_QHENOMENOLOGY_6: (0)             from PIL import Image
SNGT_QHENOMENOLOGY_7: (0)             import numpy as np
SNGT_QHENOMENOLOGY_8: (0)             from manimlib.utils.directories import get_shader_dir
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.utils.file_ops import find_file
SNGT_QHENOMENOLOGY_10: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_11: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_12: (4)                from typing import Sequence, Optional
SNGT_QHENOMENOLOGY_13: (0)            PROGRAM_UNIFORM_MIRRORS: dict[int, dict[str, float |
tuple]] = dict()
SNGT_QHENOMENOLOGY_14: (0)            @lru_cache()
SNGT_QHENOMENOLOGY_15: (0)            def image_path_to_texture(path: str, ctx: moderngl.Context)
-> moderngl.Texture:
SNGT_QHENOMENOLOGY_16: (4)                im = Image.open(path).convert("RGBA")
SNGT_QHENOMENOLOGY_17: (4)                return ctx.texture(
SNGT_QHENOMENOLOGY_18: (8)                    size=im.size,
SNGT_QHENOMENOLOGY_19: (8)                    components=len(im.getbands()),
SNGT_QHENOMENOLOGY_20: (8)                    data=im.tobytes(),
SNGT_QHENOMENOLOGY_21: (4)                )
SNGT_QHENOMENOLOGY_22: (0)            @lru_cache()
SNGT_QHENOMENOLOGY_23: (0)            def get_shader_program(
SNGT_QHENOMENOLOGY_24: (8)                    ctx: moderngl.context.Context,
SNGT_QHENOMENOLOGY_25: (8)                    vertex_shader: str,
SNGT_QHENOMENOLOGY_26: (8)                    fragment_shader: Optional[str] = None,
SNGT_QHENOMENOLOGY_27: (8)                    geometry_shader: Optional[str] = None,
SNGT_QHENOMENOLOGY_28: (0)            ) -> moderngl.Program:
SNGT_QHENOMENOLOGY_29: (4)                return ctx.program(
SNGT_QHENOMENOLOGY_30: (8)                    vertex_shader=vertex_shader,
SNGT_QHENOMENOLOGY_31: (8)                    fragment_shader=fragment_shader,
SNGT_QHENOMENOLOGY_32: (8)                    geometry_shader=geometry_shader,
SNGT_QHENOMENOLOGY_33: (4)                )
SNGT_QHENOMENOLOGY_34: (0)            def set_program_uniform(
SNGT_QHENOMENOLOGY_35: (4)                program: moderngl.Program,
SNGT_QHENOMENOLOGY_36: (4)                name: str,
SNGT_QHENOMENOLOGY_37: (4)                value: float | tuple | np.ndarray
SNGT_QHENOMENOLOGY_38: (0)            ) -> bool:
SNGT_QHENOMENOLOGY_39: (4)                """
SNGT_QHENOMENOLOGY_40: (4)                Sets a program uniform, and also keeps track of a
dictionary
SNGT_QHENOMENOLOGY_41: (4)                of previously set uniforms for that program so that it
SNGT_QHENOMENOLOGY_42: (4)                doesn't needlessly reset it, requiring an exchange with
gpu
SNGT_QHENOMENOLOGY_43: (4)                memory, if it sees the same value again.
SNGT_QHENOMENOLOGY_44: (4)                Returns True if changed the program, False if it left
it as is.
SNGT_QHENOMENOLOGY_45: (4)                """
SNGT_QHENOMENOLOGY_46: (4)                pid = id(program)
SNGT_QHENOMENOLOGY_47: (4)                if pid not in PROGRAM_UNIFORM_MIRRORS:
SNGT_QHENOMENOLOGY_48: (8)                    PROGRAM_UNIFORM_MIRRORS[pid] = dict()
SNGT_QHENOMENOLOGY_49: (4)                uniform_mirror = PROGRAM_UNIFORM_MIRRORS[pid]
SNGT_QHENOMENOLOGY_50: (4)                if type(value) is np.ndarray and value.ndim > 0:
SNGT_QHENOMENOLOGY_51: (8)                    value = tuple(value.flatten())
SNGT_QHENOMENOLOGY_52: (4)                if uniform_mirror.get(name, None) == value:
SNGT_QHENOMENOLOGY_53: (8)                    return False
SNGT_QHENOMENOLOGY_54: (4)                try:
SNGT_QHENOMENOLOGY_55: (8)                    program[name].value = value
SNGT_QHENOMENOLOGY_56: (4)                except KeyError:
SNGT_QHENOMENOLOGY_57: (8)                    return False
SNGT_QHENOMENOLOGY_58: (4)                uniform_mirror[name] = value
SNGT_QHENOMENOLOGY_59: (4)                return True
SNGT_QHENOMENOLOGY_60: (0)            @lru_cache()
SNGT_QHENOMENOLOGY_61: (0)            def get_shader_code_from_file(filename: str) -> str | None:
SNGT_QHENOMENOLOGY_62: (4)                if not filename:
```

```
SNGT_QHENOMENOLOGY_63: (8)                    return None
SNGT_QHENOMENOLOGY_64: (4)                try:
SNGT_QHENOMENOLOGY_65: (8)                    filepath = find_file(
SNGT_QHENOMENOLOGY_66: (12)                       filename,
SNGT_QHENOMENOLOGY_67: (12)                       directories=[get_shader_dir(), "/"],
SNGT_QHENOMENOLOGY_68: (12)                       extensions=[],
SNGT_QHENOMENOLOGY_69: (8)                    )
SNGT_QHENOMENOLOGY_70: (4)                except IOError:
SNGT_QHENOMENOLOGY_71: (8)                    return None
SNGT_QHENOMENOLOGY_72: (4)                with open(filepath, "r") as f:
SNGT_QHENOMENOLOGY_73: (8)                    result = f.read()
SNGT_QHENOMENOLOGY_74: (4)                insertions = re.findall(r"^#INSERT .*\.glsl$", result,
flags=re.MULTILINE)
SNGT_QHENOMENOLOGY_75: (4)                for line in insertions:
SNGT_QHENOMENOLOGY_76: (8)                    inserted_code = get_shader_code_from_file(
SNGT_QHENOMENOLOGY_77: (12)                       os.path.join("inserts", line.replace("#INSERT
", ""))
SNGT_QHENOMENOLOGY_78: (8)                    )
SNGT_QHENOMENOLOGY_79: (8)                    result = result.replace(line, inserted_code)
SNGT_QHENOMENOLOGY_80: (4)                return result
SNGT_QHENOMENOLOGY_81: (0)            def get_colormap_code(rgb_list: Sequence[float]) -> str:
SNGT_QHENOMENOLOGY_82: (4)                data = ",".join(
SNGT_QHENOMENOLOGY_83: (8)                    "vec3({}, {}, {})".format(*rgb)
SNGT_QHENOMENOLOGY_84: (8)                    for rgb in rgb_list
SNGT_QHENOMENOLOGY_85: (4)                )
SNGT_QHENOMENOLOGY_86: (4)                return f"vec3[{len(rgb_list)}]({data})"
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 86 - dict_ops.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             import itertools as it
SNGT_QHENOMENOLOGY_2: (0)             import numpy as np
SNGT_QHENOMENOLOGY_3: (0)             def merge_dicts_recursively(*dicts):
SNGT_QHENOMENOLOGY_4: (4)                 """
SNGT_QHENOMENOLOGY_5: (4)                 Creates a dict whose keyset is the union of all the
SNGT_QHENOMENOLOGY_6: (4)                 input dictionaries.  The value for each key is based
SNGT_QHENOMENOLOGY_7: (4)                 on the first dict in the list with that key.
SNGT_QHENOMENOLOGY_8: (4)                 dicts later in the list have higher priority
SNGT_QHENOMENOLOGY_9: (4)                 When values are dictionaries, it is applied recursively
SNGT_QHENOMENOLOGY_10: (4)                """
SNGT_QHENOMENOLOGY_11: (4)                result = dict()
SNGT_QHENOMENOLOGY_12: (4)                all_items = it.chain(*[d.items() for d in dicts])
SNGT_QHENOMENOLOGY_13: (4)                for key, value in all_items:
SNGT_QHENOMENOLOGY_14: (8)                    if key in result and isinstance(result[key], dict)
and isinstance(value, dict):
SNGT_QHENOMENOLOGY_15: (12)                       result[key] =
merge_dicts_recursively(result[key], value)
SNGT_QHENOMENOLOGY_16: (8)                    else:
SNGT_QHENOMENOLOGY_17: (12)                       result[key] = value
SNGT_QHENOMENOLOGY_18: (4)                return result
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 87 - file_ops.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import os
SNGT_QHENOMENOLOGY_3: (0)             from pathlib import Path
SNGT_QHENOMENOLOGY_4: (0)             import hashlib
SNGT_QHENOMENOLOGY_5: (0)             import numpy as np
SNGT_QHENOMENOLOGY_6: (0)             import validators
SNGT_QHENOMENOLOGY_7: (0)             import urllib.request
SNGT_QHENOMENOLOGY_8: (0)             import manimlib.utils.directories
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.utils.simple_functions import hash_string
SNGT_QHENOMENOLOGY_10: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_11: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_12: (4)                from typing import Iterable
SNGT_QHENOMENOLOGY_13: (0)            def guarantee_existence(path: str | Path) -> Path:
```

```
SNGT_QHENOMENOLOGY_14: (4)                 path = Path(path)
SNGT_QHENOMENOLOGY_15: (4)                 path.mkdir(parents=True, exist_ok=True)
SNGT_QHENOMENOLOGY_16: (4)                 return path.absolute()
SNGT_QHENOMENOLOGY_17: (0)             def find_file(
SNGT_QHENOMENOLOGY_18: (4)                 file_name: str,
SNGT_QHENOMENOLOGY_19: (4)                 directories: Iterable[str] | None = None,
SNGT_QHENOMENOLOGY_20: (4)                 extensions: Iterable[str] | None = None
SNGT_QHENOMENOLOGY_21: (0)             ) -> Path:
SNGT_QHENOMENOLOGY_22: (4)                 if validators.url(file_name):
SNGT_QHENOMENOLOGY_23: (8)                     suffix = Path(file_name).suffix
SNGT_QHENOMENOLOGY_24: (8)                     file_hash = hash_string(file_name)
SNGT_QHENOMENOLOGY_25: (8)                     folder =
manimlib.utils.directories.get_downloads_dir()
SNGT_QHENOMENOLOGY_26: (8)                     path = Path(folder, file_hash).with_suffix(suffix)
SNGT_QHENOMENOLOGY_27: (8)                     urllib.request.urlretrieve(file_name, path)
SNGT_QHENOMENOLOGY_28: (8)                     return path
SNGT_QHENOMENOLOGY_29: (4)                 if os.path.exists(file_name):
SNGT_QHENOMENOLOGY_30: (8)                     return Path(file_name)
SNGT_QHENOMENOLOGY_31: (4)                 directories = directories or [""]
SNGT_QHENOMENOLOGY_32: (4)                 extensions = extensions or [""]
SNGT_QHENOMENOLOGY_33: (4)                 possible_paths = (
SNGT_QHENOMENOLOGY_34: (8)                     Path(directory, file_name + extension)
SNGT_QHENOMENOLOGY_35: (8)                     for directory in directories
SNGT_QHENOMENOLOGY_36: (8)                     for extension in extensions
SNGT_QHENOMENOLOGY_37: (4)                 )
SNGT_QHENOMENOLOGY_38: (4)                 for path in possible_paths:
SNGT_QHENOMENOLOGY_39: (8)                     if path.exists():
SNGT_QHENOMENOLOGY_40: (12)                        return path
SNGT_QHENOMENOLOGY_41: (4)                 raise IOError(f"{file_name} not Found")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 88 - iterables.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from colour import Color
SNGT_QHENOMENOLOGY_3: (0)              import numpy as np
SNGT_QHENOMENOLOGY_4: (0)              import random
SNGT_QHENOMENOLOGY_5: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_6: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_7: (4)                  from typing import Callable, Iterable, Sequence,
TypeVar
SNGT_QHENOMENOLOGY_8: (4)                  T = TypeVar("T")
SNGT_QHENOMENOLOGY_9: (4)                  S = TypeVar("S")
SNGT_QHENOMENOLOGY_10: (0)             def remove_list_redundancies(lst: Sequence[T]) -> list[T]:
SNGT_QHENOMENOLOGY_11: (4)                 """
SNGT_QHENOMENOLOGY_12: (4)                 Remove duplicate elements while preserving order.
SNGT_QHENOMENOLOGY_13: (4)                 Keeps the last occurrence of each element
SNGT_QHENOMENOLOGY_14: (4)                 """
SNGT_QHENOMENOLOGY_15: (4)                 return list(reversed(dict.fromkeys(reversed(lst))))
SNGT_QHENOMENOLOGY_16: (0)             def list_update(l1: Iterable[T], l2: Iterable[T]) ->
list[T]:
SNGT_QHENOMENOLOGY_17: (4)                 """
SNGT_QHENOMENOLOGY_18: (4)                 Used instead of list(set(l1).update(l2)) to maintain
order,
SNGT_QHENOMENOLOGY_19: (4)                 making sure duplicates are removed from l1, not l2.
SNGT_QHENOMENOLOGY_20: (4)                 """
SNGT_QHENOMENOLOGY_21: (4)                 return remove_list_redundancies([*l1, *l2])
SNGT_QHENOMENOLOGY_22: (0)             def list_difference_update(l1: Iterable[T], l2:
Iterable[T]) -> list[T]:
SNGT_QHENOMENOLOGY_23: (4)                 return [e for e in l1 if e not in l2]
SNGT_QHENOMENOLOGY_24: (0)             def adjacent_n_tuples(objects: Sequence[T], n: int) ->
zip[tuple[T, ...]]:
SNGT_QHENOMENOLOGY_25: (4)                 return zip(*[
SNGT_QHENOMENOLOGY_26: (8)                     [*objects[k:], *objects[:k]]
SNGT_QHENOMENOLOGY_27: (8)                     for k in range(n)
SNGT_QHENOMENOLOGY_28: (4)                 ])
SNGT_QHENOMENOLOGY_29: (0)             def adjacent_pairs(objects: Sequence[T]) -> zip[tuple[T,
T]]:
```

```
SNGT_QHENOMENOLOGY_30: (4)                 return adjacent_n_tuples(objects, 2)
SNGT_QHENOMENOLOGY_31: (0)             def batch_by_property(
SNGT_QHENOMENOLOGY_32: (4)                 items: Iterable[T],
SNGT_QHENOMENOLOGY_33: (4)                 property_func: Callable[[T], S]
SNGT_QHENOMENOLOGY_34: (0)             ) -> list[tuple[T, S]]:
SNGT_QHENOMENOLOGY_35: (4)                 """
SNGT_QHENOMENOLOGY_36: (4)                 Takes in a list, and returns a list of tuples, (batch,
prop)
SNGT_QHENOMENOLOGY_37: (4)                 such that all items in a batch have the same output
when
SNGT_QHENOMENOLOGY_38: (4)                 put into property_func, and such that chaining all
these
SNGT_QHENOMENOLOGY_39: (4)                 batches together would give the original list (i.e.
order is
SNGT_QHENOMENOLOGY_40: (4)                 preserved)
SNGT_QHENOMENOLOGY_41: (4)                 """
SNGT_QHENOMENOLOGY_42: (4)                 batch_prop_pairs = []
SNGT_QHENOMENOLOGY_43: (4)                 curr_batch = []
SNGT_QHENOMENOLOGY_44: (4)                 curr_prop = None
SNGT_QHENOMENOLOGY_45: (4)                 for item in items:
SNGT_QHENOMENOLOGY_46: (8)                     prop = property_func(item)
SNGT_QHENOMENOLOGY_47: (8)                     if prop != curr_prop:
SNGT_QHENOMENOLOGY_48: (12)                        if len(curr_batch) > 0:
SNGT_QHENOMENOLOGY_49: (16)                            batch_prop_pairs.append((curr_batch,
curr_prop))
SNGT_QHENOMENOLOGY_50: (12)                        curr_prop = prop
SNGT_QHENOMENOLOGY_51: (12)                        curr_batch = [item]
SNGT_QHENOMENOLOGY_52: (8)                     else:
SNGT_QHENOMENOLOGY_53: (12)                        curr_batch.append(item)
SNGT_QHENOMENOLOGY_54: (4)                 if len(curr_batch) > 0:
SNGT_QHENOMENOLOGY_55: (8)                     batch_prop_pairs.append((curr_batch, curr_prop))
SNGT_QHENOMENOLOGY_56: (4)                 return batch_prop_pairs
SNGT_QHENOMENOLOGY_57: (0)             def listify(obj: object) -> list:
SNGT_QHENOMENOLOGY_58: (4)                 if isinstance(obj, str):
SNGT_QHENOMENOLOGY_59: (8)                     return [obj]
SNGT_QHENOMENOLOGY_60: (4)                 try:
SNGT_QHENOMENOLOGY_61: (8)                     return list(obj)
SNGT_QHENOMENOLOGY_62: (4)                 except TypeError:
SNGT_QHENOMENOLOGY_63: (8)                     return [obj]
SNGT_QHENOMENOLOGY_64: (0)             def shuffled(iterable: Iterable) -> list:
SNGT_QHENOMENOLOGY_65: (4)                 as_list = list(iterable)
SNGT_QHENOMENOLOGY_66: (4)                 random.shuffle(as_list)
SNGT_QHENOMENOLOGY_67: (4)                 return as_list
SNGT_QHENOMENOLOGY_68: (0)             def resize_array(nparray: np.ndarray, length: int) ->
np.ndarray:
SNGT_QHENOMENOLOGY_69: (4)                 if len(nparray) == length:
SNGT_QHENOMENOLOGY_70: (8)                     return nparray
SNGT_QHENOMENOLOGY_71: (4)                 return np.resize(nparray, (length, *nparray.shape[1:]))
SNGT_QHENOMENOLOGY_72: (0)             def resize_preserving_order(nparray: np.ndarray, length:
int) -> np.ndarray:
SNGT_QHENOMENOLOGY_73: (4)                 if len(nparray) == 0:
SNGT_QHENOMENOLOGY_74: (8)                     return np.resize(nparray, length)
SNGT_QHENOMENOLOGY_75: (4)                 if len(nparray) == length:
SNGT_QHENOMENOLOGY_76: (8)                     return nparray
SNGT_QHENOMENOLOGY_77: (4)                 indices = np.arange(length) * len(nparray) // length
SNGT_QHENOMENOLOGY_78: (4)                 return nparray[indices]
SNGT_QHENOMENOLOGY_79: (0)             def resize_with_interpolation(nparray: np.ndarray, length:
int) -> np.ndarray:
SNGT_QHENOMENOLOGY_80: (4)                 if len(nparray) == length:
SNGT_QHENOMENOLOGY_81: (8)                     return nparray
SNGT_QHENOMENOLOGY_82: (4)                 if len(nparray) == 1 or array_is_constant(nparray):
SNGT_QHENOMENOLOGY_83: (8)                     return nparray[:1].repeat(length, axis=0)
SNGT_QHENOMENOLOGY_84: (4)                 if length == 0:
SNGT_QHENOMENOLOGY_85: (8)                     return np.zeros((0, *nparray.shape[1:]))
SNGT_QHENOMENOLOGY_86: (4)                 cont_indices = np.linspace(0, len(nparray) - 1, length)
SNGT_QHENOMENOLOGY_87: (4)                 return np.array([
SNGT_QHENOMENOLOGY_88: (8)                     (1 - a) * nparray[lh] + a * nparray[rh]
SNGT_QHENOMENOLOGY_89: (8)                     for ci in cont_indices
SNGT_QHENOMENOLOGY_90: (8)                     for lh, rh, a in [(int(ci), int(np.ceil(ci)), ci %
```

```
1)]
SNGT_QHENOMENOLOGY_91: (4)                ])
SNGT_QHENOMENOLOGY_92: (0)            def make_even(
SNGT_QHENOMENOLOGY_93: (4)                iterable_1: Sequence[T],
SNGT_QHENOMENOLOGY_94: (4)                iterable_2: Sequence[S]
SNGT_QHENOMENOLOGY_95: (0)            ) -> tuple[Sequence[T], Sequence[S]]:
SNGT_QHENOMENOLOGY_96: (4)                len1 = len(iterable_1)
SNGT_QHENOMENOLOGY_97: (4)                len2 = len(iterable_2)
SNGT_QHENOMENOLOGY_98: (4)                if len1 == len2:
SNGT_QHENOMENOLOGY_99: (8)                    return iterable_1, iterable_2
SNGT_QHENOMENOLOGY_100: (4)               new_len = max(len1, len2)
SNGT_QHENOMENOLOGY_101: (4)               return (
SNGT_QHENOMENOLOGY_102: (8)                   [iterable_1[(n * len1) // new_len] for n in
range(new_len)],
SNGT_QHENOMENOLOGY_103: (8)                   [iterable_2[(n * len2) // new_len] for n in
range(new_len)]
SNGT_QHENOMENOLOGY_104: (4)               )
SNGT_QHENOMENOLOGY_105: (0)           def arrays_match(arr1: np.ndarray, arr2: np.ndarray) ->
bool:
SNGT_QHENOMENOLOGY_106: (4)               return arr1.shape == arr2.shape and (arr1 ==
arr2).all()
SNGT_QHENOMENOLOGY_107: (0)           def array_is_constant(arr: np.ndarray) -> bool:
SNGT_QHENOMENOLOGY_108: (4)               return len(arr) > 0 and (arr == arr[0]).all()
SNGT_QHENOMENOLOGY_109: (0)           def cartesian_product(*arrays: np.ndarray):
SNGT_QHENOMENOLOGY_110: (4)               """
SNGT_QHENOMENOLOGY_111: (4)               Copied from https://stackoverflow.com/a/11146645
SNGT_QHENOMENOLOGY_112: (4)               """
SNGT_QHENOMENOLOGY_113: (4)               la = len(arrays)
SNGT_QHENOMENOLOGY_114: (4)               dtype = np.result_type(*arrays)
SNGT_QHENOMENOLOGY_115: (4)               arr = np.empty([len(a) for a in arrays] + [la],
dtype=dtype)
SNGT_QHENOMENOLOGY_116: (4)               for i, a in enumerate(np.ix_(*arrays)):
SNGT_QHENOMENOLOGY_117: (8)                   arr[..., i] = a
SNGT_QHENOMENOLOGY_118: (4)               return arr.reshape(-1, la)
SNGT_QHENOMENOLOGY_119: (0)           def hash_obj(obj: object) -> int:
SNGT_QHENOMENOLOGY_120: (4)               if isinstance(obj, dict):
SNGT_QHENOMENOLOGY_121: (8)                   return hash(tuple(sorted([
SNGT_QHENOMENOLOGY_122: (12)                      (hash_obj(k), hash_obj(v)) for k, v in
obj.items()
SNGT_QHENOMENOLOGY_123: (8)                   ])))
SNGT_QHENOMENOLOGY_124: (4)               if isinstance(obj, set):
SNGT_QHENOMENOLOGY_125: (8)                   return hash(tuple(sorted(hash_obj(e) for e in
obj)))
SNGT_QHENOMENOLOGY_126: (4)               if isinstance(obj, (tuple, list)):
SNGT_QHENOMENOLOGY_127: (8)                   return hash(tuple(hash_obj(e) for e in obj))
SNGT_QHENOMENOLOGY_128: (4)               if isinstance(obj, Color):
SNGT_QHENOMENOLOGY_129: (8)                   return hash(obj.get_rgb())
SNGT_QHENOMENOLOGY_130: (4)               return hash(obj)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 89 - space_ops.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from functools import reduce
SNGT_QHENOMENOLOGY_3: (0)             import math
SNGT_QHENOMENOLOGY_4: (0)             import operator as op
SNGT_QHENOMENOLOGY_5: (0)             import platform
SNGT_QHENOMENOLOGY_6: (0)             from mapbox_earcut import triangulate_float32 as earcut
SNGT_QHENOMENOLOGY_7: (0)             import numpy as np
SNGT_QHENOMENOLOGY_8: (0)             from scipy.spatial.transform import Rotation
SNGT_QHENOMENOLOGY_9: (0)             from tqdm.auto import tqdm as ProgressDisplay
SNGT_QHENOMENOLOGY_10: (0)            from manimlib.constants import DOWN, OUT, RIGHT, UP
SNGT_QHENOMENOLOGY_11: (0)            from manimlib.constants import PI, TAU
SNGT_QHENOMENOLOGY_12: (0)            from manimlib.utils.iterables import adjacent_pairs
SNGT_QHENOMENOLOGY_13: (0)            from manimlib.utils.simple_functions import clip
SNGT_QHENOMENOLOGY_14: (0)            from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_15: (0)            if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_16: (4)                from typing import Callable, Sequence, List, Tuple
```

```
SNGT_QHENOMENOLOGY_17: (4)                from manimlib.typing import Vect2, Vect3, Vect4, VectN,
Matrix3x3, Vect3Array, Vect2Array
SNGT_QHENOMENOLOGY_18: (0)            def cross(
SNGT_QHENOMENOLOGY_19: (4)                v1: Vect3 | List[float],
SNGT_QHENOMENOLOGY_20: (4)                v2: Vect3 | List[float],
SNGT_QHENOMENOLOGY_21: (4)                out: np.ndarray | None = None
SNGT_QHENOMENOLOGY_22: (0)            ) -> Vect3 | Vect3Array:
SNGT_QHENOMENOLOGY_23: (4)                is2d = isinstance(v1, np.ndarray) and len(v1.shape) ==
2
SNGT_QHENOMENOLOGY_24: (4)                if is2d:
SNGT_QHENOMENOLOGY_25: (8)                    x1, y1, z1 = v1[:, 0], v1[:, 1], v1[:, 2]
SNGT_QHENOMENOLOGY_26: (8)                    x2, y2, z2 = v2[:, 0], v2[:, 1], v2[:, 2]
SNGT_QHENOMENOLOGY_27: (4)                else:
SNGT_QHENOMENOLOGY_28: (8)                    x1, y1, z1 = v1
SNGT_QHENOMENOLOGY_29: (8)                    x2, y2, z2 = v2
SNGT_QHENOMENOLOGY_30: (4)                if out is None:
SNGT_QHENOMENOLOGY_31: (8)                    out = np.empty(np.shape(v1))
SNGT_QHENOMENOLOGY_32: (4)                out.T[:] = [
SNGT_QHENOMENOLOGY_33: (8)                    y1 * z2 - z1 * y2,
SNGT_QHENOMENOLOGY_34: (8)                    z1 * x2 - x1 * z2,
SNGT_QHENOMENOLOGY_35: (8)                    x1 * y2 - y1 * x2,
SNGT_QHENOMENOLOGY_36: (4)                ]
SNGT_QHENOMENOLOGY_37: (4)                return out
SNGT_QHENOMENOLOGY_38: (0)            def get_norm(vect: VectN | List[float]) -> float:
SNGT_QHENOMENOLOGY_39: (4)                return sum((x**2 for x in vect))**0.5
SNGT_QHENOMENOLOGY_40: (0)            def normalize(
SNGT_QHENOMENOLOGY_41: (4)                vect: VectN | List[float],
SNGT_QHENOMENOLOGY_42: (4)                fall_back: VectN | List[float] | None = None
SNGT_QHENOMENOLOGY_43: (0)            ) -> VectN:
SNGT_QHENOMENOLOGY_44: (4)                norm = get_norm(vect)
SNGT_QHENOMENOLOGY_45: (4)                if norm > 0:
SNGT_QHENOMENOLOGY_46: (8)                    return np.array(vect) / norm
SNGT_QHENOMENOLOGY_47: (4)                elif fall_back is not None:
SNGT_QHENOMENOLOGY_48: (8)                    return np.array(fall_back)
SNGT_QHENOMENOLOGY_49: (4)                else:
SNGT_QHENOMENOLOGY_50: (8)                    return np.zeros(len(vect))
SNGT_QHENOMENOLOGY_51: (0)            def poly_line_length(points):
SNGT_QHENOMENOLOGY_52: (4)                """
SNGT_QHENOMENOLOGY_53: (4)                Return the sum of the lengths between adjacent points
SNGT_QHENOMENOLOGY_54: (4)                """
SNGT_QHENOMENOLOGY_55: (4)                diffs = points[1:] - points[:-1]
SNGT_QHENOMENOLOGY_56: (4)                return np.sqrt((diffs**2).sum(1)).sum()
SNGT_QHENOMENOLOGY_57: (0)            def quaternion_mult(*quats: Vect4) -> Vect4:
SNGT_QHENOMENOLOGY_58: (4)                """
SNGT_QHENOMENOLOGY_59: (4)                Inputs are treated as quaternions, where the real part
is the
SNGT_QHENOMENOLOGY_60: (4)                last entry, so as to follow the scipy Rotation
conventions.
SNGT_QHENOMENOLOGY_61: (4)                """
SNGT_QHENOMENOLOGY_62: (4)                if len(quats) == 0:
SNGT_QHENOMENOLOGY_63: (8)                    return np.array([0, 0, 0, 1])
SNGT_QHENOMENOLOGY_64: (4)                result = np.array(quats[0])
SNGT_QHENOMENOLOGY_65: (4)                for next_quat in quats[1:]:
SNGT_QHENOMENOLOGY_66: (8)                    x1, y1, z1, w1 = result
SNGT_QHENOMENOLOGY_67: (8)                    x2, y2, z2, w2 = next_quat
SNGT_QHENOMENOLOGY_68: (8)                    result[:] = [
SNGT_QHENOMENOLOGY_69: (12)                       w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2,
SNGT_QHENOMENOLOGY_70: (12)                       w1 * y2 + y1 * w2 + z1 * x2 - x1 * z2,
SNGT_QHENOMENOLOGY_71: (12)                       w1 * z2 + z1 * w2 + x1 * y2 - y1 * x2,
SNGT_QHENOMENOLOGY_72: (12)                       w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2,
SNGT_QHENOMENOLOGY_73: (8)                    ]
SNGT_QHENOMENOLOGY_74: (4)                return result
SNGT_QHENOMENOLOGY_75: (0)            def quaternion_from_angle_axis(
SNGT_QHENOMENOLOGY_76: (4)                angle: float,
SNGT_QHENOMENOLOGY_77: (4)                axis: Vect3,
SNGT_QHENOMENOLOGY_78: (0)            ) -> Vect4:
SNGT_QHENOMENOLOGY_79: (4)                return Rotation.from_rotvec(angle *
normalize(axis)).as_quat()
SNGT_QHENOMENOLOGY_80: (0)            def angle_axis_from_quaternion(quat: Vect4) -> Tuple[float,
```

```
Vect3]:
SNGT_QHENOMENOLOGY_81: (4)                 rot_vec = Rotation.from_quat(quat).as_rotvec()
SNGT_QHENOMENOLOGY_82: (4)                 norm = get_norm(rot_vec)
SNGT_QHENOMENOLOGY_83: (4)                 return norm, rot_vec / norm
SNGT_QHENOMENOLOGY_84: (0)             def quaternion_conjugate(quaternion: Vect4) -> Vect4:
SNGT_QHENOMENOLOGY_85: (4)                 result = np.array(quaternion)
SNGT_QHENOMENOLOGY_86: (4)                 result[:3] *= -1
SNGT_QHENOMENOLOGY_87: (4)                 return result
SNGT_QHENOMENOLOGY_88: (0)             def rotate_vector(
SNGT_QHENOMENOLOGY_89: (4)                 vector: Vect3,
SNGT_QHENOMENOLOGY_90: (4)                 angle: float,
SNGT_QHENOMENOLOGY_91: (4)                 axis: Vect3 = OUT
SNGT_QHENOMENOLOGY_92: (0)             ) -> Vect3:
SNGT_QHENOMENOLOGY_93: (4)                 rot = Rotation.from_rotvec(angle * normalize(axis))
SNGT_QHENOMENOLOGY_94: (4)                 return np.dot(vector, rot.as_matrix().T)
SNGT_QHENOMENOLOGY_95: (0)             def rotate_vector_2d(vector: Vect2, angle: float) -> Vect2:
SNGT_QHENOMENOLOGY_96: (4)                 z = complex(*vector) * np.exp(complex(0, angle))
SNGT_QHENOMENOLOGY_97: (4)                 return np.array([z.real, z.imag])
SNGT_QHENOMENOLOGY_98: (0)             def rotation_matrix_transpose_from_quaternion(quat: Vect4)
-> Matrix3x3:
SNGT_QHENOMENOLOGY_99: (4)                 return Rotation.from_quat(quat).as_matrix()
SNGT_QHENOMENOLOGY_100: (0)            def rotation_matrix_from_quaternion(quat: Vect4) ->
Matrix3x3:
SNGT_QHENOMENOLOGY_101: (4)                return
np.transpose(rotation_matrix_transpose_from_quaternion(quat))
SNGT_QHENOMENOLOGY_102: (0)            def rotation_matrix(angle: float, axis: Vect3) ->
Matrix3x3:
SNGT_QHENOMENOLOGY_103: (4)                """
SNGT_QHENOMENOLOGY_104: (4)                Rotation in R^3 about a specified axis of rotation.
SNGT_QHENOMENOLOGY_105: (4)                """
SNGT_QHENOMENOLOGY_106: (4)                return Rotation.from_rotvec(angle *
normalize(axis)).as_matrix()
SNGT_QHENOMENOLOGY_107: (0)            def rotation_matrix_transpose(angle: float, axis: Vect3) ->
Matrix3x3:
SNGT_QHENOMENOLOGY_108: (4)                return rotation_matrix(angle, axis).T
SNGT_QHENOMENOLOGY_109: (0)            def rotation_about_z(angle: float) -> Matrix3x3:
SNGT_QHENOMENOLOGY_110: (4)                cos_a = math.cos(angle)
SNGT_QHENOMENOLOGY_111: (4)                sin_a = math.sin(angle)
SNGT_QHENOMENOLOGY_112: (4)                return np.array([
SNGT_QHENOMENOLOGY_113: (8)                    [cos_a, -sin_a, 0],
SNGT_QHENOMENOLOGY_114: (8)                    [sin_a, cos_a, 0],
SNGT_QHENOMENOLOGY_115: (8)                    [0, 0, 1]
SNGT_QHENOMENOLOGY_116: (4)                ])
SNGT_QHENOMENOLOGY_117: (0)            def rotation_between_vectors(v1: Vect3, v2: Vect3) ->
Matrix3x3:
SNGT_QHENOMENOLOGY_118: (4)                atol = 1e-8
SNGT_QHENOMENOLOGY_119: (4)                if get_norm(v1 - v2) < atol:
SNGT_QHENOMENOLOGY_120: (8)                    return np.identity(3)
SNGT_QHENOMENOLOGY_121: (4)                axis = cross(v1, v2)
SNGT_QHENOMENOLOGY_122: (4)                if get_norm(axis) < atol:
SNGT_QHENOMENOLOGY_123: (8)                    axis = cross(v1, RIGHT)
SNGT_QHENOMENOLOGY_124: (4)                if get_norm(axis) < atol:
SNGT_QHENOMENOLOGY_125: (8)                    axis = cross(v1, UP)
SNGT_QHENOMENOLOGY_126: (4)                return rotation_matrix(
SNGT_QHENOMENOLOGY_127: (8)                    angle=angle_between_vectors(v1, v2),
SNGT_QHENOMENOLOGY_128: (8)                    axis=axis,
SNGT_QHENOMENOLOGY_129: (4)                )
SNGT_QHENOMENOLOGY_130: (0)            def z_to_vector(vector: Vect3) -> Matrix3x3:
SNGT_QHENOMENOLOGY_131: (4)                return rotation_between_vectors(OUT, vector)
SNGT_QHENOMENOLOGY_132: (0)            def angle_of_vector(vector: Vect2 | Vect3) -> float:
SNGT_QHENOMENOLOGY_133: (4)                """
SNGT_QHENOMENOLOGY_134: (4)                Returns polar coordinate theta when vector is project
on xy plane
SNGT_QHENOMENOLOGY_135: (4)                """
SNGT_QHENOMENOLOGY_136: (4)                return math.atan2(vector[1], vector[0])
SNGT_QHENOMENOLOGY_137: (0)            def angle_between_vectors(v1: VectN, v2: VectN) -> float:
SNGT_QHENOMENOLOGY_138: (4)                """
SNGT_QHENOMENOLOGY_139: (4)                Returns the angle between two 3D vectors.
SNGT_QHENOMENOLOGY_140: (4)                This angle will always be btw 0 and pi
```

```
SNGT_QHENOMENOLOGY_141: (4)                 """
SNGT_QHENOMENOLOGY_142: (4)                 n1 = get_norm(v1)
SNGT_QHENOMENOLOGY_143: (4)                 n2 = get_norm(v2)
SNGT_QHENOMENOLOGY_144: (4)                 if n1 == 0 or n2 == 0:
SNGT_QHENOMENOLOGY_145: (8)                     return 0
SNGT_QHENOMENOLOGY_146: (4)                 cos_angle = np.dot(v1, v2) / np.float64(n1 * n2)
SNGT_QHENOMENOLOGY_147: (4)                 return math.acos(clip(cos_angle, -1, 1))
SNGT_QHENOMENOLOGY_148: (0)             def project_along_vector(point: Vect3, vector: Vect3) ->
Vect3:
SNGT_QHENOMENOLOGY_149: (4)                 matrix = np.identity(3) - np.outer(vector, vector)
SNGT_QHENOMENOLOGY_150: (4)                 return np.dot(point, matrix.T)
SNGT_QHENOMENOLOGY_151: (0)             def normalize_along_axis(
SNGT_QHENOMENOLOGY_152: (4)                 array: np.ndarray,
SNGT_QHENOMENOLOGY_153: (4)                 axis: int,
SNGT_QHENOMENOLOGY_154: (0)             ) -> np.ndarray:
SNGT_QHENOMENOLOGY_155: (4)                 norms = np.sqrt((array * array).sum(axis))
SNGT_QHENOMENOLOGY_156: (4)                 norms[norms == 0] = 1
SNGT_QHENOMENOLOGY_157: (4)                 return array / norms[:, np.newaxis]
SNGT_QHENOMENOLOGY_158: (0)             def get_unit_normal(
SNGT_QHENOMENOLOGY_159: (4)                 v1: Vect3,
SNGT_QHENOMENOLOGY_160: (4)                 v2: Vect3,
SNGT_QHENOMENOLOGY_161: (4)                 tol: float = 1e-6
SNGT_QHENOMENOLOGY_162: (0)             ) -> Vect3:
SNGT_QHENOMENOLOGY_163: (4)                 v1 = normalize(v1)
SNGT_QHENOMENOLOGY_164: (4)                 v2 = normalize(v2)
SNGT_QHENOMENOLOGY_165: (4)                 cp = cross(v1, v2)
SNGT_QHENOMENOLOGY_166: (4)                 cp_norm = get_norm(cp)
SNGT_QHENOMENOLOGY_167: (4)                 if cp_norm < tol:
SNGT_QHENOMENOLOGY_168: (8)                     new_cp = cross(cross(v1, OUT), v1)
SNGT_QHENOMENOLOGY_169: (8)                     new_cp_norm = get_norm(new_cp)
SNGT_QHENOMENOLOGY_170: (8)                     if new_cp_norm < tol:
SNGT_QHENOMENOLOGY_171: (12)                        return DOWN
SNGT_QHENOMENOLOGY_172: (8)                     return new_cp / new_cp_norm
SNGT_QHENOMENOLOGY_173: (4)                 return cp / cp_norm
SNGT_QHENOMENOLOGY_174: (0)             def thick_diagonal(dim: int, thickness: int = 2) ->
np.ndarray:
SNGT_QHENOMENOLOGY_175: (4)                 row_indices = np.arange(dim).repeat(dim).reshape((dim,
dim))
SNGT_QHENOMENOLOGY_176: (4)                 col_indices = np.transpose(row_indices)
SNGT_QHENOMENOLOGY_177: (4)                 return (np.abs(row_indices - col_indices) <
thickness).astype('uint8')
SNGT_QHENOMENOLOGY_178: (0)             def compass_directions(n: int = 4, start_vect: Vect3 =
RIGHT) -> Vect3:
SNGT_QHENOMENOLOGY_179: (4)                 angle = TAU / n
SNGT_QHENOMENOLOGY_180: (4)                 return np.array([
SNGT_QHENOMENOLOGY_181: (8)                     rotate_vector(start_vect, k * angle)
SNGT_QHENOMENOLOGY_182: (8)                     for k in range(n)
SNGT_QHENOMENOLOGY_183: (4)                 ])
SNGT_QHENOMENOLOGY_184: (0)             def complex_to_R3(complex_num: complex) -> Vect3:
SNGT_QHENOMENOLOGY_185: (4)                 return np.array((complex_num.real, complex_num.imag,
0))
SNGT_QHENOMENOLOGY_186: (0)             def R3_to_complex(point: Vect3) -> complex:
SNGT_QHENOMENOLOGY_187: (4)                 return complex(*point[:2])
SNGT_QHENOMENOLOGY_188: (0)             def complex_func_to_R3_func(complex_func:
Callable[[complex], complex]) -> Callable[[Vect3], Vect3]:
SNGT_QHENOMENOLOGY_189: (4)                 def result(p: Vect3):
SNGT_QHENOMENOLOGY_190: (8)                     return
complex_to_R3(complex_func(R3_to_complex(p)))
SNGT_QHENOMENOLOGY_191: (4)                 return result
SNGT_QHENOMENOLOGY_192: (0)             def center_of_mass(points: Sequence[Vect3]) -> Vect3:
SNGT_QHENOMENOLOGY_193: (4)                 return np.array(points).sum(0) / len(points)
SNGT_QHENOMENOLOGY_194: (0)             def midpoint(point1: VectN, point2: VectN) -> VectN:
SNGT_QHENOMENOLOGY_195: (4)                 return center_of_mass([point1, point2])
SNGT_QHENOMENOLOGY_196: (0)             def line_intersection(
SNGT_QHENOMENOLOGY_197: (4)                 line1: Tuple[Vect3, Vect3],
SNGT_QHENOMENOLOGY_198: (4)                 line2: Tuple[Vect3, Vect3]
SNGT_QHENOMENOLOGY_199: (0)             ) -> Vect3:
SNGT_QHENOMENOLOGY_200: (4)                 """
SNGT_QHENOMENOLOGY_201: (4)                 return intersection point of two lines,
```

```
SNGT_QHENOMENOLOGY_202: (4)                 each defined with a pair of vectors determining
SNGT_QHENOMENOLOGY_203: (4)                 the end points
SNGT_QHENOMENOLOGY_204: (4)                 """
SNGT_QHENOMENOLOGY_205: (4)                 x_diff = (line1[0][0] - line1[1][0], line2[0][0] -
line2[1][0])
SNGT_QHENOMENOLOGY_206: (4)                 y_diff = (line1[0][1] - line1[1][1], line2[0][1] -
line2[1][1])
SNGT_QHENOMENOLOGY_207: (4)                 def det(a, b):
SNGT_QHENOMENOLOGY_208: (8)                     return a[0] * b[1] - a[1] * b[0]
SNGT_QHENOMENOLOGY_209: (4)                 div = det(x_diff, y_diff)
SNGT_QHENOMENOLOGY_210: (4)                 if div == 0:
SNGT_QHENOMENOLOGY_211: (8)                     raise Exception("Lines do not intersect")
SNGT_QHENOMENOLOGY_212: (4)                 d = (det(*line1), det(*line2))
SNGT_QHENOMENOLOGY_213: (4)                 x = det(d, x_diff) / div
SNGT_QHENOMENOLOGY_214: (4)                 y = det(d, y_diff) / div
SNGT_QHENOMENOLOGY_215: (4)                 return np.array([x, y, 0])
SNGT_QHENOMENOLOGY_216: (0)             def find_intersection(
SNGT_QHENOMENOLOGY_217: (4)                 p0: Vect3 | Vect3Array,
SNGT_QHENOMENOLOGY_218: (4)                 v0: Vect3 | Vect3Array,
SNGT_QHENOMENOLOGY_219: (4)                 p1: Vect3 | Vect3Array,
SNGT_QHENOMENOLOGY_220: (4)                 v1: Vect3 | Vect3Array,
SNGT_QHENOMENOLOGY_221: (4)                 threshold: float = 1e-5,
SNGT_QHENOMENOLOGY_222: (0)             ) -> Vect3:
SNGT_QHENOMENOLOGY_223: (4)                 """
SNGT_QHENOMENOLOGY_224: (4)                 Return the intersection of a line passing through p0 in
direction v0
SNGT_QHENOMENOLOGY_225: (4)                 with one passing through p1 in direction v1.  (Or array
of intersections
SNGT_QHENOMENOLOGY_226: (4)                 from arrays of such points/directions).
SNGT_QHENOMENOLOGY_227: (4)                 For 3d values, it returns the point on the ray p0 + v0
* t closest to the
SNGT_QHENOMENOLOGY_228: (4)                 ray p1 + v1 * t
SNGT_QHENOMENOLOGY_229: (4)                 """
SNGT_QHENOMENOLOGY_230: (4)                 d = len(p0.shape)
SNGT_QHENOMENOLOGY_231: (4)                 if d == 1:
SNGT_QHENOMENOLOGY_232: (8)                     is_3d = any(arr[2] for arr in (p0, v0, p1, v1))
SNGT_QHENOMENOLOGY_233: (4)                 else:
SNGT_QHENOMENOLOGY_234: (8)                     is_3d = any(z for arr in (p0, v0, p1, v1) for z in
arr.T[2])
SNGT_QHENOMENOLOGY_235: (4)                 if not is_3d:
SNGT_QHENOMENOLOGY_236: (8)                     numer = np.array(cross2d(v1, p1 - p0))
SNGT_QHENOMENOLOGY_237: (8)                     denom = np.array(cross2d(v1, v0))
SNGT_QHENOMENOLOGY_238: (4)                 else:
SNGT_QHENOMENOLOGY_239: (8)                     cp1 = cross(v1, p1 - p0)
SNGT_QHENOMENOLOGY_240: (8)                     cp2 = cross(v1, v0)
SNGT_QHENOMENOLOGY_241: (8)                     numer = np.array((cp1 * cp1).sum(d - 1))
SNGT_QHENOMENOLOGY_242: (8)                     denom = np.array((cp1 * cp2).sum(d - 1))
SNGT_QHENOMENOLOGY_243: (4)                 denom[abs(denom) < threshold] = np.inf
SNGT_QHENOMENOLOGY_244: (4)                 ratio = numer / denom
SNGT_QHENOMENOLOGY_245: (4)                 return p0 + (ratio * v0.T).T
SNGT_QHENOMENOLOGY_246: (0)             def line_intersects_path(
SNGT_QHENOMENOLOGY_247: (4)                 start: Vect2 | Vect3,
SNGT_QHENOMENOLOGY_248: (4)                 end: Vect2 | Vect3,
SNGT_QHENOMENOLOGY_249: (4)                 path: Vect2Array | Vect3Array,
SNGT_QHENOMENOLOGY_250: (0)             ) -> bool:
SNGT_QHENOMENOLOGY_251: (4)                 """
SNGT_QHENOMENOLOGY_252: (4)                 Tests whether the line (start, end) intersects
SNGT_QHENOMENOLOGY_253: (4)                 a polygonal path defined by its vertices
SNGT_QHENOMENOLOGY_254: (4)                 """
SNGT_QHENOMENOLOGY_255: (4)                 n = len(path) - 1
SNGT_QHENOMENOLOGY_256: (4)                 p1 = np.empty((n, 2))
SNGT_QHENOMENOLOGY_257: (4)                 q1 = np.empty((n, 2))
SNGT_QHENOMENOLOGY_258: (4)                 p1[:] = start[:2]
SNGT_QHENOMENOLOGY_259: (4)                 q1[:] = end[:2]
SNGT_QHENOMENOLOGY_260: (4)                 p2 = path[:-1, :2]
SNGT_QHENOMENOLOGY_261: (4)                 q2 = path[1:, :2]
SNGT_QHENOMENOLOGY_262: (4)                 v1 = q1 - p1
SNGT_QHENOMENOLOGY_263: (4)                 v2 = q2 - p2
SNGT_QHENOMENOLOGY_264: (4)                 mis1 = cross2d(v1, p2 - p1) * cross2d(v1, q2 - p1) < 0
```

```
SNGT_QHENOMENOLOGY_265: (4)                 mis2 = cross2d(v2, p1 - p2) * cross2d(v2, q1 - p2) < 0
SNGT_QHENOMENOLOGY_266: (4)                 return bool((mis1 * mis2).any())
SNGT_QHENOMENOLOGY_267: (0)             def get_closest_point_on_line(a: VectN, b: VectN, p: VectN)
-> VectN:
SNGT_QHENOMENOLOGY_268: (4)                 """
SNGT_QHENOMENOLOGY_269: (8)                     It returns point x such that
SNGT_QHENOMENOLOGY_270: (8)                     x is on line ab and xp is perpendicular to ab.
SNGT_QHENOMENOLOGY_271: (8)                     If x lies beyond ab line, then it returns nearest
edge(a or b).
SNGT_QHENOMENOLOGY_272: (4)                 """
SNGT_QHENOMENOLOGY_273: (4)                 t = np.dot(p - b, a - b) / np.dot(a - b, a - b)
SNGT_QHENOMENOLOGY_274: (4)                 if t < 0:
SNGT_QHENOMENOLOGY_275: (8)                     t = 0
SNGT_QHENOMENOLOGY_276: (4)                 if t > 1:
SNGT_QHENOMENOLOGY_277: (8)                     t = 1
SNGT_QHENOMENOLOGY_278: (4)                 return ((t * a) + ((1 - t) * b))
SNGT_QHENOMENOLOGY_279: (0)             def get_winding_number(points: Sequence[Vect2 | Vect3]) ->
float:
SNGT_QHENOMENOLOGY_280: (4)                 total_angle = 0
SNGT_QHENOMENOLOGY_281: (4)                 for p1, p2 in adjacent_pairs(points):
SNGT_QHENOMENOLOGY_282: (8)                     d_angle = angle_of_vector(p2) - angle_of_vector(p1)
SNGT_QHENOMENOLOGY_283: (8)                     d_angle = ((d_angle + PI) % TAU) - PI
SNGT_QHENOMENOLOGY_284: (8)                     total_angle += d_angle
SNGT_QHENOMENOLOGY_285: (4)                 return total_angle / TAU
SNGT_QHENOMENOLOGY_286: (0)             def cross2d(a: Vect2 | Vect2Array, b: Vect2 | Vect2Array) -
> Vect2 | Vect2Array:
SNGT_QHENOMENOLOGY_287: (4)                 if len(a.shape) == 2:
SNGT_QHENOMENOLOGY_288: (8)                     return a[:, 0] * b[:, 1] - a[:, 1] * b[:, 0]
SNGT_QHENOMENOLOGY_289: (4)                 else:
SNGT_QHENOMENOLOGY_290: (8)                     return a[0] * b[1] - b[0] * a[1]
SNGT_QHENOMENOLOGY_291: (0)             def tri_area(
SNGT_QHENOMENOLOGY_292: (4)                 a: Vect2,
SNGT_QHENOMENOLOGY_293: (4)                 b: Vect2,
SNGT_QHENOMENOLOGY_294: (4)                 c: Vect2
SNGT_QHENOMENOLOGY_295: (0)             ) -> float:
SNGT_QHENOMENOLOGY_296: (4)                 return 0.5 * abs(
SNGT_QHENOMENOLOGY_297: (8)                     a[0] * (b[1] - c[1]) +
SNGT_QHENOMENOLOGY_298: (8)                     b[0] * (c[1] - a[1]) +
SNGT_QHENOMENOLOGY_299: (8)                     c[0] * (a[1] - b[1])
SNGT_QHENOMENOLOGY_300: (4)                 )
SNGT_QHENOMENOLOGY_301: (0)             def is_inside_triangle(
SNGT_QHENOMENOLOGY_302: (4)                 p: Vect2,
SNGT_QHENOMENOLOGY_303: (4)                 a: Vect2,
SNGT_QHENOMENOLOGY_304: (4)                 b: Vect2,
SNGT_QHENOMENOLOGY_305: (4)                 c: Vect2
SNGT_QHENOMENOLOGY_306: (0)             ) -> bool:
SNGT_QHENOMENOLOGY_307: (4)                 """
SNGT_QHENOMENOLOGY_308: (4)                 Test if point p is inside triangle abc
SNGT_QHENOMENOLOGY_309: (4)                 """
SNGT_QHENOMENOLOGY_310: (4)                 crosses = np.array([
SNGT_QHENOMENOLOGY_311: (8)                     cross2d(p - a, b - p),
SNGT_QHENOMENOLOGY_312: (8)                     cross2d(p - b, c - p),
SNGT_QHENOMENOLOGY_313: (8)                     cross2d(p - c, a - p),
SNGT_QHENOMENOLOGY_314: (4)                 ])
SNGT_QHENOMENOLOGY_315: (4)                 return bool(np.all(crosses > 0) or np.all(crosses < 0))
SNGT_QHENOMENOLOGY_316: (0)             def norm_squared(v: VectN | List[float]) -> float:
SNGT_QHENOMENOLOGY_317: (4)                 return sum(x * x for x in v)
SNGT_QHENOMENOLOGY_318: (0)             def earclip_triangulation(verts: Vect3Array | Vect2Array,
ring_ends: list[int]) -> list[int]:
SNGT_QHENOMENOLOGY_319: (4)                 """
SNGT_QHENOMENOLOGY_320: (4)                 Returns a list of indices giving a triangulation
SNGT_QHENOMENOLOGY_321: (4)                 of a polygon, potentially with holes
SNGT_QHENOMENOLOGY_322: (4)                 - verts is a numpy array of points
SNGT_QHENOMENOLOGY_323: (4)                 - ring_ends is a list of indices indicating where
SNGT_QHENOMENOLOGY_324: (4)                 the ends of new paths are
SNGT_QHENOMENOLOGY_325: (4)                 """
SNGT_QHENOMENOLOGY_326: (4)                 rings = [
SNGT_QHENOMENOLOGY_327: (8)                     list(range(e0, e1))
SNGT_QHENOMENOLOGY_328: (8)                     for e0, e1 in zip([0, *ring_ends], ring_ends)
```

```
SNGT_QHENOMENOLOGY_329: (4)                ]
SNGT_QHENOMENOLOGY_330: (4)                epsilon = 1e-6
SNGT_QHENOMENOLOGY_331: (4)                def is_in(point, ring_id):
SNGT_QHENOMENOLOGY_332: (8)                    return abs(abs(get_winding_number([i - point for i
in verts[rings[ring_id]]])) - 1) < epsilon
SNGT_QHENOMENOLOGY_333: (4)                def ring_area(ring_id):
SNGT_QHENOMENOLOGY_334: (8)                    ring = rings[ring_id]
SNGT_QHENOMENOLOGY_335: (8)                    s = 0
SNGT_QHENOMENOLOGY_336: (8)                    for i, j in zip(ring[1:], ring):
SNGT_QHENOMENOLOGY_337: (12)                       s += cross2d(verts[i], verts[j])
SNGT_QHENOMENOLOGY_338: (8)                    return abs(s) / 2
SNGT_QHENOMENOLOGY_339: (4)                for i in rings:
SNGT_QHENOMENOLOGY_340: (8)                    if len(i) < 2:
SNGT_QHENOMENOLOGY_341: (12)                       continue
SNGT_QHENOMENOLOGY_342: (8)                    verts[i[0]] += (verts[i[1]] - verts[i[0]]) *
epsilon
SNGT_QHENOMENOLOGY_343: (8)                    verts[i[-1]] += (verts[i[-2]] - verts[i[-1]]) *
epsilon
SNGT_QHENOMENOLOGY_344: (4)                right = [max(verts[rings[i], 0]) for i in
range(len(rings))]
SNGT_QHENOMENOLOGY_345: (4)                left = [min(verts[rings[i], 0]) for i in
range(len(rings))]
SNGT_QHENOMENOLOGY_346: (4)                top = [max(verts[rings[i], 1]) for i in
range(len(rings))]
SNGT_QHENOMENOLOGY_347: (4)                bottom = [min(verts[rings[i], 1]) for i in
range(len(rings))]
SNGT_QHENOMENOLOGY_348: (4)                area = [ring_area(i) for i in range(len(rings))]
SNGT_QHENOMENOLOGY_349: (4)                rings_sorted = list(range(len(rings)))
SNGT_QHENOMENOLOGY_350: (4)                rings_sorted.sort(key=lambda x: area[x], reverse=True)
SNGT_QHENOMENOLOGY_351: (4)                def is_in_fast(ring_a, ring_b):
SNGT_QHENOMENOLOGY_352: (8)                    return reduce(op.and_, (
SNGT_QHENOMENOLOGY_353: (12)                       left[ring_b] <= left[ring_a] <= right[ring_a]
<= right[ring_b],
SNGT_QHENOMENOLOGY_354: (12)                       bottom[ring_b] <= bottom[ring_a] <= top[ring_a]
<= top[ring_b],
SNGT_QHENOMENOLOGY_355: (12)                       is_in(verts[rings[ring_a][0]], ring_b)
SNGT_QHENOMENOLOGY_356: (8)                    ))
SNGT_QHENOMENOLOGY_357: (4)                chilren = [[] for i in rings]
SNGT_QHENOMENOLOGY_358: (4)                ringenum = ProgressDisplay(
SNGT_QHENOMENOLOGY_359: (8)                    enumerate(rings_sorted),
SNGT_QHENOMENOLOGY_360: (8)                    total=len(rings),
SNGT_QHENOMENOLOGY_361: (8)                    leave=False,
SNGT_QHENOMENOLOGY_362: (8)                    ascii=True if platform.system() == 'Windows' else
None,
SNGT_QHENOMENOLOGY_363: (8)                    dynamic_ncols=True,
SNGT_QHENOMENOLOGY_364: (8)                    desc="SVG Triangulation",
SNGT_QHENOMENOLOGY_365: (8)                    delay=3,
SNGT_QHENOMENOLOGY_366: (4)                )
SNGT_QHENOMENOLOGY_367: (4)                for idx, i in ringenum:
SNGT_QHENOMENOLOGY_368: (8)                    for j in rings_sorted[:idx][::-1]:
SNGT_QHENOMENOLOGY_369: (12)                       if is_in_fast(i, j):
SNGT_QHENOMENOLOGY_370: (16)                           chilren[j].append(i)
SNGT_QHENOMENOLOGY_371: (16)                           break
SNGT_QHENOMENOLOGY_372: (4)                res = []
SNGT_QHENOMENOLOGY_373: (4)                used = [False] * len(rings)
SNGT_QHENOMENOLOGY_374: (4)                for i in rings_sorted:
SNGT_QHENOMENOLOGY_375: (8)                    if used[i]:
SNGT_QHENOMENOLOGY_376: (12)                       continue
SNGT_QHENOMENOLOGY_377: (8)                    v = rings[i]
SNGT_QHENOMENOLOGY_378: (8)                    ring_ends = [len(v)]
SNGT_QHENOMENOLOGY_379: (8)                    for j in chilren[i]:
SNGT_QHENOMENOLOGY_380: (12)                       used[j] = True
SNGT_QHENOMENOLOGY_381: (12)                       v += rings[j]
SNGT_QHENOMENOLOGY_382: (12)                       ring_ends.append(len(v))
SNGT_QHENOMENOLOGY_383: (8)                    res += [v[i] for i in earcut(verts[v, :2],
ring_ends)]
SNGT_QHENOMENOLOGY_384: (4)                return res
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
```

```
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 90 - family_ops.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_3: (0)             if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_4: (4)                 from typing import Iterable, List, Set, Tuple
SNGT_QHENOMENOLOGY_5: (4)                 from manimlib.mobject.mobject import Mobject
SNGT_QHENOMENOLOGY_6: (0)             def extract_mobject_family_members(
SNGT_QHENOMENOLOGY_7: (4)                 mobject_list: Iterable[Mobject],
SNGT_QHENOMENOLOGY_8: (4)                 exclude_pointless: bool = False
SNGT_QHENOMENOLOGY_9: (0)             ) -> list[Mobject]:
SNGT_QHENOMENOLOGY_10: (4)                return [
SNGT_QHENOMENOLOGY_11: (8)                    sm
SNGT_QHENOMENOLOGY_12: (8)                    for mob in mobject_list
SNGT_QHENOMENOLOGY_13: (8)                    for sm in mob.get_family()
SNGT_QHENOMENOLOGY_14: (8)                    if (not exclude_pointless) or sm.has_points()
SNGT_QHENOMENOLOGY_15: (4)                ]
SNGT_QHENOMENOLOGY_16: (0)            def recursive_mobject_remove(mobjects: List[Mobject],
to_remove: Set[Mobject]) -> Tuple[List[Mobject], bool]:
SNGT_QHENOMENOLOGY_17: (4)                """
SNGT_QHENOMENOLOGY_18: (4)                Takes in a list of mobjects, together with a set of
mobjects to remove.
SNGT_QHENOMENOLOGY_19: (4)                The first component of what's removed is a new list
such that any mobject
SNGT_QHENOMENOLOGY_20: (4)                with one of the elements from `to_remove` in its family
is no longer in
SNGT_QHENOMENOLOGY_21: (4)                the list, and in its place are its family members which
aren't in `to_remove`
SNGT_QHENOMENOLOGY_22: (4)                The second component is a boolean value indicating
whether any removals were made
SNGT_QHENOMENOLOGY_23: (4)                """
SNGT_QHENOMENOLOGY_24: (4)                result = []
SNGT_QHENOMENOLOGY_25: (4)                found_in_list = False
SNGT_QHENOMENOLOGY_26: (4)                for mob in mobjects:
SNGT_QHENOMENOLOGY_27: (8)                    if mob in to_remove:
SNGT_QHENOMENOLOGY_28: (12)                       found_in_list = True
SNGT_QHENOMENOLOGY_29: (12)                       continue
SNGT_QHENOMENOLOGY_30: (8)                    sub_list, found_in_submobjects =
recursive_mobject_remove(
SNGT_QHENOMENOLOGY_31: (12)                       mob.submobjects, to_remove
SNGT_QHENOMENOLOGY_32: (8)                    )
SNGT_QHENOMENOLOGY_33: (8)                    if found_in_submobjects:
SNGT_QHENOMENOLOGY_34: (12)                       result.extend(sub_list)
SNGT_QHENOMENOLOGY_35: (12)                       found_in_list = True
SNGT_QHENOMENOLOGY_36: (8)                    else:
SNGT_QHENOMENOLOGY_37: (12)                       result.append(mob)
SNGT_QHENOMENOLOGY_38: (4)                return result, found_in_list
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 91 - directories.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import os
SNGT_QHENOMENOLOGY_3: (0)             import tempfile
SNGT_QHENOMENOLOGY_4: (0)             import appdirs
SNGT_QHENOMENOLOGY_5: (0)             from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_6: (0)             from manimlib.config import get_manim_dir
SNGT_QHENOMENOLOGY_7: (0)             from manimlib.utils.file_ops import guarantee_existence
SNGT_QHENOMENOLOGY_8: (0)             def get_directories() -> dict[str, str]:
SNGT_QHENOMENOLOGY_9: (4)                 return manim_config.directories
SNGT_QHENOMENOLOGY_10: (0)            def get_cache_dir() -> str:
SNGT_QHENOMENOLOGY_11: (4)                return get_directories()["cache"] or
appdirs.user_cache_dir("manim")
SNGT_QHENOMENOLOGY_12: (0)            def get_temp_dir() -> str:
SNGT_QHENOMENOLOGY_13: (4)                return get_directories()["temporary_storage"] or
tempfile.gettempdir()
SNGT_QHENOMENOLOGY_14: (0)            def get_downloads_dir() -> str:
```

```
SNGT_QHENOMENOLOGY_15: (4)                 return get_directories()["downloads"] or
appdirs.user_cache_dir("manim_downloads")
SNGT_QHENOMENOLOGY_16: (0)             def get_output_dir() -> str:
SNGT_QHENOMENOLOGY_17: (4)                 return guarantee_existence(get_directories()["output"])
SNGT_QHENOMENOLOGY_18: (0)             def get_raster_image_dir() -> str:
SNGT_QHENOMENOLOGY_19: (4)                 return get_directories()["raster_images"]
SNGT_QHENOMENOLOGY_20: (0)             def get_vector_image_dir() -> str:
SNGT_QHENOMENOLOGY_21: (4)                 return get_directories()["vector_images"]
SNGT_QHENOMENOLOGY_22: (0)             def get_sound_dir() -> str:
SNGT_QHENOMENOLOGY_23: (4)                 return get_directories()["sounds"]
SNGT_QHENOMENOLOGY_24: (0)             def get_shader_dir() -> str:
SNGT_QHENOMENOLOGY_25: (4)                 return os.path.join(get_manim_dir(), "manimlib",
"shaders")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 92 - rate_functions.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              import numpy as np
SNGT_QHENOMENOLOGY_3: (0)              from manimlib.utils.bezier import bezier
SNGT_QHENOMENOLOGY_4: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_5: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_6: (4)                  from typing import Callable
SNGT_QHENOMENOLOGY_7: (0)              def linear(t: float) -> float:
SNGT_QHENOMENOLOGY_8: (4)                  return t
SNGT_QHENOMENOLOGY_9: (0)              def smooth(t: float) -> float:
SNGT_QHENOMENOLOGY_10: (4)                 s = 1 - t
SNGT_QHENOMENOLOGY_11: (4)                 return (t**3) * (10 * s * s + 5 * s * t + t * t)
SNGT_QHENOMENOLOGY_12: (0)             def rush_into(t: float) -> float:
SNGT_QHENOMENOLOGY_13: (4)                 return 2 * smooth(0.5 * t)
SNGT_QHENOMENOLOGY_14: (0)             def rush_from(t: float) -> float:
SNGT_QHENOMENOLOGY_15: (4)                 return 2 * smooth(0.5 * (t + 1)) - 1
SNGT_QHENOMENOLOGY_16: (0)             def slow_into(t: float) -> float:
SNGT_QHENOMENOLOGY_17: (4)                 return np.sqrt(1 - (1 - t) * (1 - t))
SNGT_QHENOMENOLOGY_18: (0)             def double_smooth(t: float) -> float:
SNGT_QHENOMENOLOGY_19: (4)                 if t < 0.5:
SNGT_QHENOMENOLOGY_20: (8)                     return 0.5 * smooth(2 * t)
SNGT_QHENOMENOLOGY_21: (4)                 else:
SNGT_QHENOMENOLOGY_22: (8)                     return 0.5 * (1 + smooth(2 * t - 1))
SNGT_QHENOMENOLOGY_23: (0)             def there_and_back(t: float) -> float:
SNGT_QHENOMENOLOGY_24: (4)                 new_t = 2 * t if t < 0.5 else 2 * (1 - t)
SNGT_QHENOMENOLOGY_25: (4)                 return smooth(new_t)
SNGT_QHENOMENOLOGY_26: (0)             def there_and_back_with_pause(t: float, pause_ratio: float
= 1. / 3) -> float:
SNGT_QHENOMENOLOGY_27: (4)                 a = 2. / (1. - pause_ratio)
SNGT_QHENOMENOLOGY_28: (4)                 if t < 0.5 - pause_ratio / 2:
SNGT_QHENOMENOLOGY_29: (8)                     return smooth(a * t)
SNGT_QHENOMENOLOGY_30: (4)                 elif t < 0.5 + pause_ratio / 2:
SNGT_QHENOMENOLOGY_31: (8)                     return 1
SNGT_QHENOMENOLOGY_32: (4)                 else:
SNGT_QHENOMENOLOGY_33: (8)                     return smooth(a - a * t)
SNGT_QHENOMENOLOGY_34: (0)             def running_start(t: float, pull_factor: float = -0.5) ->
float:
SNGT_QHENOMENOLOGY_35: (4)                 return bezier([0, 0, pull_factor, pull_factor, 1, 1,
1])(t)
SNGT_QHENOMENOLOGY_36: (0)             def overshoot(t: float, pull_factor: float = 1.5) -> float:
SNGT_QHENOMENOLOGY_37: (4)                 return bezier([0, 0, pull_factor, pull_factor, 1, 1])
(t)
SNGT_QHENOMENOLOGY_38: (0)             def not_quite_there(
SNGT_QHENOMENOLOGY_39: (4)                 func: Callable[[float], float] = smooth,
SNGT_QHENOMENOLOGY_40: (4)                 proportion: float = 0.7
SNGT_QHENOMENOLOGY_41: (0)             ) -> Callable[[float], float]:
SNGT_QHENOMENOLOGY_42: (4)                 def result(t):
SNGT_QHENOMENOLOGY_43: (8)                     return proportion * func(t)
SNGT_QHENOMENOLOGY_44: (4)                 return result
SNGT_QHENOMENOLOGY_45: (0)             def wiggle(t: float, wiggles: float = 2) -> float:
SNGT_QHENOMENOLOGY_46: (4)                 return there_and_back(t) * np.sin(wiggles * np.pi * t)
SNGT_QHENOMENOLOGY_47: (0)             def squish_rate_func(
```

```
SNGT_QHENOMENOLOGY_48: (4)                 func: Callable[[float], float],
SNGT_QHENOMENOLOGY_49: (4)                 a: float = 0.4,
SNGT_QHENOMENOLOGY_50: (4)                 b: float = 0.6
SNGT_QHENOMENOLOGY_51: (0)             ) -> Callable[[float], float]:
SNGT_QHENOMENOLOGY_52: (4)                 def result(t):
SNGT_QHENOMENOLOGY_53: (8)                     if a == b:
SNGT_QHENOMENOLOGY_54: (12)                        return a
SNGT_QHENOMENOLOGY_55: (8)                     elif t < a:
SNGT_QHENOMENOLOGY_56: (12)                        return func(0)
SNGT_QHENOMENOLOGY_57: (8)                     elif t > b:
SNGT_QHENOMENOLOGY_58: (12)                        return func(1)
SNGT_QHENOMENOLOGY_59: (8)                     else:
SNGT_QHENOMENOLOGY_60: (12)                        return func((t - a) / (b - a))
SNGT_QHENOMENOLOGY_61: (4)                 return result
SNGT_QHENOMENOLOGY_62: (0)             def lingering(t: float) -> float:
SNGT_QHENOMENOLOGY_63: (4)                 return squish_rate_func(lambda t: t, 0, 0.8)(t)
SNGT_QHENOMENOLOGY_64: (0)             def exponential_decay(t: float, half_life: float = 0.1) ->
float:
SNGT_QHENOMENOLOGY_65: (4)                 return 1 - np.exp(-t / half_life)
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 93 - simple_functions.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)              from functools import lru_cache
SNGT_QHENOMENOLOGY_3: (0)              import hashlib
SNGT_QHENOMENOLOGY_4: (0)              import inspect
SNGT_QHENOMENOLOGY_5: (0)              import math
SNGT_QHENOMENOLOGY_6: (0)              import numpy as np
SNGT_QHENOMENOLOGY_7: (0)              from typing import TYPE_CHECKING
SNGT_QHENOMENOLOGY_8: (0)              if TYPE_CHECKING:
SNGT_QHENOMENOLOGY_9: (4)                  from typing import Callable, TypeVar, Iterable
SNGT_QHENOMENOLOGY_10: (4)                 from manimlib.typing import FloatArray
SNGT_QHENOMENOLOGY_11: (4)                 Scalable = TypeVar("Scalable", float, FloatArray)
SNGT_QHENOMENOLOGY_12: (0)             def sigmoid(x: float | FloatArray):
SNGT_QHENOMENOLOGY_13: (4)                 return 1.0 / (1 + np.exp(-x))
SNGT_QHENOMENOLOGY_14: (0)             @lru_cache(maxsize=10)
SNGT_QHENOMENOLOGY_15: (0)             def choose(n: int, k: int) -> int:
SNGT_QHENOMENOLOGY_16: (4)                 return math.comb(n, k)
SNGT_QHENOMENOLOGY_17: (0)             def gen_choose(n: int, r: int) -> int:
SNGT_QHENOMENOLOGY_18: (4)                 return int(np.prod(range(n, n - r, -1)) /
math.factorial(r))
SNGT_QHENOMENOLOGY_19: (0)             def get_num_args(function: Callable) -> int:
SNGT_QHENOMENOLOGY_20: (4)                 return function.__code__.co_argcount
SNGT_QHENOMENOLOGY_21: (0)             def get_parameters(function: Callable) -> Iterable[str]:
SNGT_QHENOMENOLOGY_22: (4)                 return inspect.signature(function).parameters.keys()
SNGT_QHENOMENOLOGY_23: (0)             def clip(a: float, min_a: float, max_a: float) -> float:
SNGT_QHENOMENOLOGY_24: (4)                 if a < min_a:
SNGT_QHENOMENOLOGY_25: (8)                     return min_a
SNGT_QHENOMENOLOGY_26: (4)                 elif a > max_a:
SNGT_QHENOMENOLOGY_27: (8)                     return max_a
SNGT_QHENOMENOLOGY_28: (4)                 return a
SNGT_QHENOMENOLOGY_29: (0)             def arr_clip(arr: np.ndarray, min_a: float, max_a: float) -
> np.ndarray:
SNGT_QHENOMENOLOGY_30: (4)                 arr[arr < min_a] = min_a
SNGT_QHENOMENOLOGY_31: (4)                 arr[arr > max_a] = max_a
SNGT_QHENOMENOLOGY_32: (4)                 return arr
SNGT_QHENOMENOLOGY_33: (0)             def fdiv(a: Scalable, b: Scalable, zero_over_zero_value:
Scalable | None = None) -> Scalable:
SNGT_QHENOMENOLOGY_34: (4)                 """
SNGT_QHENOMENOLOGY_35: (4)                 Less heavyweight name for np.true_divide, enabling
SNGT_QHENOMENOLOGY_36: (4)                 default behavior for 0/0
SNGT_QHENOMENOLOGY_37: (4)                 """
SNGT_QHENOMENOLOGY_38: (4)                 if zero_over_zero_value is not None:
SNGT_QHENOMENOLOGY_39: (8)                     out = np.full_like(a, zero_over_zero_value)
SNGT_QHENOMENOLOGY_40: (8)                     where = np.logical_or(a != 0, b != 0)
SNGT_QHENOMENOLOGY_41: (4)                 else:
SNGT_QHENOMENOLOGY_42: (8)                     out = None
```

```
SNGT_QHENOMENOLOGY_43: (8)                    where = True
SNGT_QHENOMENOLOGY_44: (4)                return np.true_divide(a, b, out=out, where=where)
SNGT_QHENOMENOLOGY_45: (0)            def binary_search(
SNGT_QHENOMENOLOGY_46: (4)                function: Callable[[float], float],
SNGT_QHENOMENOLOGY_47: (4)                target: float,
SNGT_QHENOMENOLOGY_48: (4)                lower_bound: float,
SNGT_QHENOMENOLOGY_49: (4)                upper_bound: float,
SNGT_QHENOMENOLOGY_50: (4)                tolerance:float = 1e-4
SNGT_QHENOMENOLOGY_51: (0)            ) -> float | None:
SNGT_QHENOMENOLOGY_52: (4)                lh = lower_bound
SNGT_QHENOMENOLOGY_53: (4)                rh = upper_bound
SNGT_QHENOMENOLOGY_54: (4)                mh = (lh + rh) / 2
SNGT_QHENOMENOLOGY_55: (4)                while abs(rh - lh) > tolerance:
SNGT_QHENOMENOLOGY_56: (8)                    lx, mx, rx = [function(h) for h in (lh, mh, rh)]
SNGT_QHENOMENOLOGY_57: (8)                    if lx == target:
SNGT_QHENOMENOLOGY_58: (12)                       return lx
SNGT_QHENOMENOLOGY_59: (8)                    if rx == target:
SNGT_QHENOMENOLOGY_60: (12)                       return rx
SNGT_QHENOMENOLOGY_61: (8)                    if lx <= target and rx >= target:
SNGT_QHENOMENOLOGY_62: (12)                       if mx > target:
SNGT_QHENOMENOLOGY_63: (16)                           rh = mh
SNGT_QHENOMENOLOGY_64: (12)                       else:
SNGT_QHENOMENOLOGY_65: (16)                           lh = mh
SNGT_QHENOMENOLOGY_66: (8)                    elif lx > target and rx < target:
SNGT_QHENOMENOLOGY_67: (12)                       lh, rh = rh, lh
SNGT_QHENOMENOLOGY_68: (8)                    else:
SNGT_QHENOMENOLOGY_69: (12)                       return None
SNGT_QHENOMENOLOGY_70: (8)                    mh = (lh + rh) / 2
SNGT_QHENOMENOLOGY_71: (4)                return mh
SNGT_QHENOMENOLOGY_72: (0)            def hash_string(string: str, n_bytes=16) -> str:
SNGT_QHENOMENOLOGY_73: (4)                hasher = hashlib.sha256(string.encode())
SNGT_QHENOMENOLOGY_74: (4)                return hasher.hexdigest()[:n_bytes]
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 94 - tex_file_writing.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)             from __future__ import annotations
SNGT_QHENOMENOLOGY_2: (0)             import os
SNGT_QHENOMENOLOGY_3: (0)             import re
SNGT_QHENOMENOLOGY_4: (0)             import yaml
SNGT_QHENOMENOLOGY_5: (0)             import subprocess
SNGT_QHENOMENOLOGY_6: (0)             from functools import lru_cache
SNGT_QHENOMENOLOGY_7: (0)             from pathlib import Path
SNGT_QHENOMENOLOGY_8: (0)             import tempfile
SNGT_QHENOMENOLOGY_9: (0)             from manimlib.utils.cache import cache_on_disk
SNGT_QHENOMENOLOGY_10: (0)            from manimlib.config import manim_config
SNGT_QHENOMENOLOGY_11: (0)            from manimlib.config import get_manim_dir
SNGT_QHENOMENOLOGY_12: (0)            from manimlib.logger import log
SNGT_QHENOMENOLOGY_13: (0)            from manimlib.utils.simple_functions import hash_string
SNGT_QHENOMENOLOGY_14: (0)            def get_tex_template_config(template_name: str) ->
dict[str, str]:
SNGT_QHENOMENOLOGY_15: (4)                name = template_name.replace(" ", "_").lower()
SNGT_QHENOMENOLOGY_16: (4)                template_path = os.path.join(get_manim_dir(),
"manimlib", "tex_templates.yml")
SNGT_QHENOMENOLOGY_17: (4)                with open(template_path, encoding="utf-8") as
tex_templates_file:
SNGT_QHENOMENOLOGY_18: (8)                    templates_dict = yaml.safe_load(tex_templates_file)
SNGT_QHENOMENOLOGY_19: (4)                if name not in templates_dict:
SNGT_QHENOMENOLOGY_20: (8)                    log.warning(
SNGT_QHENOMENOLOGY_21: (12)                       "Cannot recognize template '%s', falling back
to 'default'.",
SNGT_QHENOMENOLOGY_22: (12)                       name
SNGT_QHENOMENOLOGY_23: (8)                    )
SNGT_QHENOMENOLOGY_24: (8)                    name = "default"
SNGT_QHENOMENOLOGY_25: (4)                return templates_dict[name]
SNGT_QHENOMENOLOGY_26: (0)            @lru_cache
SNGT_QHENOMENOLOGY_27: (0)            def get_tex_config(template: str = "") -> tuple[str, str]:
SNGT_QHENOMENOLOGY_28: (4)                """
```

```
SNGT_QHENOMENOLOGY_29: (4)                 Returns a compiler and preamble to use for rendering
LaTeX
SNGT_QHENOMENOLOGY_30: (4)                 """
SNGT_QHENOMENOLOGY_31: (4)                 template = template or manim_config.tex.template
SNGT_QHENOMENOLOGY_32: (4)                 config = get_tex_template_config(template)
SNGT_QHENOMENOLOGY_33: (4)                 return config["compiler"], config["preamble"]
SNGT_QHENOMENOLOGY_34: (0)             def get_full_tex(content: str, preamble: str = ""):
SNGT_QHENOMENOLOGY_35: (4)                 return "\n\n".join((
SNGT_QHENOMENOLOGY_36: (8)                     "\\documentclass[preview]{standalone}",
SNGT_QHENOMENOLOGY_37: (8)                     preamble,
SNGT_QHENOMENOLOGY_38: (8)                     "\\begin{document}",
SNGT_QHENOMENOLOGY_39: (8)                     content,
SNGT_QHENOMENOLOGY_40: (8)                     "\\end{document}"
SNGT_QHENOMENOLOGY_41: (4)                 )) + "\n"
SNGT_QHENOMENOLOGY_42: (0)             @lru_cache(maxsize=128)
SNGT_QHENOMENOLOGY_43: (0)             def latex_to_svg(
SNGT_QHENOMENOLOGY_44: (4)                 latex: str,
SNGT_QHENOMENOLOGY_45: (4)                 template: str = "",
SNGT_QHENOMENOLOGY_46: (4)                 additional_preamble: str = "",
SNGT_QHENOMENOLOGY_47: (4)                 short_tex: str = "",
SNGT_QHENOMENOLOGY_48: (4)                 show_message_during_execution: bool = True,
SNGT_QHENOMENOLOGY_49: (0)             ) -> str:
SNGT_QHENOMENOLOGY_50: (4)                 """Convert LaTeX string to SVG string.
SNGT_QHENOMENOLOGY_51: (4)                 Args:
SNGT_QHENOMENOLOGY_52: (8)                     latex: LaTeX source code
SNGT_QHENOMENOLOGY_53: (8)                     template: Path to a template LaTeX file
SNGT_QHENOMENOLOGY_54: (8)                     additional_preamble: String including any added
"\\usepackage{...}" style imports
SNGT_QHENOMENOLOGY_55: (4)                 Returns:
SNGT_QHENOMENOLOGY_56: (8)                     str: SVG source code
SNGT_QHENOMENOLOGY_57: (4)                 Raises:
SNGT_QHENOMENOLOGY_58: (8)                     LatexError: If LaTeX compilation fails
SNGT_QHENOMENOLOGY_59: (8)                     NotImplementedError: If compiler is not supported
SNGT_QHENOMENOLOGY_60: (4)                 """
SNGT_QHENOMENOLOGY_61: (4)                 if show_message_during_execution:
SNGT_QHENOMENOLOGY_62: (8)                     message = f"Writing {(short_tex or latex)[:70]}..."
SNGT_QHENOMENOLOGY_63: (4)                 else:
SNGT_QHENOMENOLOGY_64: (8)                     message = ""
SNGT_QHENOMENOLOGY_65: (4)                 compiler, preamble = get_tex_config(template)
SNGT_QHENOMENOLOGY_66: (4)                 preamble = "\n".join([preamble, additional_preamble])
SNGT_QHENOMENOLOGY_67: (4)                 full_tex = get_full_tex(latex, preamble)
SNGT_QHENOMENOLOGY_68: (4)                 return full_tex_to_svg(full_tex, compiler, message)
SNGT_QHENOMENOLOGY_69: (0)             @cache_on_disk
SNGT_QHENOMENOLOGY_70: (0)             def full_tex_to_svg(full_tex: str, compiler: str = "latex",
message: str = ""):
SNGT_QHENOMENOLOGY_71: (4)                 if message:
SNGT_QHENOMENOLOGY_72: (8)                     print(message, end="\r")
SNGT_QHENOMENOLOGY_73: (4)                 if compiler == "latex":
SNGT_QHENOMENOLOGY_74: (8)                     dvi_ext = ".dvi"
SNGT_QHENOMENOLOGY_75: (4)                 elif compiler == "xelatex":
SNGT_QHENOMENOLOGY_76: (8)                     dvi_ext = ".xdv"
SNGT_QHENOMENOLOGY_77: (4)                 else:
SNGT_QHENOMENOLOGY_78: (8)                     raise NotImplementedError(f"Compiler '{compiler}'
is not implemented")
SNGT_QHENOMENOLOGY_79: (4)                 with tempfile.TemporaryDirectory() as temp_dir:
SNGT_QHENOMENOLOGY_80: (8)                     tex_path = Path(temp_dir,
"working").with_suffix(".tex")
SNGT_QHENOMENOLOGY_81: (8)                     dvi_path = tex_path.with_suffix(dvi_ext)
SNGT_QHENOMENOLOGY_82: (8)                     tex_path.write_text(full_tex)
SNGT_QHENOMENOLOGY_83: (8)                     process = subprocess.run(
SNGT_QHENOMENOLOGY_84: (12)                        [
SNGT_QHENOMENOLOGY_85: (16)                            compiler,
SNGT_QHENOMENOLOGY_86: (16)                            "-no-pdf",
SNGT_QHENOMENOLOGY_87: (16)                            "-interaction=batchmode",
SNGT_QHENOMENOLOGY_88: (16)                            "-halt-on-error",
SNGT_QHENOMENOLOGY_89: (16)                            f"-output-directory={temp_dir}",
SNGT_QHENOMENOLOGY_90: (16)                            tex_path
SNGT_QHENOMENOLOGY_91: (12)                        ],
SNGT_QHENOMENOLOGY_92: (12)                        capture_output=True,
```

```
SNGT_QHENOMENOLOGY_93: (12)                         text=True
SNGT_QHENOMENOLOGY_94: (8)                     )
SNGT_QHENOMENOLOGY_95: (8)                     if process.returncode != 0:
SNGT_QHENOMENOLOGY_96: (12)                         error_str = ""
SNGT_QHENOMENOLOGY_97: (12)                         log_path = tex_path.with_suffix(".log")
SNGT_QHENOMENOLOGY_98: (12)                         if log_path.exists():
SNGT_QHENOMENOLOGY_99: (16)                             content = log_path.read_text()
SNGT_QHENOMENOLOGY_100: (16)                            error_match = re.search(r"(?<=\n!
).*\n.*\n", content)
SNGT_QHENOMENOLOGY_101: (16)                            if error_match:
SNGT_QHENOMENOLOGY_102: (20)                                error_str = error_match.group()
SNGT_QHENOMENOLOGY_103: (12)                        raise LatexError(error_str or "LaTeX
compilation failed")
SNGT_QHENOMENOLOGY_104: (8)                    process = subprocess.run(
SNGT_QHENOMENOLOGY_105: (12)                        [
SNGT_QHENOMENOLOGY_106: (16)                            "dvisvgm",
SNGT_QHENOMENOLOGY_107: (16)                            dvi_path,
SNGT_QHENOMENOLOGY_108: (16)                            "-n",  # no fonts
SNGT_QHENOMENOLOGY_109: (16)                            "-v", "0",  # quiet
SNGT_QHENOMENOLOGY_110: (16)                            "--stdout",  # output to stdout instead of
file
SNGT_QHENOMENOLOGY_111: (12)                        ],
SNGT_QHENOMENOLOGY_112: (12)                        capture_output=True
SNGT_QHENOMENOLOGY_113: (8)                    )
SNGT_QHENOMENOLOGY_114: (8)                    result = process.stdout.decode('utf-8')
SNGT_QHENOMENOLOGY_115: (4)                if message:
SNGT_QHENOMENOLOGY_116: (8)                    print(" " * len(message), end="\r")
SNGT_QHENOMENOLOGY_117: (4)                return result
SNGT_QHENOMENOLOGY_118: (0)            class LatexError(Exception):
SNGT_QHENOMENOLOGY_119: (4)                pass
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 95 - tex_to_symbol_count.py:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              TEX_TO_SYMBOL_COUNT = {
SNGT_QHENOMENOLOGY_2: (4)                  R"\!": 0,
SNGT_QHENOMENOLOGY_3: (4)                  R"\,": 0,
SNGT_QHENOMENOLOGY_4: (4)                  R"\-": 0,
SNGT_QHENOMENOLOGY_5: (4)                  R"\/": 0,
SNGT_QHENOMENOLOGY_6: (4)                  R"\:": 0,
SNGT_QHENOMENOLOGY_7: (4)                  R"\;": 0,
SNGT_QHENOMENOLOGY_8: (4)                  R"\>": 0,
SNGT_QHENOMENOLOGY_9: (4)                  R"\aa": 0,
SNGT_QHENOMENOLOGY_10: (4)                 R"\AA": 0,
SNGT_QHENOMENOLOGY_11: (4)                 R"\ae": 0,
SNGT_QHENOMENOLOGY_12: (4)                 R"\AE": 0,
SNGT_QHENOMENOLOGY_13: (4)                 R"\arccos": 6,
SNGT_QHENOMENOLOGY_14: (4)                 R"\arcsin": 6,
SNGT_QHENOMENOLOGY_15: (4)                 R"\arctan": 6,
SNGT_QHENOMENOLOGY_16: (4)                 R"\arg": 3,
SNGT_QHENOMENOLOGY_17: (4)                 R"\author": 0,
SNGT_QHENOMENOLOGY_18: (4)                 R"\bf": 0,
SNGT_QHENOMENOLOGY_19: (4)                 R"\bibliography": 0,
SNGT_QHENOMENOLOGY_20: (4)                 R"\bibliographystyle": 0,
SNGT_QHENOMENOLOGY_21: (4)                 R"\big": 0,
SNGT_QHENOMENOLOGY_22: (4)                 R"\Big": 0,
SNGT_QHENOMENOLOGY_23: (4)                 R"\bigodot": 4,
SNGT_QHENOMENOLOGY_24: (4)                 R"\bigoplus": 5,
SNGT_QHENOMENOLOGY_25: (4)                 R"\bigskip": 0,
SNGT_QHENOMENOLOGY_26: (4)                 R"\bmod": 3,
SNGT_QHENOMENOLOGY_27: (4)                 R"\boldmath": 0,
SNGT_QHENOMENOLOGY_28: (4)                 R"\bottomfraction": 2,
SNGT_QHENOMENOLOGY_29: (4)                 R"\bowtie": 2,
SNGT_QHENOMENOLOGY_30: (4)                 R"\cal": 0,
SNGT_QHENOMENOLOGY_31: (4)                 R"\cdots": 3,
SNGT_QHENOMENOLOGY_32: (4)                 R"\centering": 0,
SNGT_QHENOMENOLOGY_33: (4)                 R"\cite": 2,
SNGT_QHENOMENOLOGY_34: (4)                 R"\cong": 2,
```

```
SNGT_QHENOMENOLOGY_35: (4)                R"\contentsline": 0,
SNGT_QHENOMENOLOGY_36: (4)                R"\cos": 3,
SNGT_QHENOMENOLOGY_37: (4)                R"\cosh": 4,
SNGT_QHENOMENOLOGY_38: (4)                R"\cot": 3,
SNGT_QHENOMENOLOGY_39: (4)                R"\coth": 4,
SNGT_QHENOMENOLOGY_40: (4)                R"\csc": 3,
SNGT_QHENOMENOLOGY_41: (4)                R"\date": 0,
SNGT_QHENOMENOLOGY_42: (4)                R"\dblfloatpagefraction": 2,
SNGT_QHENOMENOLOGY_43: (4)                R"\dbltopfraction": 2,
SNGT_QHENOMENOLOGY_44: (4)                R"\ddots": 3,
SNGT_QHENOMENOLOGY_45: (4)                R"\deg": 3,
SNGT_QHENOMENOLOGY_46: (4)                R"\det": 3,
SNGT_QHENOMENOLOGY_47: (4)                R"\dim": 3,
SNGT_QHENOMENOLOGY_48: (4)                R"\displaystyle": 0,
SNGT_QHENOMENOLOGY_49: (4)                R"\div": 2,
SNGT_QHENOMENOLOGY_50: (4)                R"\doteq": 2,
SNGT_QHENOMENOLOGY_51: (4)                R"\dotfill": 0,
SNGT_QHENOMENOLOGY_52: (4)                R"\dots": 3,
SNGT_QHENOMENOLOGY_53: (4)                R"\emph": 0,
SNGT_QHENOMENOLOGY_54: (4)                R"\exp": 3,
SNGT_QHENOMENOLOGY_55: (4)                R"\fbox": 4,
SNGT_QHENOMENOLOGY_56: (4)                R"\floatpagefraction": 2,
SNGT_QHENOMENOLOGY_57: (4)                R"\flushbottom": 0,
SNGT_QHENOMENOLOGY_58: (4)                R"\footnotesize": 0,
SNGT_QHENOMENOLOGY_59: (4)                R"\footnotetext": 0,
SNGT_QHENOMENOLOGY_60: (4)                R"\frame": 2,
SNGT_QHENOMENOLOGY_61: (4)                R"\framebox": 4,
SNGT_QHENOMENOLOGY_62: (4)                R"\fussy": 0,
SNGT_QHENOMENOLOGY_63: (4)                R"\gcd": 3,
SNGT_QHENOMENOLOGY_64: (4)                R"\ghost": 0,
SNGT_QHENOMENOLOGY_65: (4)                R"\glossary": 0,
SNGT_QHENOMENOLOGY_66: (4)                R"\hfill": 0,
SNGT_QHENOMENOLOGY_67: (4)                R"\hom": 3,
SNGT_QHENOMENOLOGY_68: (4)                R"\hookleftarrow": 2,
SNGT_QHENOMENOLOGY_69: (4)                R"\hookrightarrow": 2,
SNGT_QHENOMENOLOGY_70: (4)                R"\hrulefill": 0,
SNGT_QHENOMENOLOGY_71: (4)                R"\huge": 0,
SNGT_QHENOMENOLOGY_72: (4)                R"\Huge": 0,
SNGT_QHENOMENOLOGY_73: (4)                R"\hyphenation": 0,
SNGT_QHENOMENOLOGY_74: (4)                R"\iff": 2,
SNGT_QHENOMENOLOGY_75: (4)                R"\Im": 2,
SNGT_QHENOMENOLOGY_76: (4)                R"\index": 0,
SNGT_QHENOMENOLOGY_77: (4)                R"\inf": 3,
SNGT_QHENOMENOLOGY_78: (4)                R"\it": 0,
SNGT_QHENOMENOLOGY_79: (4)                R"\ker": 3,
SNGT_QHENOMENOLOGY_80: (4)                R"\l": 0,
SNGT_QHENOMENOLOGY_81: (4)                R"\L": 0,
SNGT_QHENOMENOLOGY_82: (4)                R"\label": 0,
SNGT_QHENOMENOLOGY_83: (4)                R"\large": 0,
SNGT_QHENOMENOLOGY_84: (4)                R"\Large": 0,
SNGT_QHENOMENOLOGY_85: (4)                R"\LARGE": 0,
SNGT_QHENOMENOLOGY_86: (4)                R"\ldots": 3,
SNGT_QHENOMENOLOGY_87: (4)                R"\lefteqn": 0,
SNGT_QHENOMENOLOGY_88: (4)                R"\left": 0,
SNGT_QHENOMENOLOGY_89: (4)                R"\lg": 2,
SNGT_QHENOMENOLOGY_90: (4)                R"\lim": 3,
SNGT_QHENOMENOLOGY_91: (4)                R"\liminf": 6,
SNGT_QHENOMENOLOGY_92: (4)                R"\limsup": 6,
SNGT_QHENOMENOLOGY_93: (4)                R"\linebreak": 0,
SNGT_QHENOMENOLOGY_94: (4)                R"\ln": 2,
SNGT_QHENOMENOLOGY_95: (4)                R"\log": 3,
SNGT_QHENOMENOLOGY_96: (4)                R"\longleftarrow": 2,
SNGT_QHENOMENOLOGY_97: (4)                R"\Longleftarrow": 2,
SNGT_QHENOMENOLOGY_98: (4)                R"\longleftrightarrow": 2,
SNGT_QHENOMENOLOGY_99: (4)                R"\Longleftrightarrow": 2,
SNGT_QHENOMENOLOGY_100: (4)               R"\longmapsto": 3,
SNGT_QHENOMENOLOGY_101: (4)               R"\longrightarrow": 2,
SNGT_QHENOMENOLOGY_102: (4)               R"\Longrightarrow": 2,
SNGT_QHENOMENOLOGY_103: (4)               R"\makebox": 0,
```

```
SNGT_QHENOMENOLOGY_104: (4)                R"\mapsto": 2,
SNGT_QHENOMENOLOGY_105: (4)                R"\markright": 0,
SNGT_QHENOMENOLOGY_106: (4)                R"\mathds": 0,
SNGT_QHENOMENOLOGY_107: (4)                R"\max": 3,
SNGT_QHENOMENOLOGY_108: (4)                R"\mbox": 0,
SNGT_QHENOMENOLOGY_109: (4)                R"\medskip": 0,
SNGT_QHENOMENOLOGY_110: (4)                R"\min": 3,
SNGT_QHENOMENOLOGY_111: (4)                R"\mit": 0,
SNGT_QHENOMENOLOGY_112: (4)                R"\models": 2,
SNGT_QHENOMENOLOGY_113: (4)                R"\ne": 2,
SNGT_QHENOMENOLOGY_114: (4)                R"\neq": 2,
SNGT_QHENOMENOLOGY_115: (4)                R"\newline": 0,
SNGT_QHENOMENOLOGY_116: (4)                R"\noindent": 0,
SNGT_QHENOMENOLOGY_117: (4)                R"\nolinebreak": 0,
SNGT_QHENOMENOLOGY_118: (4)                R"\nonumber": 0,
SNGT_QHENOMENOLOGY_119: (4)                R"\nopagebreak": 0,
SNGT_QHENOMENOLOGY_120: (4)                R"\normalmarginpar": 0,
SNGT_QHENOMENOLOGY_121: (4)                R"\normalsize": 0,
SNGT_QHENOMENOLOGY_122: (4)                R"\notin": 2,
SNGT_QHENOMENOLOGY_123: (4)                R"\o": 0,
SNGT_QHENOMENOLOGY_124: (4)                R"\O": 0,
SNGT_QHENOMENOLOGY_125: (4)                R"\obeycr": 0,
SNGT_QHENOMENOLOGY_126: (4)                R"\oe": 0,
SNGT_QHENOMENOLOGY_127: (4)                R"\OE": 0,
SNGT_QHENOMENOLOGY_128: (4)                R"\overbrace": 4,
SNGT_QHENOMENOLOGY_129: (4)                R"\pagebreak": 0,
SNGT_QHENOMENOLOGY_130: (4)                R"\pagenumbering": 0,
SNGT_QHENOMENOLOGY_131: (4)                R"\pageref": 2,
SNGT_QHENOMENOLOGY_132: (4)                R"\pmod": 5,
SNGT_QHENOMENOLOGY_133: (4)                R"\Pr": 2,
SNGT_QHENOMENOLOGY_134: (4)                R"\protect": 0,
SNGT_QHENOMENOLOGY_135: (4)                R"\qquad": 0,
SNGT_QHENOMENOLOGY_136: (4)                R"\quad": 0,
SNGT_QHENOMENOLOGY_137: (4)                R"\raggedbottom": 0,
SNGT_QHENOMENOLOGY_138: (4)                R"\raggedleft": 0,
SNGT_QHENOMENOLOGY_139: (4)                R"\raggedright": 0,
SNGT_QHENOMENOLOGY_140: (4)                R"\Re": 2,
SNGT_QHENOMENOLOGY_141: (4)                R"\ref": 2,
SNGT_QHENOMENOLOGY_142: (4)                R"\restorecr": 0,
SNGT_QHENOMENOLOGY_143: (4)                R"\reversemarginpar": 0,
SNGT_QHENOMENOLOGY_144: (4)                R"\right": 0,
SNGT_QHENOMENOLOGY_145: (4)                R"\rm": 0,
SNGT_QHENOMENOLOGY_146: (4)                R"\sc": 0,
SNGT_QHENOMENOLOGY_147: (4)                R"\scriptscriptstyle": 0,
SNGT_QHENOMENOLOGY_148: (4)                R"\scriptsize": 0,
SNGT_QHENOMENOLOGY_149: (4)                R"\scriptstyle": 0,
SNGT_QHENOMENOLOGY_150: (4)                R"\sec": 3,
SNGT_QHENOMENOLOGY_151: (4)                R"\sf": 0,
SNGT_QHENOMENOLOGY_152: (4)                R"\shortstack": 0,
SNGT_QHENOMENOLOGY_153: (4)                R"\sin": 3,
SNGT_QHENOMENOLOGY_154: (4)                R"\sinh": 4,
SNGT_QHENOMENOLOGY_155: (4)                R"\sl": 0,
SNGT_QHENOMENOLOGY_156: (4)                R"\sloppy": 0,
SNGT_QHENOMENOLOGY_157: (4)                R"\small": 0,
SNGT_QHENOMENOLOGY_158: (4)                R"\Small": 0,
SNGT_QHENOMENOLOGY_159: (4)                R"\smallskip": 0,
SNGT_QHENOMENOLOGY_160: (4)                R"\sqrt": 2,
SNGT_QHENOMENOLOGY_161: (4)                R"\ss": 0,
SNGT_QHENOMENOLOGY_162: (4)                R"\sup": 3,
SNGT_QHENOMENOLOGY_163: (4)                R"\tan": 3,
SNGT_QHENOMENOLOGY_164: (4)                R"\tanh": 4,
SNGT_QHENOMENOLOGY_165: (4)                R"\text": 0,
SNGT_QHENOMENOLOGY_166: (4)                R"\textbf": 0,
SNGT_QHENOMENOLOGY_167: (4)                R"\textfraction": 2,
SNGT_QHENOMENOLOGY_168: (4)                R"\textstyle": 0,
SNGT_QHENOMENOLOGY_169: (4)                R"\thicklines": 0,
SNGT_QHENOMENOLOGY_170: (4)                R"\thinlines": 0,
SNGT_QHENOMENOLOGY_171: (4)                R"\thinspace": 0,
SNGT_QHENOMENOLOGY_172: (4)                R"\tiny": 0,
```

```
SNGT_QHENOMENOLOGY_173: (4)                R"\title": 0,
SNGT_QHENOMENOLOGY_174: (4)                R"\today": 15,
SNGT_QHENOMENOLOGY_175: (4)                R"\topfraction": 2,
SNGT_QHENOMENOLOGY_176: (4)                R"\tt": 0,
SNGT_QHENOMENOLOGY_177: (4)                R"\typeout": 0,
SNGT_QHENOMENOLOGY_178: (4)                R"\unboldmath": 0,
SNGT_QHENOMENOLOGY_179: (4)                R"\underbrace": 6,
SNGT_QHENOMENOLOGY_180: (4)                R"\underline": 0,
SNGT_QHENOMENOLOGY_181: (4)                R"\value": 0,
SNGT_QHENOMENOLOGY_182: (4)                R"\vdots": 3,
SNGT_QHENOMENOLOGY_183: (4)                R"\vline": 0
SNGT_QHENOMENOLOGY_184: (0)            }
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_File 96 -
SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRYCOMBINER_aligner_20_characters_for_pythons_codes.p
y:
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_1: (0)              import os
SNGT_QHENOMENOLOGY_2: (0)              from datetime import datetime
SNGT_QHENOMENOLOGY_3: (0)              def get_file_info(root_folder):
SNGT_QHENOMENOLOGY_4: (4)                  file_info_list = []
SNGT_QHENOMENOLOGY_5: (4)                  for root, dirs, files in os.walk(root_folder):
SNGT_QHENOMENOLOGY_6: (8)                      for file in files:
SNGT_QHENOMENOLOGY_7: (12)                         try:
SNGT_QHENOMENOLOGY_8: (16)                             if file.endswith('.py'):
SNGT_QHENOMENOLOGY_9: (20)                                 file_path = os.path.join(root, file)
SNGT_QHENOMENOLOGY_10: (20)                                creation_time =
datetime.fromtimestamp(os.path.getctime(file_path))
SNGT_QHENOMENOLOGY_11: (20)                                modified_time =
datetime.fromtimestamp(os.path.getmtime(file_path))
SNGT_QHENOMENOLOGY_12: (20)                                file_extension = os.path.splitext(file)
[1].lower()
SNGT_QHENOMENOLOGY_13: (20)                                file_info_list.append([file, file_path,
creation_time, modified_time, file_extension, root])
SNGT_QHENOMENOLOGY_14: (12)                        except Exception as e:
SNGT_QHENOMENOLOGY_15: (16)                            print(f"Error processing file {file}: {e}")
SNGT_QHENOMENOLOGY_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
SNGT_QHENOMENOLOGY_17: (4)                 return file_info_list
SNGT_QHENOMENOLOGY_18: (0)             def process_file(file_info_list):
SNGT_QHENOMENOLOGY_19: (4)                 combined_output = []
SNGT_QHENOMENOLOGY_20: (4)                 for idx, (file_name, file_path, creation_time,
modified_time, file_extension, root) in enumerate(file_info_list):
SNGT_QHENOMENOLOGY_21: (8)                     with open(file_path, 'r', encoding='utf-8',
errors='ignore') as f:
SNGT_QHENOMENOLOGY_22: (12)                        content = f.read()
SNGT_QHENOMENOLOGY_23: (12)                        content = "\n".join([line for line in
content.split('\n') if line.strip() and not line.strip().startswith("#")])
SNGT_QHENOMENOLOGY_24: (12)                        content = content.replace('\t', '    ')
SNGT_QHENOMENOLOGY_25: (12)                        processed_lines = []
SNGT_QHENOMENOLOGY_26: (12)                        for i, line in enumerate(content.split('\n')):
SNGT_QHENOMENOLOGY_27: (16)                            leading_spaces = len(line) -
len(line.lstrip(' '))
SNGT_QHENOMENOLOGY_28: (16)                            line_number_str = f"{i+1}:
({leading_spaces})"
SNGT_QHENOMENOLOGY_29: (16)                            padding = ' ' * (20 - len(line_number_str))
SNGT_QHENOMENOLOGY_30: (16)                            processed_line = f"{line_number_str}
{padding}{line}"
SNGT_QHENOMENOLOGY_31: (16)                            processed_lines.append(processed_line)
SNGT_QHENOMENOLOGY_32: (12)                        content_with_line_numbers =
"\n".join(processed_lines)
SNGT_QHENOMENOLOGY_33: (12)                        combined_output.append(f"File {idx + 1} -
{file_name}:\n")
SNGT_QHENOMENOLOGY_34: (12)
combined_output.append(content_with_line_numbers)
SNGT_QHENOMENOLOGY_35: (12)                        combined_output.append("\n" + "-"*40 + "\n")
SNGT_QHENOMENOLOGY_36: (4)                 return combined_output
```

```
SNGT_QHENOMENOLOGY_37: (0)             root_folder_path = '.'  # Set this to the desired folder
SNGT_QHENOMENOLOGY_38: (0)             file_info_list = get_file_info(root_folder_path)
SNGT_QHENOMENOLOGY_39: (0)             combined_output = process_file(file_info_list)
SNGT_QHENOMENOLOGY_40: (0)             output_file =
'SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRY_combined_python_files_20_chars.txt'
SNGT_QHENOMENOLOGY_41: (0)             with open(output_file, 'w', encoding='utf-8') as logfile:
SNGT_QHENOMENOLOGY_42: (4)                 logfile.write("\n".join(combined_output))
SNGT_QHENOMENOLOGY_43: (0)             print(f"Processed file info logged to {output_file}")
SNGT_QHENOMENOLOGY_
SNGT_QHENOMENOLOGY_----------------------------------------
SNGT_QHENOMENOLOGY_
```