Deliverable 1

Team 28 - Big Software Energy 2.0
Muhammad Farooq, Jadin Luong, Vincent Teng, Aster Wu
February 26th 2020

Table of Contents

What is Matplotlib?

Matplotlib is a Python library that assists users with creating plots/graphs and outputting it graphically. Matplotlib is commonly used for scientific applications because of its ability to produce publication worthy graphs in many formats and interactive environments. Matplotlib can be used in Python, Jupyter, web apps, and GUI toolkits. Matplotlib is currently built using Python 3.

High Level System Design

Backend Layer

The Backend layer consists of three key components:

1. FigureCanvas – represents the surface/area in which figures will be drawn on
2. Renderer – responsible for drawing and rendering the figure onto the canvas
3. Event – handles user input/events

This is the bottom layer among the three layers in the stack of matplotlib's architecture and is essentially responsible for formulating/drawing figures onto the canvas and allowing users to interact with the figures. Since this layer is at the bottom of the architecture, it is not aware of the upper layers within the architecture, thus, it does not directly interact with those layers.

Artist Layer

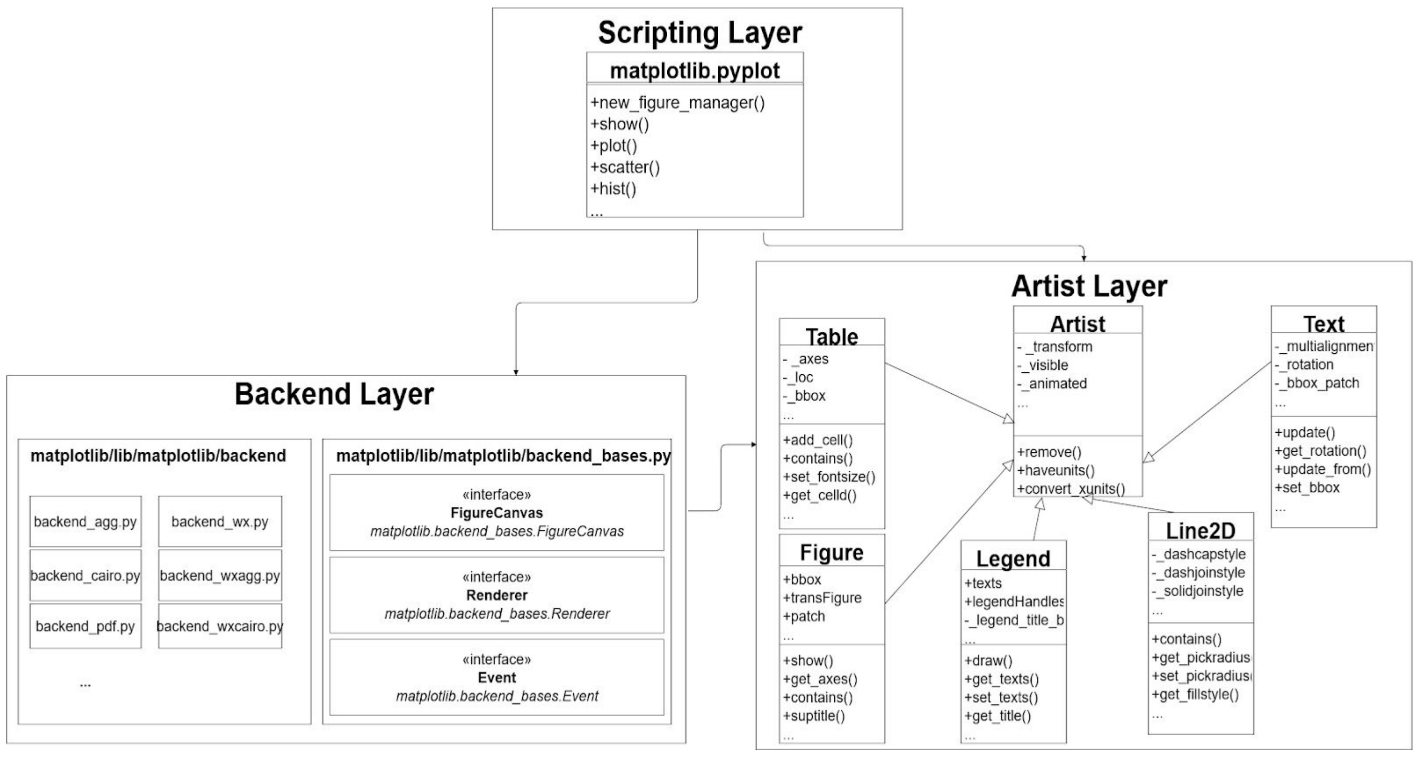
The Artist layer represents the middle layer of matplotlib's architecture. This layer consists of one main component which is the Artist class/object. Being the middle layer of matplotlib's architecture means that it is aware of bottom layers, in this case, it is aware and able to interact with the Backend Layer which is exactly what it does. The Artist layer tells the Backend layer how to draw/render on the canvas.

There are two main types of Artists:

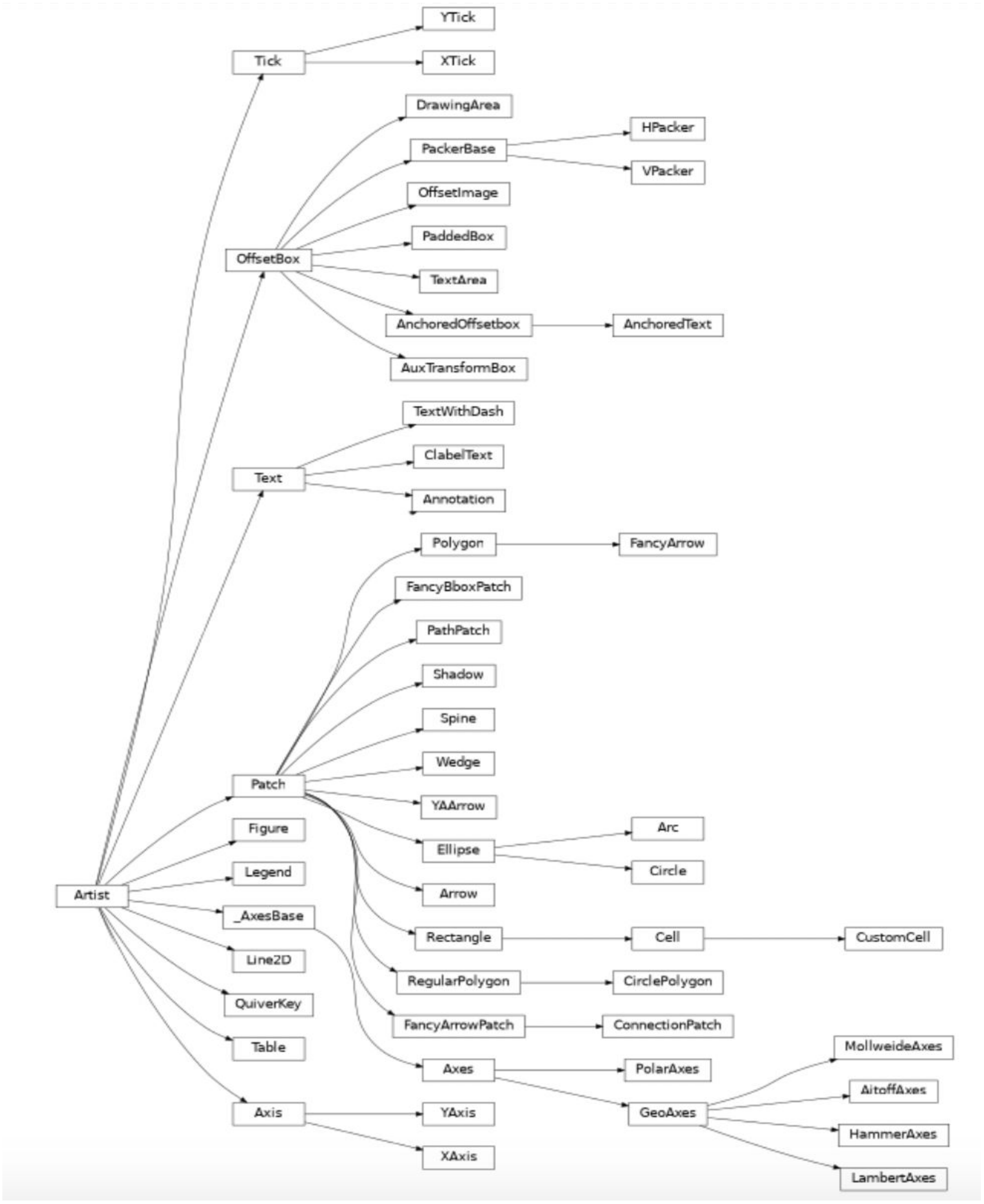
1. Primitive Artists
2. Composite Artists

Scripting Layer

The scripting layer is simply the Matplotlib.pyplot interface and it allows people who are not necessarily professional programmers to analyze and interact with the data portrayed within each constructed figure. The scripting layer interacts with both, the Backend Layer and Artist Layer as it acts like a controller that connects both layers with each other giving the layers the ability to communicate with one another. Overall, this layer allows people to easily visualize their data, making data analysis processes easier.



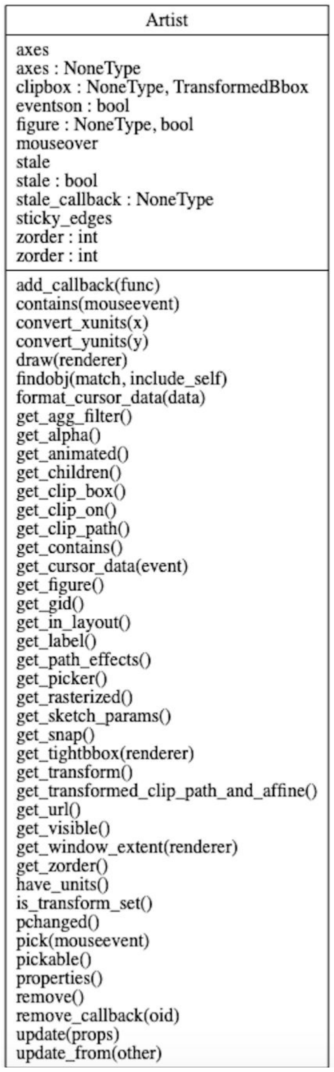
The Artist Layer



Overall Artist Layer Design

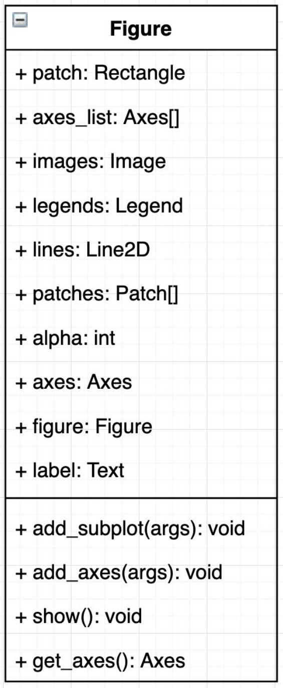
The Artist Layer is responsible for producing the visuals of matplotlib. The Artist object works with the Renderer object in the back end to produce the figures, lines, texts, and shapes that are displayed to the user. As stated in the high-level system design explanation, the 2 types of Artist objects are primitives and containers. Primitives are the individual graphical objects that we want to put on the canvas like Text, Line2D (Lines), Rectangle, Image. Containers are objects where we want to place our primitives on or inside. The Axis, Axes, and Figure objects are examples of containers.

Artist Instance



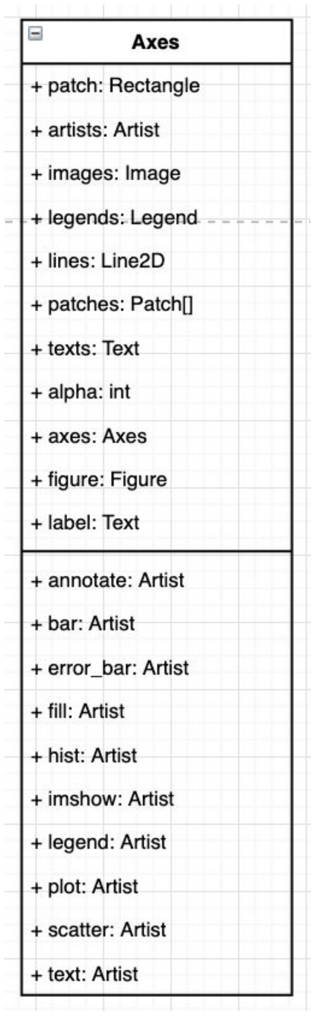
In most cases, the Artist object will create a Figure instance to draw an Axes container. The Axes container instance will then use its methods to draw either an Axis instance or more Figure instances. All containers can contain either more containers or more primitives. Containers are also instantiated with primitives inside. For example, the Axes container uses a Rectangle instance as the bounding box and a Figure instance will have a Rectangle instance that is the same size. The instantiated rectangle is called Patch (Figure.patch, Axes.patch) inside the instance that uses it.

Figure Instance



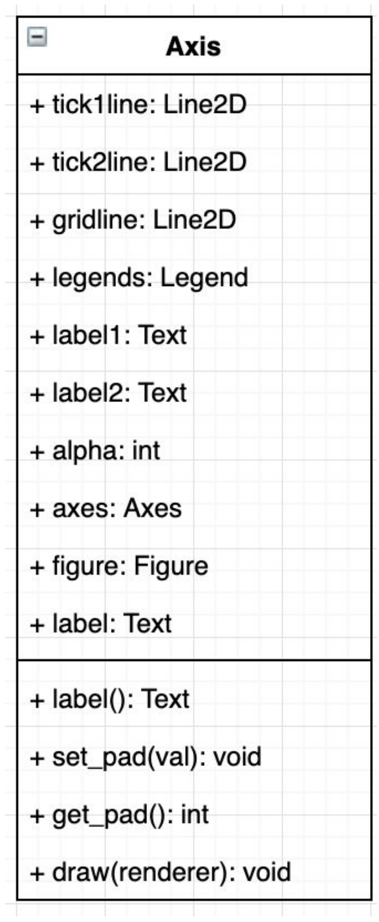
The Figure instance is the top level container. You can add Axes or primitives or even other figures to the instance. The attributes hold what primitives have been created inside the figure.

Axes Instance



The Axes instance contains most of the elements that Figure has and is also responsible for the coordinate system. It can also be considered the area where the user will plot the graph. The Axes instance allows the user to draw histograms, scatter plots, bar charts and more.

Axis Instance



The Axis instance is made up of primitive Line2D instances for the tick markers, and the grid marker associated with the label. The ticks themselves have their own classes so the artists can draw each tick (including the label text and the gridline). The Axis instance is also made up of its own primitive Text instances for its labels.

The Backend Layer

Since different users have different requirements when it comes to what kind of renderer or GUI library they want to use, matplotlib does not have one single backend but rather several different backend implementations to satisfy different user requirements. All the backend implementations inherit abstract base class from the base backend classes found in: https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/backend_bases.py:

There are five main components in the base backend file that act as the building blocks to create Figures in matplotlib. Each backend implementation inherits from these classes. We will go into the details of the different backend implementations after but first we will focus on each of the five main base components.

Event

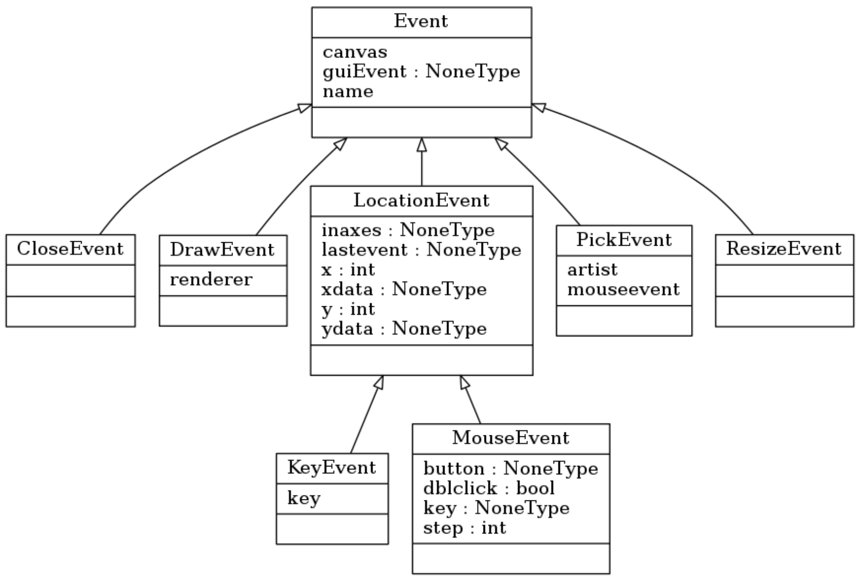
This is the base abstract class responsible for all event handling in matplotlib. Each inherited class is responsible for a different type of event.

CloseEvent -> Event's triggered by a Figure (see Artist section for Figures) being closed

DrawEvent -> Event triggered by something being drawn on the canvas (see Figure Canvas section below)

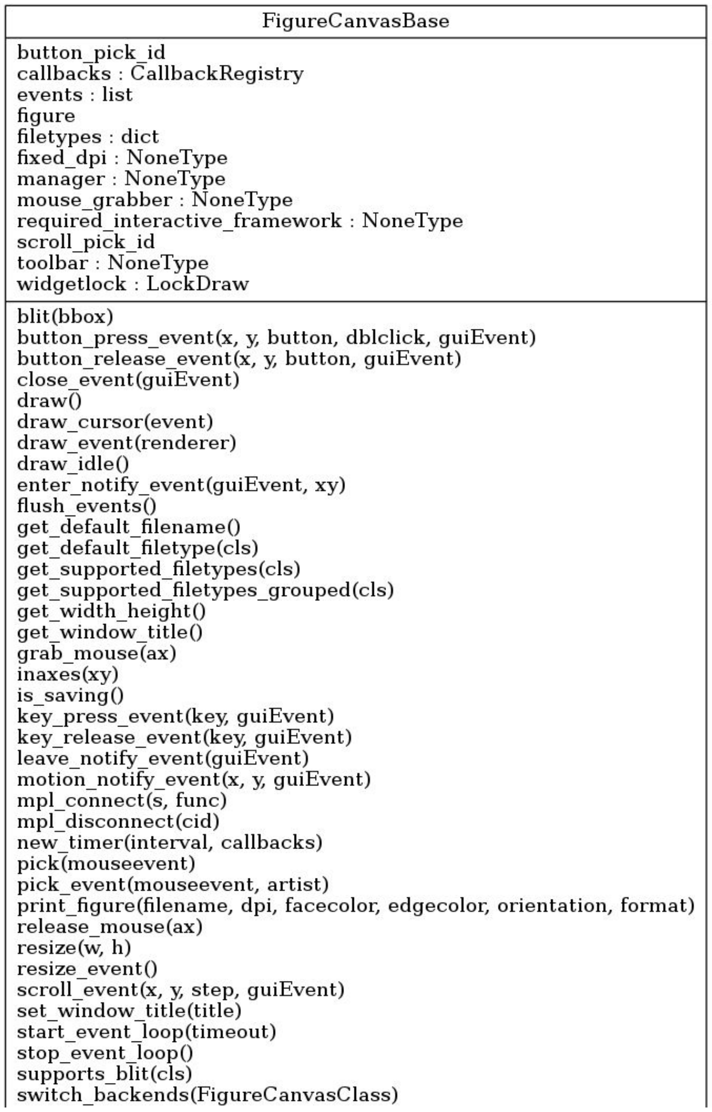
Etc.

Other Events are similarly described in their name.



FigureCanvasBase

This is the base abstract class to define a drawing “canvas”. It represents a surface to draw on. This is separated from the actual drawing itself (see RendererBase).



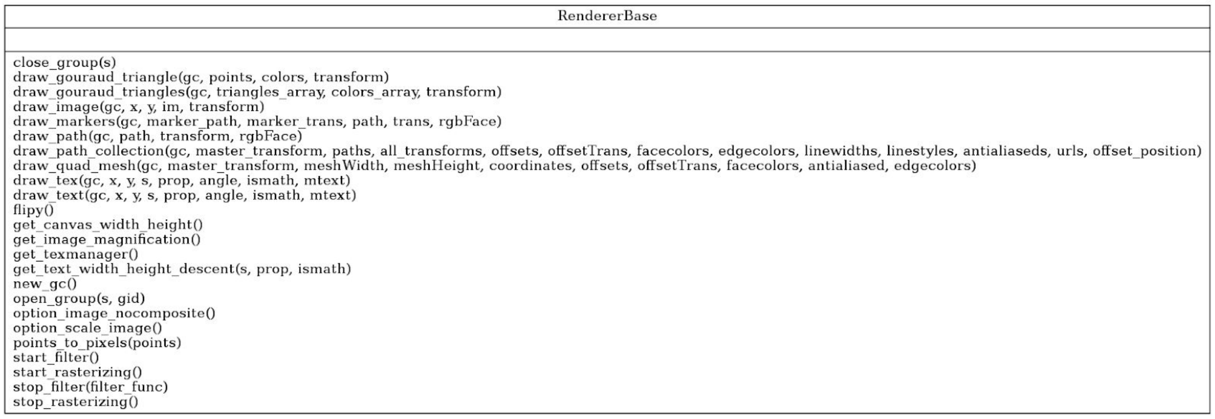
```
FigureCanvasBase

button_pick_id
callbacks : CallbackRegistry
events : list
figure
filetypes : dict
fixed_dpi : NoneType
manager : NoneType
mouse_grabber : NoneType
required_interactive_framework : NoneType
scroll_pick_id
toolbar : NoneType
widgetlock : LockDraw

blit(bbox)
button_press_event(x, y, button, dblclick, guiEvent)
button_release_event(x, y, button, guiEvent)
close_event(guiEvent)
draw()
draw_cursor(event)
draw_event(renderer)
draw_idle()
enter_notify_event(guiEvent, xy)
flush_events()
get_default_filename()
get_default_filetype(cls)
get_supported_filetypes(cls)
get_supported_filetypes_grouped(cls)
get_width_height()
get_window_title()
grab_mouse(ax)
inaxes(xy)
is_saving()
key_press_event(key, guiEvent)
key_release_event(key, guiEvent)
leave_notify_event(guiEvent)
motion_notify_event(x, y, guiEvent)
mpl_connect(s, func)
mpl_disconnect(cid)
new_timer(interval, callbacks)
pick(mouseevent)
pick_event(mouseevent, artist)
print_figure(filename, dpi, facecolor, edgecolor, orientation, format)
release_mouse(ax)
resize(w, h)
resize_event()
scroll_event(x, y, step, guiEvent)
set_window_title(title)
start_event_loop(timeout)
stop_event_loop()
supports_blit(cls)
switch_backends(FigureCanvasClass)
```

RendererBase

This is the base abstract class responsible for drawing and rendering operations.

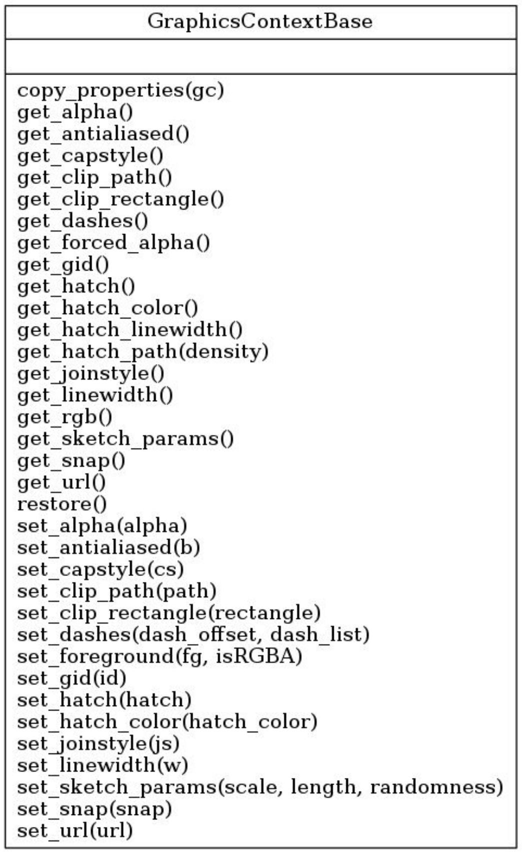


```
RendererBase

close_group(s)
draw_gouraud_triangle(gc, points, colors, transform)
draw_gouraud_triangles(gc, triangles_array, colors_array, transform)
draw_image(gc, x, y, im, transform)
draw_markers(gc, marker_path, marker_trans, path, trans, rgbFace)
draw_path(gc, path, transform, rgbFace)
draw_path_collection(gc, master_transform, paths, all_transforms, offsets, offsetTrans, facecolors, edgecolors, linewidths, linestyles, antialiaseds, urls, offset_position)
draw_quad_mesh(gc, master_transform, meshWidth, meshHeight, coordinates, offsets, offsetTrans, facecolors, antialiased, edgecolors)
draw_tex(gc, x, y, s, prop, angle, ismath, mtext)
draw_text(gc, x, y, s, prop, angle, ismath, mtext)
flipy()
get_canvas_width_height()
get_image_magnification()
get_texmanager()
get_text_width_height_descent(s, prop, ismath)
new_gc()
open_group(s, gid)
option_image_nocomposite()
option_scale_image()
points_to_pixels(points)
start_filter()
start_rasterizing()
stop_filter(filter_func)
stop_rasterizing()
```

GraphicsContextBase

The base abstract class responsible for providing color and styling of Figures.

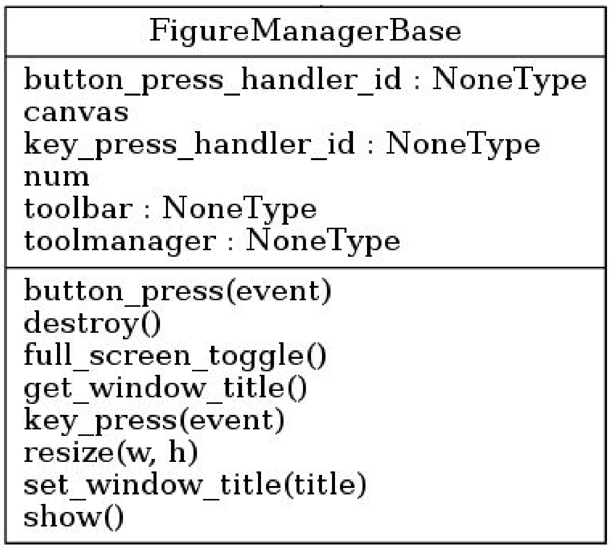


```
GraphicsContextBase

copy_properties(gc)
get_alpha()
get_antialiased()
get_capstyle()
get_clip_path()
get_clip_rectangle()
get_dashes()
get_forced_alpha()
get_gid()
get_hatch()
get_hatch_color()
get_hatch_linewidth()
get_hatch_path(density)
get_joinstyle()
get_linewidth()
get_rgb()
get_sketch_params()
get_snap()
get_url()
restore()
set_alpha(alpha)
set_antialiased(b)
set_capstyle(cs)
set_clip_path(path)
set_clip_rectangle(rectangle)
set_dashes(dash_offset, dash_list)
set_foreground(fg, isRGBA)
set_gid(id)
set_hatch(hatch)
set_hatch_color(hatch_color)
set_joinstyle(js)
set_linewidth(w)
set_sketch_params(scale, length, randomness)
set_snap(snap)
set_url(url)
```

FigureManagerBase

This abstract class is used by Pyplot (see Scripting Layer) as an adapter for the real backend GUI framework. It allows users to interact with the Figure canvas (i.e. the window of the figure) without interacting with the actual GUI framework that is building the figure (see below for differences between non-interactive and interactive backends).



Since different users use matplotlib in different ways, matplotlib implements a different backend to handle each use case and each output type. Some users want to use a specific graphics rendering library combined with a specific GUI library. Users can use a different backend implementation depending on their use case.

All backend implementation can be found in:
https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/backends/

There are two types of backend implementation: non-interactive and interactive.

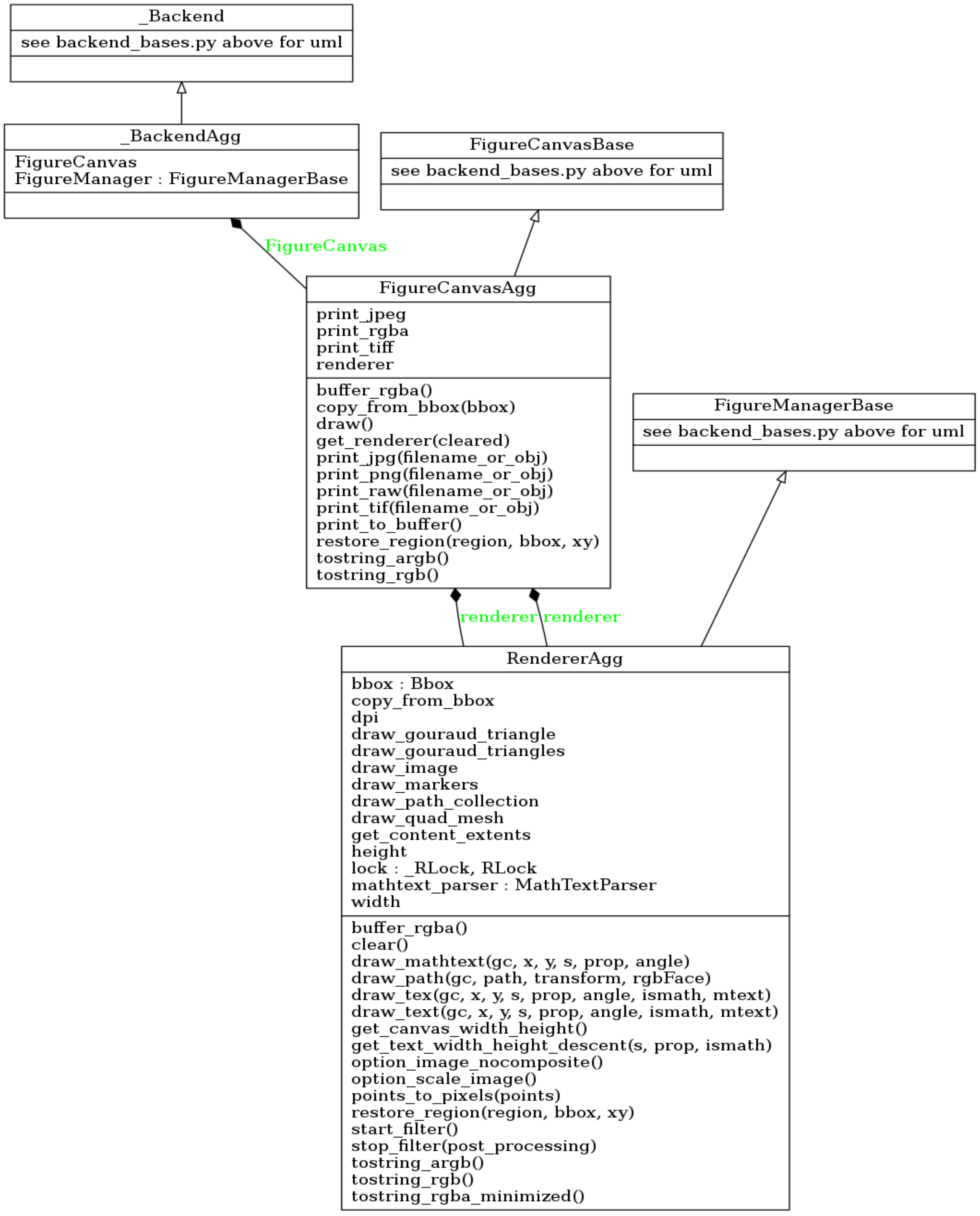
Non-interactive

These backends use graphic renderers like agg (anti grain geometry) and Cairo (PyCairo) or simply PDF, PNG, SVG, etc. to render non-interactive graphics of data.

Example: A matplotlib backend implementation for anti grain geometry rendering library.

https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/backends/backend_agg.py

(Imports agg library from /lib/matplotlib/backends/_backend_agg.cp38-win_amd64)

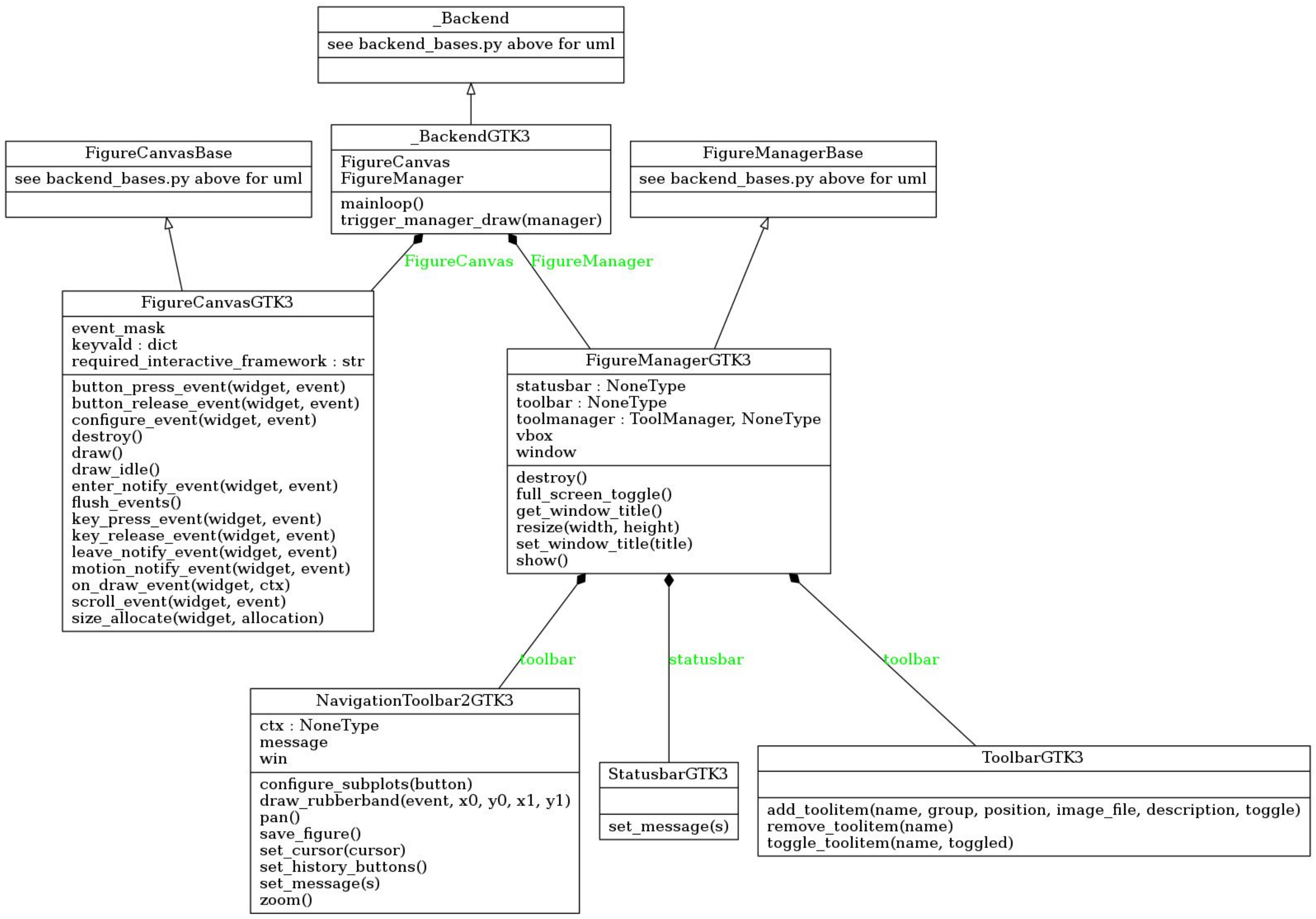

_Backend
see backend_bases.py above for uml
_BackendAgg
FigureCanvas
FigureManager : FigureManagerBase
FigureCanvasBase
see backend_bases.py above for uml
FigureCanvas
FigureCanvasAgg
print_jpeg
print_rgba
print_tiff
renderer
buffer_rgba()
copy_from_bbox(bbox)
draw()
get_renderer(cleared)
print_jpg(filename_or_obj)
print_png(filename_or_obj)
print_raw(filename_or_obj)
print_tif(filename_or_obj)
print_to_buffer()
restore_region(region, bbox, xy)
tostring_argb()
tostring_rgb()
FigureManagerBase
see backend_bases.py above for uml
renderer
renderer
RendererAgg
bbox : Bbox
copy_from_bbox
dpi
draw_gouraud_triangle
draw_gouraud_triangles
draw_image
draw_markers
draw_path_collection
draw_quad_mesh
get_content_extents
height
lock : _RLock, RLock
mathtext_parser : MathTextParser
width
buffer_rgba()
clear()
draw_mathtext(gc, x, y, s, prop, angle)
draw_path(gc, path, transform, rgbFace)
draw_tex(gc, x, y, s, prop, angle, ismath, mtext)
draw_text(gc, x, y, s, prop, angle, ismath, mtext)
get_canvas_width_height()
get_text_width_height_descent(s, prop, ismath)
option_image_nocomposite()
option_scale_image()
points_to_pixels(points)
restore_region(region, bbox, xy)
start_filter()
stop_filter(post_processing)
tostring_argb()
tostring_rgb()
tostring_rgba_minimized()

Interactive (GUI)

These backends use a combination of one the matplotlib rendering backends (backend_agg.py or backend_cairo.py) and a GUI library like GTK, TKinter, etc. to create an interactive graphic.

Example GUI backend: (some details removed)

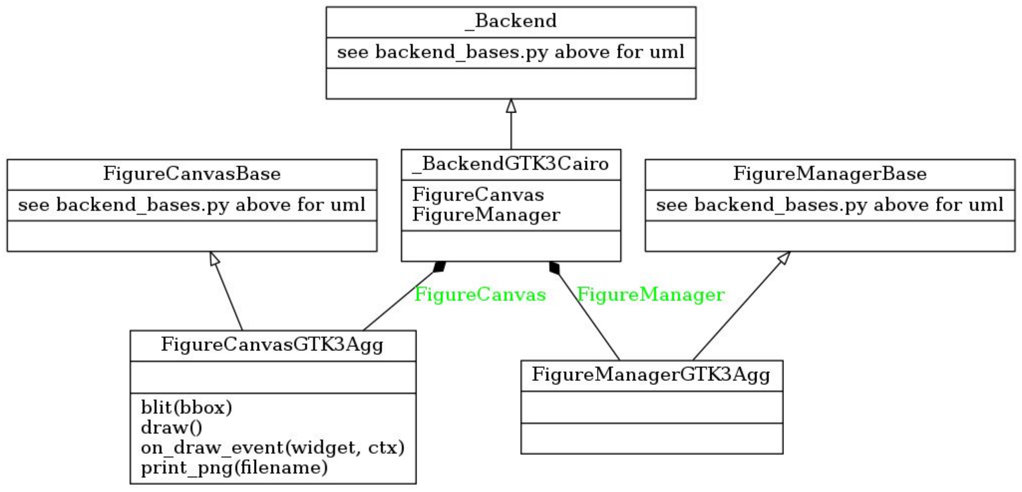
https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/backends/backend_gtk3.py



_Backend
see backend_bases.py above for uml
_BackendGTK3
FigureCanvas
FigureManager
mainloop()
trigger_manager_draw(manager)
FigureCanvasBase
see backend_bases.py above for uml
FigureManagerBase
see backend_bases.py above for uml
FigureCanvas
FigureManager
FigureCanvasGTK3
event_mask
keyvald : dict
required_interactive_framework : str
button_press_event(widget, event)
button_release_event(widget, event)
configure_event(widget, event)
destroy()
draw()
draw_idle()
enter_notify_event(widget, event)
flush_events()
key_press_event(widget, event)
key_release_event(widget, event)
leave_notify_event(widget, event)
motion_notify_event(widget, event)
on_draw_event(widget, ctx)
scroll_event(widget, event)
size_allocate(widget, allocation)
FigureManagerGTK3
statusbar : NoneType
toolbar : NoneType
toolmanager : ToolManager, NoneType
vbox
window
destroy()
full_screen_toggle()
get_window_title()
resize(width, height)
set_window_title(title)
show()
toolbar
statusbar
toolbar
NavigationToolbar2GTK3
ctx : NoneType
message
win
configure_subplots(button)
draw_rubberband(event, x0, y0, x1, y1)
pan()
save_figure()
set_cursor(cursor)
set_history_buttons()
set_message(s)
zoom()
StatusbarGTK3
set_message(s)
ToolbarGTK3
add_toolitem(name, group, position, image_file, description, toggle)
remove_toolitem(name)
toggle_toolitem(name, toggled)

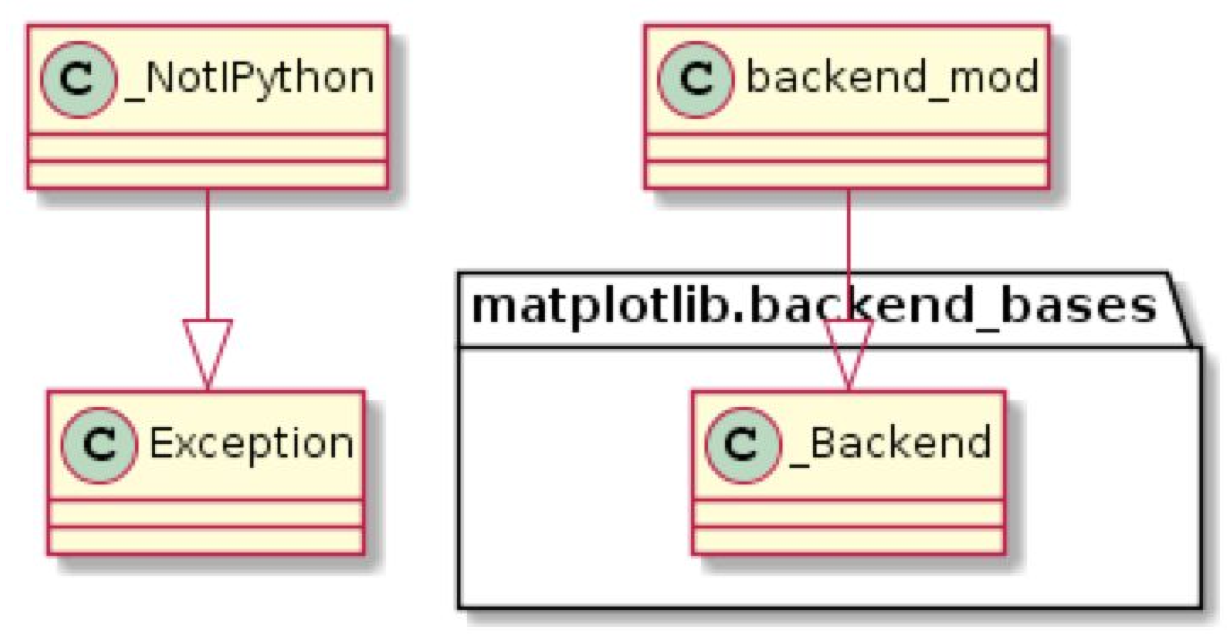
A matplotlib GUI backend and renderer backend are combined (i.e. inherited) to make a backend for interactive graphics:

Example: (agg as renderer and gtk3 as a GUI) https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/backends/backend_gtk3agg.py



The Scripting Layer

pyplot.py UML



The top level `state-machine` provided. A stateful interface (to the underlying object-oriented plotting library) which handles much of the boilerplate for creating figures and axes and connecting them to a desired backend. The interface also maintains module level data structure which represents the current figure and axes.

With respect to the backend and artist layer:
The scripting layer has a set of getter/setter functions to extract info from the artist layer in getp, get, setp.
The scripting layer always utilizes one backend module thus some of the scripting layer's global functions depend on the backend and are unified by the backend end modules inheriting default methods implemented in backend_bases._Backend

The file provides a number of simple functions used to add plot elements (lines, images, text, etc.) to the current axes in the current figure. See below

Highlighted Global Functions

switch_backend(newbackend):
- Close all open figures and set the Matplotlib backend
- The argument is case-insensitive. Switching to an interactive backend is possible only if no event loop for another interactive backend has started. Switching to and from non-interactive backends is always possible.

show(*args, **kwargs):
- Display all figures

xkcd(scale=1, length=100, randomness=2):
- Turns on xkcd, sketch style drawing mode

figure(num=None, # autoincrement if None, else integer from 1-N figsize=None, # defaults to rc figure.figsize dpi=None, # defaults to rc figure.dpi facecolor=None, # defaults to rc figure.facecolor edgecolor=None, # defaults to rc figure.edgecolor frameon=True, FigureClass=Figure, clear=False, **kwargs):
- Create a new figure, or activate an existing figure.

Software Development Process

Pros/Cons of each development process:

Waterfall

Pros	Cons
Each phase is clear/concise making the model very easy to understand and utilize	Product/software is not produced until later in the life cycle
Very useful for smaller projects where the requirements are well defined	Not useful for projects that may be subject to a lot of changes and modifications in the future
Each stage and goals are well defined and documented	Requirements can not be changed throughout the life cycle
Planning is done carefully, which may reduce number of issues that may arise in the future	Difficult to measure progress within each phase
Tasks are stable through development process, makes it easier for team members to understand goals of each task	May be very strict since requirements, goals and results must be known before development

Kanban

Pros	Cons
Team will always have a bird's eye view of the progress of the project due to the nature of the kanban board.	Requires more discipline on developers part as there are less strict deadlines on when tasks need to be completed.
Allows each team member to work at their own pace according to their own schedule in contrast to Scrum where team members maybe be burdened by the end of sprints.	Requires constant monitoring of the Kanban board. Outdated kanban boards can lead to miscommunication and bottlenecks in development.
No stops in the development process. Work is always continuously being done without stoppages and waiting time between tasks.	Difficult to build a timeline on when tasks should be completed due to lack of strict rules and deadlines.

Kanban is very responsive to change. We can easily add new tasks to our kanban board without worrying about sprint burndown, project replanning, and other time consuming activities.	
Daily kanban meetings will ensure we address blockers as they arise.	
By limiting the “Work in Progress” of the team, we reduce the impact of scope change, allowing the team to only focus on one set of tasks at a time.	
Kanban is easy to understand and adopt.	
Learning effective use of Kanban will be valuable for our future careers as industry further adopts Kanban principles.	

Scrum

Pros	Cons
Relatively short sprints for frequent feedback	Daily meetings required by SCRUM may not be achievable with time differences
Is agile, thus able to take feedback from customers and stake-holders	Difficult to execute in practice
Product + sprint backlog helps track issues for the entire team. Providing visibility/transparency to the project	Heavily hindered by team member(s) exiting in middle of project
Daily meetings help identify problems quickly and resolve road-blocks early	With no deadline to deliver, could lead to scope creep/allowing project managers to demand new functionalities
Iterative nature constantly improves product	Higher pressure on team members and more time is required to be spend on project development
By nature ensures mistakes are quickly identified and ratified	Only practical with smaller teams

XP

Pros	Cons
Focused on coding rather than meetings	Very little to no documentation of code
Pair programming can improve code by sharing ideas	Pair Programming can hinder developers who work well independently
Constant feedback from others (every week rather than 2 weeks like SCRUM)	Need to constantly test code to be able to push out small releases
Small packages of well-tested code	Relies on the developers following programming best practices consistently
Laidback with respect to how things get done as long as the code works	Code can get messy and confusing if the above is not true
Can respond well to fast-changing requirements because features can be changed as long as they have not been worked on	

We will be using Kanban as our software development process. Kanban is a recently created software development process that focuses on the visualization of tasks.

Why Kanban?

We chose Kanban as our development process primarily due to the flexibility it provides. Unlike Waterfall which restricts to rigid development structure with a lot of planning overhead and Scrum which is difficult to execute given our varying schedules and workloads, Kanban allows us to work at our own pace and adapt to changing requirements easily and quickly.
The cons of using Kanban were minor for our team while the pros were more significant (and in greater number) than other forms of software development. For each of the major disadvantages for using Kanban, we had a simple solution in place. The first con was Kanban requires a greater degree of discipline in each of our team members. As we already have experience working together in a group, we already have agreed upon expectations on the required effort that needs to be put into this process. The second con of using Kanban was constant monitoring and updating of the Kanban board. As we already have experience with using a project management tool (Jira), we all have experience with ensuring the board stays up-tp-date and we have previously discussed the importance of staying on top of the project board. The last con of using Kanban we identified was dealing with ambiguous timelines of

tasks. Since we already have fixed deadlines in place for deliverables, this problem will already be addressed. Overall, we agreed Kanban would work best with our team dynamic.

Our Development Process

We'll start each deliverable with an initial planning phase where we break down the larger goals into tasks which we will use as our tickets in our project planning software.

Tasks will be divided into five categories: To-Do, In Progress, Pending Review, In Review, Done.

After a tasks has been completed (i.e. implemented to satisfy all requirements, task-specific unit tests passed, and all matplotlib unit tests passed), the task should be moved by the developer to "Pending Review" until a team member with no task currently "In Review" has agreed to review the task. The task should then be moved to the "In Review" column by the developer who has worked on the task and the task should then be assigned to the reviewer. At this stage, code review takes place and if any issues are found, a comment should be added to the task stating the issue and potential solutions if possible, and the task should be moved back into the :In Progress" column where the developer will fix the issue. Otherwise, the issue can be moved into the "Done" column by the reviewer. At this point the code should be fully implemented, tested, reviewed, merged into our teams matplotlib master branch.

We will define "Done" as the feature has been tested manually by (at least) two members, they passed all of our own unit tests, and all the matplotlib unit tests pass as well. The code should not cause any regression related issues and should be merged into our teams repo of matplotlib.

Tasks will be divided into priorities based on what needs to be done first ranging form 1 (Top Priority) to 4 (if time permits).

For the "In Progress" and "In Review" columns, we will have a maximum column "Work in Progress" limit of 4 tasks.

Each team member can only have a maximum of one task in progress and one in review at any time. No member should have 0 tasks in progress unless the To Do column is empty i.e. all work is completed.

Daily meetings will take place everyday via Zoom/In person/Slack (if others not possible) at 10:30am for 15 minutes. During the meeting, we will give a short description of any progress done and we address any blockers to ensure work continues to flow.