How to draw Rectangles in Matplotlib (With Examples)

Authored by

stats writer

December 18, 2025

The Python ecosystem relies heavily on powerful visualization tools, and among the most prominent is Matplotlib. This robust library is celebrated for its capability to generate intricate charts, graphs, and complex data visualizations. Beyond standard plotting, Matplotlib also offers precise tools for drawing geometric shapes, such as rectangles, which are essential for highlighting regions of interest (ROIs) on plots or images, or for constructing custom graphical elements.

Drawing a rectangle in Matplotlib involves more than just defining its corner points; it requires specifying detailed properties like the starting coordinates, height, width, rotation angle, and extensive styling options including line color, line width, and fill color. This guide serves as an authoritative resource, demonstrating the necessary functions and providing practical, detailed examples to illustrate how you can seamlessly integrate custom rectangles into your visualizations using the Matplotlib framework.

To instantiate a rectangle within a Matplotlib figure, developers utilize a specific class from the patches module: `matplotlib.patches.Rectangle`. This function is fundamental to adding geometric primitives to an existing set of axes (or an image) and is highly configurable. Understanding its core syntax is the first step toward effective implementation.

The standard signature for generating a rectangle object is as follows, requiring the definition of its anchoring position and dimensions:

matplotlib.patches.Rectangle(xy, width, height, angle=0.0, **kwargs)

The essential arguments that define the rectangle's placement and size are:

xy: This tuple specifies the coordinate system for the rectangle's anchor point, typically the bottom-left corner of the shape.
width: A numerical value defining the horizontal extent of the rectangle.
height: A numerical value defining the vertical extent of the rectangle.
angle: An optional float value specifying the rotation of the rectangle in degrees, measured counter-clockwise from the horizontal axis. The default value is 0.

Once the rectangle object is created using `Rectangle()`, it must be explicitly added to a set of axes using the `ax.add_patch()` method. The examples below comprehensively detail this process, demonstrating implementation on a standard plot and on imported image data.

Example 1: Implementing a Basic Rectangle on a Plot

The most common application is adding a rectangle to an existing Matplotlib plot, often used to demarcate a significant area on a scatter plot or line graph. This initial example focuses solely on the necessary geometry: defining the anchor point, width, and height. We begin by setting up a basic figure and axis environment using `matplotlib.pyplot`.

In this demonstration, we create a rectangle anchored at the coordinates (1, 1), with a width of 2 units and a height of 6 units. Note that we must import both `pyplot` (for setting up the figure) and `Rectangle` (the class used to define the shape) from their respective modules.

The following code snippet executes these steps, resulting in a standard, unstyled rectangle overlaid on a simple diagonal line plot, illustrating how `ax.add_patch()` incorporates the shape into the visualization context.

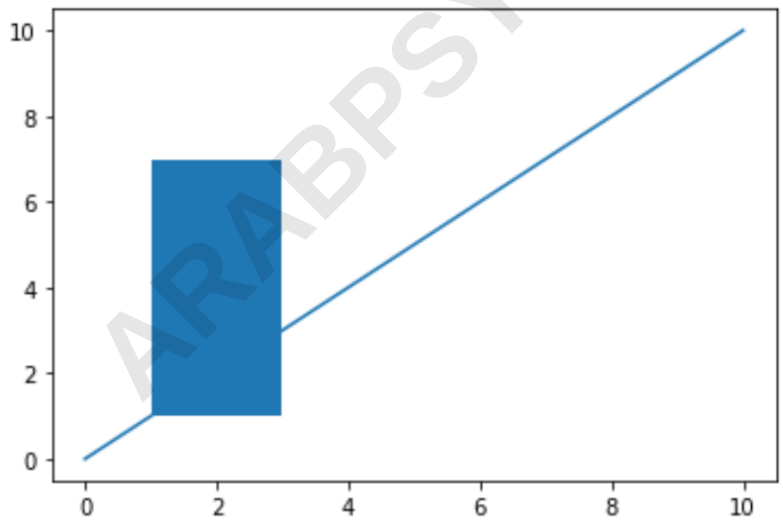
```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

#define Matplotlib figure and axis
fig, ax = plt.subplots()

#create simple line plot
ax.plot(,)

#add rectangle to plot
ax.add_patch(Rectangle((1, 1), 2, 6))

#display plot
plt.show()
```



Example 2: Customizing Rectangle Appearance with Styling

While the default rectangle provides basic functionality, Matplotlib excels at customization. The `Rectangle` class accepts numerous keyword arguments (`**kwargs`) that control visual attributes

such as color, transparency, line style, and thickness. Applying style significantly enhances the clarity and aesthetic appeal of the visualization, allowing the rectangle to serve as an effective overlay or background element.

Key styling arguments include: `edgecolor` (defining the color of the border), `facecolor` (defining the fill color inside the shape), `fill` (a boolean flag to enable or disable internal filling), and `lw` (line width, which controls the thickness of the border). These parameters are passed directly into the `Rectangle` constructor alongside the geometric parameters.

The subsequent code demonstrates how to apply robust styling. We set a pink edge, a blue interior, enable filling, and increase the line width (`lw`) to 5 units for greater prominence. This illustrates the flexibility available when presenting data visually.

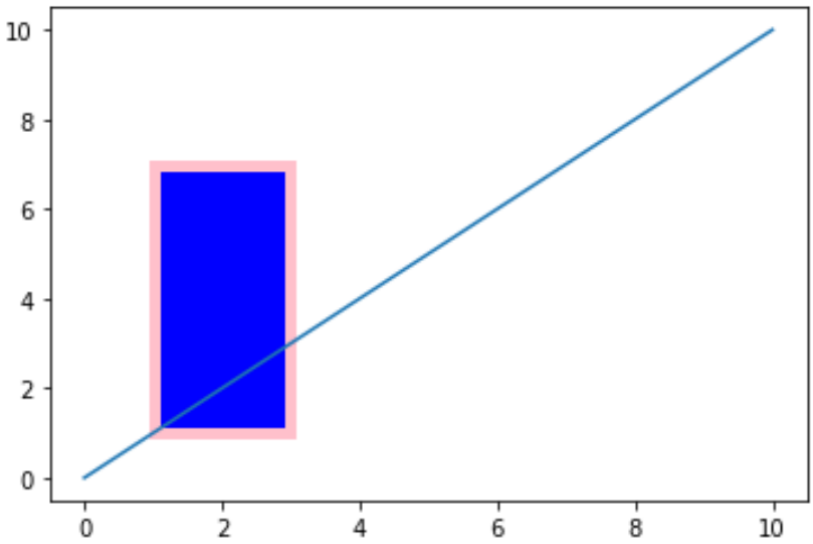
```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

#define Matplotlib figure and axis
fig, ax = plt.subplots()

#create simple line plot
ax.plot(,)

#add rectangle to plot
ax.add_patch(Rectangle((1, 1), 2, 6,
edgecolor = 'pink',
facecolor = 'blue',
fill=True,
lw=5))

#display plot
plt.show()
```



For advanced customization, a comprehensive list of all possible properties and keyword arguments that can be applied to a `Rectangle` patch is available in the Matplotlib official documentation, which provides granular control over nearly every visual aspect of the shape. You can find a complete list of styling properties that you can apply to a rectangle here.

Example 3: Applying Rectangles to Image Data

A particularly powerful application of the `Rectangle` patch is in image processing and analysis, where rectangles are frequently used to delineate objects or areas of interest directly on an image. When working with images, Matplotlib's axes automatically adjust their coordinate system to match the pixel dimensions of the loaded image, simplifying the placement process.

To execute this, we must first load the image using an external library, such as the Python Imaging Library (PIL), and then display it using `plt.imshow()`. The coordinates for the rectangle will correspond directly to the pixel locations within the image.

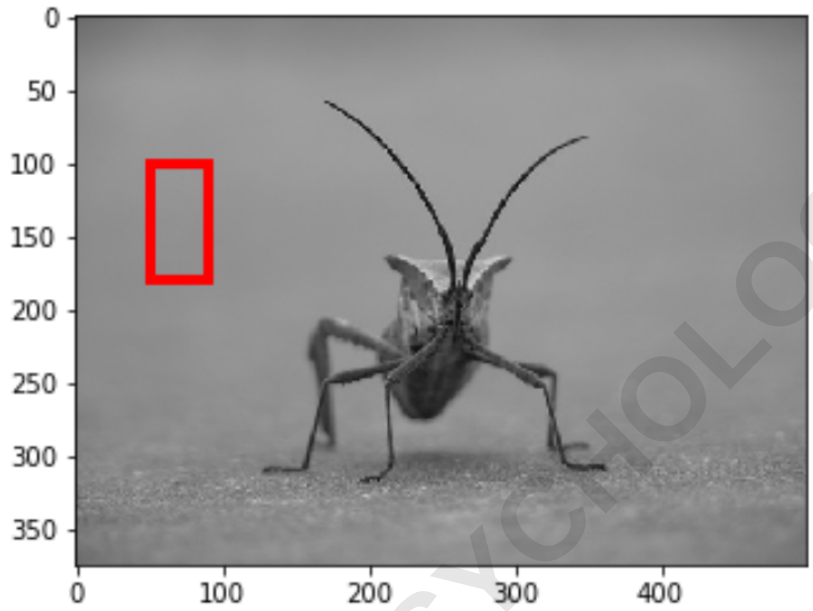
The example below utilizes an image (a stinkbug photo commonly referenced in Matplotlib tutorials) and draws a rectangle around a specific area. Note the use of `plt.gca()`, which retrieves the current active axes, allowing us to attach the patch directly to the displayed image. To ensure the image content remains visible, the `facecolor` is set to 'none'.

To replicate this example, you must download the photo of the stinkbug (e.g., from the Matplotlib image tutorial referenced below) and save it as 'stinkbug.png' in your working directory.

import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image

```
#display the image
plt.imshow(Image.open('stinkbug.png'))

#add rectangle
plt.gca().add_patch(Rectangle((50,100),40,80,
edgecolor='red',
facecolor='none',
lw=4))
```



Exploring Rotation and Advanced Parameters (Angle)

The functionality of the `Rectangle` class extends beyond simple alignment. By utilizing the optional `angle` parameter, users can rotate the rectangle around its anchor point (`xy` coordinates) by a specified number of degrees. This is particularly useful for aligning ROIs with non-orthogonal features in images or data space, or for purely aesthetic purposes in visualization.

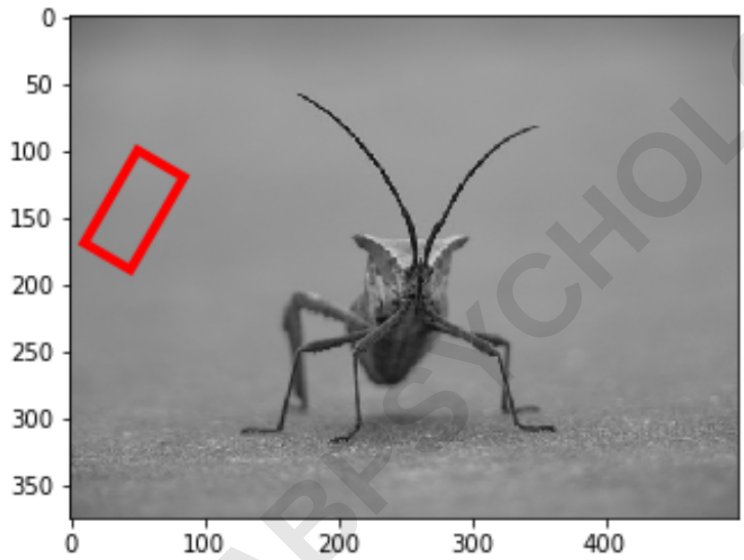
The angle is defined in degrees, and rotation proceeds in a counter-clockwise direction relative to the horizontal axis. For instance, an angle of 30 will tilt the rectangle 30 degrees counter-clockwise. It is crucial to remember that the rotation is centered on the anchor point defined by `xy`.

Building upon the previous image example, we now incorporate the `angle=30` argument into the `Rectangle` definition. All other styling parameters remain constant, demonstrating how easily the rotation parameter can be integrated into complex patch definitions without disrupting other visual properties.

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image

#display the image
plt.imshow(Image.open('stinkbug.png'))

#add rectangle
plt.gca().add_patch(Rectangle((50,100),40,80,
angle=30,
edgecolor='red',
facecolor='none',
lw=4))
```



The use of the `matplotlib.patches.Rectangle` class is a fundamental technique for adding geometric shapes to Matplotlib visualizations. Whether for annotating data points, defining boundaries, or performing basic image processing, mastering the parameters of this function allows for precise and highly customized graphical output.

How to Plot Circles in Matplotlib (With Examples)