What is Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\rmnjs\appdata\
local\programs\python\python39\lib\site-packages (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\rmnjs\appdata\
local\programs\python\python39\lib\site-packages (from matplotlib)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (1.26.0)
Requirement already satisfied: packaging>=20.0 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\rmnjs\
appdata\local\programs\python\python39\lib\site-packages (from
```

```
matplotlib) (2.8.2)
Requirement already satisfied: importlib-resources>=3.2.0 in c:\users\
rmnjs\appdata\local\programs\python\python39\lib\site-packages (from
matplotlib) (6.1.0)
Requirement already satisfied: zipp>=3.1.0 in c:\users\rmnjs\appdata\
local\programs\python\python39\lib\site-packages (from importlib-
resources>=3.2.0->matplotlib) (3.17.0)
Requirement already satisfied: six>=1.5 in c:\users\rmnjs\appdata\
local\programs\python\python39\lib\site-packages (from python-
dateutil>=2.7->matplotlib) (1.16.0)
```

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding the import module statement:

```
import matplotlib
```

Checking Matplotlib Version

The version string is stored under `__version__` attribute.

```
import matplotlib

print(matplotlib.__version__)
```

```
3.8.0
```

Matplotlib Pyplot

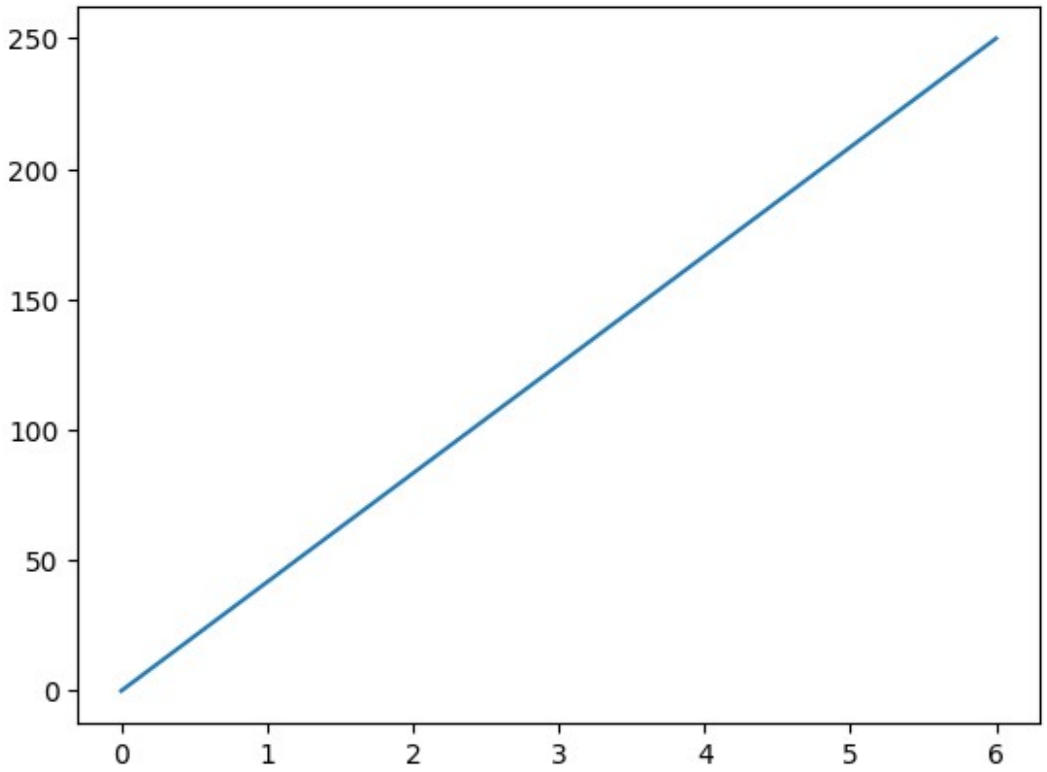
Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

import matplotlib.pyplot as plt Now the Pyplot package can be referred to as plt.

Example: Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



Matplotlib Plotting

Plotting x and y points

The `plot()` function is used to draw points (markers) in a diagram.

By default, the `plot()` function draws a line from point to point.

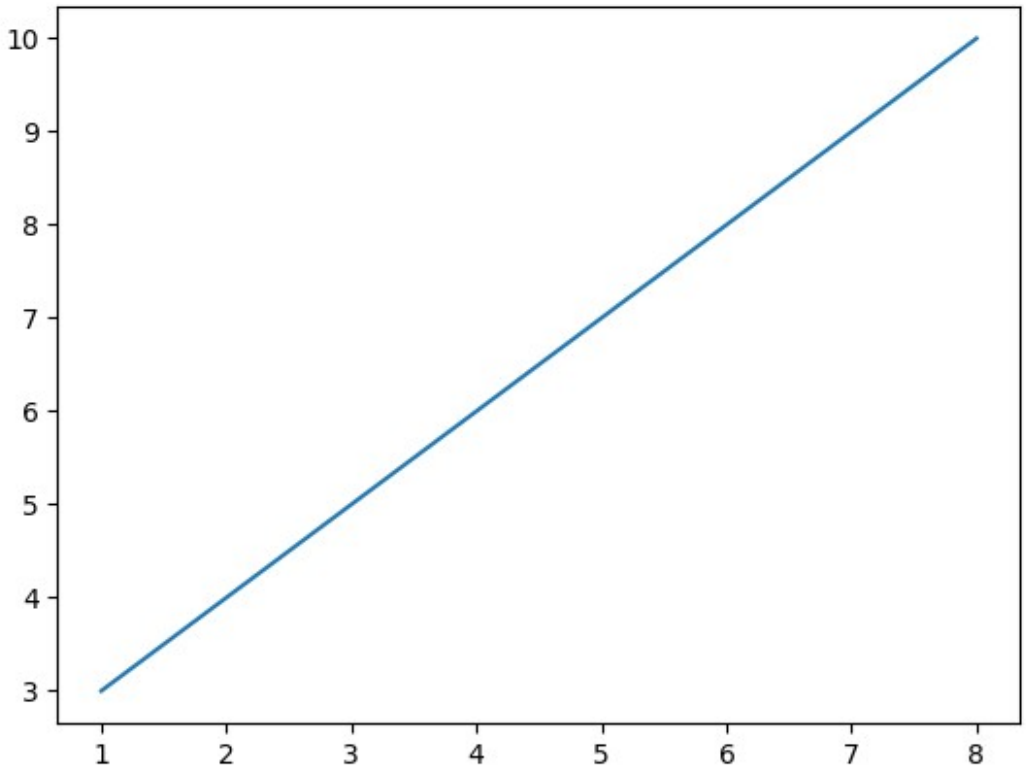
The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the **x-axis**.

Parameter 2 is an array containing the points on the **y-axis**.

If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

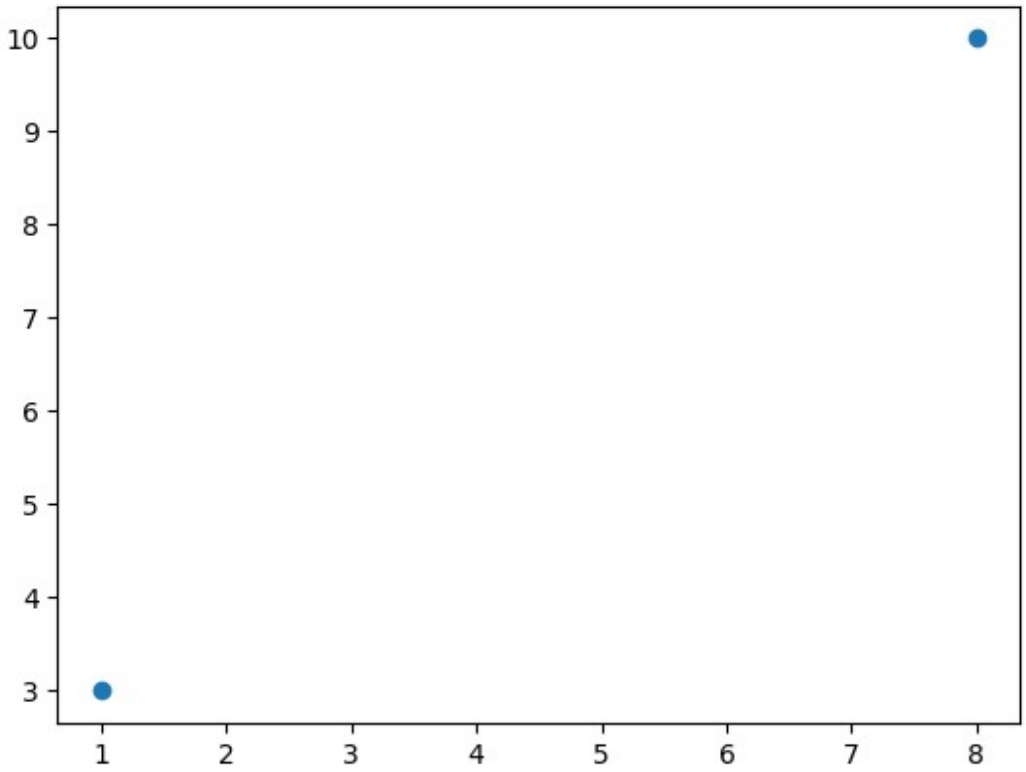


Plotting Without Line

To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

Example Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

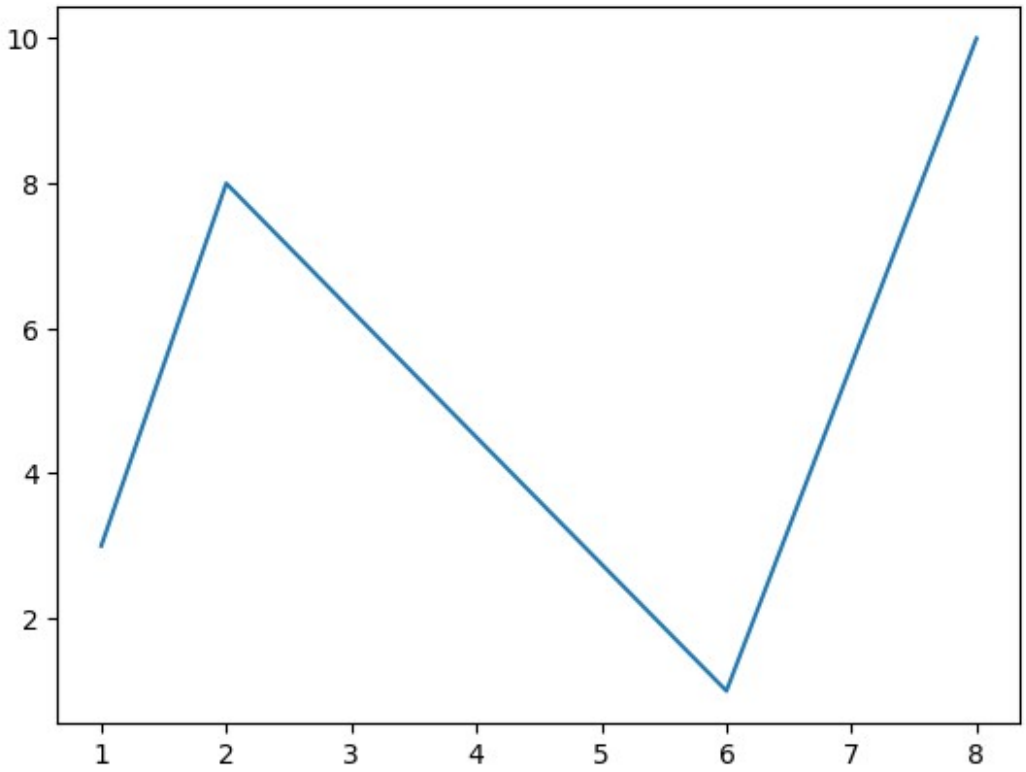


Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



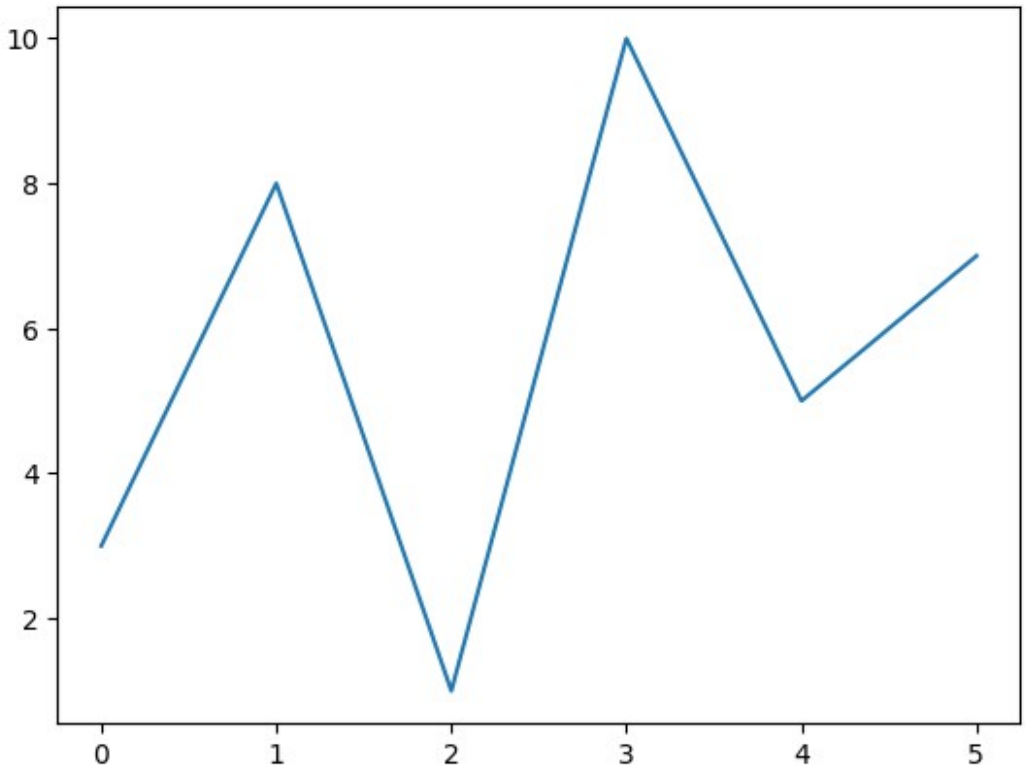
Default X-Points

If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 etc., depending on the length of the y-points.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example Plotting without x-points:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints)
plt.show()
```



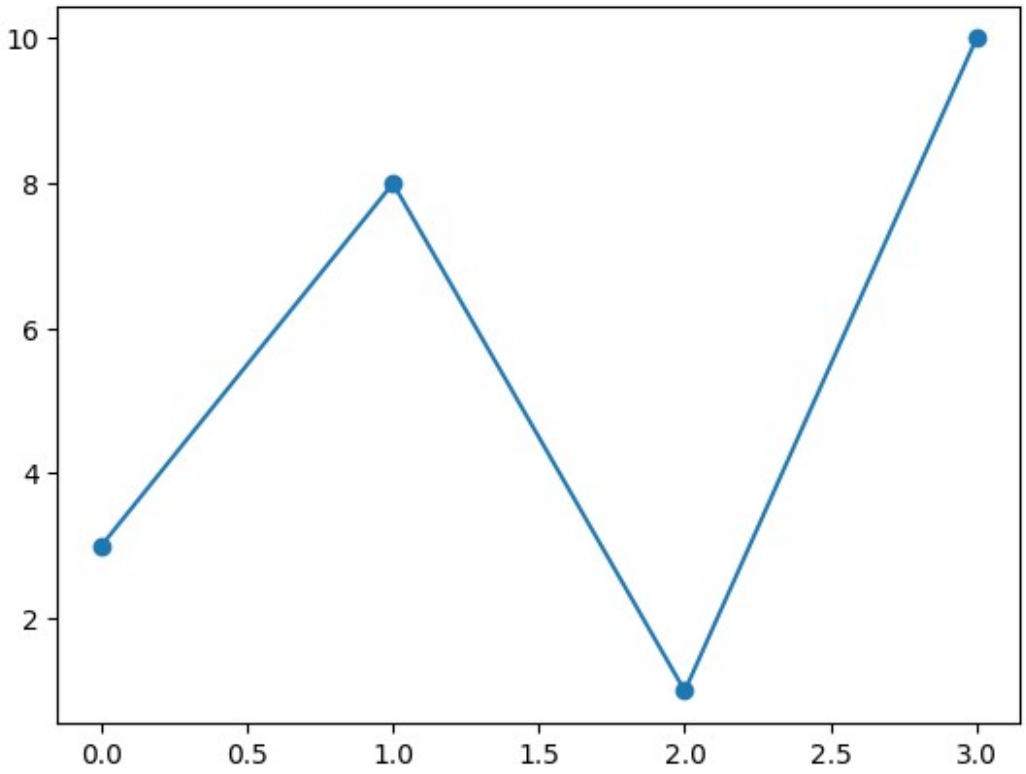
Matplotlib Markers

Markers

You can use the keyword argument marker to emphasize each point with a specified marker:

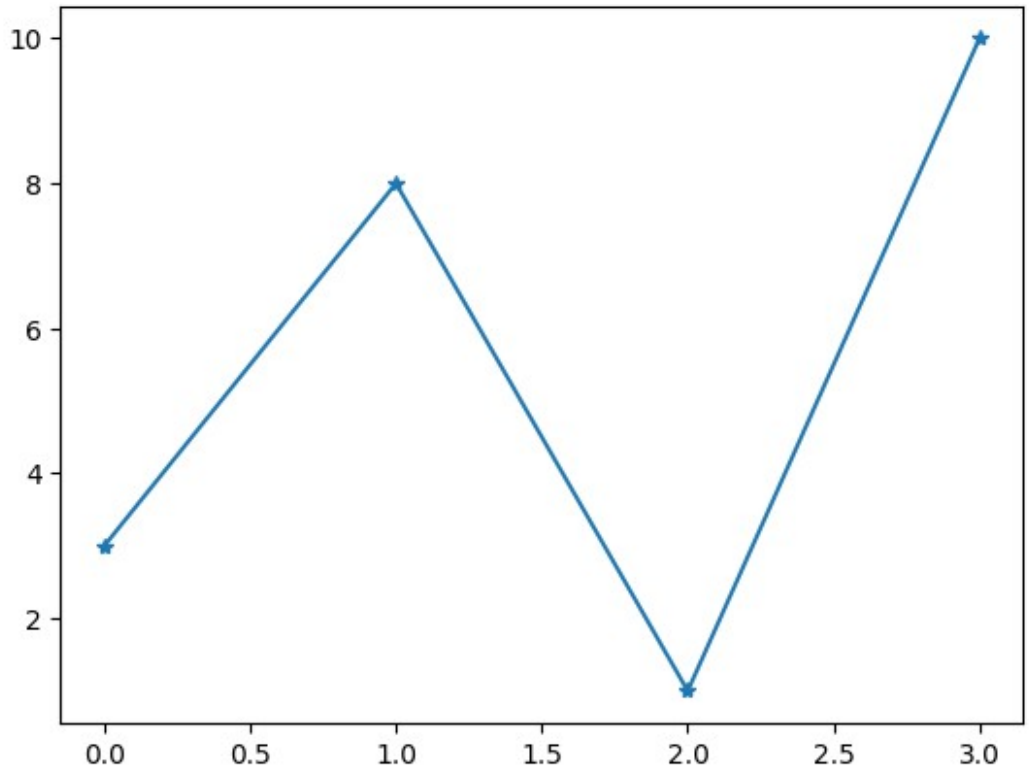
Example Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```




```
plt.plot(ypoints, marker = '*')
```

```
[<matplotlib.lines.Line2D at 0x26e368dd520>]
```



Marker Reference

You can choose any of these markers:

Marker Descriptions

Marker	Description
`'o'`	Circle
`'*'`	Star
`'.'`	Point
`','`	Pixel
`'x'`	X
`'X'`	X (filled)
`'+'`	Plus
`'P'`	Plus (filled)
`'s'`	Square
`'D'`	Diamond
`'d'`	Diamond (thin)
`'p'`	Pentagon
`'H'`	Hexagon
`'h'`	Hexagon

Marker	Description
`'v'`	Triangle Down
`'^'`	Triangle Up
`'<'`	Triangle Left
`'>'`	Triangle Right
`'1'`	Tri Down
`'2'`	Tri Up
`'3'`	Tri Left
`'4'`	Tri Right
`'|'`	Vline
`'_'`	Hline

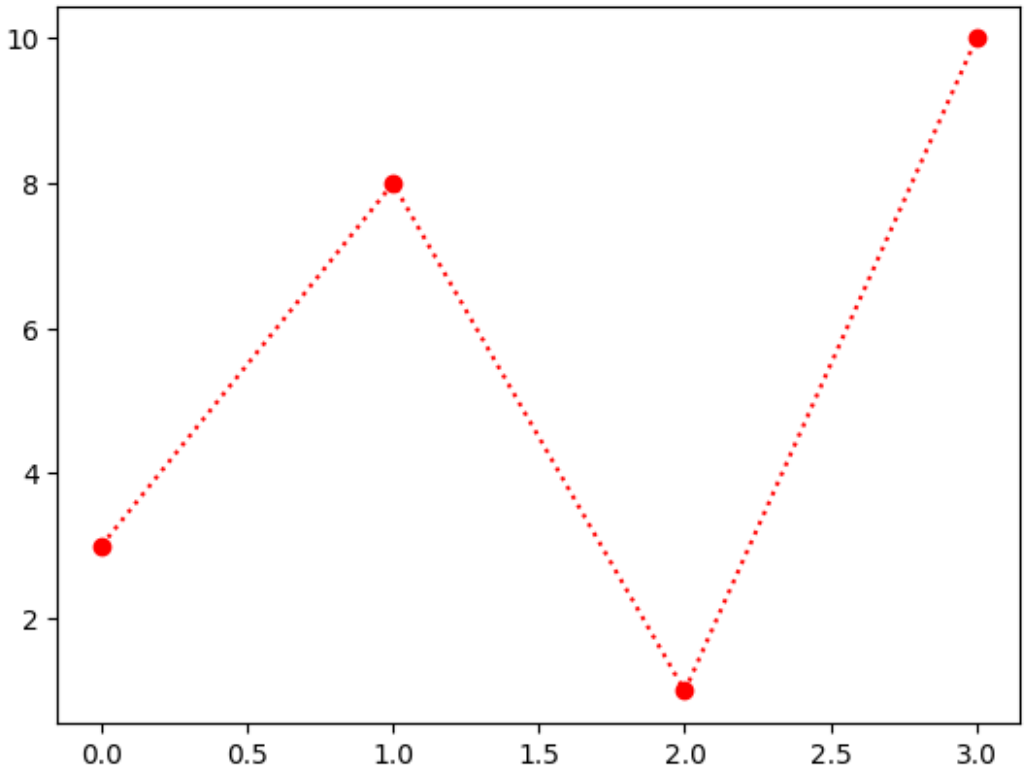
Format Strings fmt

You can also use the shortcut string notation parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

marker|line|color

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```



Line Reference

Line Syntax	Description
`'-'`	Solid line
`':'`	Dotted line
`'--'`	Dashed line
`'-.'`	Dashed/dotted line

Note: If you leave out the line value in the `fmt` parameter, no line will be plotted.

Color Reference

Color Syntax	Description
`'r'`	Red
`'g'`	Green
`'b'`	Blue
`'c'`	Cyan
`'m'`	Magenta
`'y'`	Yellow
`'k'`	Black
`'w'`	White

Marker Size

You can use the keyword argument `markersize` or the shorter version, `ms`, to set the size of the markers.

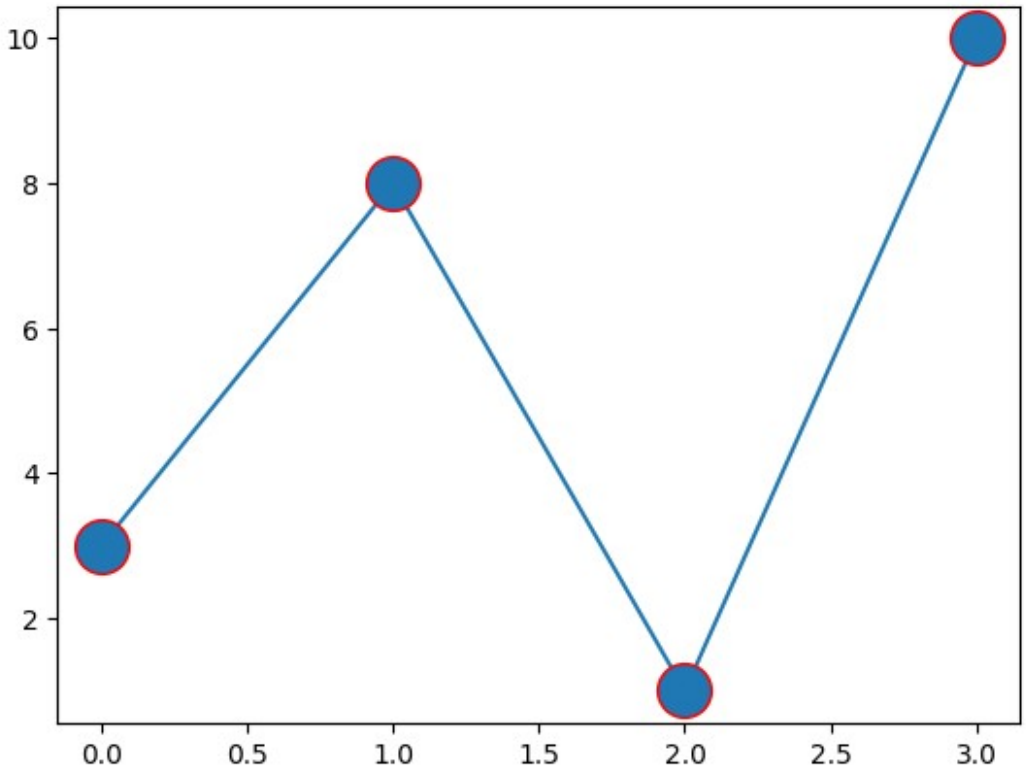
Example

Set the size of the markers to 20:

```python plt.plot(x, y, marker='o', markersize=20)

# Marker Color

You can use the keyword argument `markeredgecolor` or the shorter `mec` to set the color of the edge of the markers and also You can use the keyword argument `markerfacecolor` or the shorter `mfc` to set the color inside the edge of the markers.

Example Set the EDGE color to red:

```
import matplotlib.pyplot as plt
import numpy as np

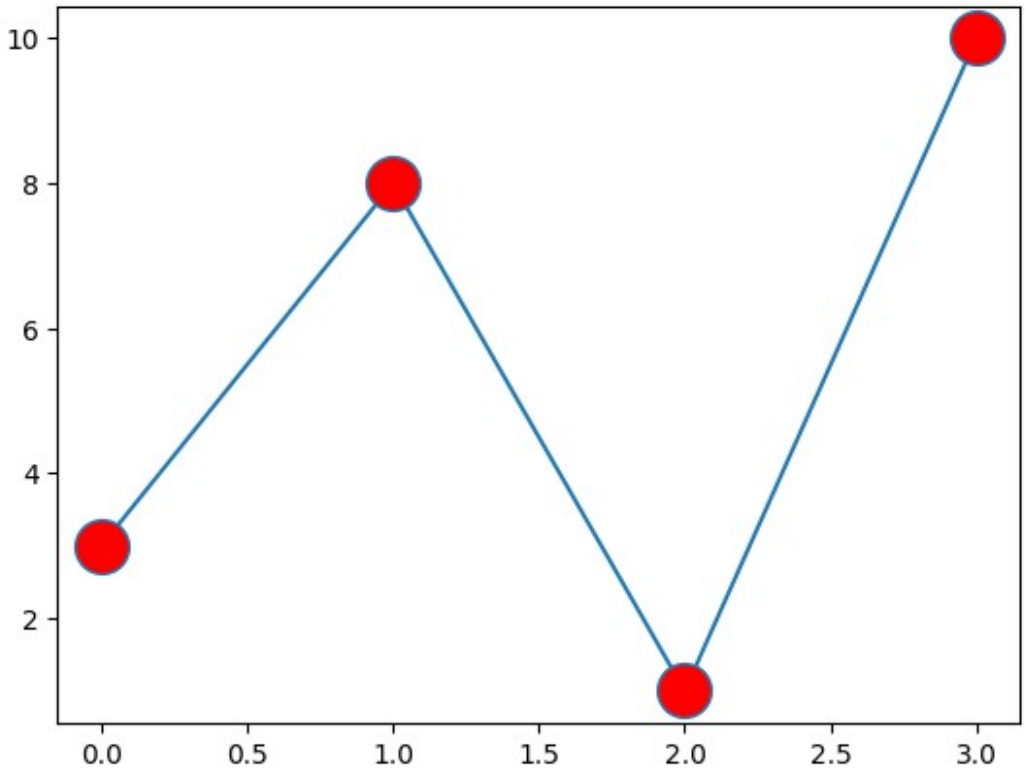
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```
```




```
plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
#plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
```

```
[<matplotlib.lines.Line2D at 0x26e346d7370>]
```



Matplotlib Line

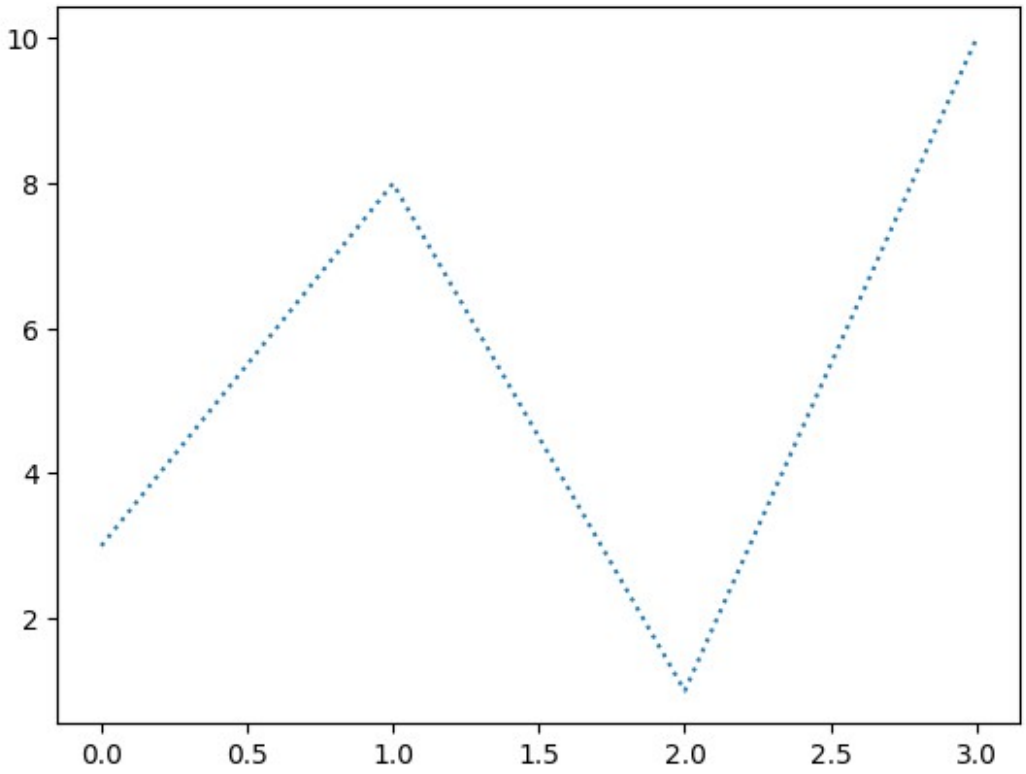
Linestyle

You can use the keyword argument `linestyle`, or shorter `ls`, to change the style of the plotted line:

Line Color

You can use the keyword argument `color` or the shorter `c` to set the color of the line:

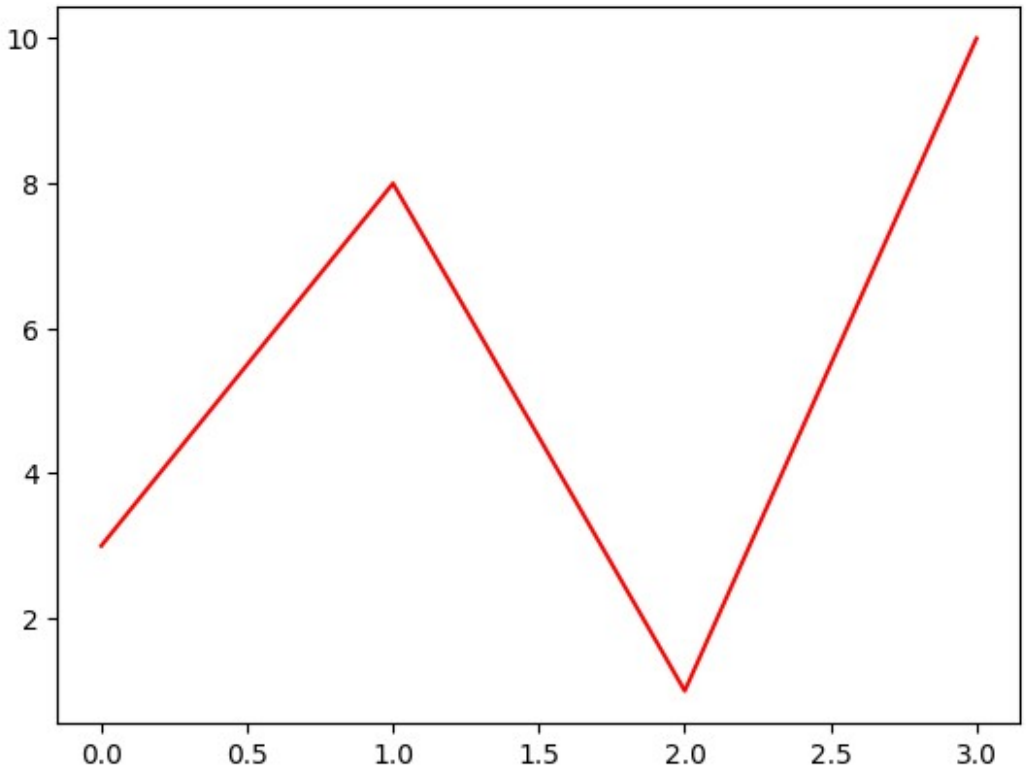
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



Line Style	Code	Description
Solid Line	`'-'` or `'solid'`	Default line style
Dotted Line	`':'` or `'dotted'`	Dotted line
Dashed Line	`'--'` or `'dashed'`	Dashed line
Dash-dot Line	`'-.'` or `'dashdot'`	Dash-dot line
No Line	`''` or `' '` or `'None'`	No line

```
plt.plot(ypoints, color = 'r')
#plt.plot(ypoints, c = '#4CAF50')
#plt.plot(ypoints, c = 'hotpink')
```

```
[<matplotlib.lines.Line2D at 0x26e34728fd0>]
```

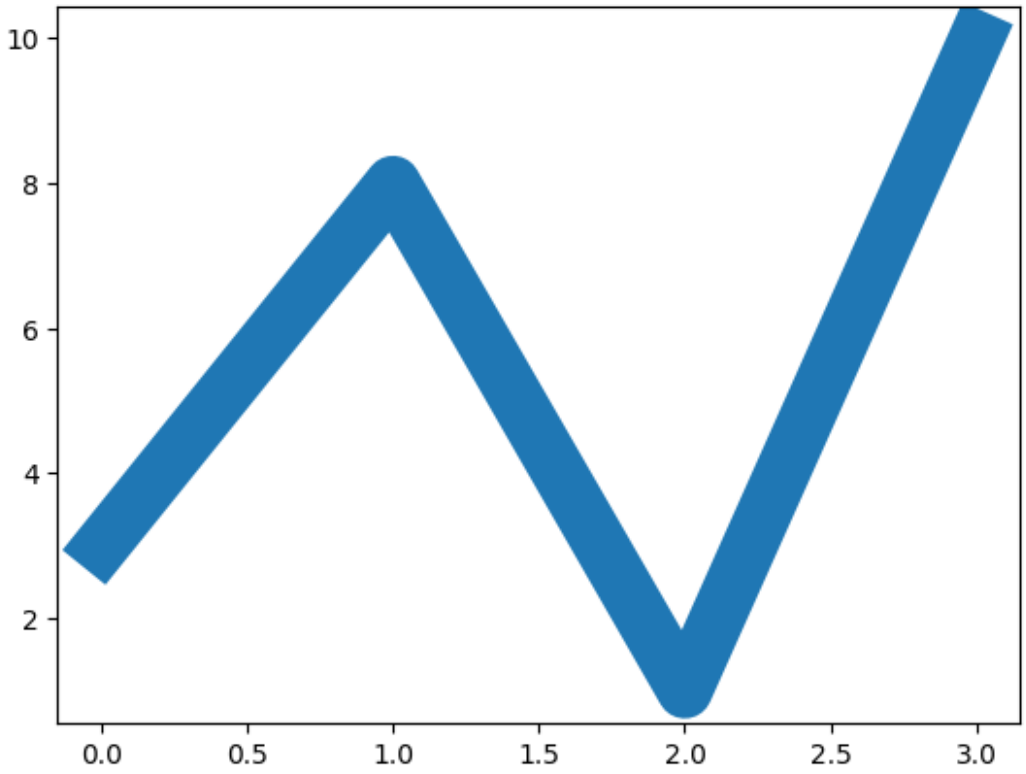


Line Width

You can use the keyword argument `linewidth`or the shorter `lw` to change the width of the line.

The value is a floating number, in points:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```

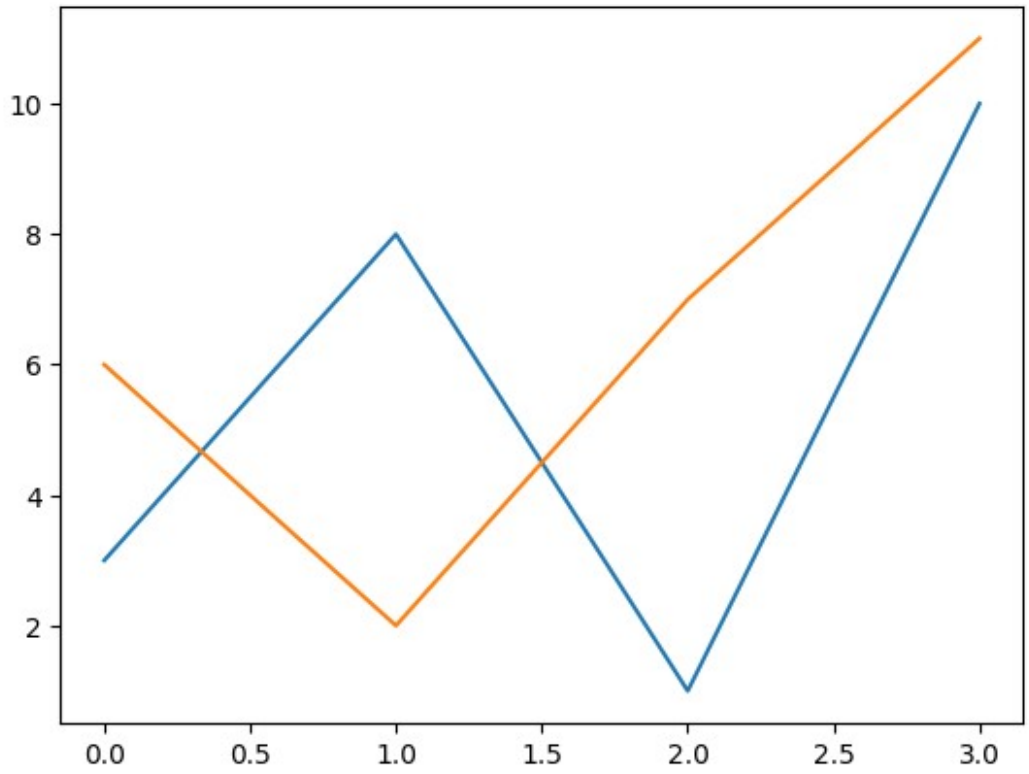


Multiple Lines

You can plot as many lines as you like by simply adding more `plt.plot()` functions:

Example Draw two lines by specifying a `plt.plot()` function for each line:

```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```



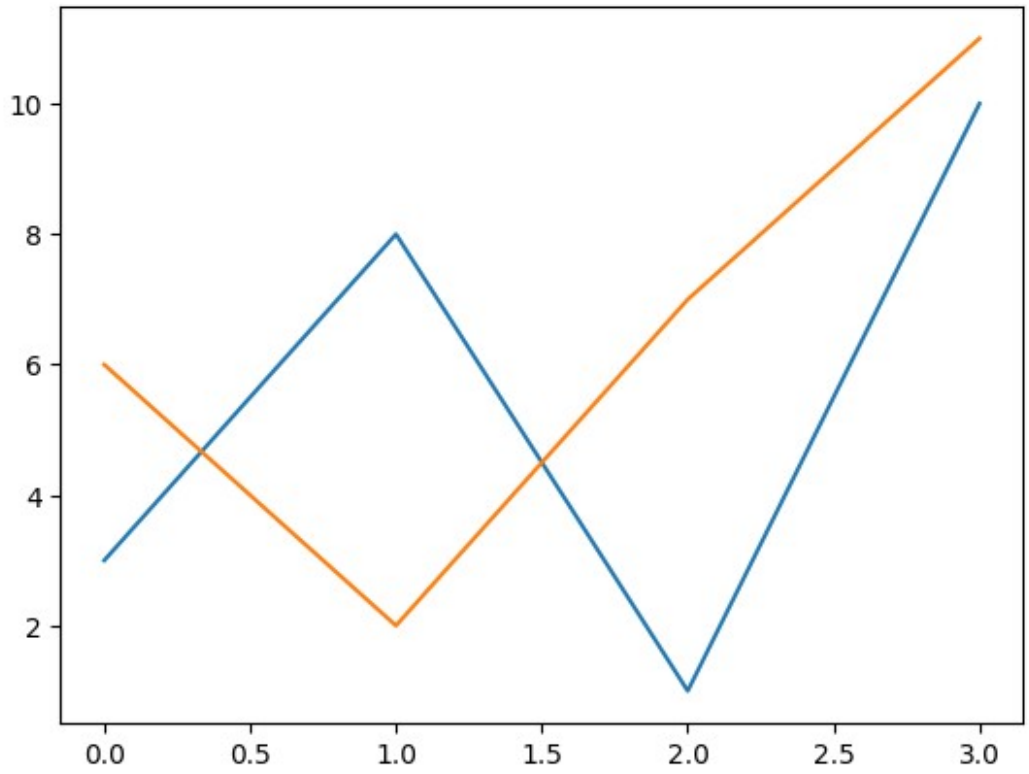
You can also plot many lines by adding the points for the x- and y-axis for each line in the same plt.plot() function.

(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)

The x- and y- values come in pairs:

Example Draw two lines by specifiyng the x- and y-point values for both lines:

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```

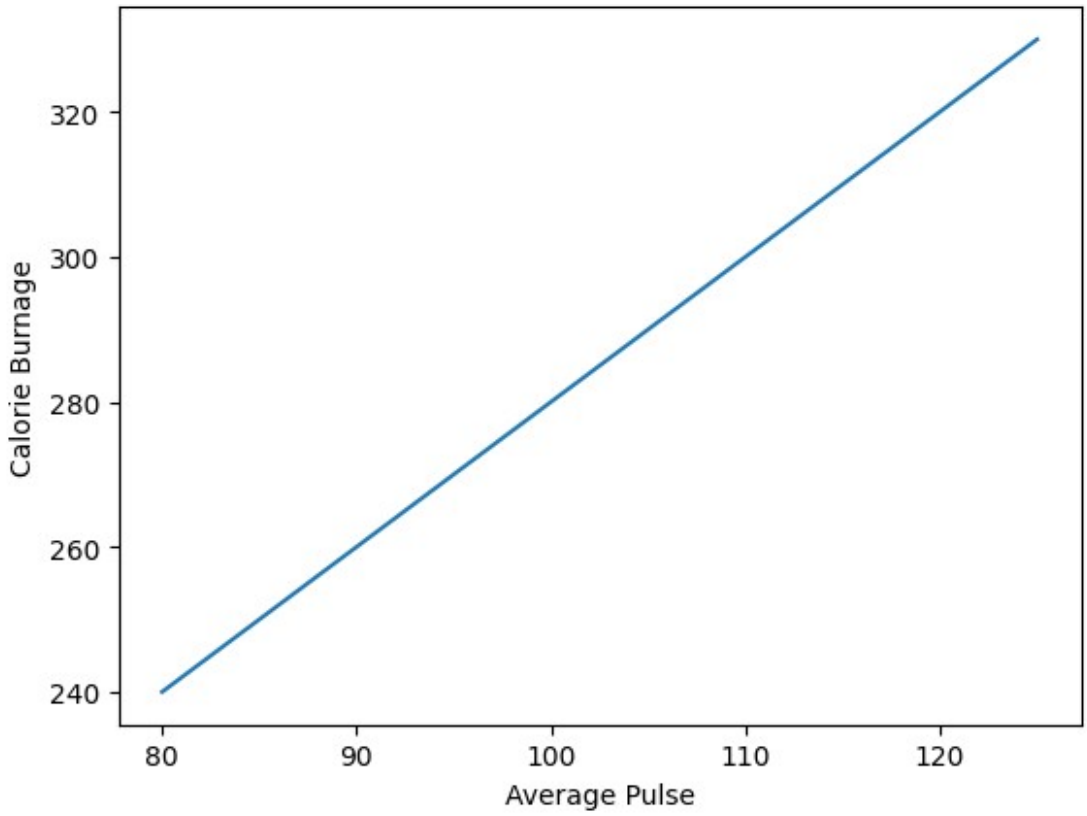


Matplotlib Labels and Title

Create Labels for a Plot

With Pyplot, you can use the `xlabel()` and `ylabel()` functions to set a label for the x- and y-axis.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



Create a Title for a Plot

With Pyplot, you can use the `title()` function to set a title for the plot.

Set Font Properties for Title and Labels

You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels.

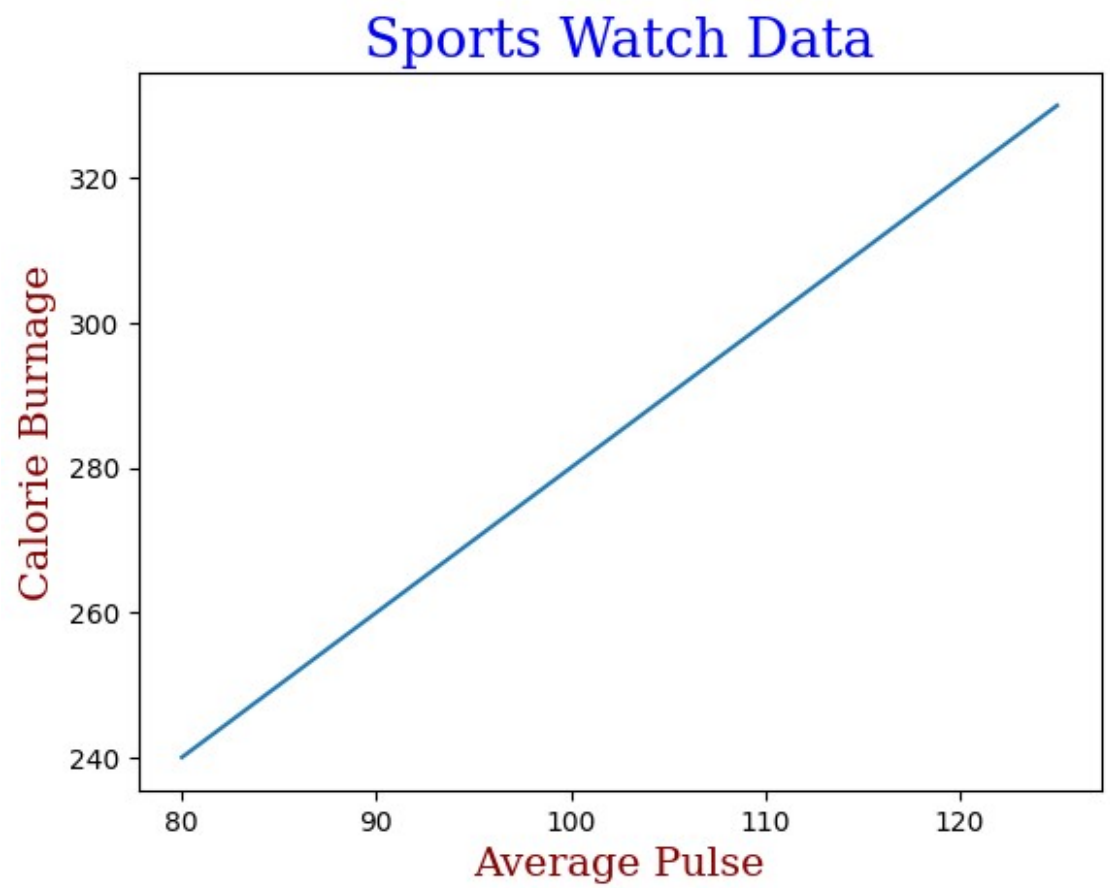
Position the Title

You can use the `loc` parameter in `title()` to position the title.

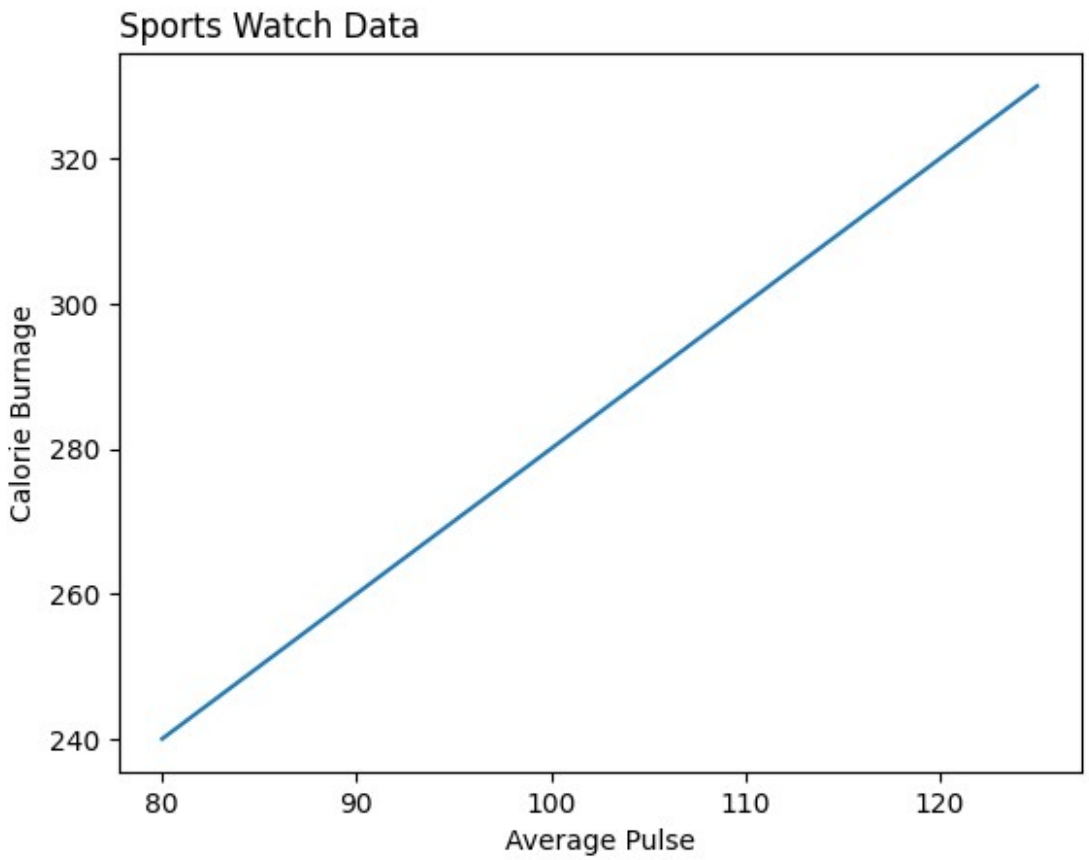
Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
```

```
plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```




```
#Position the title to the left:

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.show()
```



Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

With Pyplot, you can use the `grid()` function to add grid lines to the plot.

Specify Which Grid Lines to Display

You can use the `axis` parameter in the `grid()` function to specify which grid lines to display.

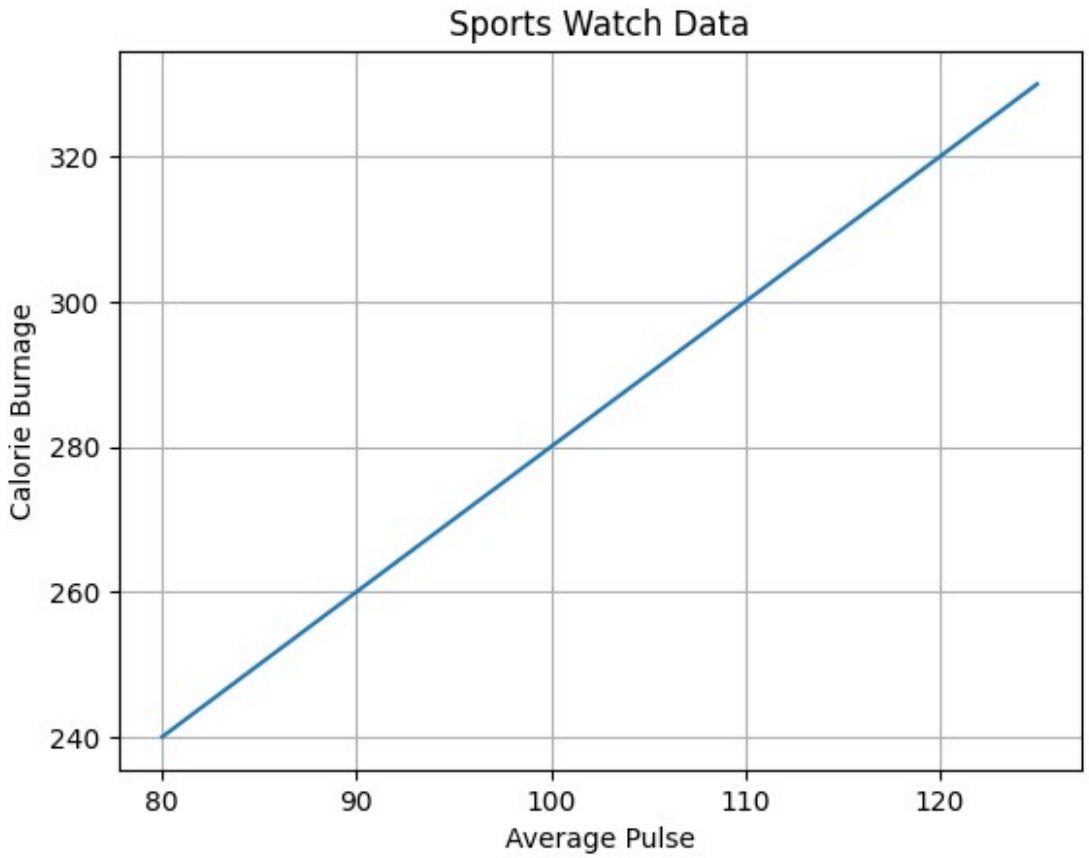
Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

Set Line Properties for the Grid

You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

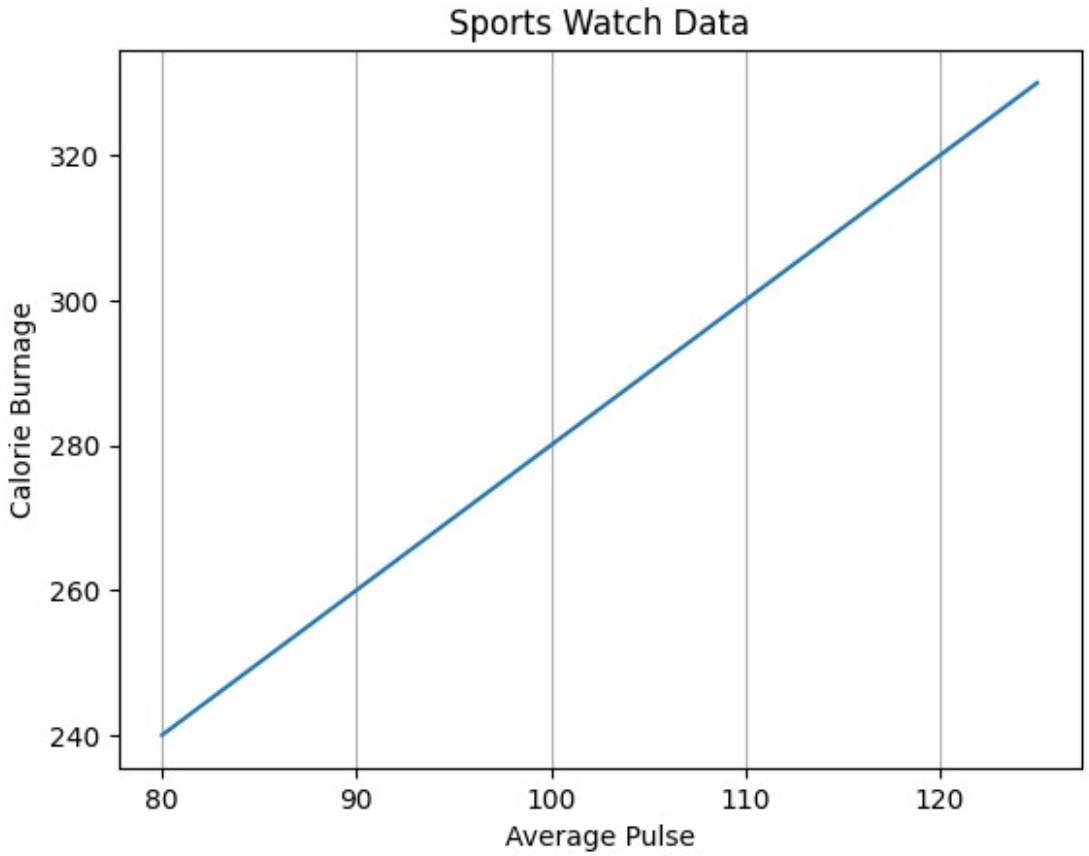
```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```



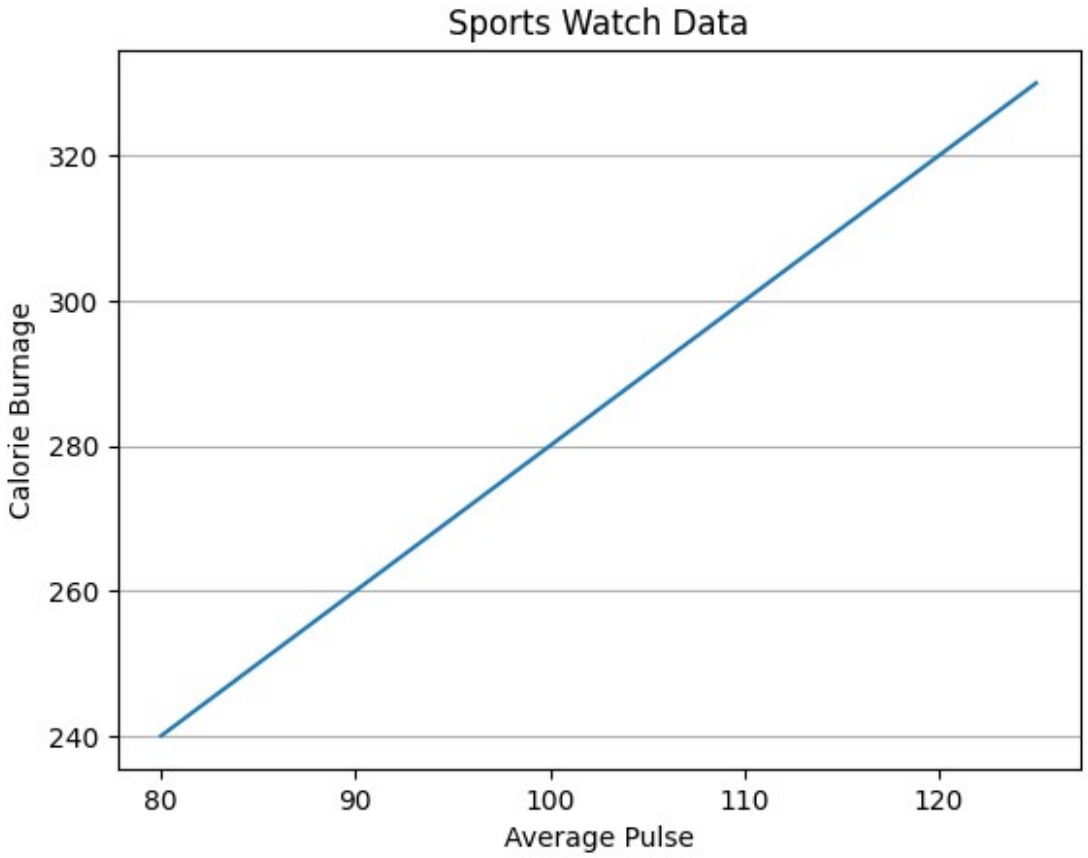

```
#Display only grid lines for the x-axis:
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
```

```
plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```

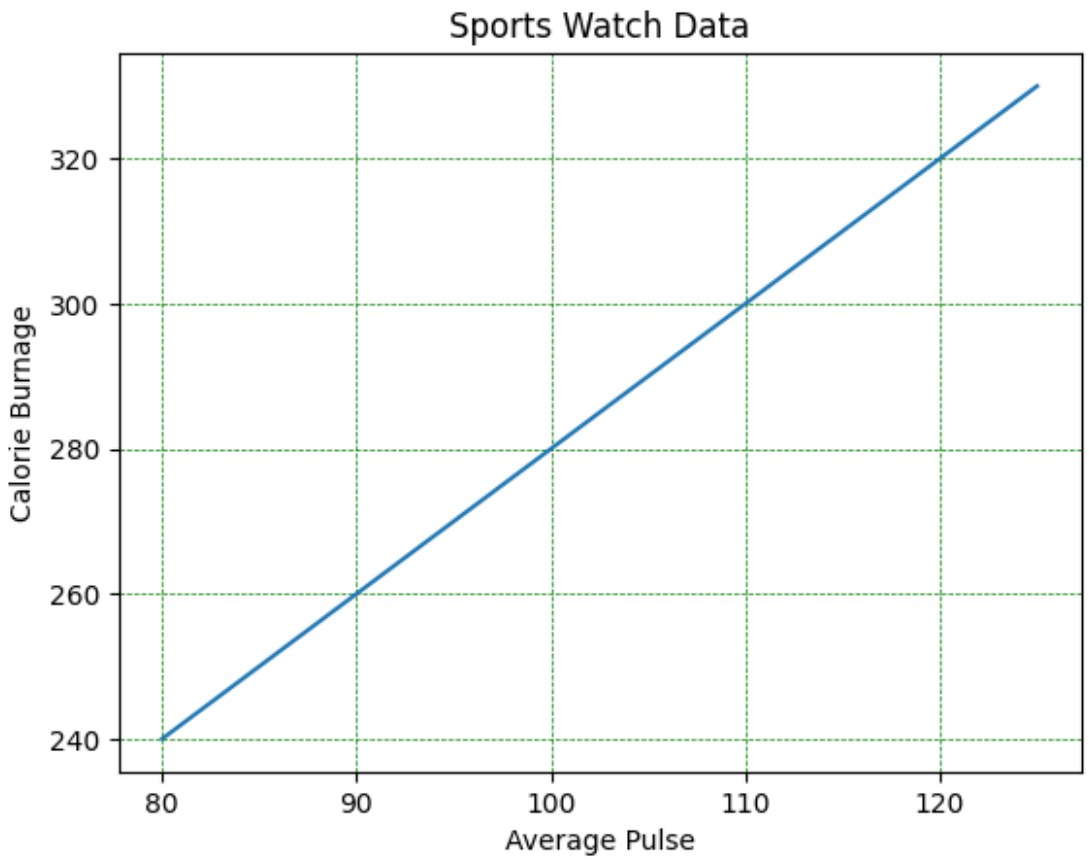



```
#Display only grid lines for the y-axis:

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'y')
```

```
plt.show()
```




```
#Set the line properties of the grid:

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```



Matplotlib Subplot

Display Multiple Plots

With the `subplot()` function you can draw multiple plots in one figure:

The subplot() Function

The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the first and second argument.

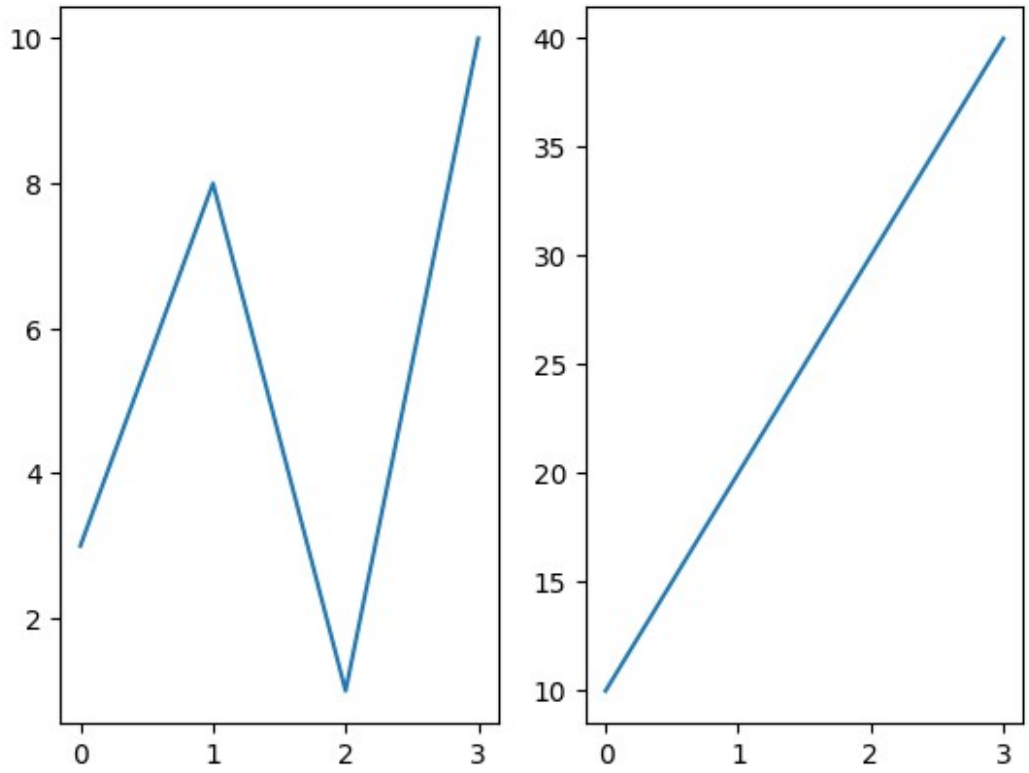
The third argument represents the index of the current plot.

```PYTHON
plt.subplot(1, 2, 1)

#the figure has 1 row, 2 columns, and this plot is the first plot.

plt.subplot(1, 2, 2)
```

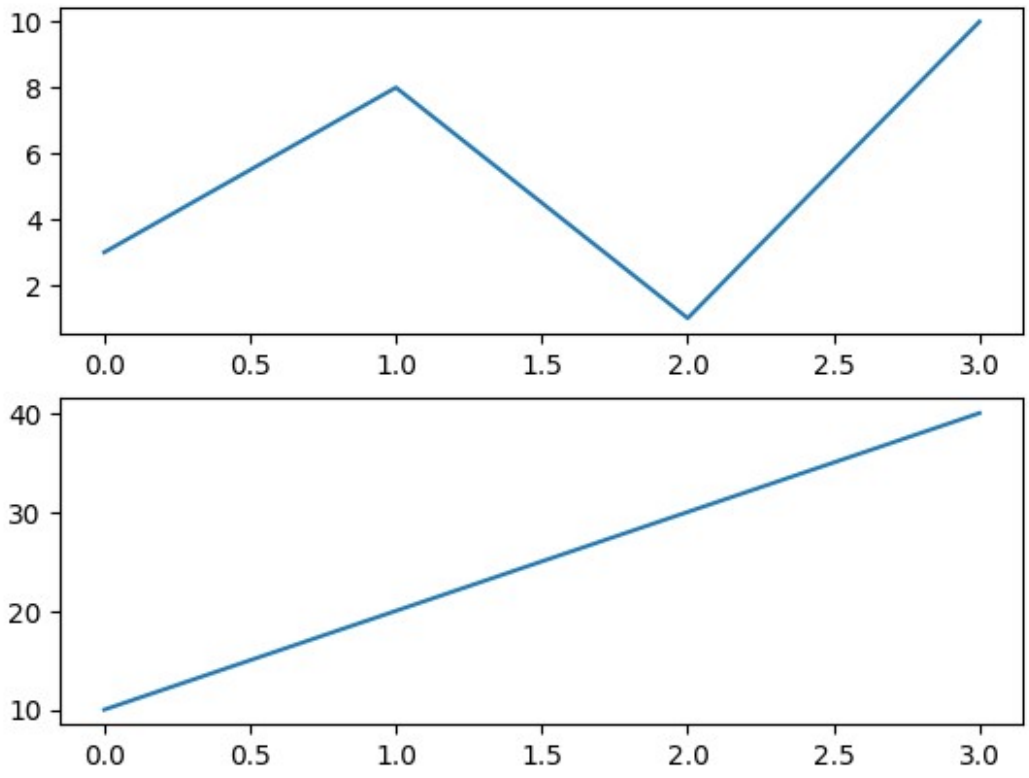
#the figure has 1 row, 2 columns, and this plot is the second plot.

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

```
#Draw 2 plots:

import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```

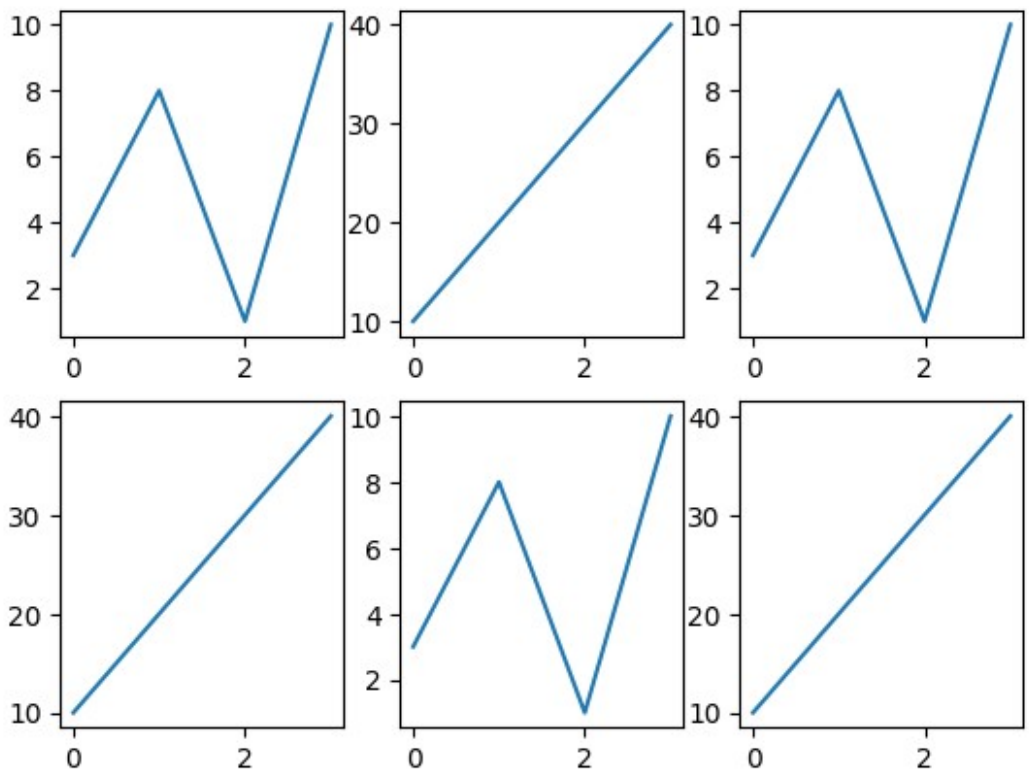


```
#Example
#Draw 2 plots on top of each other:

import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```




```
#Draw 6 plots:

import matplotlib.pyplot as plt
```

```
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



Title

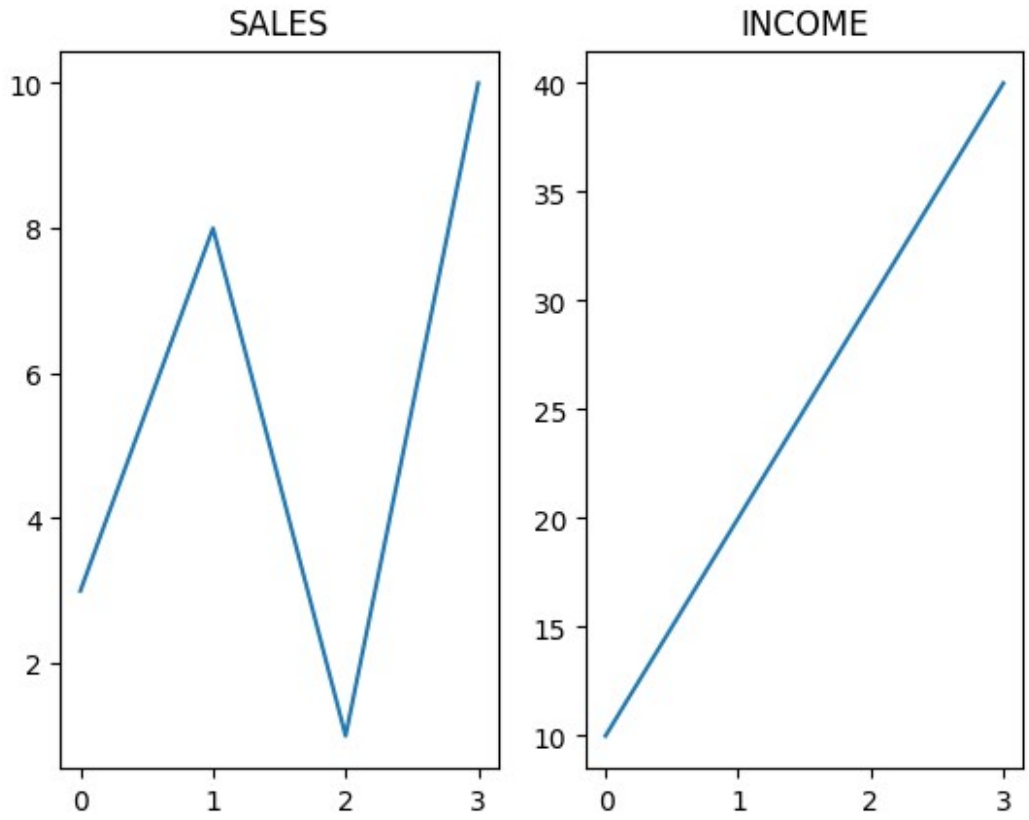
You can add a title to each plot with the `title()` function:

Super Title

You can add a title to the entire figure with the `suptitle()` function:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
```

```
plt.title("INCOME")

plt.show()
```




```
#Add a title for the entire figure:

import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
```

```
plt.suptitle("MY SHOP")
plt.show()
```