Matplotlib

```
pip install matplotlib
```

If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

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__)
```

Pyplot

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

```
import matplotlib.pyplot as plt
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.

Draw a line in a diagram from position (1, 3) to 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)
plt.show()
```

The x-axis is the horizontal axis.

The y-axis is the vertical axis.

Plotting Without Line

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

Markers

Markers

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

```
import matplotlib.pyplot as plt

import numpy as np

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

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

plt.show()
```

Mark each point with a star:

```
...

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

...
```

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**
'v'	**Triangle Down**
'^'	**Triangle Up**

'<'	**Triangle Left**
'>'	**Triangle Right**
'1'	**Tri Down**
'2'	**Tri Up**
'3'	**Tri Left**
'4'	**Tri Right**
'|'	**Vline**
'_'	**Hline**

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

The marker value can be anything from the Marker Reference above.

The line value can be one of the following:

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.

The short color value can be one of the following:

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:

Set the size of the markers to 20:

```
import matplotlib.pyplot as plt

import numpy as np


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


plt.plot(ypoints, marker = 'o', ms = 20)

plt.show()
```

Marker Color

You can use the keyword argument `markeredgecolor` or the shorter `mec` to set the color of the *edge* of the markers:

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()
```

You can use the keyword argument `markerfacecolor` or the shorter `mfc` to set the color inside the edge of the markers:

Set the FACE 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, mfc = 'r')

plt.show()
```

Use *both* the `mec` and `mfc` arguments to color the entire marker:

Set the color of both the *edge* and the *face* 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', mfc =
'r')

plt.show()
```

Mark each point with a beautiful green color:

```
...

plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc
= '#4CAF50')

...
```

Mark each point with the color named "hotpink":

```
...

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc
= 'hotpink')

...
```

Matplotlib Line

Linestyle

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

Use a dotted line:

```
import matplotlib.pyplot as plt

import numpy as np

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

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

plt.show()
```

Use a dashed line:

```
plt.plot(ypoints, linestyle = 'dashed')
```

Shorter Syntax

The line style can be written in a shorter syntax:

`linestyle` **can be written as** `ls`**.**

`dotted` **can be written as** `:`**.**

`dashed` **can be written as** `--`**.**

```
plt.plot(ypoints, ls = ':')
```

Line Styles

You can choose any of these styles:

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

Line Color

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

Set the line color to red:

```
import matplotlib.pyplot as plt

import numpy as np

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

plt.plot(ypoints, color = 'r')

plt.show()
```

Plot with a beautiful green line:

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

Plot with the color named "hotpink":

```
...
plt.plot(ypoints, c = 'hotpink')
...
```

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:

Plot with a 20.5pt wide line:

```
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:

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:

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.

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.

Add labels to 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.

Add a plot title and labels 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.title("Sports Watch Data")

plt.xlabel("Average Pulse")

plt.ylabel("Calorie Burnage")

plt.show()
```

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.

Set font properties for the title and labels:

```
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

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

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

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.

Add grid lines to the plot:

```
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 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()
```

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'.

```
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 Line Properties for the Grid

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

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:

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()
```

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.

```
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 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()
```

You can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.
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:

2 plots, with titles:

```
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()
```

Super Title

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

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()
```