Matplotlib

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.

Installing Matplotlib

If not installed, use:

pip install matplotlib

Import Matplotlib

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

import matplotlib

Now Matplotlib is imported and ready to use:

Checking Matplotlib Version

The version string is stored under __version__ attribute.

Example

```
import matplotlib
print(matplotlib.__version__)
```

Importing Pyplot

```
import matplotlib.pyplot as plt
```

Pyplot

pyplot is a **module in Matplotlib** used for **creating visualizations** like line plots, bar charts, histograms, and scatter plots. It provides an **interface similar to MATLAB** and makes it easy to plot and customize graphs.

- 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.

Markers

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

Marker Reference

You can choose any of these markers:

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

Line Reference

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

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:

```
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**:

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()
```

Result:

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

Example

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()
```

Result:

Line Color

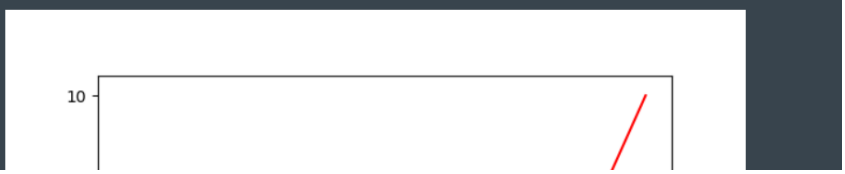
You can use the keyword argument color or the shorter **c** to **set the color of the line**:

Example

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()
```

Result:



Line Width

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

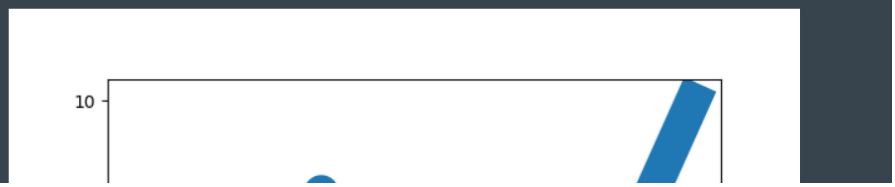
The value is a floating number, in points:

Example

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()
```

Result:



Multiple Lines

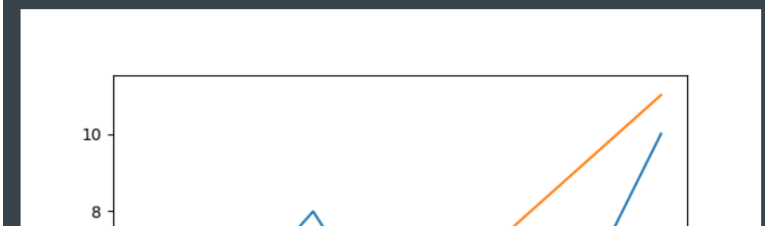
You can plot as many lines as you like by simply adding more **plt.plot()** functions:

Example

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

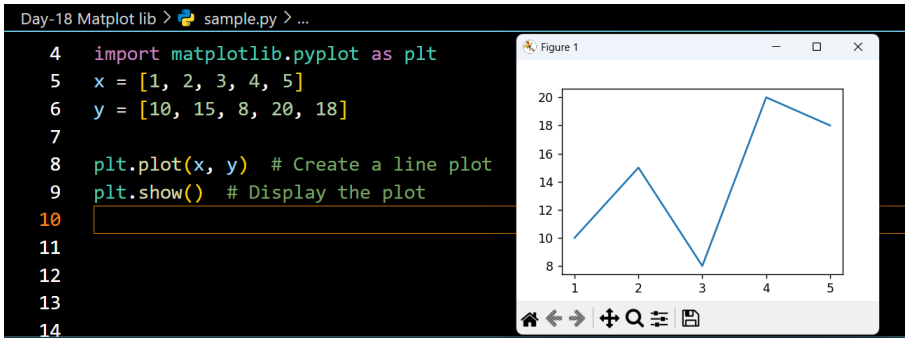
```
import matplotlib.pyplot as plt
import numpy as np

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

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

plt.show()
```

Result:



Basic Structure of Pyplot

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 15, 8, 20, 18]

plt.plot(x, y)  # Create a line plot
plt.show()  # Display the plot
```



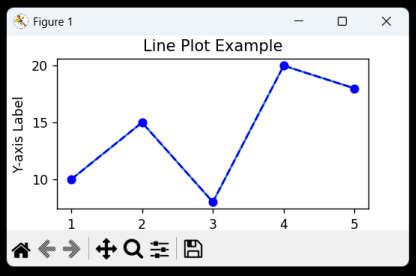
Adding Labels & Title

- With Pyplot, you can use the **xlabel() and ylabel() functions to set a label for the x- and y-axis**.
- With Pyplot, you can use the **title()** function to set a title for the plot.
- You can use the loc parameter in **title()** to **position the title**. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

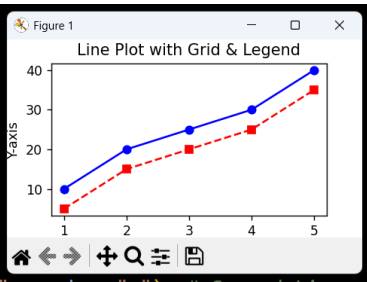
plt.plot(x, y, marker="o", linestyle="--", color="b")

plt.xlabel("X-axis Label") # Label for X-axis
plt.ylabel("Y-axis Label") # Label for Y-axis
plt.title("Line Plot Example") # Title of the Graph
plt.show()

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 15, 8, 20, 18]

plt.plot(x, y)  # Create a line plot
#plt.show()  # Display the plot
plt.plot(x, y, marker="o", linestyle="--", color="b")

plt.xlabel("X-axis Label")  # Label for X-axis
plt.ylabel("Y-axis Label")  # Label for Y-axis
plt.title("Line Plot Example")  # Title of the Graph
plt.show()
```




```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y1 = [10, 20, 25, 30, 40]
y2 = [5, 15, 20, 25, 35]

# Create the plot
plt.plot(x, y1, label="Line 1", color="blue", marker="o")
plt.plot(x, y2, label="Line 2", color="red", linestyle="--", marker="s")  # Second Line

# Add Title & Labels
plt.title("Line Plot with Grid & Legend")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
```



Adding a Grid & Legend

- In **Matplotlib**, you can add a **grid** using **plt.grid(True)** and a **legend** using **plt.legend()**
- 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'.
- You can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*).

```python
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y1 = [10, 20, 25, 30, 40]
y2 = [5, 15, 20, 25, 35]

# Create the plot
plt.plot(x, y1, label="Line 1", color="blue", marker="o")  # First Line
plt.plot(x, y2, label="Line 2", color="red", linestyle="--", marker="s")  # Second Line

# Add Title & Labels
plt.title("Line Plot with Grid & Legend")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# # Add Grid
plt.grid(True)  # Show grid lines

# # Add Legend
plt.legend()  # Show legend

# Show the plot
plt.show()
```

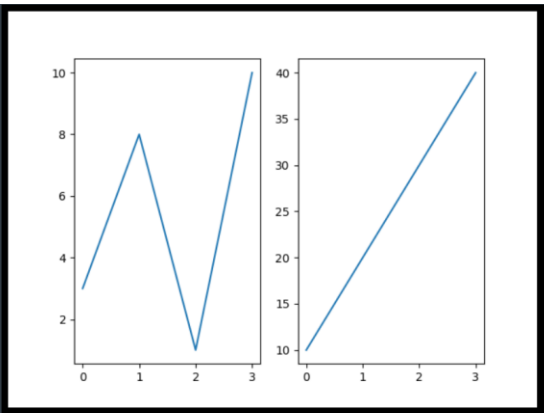
Display Multiple Plots

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
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

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()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



Different Plot Types

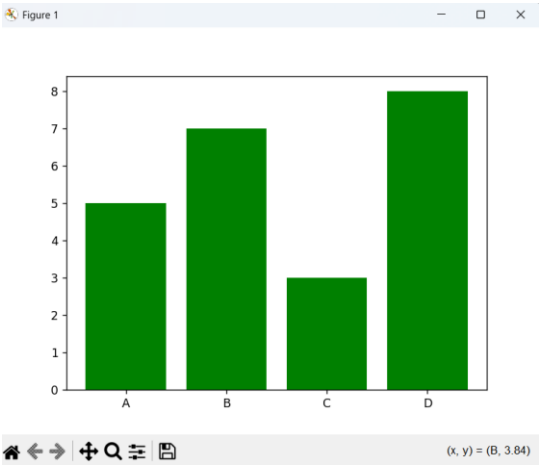
Line Plot

plt.plot(x, y, marker="o", linestyle="-", color="blue")
plt.show()

Bar Chart

- With Pyplot, you can use the **bar() function to draw bar graphs**
- If you want the bars to be **displayed horizontally** instead of vertically, use the **barh()** function
- The **bar() and barh()** take the keyword argument color to set the color of the bars
- plt.bar(x, y, color = "red")
- The **bar()** takes the keyword argument **width** to **set the width of the bars**
- The **barh()** takes the keyword argument **height** to **set the height of the bars**

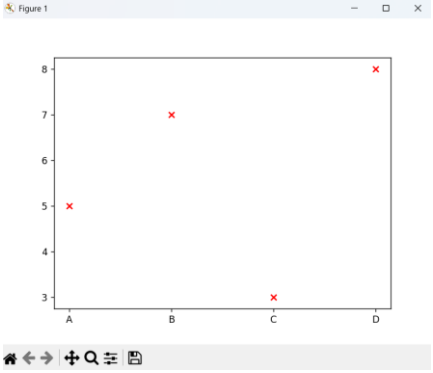
```
categories = ["A", "B", "C", "D"]
values = [5, 7, 3, 8]

plt.bar(categories, values, color="green")
plt.show()
```



Scatter Plot

- With Pyplot, you can use the **scatter()** function to draw a scatter plot.
- The **scatter()** function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis
- You can even set a specific **color for each dot** by using an array of colors as value for the **c argument**
- You can change the **size of the dots** with the **s** argument.
- You can **adjust the transparency** of the dots with the **alpha** argument.

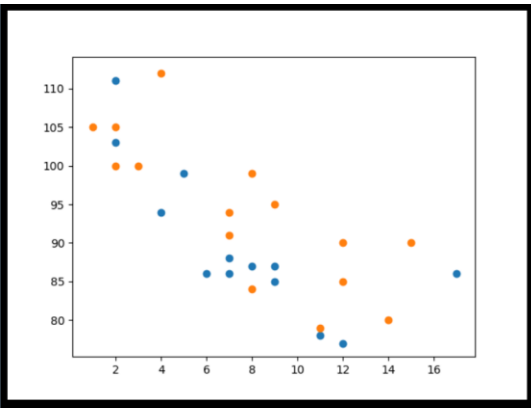
```
plt.scatter(categories, values, color="red", marker="x")
plt.show()
```



Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

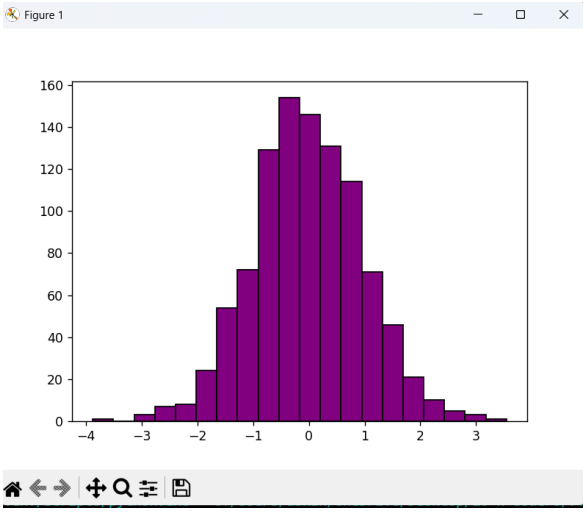
```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



Histogram

- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- The **hist()** function will read the array and produce a histogram

```
import numpy as np

data = np.random.randn(1000)  # Random data
plt.hist(data, bins=20, color="purple", edgecolor="black")
plt.show()
```



Pie Chart

- With Pyplot, you can use the **pie()** function to draw pie charts
- Add labels to the pie chart with the **labels** parameter.
- The labels parameter must be an array with one label for each wedge
- Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.
- The **explode parameter**, if specified, and not None, must be an array with one value for each wedge.
- **Add a shadow** to the pie chart by setting the **shadows** parameter to True

- To **add a list of explanation for each wedge**, use the **legend()** function
- To **add a header to the legend**, add the **title parameter to the legend function**.

```
labels = ["Python", "Java", "C++", "JavaScript"]
sizes = [40, 25, 20, 15]

plt.pie(sizes, labels=labels, autopct="%1.1f%%", colors=["blue", "red", "green", "yellow"])
plt.title("Pie Chart Example")
plt.show()
```