ABDELGHAFOR'S VIRTUAL INTERNSHIP

# *DATA ANALYSIS PROGRAM*

SESSION (2)

PREPARED BY : MARK KOSTANTINE

# *Agenda Overview*

# *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.

## Installation and Import of Matplotlib

- Install it using this command : pip install **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__)
```

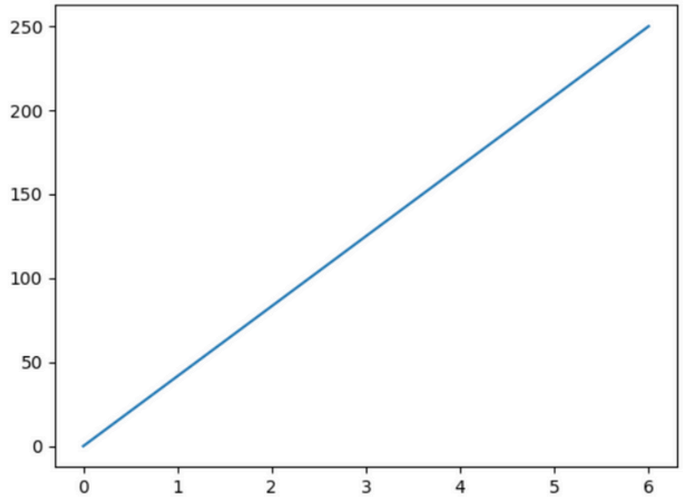
# *PYPLOT*

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

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

Now the Pyplot package can be referred to 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()
```

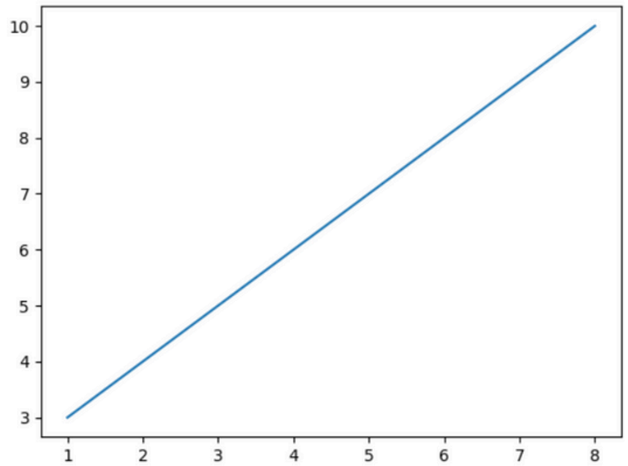


# PLOTTING

## 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()
```

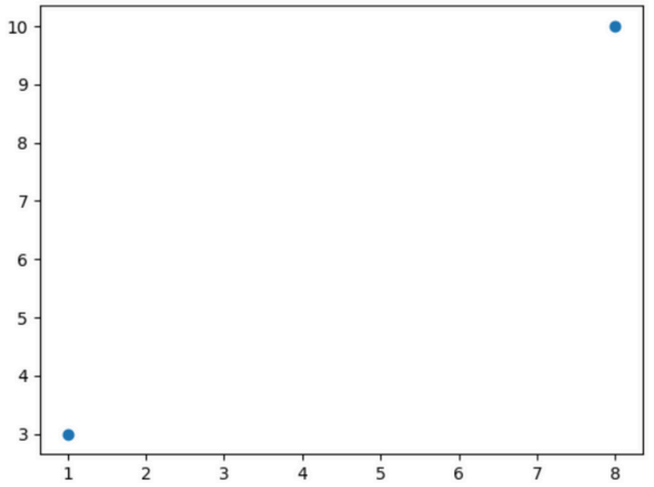


# PLOTTING

## Plotting Without Line

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

```
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()
```

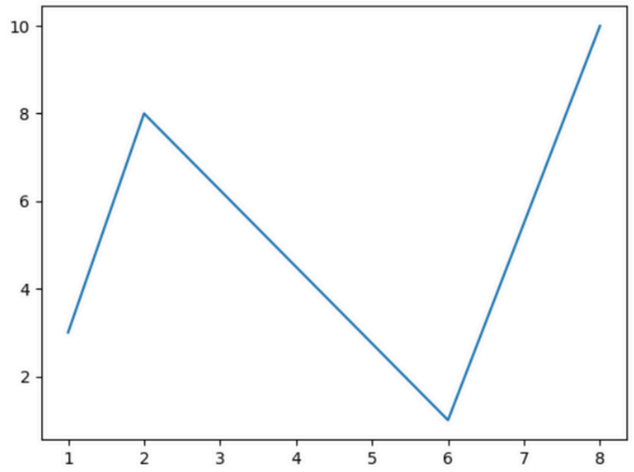


# PLOTTING

## Multiple Points

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

```
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()
```

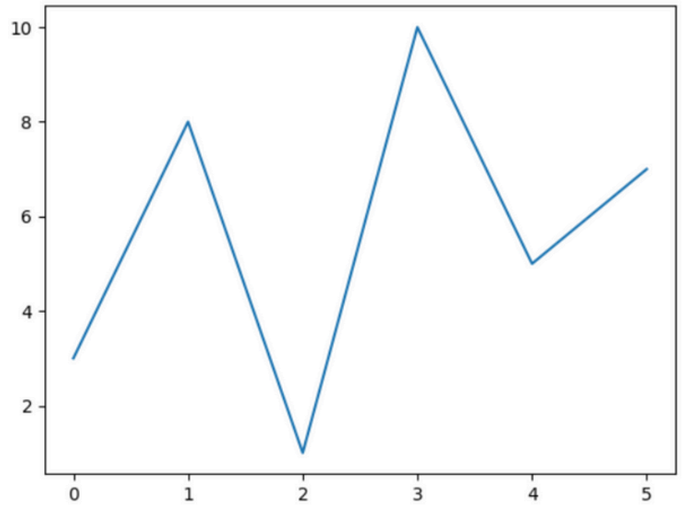


# PLOTTING

## 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.

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

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

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

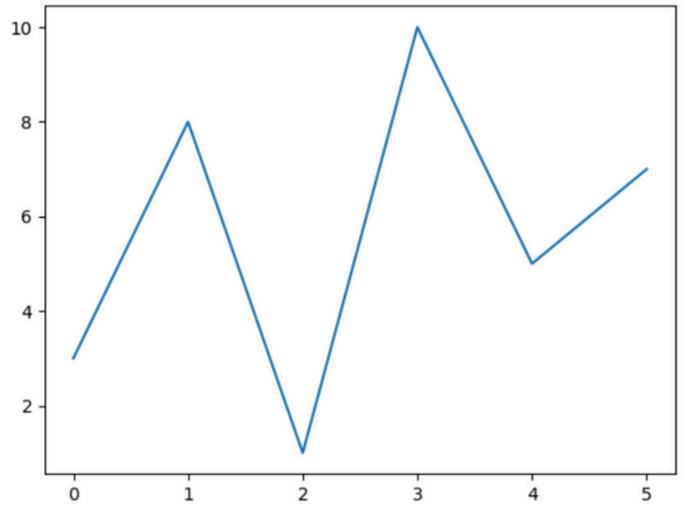


# PLOTTING

## 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.

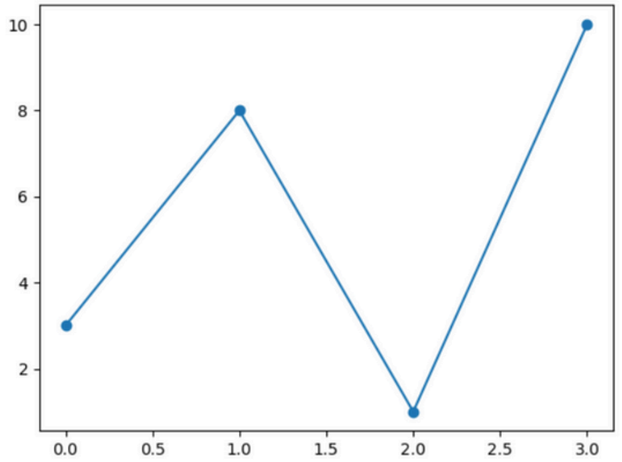
```
import matplotlib.pyplot as plt
import numpy as np

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

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



# *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()
```




## Marker Reference

| Marker | Description |
|---|---|
| 'o' | Circle |
| '*' | Star |
| '.' | Point |

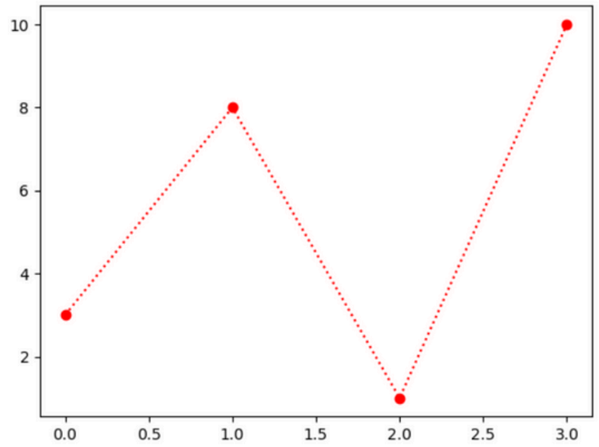
| Marker | Description |
|---|---|
| ',' | Pixel |
| 'x' | X |
| 'X' | X (filled) |

# *FORMAT STRINGS*

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

# *REFERENCES*

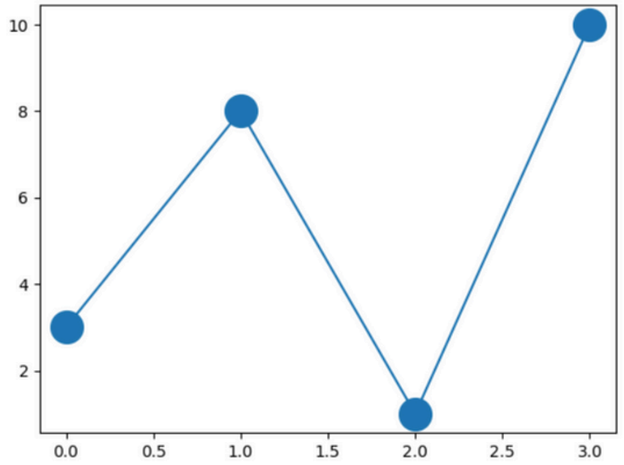
## Color Reference

| 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

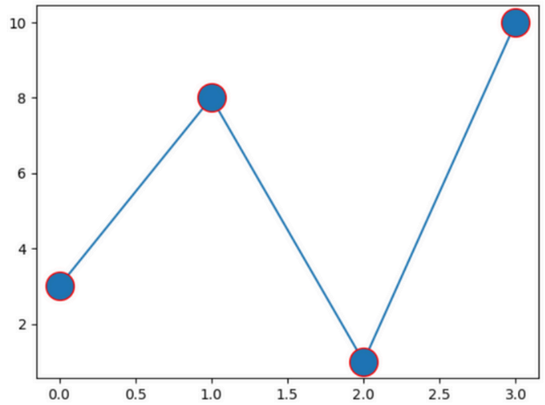
```
import matplotlib.pyplot as plt
import numpy as np

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

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



# *MARKER COLOR*

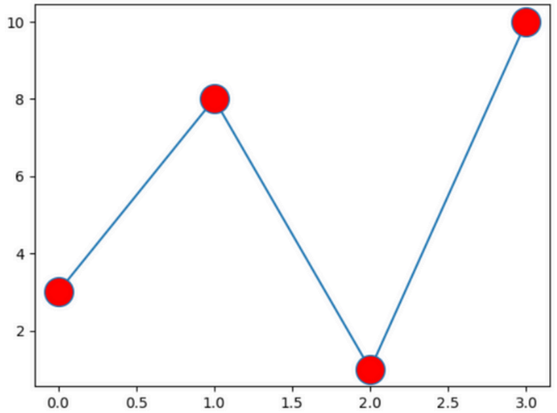
You can use the keyword argument **markeredgecolor** or the shorter **mec** to set the color of the edge of the markers

```
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

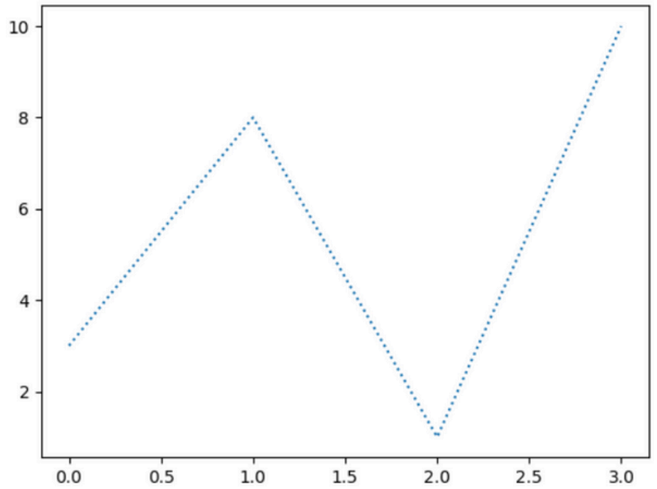
```
import matplotlib.pyplot as plt
import numpy as np

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

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



# *LINES*

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

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

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

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




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

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

# *LINES*

- 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 **- -**

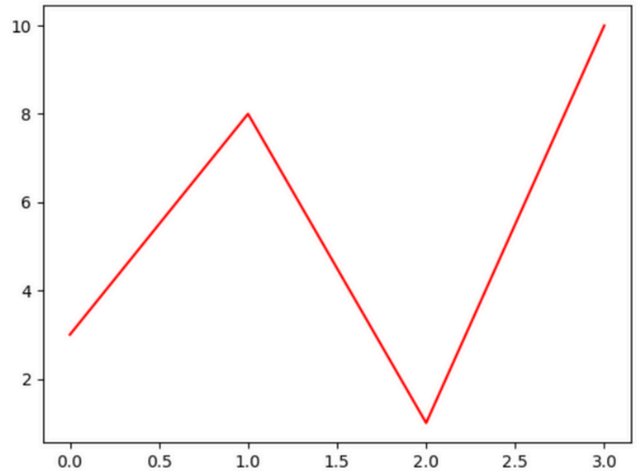
## Line Styles

| 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

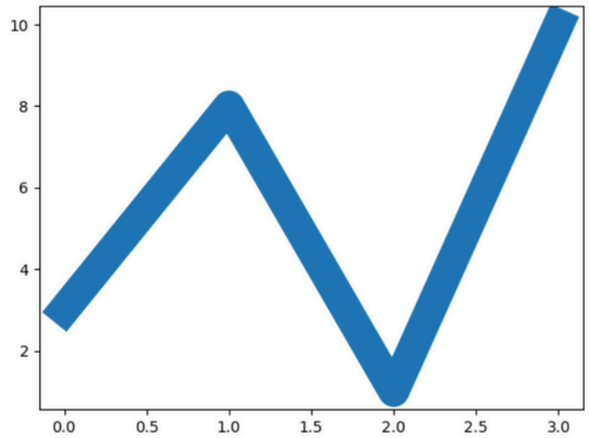
```
import matplotlib.pyplot as plt
import numpy as np

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

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



# *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 :

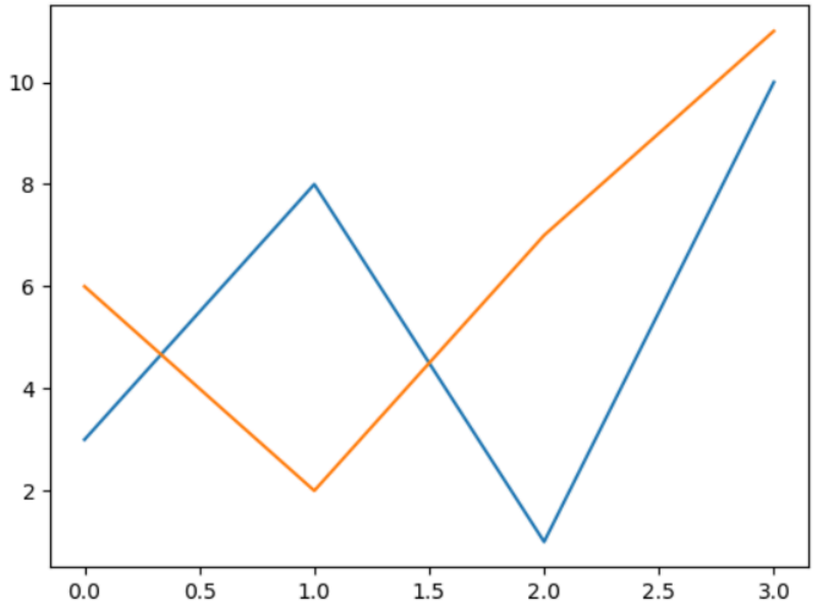
```
import matplotlib.pyplot as plt
import numpy as np

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 :

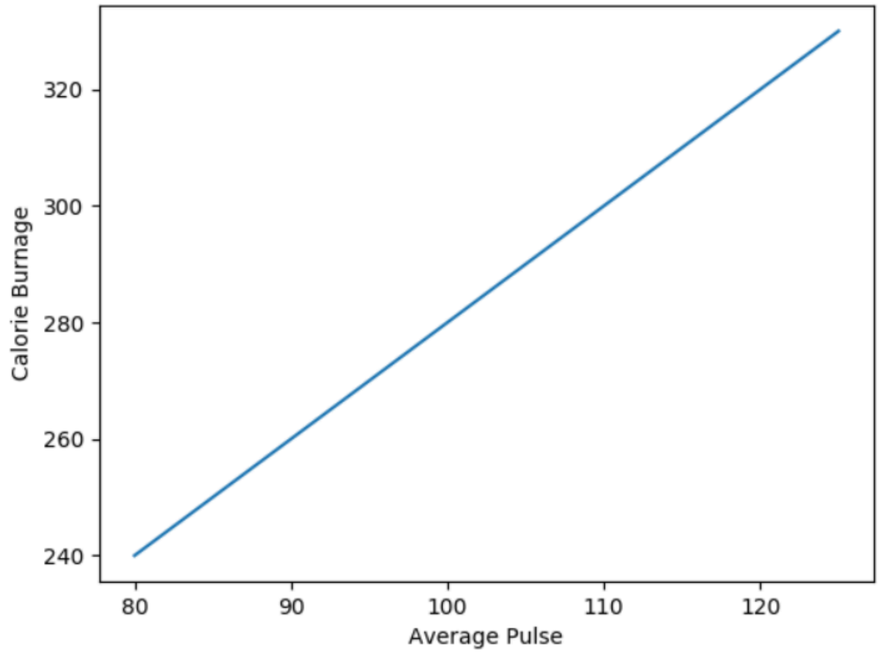
```
import matplotlib.pyplot as plt
import numpy as np

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

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

plt.show()
```



# *LABELS*

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

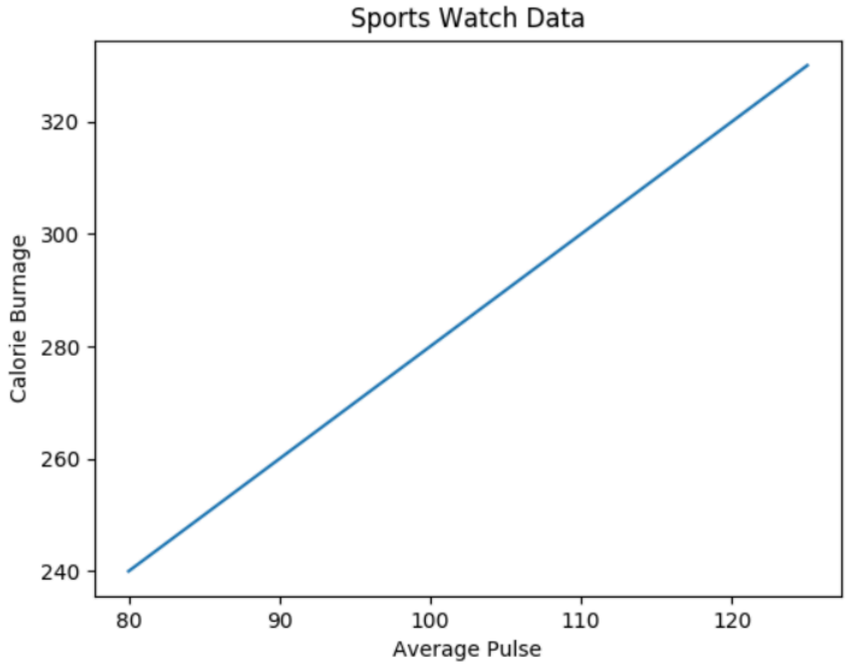
```
import numpy as np
import matplotlib.pyplot as plt

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()
```



# TITLES

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

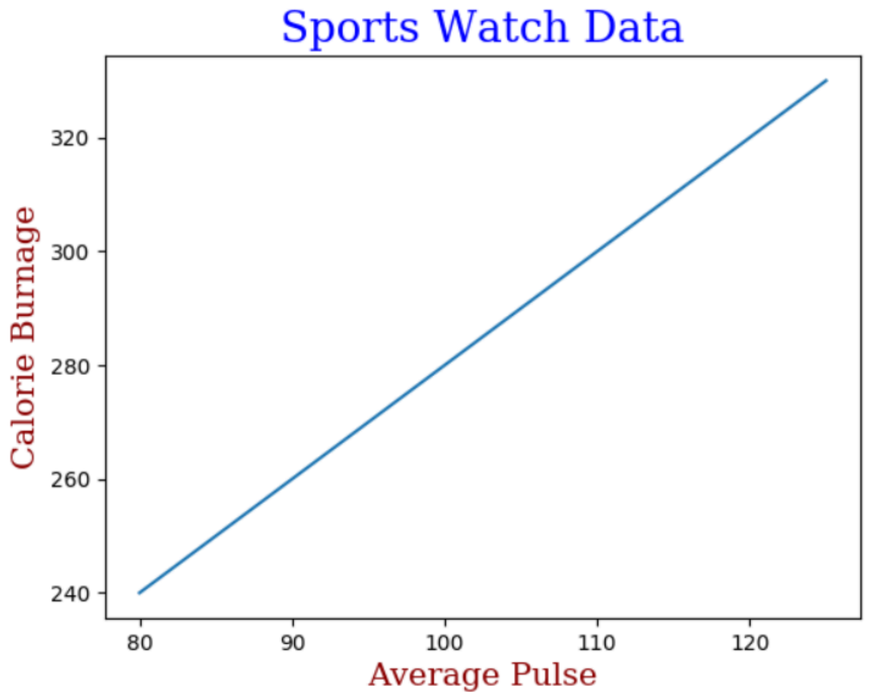
```
import numpy as np
import matplotlib.pyplot as plt

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.plot(x, y)

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

plt.show()
```



# *FONT PROPERTIES*

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

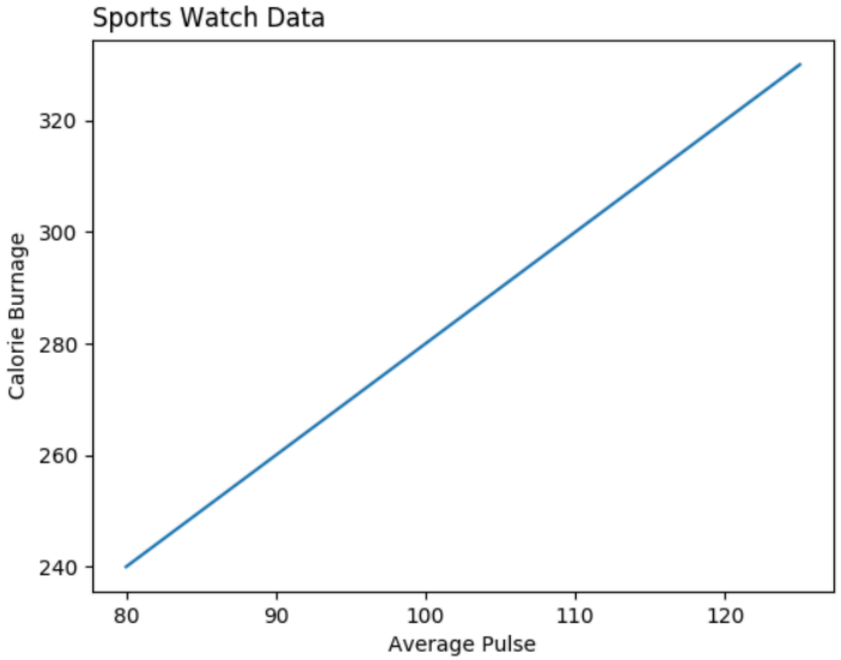
```
import numpy as np
import matplotlib.pyplot as plt

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 TITLE*

- You can use the **loc** parameter in **title( )** to position the title.
- Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

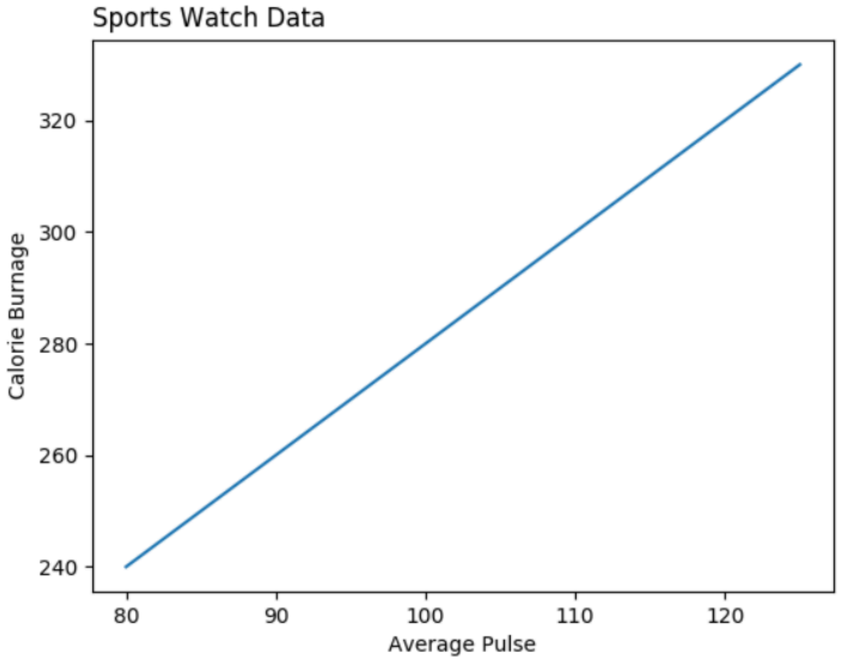
```
import numpy as np
import matplotlib.pyplot as plt

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])

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

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



# *POSITION 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])

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

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

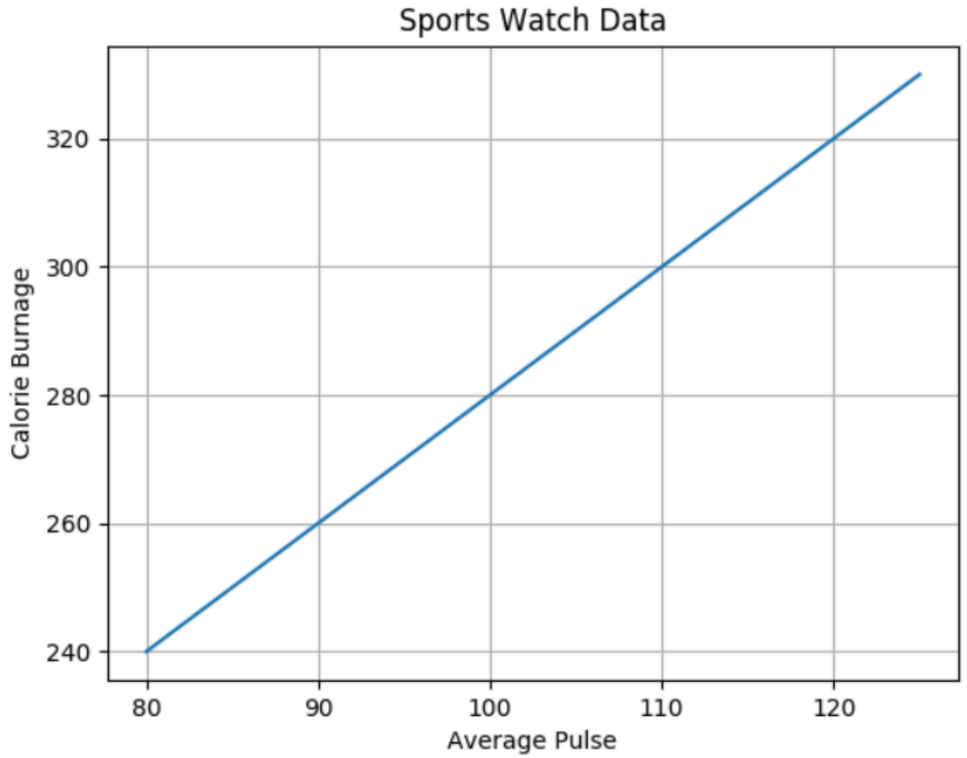


# ADDING GRID LINES

## Add Grid Lines to a Plot

With Pyplot, you can use the **grid( )** function to 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()
```

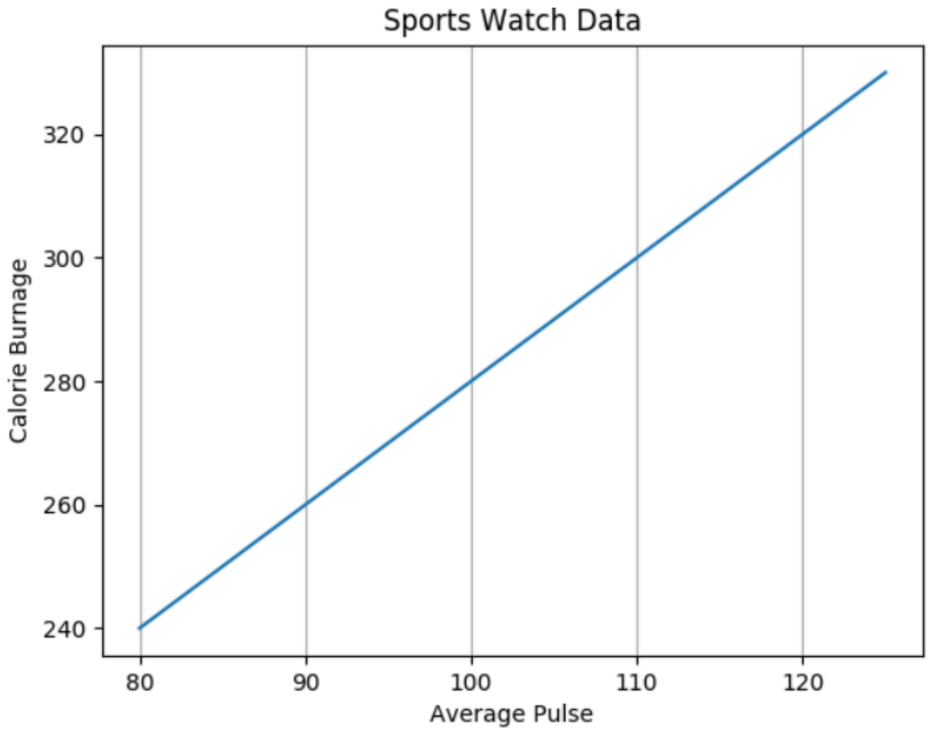


# ADDING GRID LINES

## 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()
```

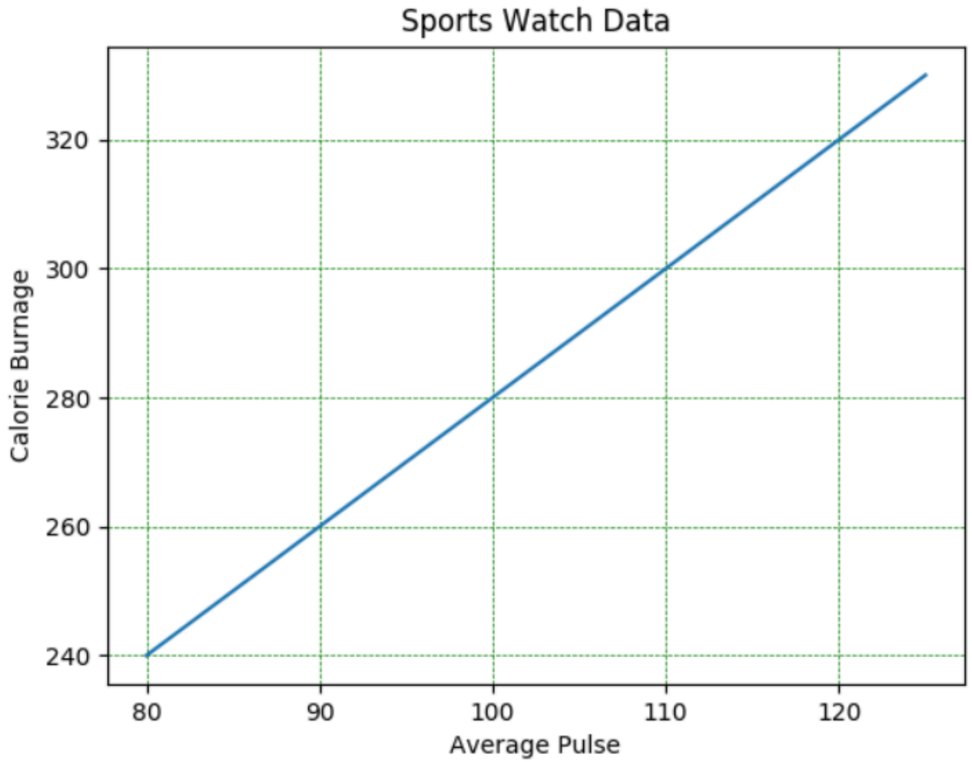


# ADDING GRID LINES

## 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(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```

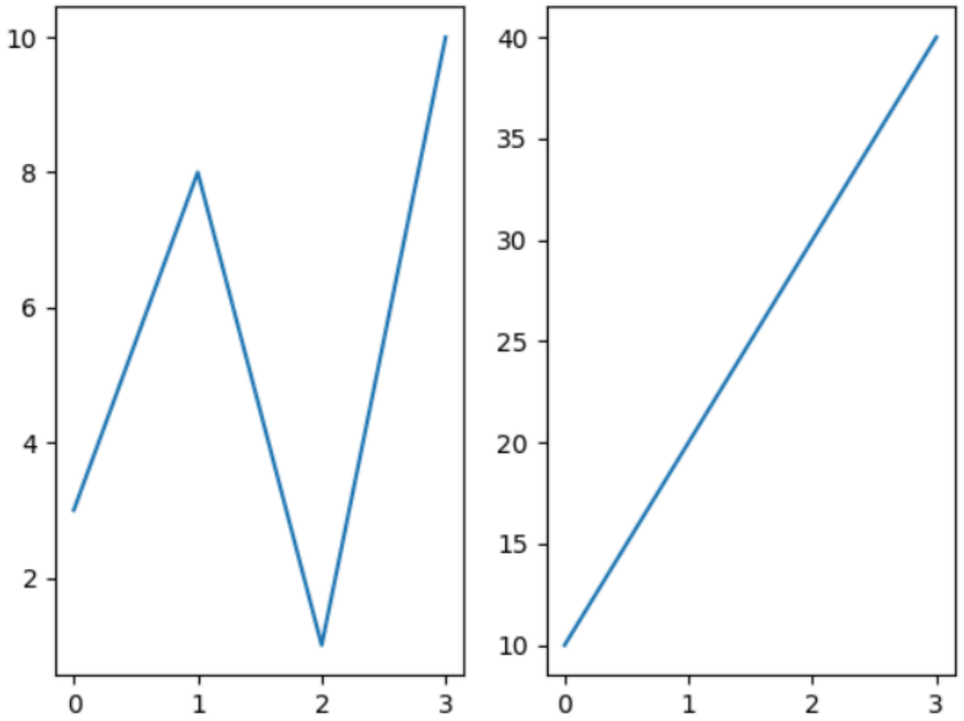


# *SUBPLOT*

## Display Multiple Plots

With the **subplot( )** function you can draw multiple plots in one 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)

#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()
```



# *SUBPLOT*

## 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.
```

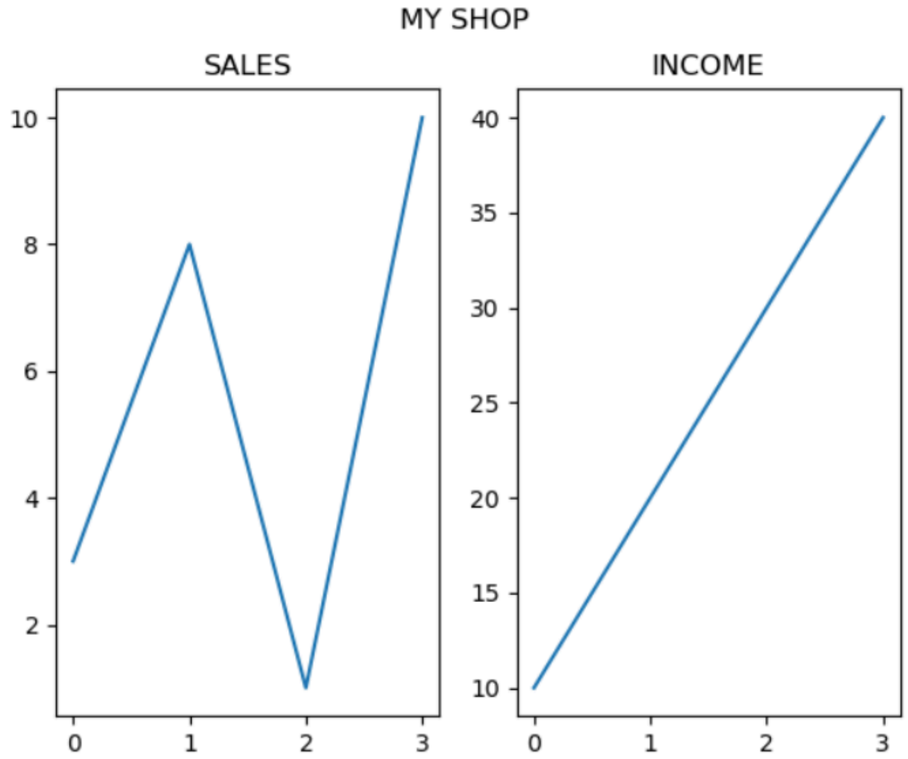
**Note:** You can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.

# SUBPLOT

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

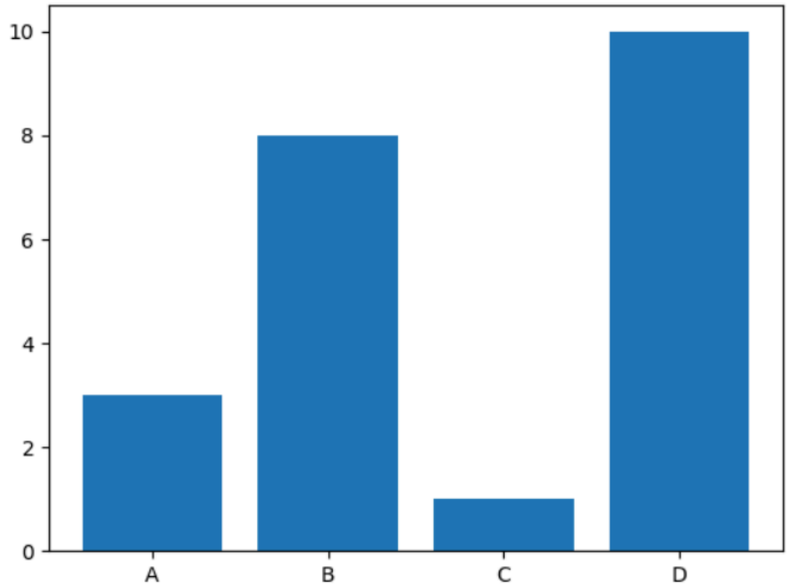


# BARS

## Creating Bars

With Pyplot, you can use the **bar( )** function to draw bar graphs

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



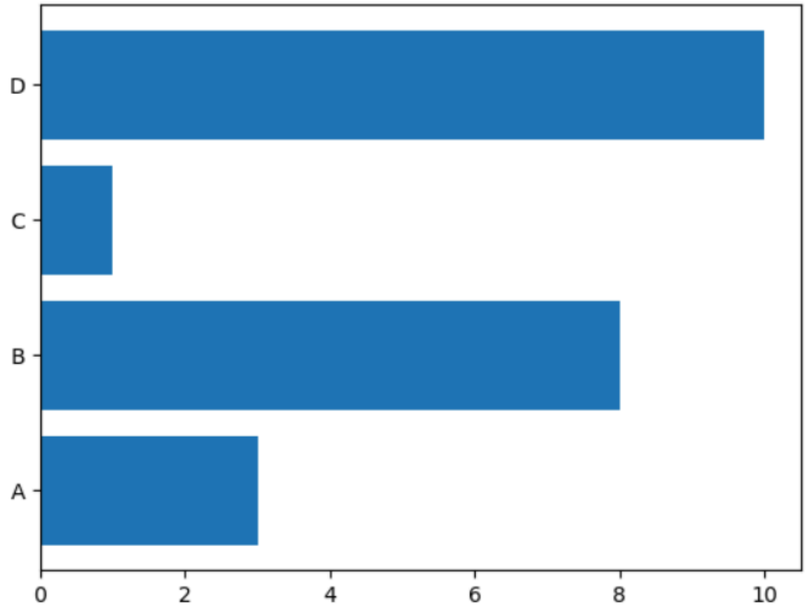

The **bar( )** function takes arguments that describes the layout of the bars.

# BARS

## Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the **barh( )** function

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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

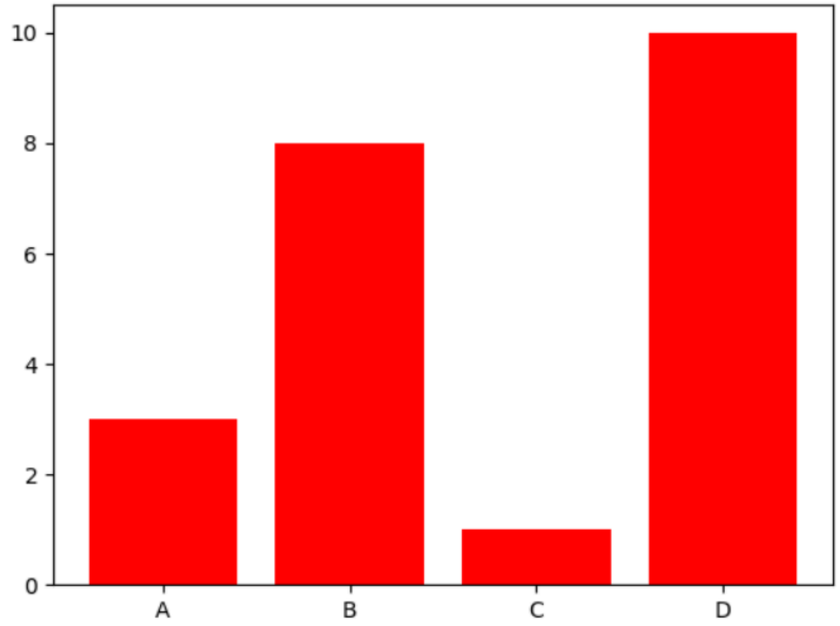


# BARS

## Bar Color

The **bar( )** and **barh( )** take the keyword argument **color** to set the color of the bars

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

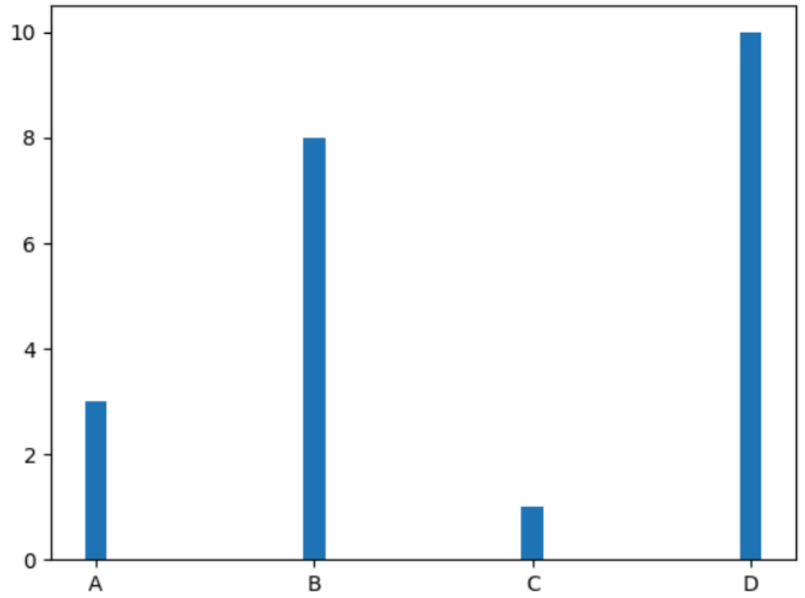


# BARS

## Bar Width

The **bar( )** takes the keyword argument width to set the **width** of the bars

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```



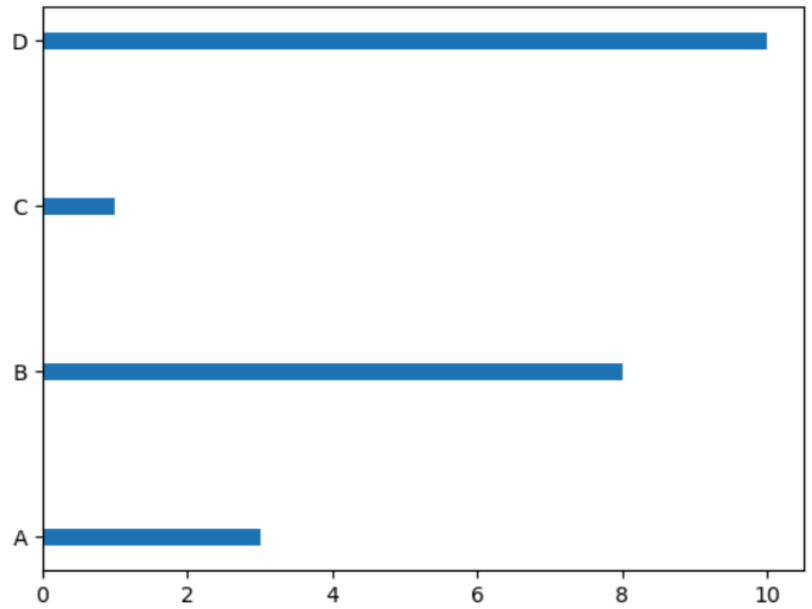

**Note:** For horizontal bars, use **height** instead of **width**

# BARS

## Bar Height

The **barh( )** takes the keyword argument height to set the height of the bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```



# PIE CHARTS

## Creating Pie Charts

With Pyplot, you can use the **pie()** function to draw pie charts

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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

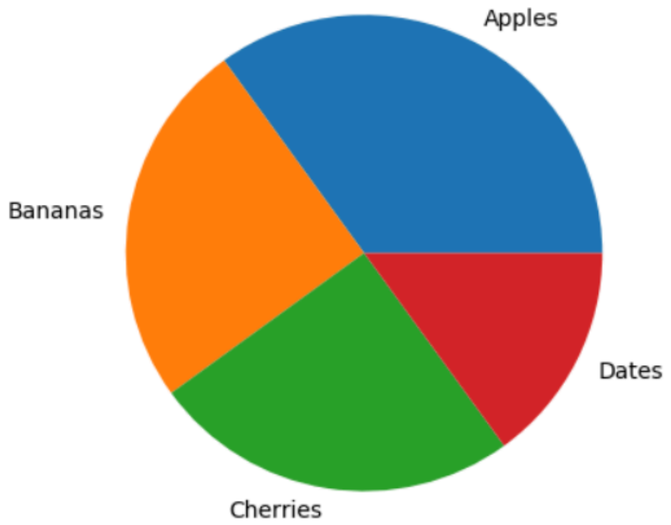
**Note:** The size of each wedge is determined by comparing the value with all the other values, by using this formula: The value divided by the sum of all values: **x/sum(x)**

# PIE CHARTS

## Labels

- Add labels to the pie chart with the **labels** parameter.
- The labels parameter must be an **array** with one label for each wedge

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

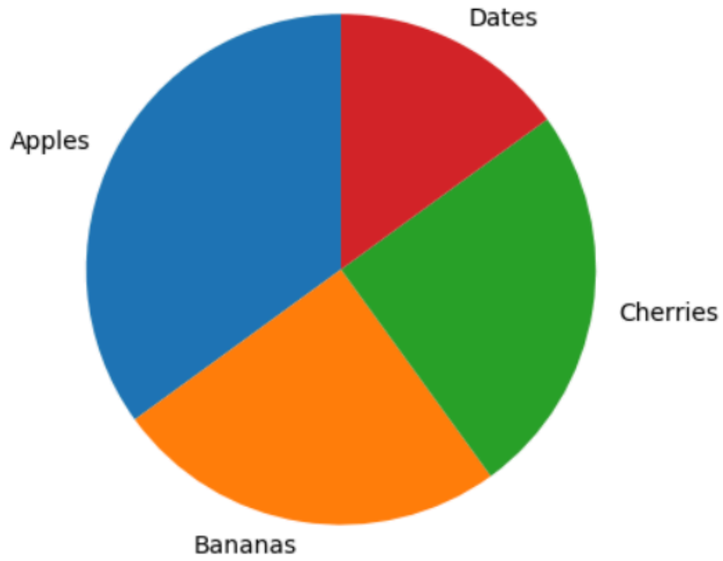


# PIE CHARTS

## Start Angle

- As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a **startangle** parameter.
- The **startangle** parameter is defined with an angle in degrees, default angle is 0

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```

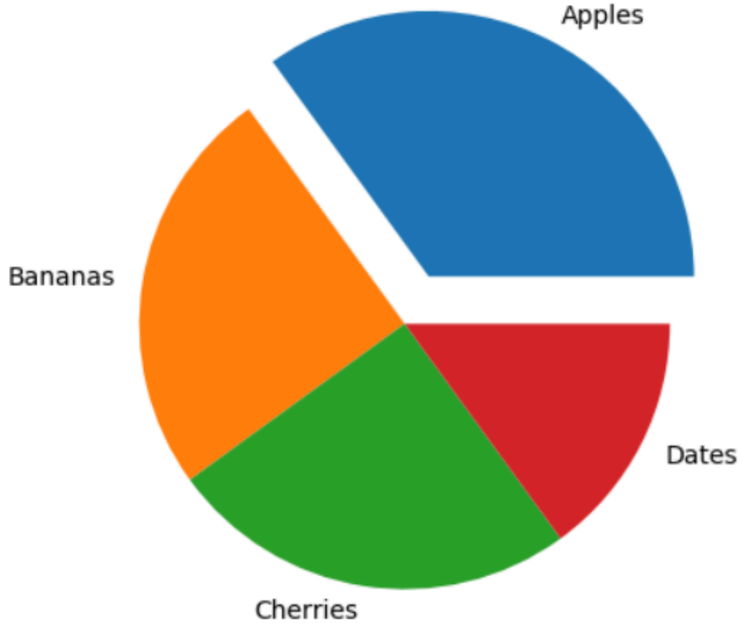


# PIE CHARTS

## Explode

- 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.
- Each value represents how far from the center each wedge is displayed

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```



# PIE CHARTS

## Colors

- You can set the color of each wedge with the **colors** parameter.
- The **colors** parameter, if specified, must be an array with one value for each wedge

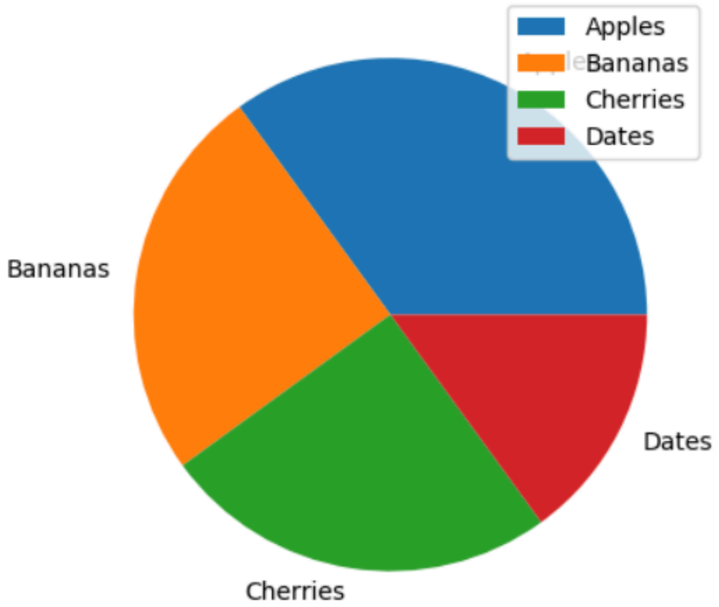
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

# PIE CHARTS

## Legend

To add a list of explanation for each wedge, use the **legend()** function

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```

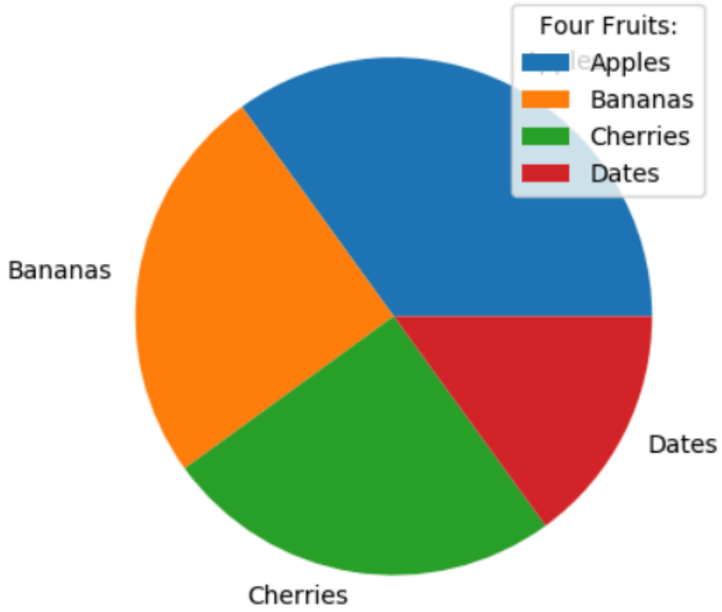


# PIE CHARTS

## Legend With Header

To add a header to the legend, add the **title** parameter to the **legend** function

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```



# TASKS

Choose 2 of Beginner or 1 from either Intermediate or Advanced

## Beginner Level

- **Dual-Line Plot with Markers and Labels:** Plot the revenue and profit for a company over six months on the same graph. Use different markers for each line and ensure the plot has labeled axes, a title, and a legend. Ensure the markers are distinguishable and the lines have different styles.

- **Vertical Bar Chart with Annotations:** Create a vertical bar chart showing the number of books read by students in five different grades. Annotate the top of each bar with the exact number of books read. Ensure the bars are colored differently and labeled.

- **Customized Pie Chart:** Create a pie chart showing the market share of five smartphone brands. Ensure the slices have different colors, labels with percentages, and explode the slice with the highest market share. Also, rotate the chart to start from a different angle.

# TASKS

Choose 2 of Beginner or 1 from either Intermediate or Advanced

## Intermediate Level

- **Subplots with Shared Y-axis:** Create two subplots in a single figure where both plots share the same y-axis. The first plot should show the number of hours studied over a week, and the second should display the number of hours of sleep. Customize the plots with different line styles, markers, and colors. Add a shared y-axis label and individual x-axis labels.

- **Stacked Bar Chart with Labels**: Create a stacked bar chart showing the performance of three teams in four different tasks. Each team's performance in each task should be stacked on top of the others. Add labels inside the bars to indicate the performance numbers, and customize the colors.

- **Complex Pie Chart with Nested Categories:** Create a pie chart showing the distribution of expenses for a family in various categories (e.g., Housing, Food, Transportation). Add an inner ring to show a breakdown of each major category (e.g., Food might include Groceries, Dining Out). Customize with labels, colors, and an exploded slice.

# TASKS

Choose 2 of Beginner or 1 from either Intermediate or Advanced

## Advanced Level

- **Overlapping Line Plots with Annotations:** Plot three trigonometric functions (sine, cosine, and tangent) on the same graph over the range of 0 to 2π. Ensure each line has a different style and color. Add annotations for key points where the functions intersect or reach significant values (e.g., sine and cosine crossing at π/4). Include a legend and grid.

- **Advanced Subplots with Different Scales:** Create a figure with three subplots: the first subplot shows a linear function, the second shows an exponential function, and the third shows a logarithmic function. Use different y-axis scales (linear, log, and symlog) for each subplot. Add a grid to all subplots and customize the lines and markers.

- **Complex Bar Chart with Custom Legends and Colors:** Create a horizontal grouped bar chart showing the test scores of four different students in five subjects. Each group of bars should represent one subject with bars colored differently for each student. Add a custom legend, and ensure the bars are labeled with their respective scores and ordered by the highest to lowest scores for each subject.

# *ANY QUESTIONS ?*

DATA ANALYSIS PROGRAM

# *THANK YOU*

UPCOMING NEXT WEEK : SESSION (3)