

# Line Plot and Bar Graph by Mrittika Megaraj

```
In [1]: import numpy as np
import pandas as pd
```

## 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 of Matplotlib

- First, we should ensure that, **Python** and **PIP** are installed in the system, then can install matplotlib using the following pip command:

```
pip install matplotlib
```

### Import Matplotlib:

- After successful installation of **matplotlib**, we can import it using the following import module statement:

```
import matplotlib
```

```
In [2]: import matplotlib
```

### Checking Matplotlib Version

- The version string is stored under **\_\_version\_\_** attribute.

```
In [3]: import matplotlib

print(matplotlib.__version__)
```

3.7.1

# Matplotlib Pyplot

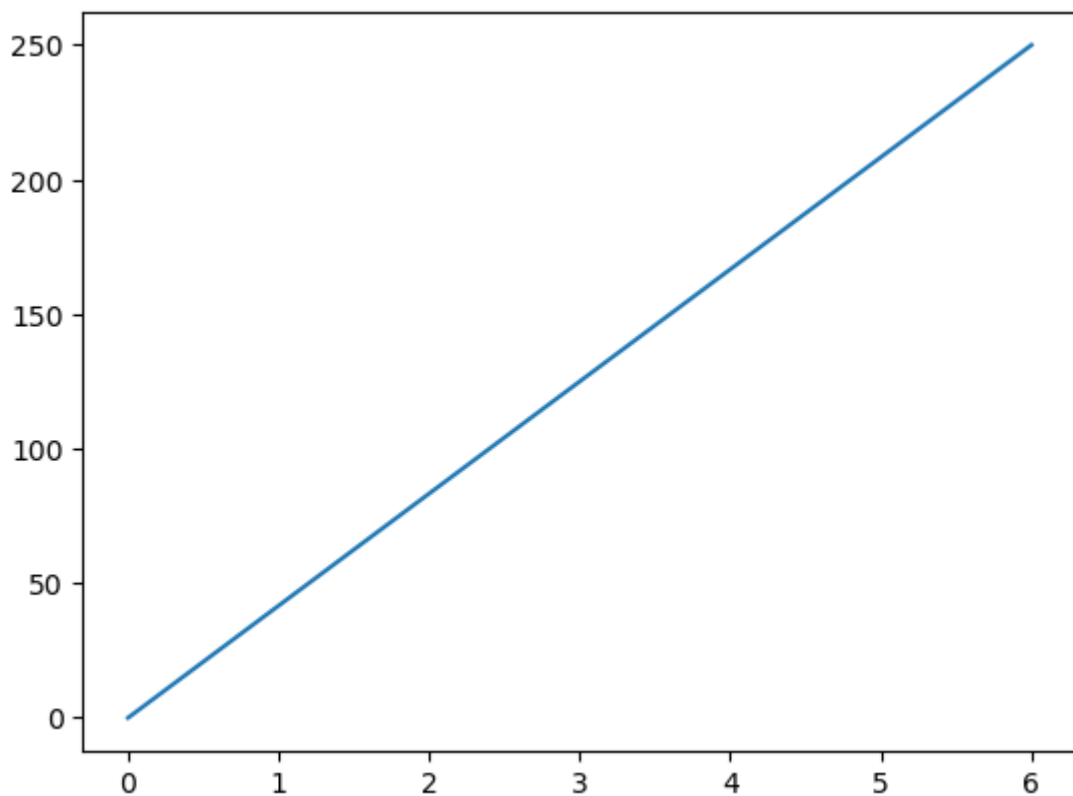
---

## Pyplot

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

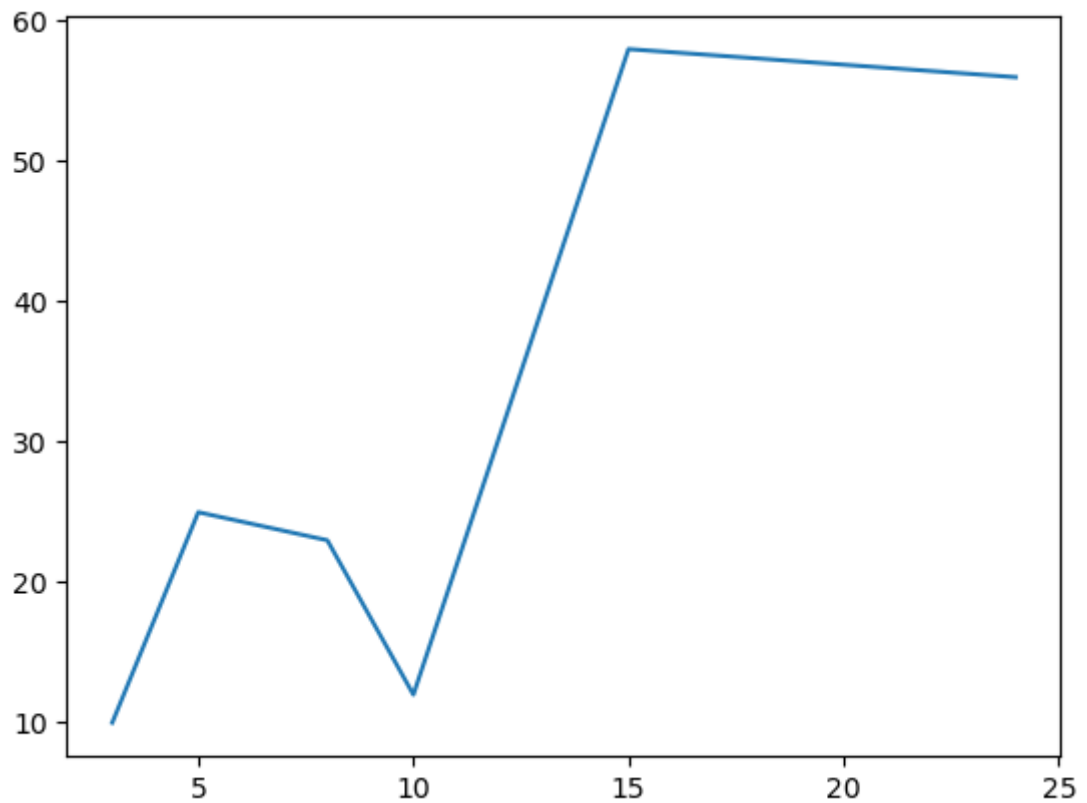
```
In [4]: import matplotlib.pyplot as plt
```

```
In [5]: # Draw a line in a diagram from position (0,0) to position (6,250):  
  
# From these points, can draw straight line  
xpoints = np.array([0, 6])  
ypoints = np.array([0, 250])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```



In [6]: *# plotting points, which are not in a straight line*

```
x_pts = np.array([3, 5, 8, 10, 15, 24])  
y_pts = np.array([10, 25, 23, 12, 58, 56])  
  
plt.plot(x_pts, y_pts)  
plt.show()
```

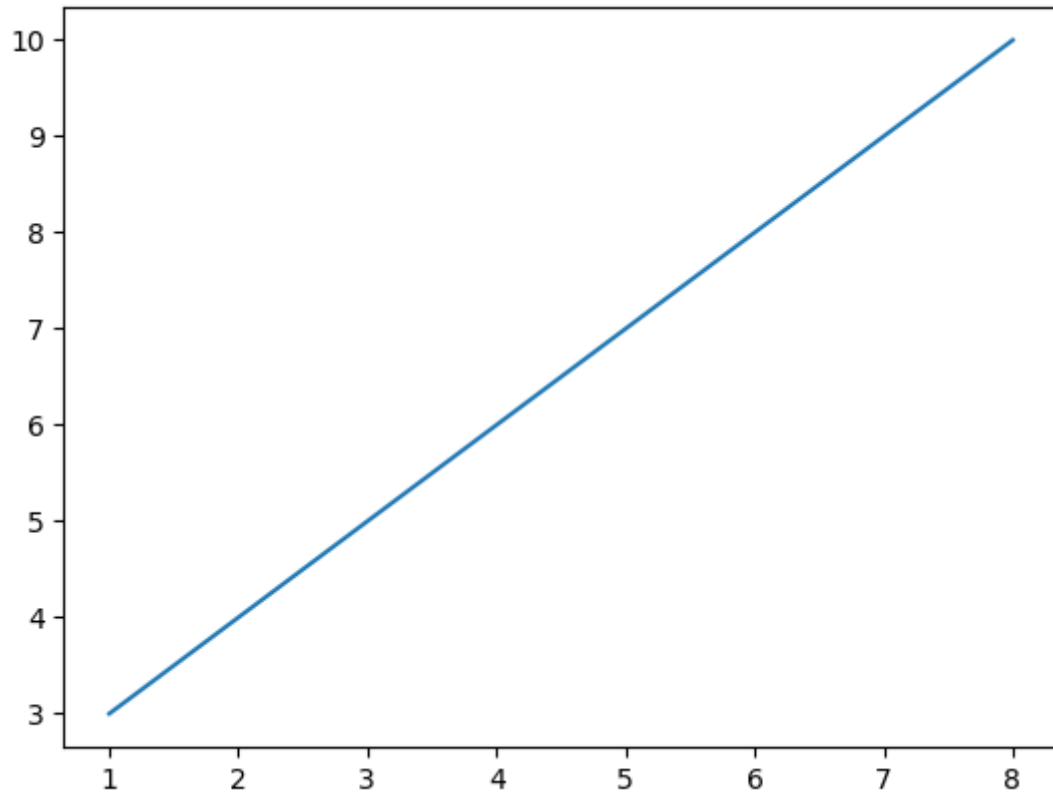


#### Plotting x and y points:

- 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.

In [7]: *# Draw straight line between 2 points (1,3) and (8,10)*

```
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```

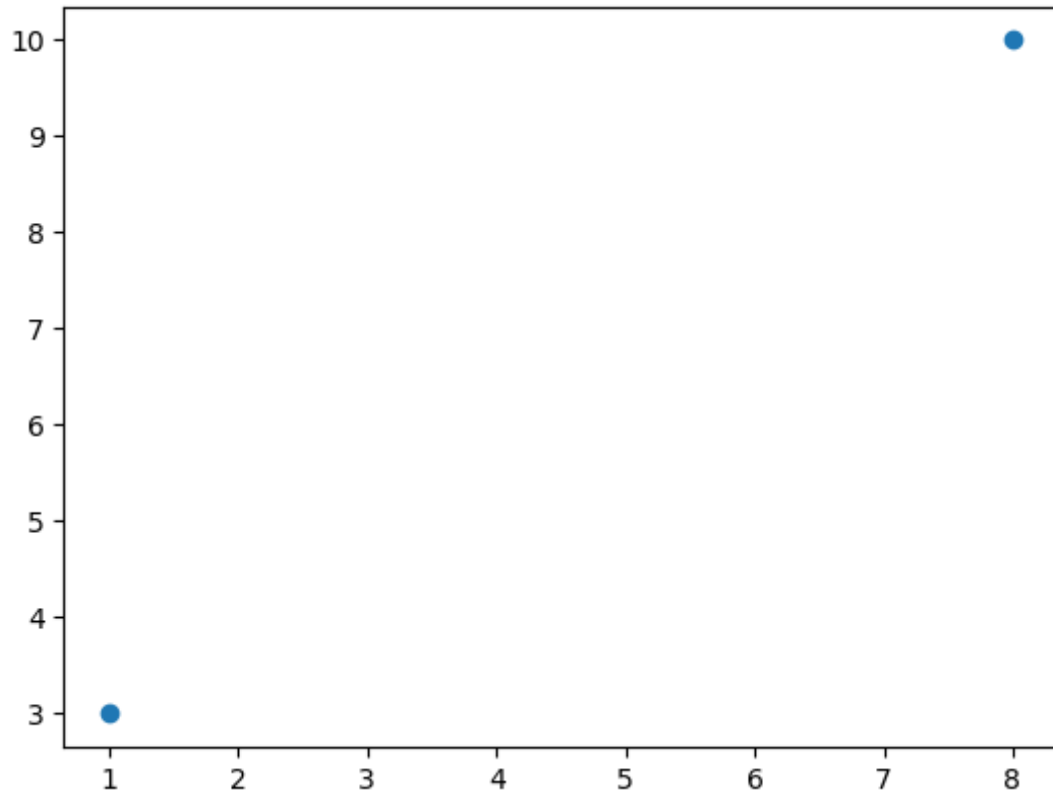


### Plotting Without Line:

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

```
In [8]: # Just mark 2 points at the places: (1,3) and (8,10)
```

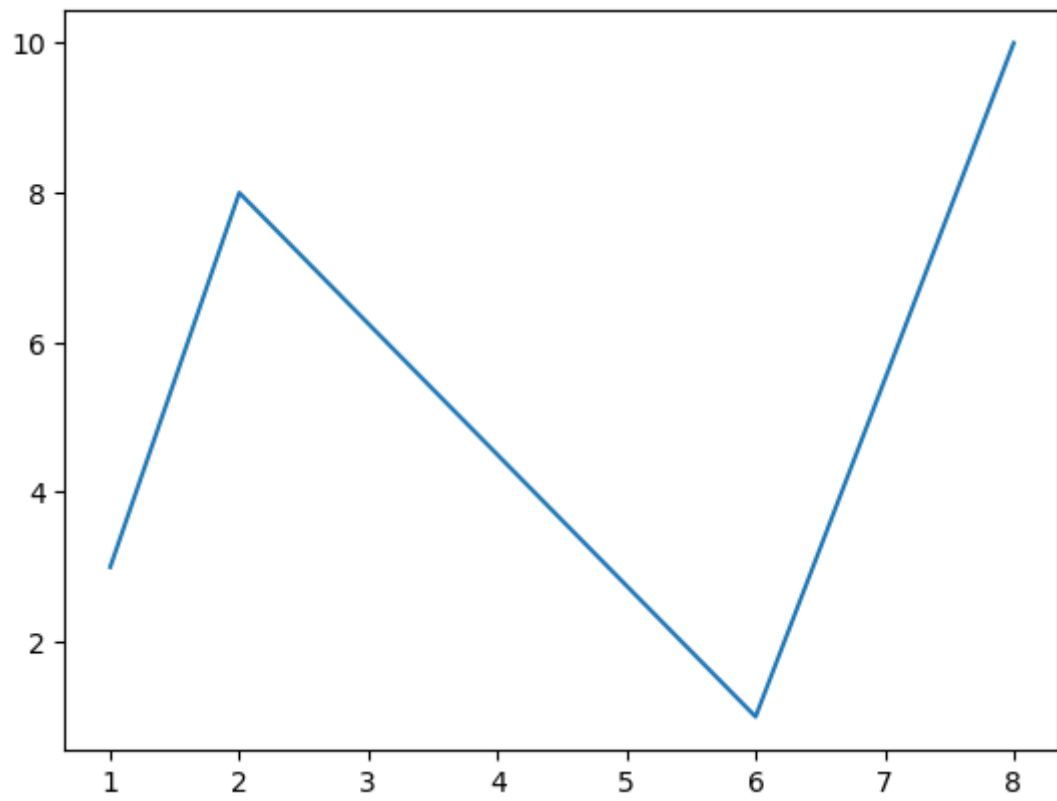
```
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints, 'o')  
plt.show()
```



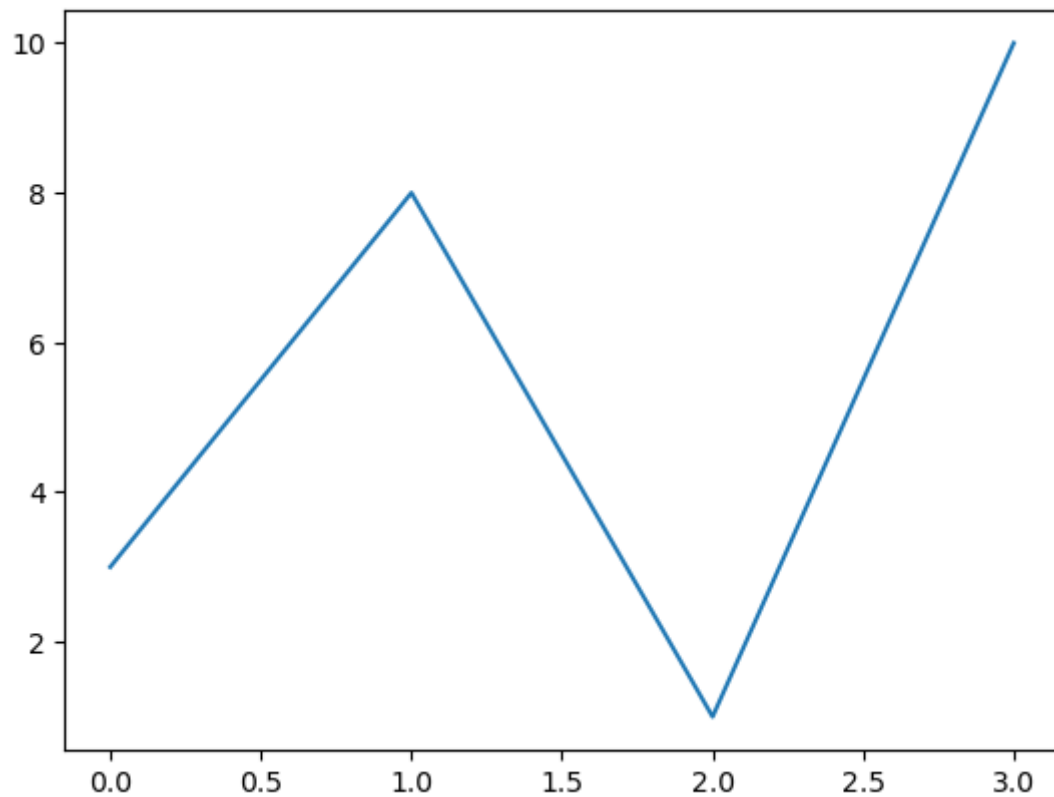
### Multiple Points:

- We can plot any number of points on the graph, just have to be esure, that both arrays have same number of points.

```
In [9]: xpoints = np.array([1, 2, 6, 8])  
        ypoints = np.array([3, 8, 1, 10])  
  
        plt.plot(xpoints, ypoints)  
        plt.show()
```



```
In [10]: ypoints = np.array([3, 8, 1, 10])  
  
# If we give only one arg, then considered as y_pts, and  
# x-axis contains nums from 0 till length of the y_pts array  
plt.plot(ypoints)  
plt.show()
```

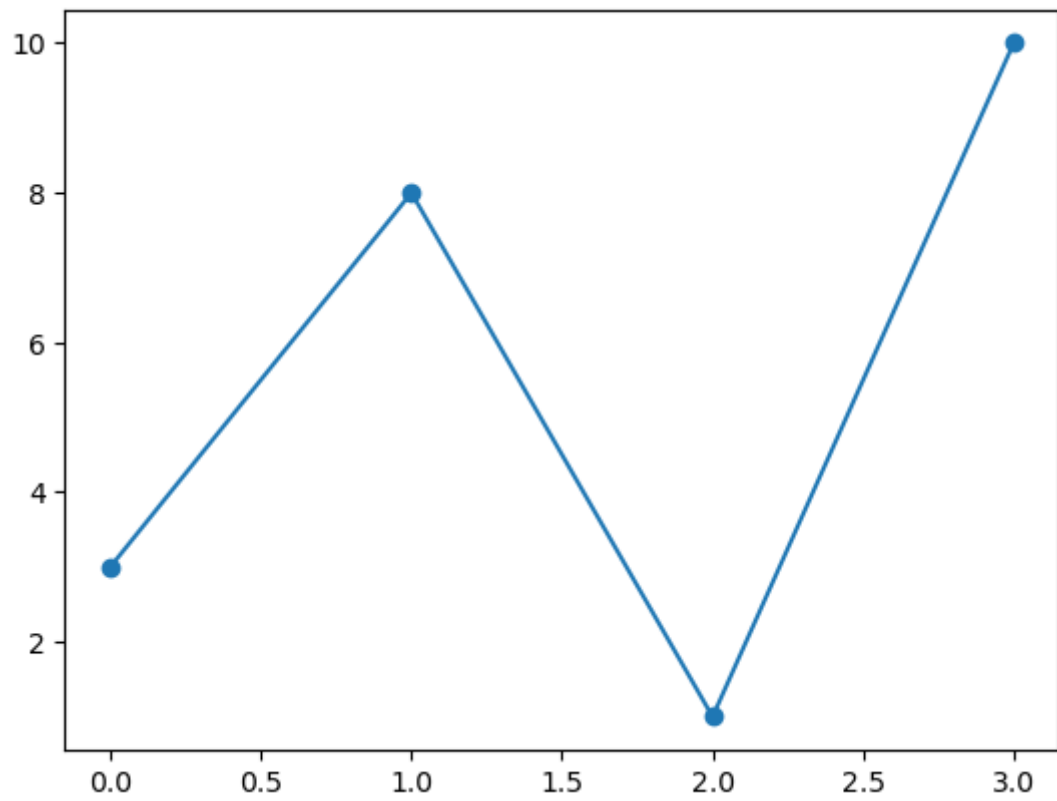


## Matplotlib Markers

---

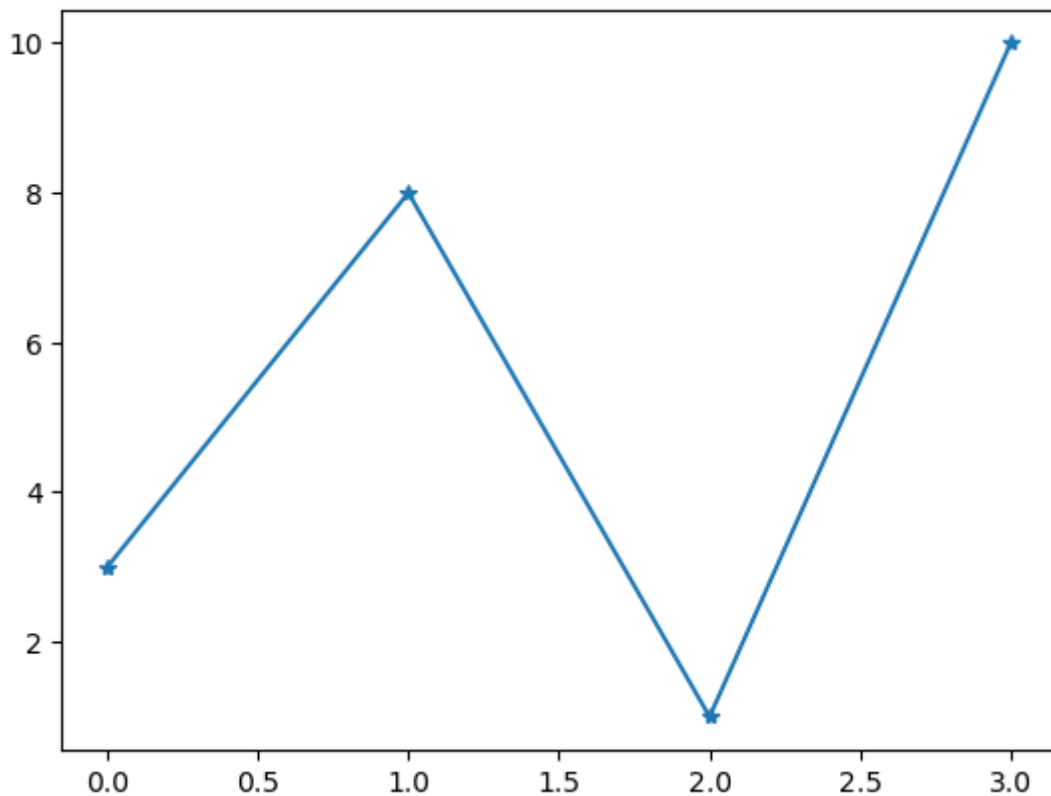
- We can use the keyword argument **marker** to emphasize each point with a specified marker.

```
In [11]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o')  
plt.show()
```





```
In [12]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = '*')  
plt.show()
```



### Marker Types:

- '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

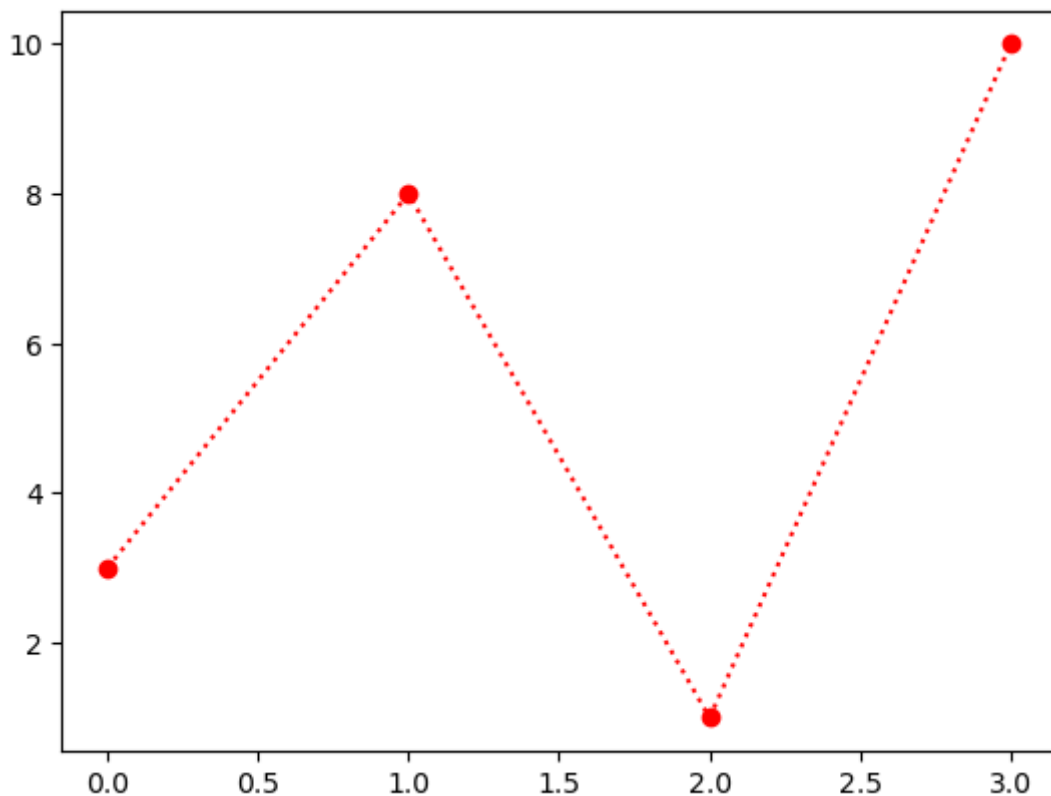
- `'_'` Hline

### Format Strings `fmt`:

- We can also use the **shortcut string notation** parameter to specify the marker.
- This parameter is also called **`fmt`**, and is written with the following syntax:

`marker|line|color`

```
In [13]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, 'o:r')  
plt.show()
```



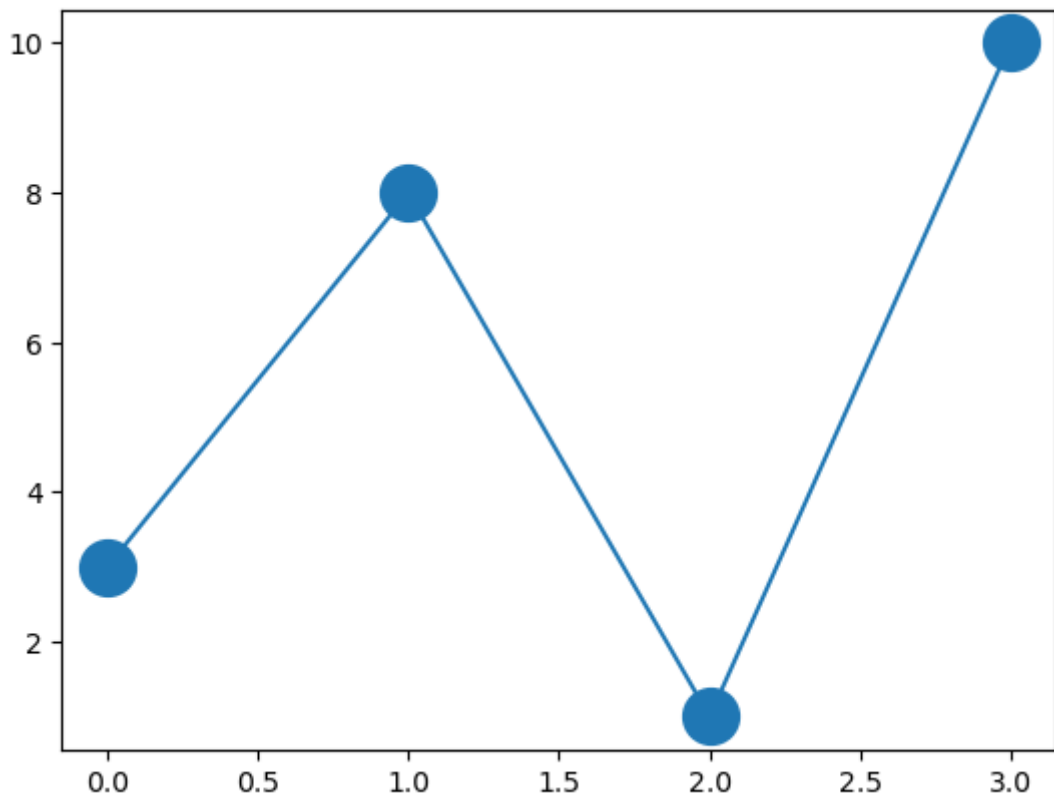
### Line Values:

- `'-'` Solid line
- `'.'` Dotted line
- `'--'` Dashed line
- `'-.'` Dashed/dotted line

### Marker Size:

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

```
In [14]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o', ms = 20)  
plt.show()
```

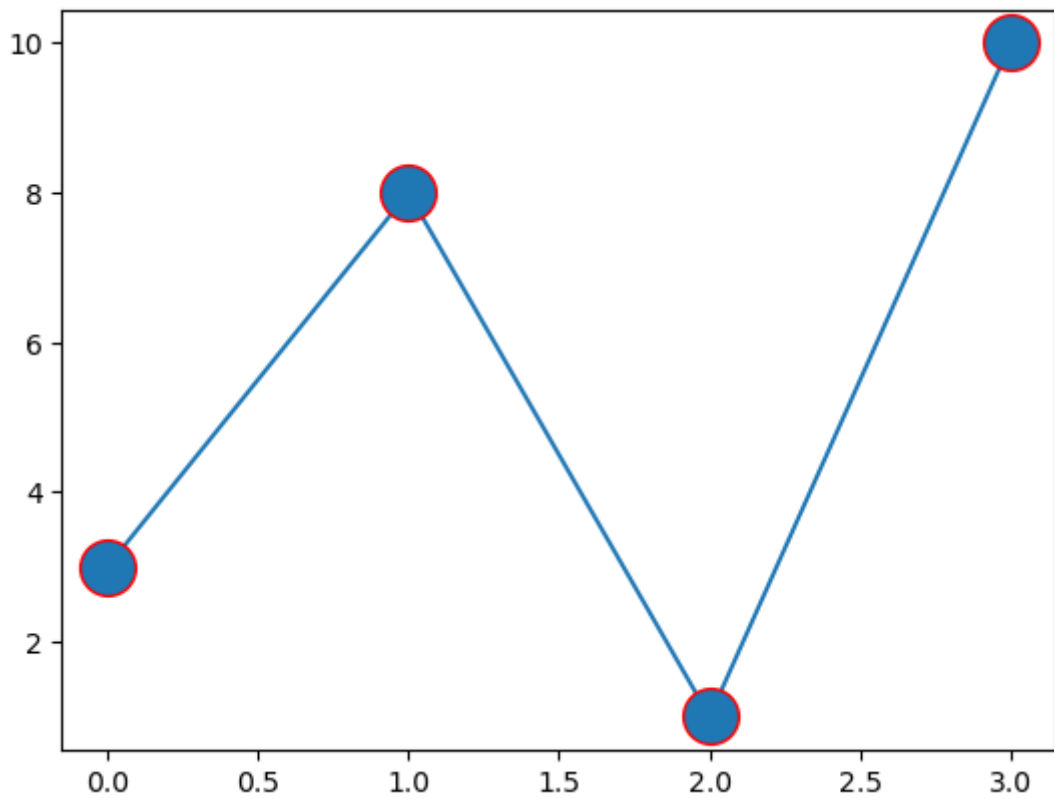


#### Marker Color:

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

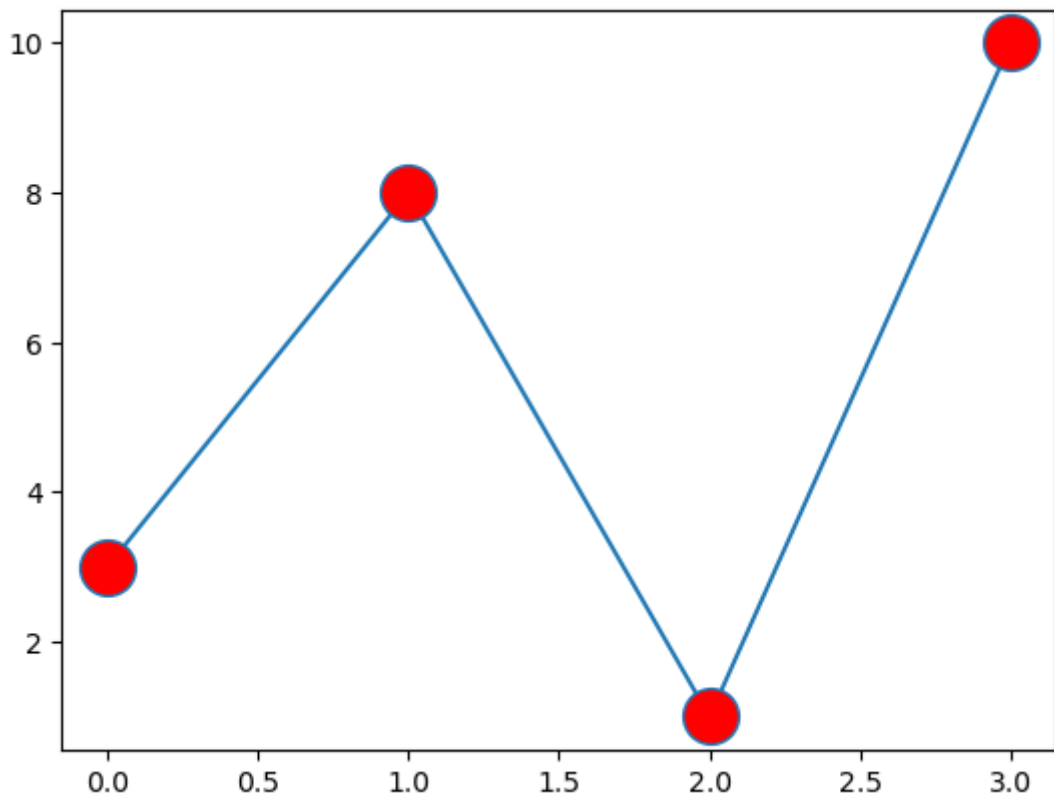
```
In [15]: ypoints = np.array([3, 8, 1, 10])

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



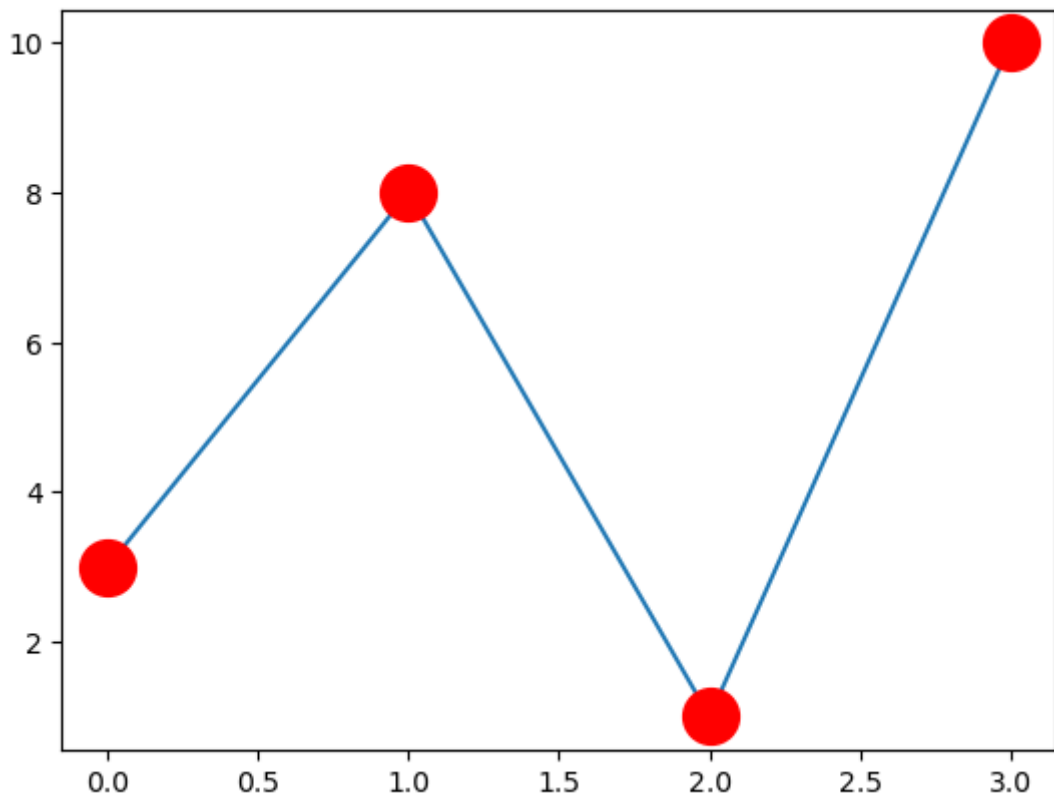
- We can use the keyword argument **markerfacecolor** or the shorter **mfc** to set the color inside the edge of the markers.

```
In [16]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')  
plt.show()
```



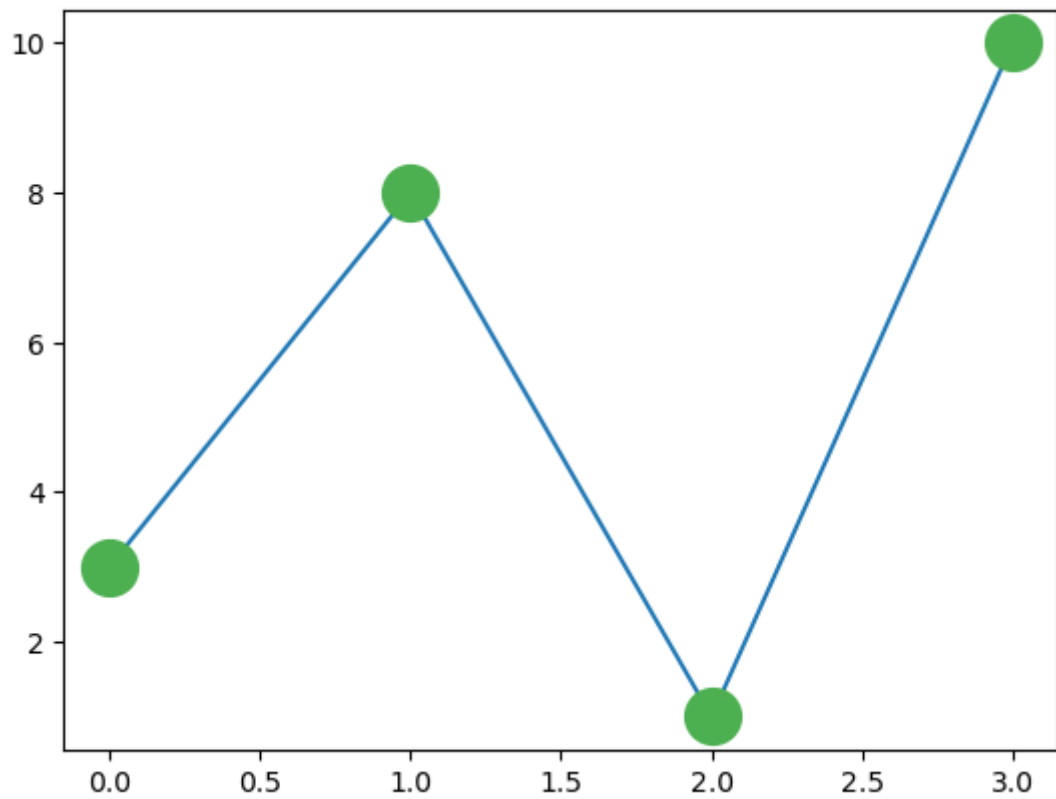
- Use both the **mec** and **mfc** arguments to color the entire marker.

```
In [17]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')  
plt.show()
```



- Use of Hexadecimal color values:

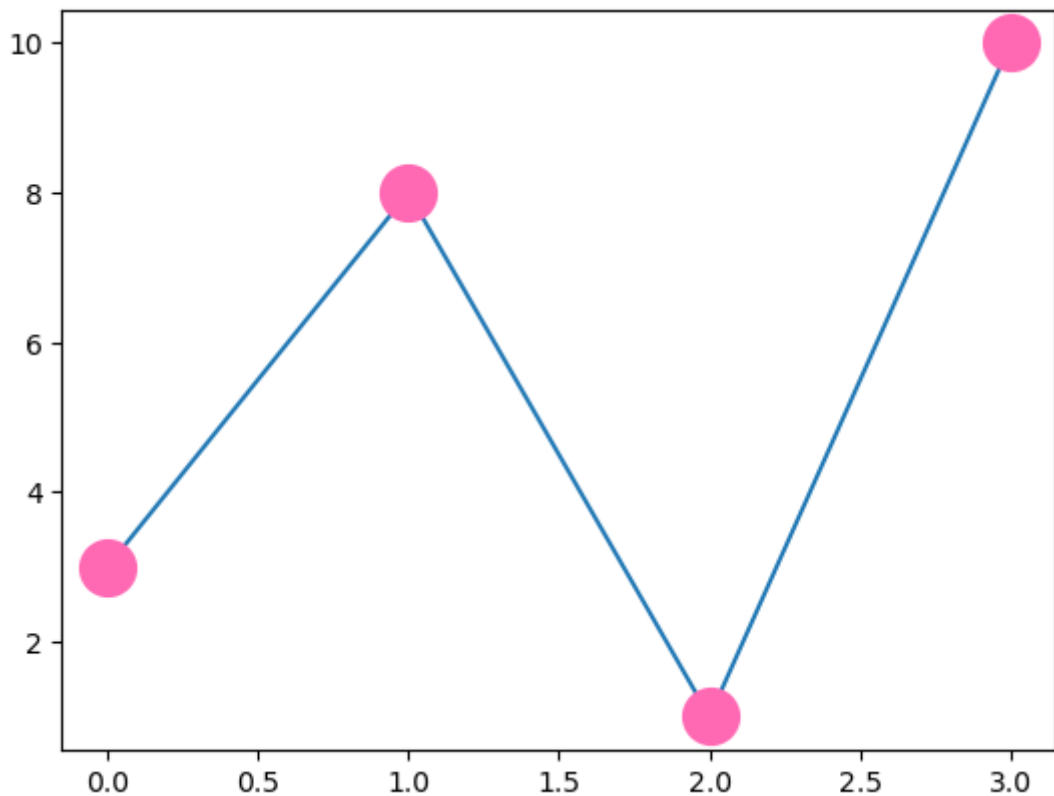
```
In [18]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')  
plt.show()
```



- Use of supported color names:

```
In [19]: ypoints = np.array([3, 8, 1, 10])

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



## Matplotlib Line

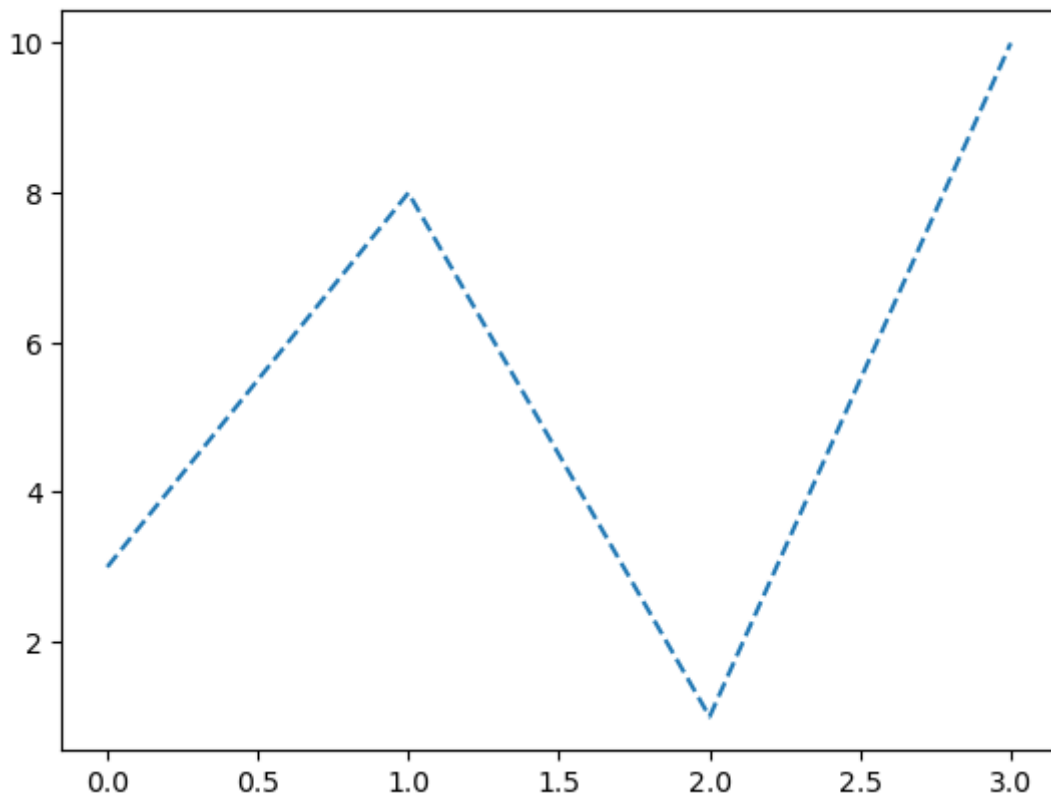
---

### Linestyle:

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



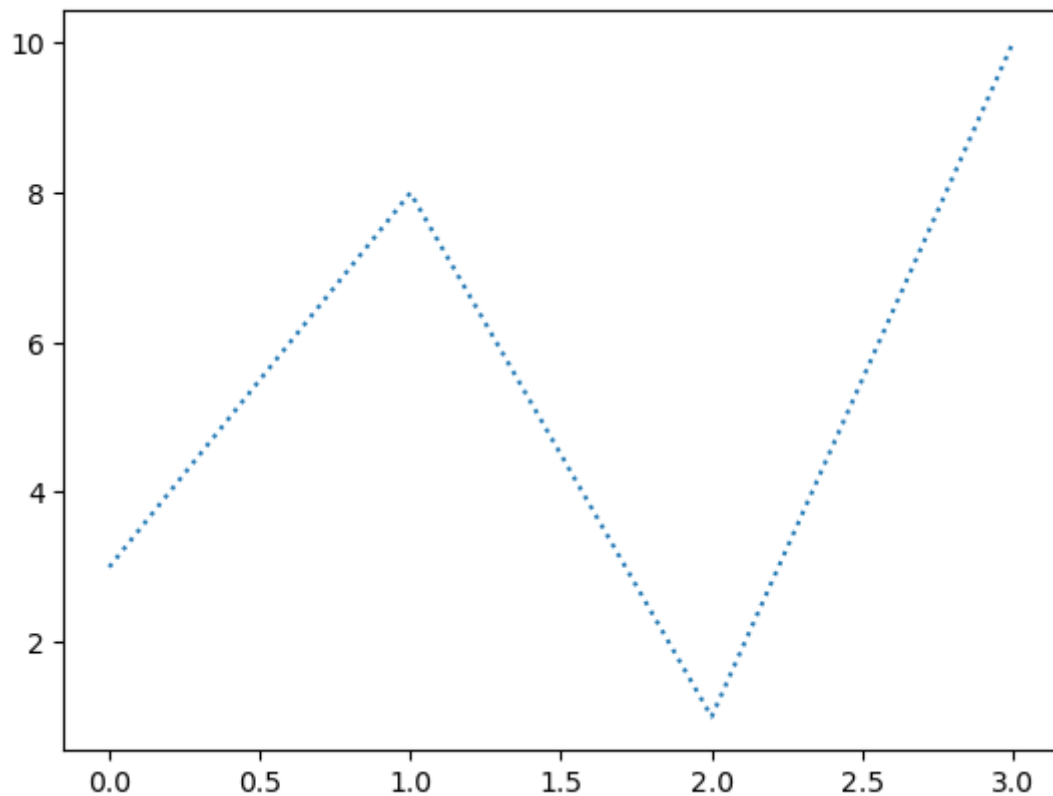
```
In [20]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, linestyle = 'dashed')  
plt.show()
```



#### 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 **--**.

```
In [21]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, ls = ':')  
plt.show()
```



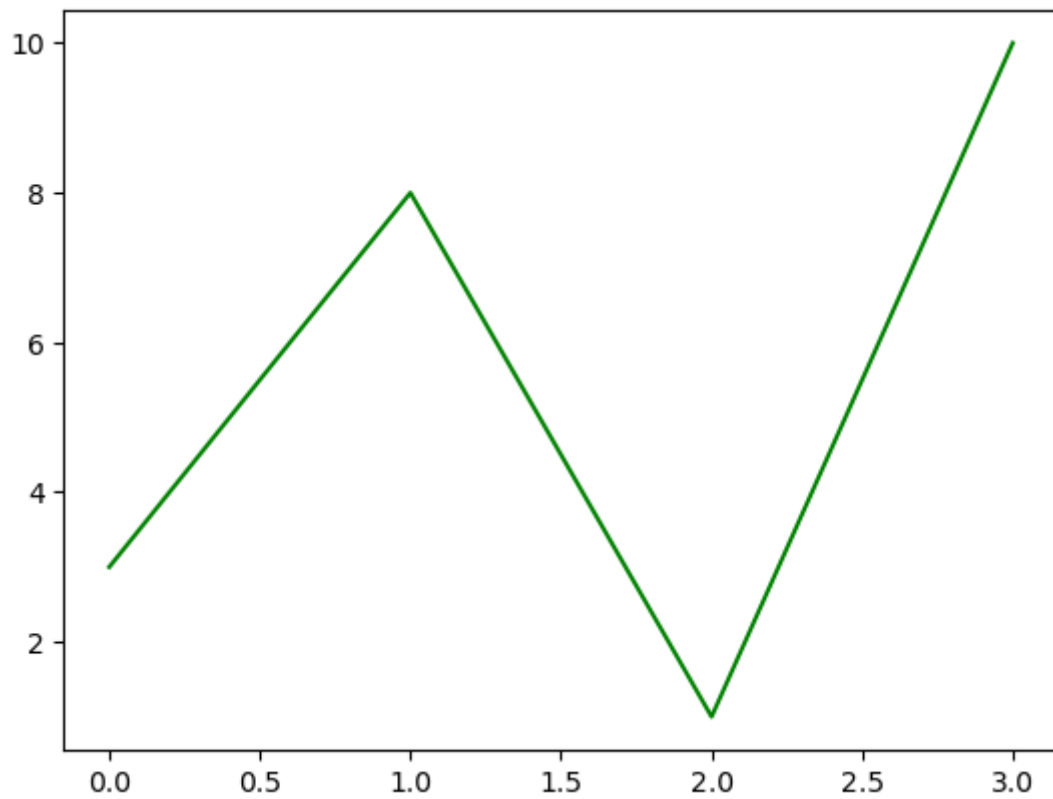
### Line Styles

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

### Line Color:

- We can use the keyword argument **color** or the shorter **c** to set the color of the line.

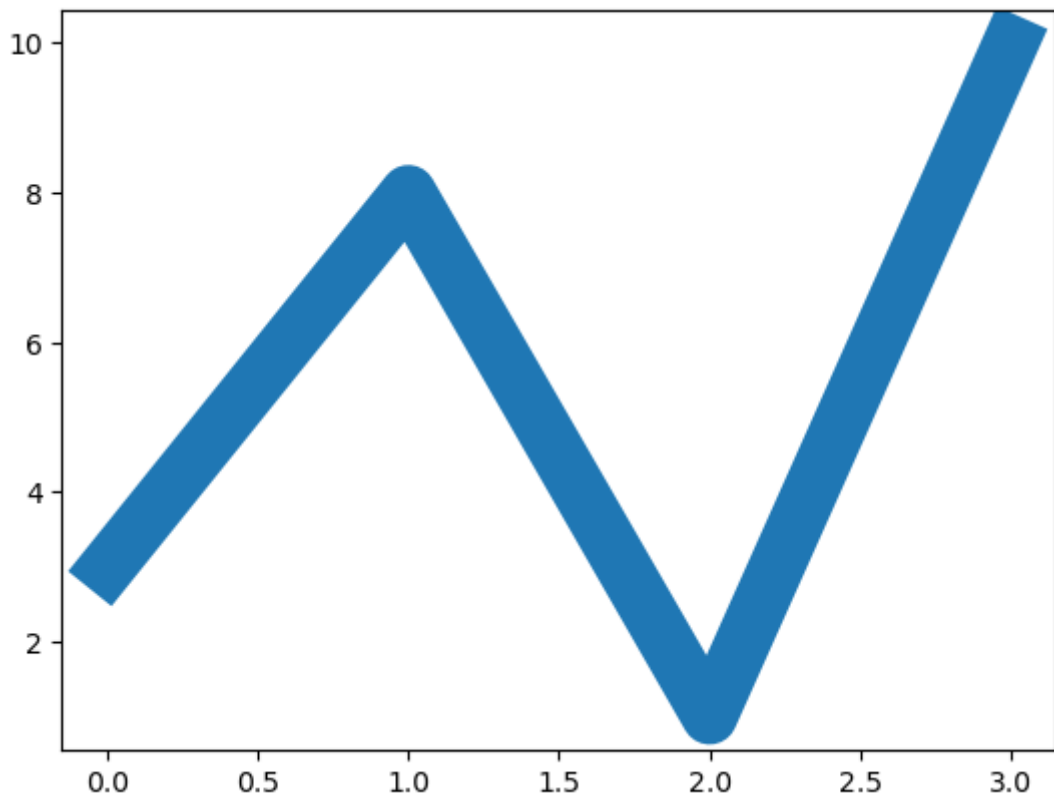
```
In [22]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, color = 'g')  
plt.show()
```



#### Line Width:

- We 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.

```
In [23]: ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, linewidth = '20')  
plt.show()
```



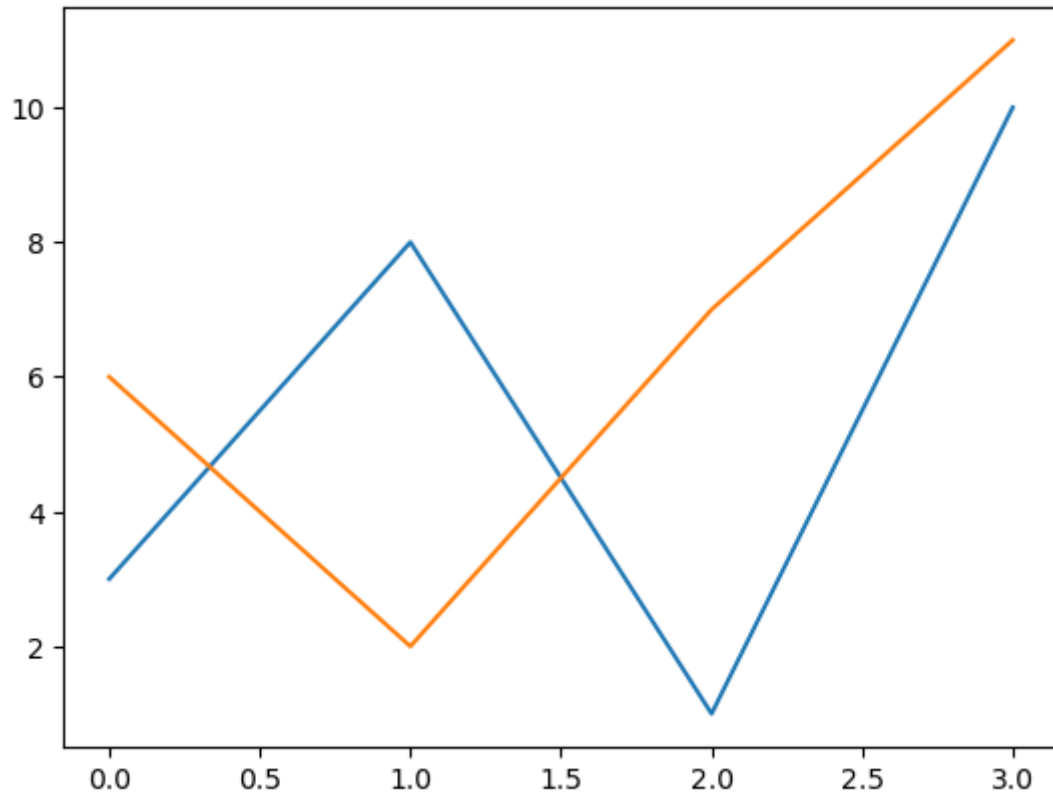
#### Multiple Lines in Single Plot:

- We can plot as many lines, by adding **plt.plot()** block.

```
In [24]: y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

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

plt.show()
```



## Matplotlib Labels and Title

---

### Create Labels for a Plot:

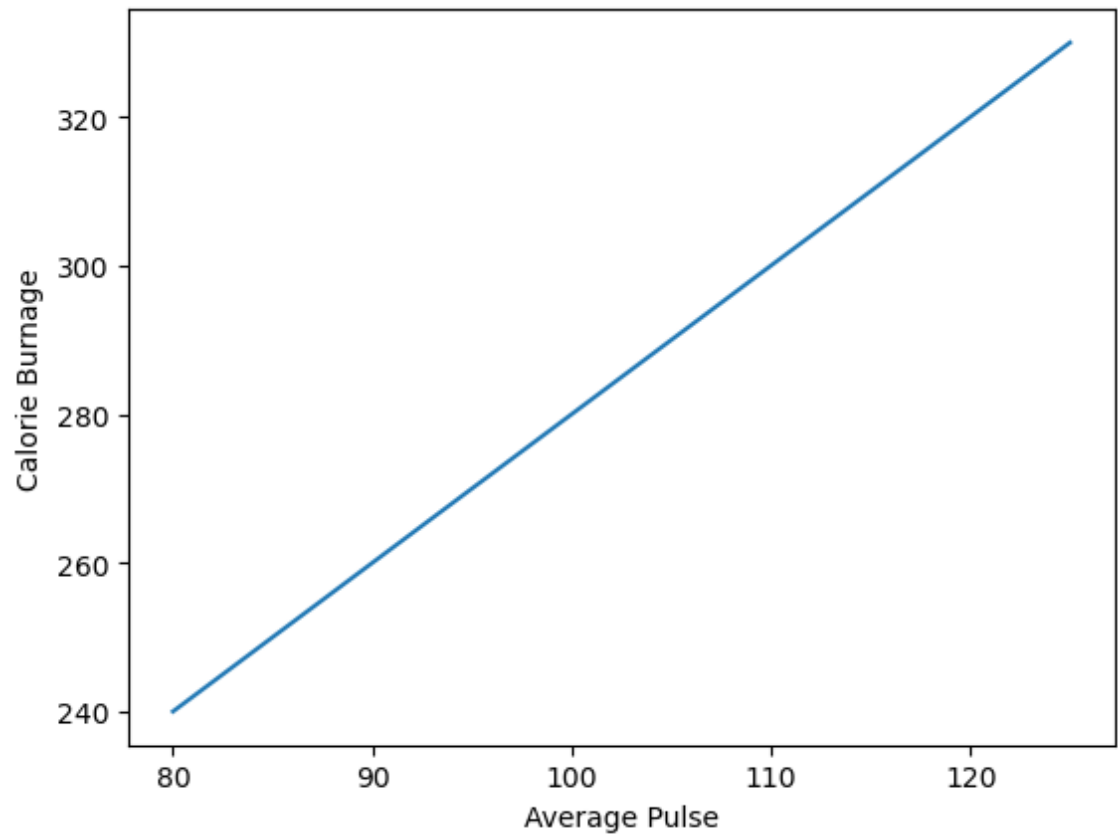
- With Pyplot, can use the **xlabel()** and **ylabel()** functions to set a label for the x- and y-axis.

```
In [25]: 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:

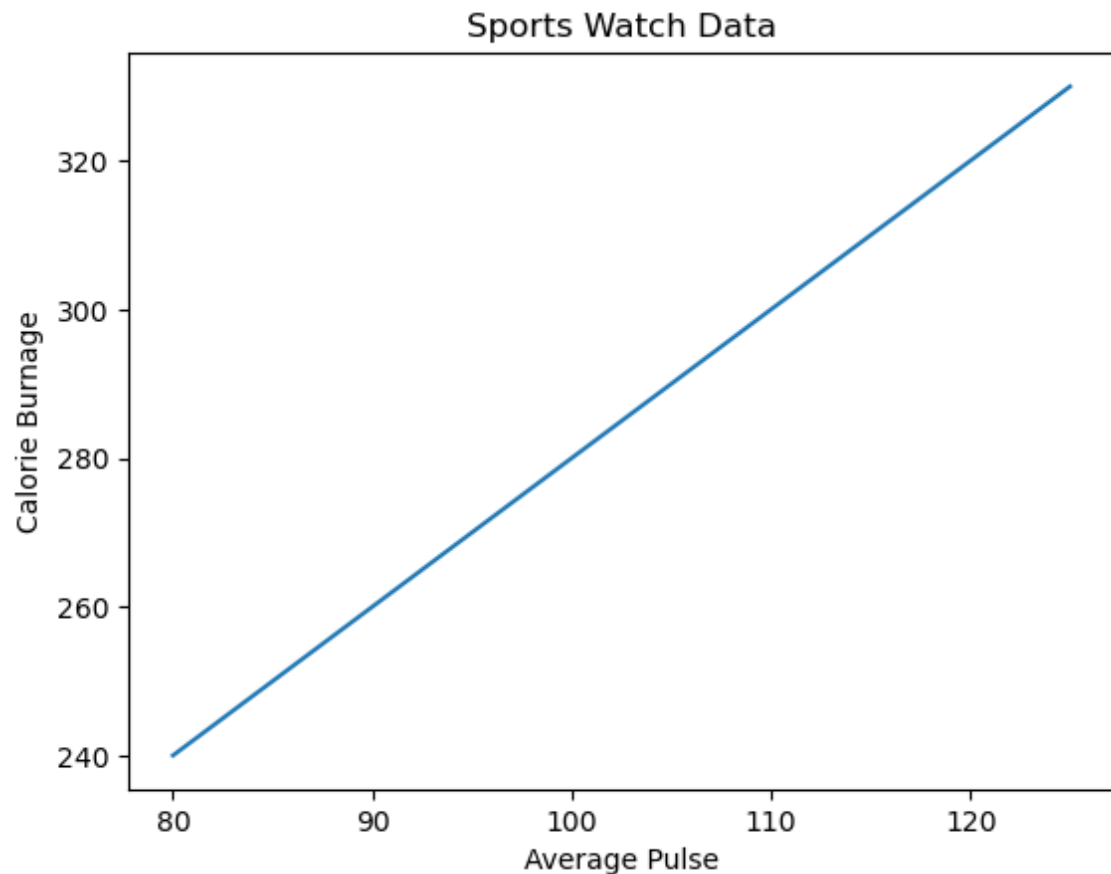
- With Pyplot, we can use the **title()** function to set a title for the plot.

```
In [26]: 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:

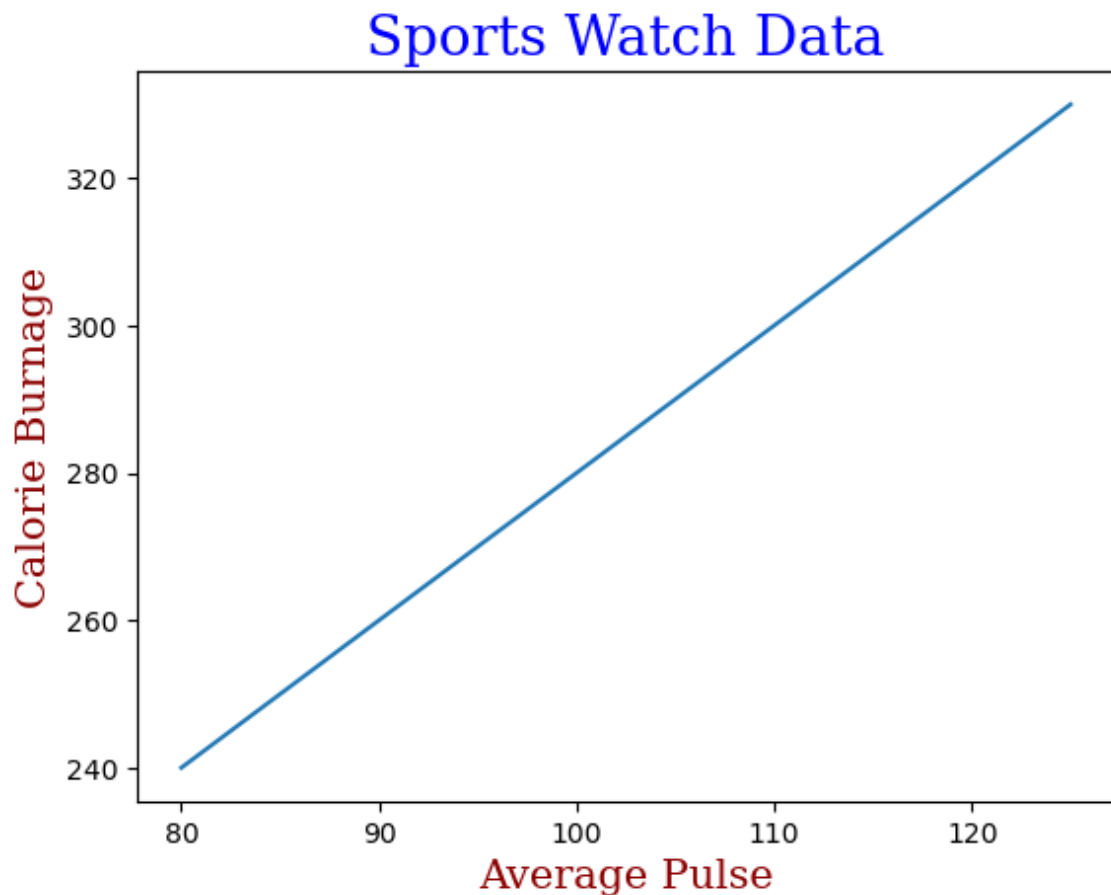
- We can use the **fontdict** parameter in **xlabel()**, **ylabel()**, and **title()** to set font properties for the title and labels.

```
In [27]: 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:

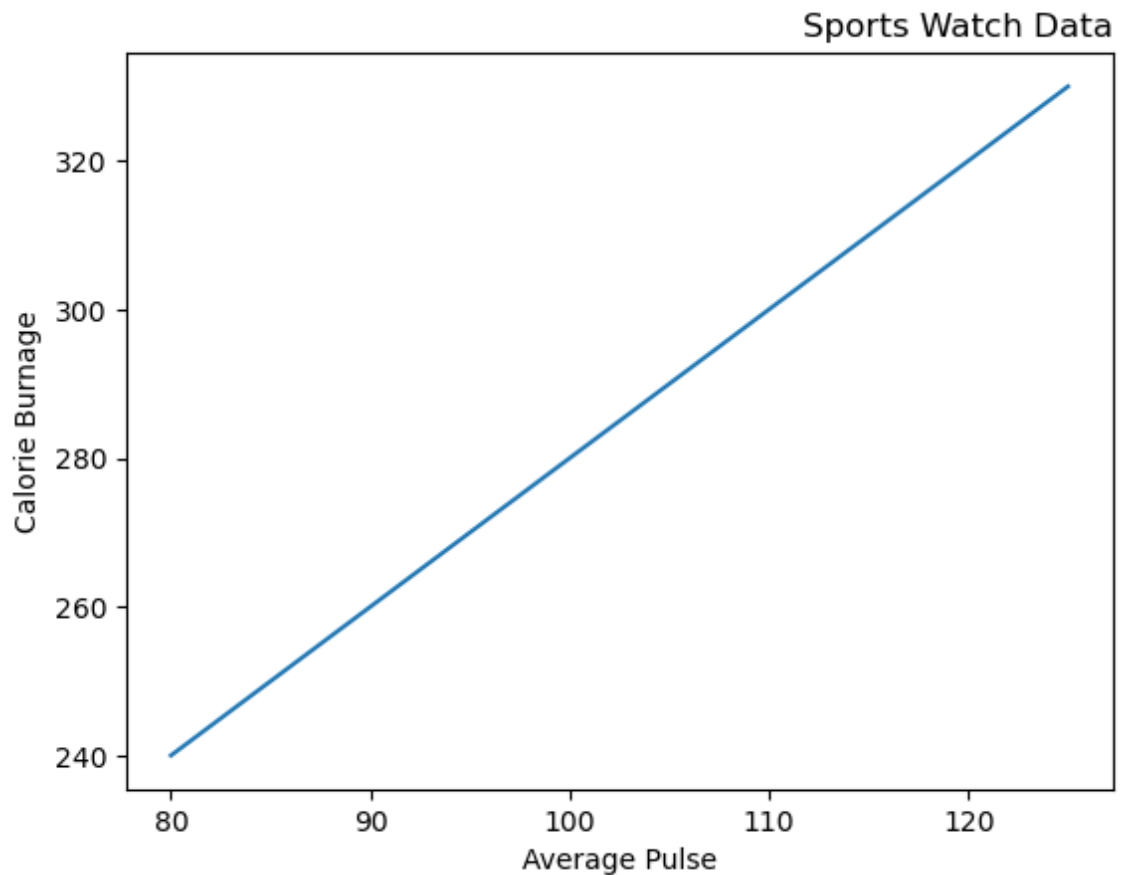
- We can use the **loc** parameter in **title()** to position the title.
- Legal values are: **'left', 'right', and 'center'**. Default value is **'center'**.



```
In [28]: 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 = 'right')
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, we can use the **grid()** function to add grid lines to the plot.

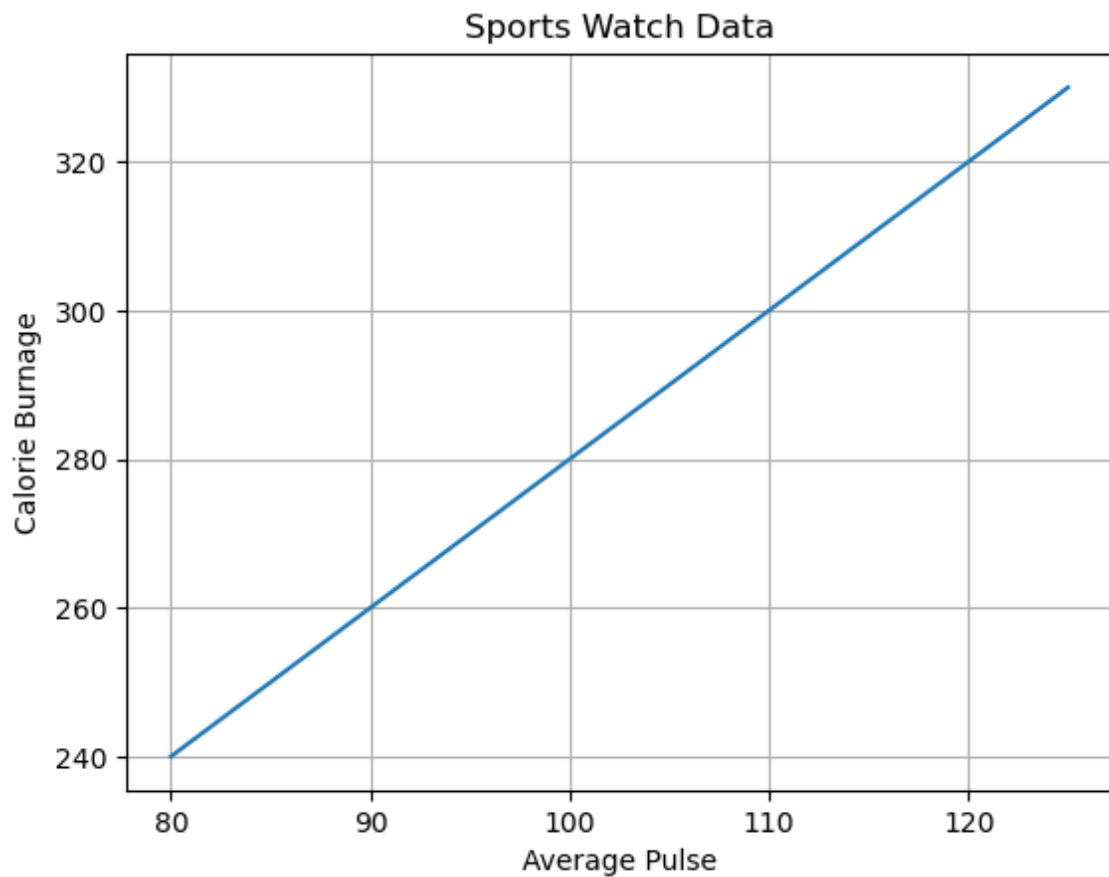
```
In [29]: x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

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

plt.plot(x, y)

plt.grid()

plt.show()
```



### 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'.

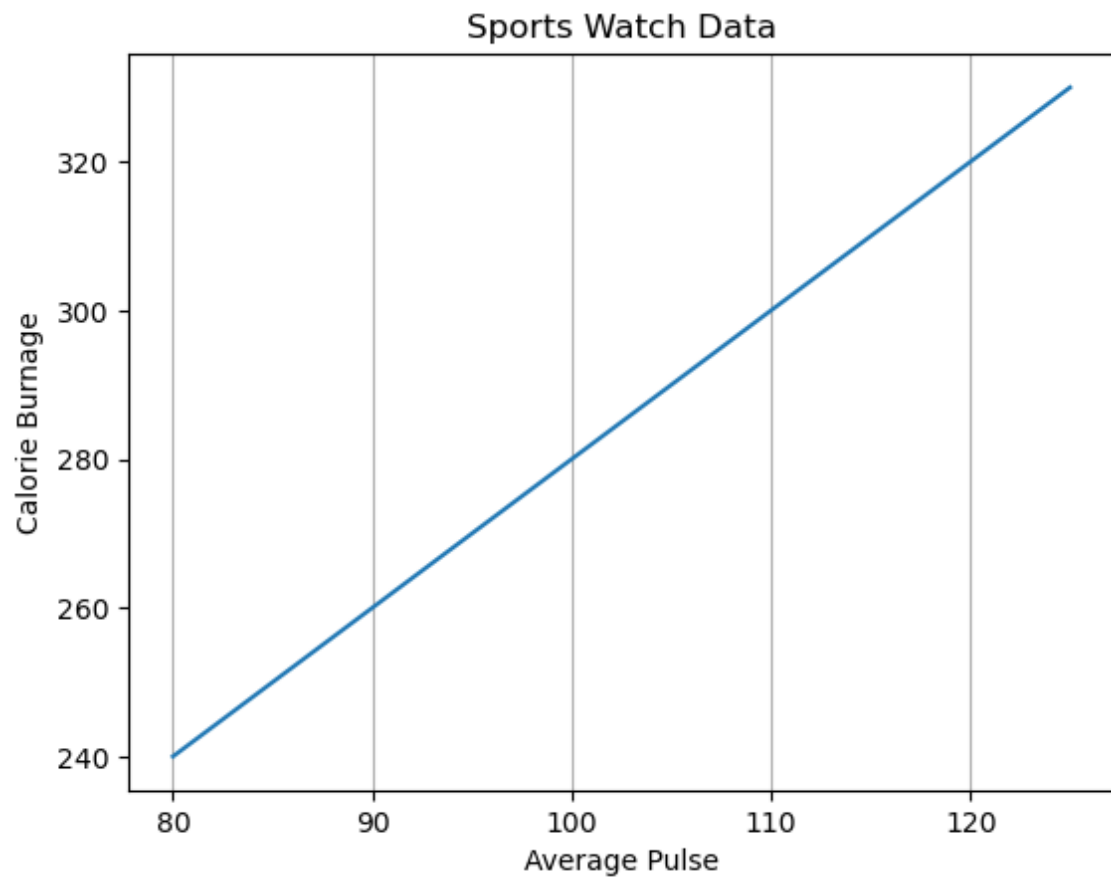
```
In [30]: x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

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

plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```



### Set Line Properties for the Grid:

- We can also set the line properties of the grid, like the following command:

```
grid(color = 'color', linestyle = 'linestyle', linewidth = number).
```

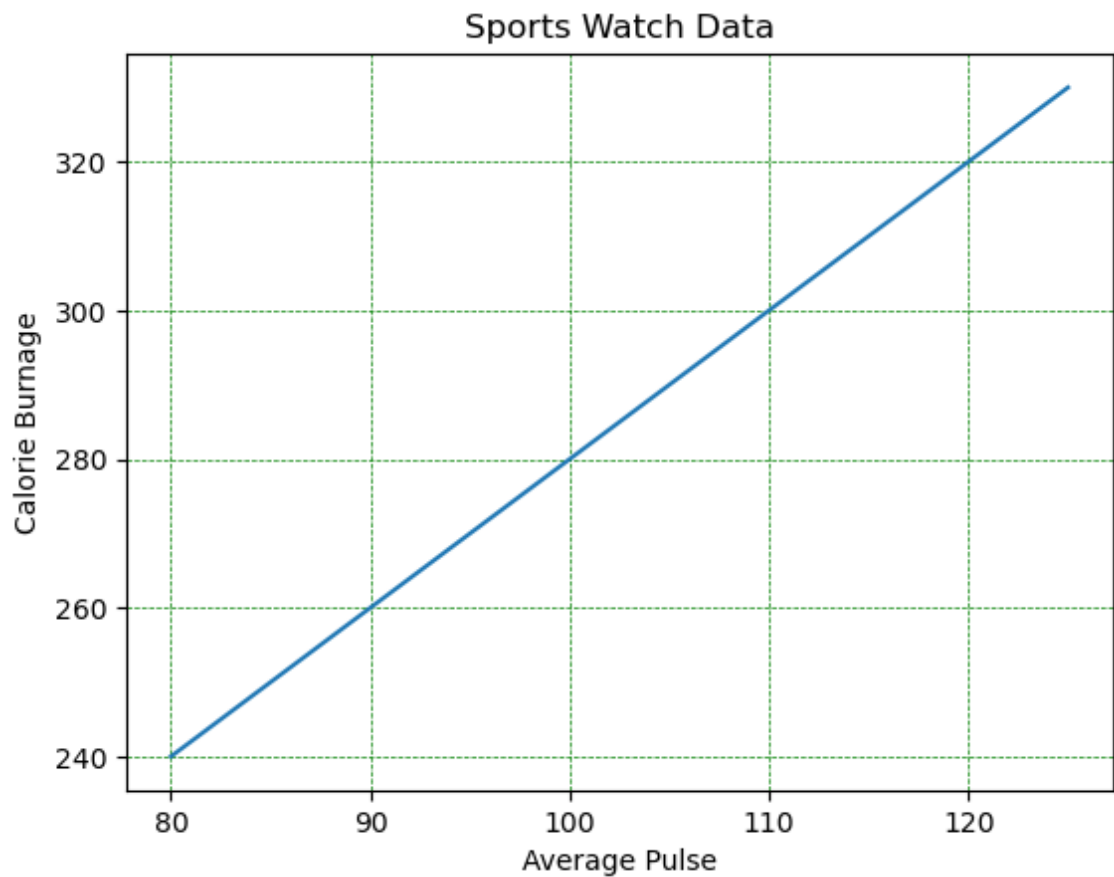
```
In [31]: x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

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()
```

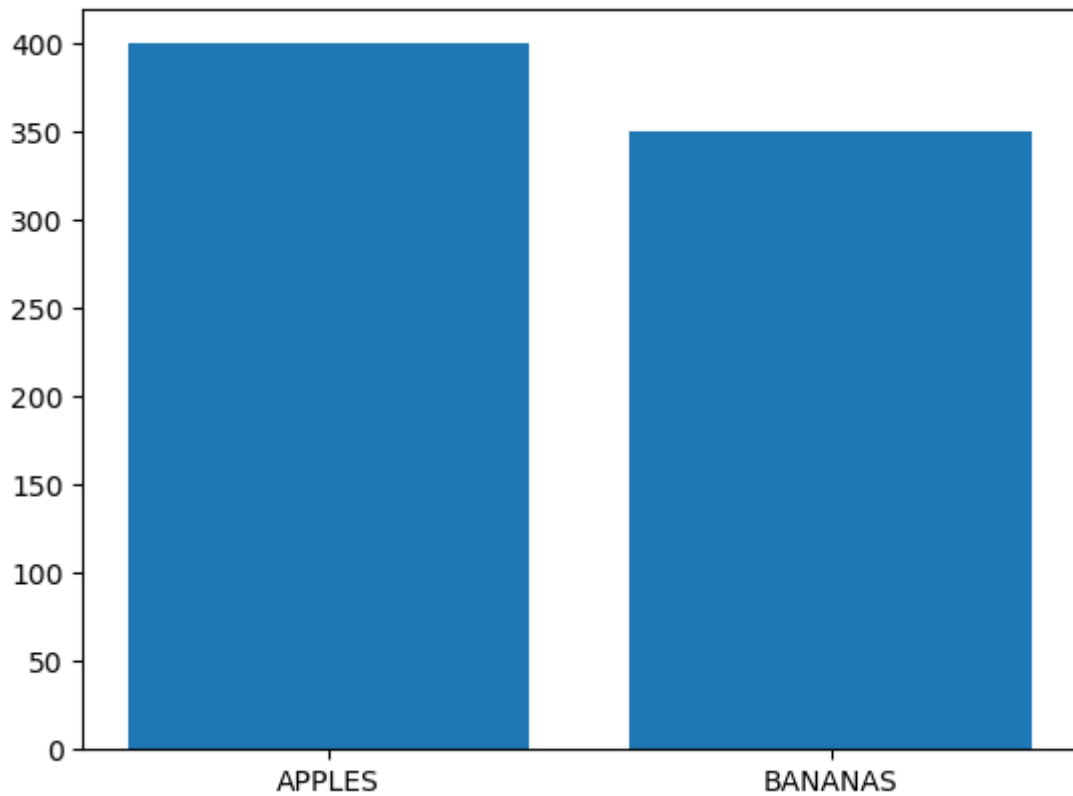


## Matplotlib Bars

- The **bar()** function takes arguments that describes the layout of the bars.
- The categories and their values represented by the first and second argument as arrays.

```
In [32]: x = ["APPLES", "BANANAS"]  
y = [400, 350]  
plt.bar(x, y)
```

Out[32]: <BarContainer object of 2 artists>

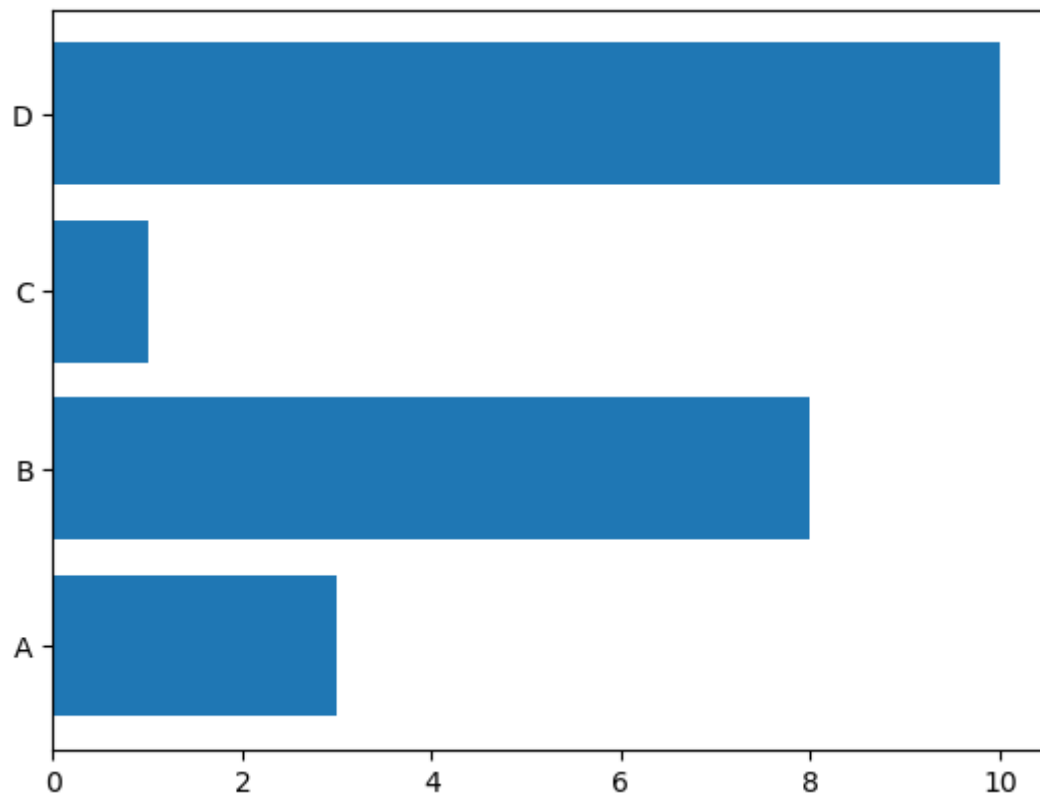


#### Horizontal Bars:

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

```
In [33]: x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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

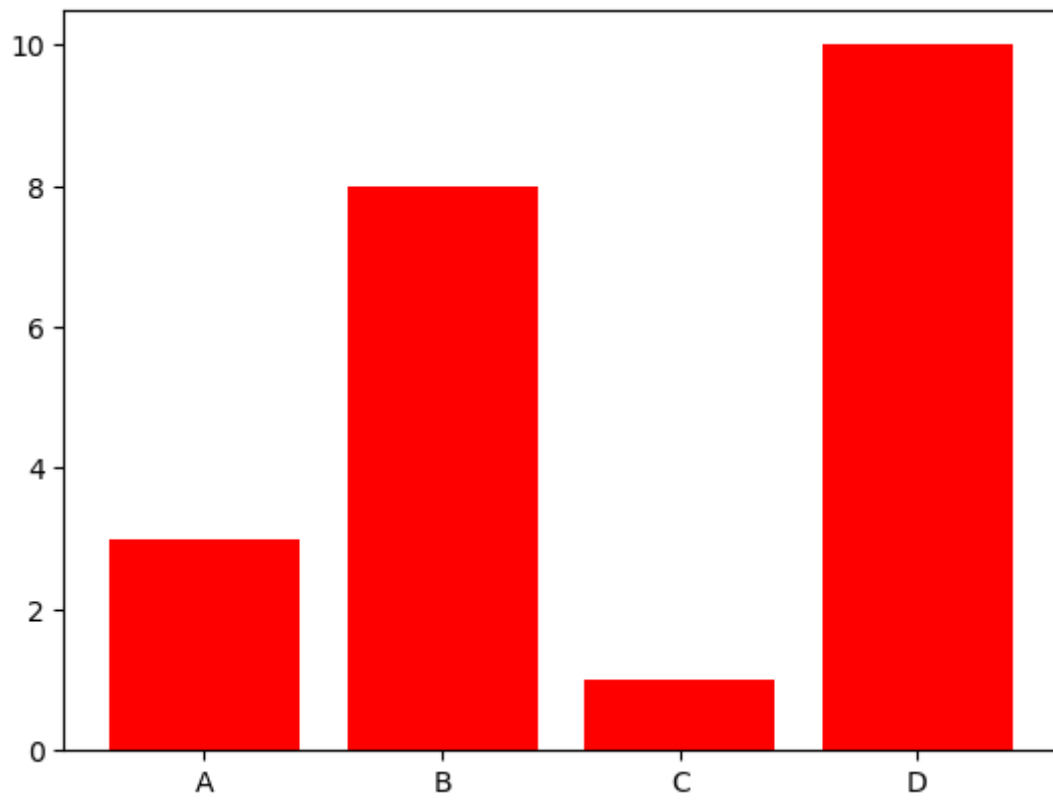


#### Bar Color:

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

```
In [34]: x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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

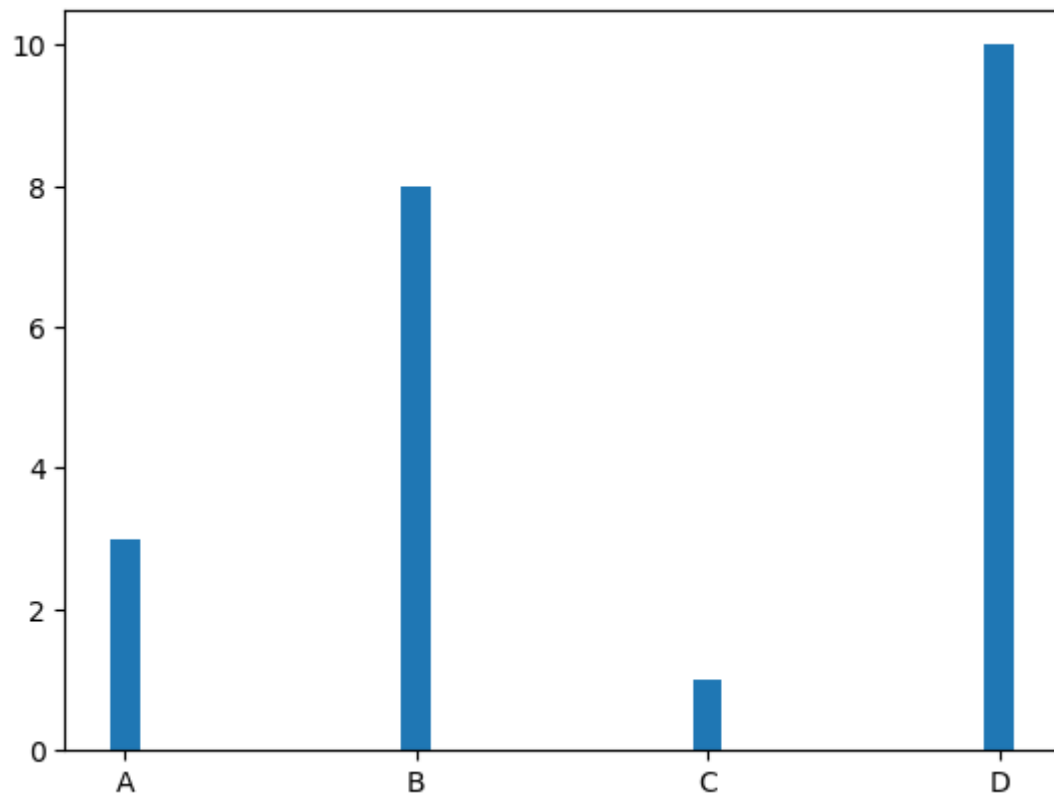


#### Bar Width:

- The **bar()** takes the keyword argument **width** to set the width of the bars.
- The default width value is 0.8.
- **Note:** For horizontal bars, use **height** instead of width.

```
In [35]: x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



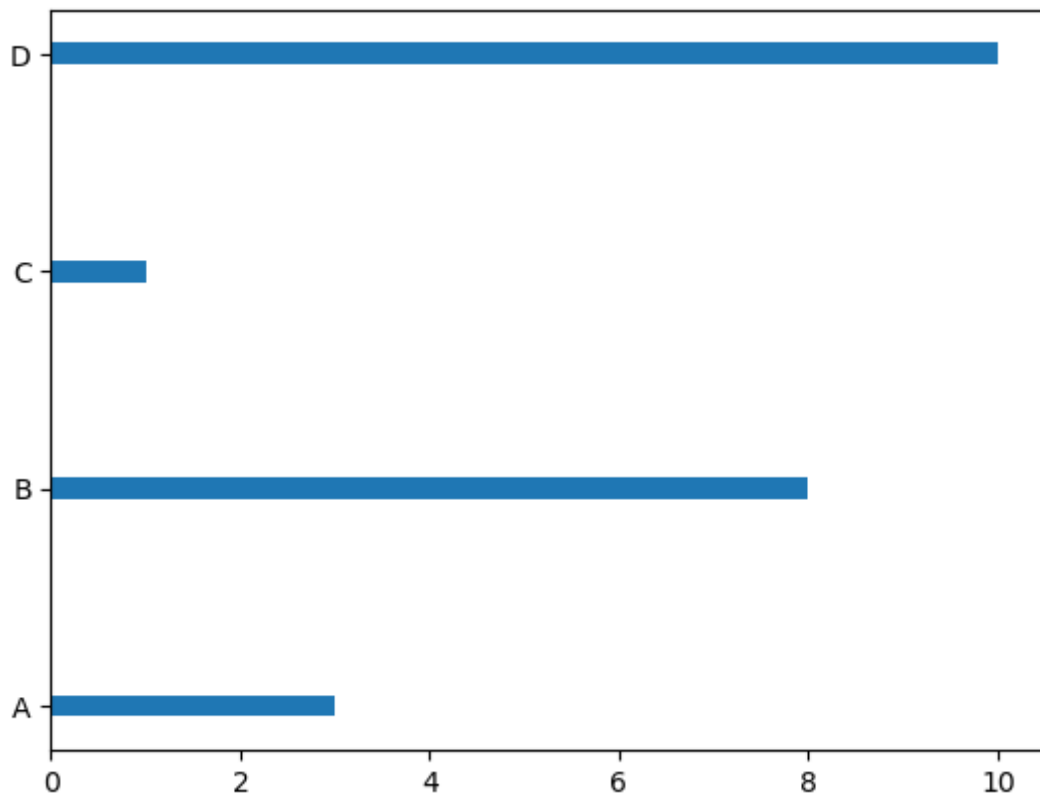
#### Bar Height:

- The **barh()** takes the keyword argument **height** to set the height of the bars.
- The default height value is **0.8**



```
In [36]: x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



In [37]:

```
# Expense categories
categories = ['Housing', 'Transportation', 'Food', 'Entertainment', 'Utilities']

# Monthly expenses for Alice, Bob, and Carol
alice_expenses = [1200, 300, 400, 200, 150]
bob_expenses = [1100, 320, 380, 180, 140]
carol_expenses = [1300, 280, 420, 220, 160]

# Create an array for the x-axis positions
x = np.arange(len(categories))

# Width of the bars
bar_width = 0.2

# Create bars for Alice's expenses
plt.bar(x - bar_width, alice_expenses, width=bar_width, label='Alice', color='lightcoral')

# Create bars for Bob's expenses
plt.bar(x, bob_expenses, width=bar_width, label='Bob', color='lightcoral')

# Create bars for Carol's expenses
plt.bar(x + bar_width, carol_expenses, width=bar_width, label='Carol', color='lightcoral')

# Add labels, a title, and a legend
plt.xlabel('Expense Categories')
plt.ylabel('Monthly Expenses (USD)')
plt.title('Monthly Expenses Comparison')
plt.xticks(x, categories)
plt.legend()

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



In [ ]: