Name: Prem Vilas Dhanawade

Github link:https://github.com/premdhanawade/Python/tree/main/P (https://github.com/premdhanawade/Python/tree/main/Pyth

- Python Matplotlib
  - Matplotlib Intro
  - Matplotlib Get Started
  - Matplotlib Pyplot
  - Matplotlib Plotting
  - Matplotlib Markers
  - Matplotlib Line
  - Matplotlib Labels
  - Matplotlib Grid
  - Matplotlib Subplots
  - Matplotlib Scatter
  - Matplotlib Bars
  - Matplotlib Histograms
  - Matplotlib Pie Charts

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.

Where is the Matplotlib Codebase?

- The source code for Matplotlib is located at this github repository https://github.com/matplotlib/matplotlib (https://github.com/matplotlib/matplotlib)

Checking Matplotlib Version

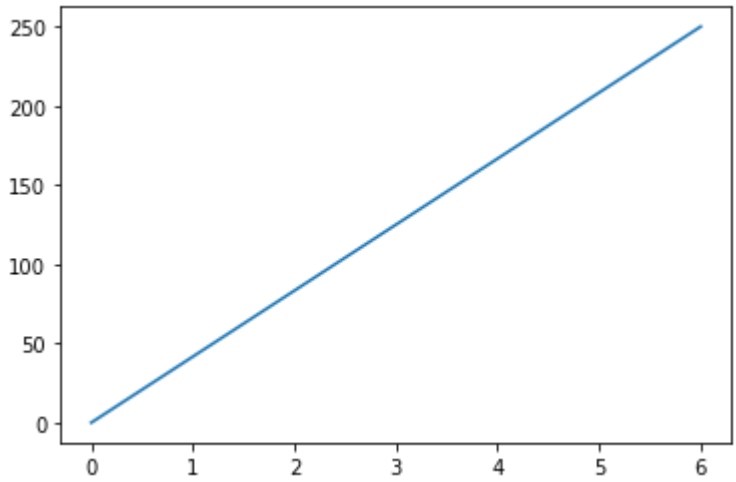
- The version string is stored under **version** attribute.

In [1]:
```
import matplotlib

print(matplotlib.__version__)
```

```
3.3.2
```

Matplotlib Pyplot

In [1]:
```
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()
```

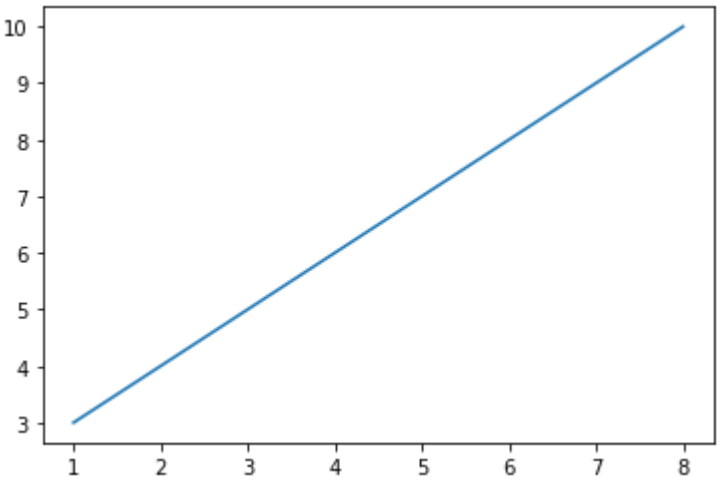


Matplotlib Plotting

Plotting x and y points

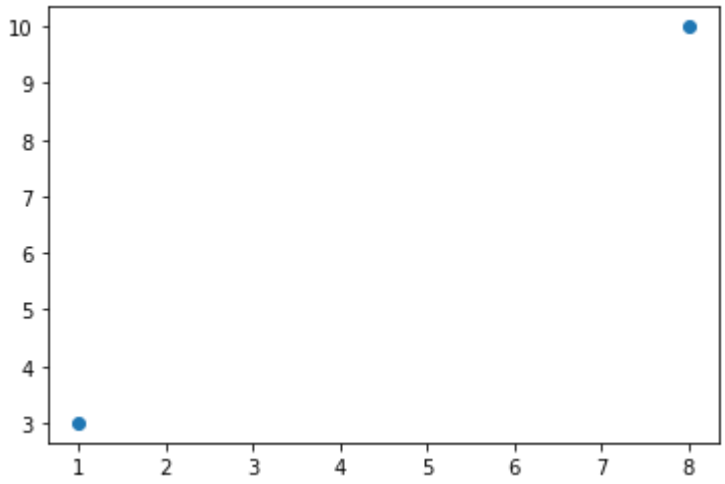
- The plot() function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the x-axis.
- Parameter 2 is an array containing 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.

In [2]:
```
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

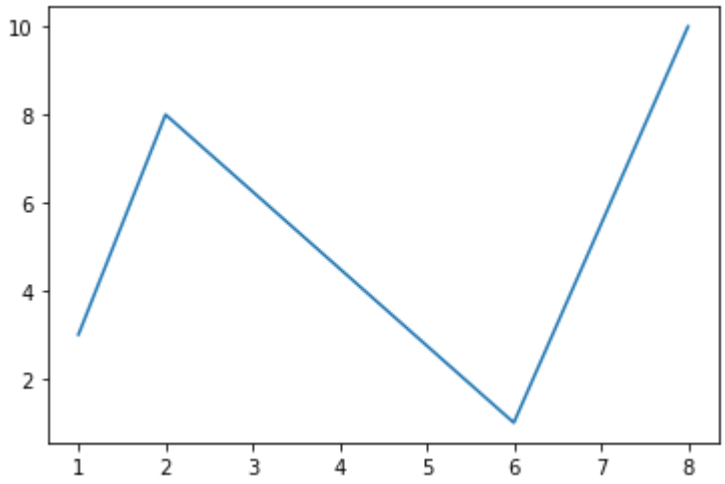
In [3]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xpoints = np.array([1, 8])
5 ypoints = np.array([3, 10])
6
7 plt.plot(xpoints, ypoints, 'o')
8 plt.show()
```



Multiple Points

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

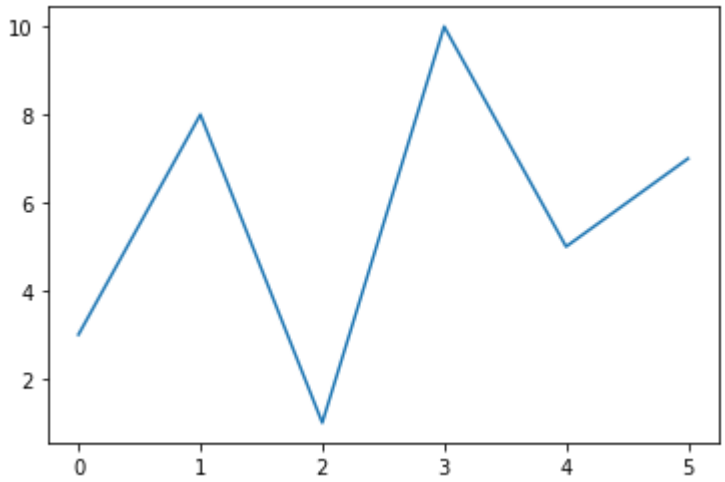
In [4]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xpoints = np.array([1, 2, 6, 8])
5 ypoints = np.array([3, 8, 1, 10])
6
7 plt.plot(xpoints, ypoints)
8 plt.show()
```



Default X-Points

- If we do not specify the points in 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:

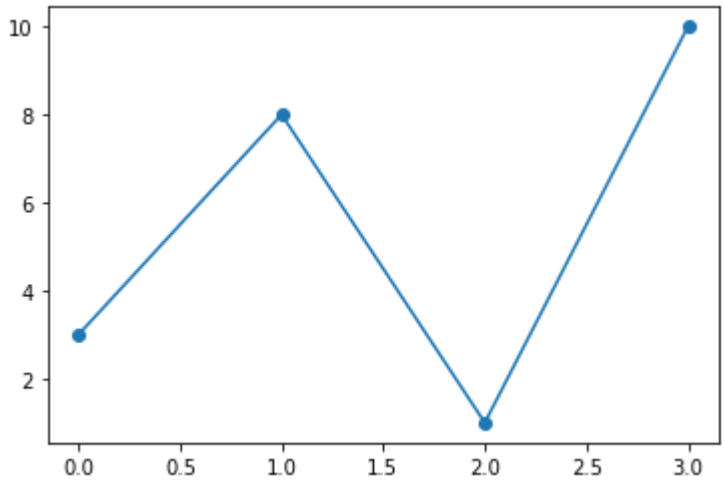
In [5]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10, 5, 7])
5
6 plt.plot(ypoints)
7 plt.show()
```



Matplotlib Markers

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

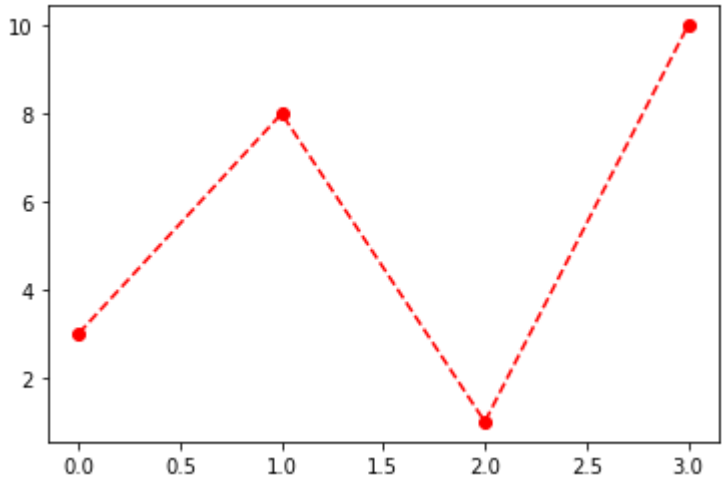
In [6]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, marker = 'o')
7 plt.show()
```



Format Strings fmt

- You can use 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

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Color Reference

- 'r'----- Red
- 'g'----- Green
- 'b'----- Blue
- 'c'----- Cyan
- 'm'----- Magenta
- 'y'----- Yellow
- 'k'----- Black
- 'w'----- White

Line Reference

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

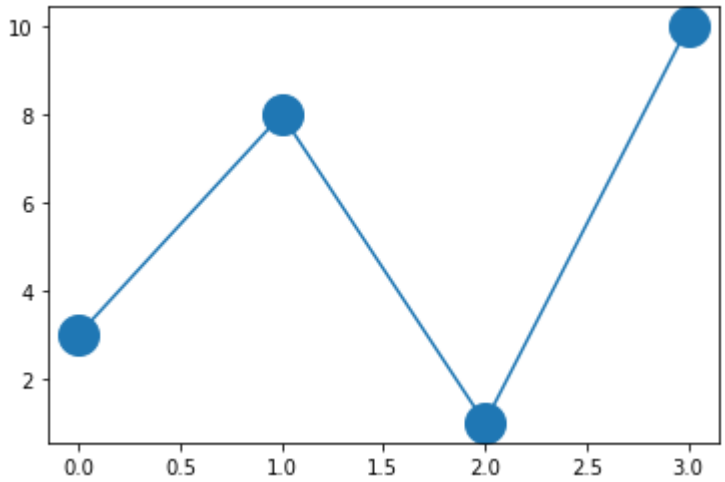
- 'o'------(Circle)

- '*'------(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)

Marker Size

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

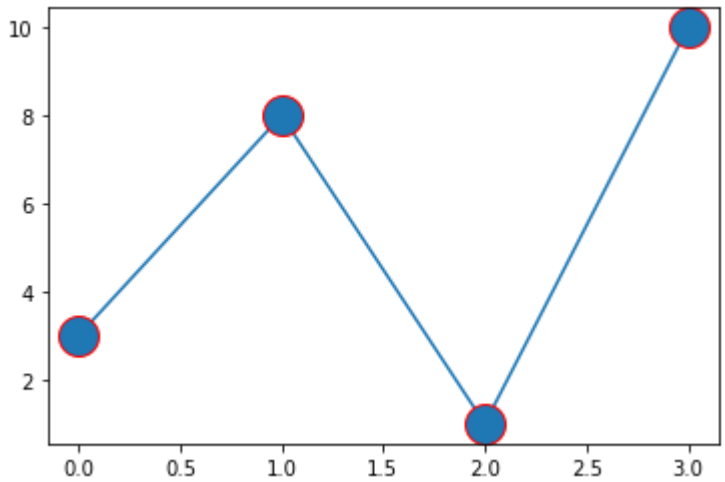
In [9]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Marker Color

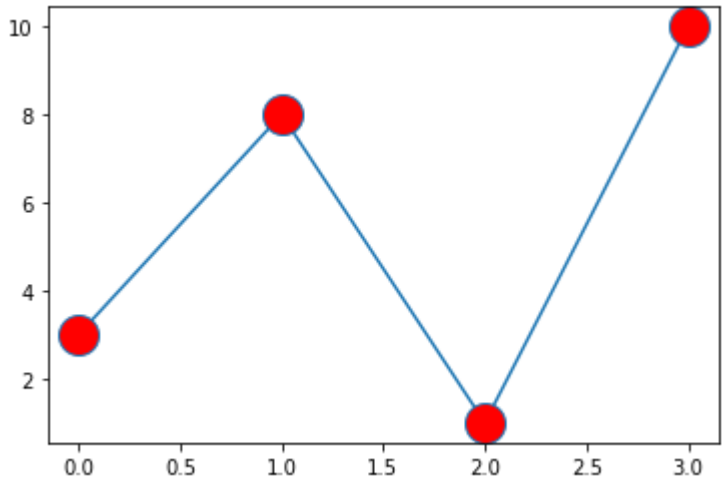
- You can use the keyword argument markeredgecolor or the shorter mec to set the color of the edge of the markers:

In [10]:
```
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()
```

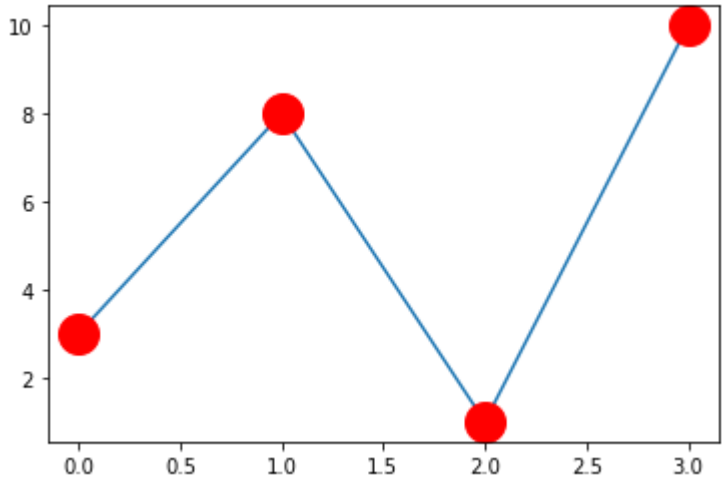


In [ ]:
```
1
```

In [11]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
7 plt.show()
8
```



In [12]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
7 plt.show()
```



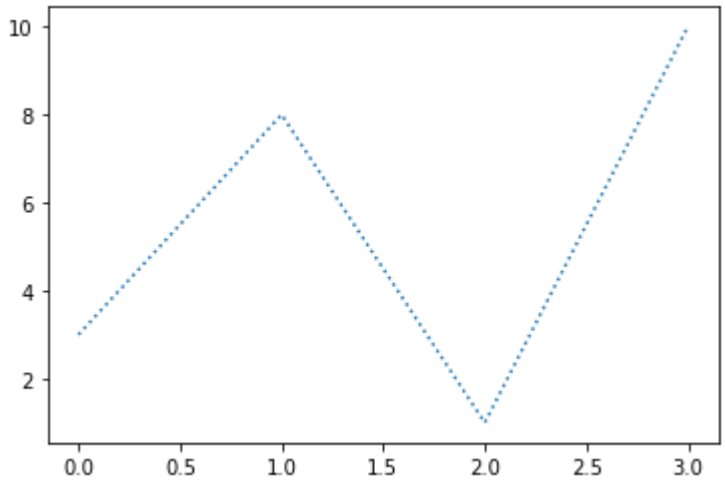
Set the color of both the edge and the face to red:

Matplotlib Line

Linestyle

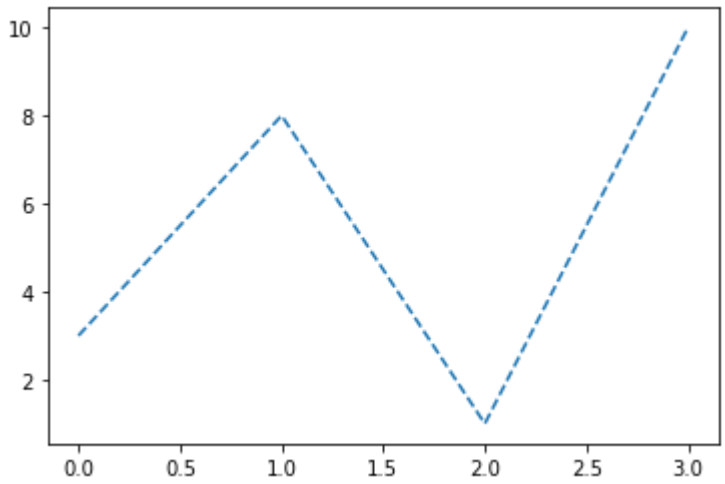
- You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

In [13]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, linestyle = 'dotted')
7 plt.show()
```



In [14]:
```
1 plt.plot(ypoints, linestyle = 'dashed')
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x5893dc0>]
```



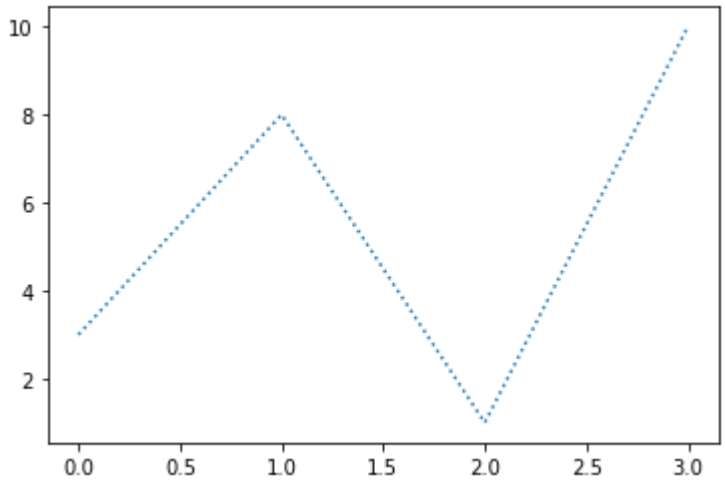
Shorter Syntax

- 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 [15]:
```
1 plt.plot(ypoints, ls = ':')
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x6d13a60>]
```



Line Styles

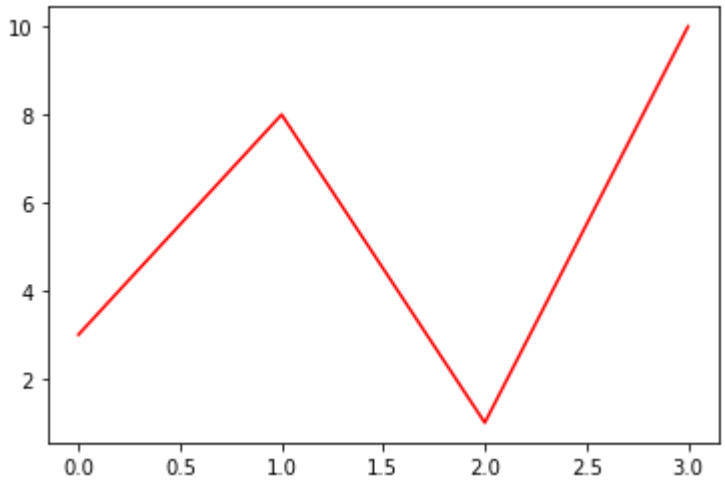
You can choose any of these styles:

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

Line Color

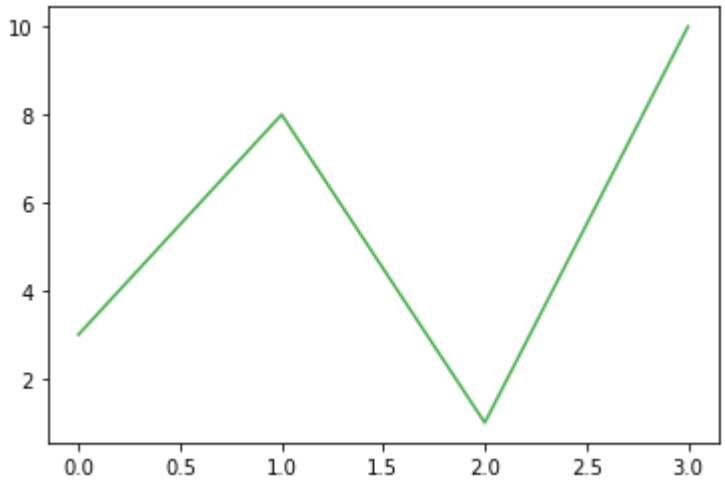
- You can use the keyword argument color or the shorter c to set the color of the line:

In [16]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, color = 'r')
7 plt.show()
```



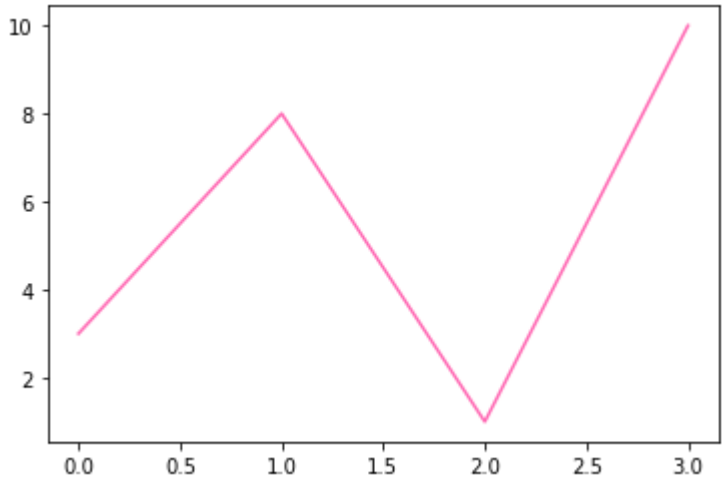
In [17]:
```
1 plt.plot(ypoints, c = '#4CAF50')
```

Out[17]: [<matplotlib.lines.Line2D at 0x6da05e0>]



In [18]:
```
1 plt.plot(ypoints, c = 'hotpink')
```

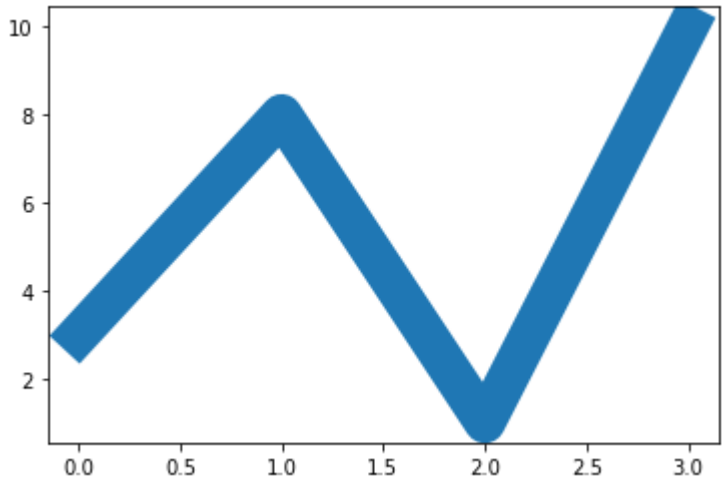
Out[18]:
```
[<matplotlib.lines.Line2D at 0x6df48b0>]
```



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:

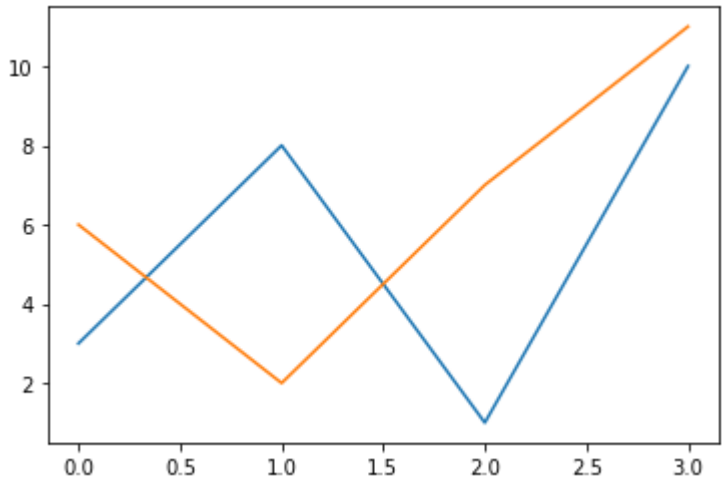
In [19]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, linewidth = '20.5')
7 plt.show()
```



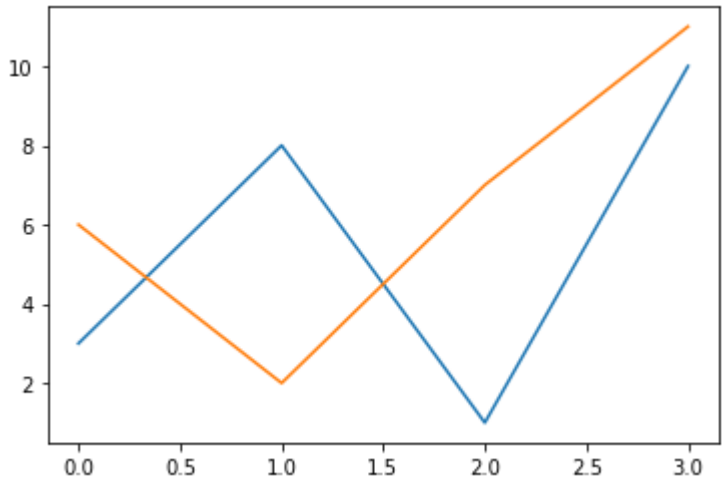
Multiple Lines

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

In [20]:
```
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()
```



In [21]:
```
#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()
```

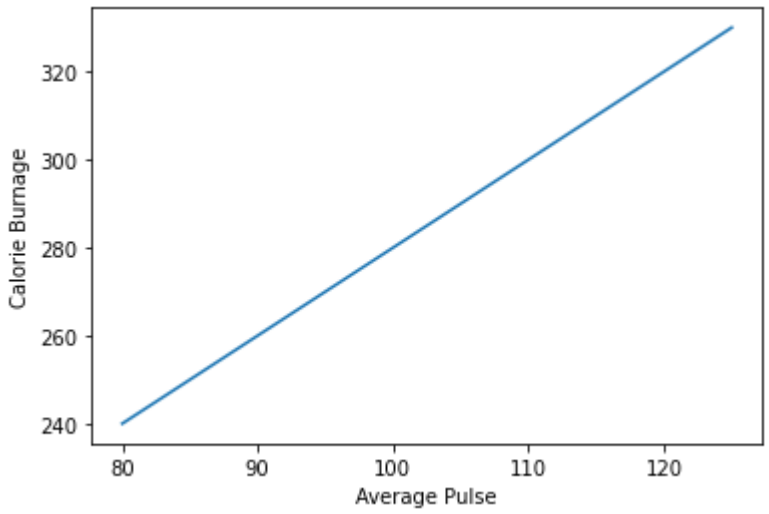


Matplotlib Labels and Title

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.

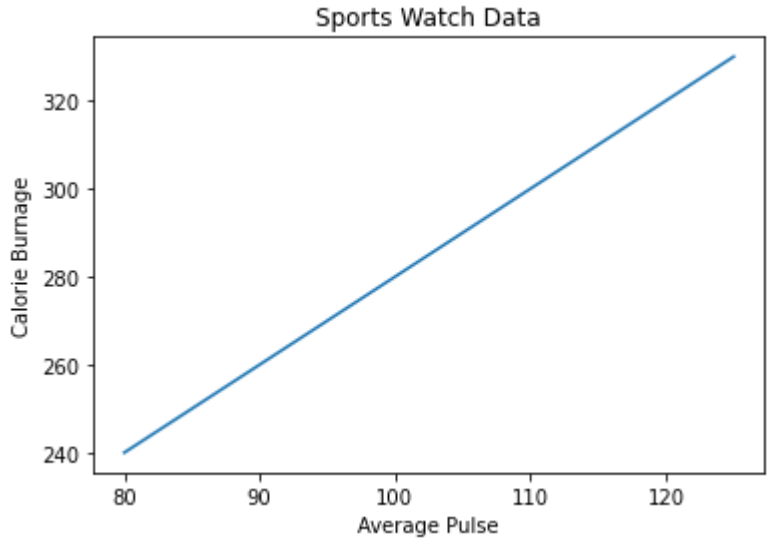
In [22]:
```
import numpy as np
import matplotlib.pyplot as plt

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.

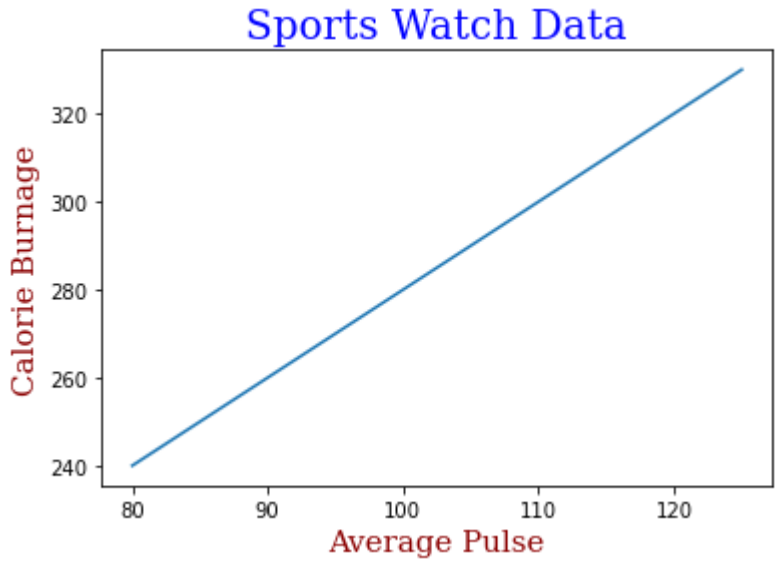
In [23]:
```
import numpy as np
import matplotlib.pyplot as plt

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.

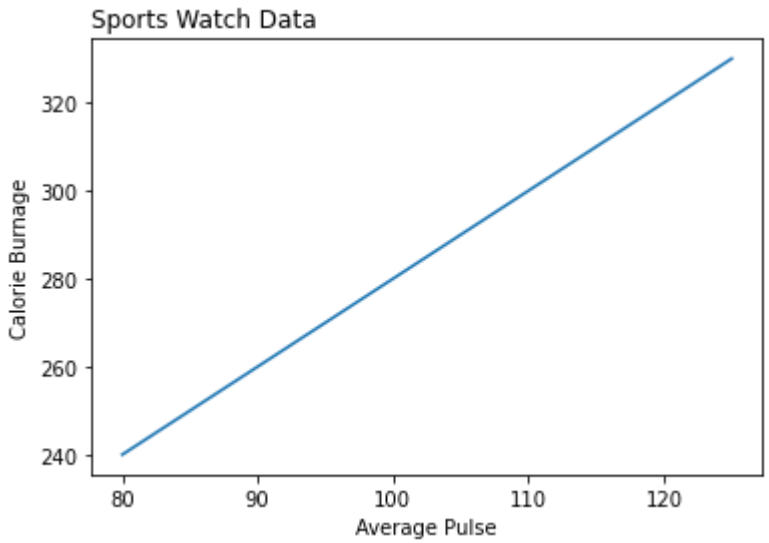
In [24]:
```
import numpy as np
import matplotlib.pyplot as plt

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'.

In [25]:
```
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()
```

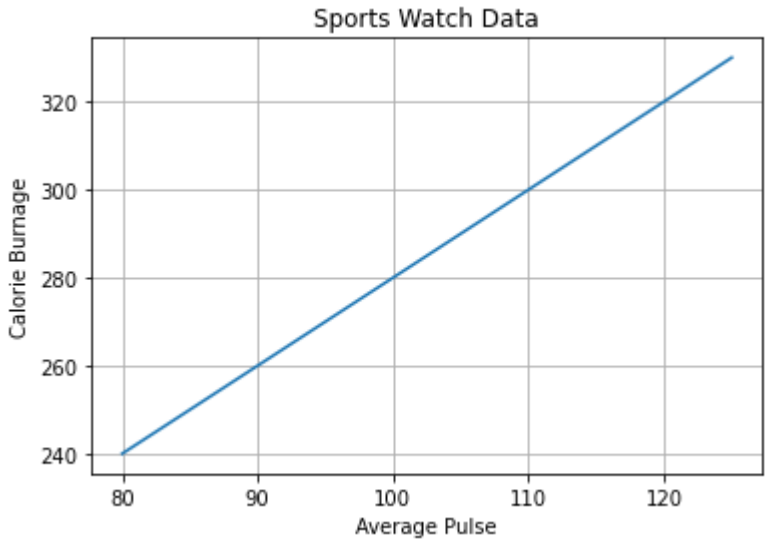


Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

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

In [26]:
```
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()
```

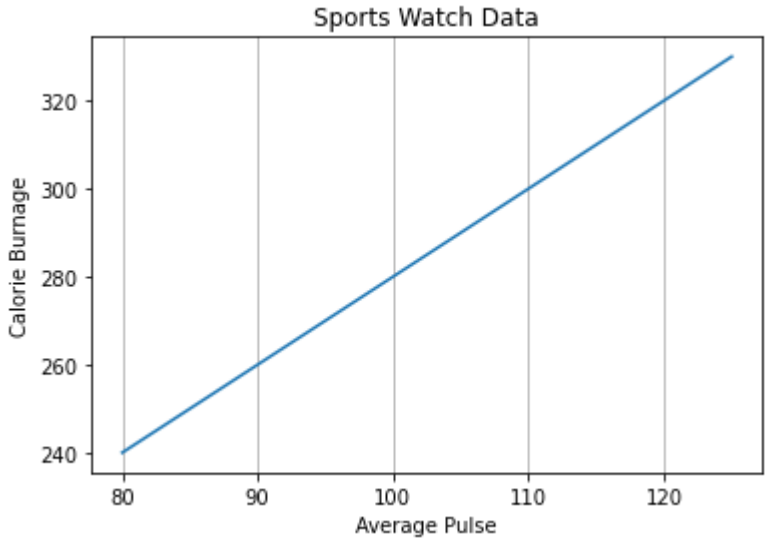


Specify Which Grid Lines to Display

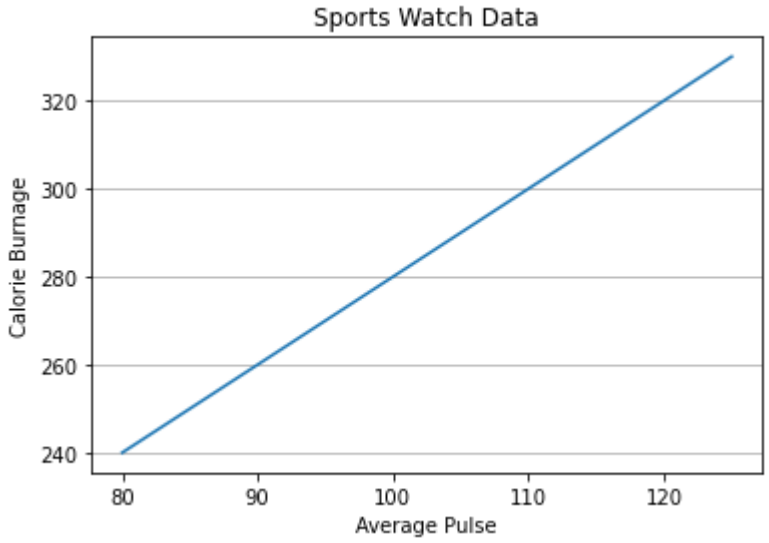
- You can use the axis parameter in the grid() function to specify which grid lines to display.
- Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

In [27]:

```
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()
```



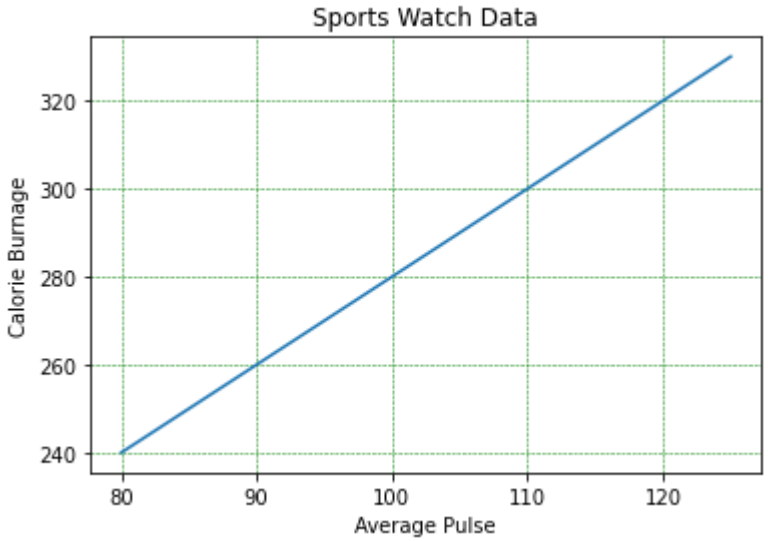
In [28]:
```
import numpy as np
import matplotlib.pyplot as plt

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).

In [29]:
```
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()
```

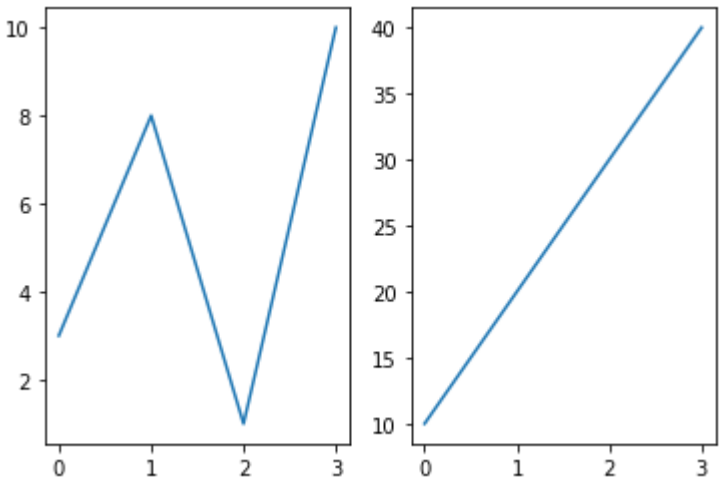


Matplotlib Subplots

Display Multiple Plots

- With the subplots() function you can draw multiple plots in one figure:

In [30]:
```
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()
```

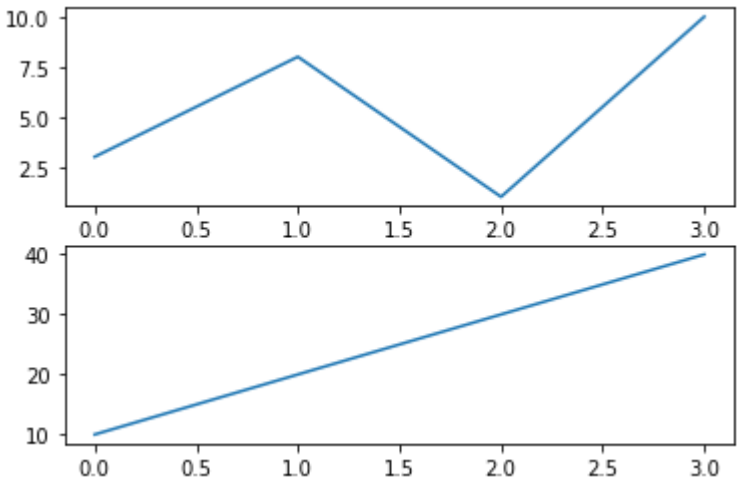


The subplots() Function

- The subplots() 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.

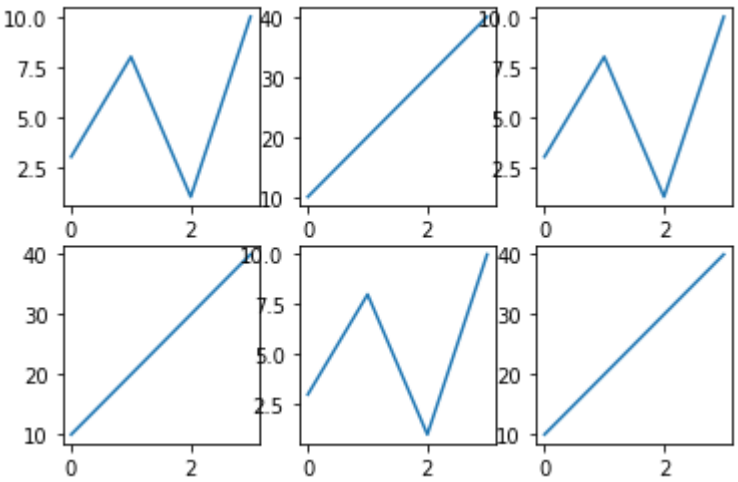
- So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

In [31]:
```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  #plot 1:
5  x = np.array([0, 1, 2, 3])
6  y = np.array([3, 8, 1, 10])
7
8  plt.subplot(2, 1, 1)
9  plt.plot(x,y)
10
11 #plot 2:
12 x = np.array([0, 1, 2, 3])
13 y = np.array([10, 20, 30, 40])
14
15 plt.subplot(2, 1, 2)
16 plt.plot(x,y)
17
18 plt.show()
```



In [32]:

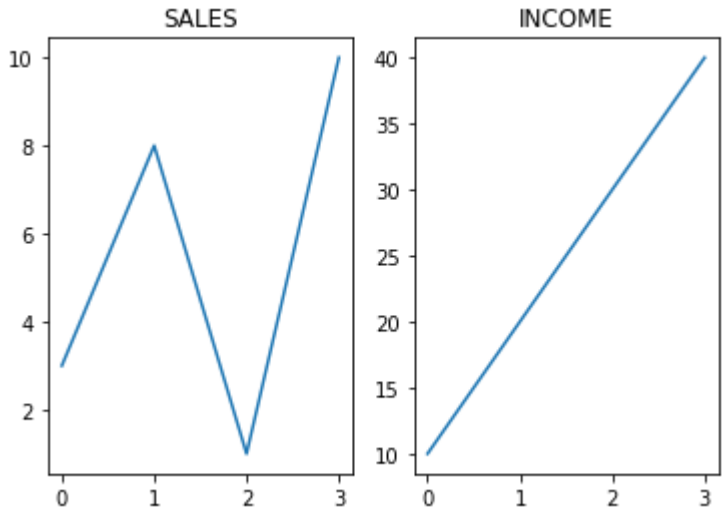
```
import matplotlib.pyplot as plt
import numpy as np

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:

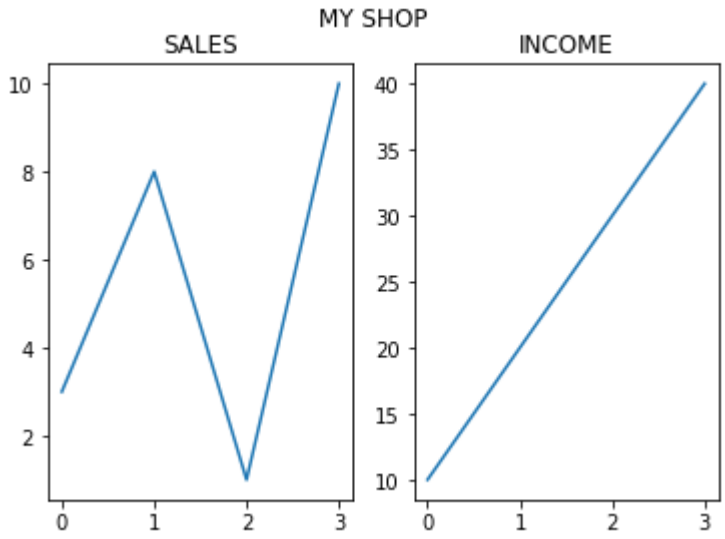
In [33]:
```
import matplotlib.pyplot as plt
import numpy as np

#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:

In [34]:
```
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()
```



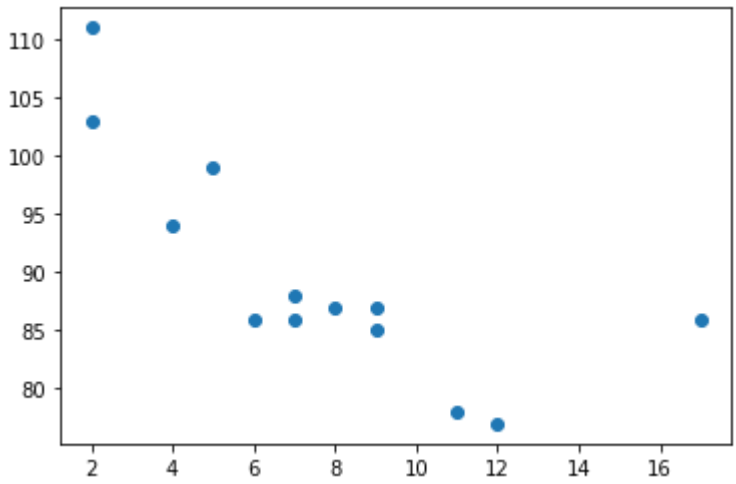
Matplotlib Scatter

Creating Scatter Plots

- 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:

In [35]:
```
import matplotlib.pyplot as plt
import numpy as np

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)
plt.show()
```



The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.

The Y-axis shows the speed of the car when it passes.

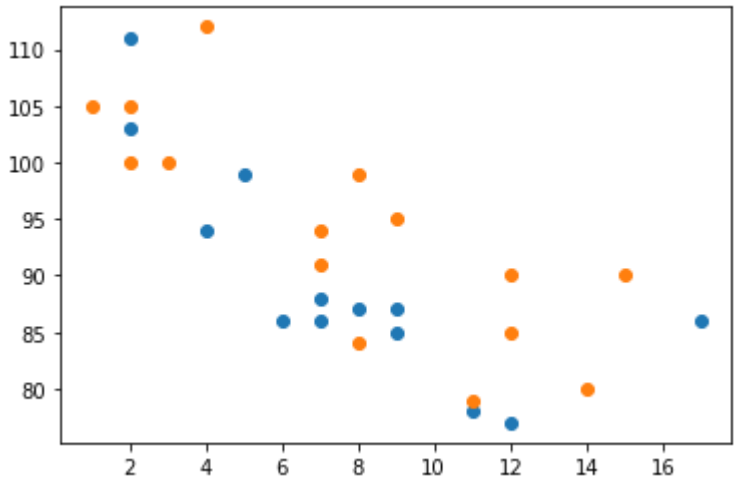
Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

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?

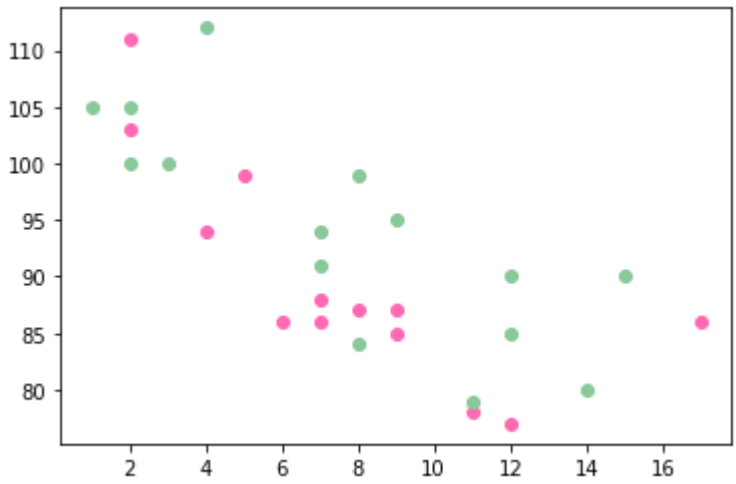
In [36]:
```
import matplotlib.pyplot as plt
import numpy as np

#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()
```



Colors

- You can set your own color for each scatter plot with the color or the c argument:

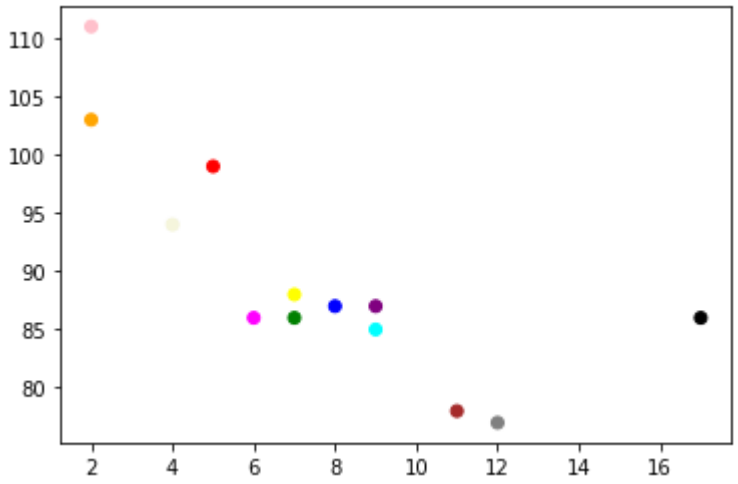
In [37]:
```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
5  y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
6  plt.scatter(x, y, color = 'hotpink')
7
8  x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
9  y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
10 plt.scatter(x, y, color = '#88c999')
11
12 plt.show()
13
```



Color Each Dot

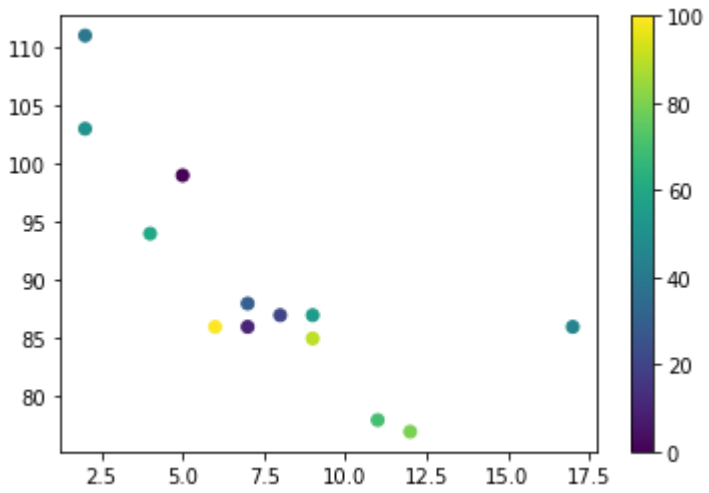
- You can even set a specific color for each dot by using an array of colors as value for the c argument:

In [38]:
```
import matplotlib.pyplot as plt
import numpy as np

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])
colors = np.array(["red","green","blue","yellow","pink","black","orange","pu

plt.scatter(x, y, c=colors)

plt.show()
```



- You can include the colormap in the drawing by including the plt.colorbar() statement:

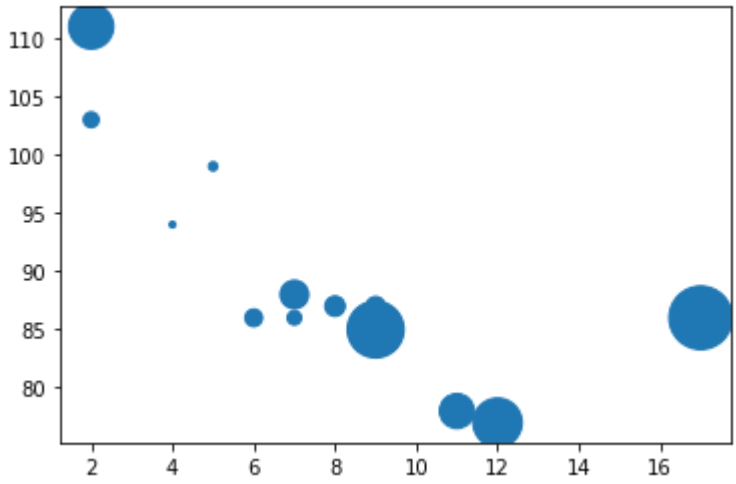
In [39]:
```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
5  y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
6  colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
7
8  plt.scatter(x, y, c=colors, cmap='viridis')
9
10 plt.colorbar()
11
12 plt.show()
```



Size

- You can change the size of the dots with the s argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

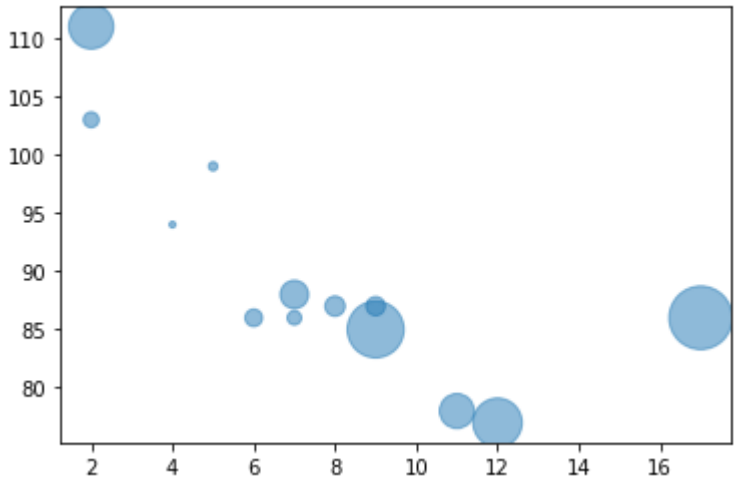
In [40]:
```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
5  y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
6  sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
7
8  plt.scatter(x, y, s=sizes)
9
10 plt.show()
```



Alpha

- You can adjust the transparency of the dots with the alpha argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

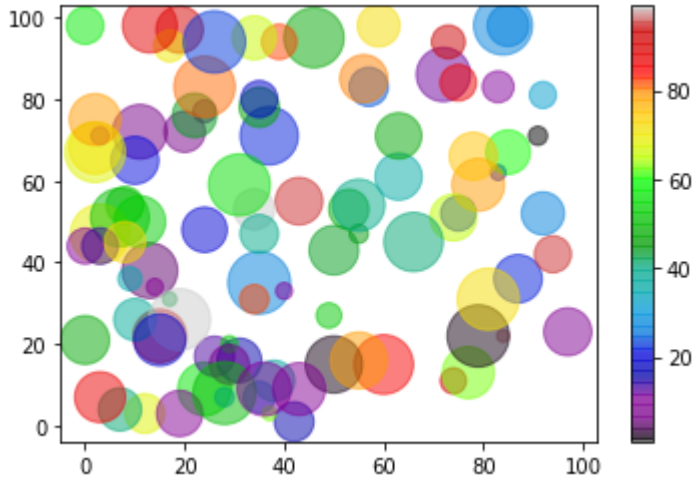
In [41]:
```
import matplotlib.pyplot as plt
import numpy as np

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])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```



Combine Color Size and Alpha

- You can combine a colormap with different sizes on the dots. This is best visualized if the dots are transparent:

In [42]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```

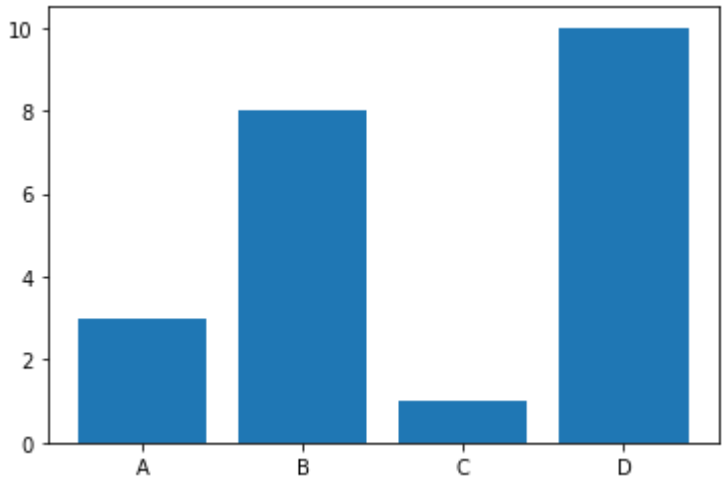


Matplotlib Bars

Creating Bars

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

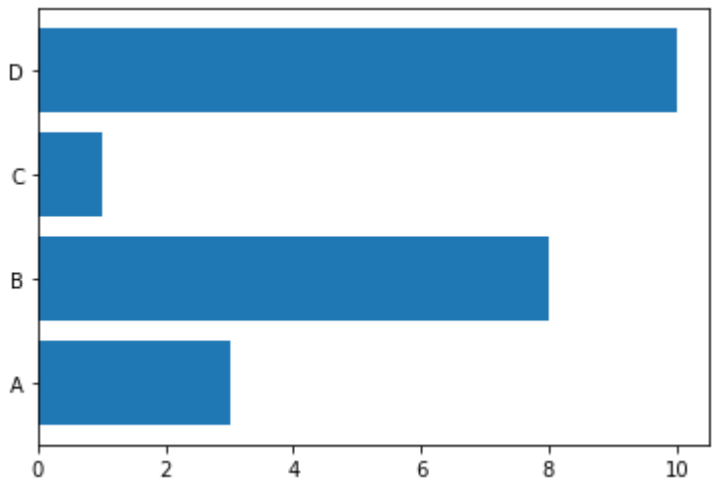
In [43]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.array(["A", "B", "C", "D"])
5 y = np.array([3, 8, 1, 10])
6
7 plt.bar(x,y)
8 plt.show()
```



Horizontal Bars

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

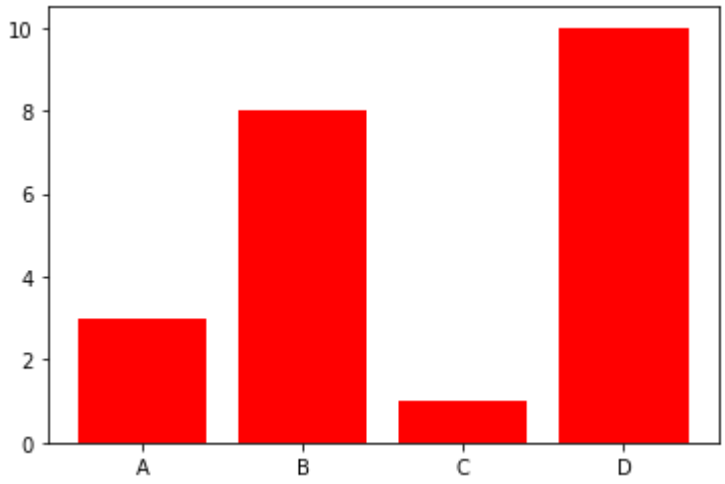
In [1]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.array(["A", "B", "C", "D"])
5 y = np.array([3, 8, 1, 10])
6
7 plt.barh(x, y)
8 plt.show()
```



Bar Color

- The bar() and barh() takes the keyword argument color to set the color of the bars:

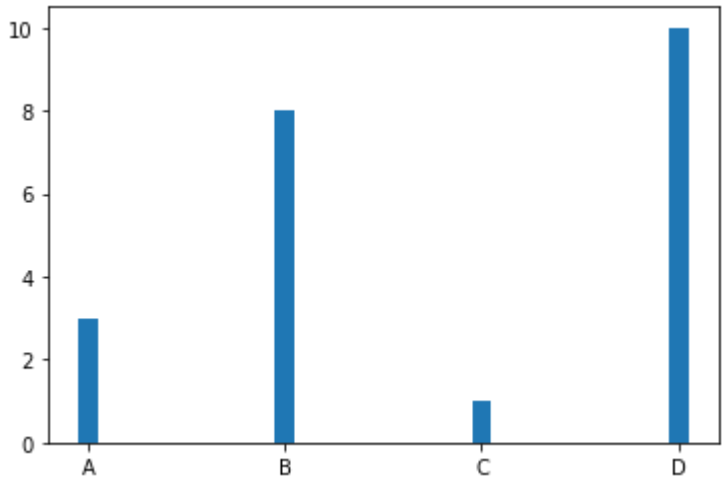
In [2]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Bar Width

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

In [3]:
```
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()
```

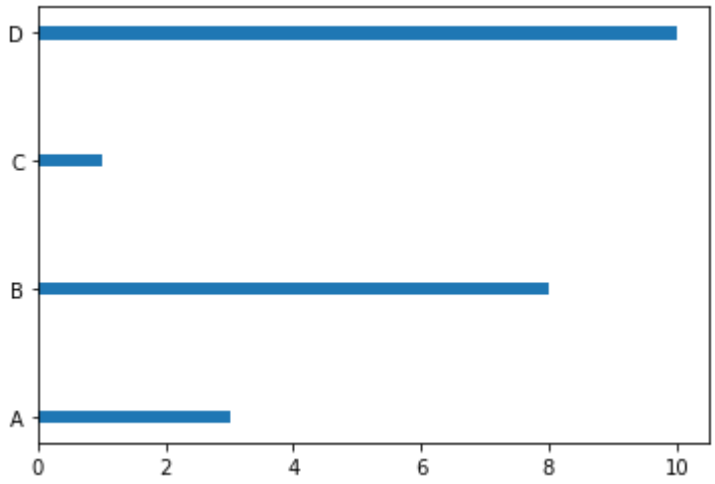


The default width value is 0.8

Bar Height

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

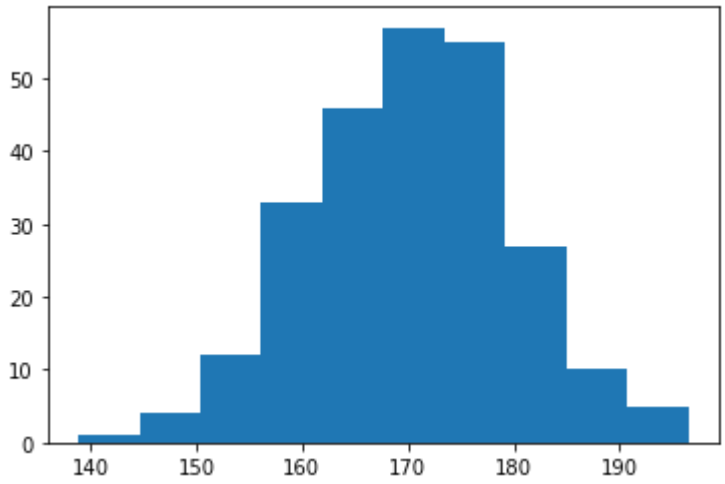
In [4]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.array(["A", "B", "C", "D"])
5 y = np.array([3, 8, 1, 10])
6
7 plt.barh(x, y, height = 0.1)
8 plt.show()
```



Matplotlib Histograms

- A histogram is a graph showing frequency distributions.
- It is a graph showing the number of observations within each given interval.

In [5]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.random.normal(170, 10, 250)
5
6 plt.hist(x)
7 plt.show()
```



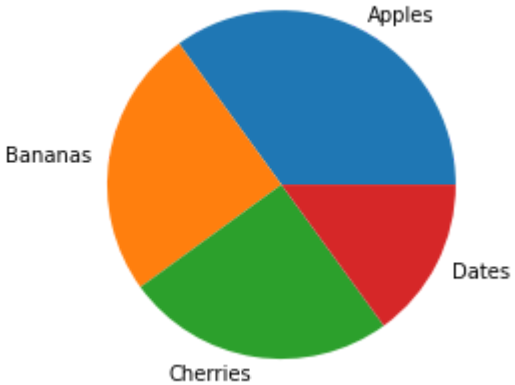
Matplotlib Pie Charts

In [6]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 y = np.array([35, 25, 25, 15])
5
6 plt.pie(y)
7 plt.show()
```

Labels

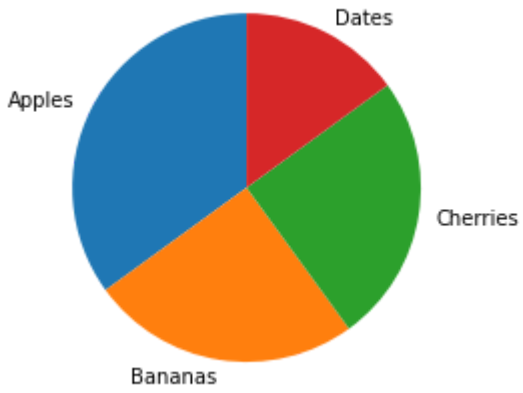
- Add labels to the pie chart with the label parameter.
- The label parameter must be an array with one label for each wedge:

In [7]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 y = np.array([35, 25, 25, 15])
5 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
6
7 plt.pie(y, labels = mylabels)
8 plt.show()
```



- Start the first wedge at 90 degrees:

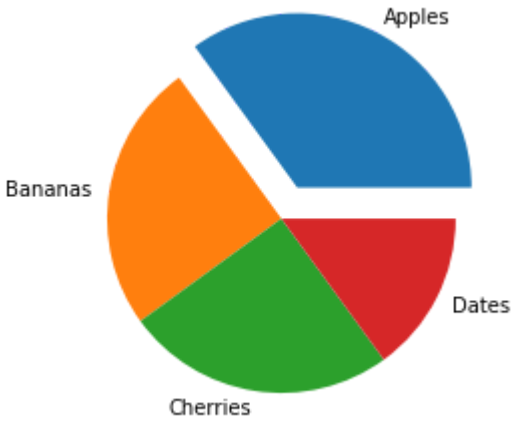
In [8]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



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:

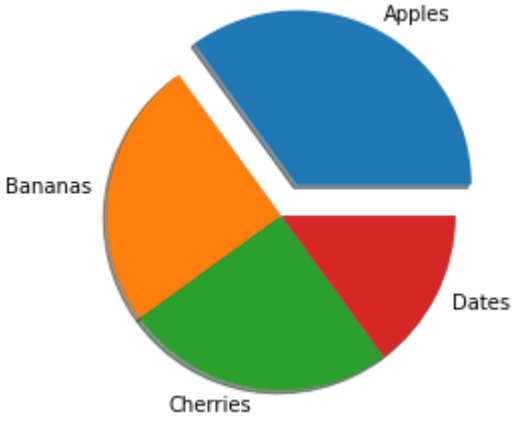
In [9]:
```
import matplotlib.pyplot as plt
import numpy as np

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()
```



Shadow

- Add a shadow to the pie chart by setting the shadows parameter to True:

In [10]:
```
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, shadow = True)
plt.show()
```



Colors

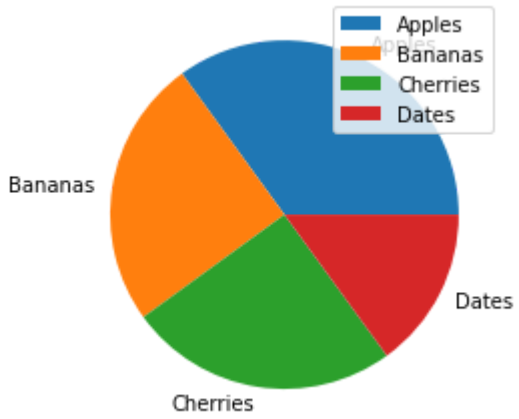
- You can set the color of each wedge with the colors parameter.
- The colors parameter, if specified, must be an array with one value for each wedge:

In [11]:
```
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()
```

Legend

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

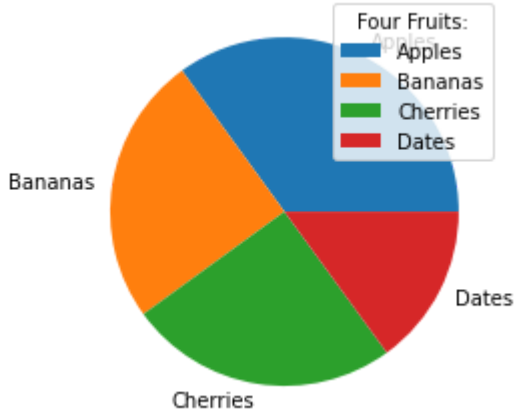
In [12]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Legend With Header

- To add a header to the legend, add the title parameter to the legend function.

In [13]:
```
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()
```



In [ ]:
```
1
```