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.

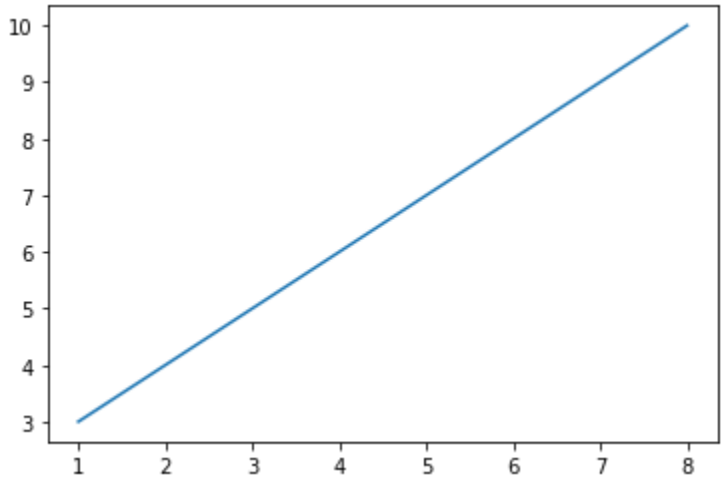
The function takes parameters for specifying points in the diagram.

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.

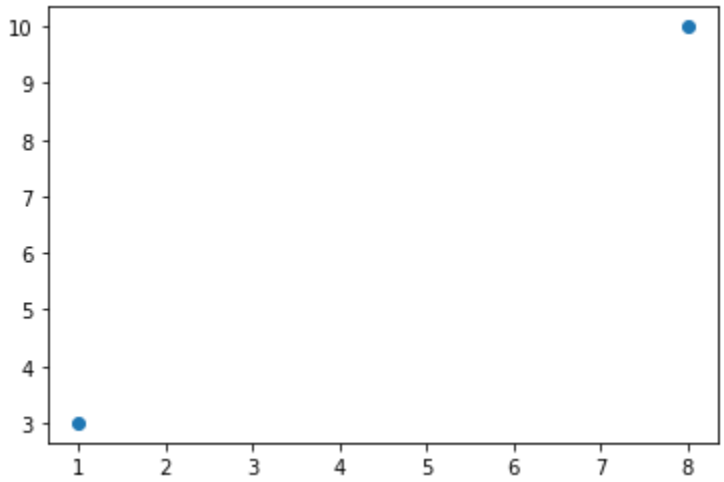
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Plotting Without Line

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

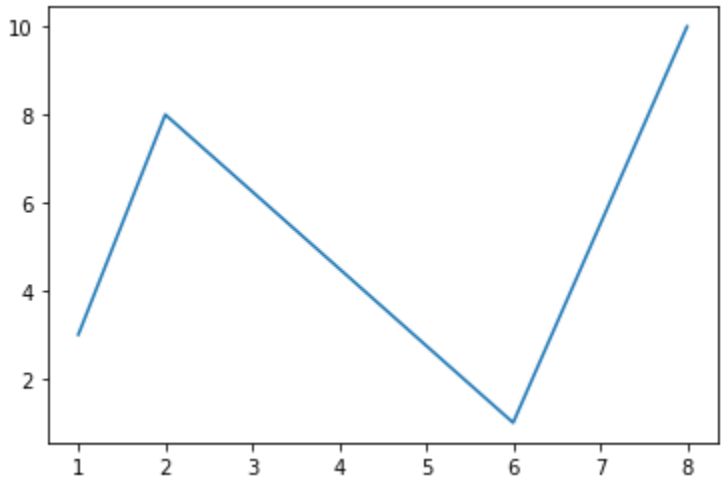
In [2]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(xpoints, ypoints, 'o')
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 [3]:
```
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()
```

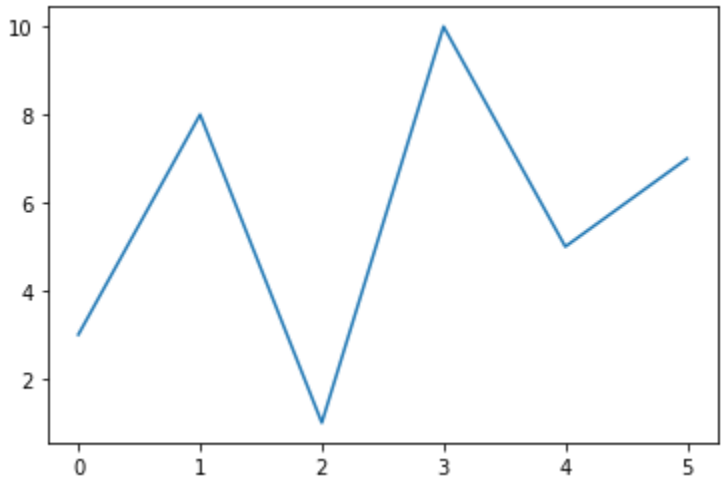


Default X-Points

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.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

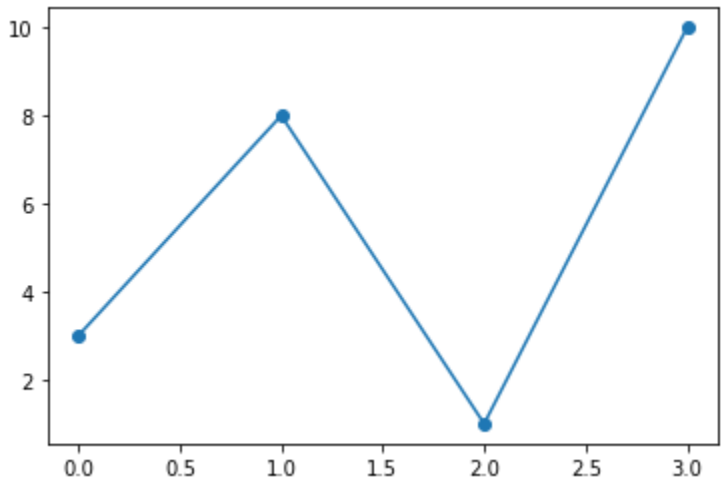
In [4]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Markers

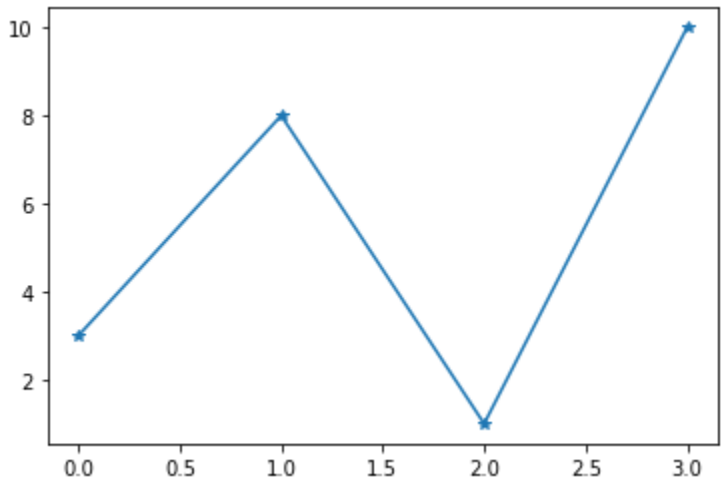
You can use the keyword argument marker to emphasize each point with a specified marker:

In [5]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



In [6]:
```
1 plt.plot(ypoints, marker = '*')
```

Out[6]: [<matplotlib.lines.Line2D at 0x1db8c785be0>]



Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
'|'	Vline
'_'	Hline

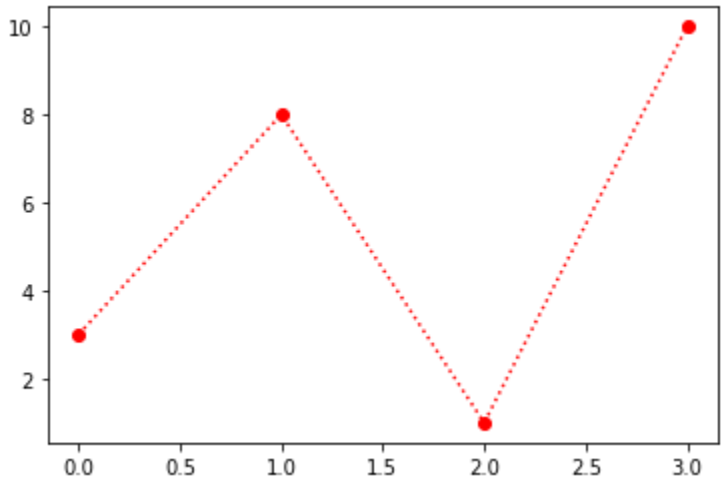
Format Strings fmt

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 [9]:
```
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, 'o:r')
7 plt.show()
```



Line Reference

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

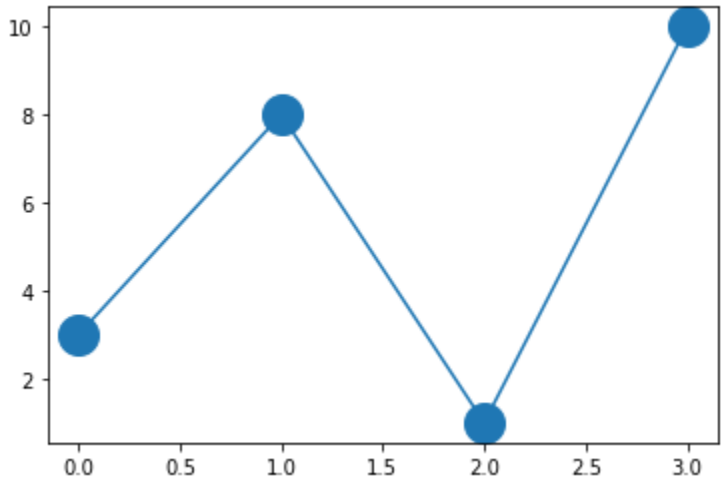
Color Reference

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

Marker Size

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

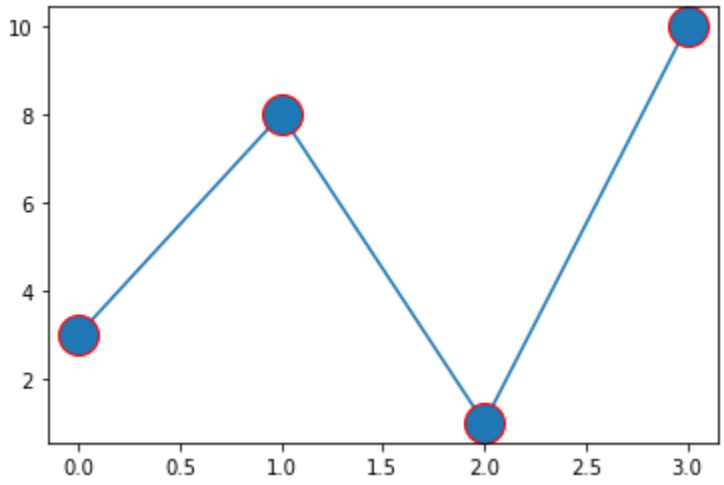
In [10]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 ypoints = np.array([3, 8, 1, 10])
5
6 plt.plot(ypoints, marker = 'o', ms = 20)
7 plt.show()
```



Marker Color

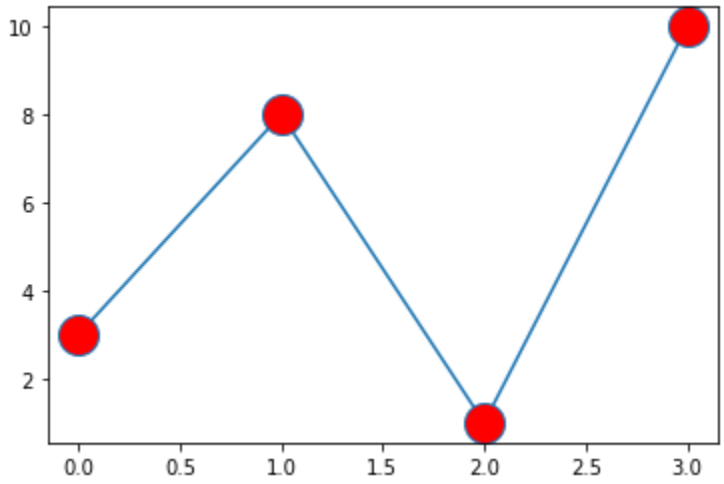
You can use the keyword argument markeredgecolor or the shorter mec to set the color of the edge of the markers:

In [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, mec = 'r')
7 plt.show()
```



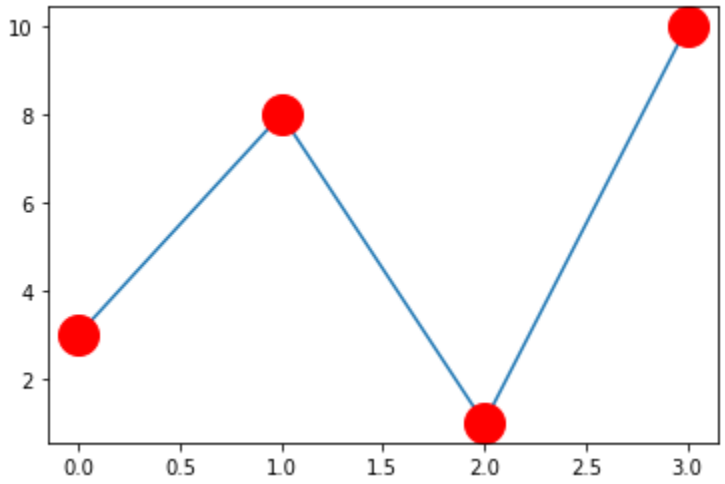
You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers

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, mfc = 'r')
7 plt.show()
```



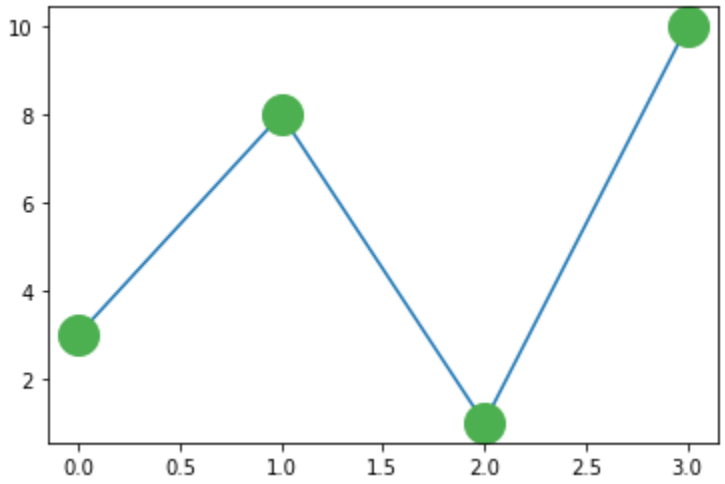
Use both the mec and mfc arguments to color the entire marker:

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, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
7 plt.show()
```



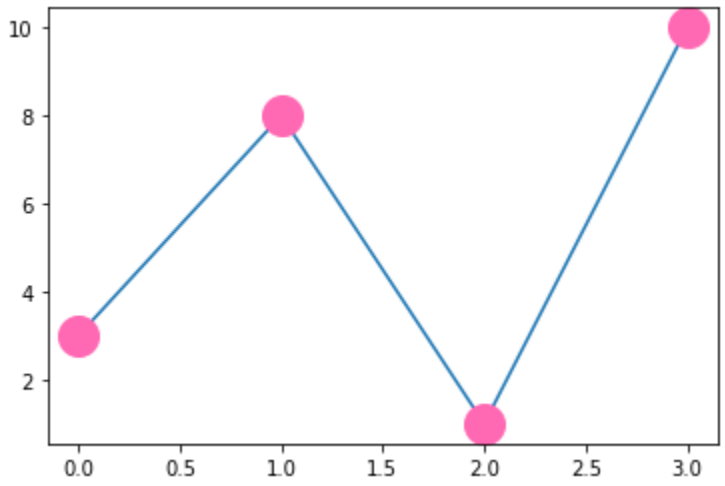
In [14]:
```
1 plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
```

Out[14]: [<matplotlib.lines.Line2D at 0x1db8c956970>]



In [15]:
```
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink' )
```

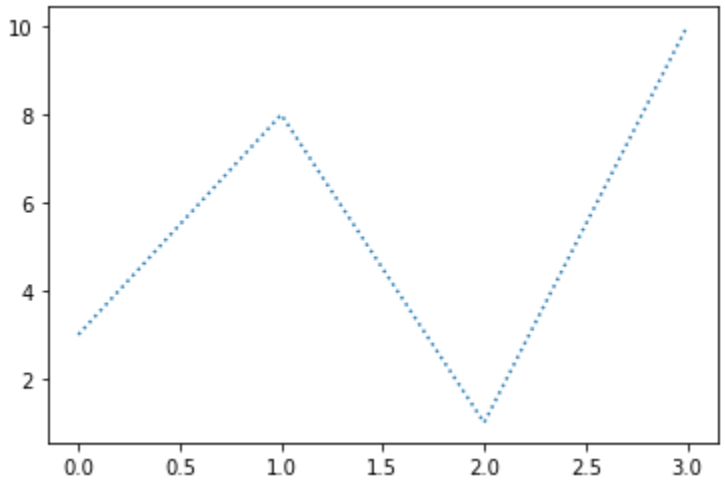
Out[15]:
```
[<matplotlib.lines.Line2D at 0x1db8c8acbb0>]
```



Linestyle

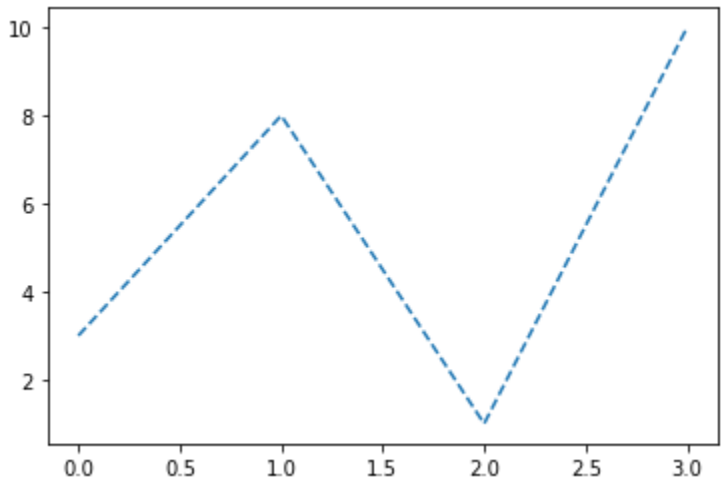
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

In [16]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



In [17]:
```
1 plt.plot(ypoints, linestyle = 'dashed')
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x1db8c6f6130>]
```



Shorter Syntax

The line style can be written in a shorter syntax:

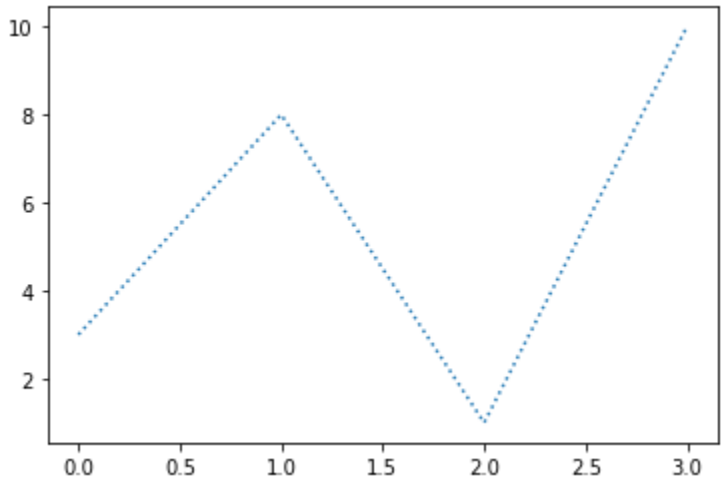
linestyle can be written as ls.

dotted can be written as :.

dashed can be written as --.

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

Out[18]:
```
[<matplotlib.lines.Line2D at 0x1db8c604160>]
```



Line Styles

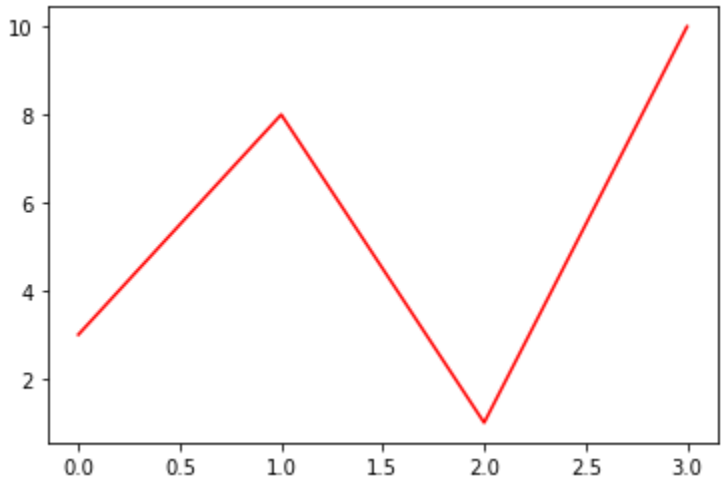
You can choose any of these styles:

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

Line Color

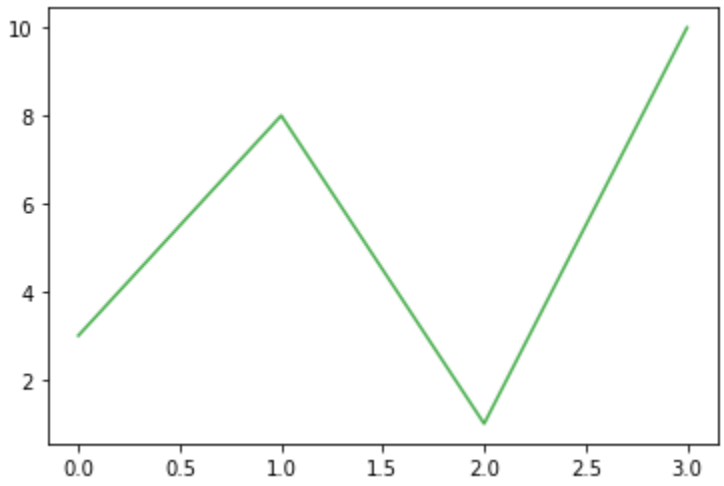
You can use the keyword argument `color` or the shorter `c` to set the color of the line:

In [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, color = 'r')
7 plt.show()
```



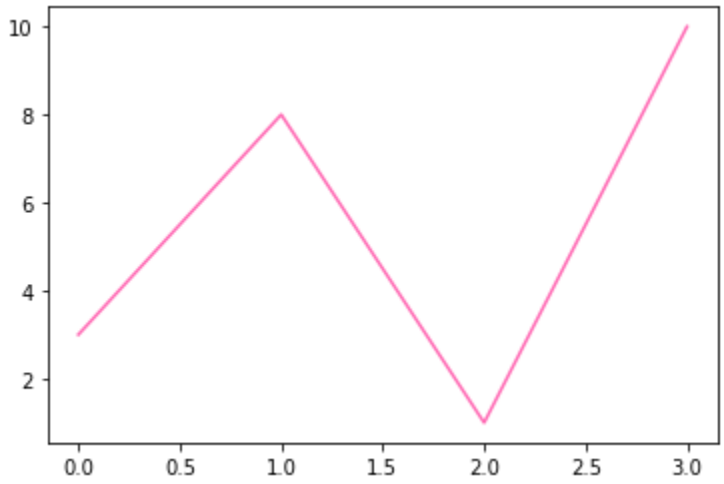
In [20]:
```
1 plt.plot(ypoints, c = '#4CAF50')
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x1db8c6d19a0>]
```



In [21]:
```
1 plt.plot(ypoints, c = 'hotpink')
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x1db8c66c2e0>]
```

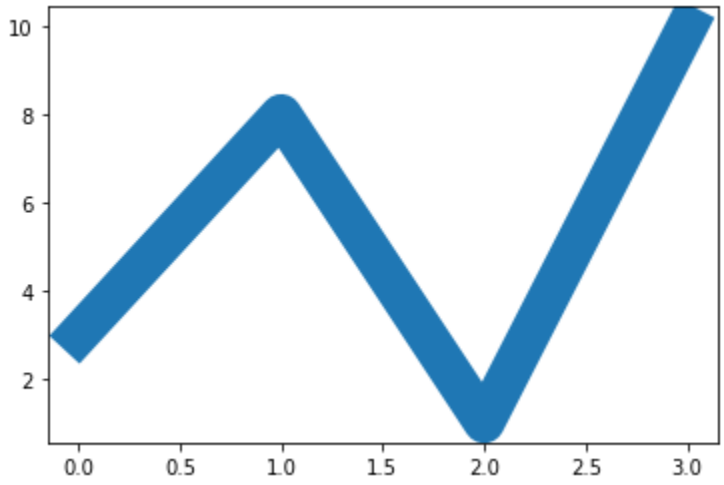


Line Width

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:

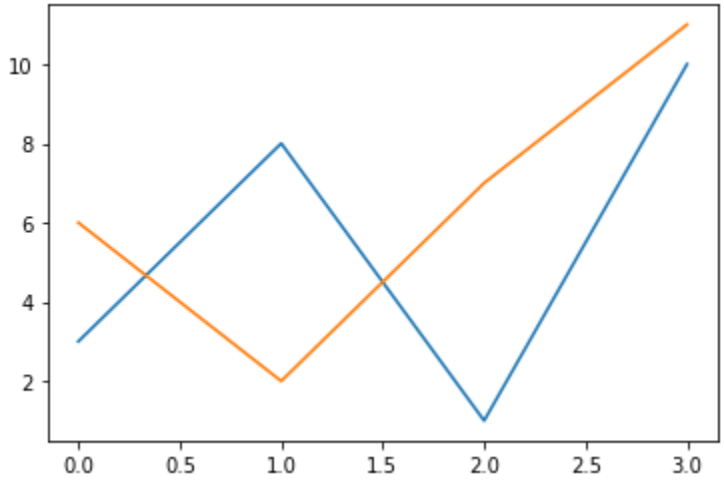
In [22]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```



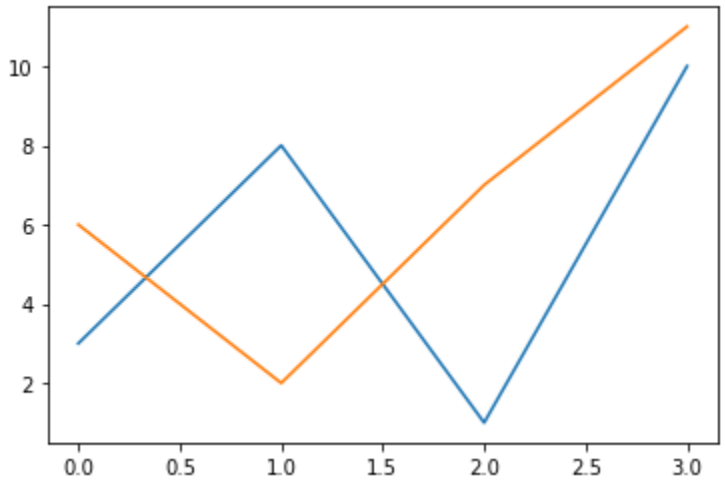
Multiple Lines

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

In [23]:
```
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()
```



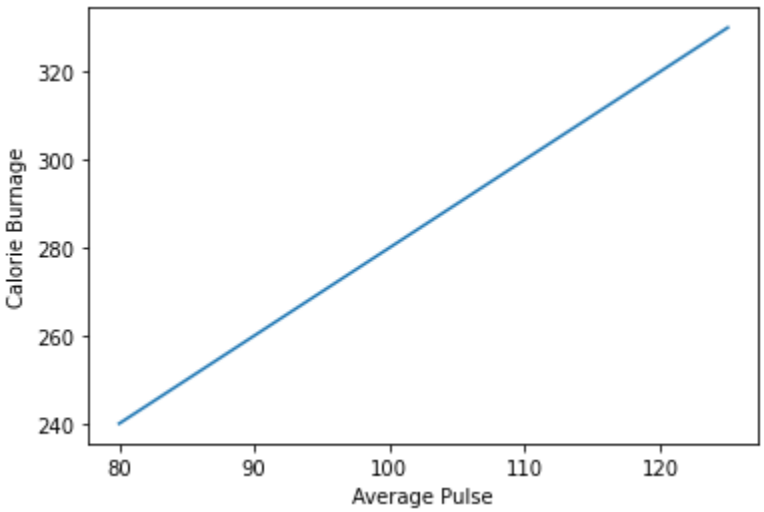
In [24]:
```
import matplotlib.pyplot as plt
import numpy as np

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()
```



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.

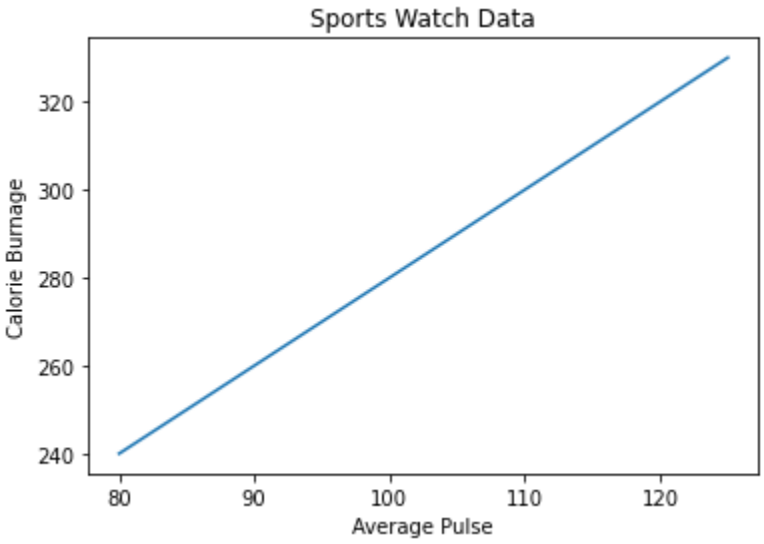
In [25]:
```
import numpy as np
import matplotlib.pyplot as plt

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.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 [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.plot(x, y)

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

plt.show()
```

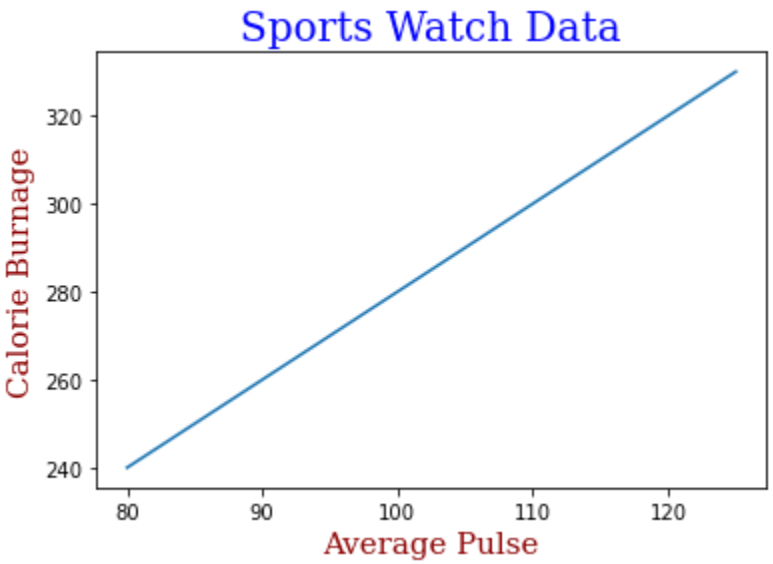


Set Font Properties for Title and Labels

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

In [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])

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()
```

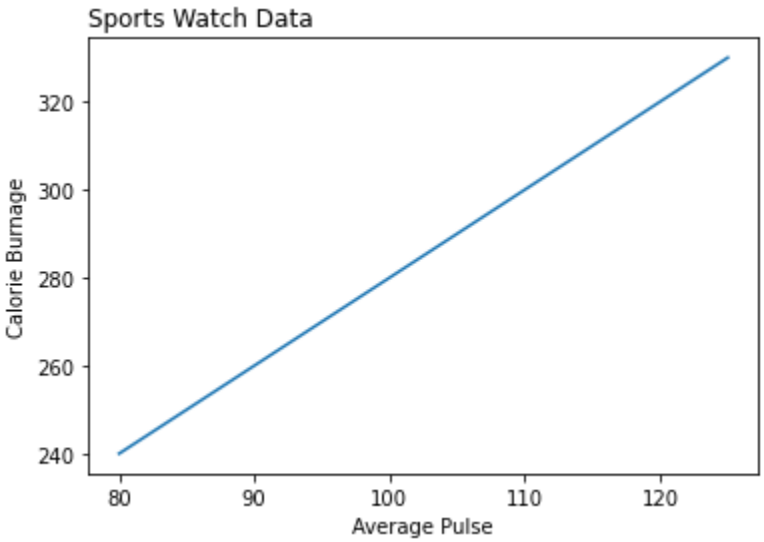


Position the Title

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

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

In [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", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

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

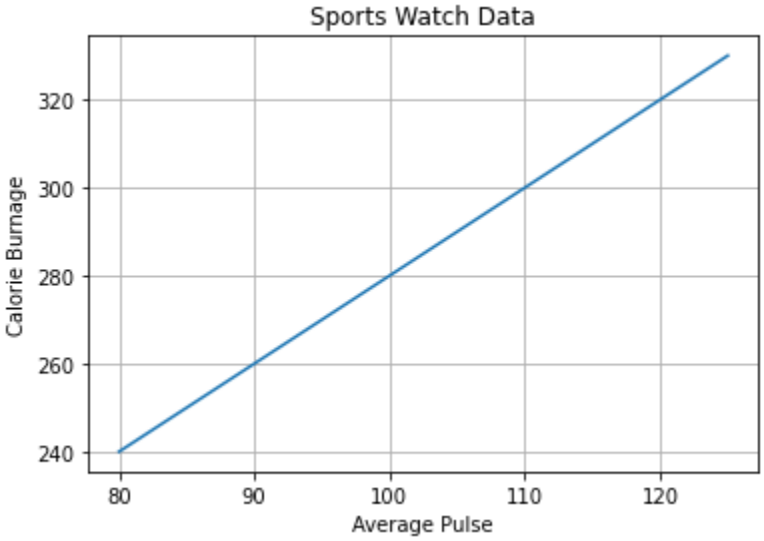


Add Grid Lines to a Plot

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

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()

plt.show()
```



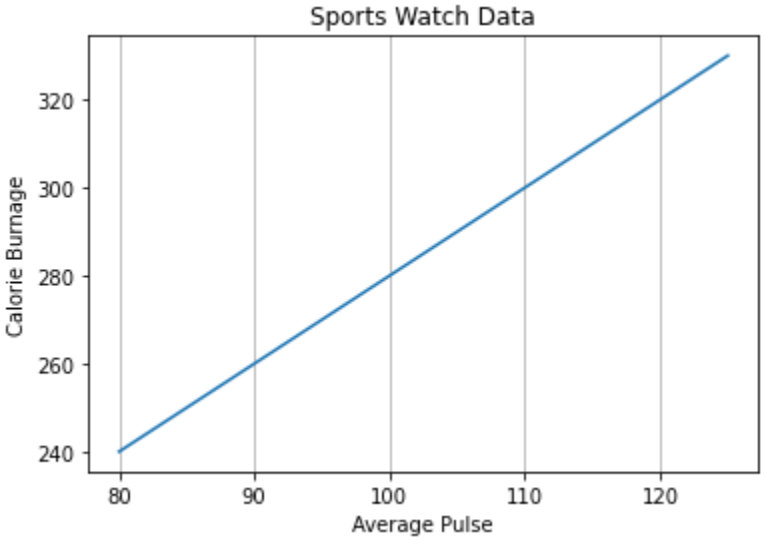
Specify Which Grid Lines to Display

You can use the axis parameter in the grid() function to specify which grid lines to display.

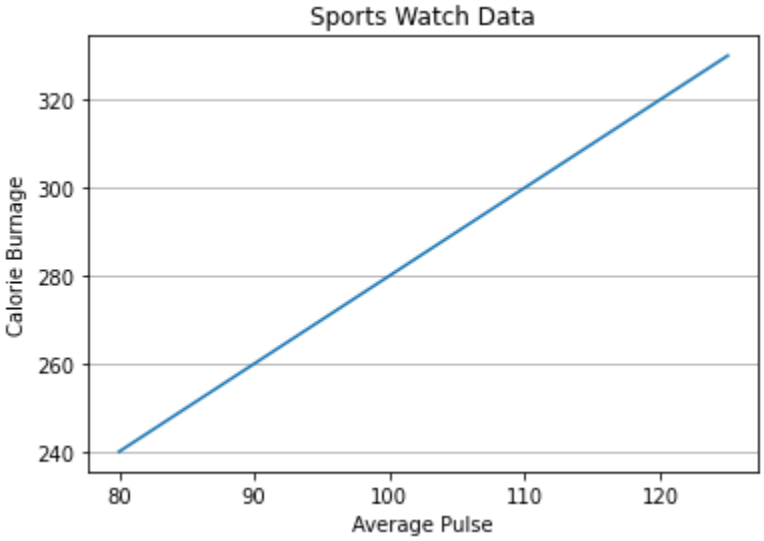
Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

In [31]:

```
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 [32]:
```
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()
```

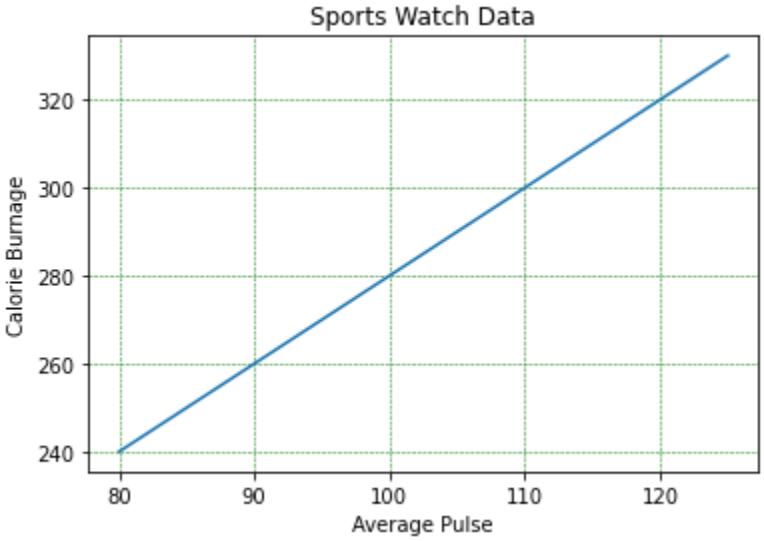


Set Line Properties for the Grid

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

In [33]:

```
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()
```

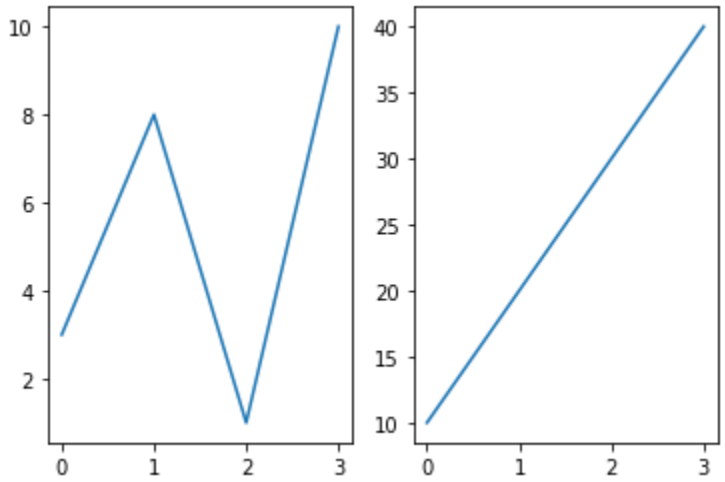


Display Multiple Plots

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

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)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



The subplot() Function

The `subplot()` function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the *first* and *second* argument.

The third argument represents the index of the current plot.

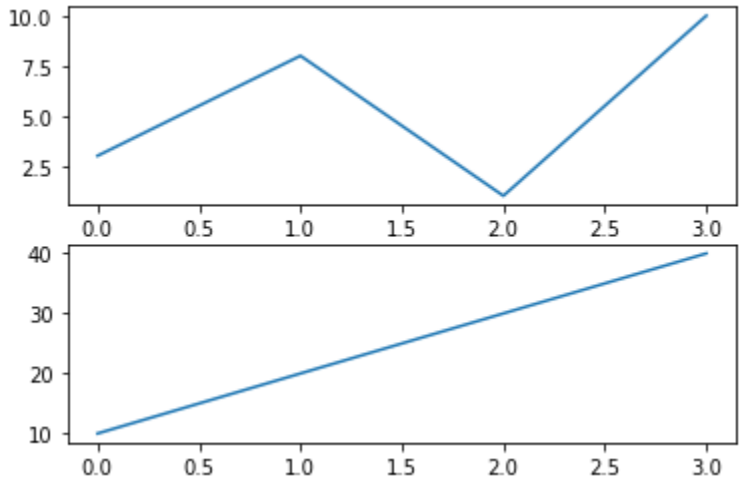
```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

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

In [35]:

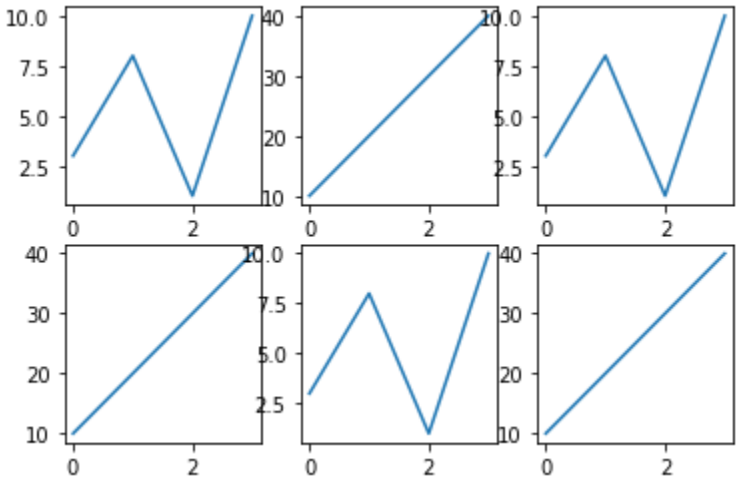
```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



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

In [36]:

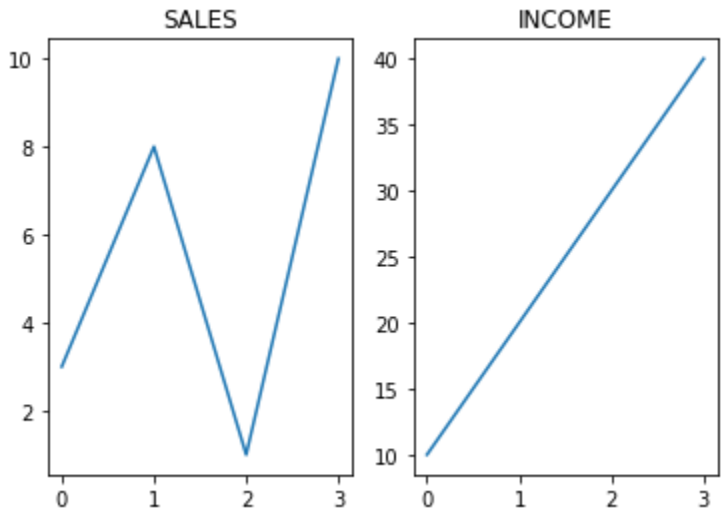
```
import matplotlib.pyplot as plt
import numpy as np

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 [37]:
```
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()
```

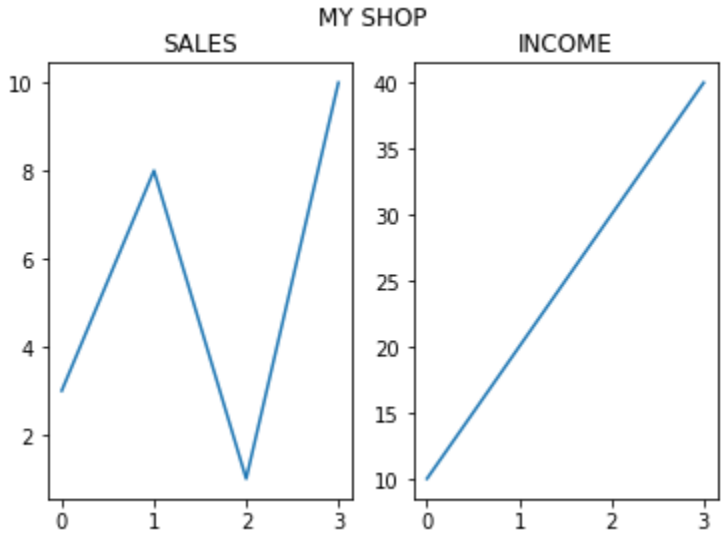


Super Title

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

In [39]:

```
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()
```

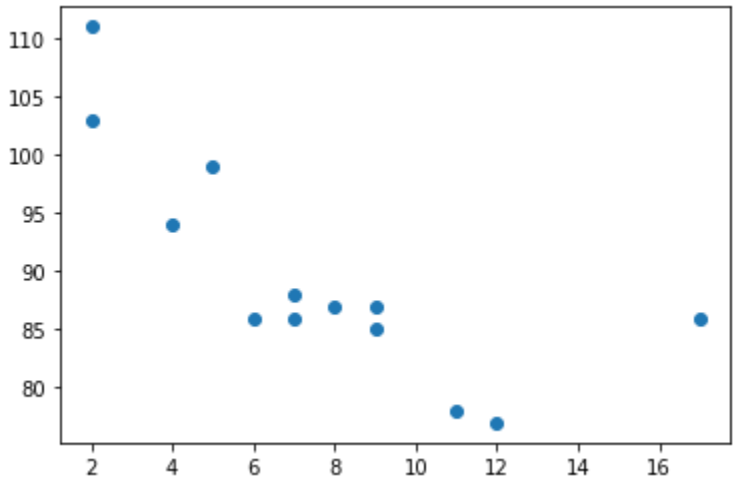


Creating Scatter Plots

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 [40]:
```
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.

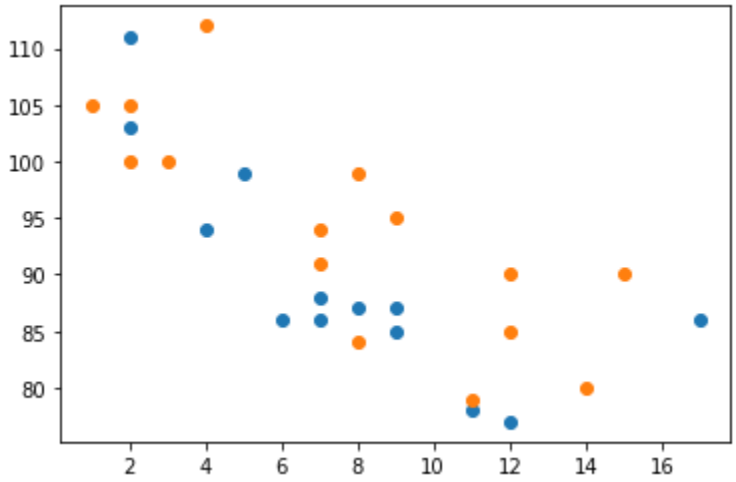
Are there any relationships between the observations?

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 [41]:
```
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()
```



Note: The two plots are plotted with two different colors, by default blue and orange, you will learn how to change colors later in this chapter.

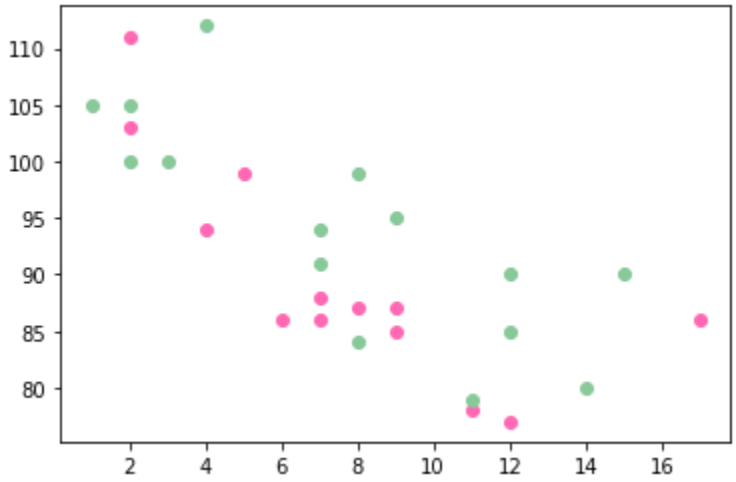
By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

Colors

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

In [42]:

```
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, color = 'hotpink')

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, color = '#88c999')

plt.show()

```

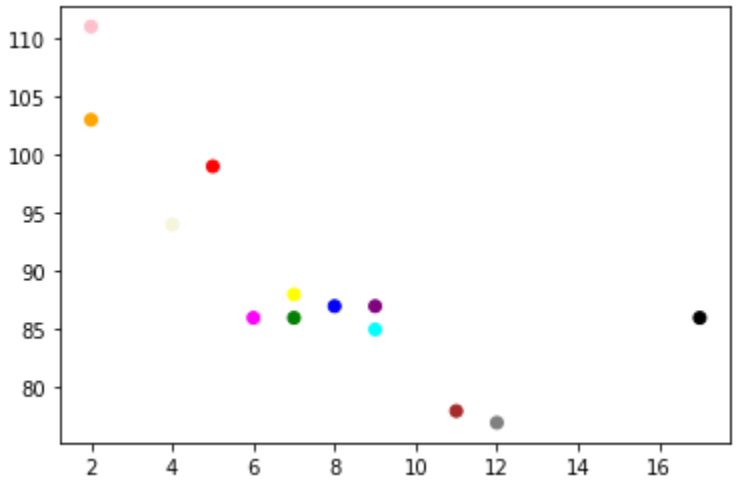


Color Each Dot

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

Note: You cannot use the color argument for this, only the c argument.

In [43]:
```
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","purple","b

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

plt.show()
```



ColorMap

The Matplotlib module has a number of available colormaps.

A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.

Here is an example of a colormap:

This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.

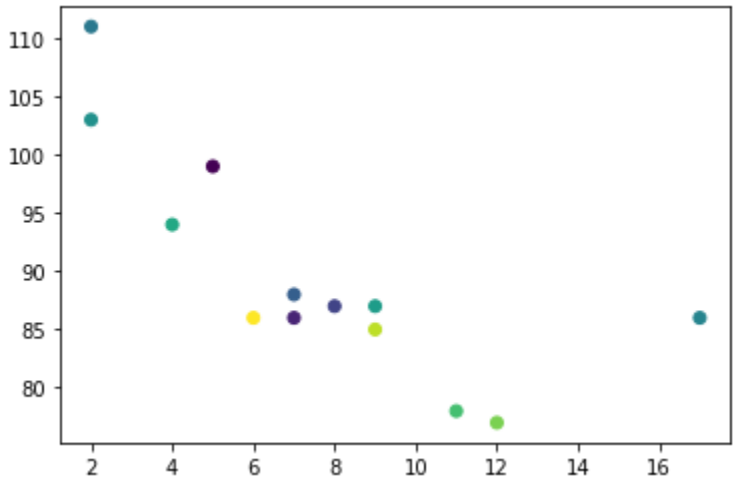
How to Use the ColorMap

You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

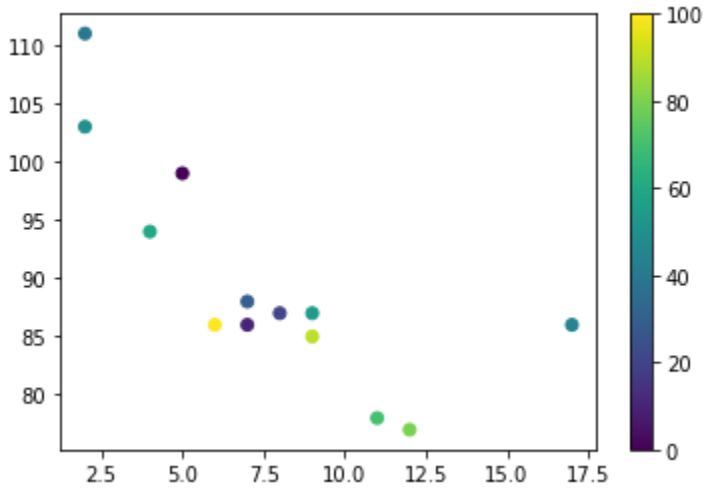
In [44]:

```
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([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```



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

In [45]:
```
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([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```

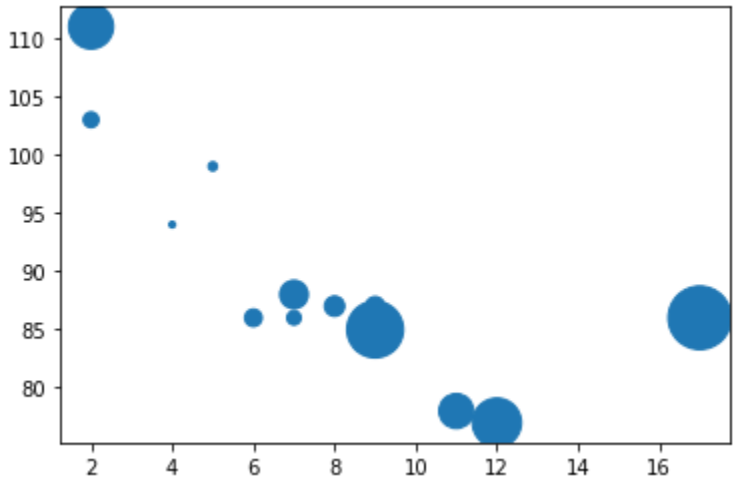


Size

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 [46]:
```
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)

plt.show()
```

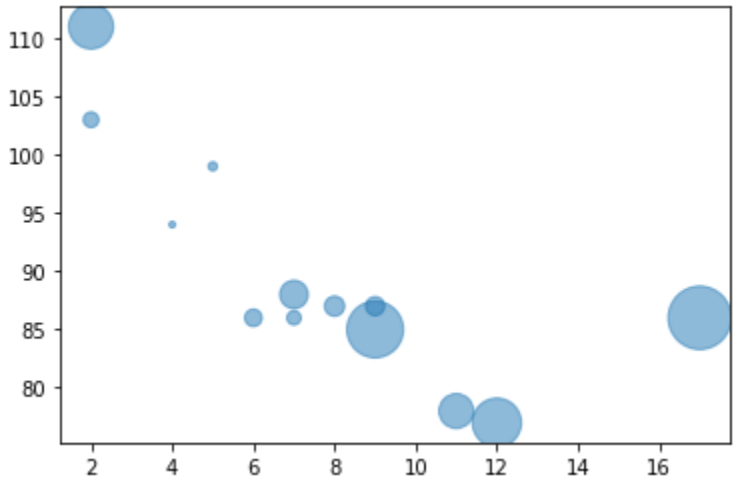


Alpha

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 [47]:
```
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()
```

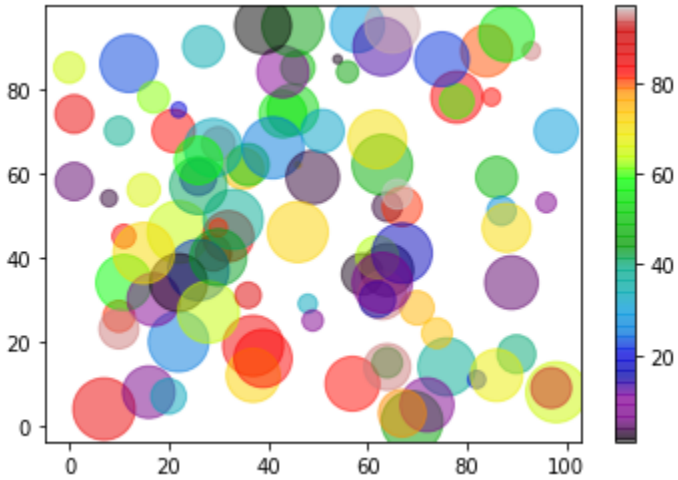


Combine Color Size and Alpha

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

In [48]:

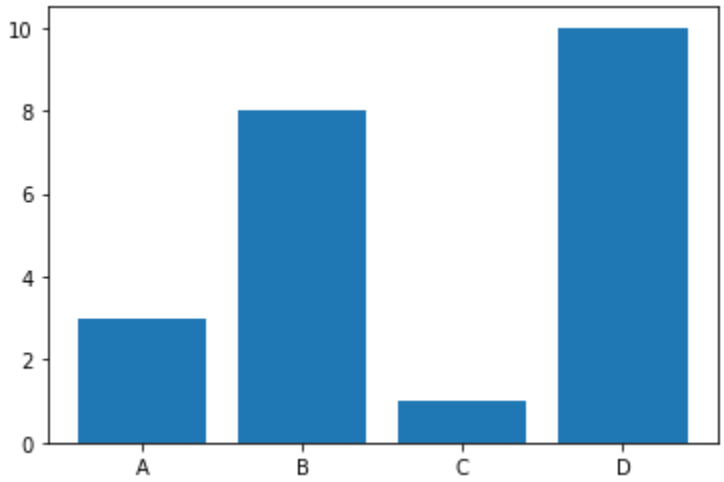
```
import matplotlib.pyplot as plt
import numpy as np

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()
```



Creating Bars

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

In [50]:
```
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()
```

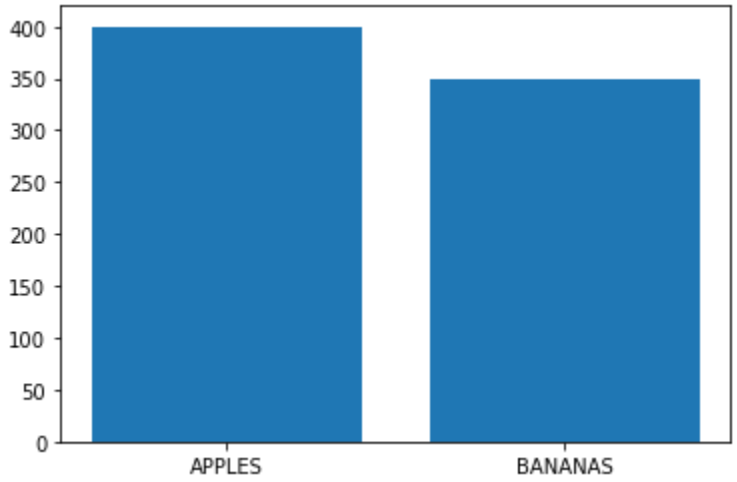


The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the first and second argument as arrays.

In [51]:
```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```

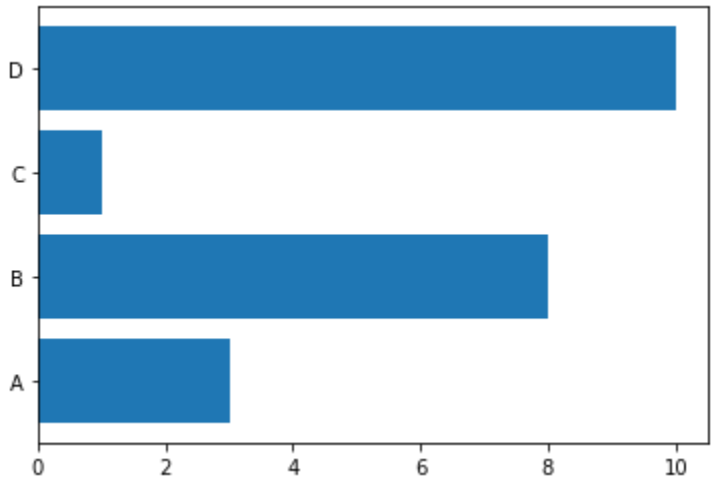
Out[51]: <BarContainer object of 2 artists>



Horizontal Bars

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

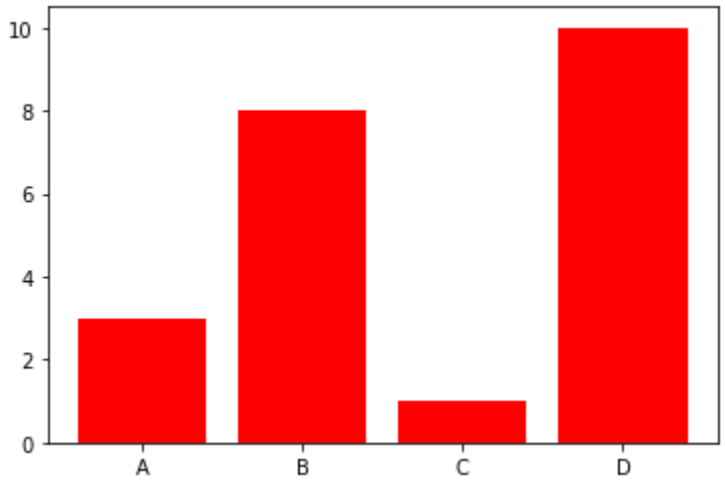
In [52]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



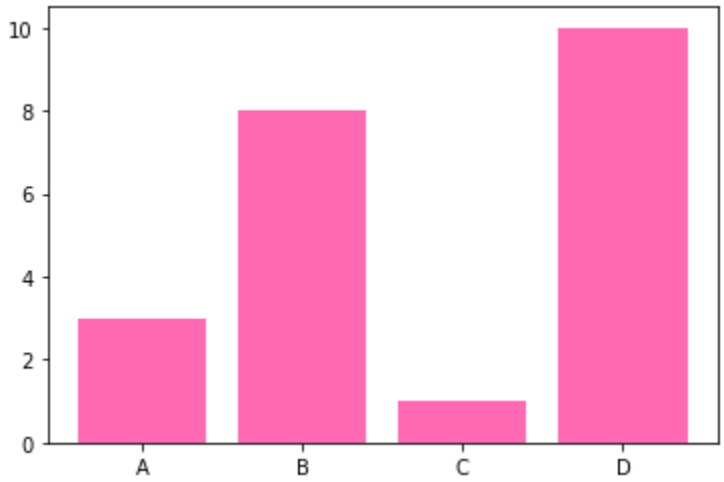
Bar Color

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

In [54]:
```
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()
```



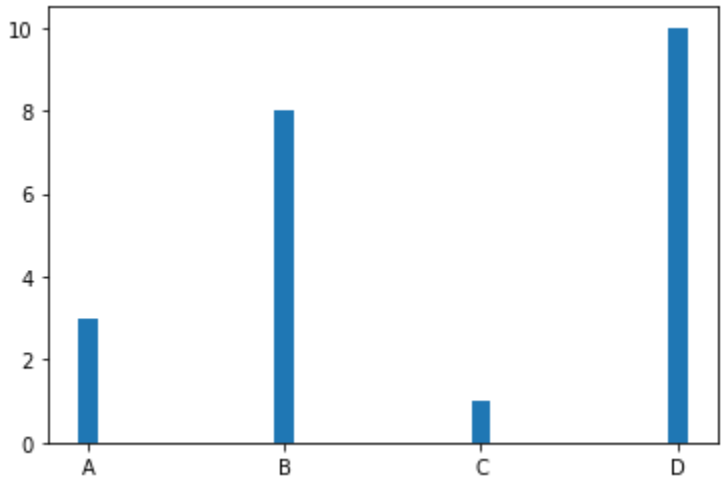
In [55]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Bar Width

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

In [56]:
```
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()
```



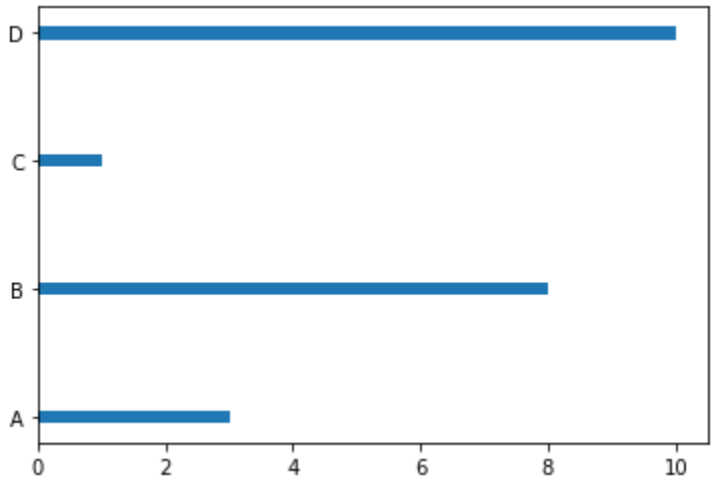
The default width value is 0.8

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

Bar Height

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

In [58]:
```
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()
```



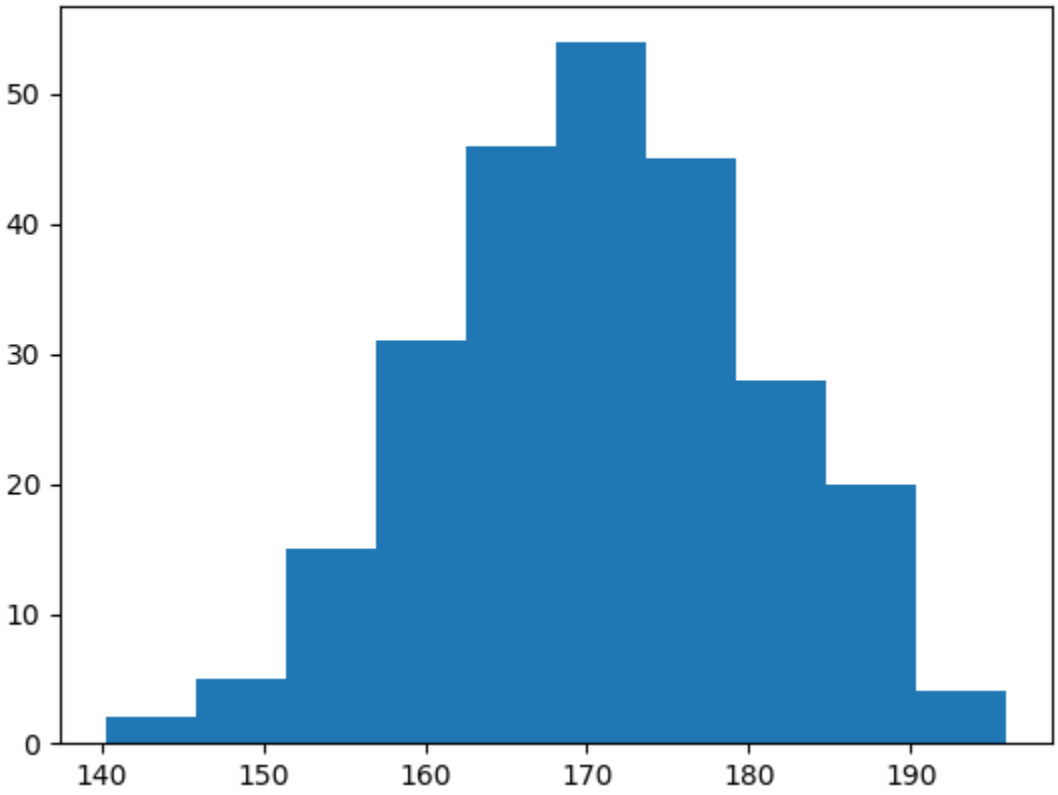
The default height value is 0.8

Histogram

A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



You can read from the histogram that there are approximately:

2 people from 140 to 145cm

5 people from 145 to 150cm

15 people from 151 to 156cm

31 people from 157 to 162cm

46 people from 163 to 168cm

53 people from 168 to 173cm

45 people from 173 to 178cm

28 people from 179 to 184cm

21 people from 185 to 190cm

4 people from 190 to 195cm

Create Histogram

In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

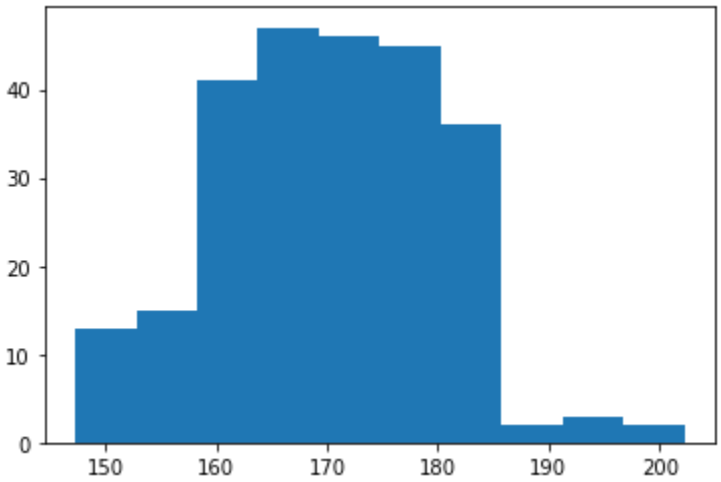
For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10

In [59]:

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```

```
[171.69702763 175.28472222 174.65448623 158.36283716 161.8672084
 169.62581756 170.17646978 167.41331991 174.63097785 157.2818272
 164.28440836 150.49731385 180.36683822 169.11165023 171.77829392
 170.6569192  188.22949679 182.68124533 178.00367579 175.49445674
 165.23493468 164.56875856 168.31041492 182.72930388 175.44671493
 165.29775609 161.53485037 167.03002145 159.56714047 170.24802025
 168.34479995 169.23714091 182.65651137 148.10980675 174.43607156
 156.03954306 169.50518009 175.68143113 169.30452336 166.31049568
 172.32392107 178.19022572 151.27011586 179.24810419 156.99271887
 164.96287465 167.0756313  174.00942535 178.67783492 176.10869065
 161.30149716 163.40707352 174.96867685 179.67535833 180.52513927
 167.08986488 169.96802942 154.75700023 165.93716613 160.09013287
 173.31437419 182.26667136 171.76515963 171.74251716 150.23274684
 159.17007765 164.88735194 165.41665915 179.76732043 170.2885376
 161.43264393 157.61391279 172.5805789  178.75062512 184.63121195
 176.81103729 158.98912063 179.76736927 168.49199423 170.18057973
 159.36365825 169.31788331 167.67623613 164.07307603 178.90013802
 187.54817768 135.4219617  176.67626916 173.90302949 171.16121544
 170.2392474  171.19824198 162.85377287 167.8717925  158.97893343
 177.82743313 174.90135824 171.32464828 184.39539614 176.10164219
 168.71175292 172.00041123 155.95286956 185.66417035 167.14766952
 163.78447067 167.56283093 163.67311631 180.17483311 161.50395215
 188.98675688 166.74939764 174.26386391 167.00958571 171.60287857
 183.93110467 156.87839728 171.97042678 163.53253835 157.96832526
 176.38556741 172.41509821 166.79997334 170.69616125 161.70114883
 180.29063666 157.54857993 156.27940989 182.42220489 166.60006312
 180.57215145 184.68504248 169.80241534 175.47800728 184.54899785
 160.60553831 161.44002302 171.64269816 163.30101178 176.3952201
 162.70359072 172.90672481 162.91623007 170.91703308 174.62146588
 157.63384923 170.9495541  154.68814672 180.19800172 170.38611579
 165.67502072 149.19956099 167.54956111 158.66991929 177.87609642
 172.23726516 170.12510486 184.14427481 167.78370486 185.75893266
 185.18491695 148.08376896 184.61851417 168.65967066 168.48854986
 171.44859909 194.4948729  153.87112341 187.36351322 170.39610037
 167.48790648 168.57950264 161.31927506 164.65965615 179.72347357
 178.48473706 168.02841745 142.24847881 169.43796843 172.44850727
 142.96494042 178.7309139  160.74623815 161.10320377 179.83392409
 153.9979705  164.67422265 167.05026094 175.77772016 171.98215642
 183.16372463 173.13419624 180.92092013 182.12546398 180.10299876
 161.5375926  151.45230796 184.15259261 151.26528622 145.52681747
 166.20328647 168.58327334 162.97251507 168.55490035 167.48989265
 164.88126636 171.49847929 171.4896219  195.60079537 181.05569838
 169.30822957 175.15605974 190.72792461 181.70968517 168.06106299
 163.00689192 189.8571325  171.06202593 164.79181355 166.69661665
 168.09584626 174.12157846 161.01721789 187.22282386 157.11747794
 159.9916817  156.93036657 157.11097415 158.86761937 176.46945416
 180.01729769 178.91345616 172.23719021 168.03168976 162.09986914
 161.13360187 185.0620213  180.26987878 159.62708397 168.94363424
 160.8982587  171.13924415 171.95193195 162.7314961  179.32346557
 164.07397378 173.95182915 182.51801821 163.56995957 164.15986327]
```

In [60]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



x : (n,) array or sequence of (n,) arrays

Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length.

bins : int or sequence or str, optional

If an integer is given, `bins + 1` bin edges are calculated and returned, consistent with `numpy.histogram`.

If `bins` is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, `bins` is returned unmodified.

All but the last (righthand-most) bin is half-open. In other words, if `bins` is:

```
[1, 2, 3, 4]
```

then the first bin is `[1, 2)` (including 1, but excluding 2) and the second `[2, 3)`. The last bin, however, is `[3, 4]`, which *includes* 4.

Unequally spaced bins are supported if *bins* is a sequence.

With Numpy 1.11 or newer, you can alternatively provide a string describing a binning strategy, such as 'auto', 'sturges', 'fd', 'doane', 'scott', 'rice' or 'sqrt', see `numpy.histogram`.

The default is taken from `rcParams["hist.bins"] = 10`.

align : {'left', 'mid', 'right'}, optional

Controls how the histogram is plotted.

- 'left': bars are centered on the left bin edges.
- 'mid': bars are centered between the bin edges.
- 'right': bars are centered on the right bin edges.

Default is 'mid'

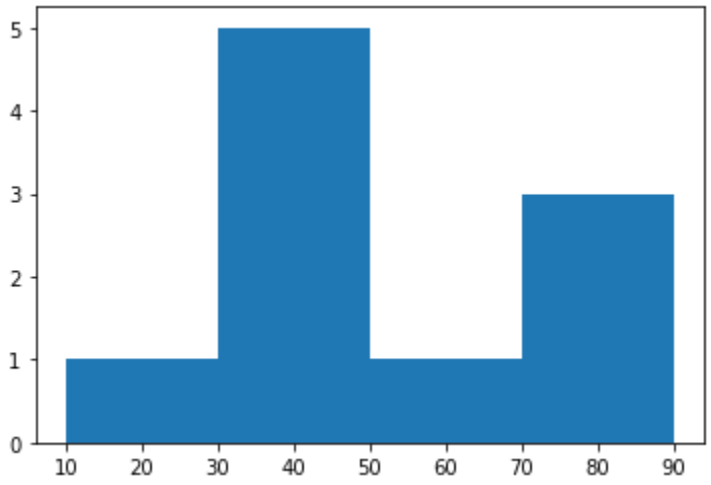
orientation : {'horizontal', 'vertical'}, optional

If 'horizontal', `barh` will be used for bar-type histograms and the *bottom* kwarg will be the left edges.

In [76]:

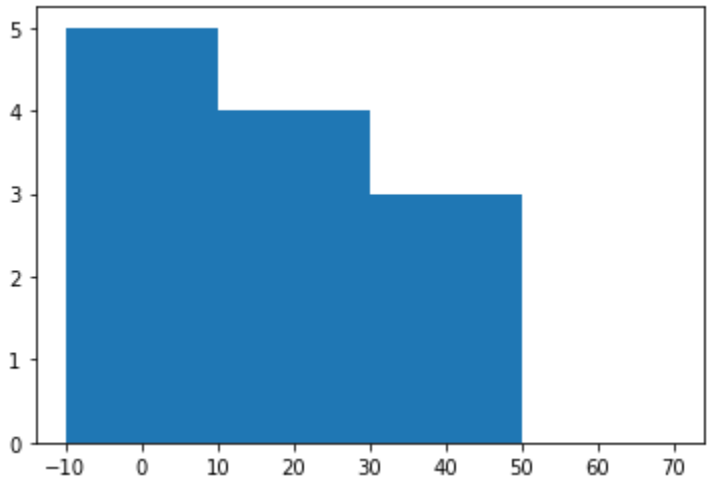
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(0, 100,12 )
print(x)
plt.hist(x,bins=[0,20,40,60,80.100],align="right",orientation="vertical")
plt.show()
```

```
[10 36 97 22 87 73 48 28 66 70 27 26]
```



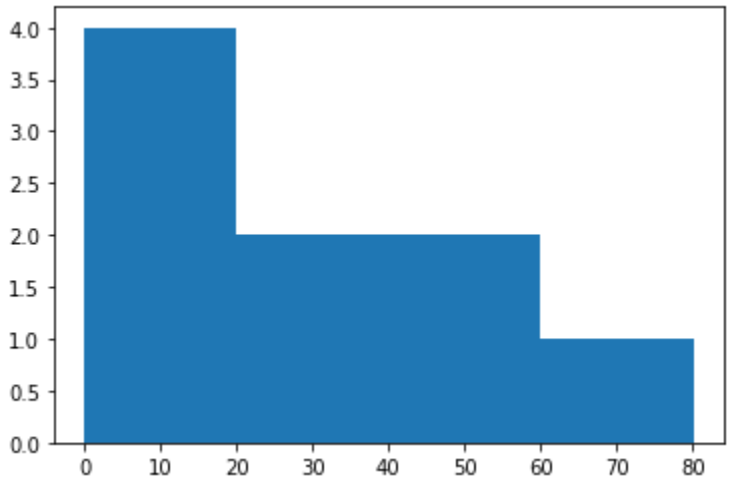
In [77]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.random.randint(0, 100,12 )
5 print(x)
6 plt.hist(x,bins=[0,20,40,60,80.100],align="left",orientation="vertical")
7 plt.show()
```

```
[13 10 34 40 22 40 10 13 37  6 45 20]
```



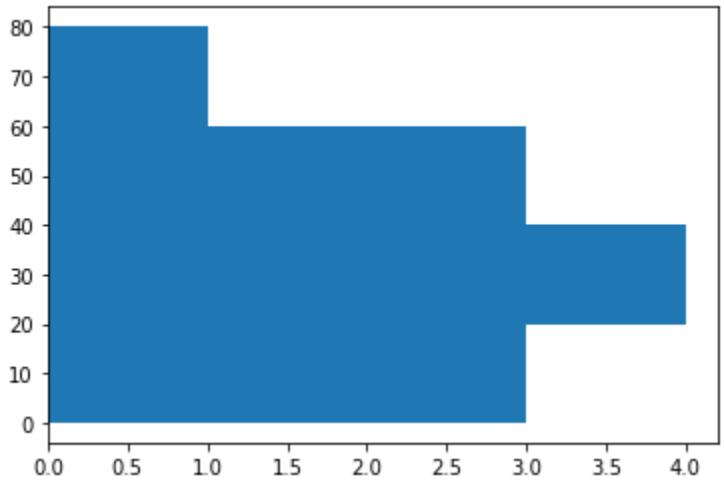
In [78]:
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.random.randint(0, 100,12 )
5 print(x)
6 plt.hist(x,bins=[0,20,40,60,80.100],align="mid",orientation="vertical")
7 plt.show()
```

```
[43  6 93 28 91  6 11 83 41 78 12 27]
```



In [80]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(0, 100,12 )
print(x)
plt.hist(x,bins=[0,20,40,60,80.100],orientation="horizontal")
plt.show()
```

```
[59 95 48 45 34 13 21 21 22  9 67  9]
```



Creating Pie Charts

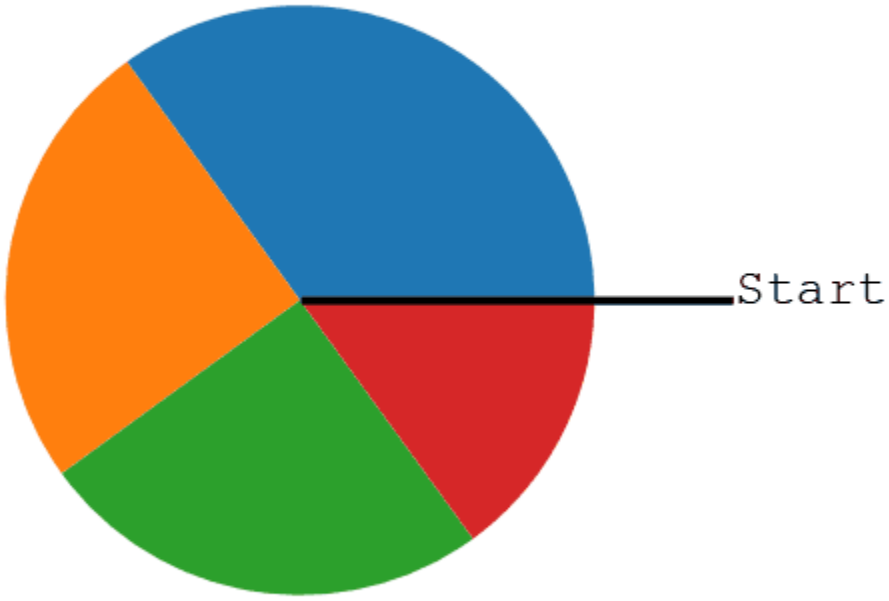
With Pyplot, you can use the pie() function to draw pie charts:

In [62]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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

As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35,

25, 25, 15]).

By default the plotting of the first wedge starts from the x-axis and moves counterclockwise:



Note: The size of each wedge is determined by comparing the value with all the other values, by using this formula:

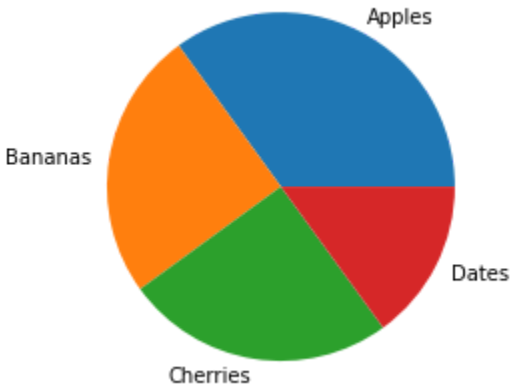
The value divided by the sum of all values: x/sum(x)

Labels

Add labels to the pie chart with the label parameter.

The label parameter must be an array with one label for each wedge:

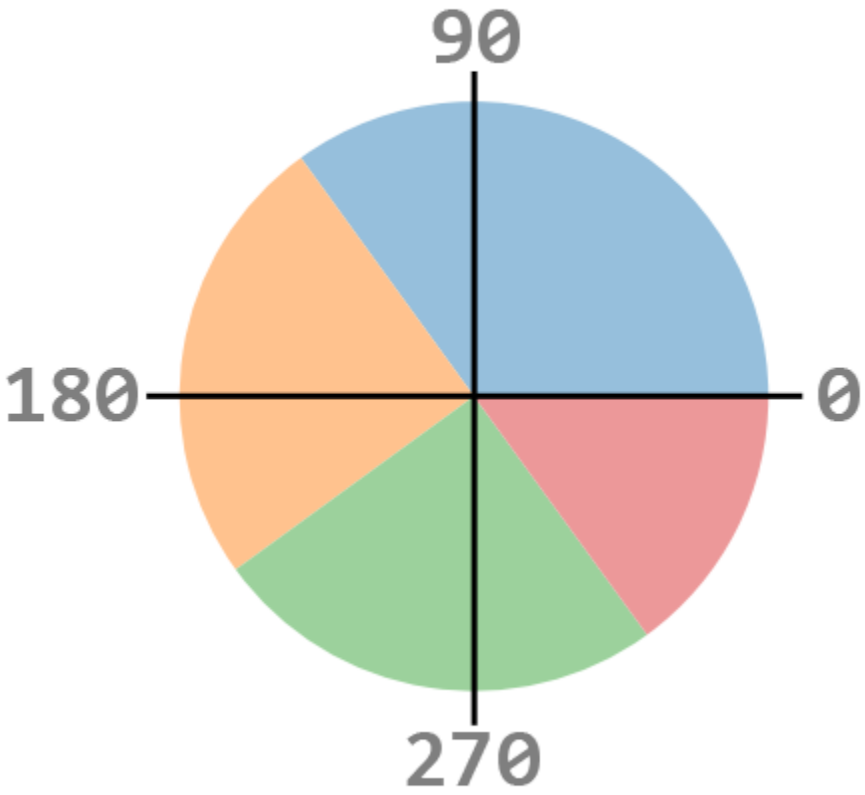
In [64]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Start Angle

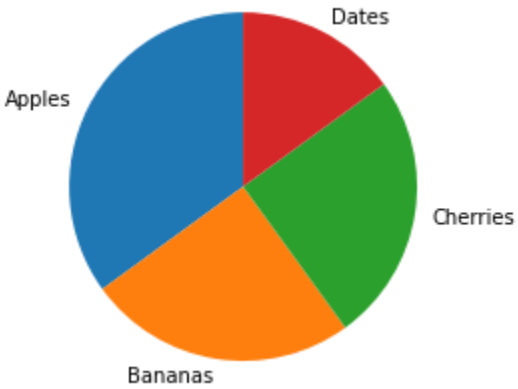
As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

The startangle parameter is defined with an angle in degrees, default angle is 0:



In [65]:

```
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, startangle = 90)
8 plt.show()
```



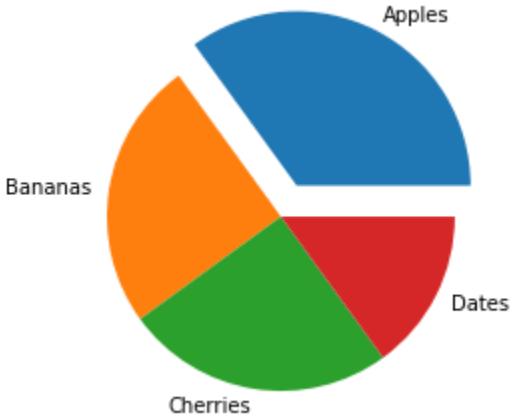
Explode

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 [66]:
```
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:

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 [68]:
```
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()
```

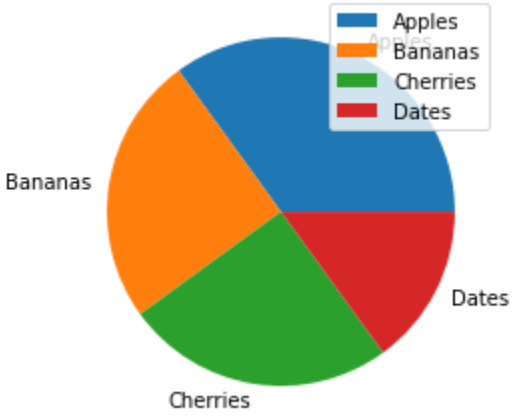
You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:

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

Legend

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

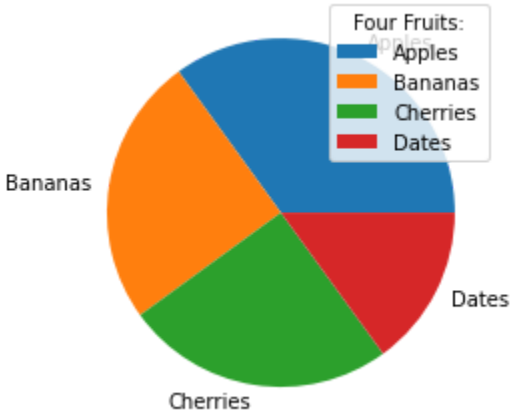
In [69]:
```
import matplotlib.pyplot as plt
import numpy as np

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 [70]:
```
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
```