By Shreeyansh, source: w3Schools

In [1]:
```
import matplotlib
```

In [2]:
```
print(matplotlib.__version__)
```

```
3.3.4
```

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

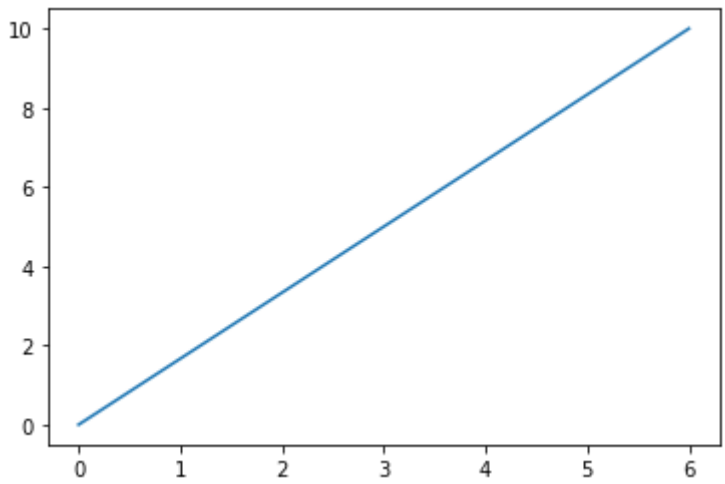
---

1. Matplotlib Pyplot

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

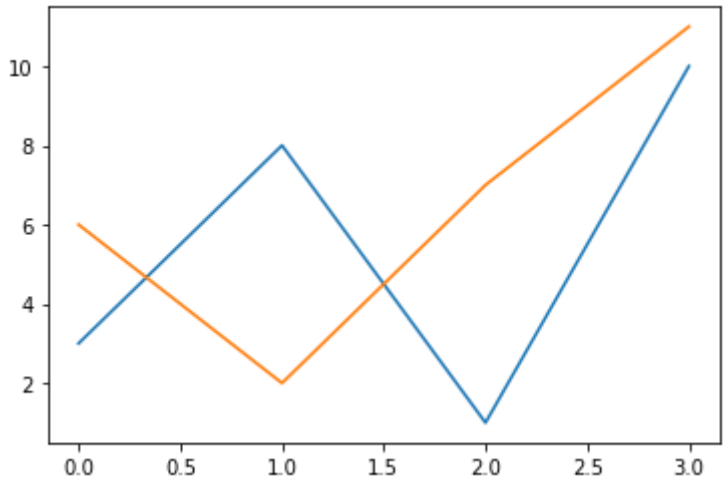
In [4]:
```
import matplotlib.pyplot as plt
```

In [5]:
```
X = np.array([0,6])
Y = np.array([0,10])

plt.plot(X,Y)
plt.show()
```



In [6]:
```
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()
```



---

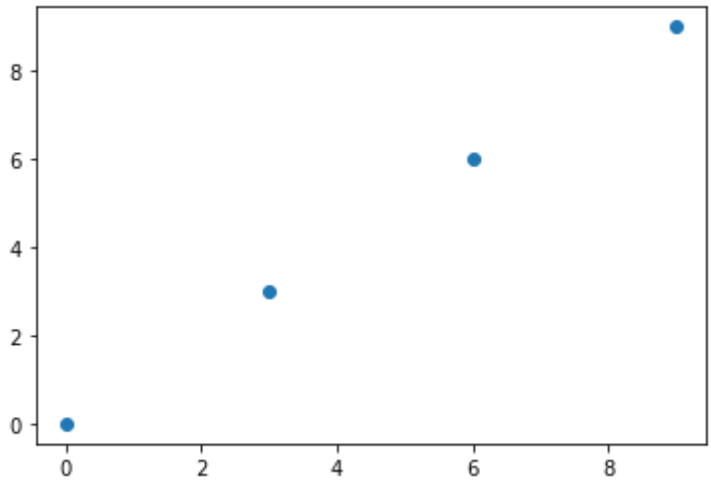
2. Matplotlib Plotting

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. 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.

2.1 Plotting w/o Line

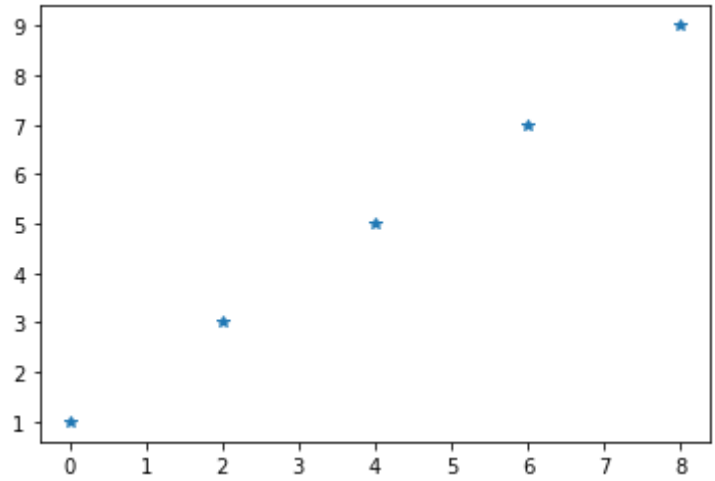
To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

In [7]:
```
X = np.arange(0,12,3)
Y = np.arange(0,12,3)
plt.plot(X,Y,'o')
plt.show()
```



In [8]:
```
X = np.array([0,2,4,6,8])
Y = np.array([1,3,5,7,9])
```

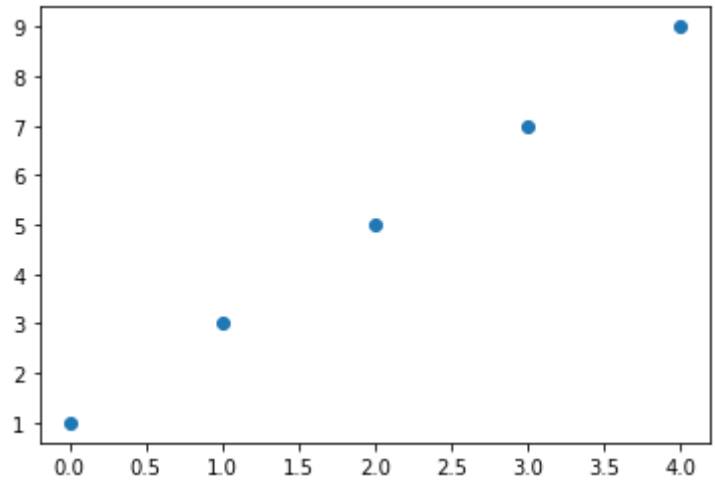
```
plt.plot(X,Y,'*')
plt.show()
```



2.2 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 [9]:
```
plt.plot(Y, 'o')
plt.show()
```



---

3. Matplotlib Markers and Lines

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

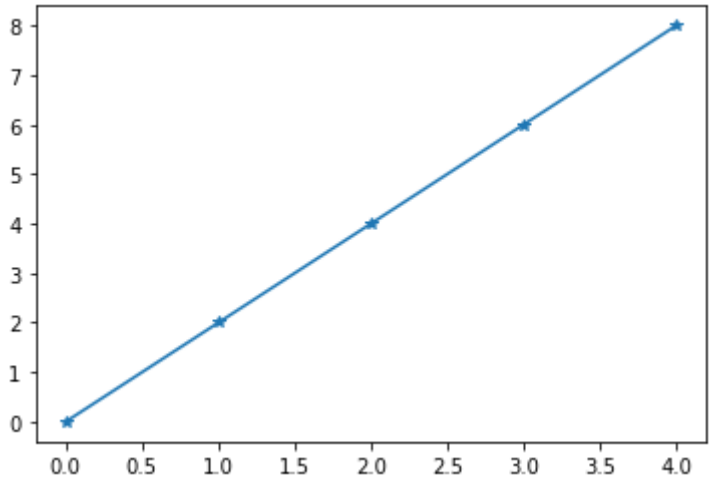
3.1 Marker Reference

```
'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
```

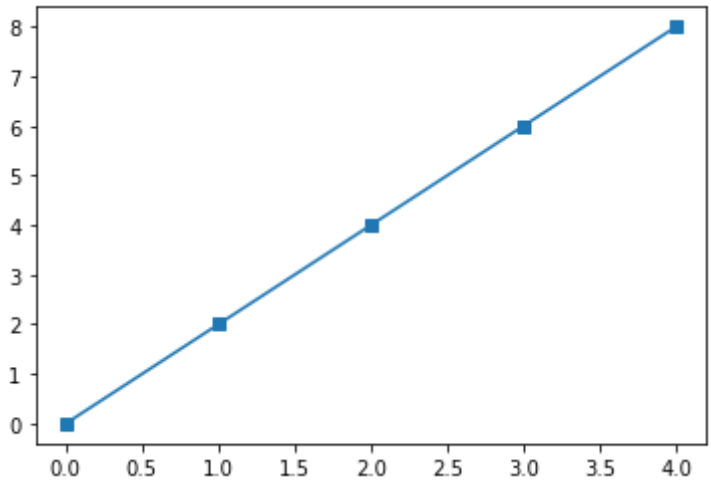
In [10]:

```
plt.plot(X, marker = '*')
plt.show()
```



In [11]:

```
plt.plot(X, marker = 's')
plt.show()
```



3.2 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`. ***Note: If you leave out the line value in the fmt parameter, no line will be plottet.***

Marker Reference As above

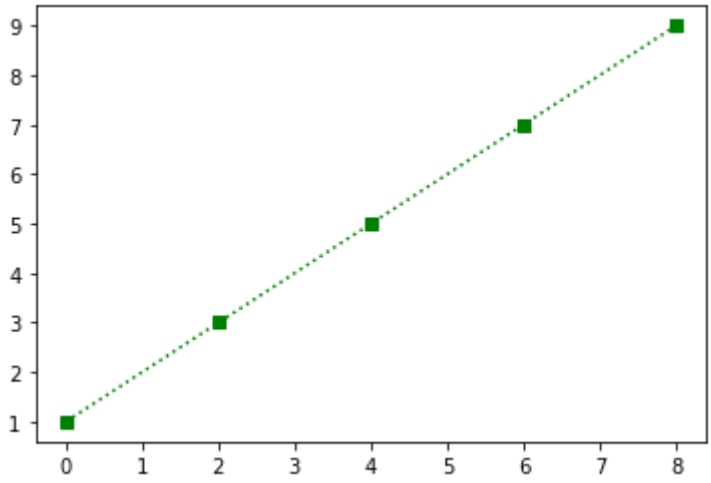
Line Reference

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

Color Reference

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

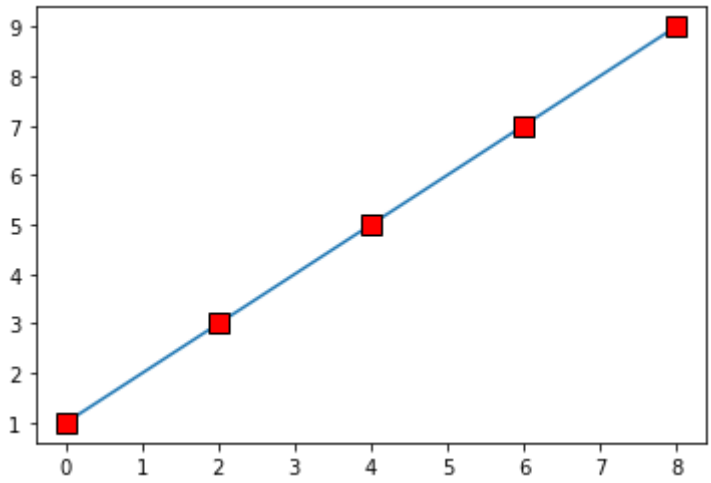
In [12]:
```
plt.plot(X, Y, 's:g')       #s:g - s = square marker, : = dotted lines, g = green co
plt.show()
```



3.3 Marker Colors and Size

`mec` - To set Marker Edge Color `mfc` - To set Marker Face Color `ms` - To set Marker Size

In [13]:
```
plt.plot(X, Y, marker = 's', mec = 'k', mfc = 'r', ms = 10)
plt.show()
```



3.4 Lines

The line style can be written in a shorter syntax:

`linestyle` can be written as `ls`.

`dotted` can be written as `:`.

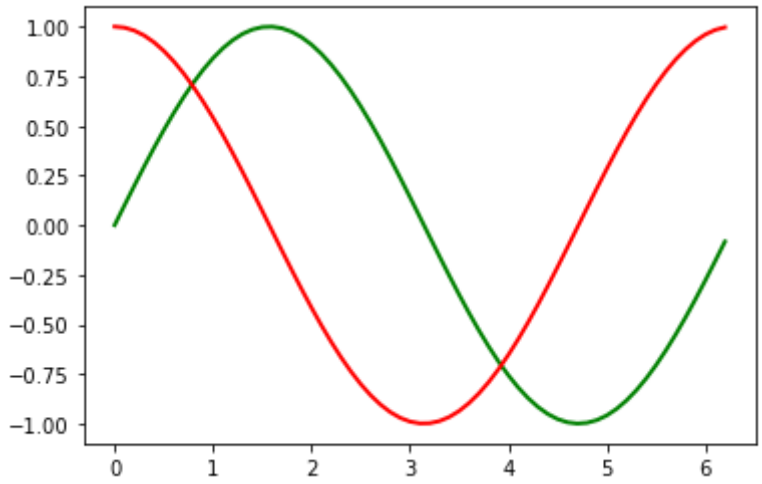
`dashed` can be written as `--`.

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

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.

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

In [14]:
```
import math
X = np.arange(0,2*math.pi,0.1)
A = np.sin(X)
B = np.cos(X)

plt.plot(X,A, ls = '-', c = 'g', lw = 2)
plt.plot(X,B, ls = '-', c = 'r', lw = 2)

plt.show()
```



---

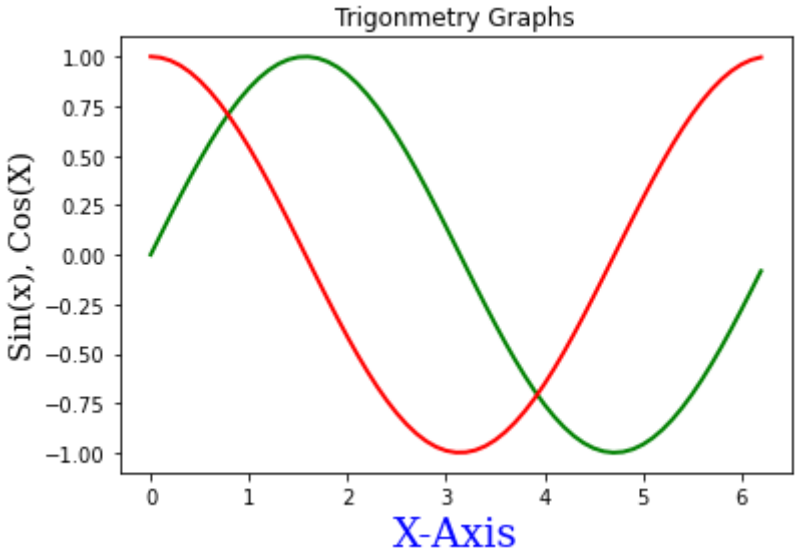
4. Matplotlib Labels

With Pyplot, you can use

- the `xlabel()` and `ylabel()` functions to set a label for the x- and y-axis.
- the `title()` function to set a title for the plot.
- the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels. `fontdict` should be a font dictionary with properties for the font.
- the `loc` parameter in `title()` to position the title. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

In [15]:
```
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'black','size':15}
```

In [16]:
```
plt.plot(X,A, ls = '-', c = 'g', lw = 2)
plt.plot(X,B, ls = '-', c = 'r', lw = 2)

plt.title("Trigonmetry Graphs")
plt.xlabel("X-Axis", fontdict = font1)
plt.ylabel("Sin(x), Cos(X)", fontdict = font2)
plt.show()
```

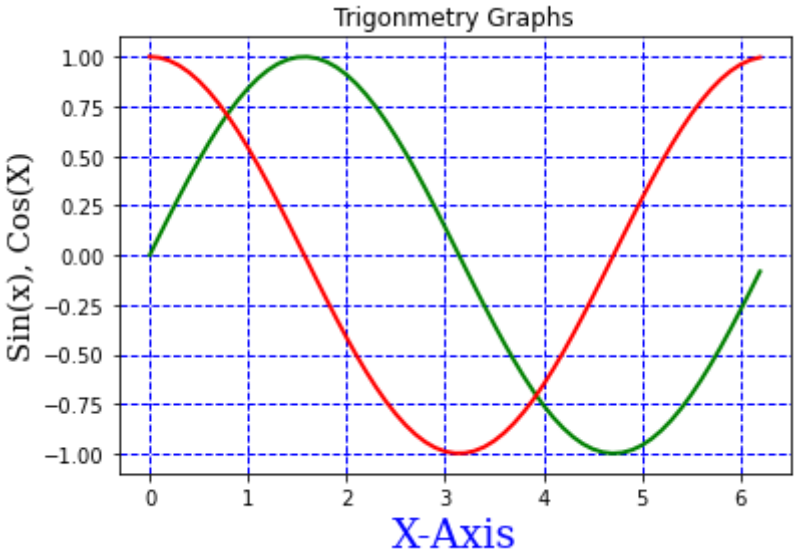


5. Matplotlib Gridlines

With Pyplot, you can use the

- `grid()` function to add grid lines to the plot.
- 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'.
- the line properties of the grid, like this: `grid(color = 'color', linestyle = 'linestyle', linewidth = number)`.

In [17]:
```
plt.plot(X,A, ls = '-', c = 'g', lw = 2)
plt.plot(X,B, ls = '-', c = 'r', lw = 2)

plt.title("Trigonmetry Graphs")
plt.xlabel("X-Axis", fontdict = font1)
plt.ylabel("Sin(x), Cos(X)", fontdict = font2)

plt.grid(axis = 'both', c = 'b', ls = '--', lw = 1)
plt.show()
```



---

6. Matplotlib Subplots

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.

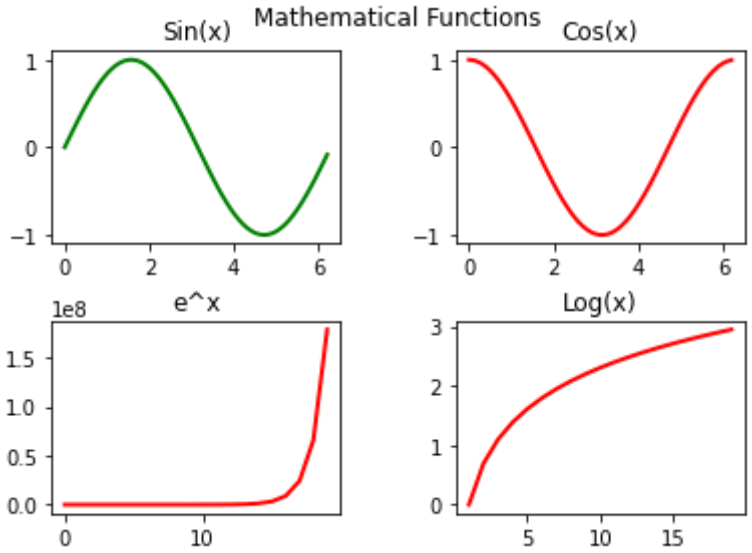
We can use the `plt.subplots_adjust()` method to change the space between Matplotlib subplots. The parameters `wspace` and `hspace` specify the space reserved between Matplotlib subplots. They are the fractions of axis width and height, respectively. And the parameters `left`, `right`, `top` and `bottom` parameters specify four sides of the subplots' positions. They are the fractions of the width and height of the figure.

In [18]:
```
#  -------------- --------------
#  |            | |            |
#  |            | |            |
#  |      1     | |      2     |
#  |            | |            |                1 - (2,2,1)
#  |            | |            |                2 - (2,2,2)
#  -------------- --------------                3 - (2,2,3)
#  -------------- --------------                4 - (2,2,4)
```

```
#  |              | |              |
#  |              | |              |
#  |      3       | |       4      |
#  |              | |              |
#  |              | |              |
#  --------------- ---------------
```

In [19]:

```
import math
X1 = np.arange(0,2*math.pi,0.1)
X2 = np.arange(0,20)
A = np.sin(X1)
B = np.cos(X1)
C = np.exp(X2)
D = np.log(np.asarray(X2)[1:])      #Excluding the 1st element '0' in range X2. Firs

plt.subplot(2,2,1)                                    #plotting 1st graph
plt.plot(X1,A, ls = '-', c = 'g', lw = 2)
plt.title("Sin(x)")

plt.subplot(2,2,2)                                    #plotting 2nd graph
plt.plot(X1,B, ls = '-', c = 'r', lw = 2)
plt.title("Cos(x)")

plt.subplot(2,2,3)                                    #plotting 3rd graph
plt.plot(X2,C, ls = '-', c = 'r', lw = 2)
plt.title("e^x")

plt.subplot(2,2,4)                                    #plotting 4th graph
plt.plot(np.asarray(X2)[1:],D, ls = '-', c = 'r', lw = 2)
plt.title("Log(x)")


plt.suptitle("Mathematical Functions")
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```

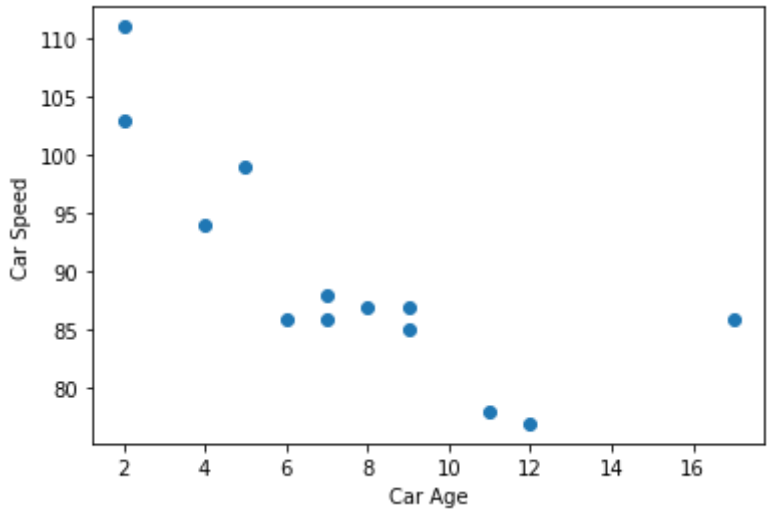


7. Matplotlib Scatterplot

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 [20]:
```
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])
```

In [21]:
```
plt.scatter(x,y)
plt.xlabel("Car Age")
plt.ylabel("Car Speed")
plt.show()
```

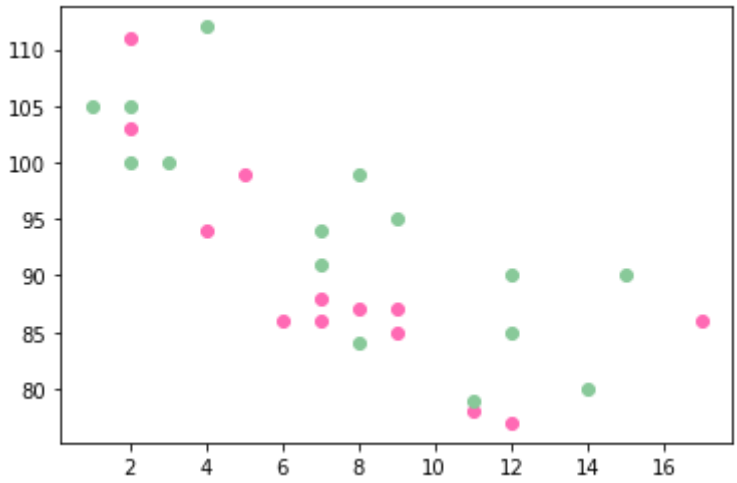


The observation in the example above is the result of 13 cars passing by. The X-axis shows how old the car is. The Y-axis shows the speed of the car when it passes. Are there any relationships between the observations? It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

7.1 Comparison by Coloring

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

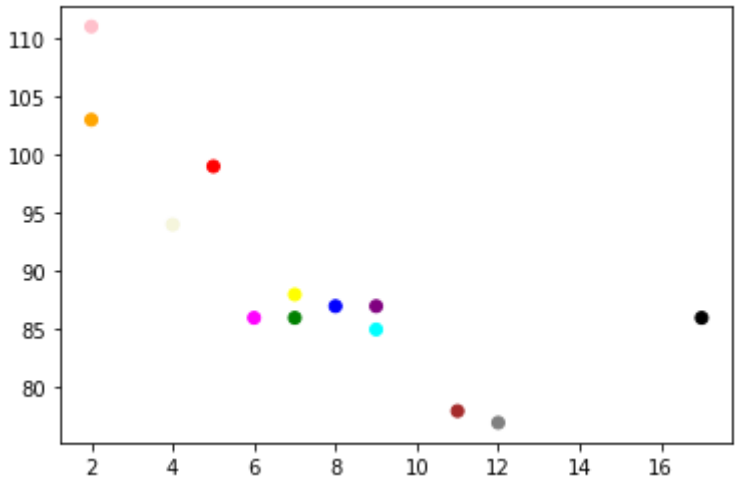
In [22]:
```
#day 1
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

#day 2
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()
```



If we pass a *color array* as an argument instead of a single color, we can set the color for each dot in the scatter plot. You cannot use the `color` argument for this, only the `c` argument.

In [23]:

```
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()
```



7.2 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.

Arguments

- This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, and up to 100, which is a yellow color. 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 of the point in the scatter plot.**
- You can include the colormap in the drawing by including the `plt.colorbar()` statement.
- 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.
- 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.

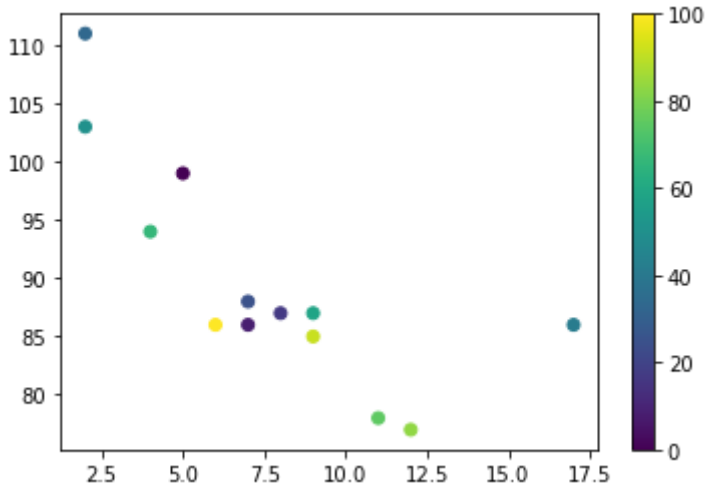
In [2]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

colors = np.linspace(0,100,13)
colors
```

Out[2]:
```
array([  0.        ,   8.33333333,  16.66666667,  25.        ,
        33.33333333,  41.66666667,  50.        ,  58.33333333,
        66.66666667,  75.        ,  83.33333333,  91.66666667,
       100.        ])
```

There are 13 max elements in both arrays so we need to ensure there are 13 elements in `colors` array.

In [5]:
```
plt.scatter(x,y, c = colors, cmap = 'viridis')
plt.colorbar()
```

```
plt.show()
```

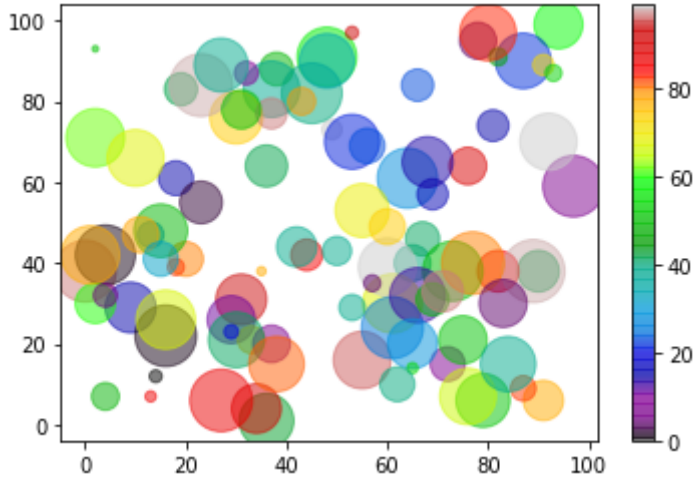


Some built-in colormaps are:

- Accent, Accent_r
- Blues, Blues_r
- BrBG, BrBG_r
- BuGn, BuGn_r
- BuPu, BuPu_r
- CMRmap, CMRmap_r
- Dark2, Dark2_r
- GnBu, GnBu_r
- Greens, Greens_r
- Greys, Greys_r
- OrRd, OrRd_r
- Oranges, Oranges_r
- PRGn, PRGn_r
- Paired, Paired_r
- Pastel1, Pastel1_r

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

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



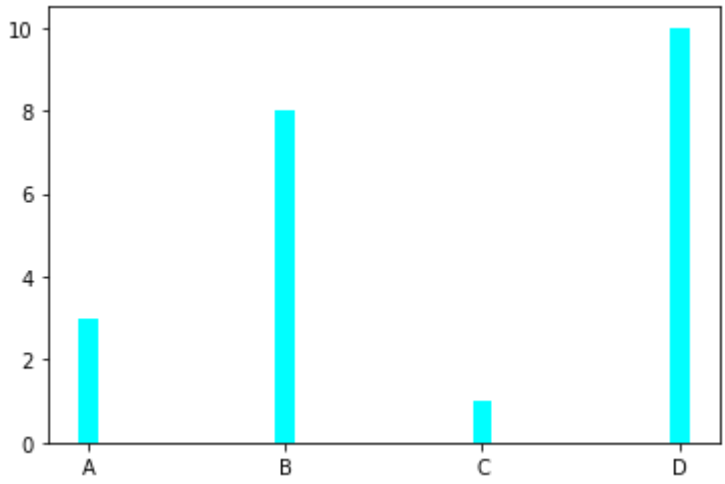
---

8. Matplotlib Bars

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

- If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function. The `bar()` and `barh()` takes the keyword argument color to set the color of the bars.
- The `bar()` takes the keyword argument `width` to set the width of the bars.
- The `barh()` takes the keyword argument height to set the height of the bars. Default value is 0.8

In [28]:
```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



---

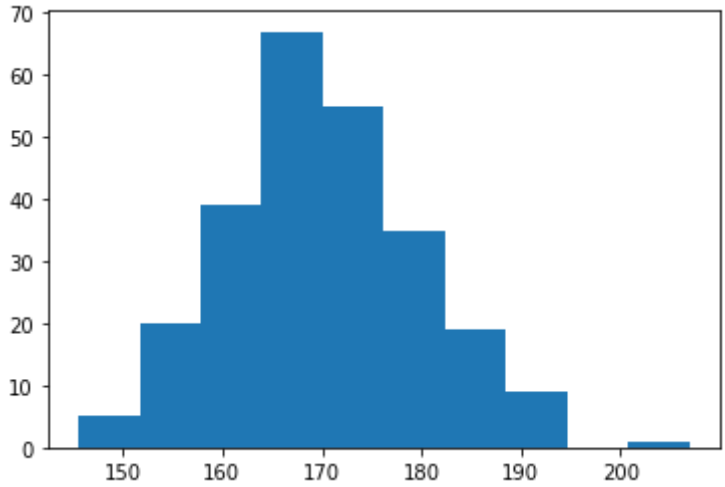
9. Matplotlib Histogram

A histogram is a graph showing *frequency* distributions. It is a graph showing the number of observations within each given interval. 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 [29]:
```
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```



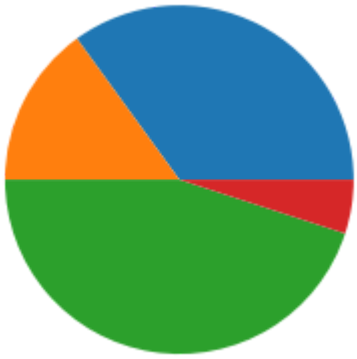
10. Matplotlib Piechart

With Pyplot, you can use the `pie()` function to draw pie charts. 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 move counterclockwise.**

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)`

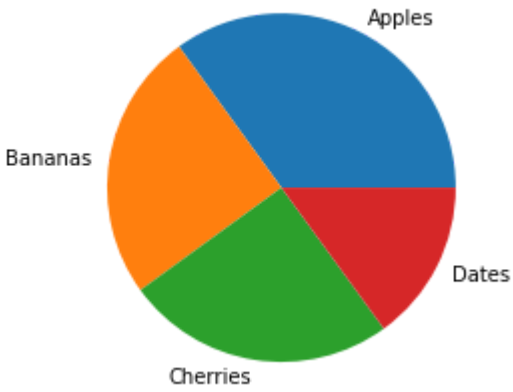
In [31]:
```
y = np.array([35,25,25,15])
plt.pie(y)
plt.show()
```



- ### `labels`

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

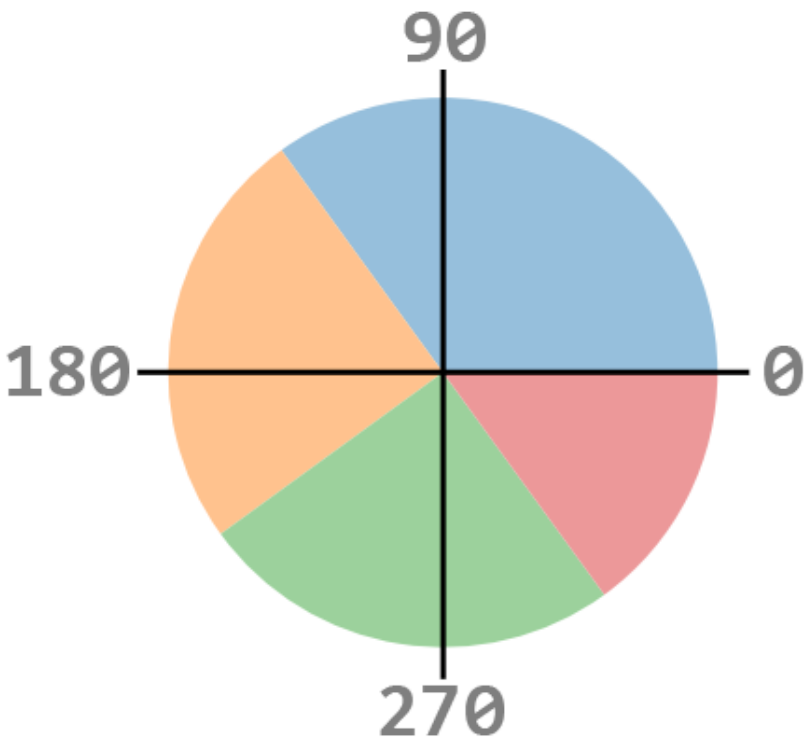
In [32]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

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



- ### `startangle`

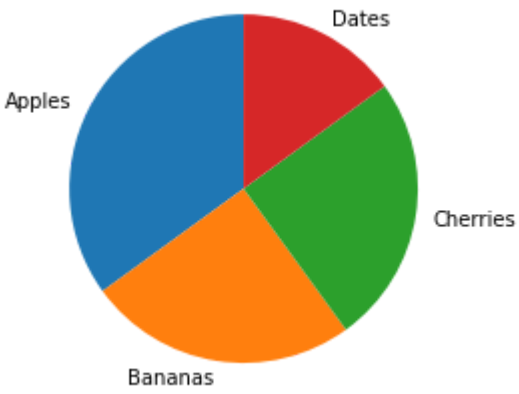
The default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter. The `startangle` parameter is defined with an angle in degrees, default

angle is 0.



In [33]:

```
#start the first wedge at 90 degrees

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

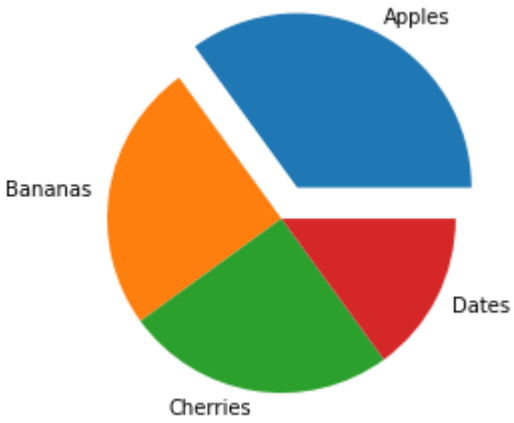


- ### `explode`

The `explode` parameter allows you to make one wedge stand out. 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 [36]:

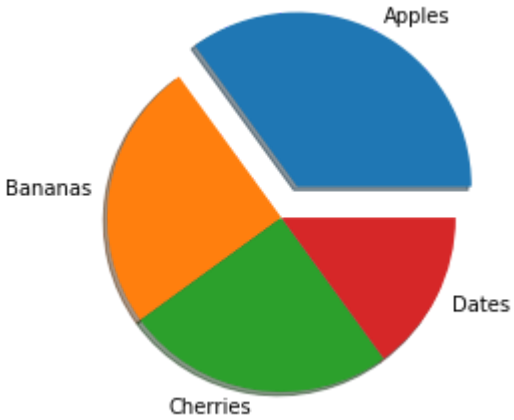
```
myexplode = [0.2, 0, 0, 0]

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



- ### `shadow`

Add a shadow to the pie chart by setting the `shadow` parameter to True.

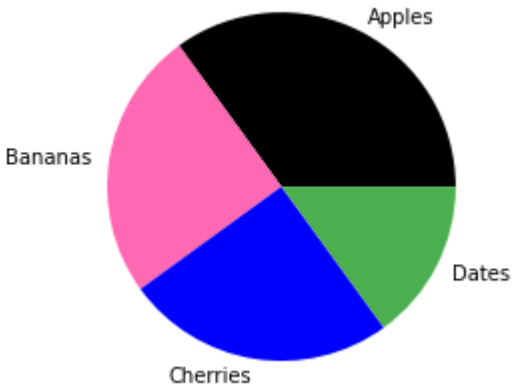
In [38]:
```
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```



- ### `colors`

Each wedge can be assigned a certain color with the `color` parameter which is an array with one value for each wedge.

In [40]:
```
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 with `title` argument to set the title for legend.

In [43]:
```
plt.pie(y, labels = mylabels, colors = mycolors)

plt.legend(title = "4 fruits")
plt.show()
```