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 witten in C,Objectives-C and Javascript for Platform compatibility.

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

In [147]:
```
import matplotlib

print(matplotlib.__version__)
```

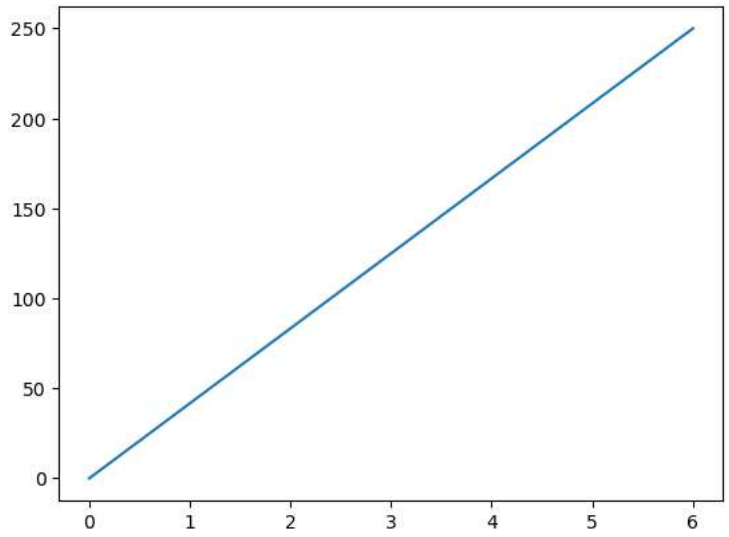
```
3.5.2
```

pyplot

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

In [145]:
```
import matplotlib.pyplot as plt
```

In [143]:
```
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()nfunction draws a line from point to point.

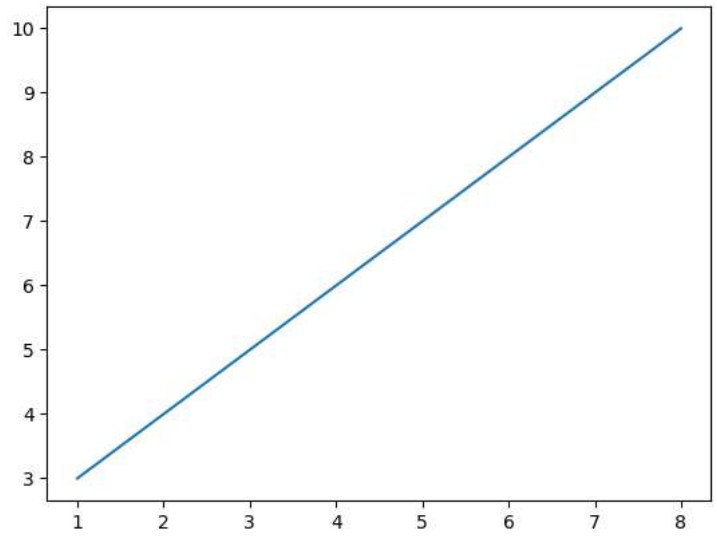
The function takes parameters for specifying points in the diagram.

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 [142]:
```
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()
```

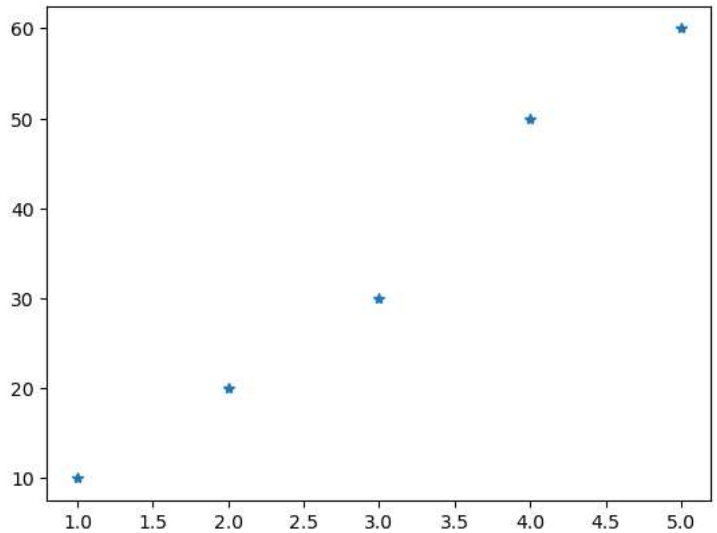


Plotting Without Line

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

In [ ]:
```
# Draw two points in the diagram, one at position (1, 3) and one in position (8,)
```

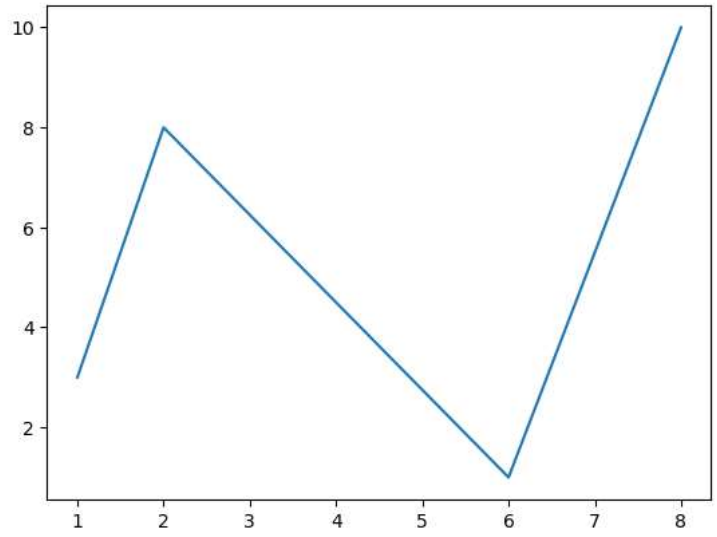
In [141]:
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1,2,3,4,5])
ypoints = np.array([10,20,30,50,60])

plt.plot(xpoints,ypoints, '*')
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 [140]:
```
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()
```

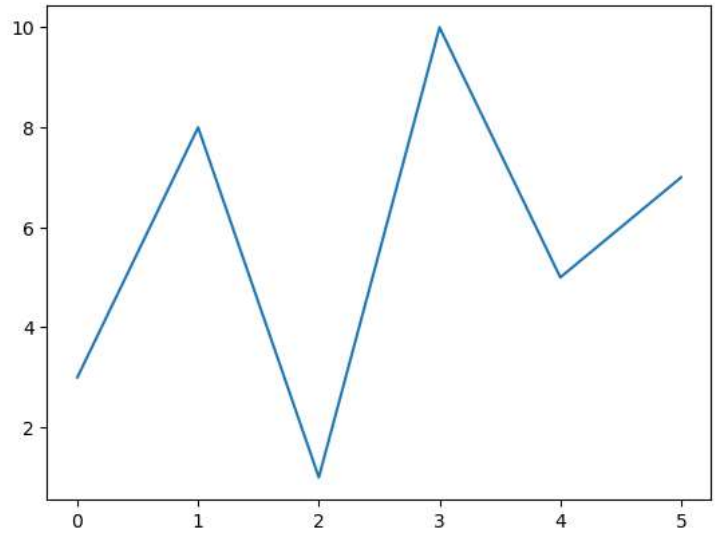


Default X-Points

If we do not specify the points 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 examples as above, and leave out the x-points,the diagram will look like this:

In [139]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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

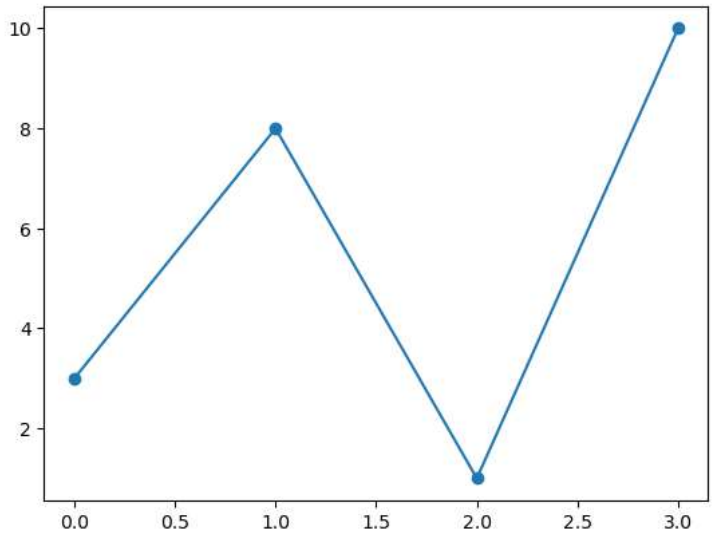


Matplotlib Markers

Markers

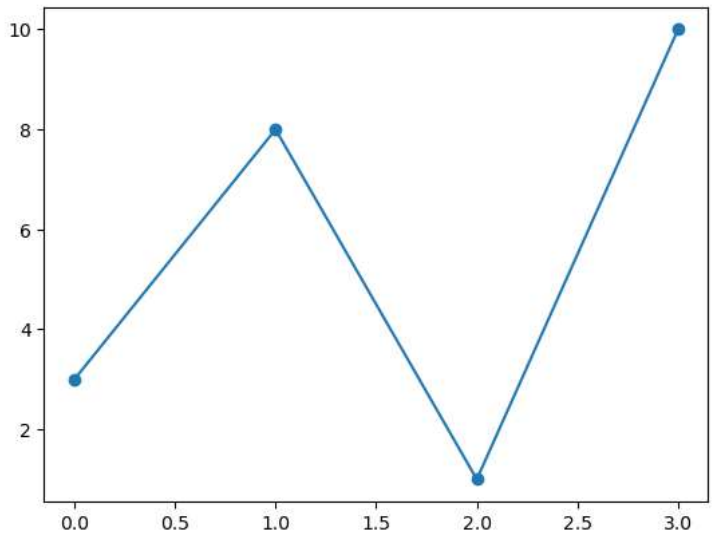
You can use the keyword argument marker to emphasize each point with a specified marker:

In [138]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



In [137]:
```
...
plt.plot(ypoints, marker = 'o')
...
```

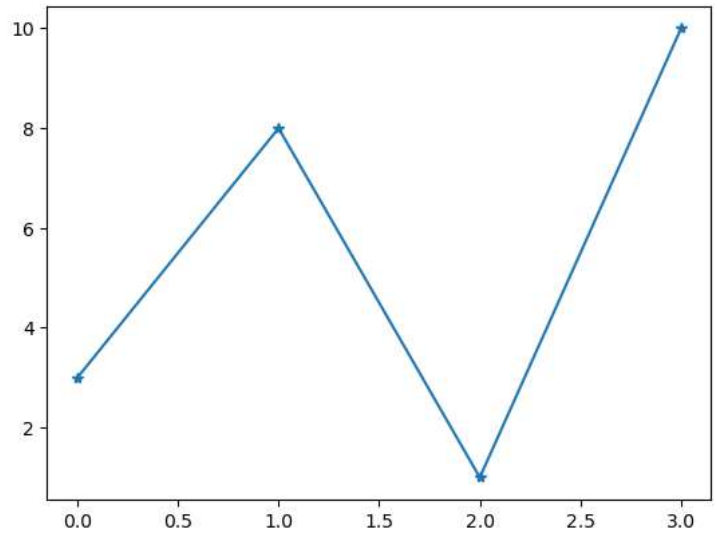
Out[137]: Ellipsis



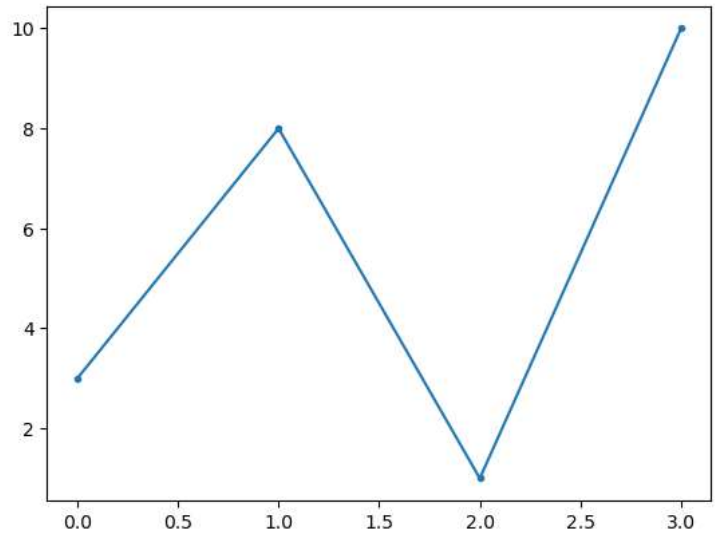
Marker Reference

You can choose any of these markers:

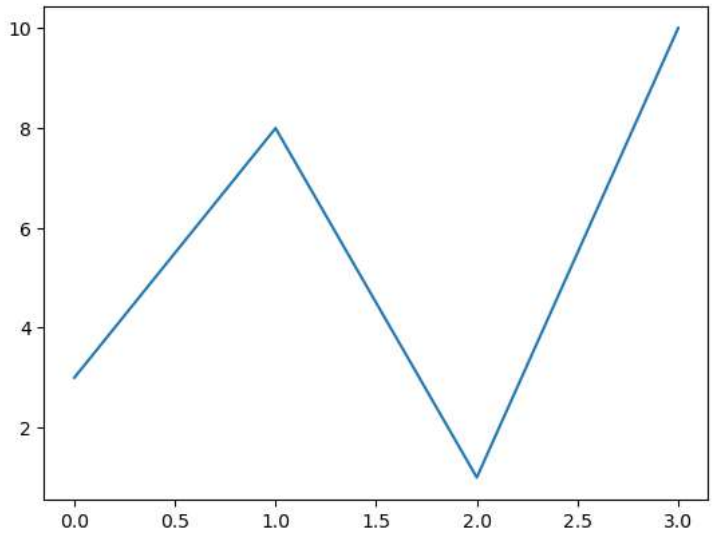
In [136]:
```
# Star
import matplotlib.pyplot as plt
import numpy as np

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

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



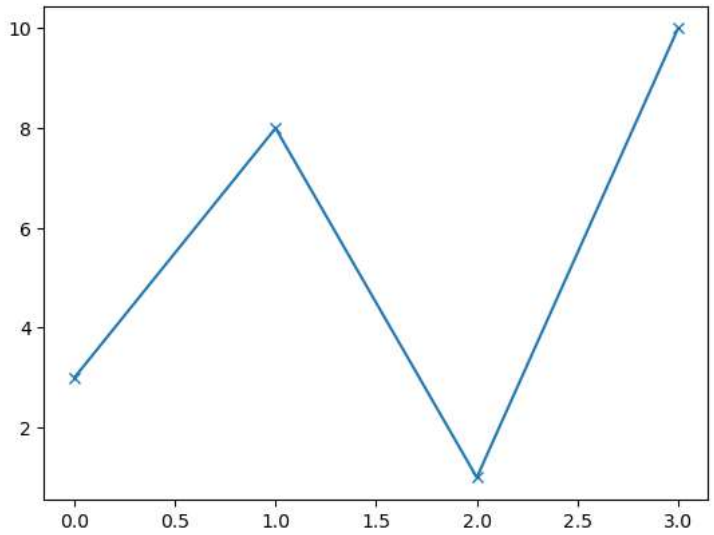
In [135]:
```
# point
import matplotlib.pyplot as plt
import numpy as np

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

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



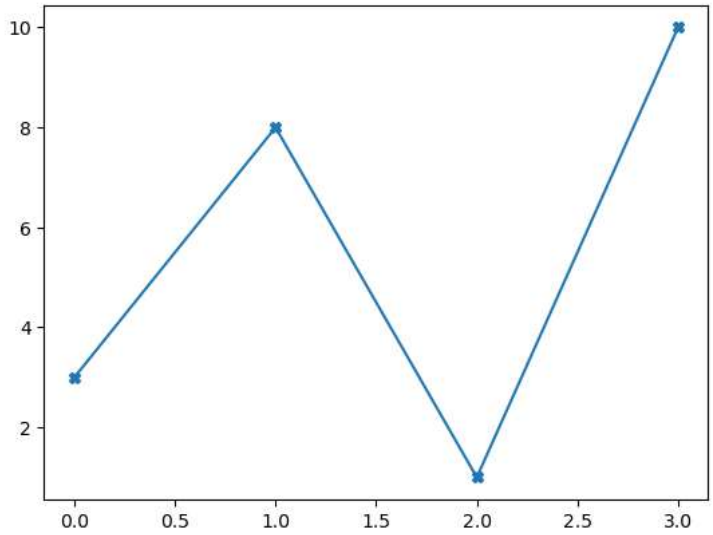
In [134]:
```
# Pixel
import matplotlib.pyplot as plt
import numpy as np

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

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



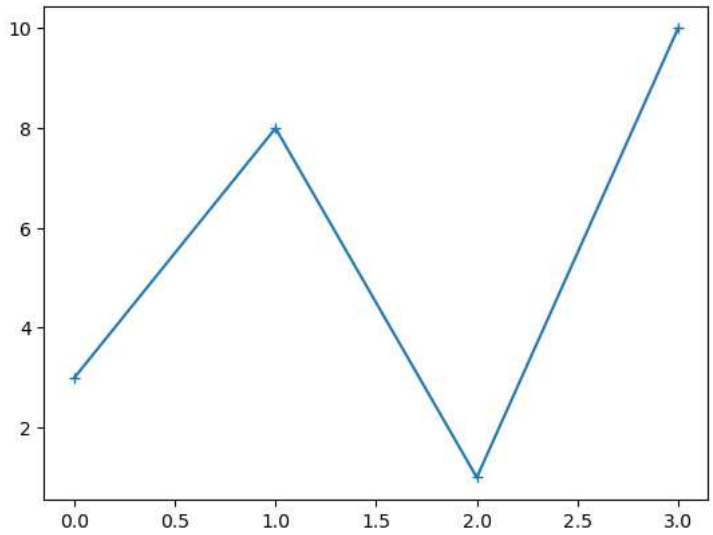
In [133]:
```
# X
import matplotlib.pyplot as plt
import numpy as np

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

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



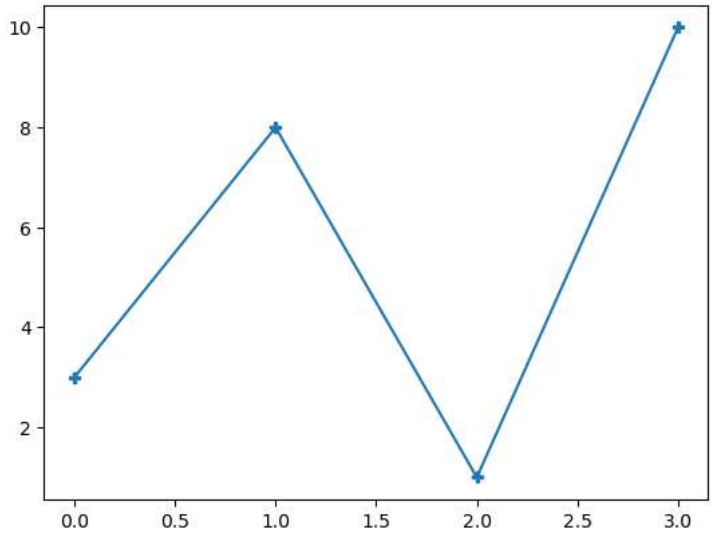
In [132]:
```
# x
import matplotlib.pyplot as plt
import numpy as np

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

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



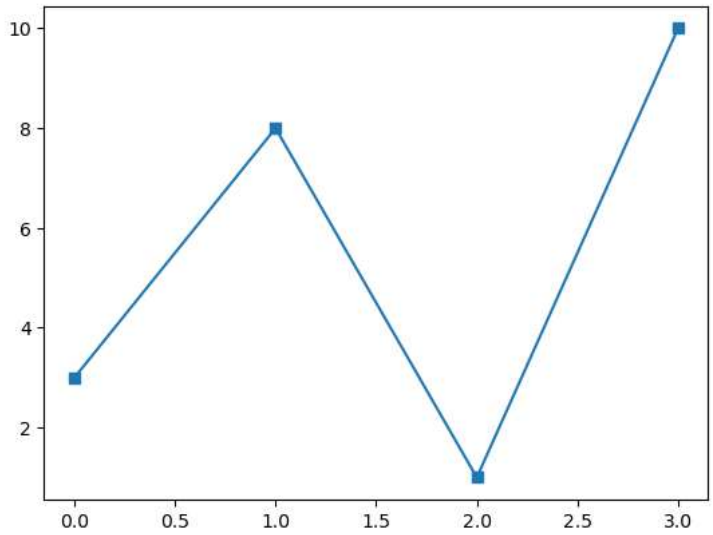
In [131]:
```
# Plus
import matplotlib.pyplot as plt
import numpy as np

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

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



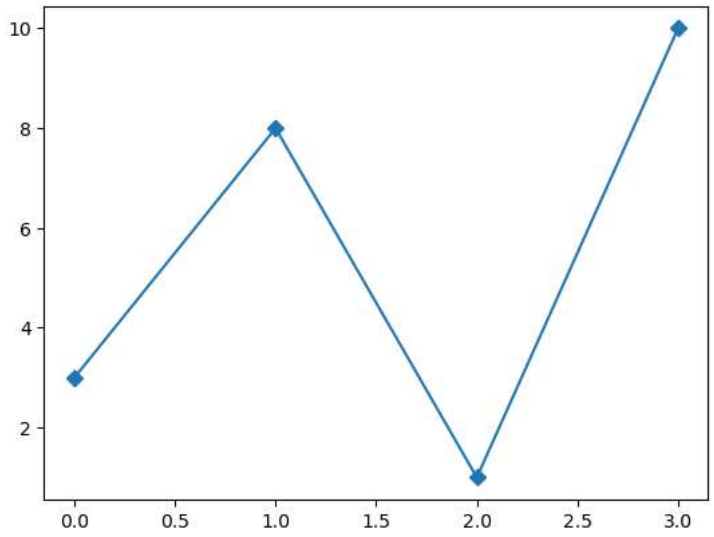
In [130]:
```
# Plus (filled) - 'p'
import matplotlib.pyplot as plt
import numpy as np

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

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



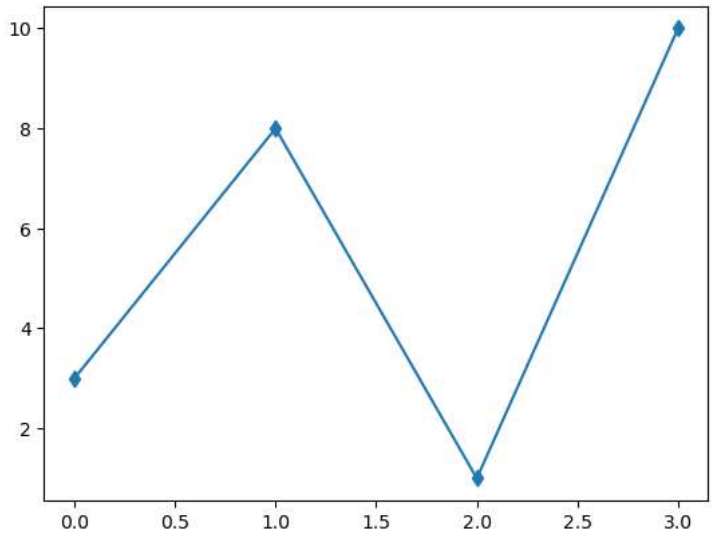
In [129]:
```
#'S' Square
import matplotlib.pyplot as plt
import numpy as np

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

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



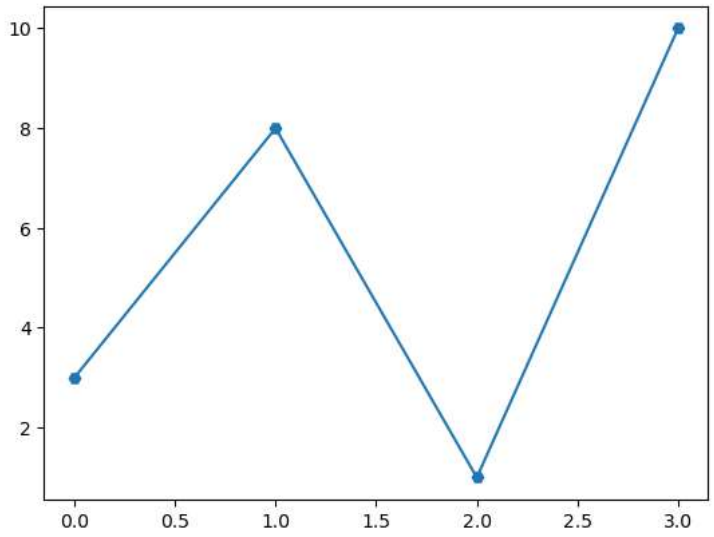
In [128]:
```
# Diamond-"D"
import matplotlib.pyplot as plt
import numpy as np

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

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



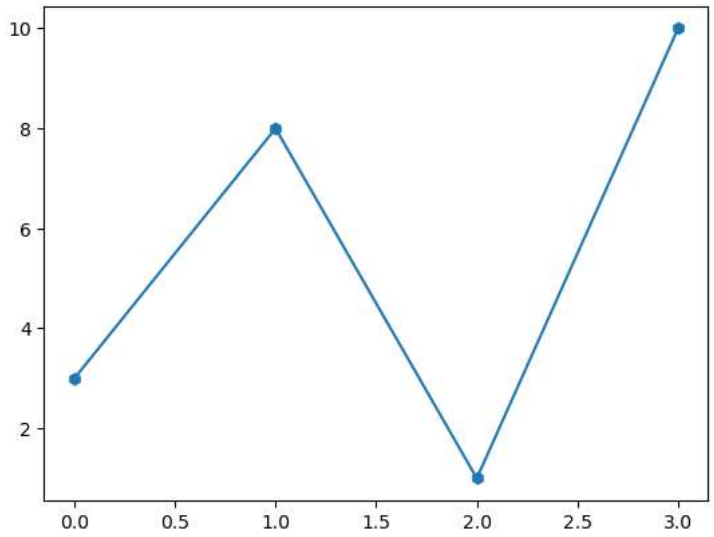
In [127]:
```
# Diamond(thin)-'d'
import matplotlib.pyplot as plt
import numpy as np

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

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



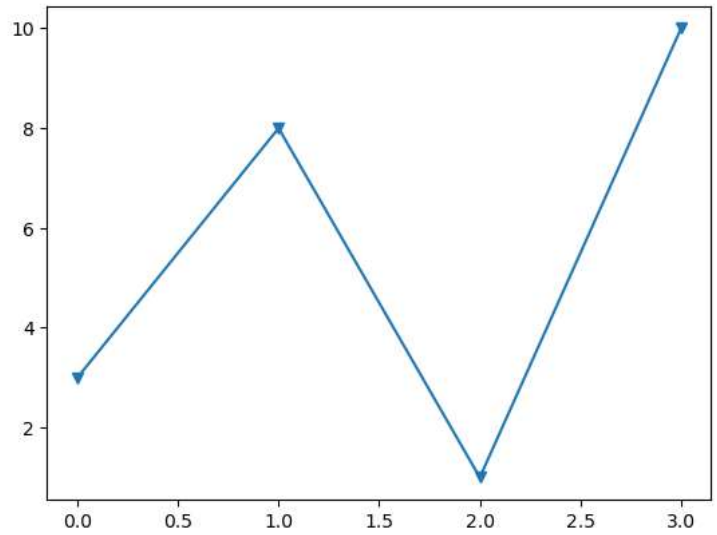
In [126]:
```
# Hexagon-'H'
import matplotlib.pyplot as plt
import numpy as np

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

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



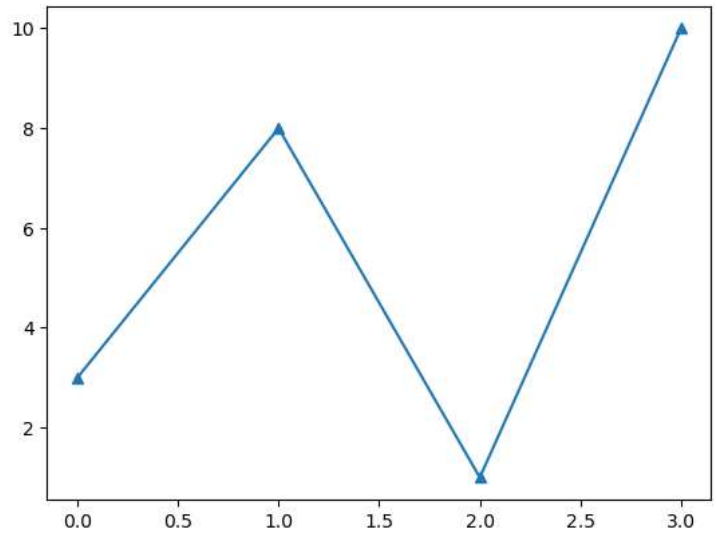
In [125]:
```
# Hexagon-'h'
import matplotlib.pyplot as plt
import numpy as np

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

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



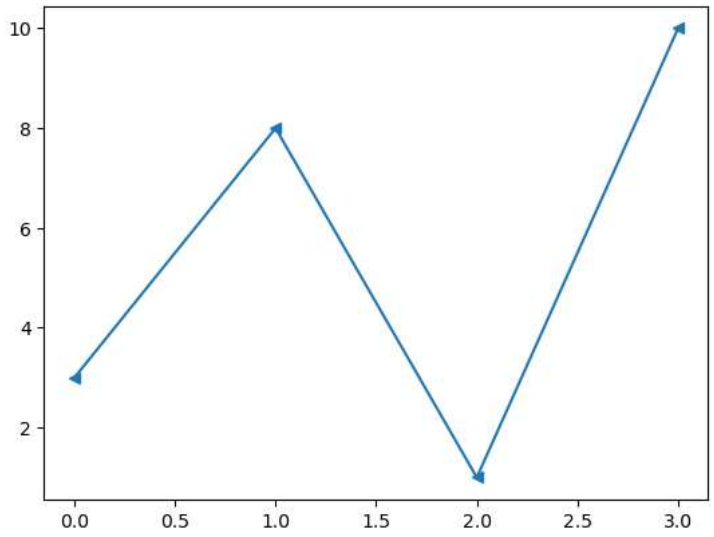
In [124]:
```
# Triangle Down-'v'-(small "v")
import matplotlib.pyplot as plt
import numpy as np

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

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



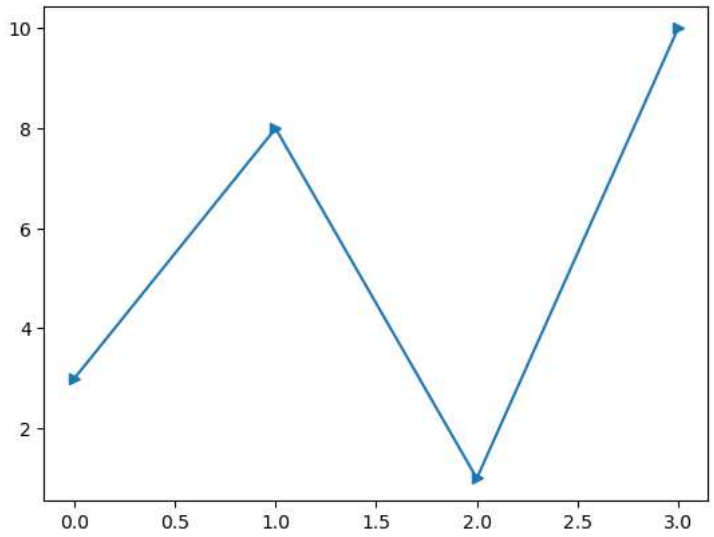
In [123]:
```
# Triangle Up-'^'
import matplotlib.pyplot as plt
import numpy as np

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

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



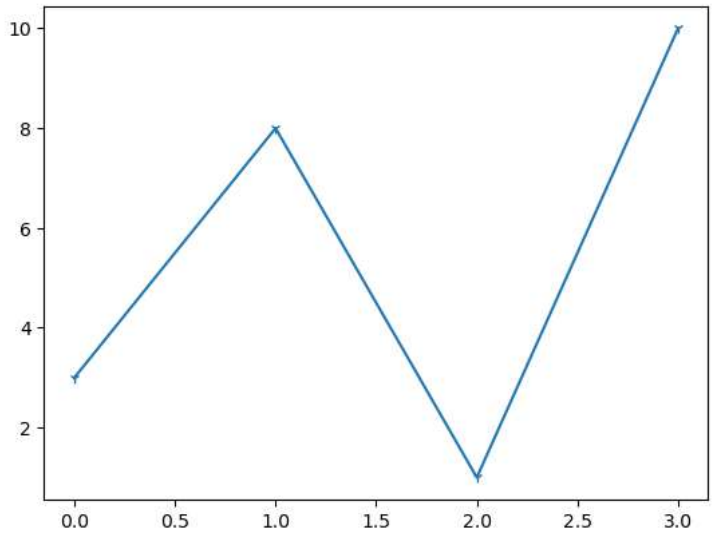
In [122]:
```
# Triangle Left- "<"
import matplotlib.pyplot as plt
import numpy as np

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

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



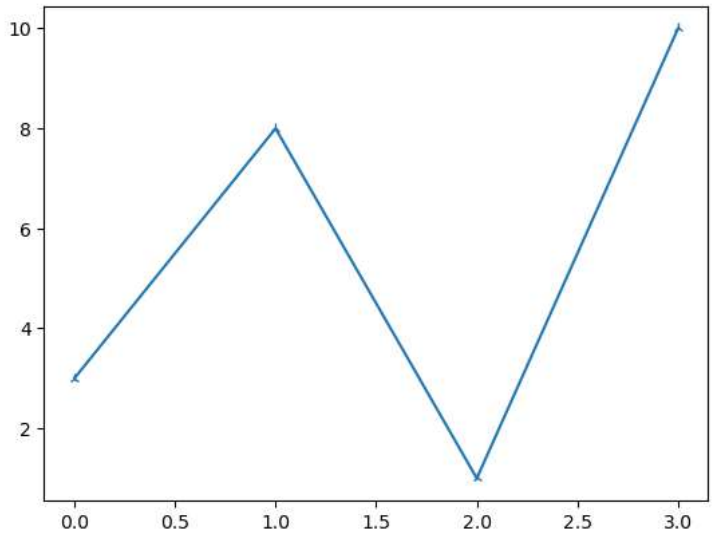
In [121]:
```
# Triangle Right - ">"
import matplotlib.pyplot as plt
import numpy as np

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

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



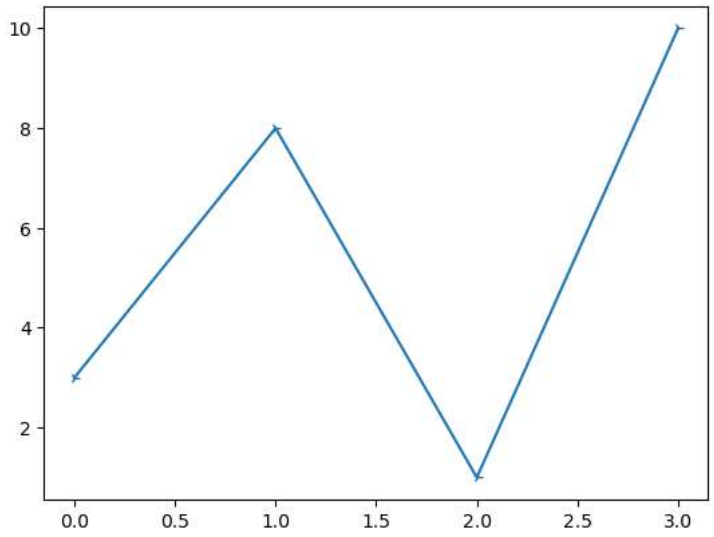
In [120]:
```
# Tri Down "1"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker ='1')
plt.show()
```



In [119]:
```
# Tri up - "2"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker = '2')
plt.show()
```

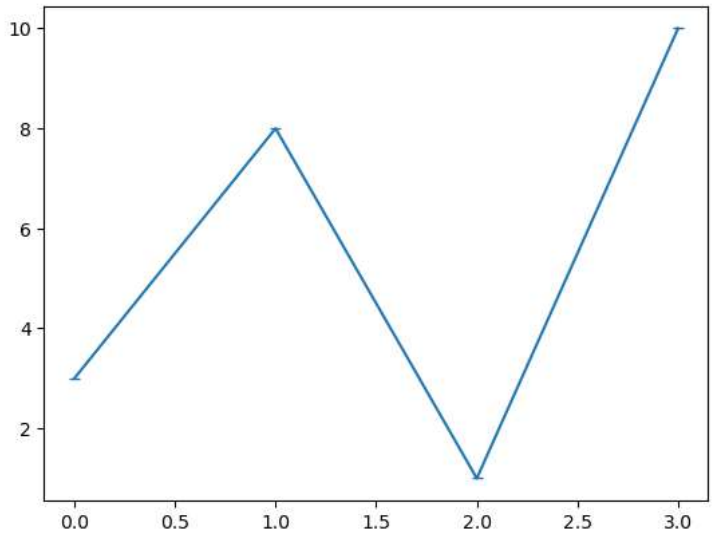


In [118]:
```
# Tri left -" 4"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker = '4')
plt.show()
```



In [ ]:
```

```

In [117]:
```
# Hline - '_'
import matplotlib.pyplot as plt
import numpy as np

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

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



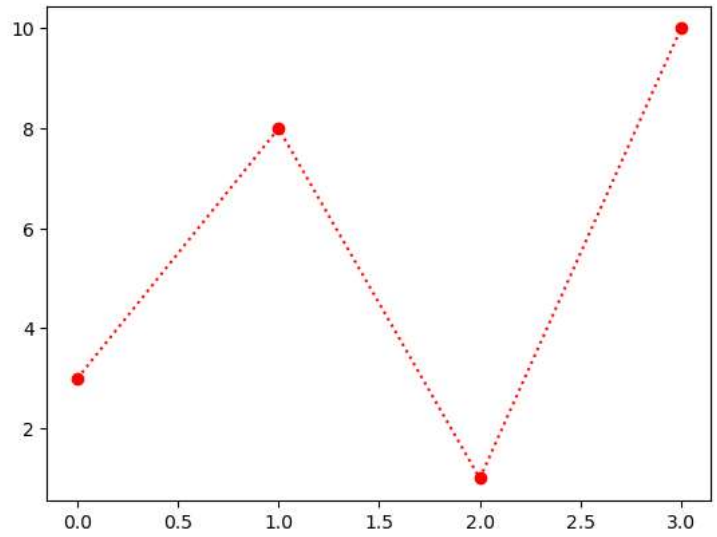
Format Strings "fmt"

You can use also the shortcut string notation parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

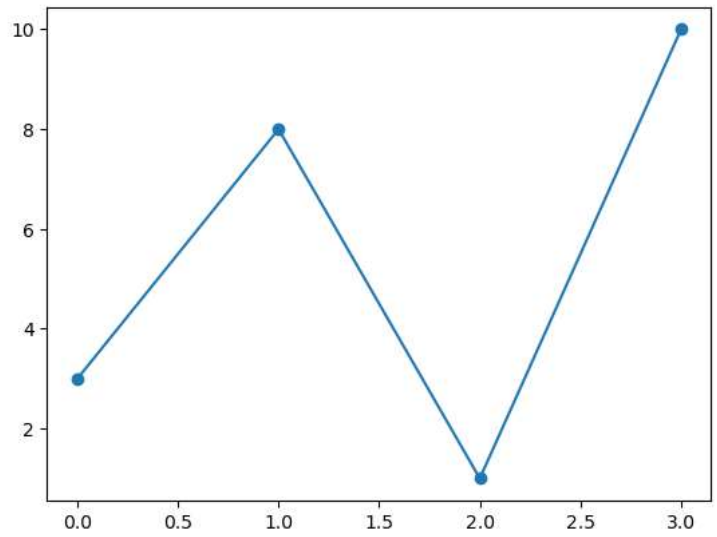
marker line color

In [116]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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

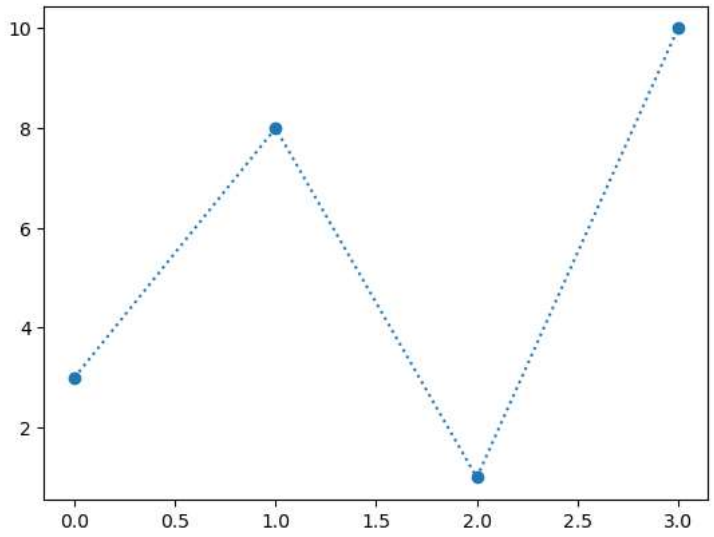


Line Reference

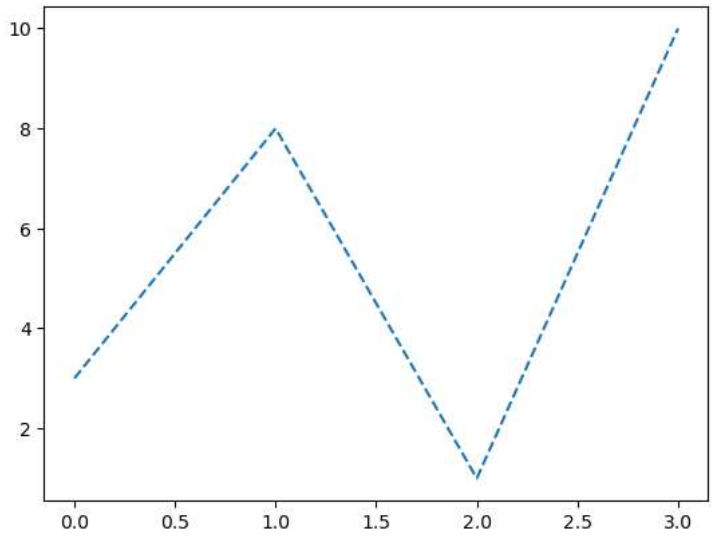
In [115]:
```
# solid line - "-"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, "o-")
plt.show()
```



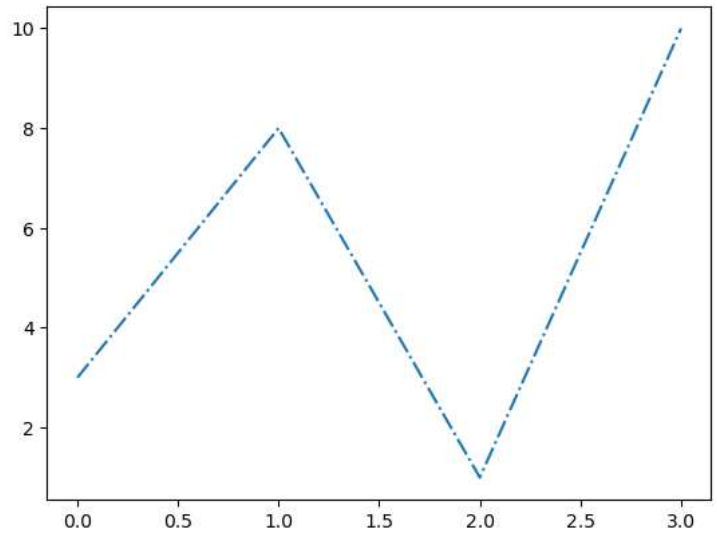
In [114]:
```
# Dotted line - ":"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, "o:")
plt.show()
```



In [113]:
```
# Dashed line - "--"
import matplotlib.pyplot as plt
import numpy as np

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

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

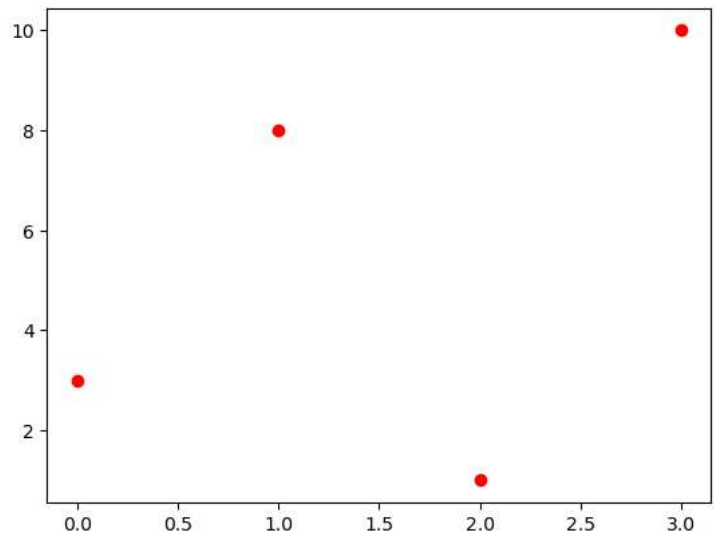


In [112]:
```
# Dashed/dotted Line - '-.'
import matplotlib.pyplot as plt
import numpy as np

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

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

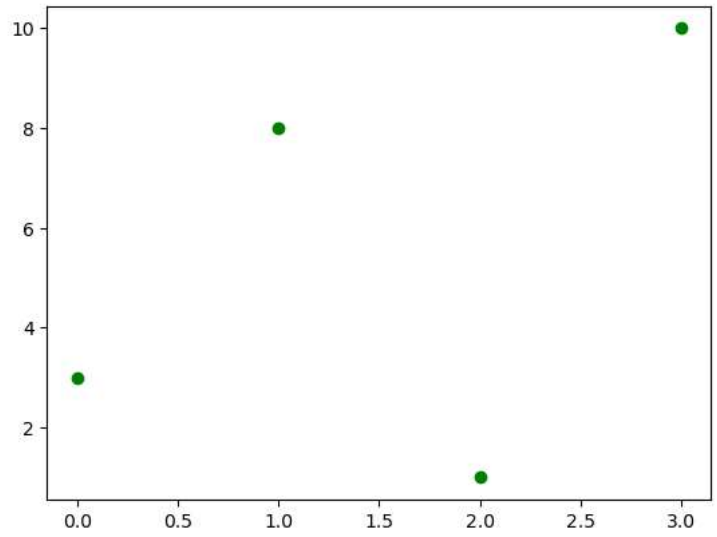


Color Reference

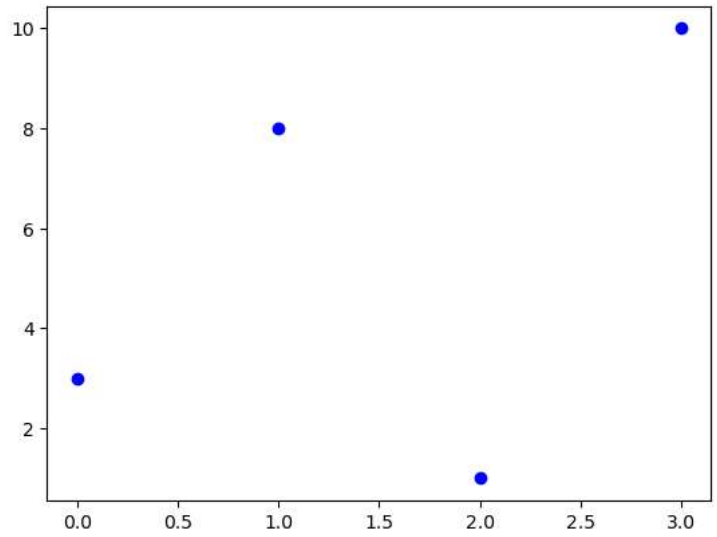
In [111]:
```
# Red - 'r'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'or')
plt.show()
```



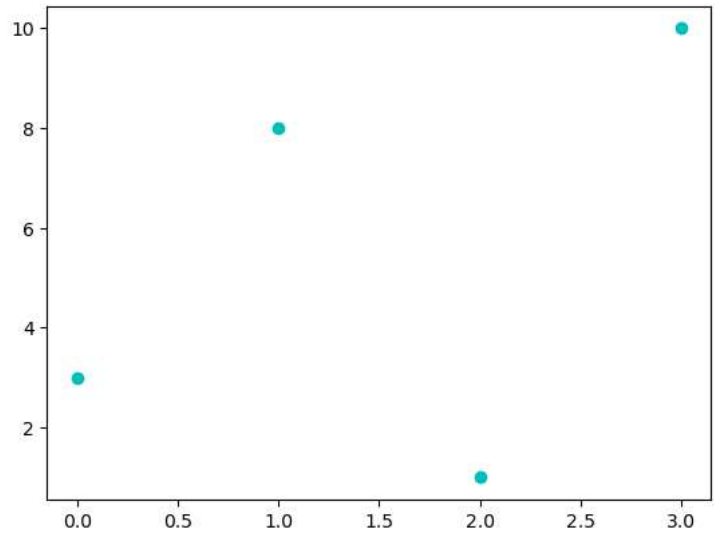
In [110]:
```
# Green - "g"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'og')
plt.show()
```



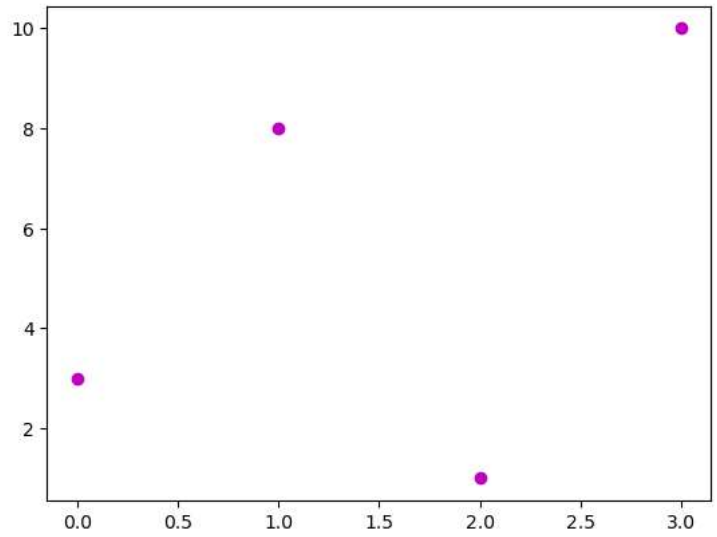
In [109]:
```
# Blue - "b"
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'ob')
plt.show()
```



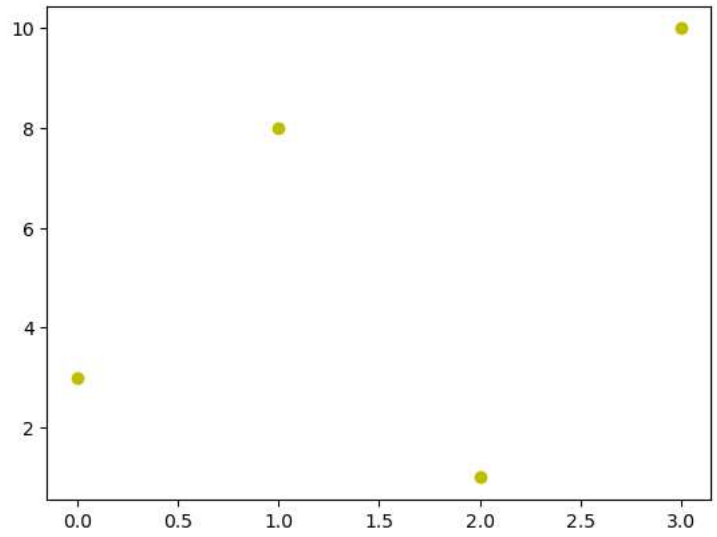
In [108]:
```
# Cyan - 'c'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'oc')
plt.show()
```



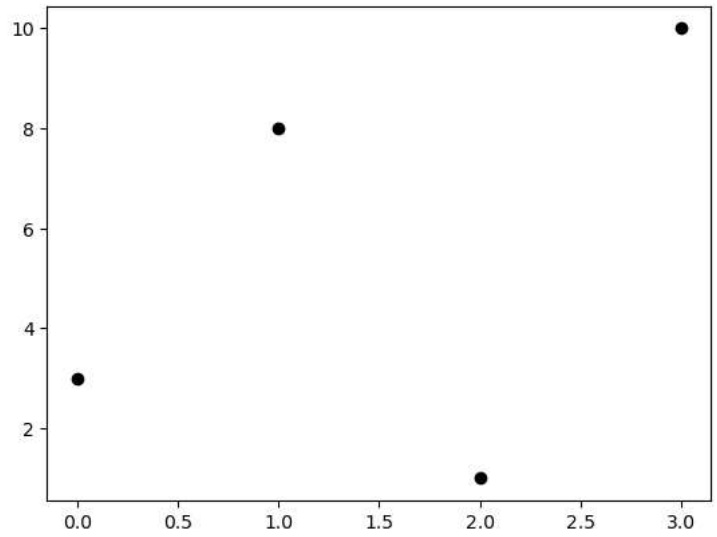
In [107]:
```
# Megenta - 'm'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'om')
plt.show()
```



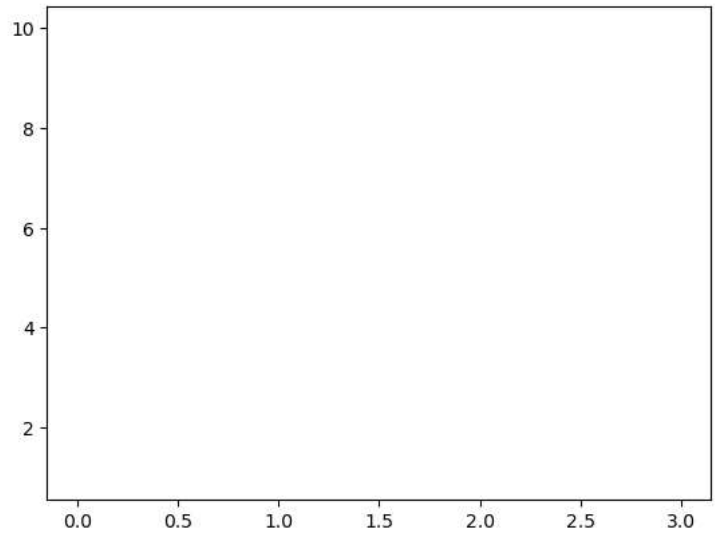
In [106]:
```
# Yellow - 'y'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'oy')
plt.show()
```



In [105]:
```
# Black - 'k'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'ok')
plt.show()
```



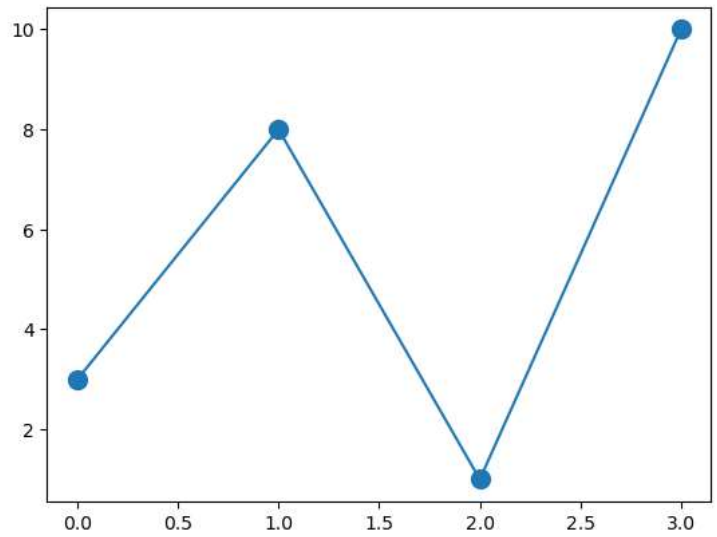
In [104]:
```
# White - 'w'
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, 'ow')
plt.show()
```



Marker Size

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

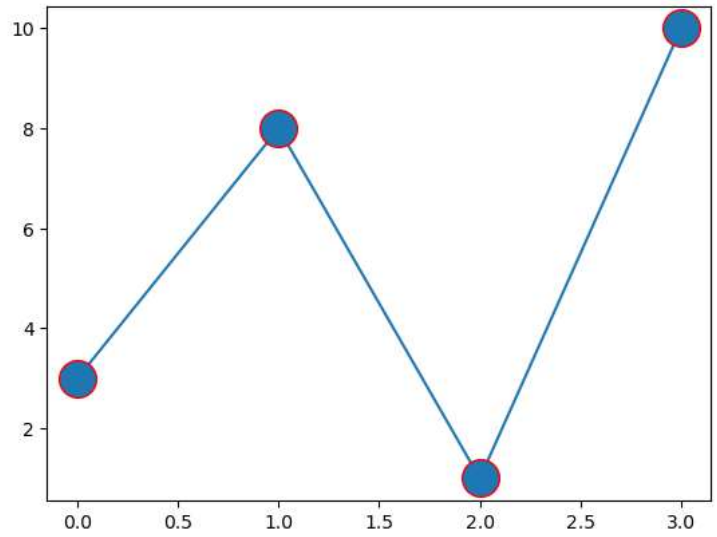
In [103]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker = 'o', ms = 10)
plt.show()
```



Marker Color

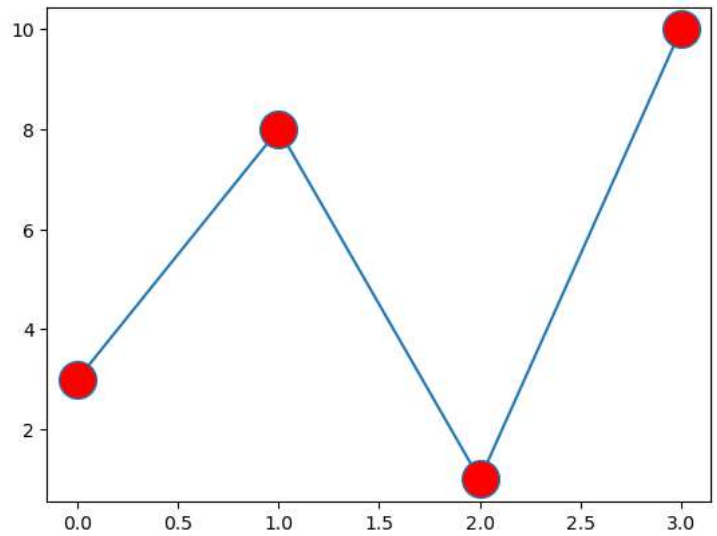
You can use the keyword argument"markeredgecolor"or the shorter"mec"to set the color of the edge of the markers:

In [102]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



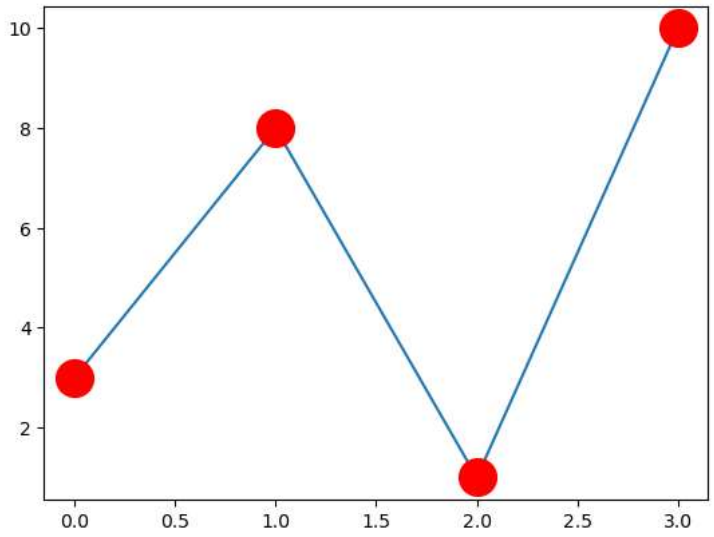
You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

In [101]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



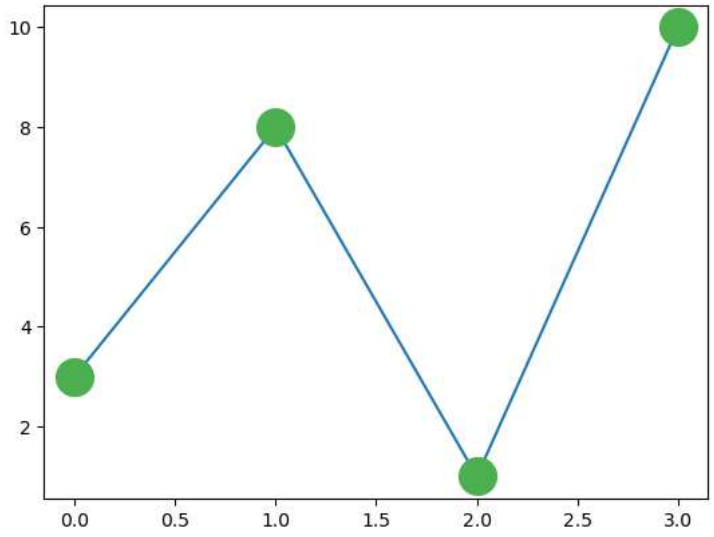
Use both the mec and mfc arguments to color the entire marker:

In [100]:
```
# Set the color of both the edge the face to red:
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', mfc = 'r')
plt.show()
```



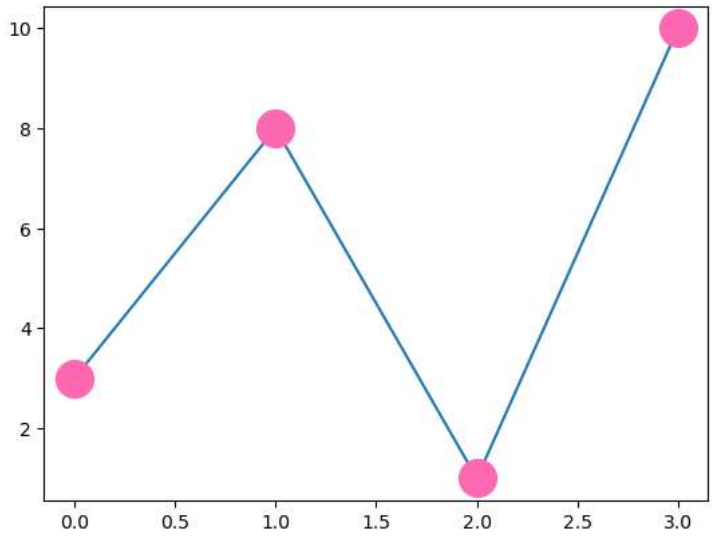
In [99]:
```
# Mark each point with a beautiful green color:
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
...
```

Out[99]:
```
[<matplotlib.lines.Line2D at 0x216fc8680a0>]
```



In [98]:
```
# Mark each point with the color named "hotpink":
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')
...
```

Out[98]: [<matplotlib.lines.Line2D at 0x216fc672a30>]

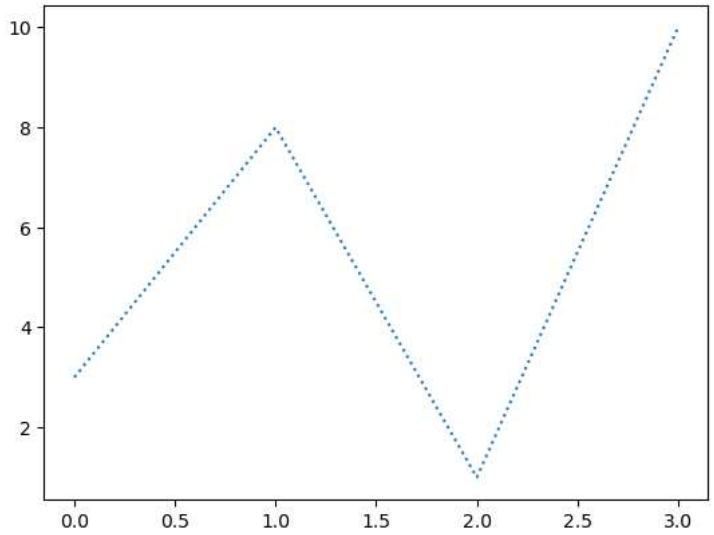


Matplotlib Line

Linestyle

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

In [97]:
```
# Use a dotted line:
import matplotlib.pyplot as plt
import numpy as np

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

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



In [ ]:
```
# Use a dashed line:

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

Shorter Syntax

The line style can be written in a shorter syntax:

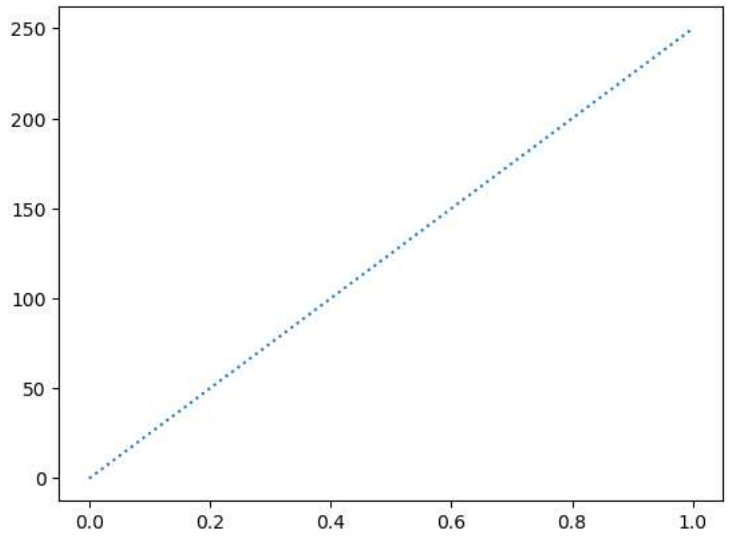
"linestyle" can be written as "ls."

"dotted"can be written as".."

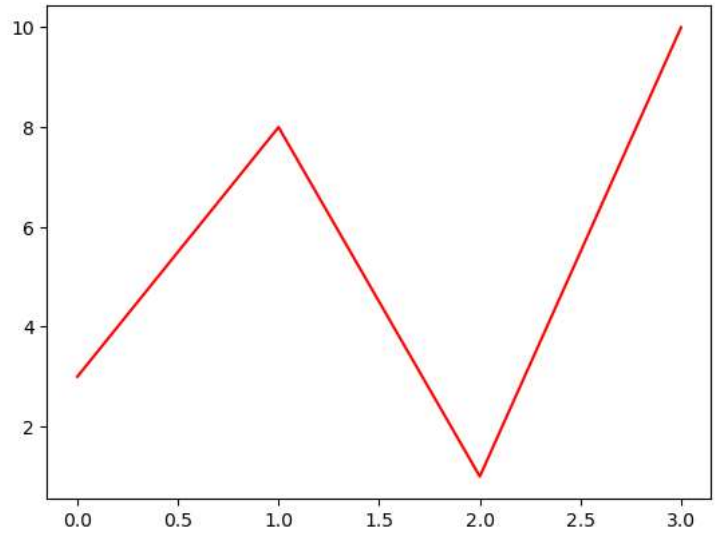
"dotted"can be written as"--."

In [148]:
```
# Shorter syntax:
plt.plot(ypoints, ls = ':')
```

Out[148]:
```
[<matplotlib.lines.Line2D at 0x216fdf3ac70>]
```

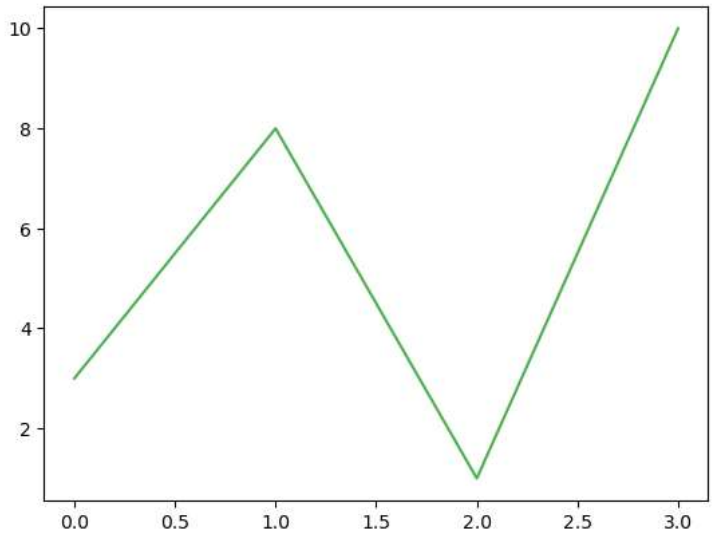


In [96]:
```
# Set the line color to red:
import matplotlib.pyplot as plt
import numpy as np

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

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



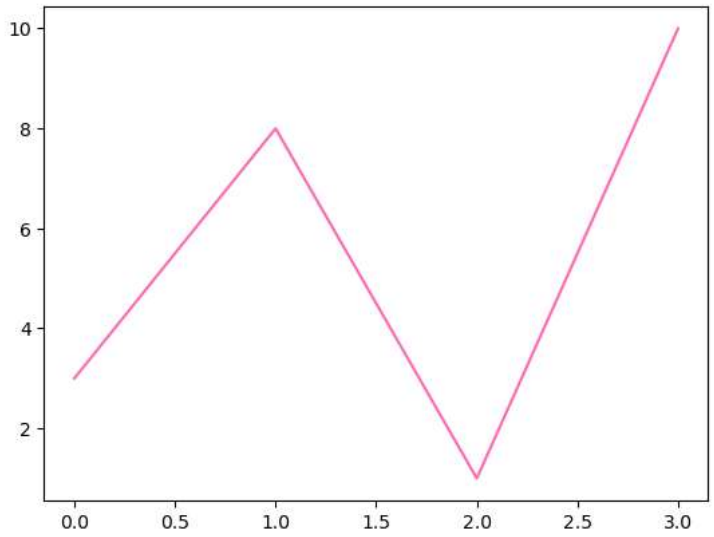
In [94]:
```
# plot with a beautiful green line:
...
plt.plot(ypoints, c = '#4CAF50')
...
```

Out[94]: [<matplotlib.lines.Line2D at 0x216fc9b3d00>]



In [95]:
```
# Plot with the color named "hotpink":
...
plt.plot(ypoints, c = 'hotpink')
...
```

Out[95]: [<matplotlib.lines.Line2D at 0x216fb331760>]

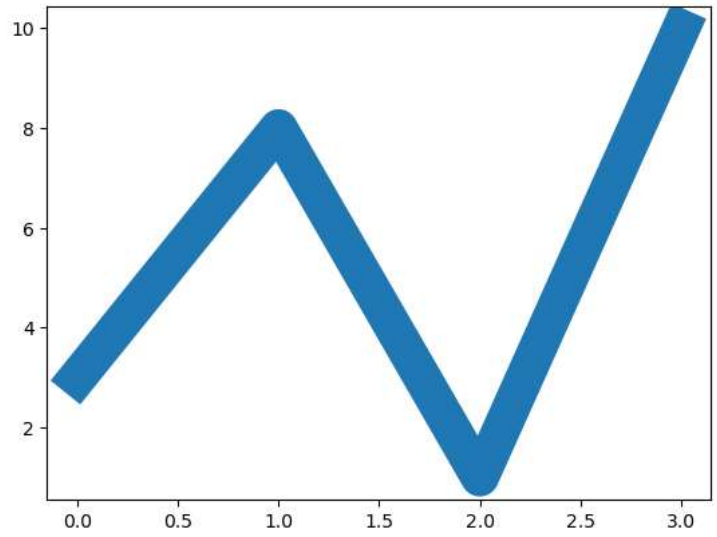


Line Width

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 [92]:
```
# Plot with a 20.5pt wide line;
import matplotlib.pyplot as plt
import numpy as np

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

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



import matplotlib.pyplot as plt

import numpy as np

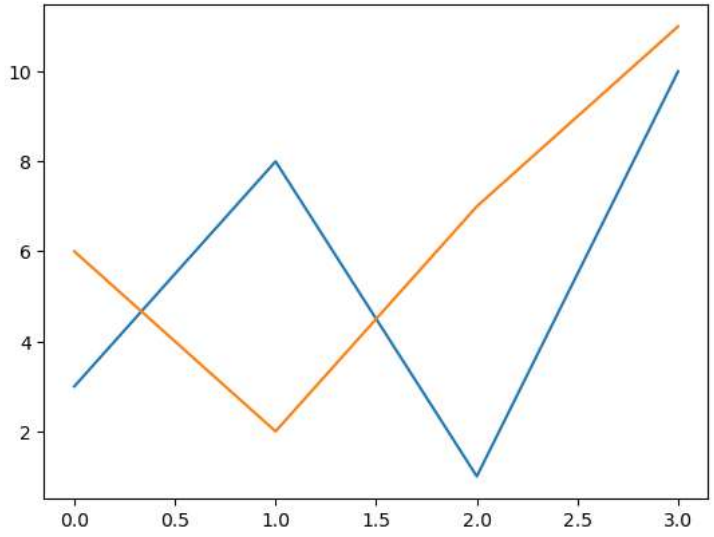
ypoints=np.array([3,8,1,10])

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 [91]:
```
# Draw two lines by specifying a plt.plot() function for each line:
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()
```

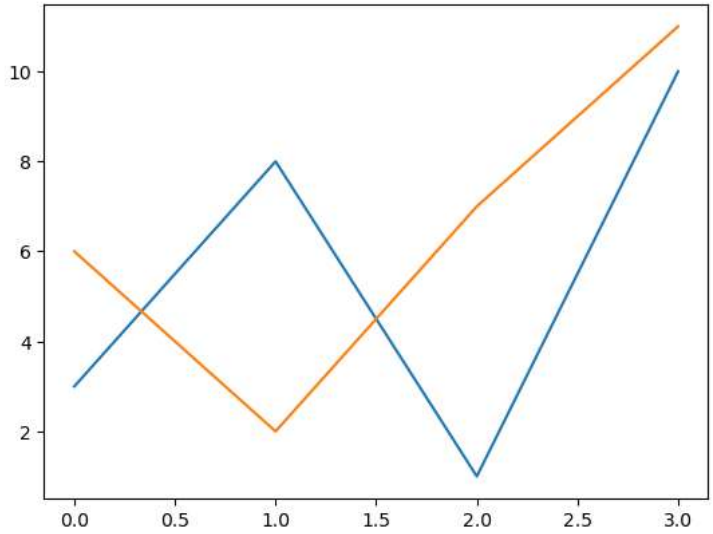


You can also plot many lines by adding the points for the x-and y-axis for each line in the same plt.plot()function.

(In the examples above we only specified the points on the y-axis,meaning that the points on the x-axis got the default values (0,1,2,3).)

The x-axis and y-values come in pairs:

In [90]:
```
# Draw two lines by speacifing 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()
```

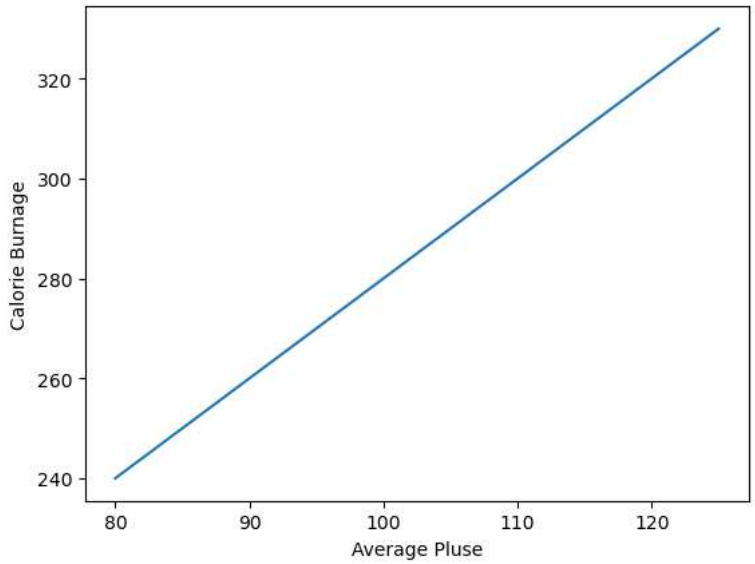


Matplotlib Lables and Title

Create Lables for a Plot

With Pyplot,you can use the xlable() and ylable() functions to set a lable for the x-and y-axis.

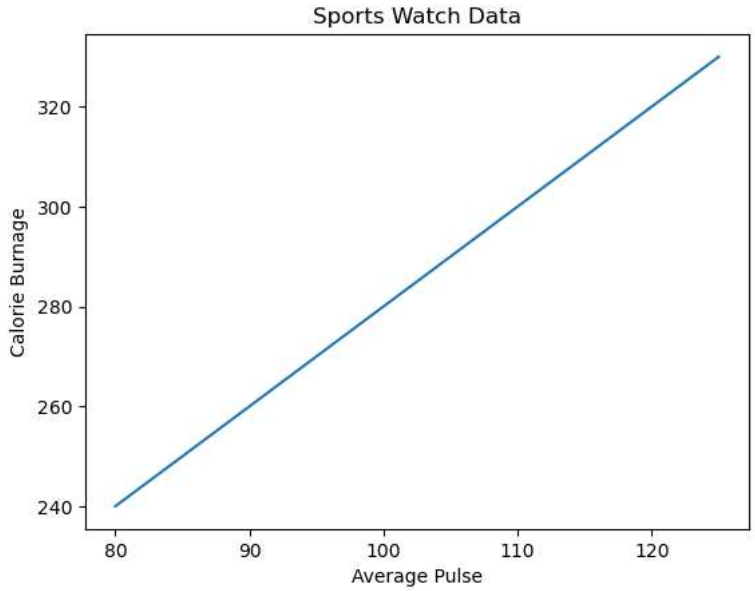
In [89]:
```
# Add lables to the x- and y-axis:
import numpy as np
import matplotlib.pyplot as plt

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

plt.plot(x, y)

plt.xlabel("Average Pluse")
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.

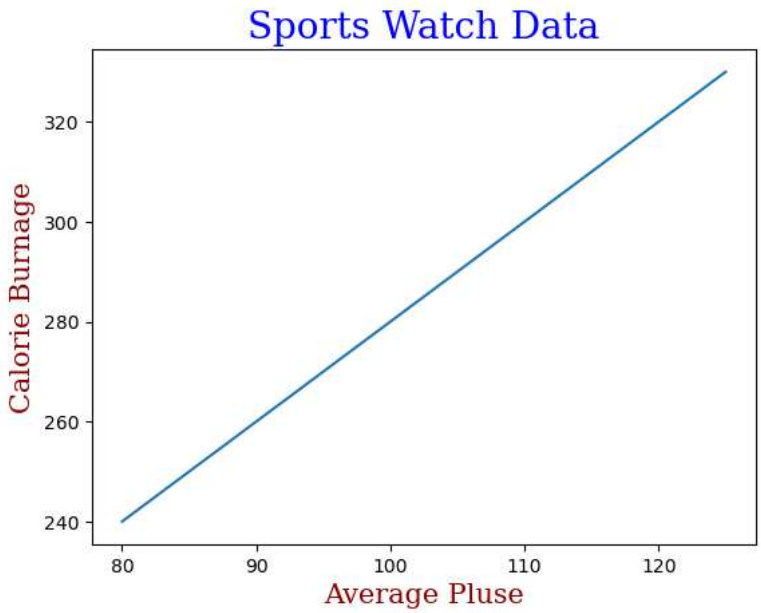
In [76]:
```
# Add a plot title and lables for the x- and y-axis:
import numpy as np
import matplotlib.pyplot as plt

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

plt.plot(x, y)

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

plt.show()
```



Set Font Properties for Title and Labels

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

In [77]:
```
# Set font properties for then title and lables:
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 Pluse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

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

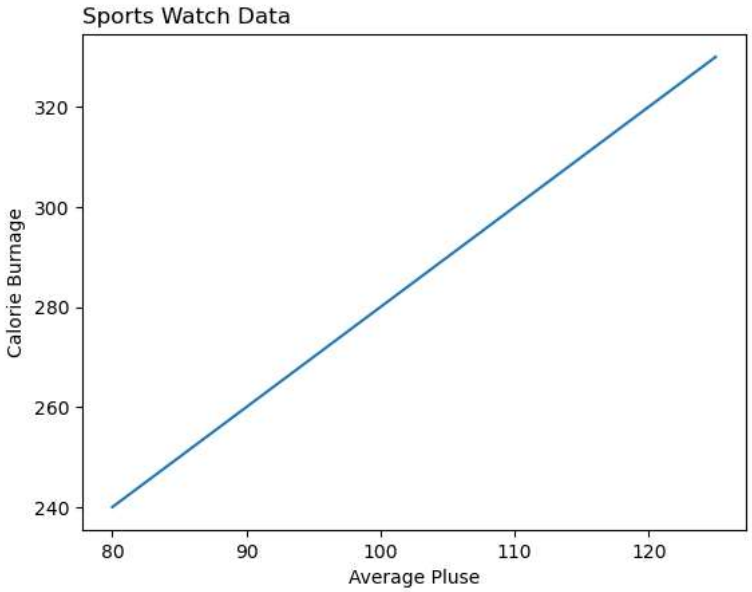


Position the Title

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

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

In [78]:
```
# Position the title to the left:
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 Pluse")
plt.ylabel("Calorie Burnage")

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

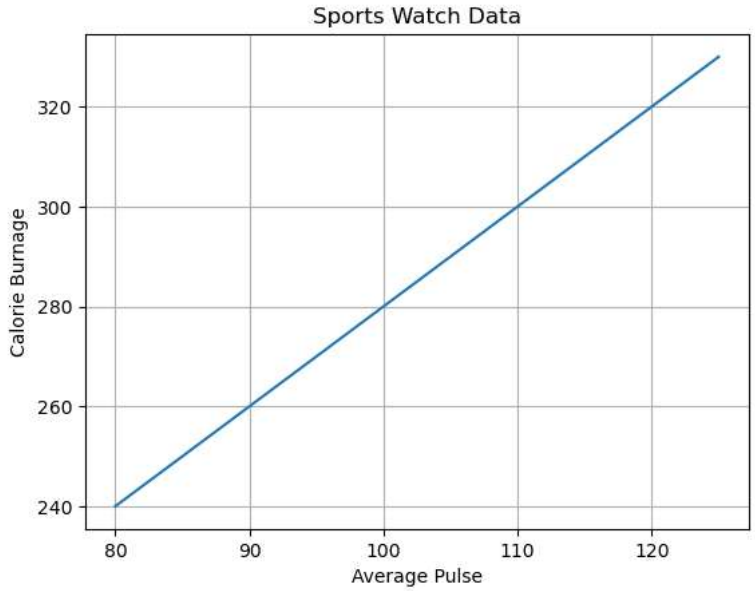


Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

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

In [79]:
```
# Add grid lines to the plot:
import numpy as np
import matplotlib.pyplot as plt

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

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

plt.plot(x, y)

plt.grid()

plt.show()
```

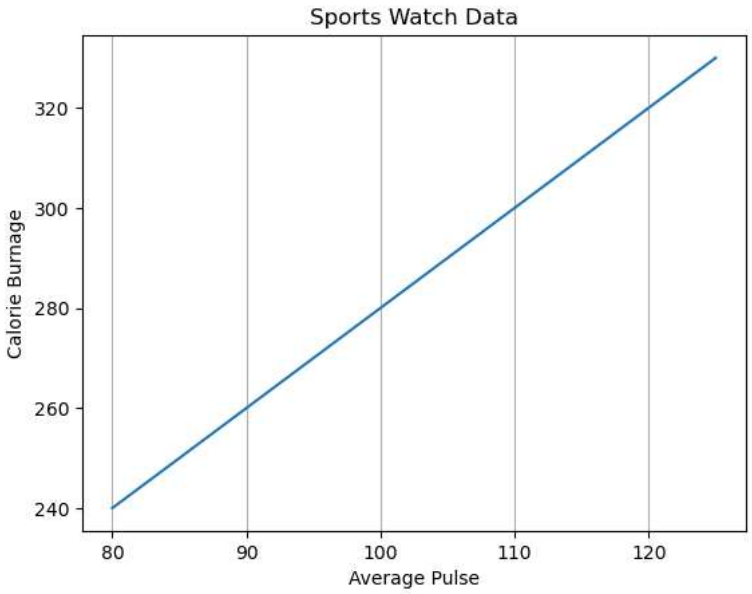


Specify Which Grid Lines to Display

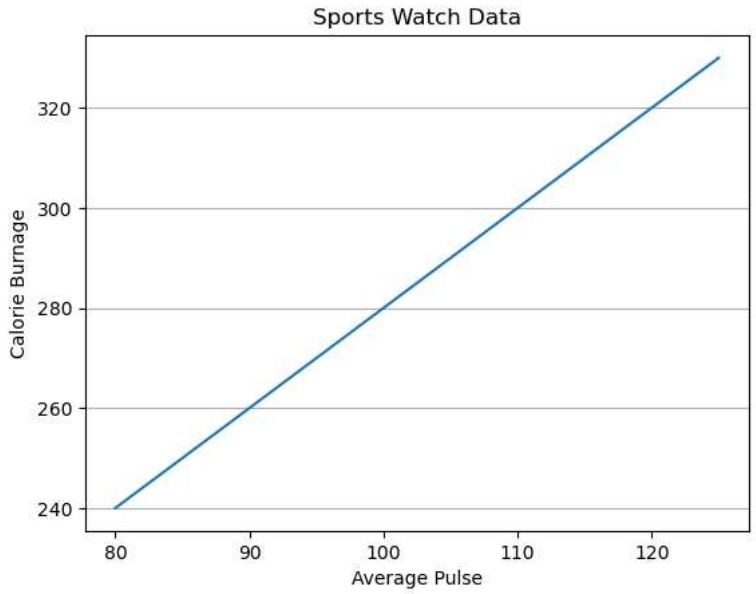
You can the axis parameter in the grid() function specify which grid lines to display.

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

In [87]:
```
# Display only grid lines for the x-axis:
import numpy as np
import matplotlib.pyplot as plt

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

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

plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```



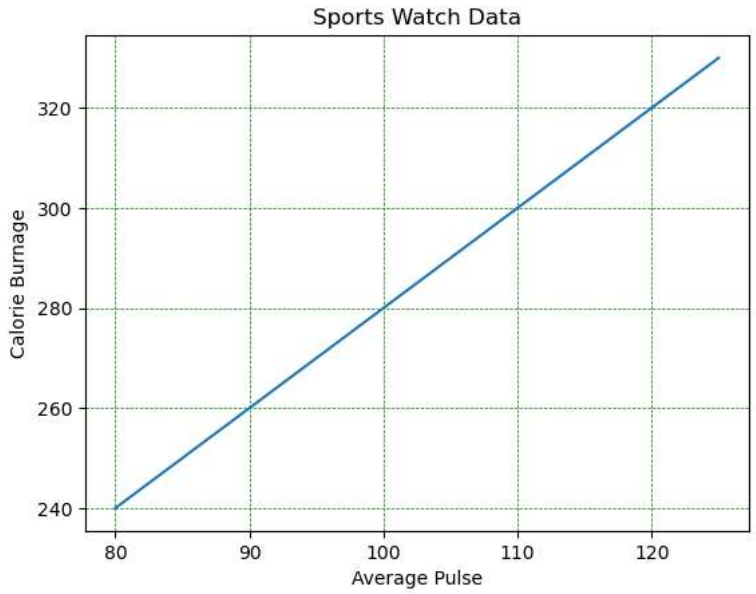
In [88]:
```
# Display only grid lines for the y-axis:
import numpy as np
import matplotlib.pyplot as plt

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

plt.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 [80]:
```
# Set the line properties of the grid:
import numpy as np
import matplotlib.pyplot as plt

x =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()
```

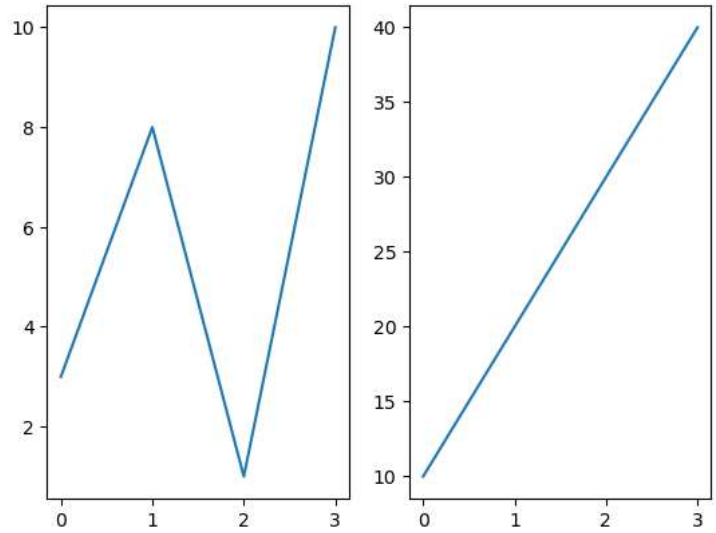


Matplotlib Subplot

Display Multiple Plots

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

In [81]:
```
# Draw 2 plots:
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()
```



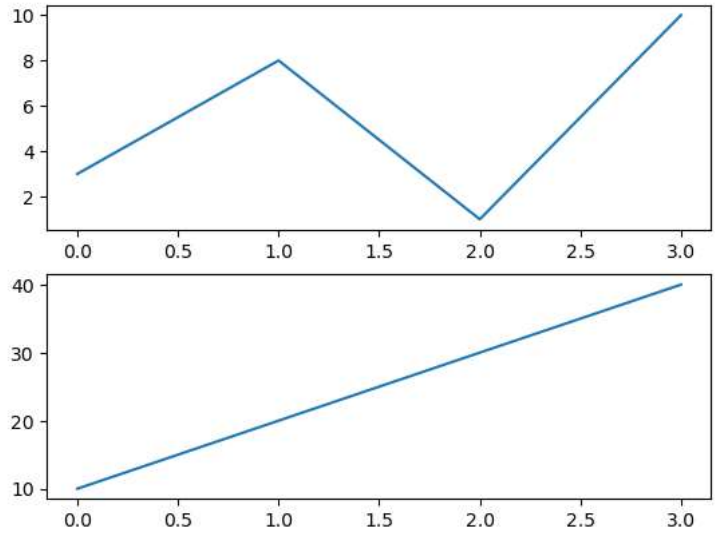
The Subplot() Function

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

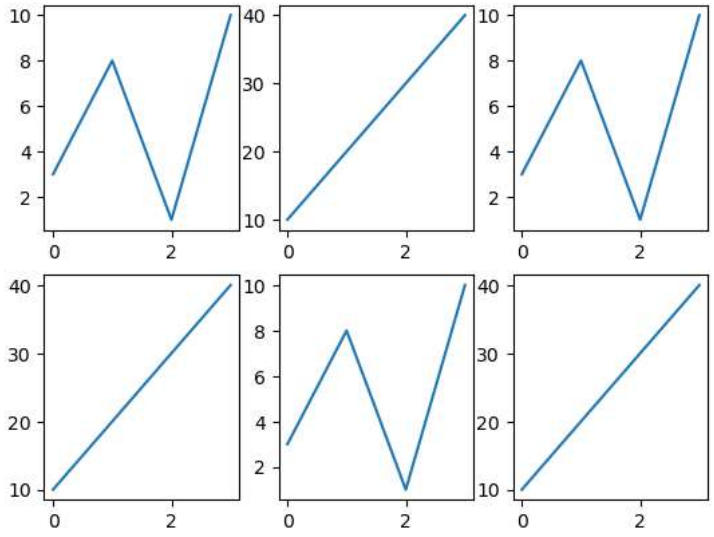
The layout is organized in rows and columns,whoch are represented by the first and second argument.

The third argument represents the index of the current plot.

In [82]:
```
# Draw 3 plots on top of each other:
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()
```



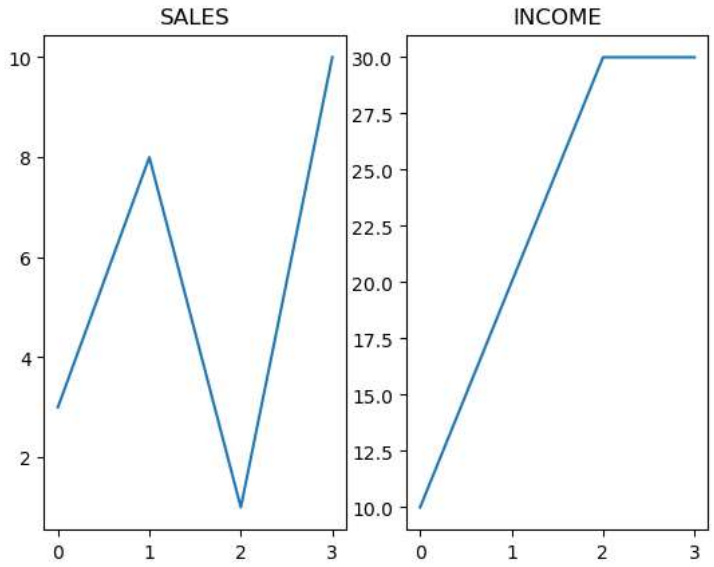
In [83]:
```
# Draw 6 plots:
import matplotlib.pyplot as plt
import numpy as np

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:

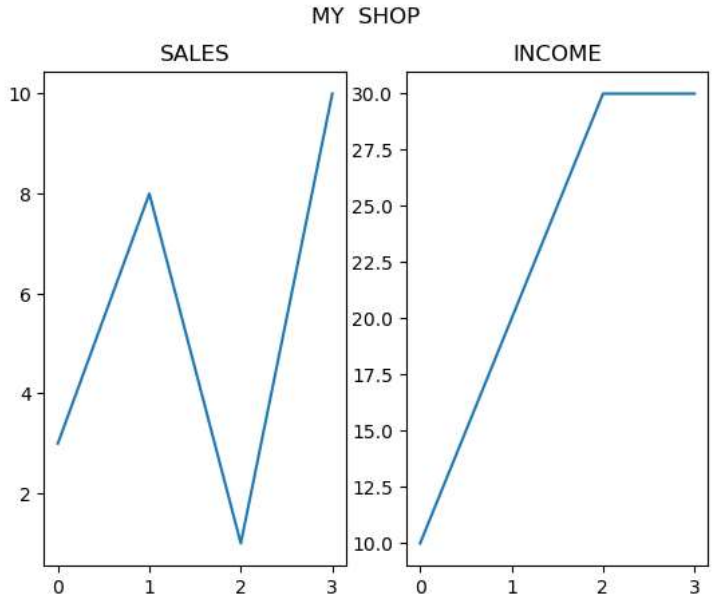
In [84]:
```
# 2 plots, with titles:
import matplotlib.pyplot as plt
import numpy as np

#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, 30])

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 [85]:
```
# Add a title for the entire figure:
import matplotlib.pyplot as plt
import numpy as np

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

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

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

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

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



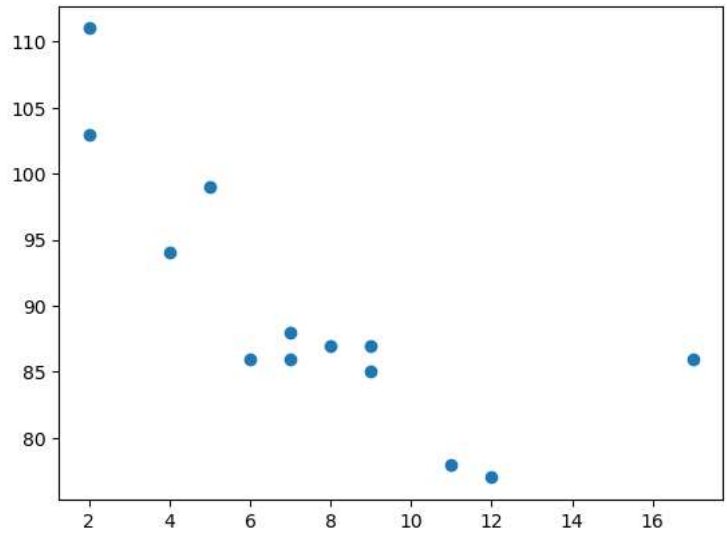
Matplotlib Scatter

Creating scatter plots

With Pyplot,you 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 the y-axis:

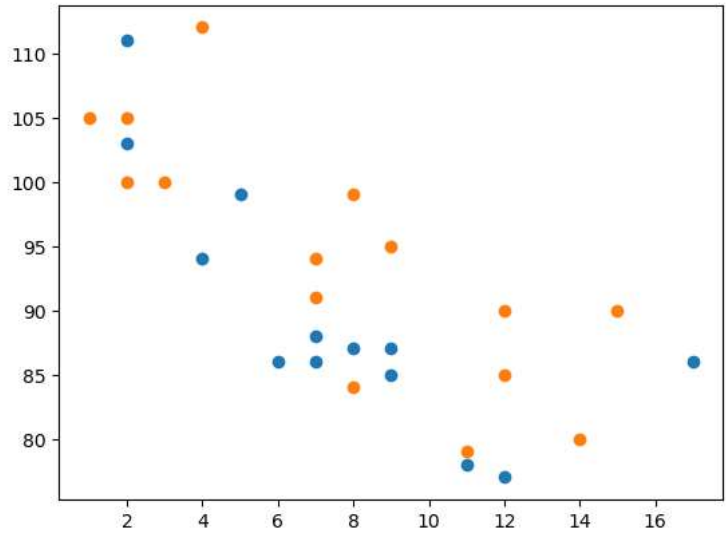
In [86]:
```
# A simple scatter plot:
import matplotlib.pyplot as plt
import numpy as np

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()
```



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?

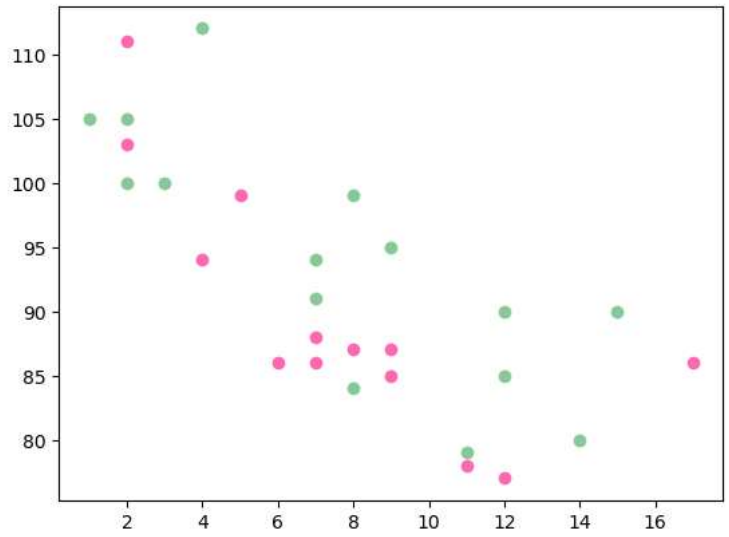
In [150]:
```
# Draw two plots on the same figure:
import matplotlib.pyplot as plt
import numpy as np

#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 [152]:
```
# Set your own color of the markers:
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')

#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, color = '#88c999')

plt.show()
```



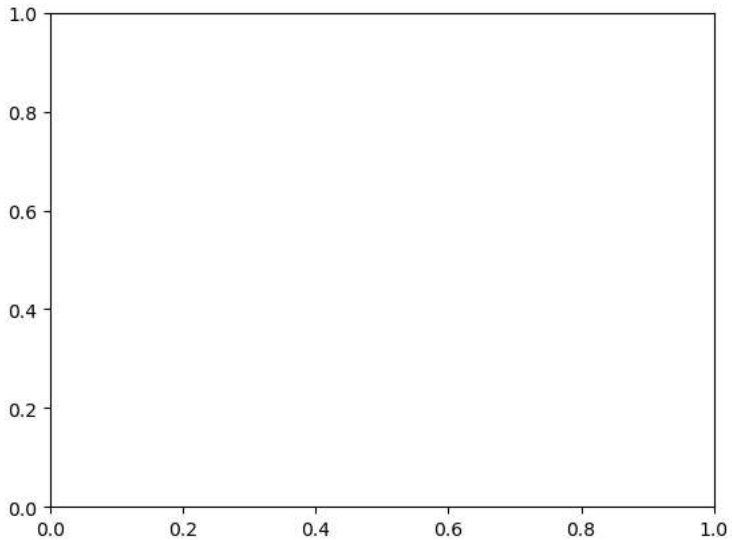
Color Each Dot

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 [160]:
```
# Set your own color of the markers:
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",])

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

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6484\2409236889.py in <module>
      7 colors = np.array(["red","green","blue","yellow","pink","black","orange","purple",])
      8
----> 9 plt.scatter(x, y, c=colors)
     10
     11 plt.show()

~\Downloads\anaconda\lib\site-packages\matplotlib\pyplot.py in scatter(x, y, s, c, marker, cmap, norm, vmin, vmax, alpha,
linewidths, edgecolors, plotnonfinite, data, **kwargs)
   2817         vmin=None, vmax=None, alpha=None, linewidths=None, *,
   2818         edgecolors=None, plotnonfinite=False, data=None, **kwargs):
-> 2819     __ret = gca().scatter(
   2820         x, y, s=s, c=c, marker=marker, cmap=cmap, norm=norm,
   2821         vmin=vmin, vmax=vmax, alpha=alpha, linewidths=linewidths,

~\Downloads\anaconda\lib\site-packages\matplotlib\__init__.py in inner(ax, data, *args, **kwargs)
   1410     def inner(ax, *args, data=None, **kwargs):
   1411         if data is None:
-> 1412             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1413
   1414         bound = new_sig.bind(ax, *args, **kwargs)

~\Downloads\anaconda\lib\site-packages\matplotlib\axes\_axes.py in scatter(self, x, y, s, c, marker, cmap, norm, vmin, vma
x, alpha, linewidths, edgecolors, plotnonfinite, **kwargs)
   4378             orig_edgecolor = kwargs.get('edgecolor', None)
   4379         c, colors, edgecolors = \
-> 4380             self._parse_scatter_color_args(
   4381                 c, edgecolors, kwargs, x.size,
   4382                 get_next_color_func=self._get_patches_for_fill.get_next_color)

~\Downloads\anaconda\lib\site-packages\matplotlib\axes\_axes.py in _parse_scatter_color_args(c, edgecolors, kwargs, xsize,
get_next_color_func)
   4228                     # NB: remember that a single color is also acceptable.
   4229                     # Besides *colors* will be an empty array if c == 'none'.
-> 4230                     raise invalid_shape_exception(len(colors), xsize)
   4231         else:
   4232             colors = None  # use cmap, norm after collection is created

ValueError: 'c' argument has 8 elements, which is inconsistent with 'x' and 'y' with size 13.
```



ColorMap

The Matplotlib module has a number of available colormaps.

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

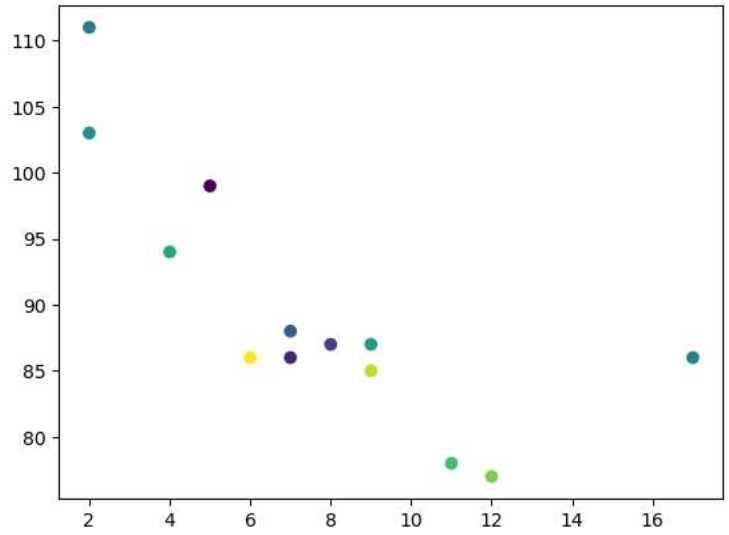
Here is an example of a colormap:

How to Use the ColorMap

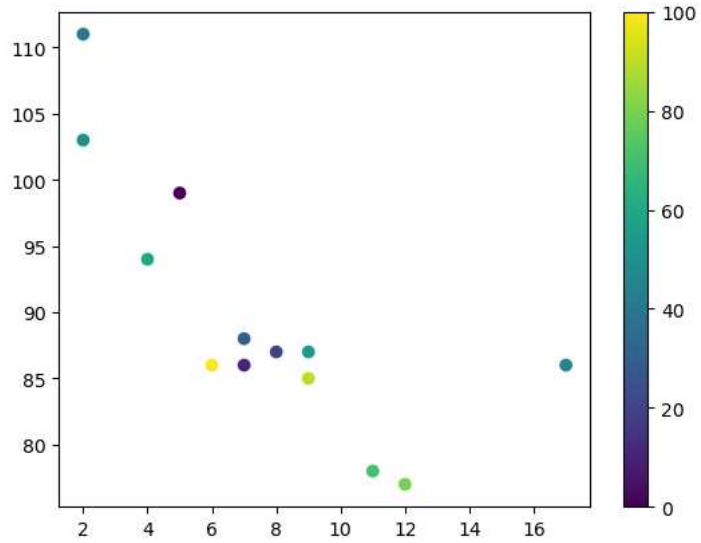
You can specify the colormap with the keyword argument cmap with the value of the colormap,in this case 'viridis'which is one of the built-in colormaps available in Matplotlib.

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

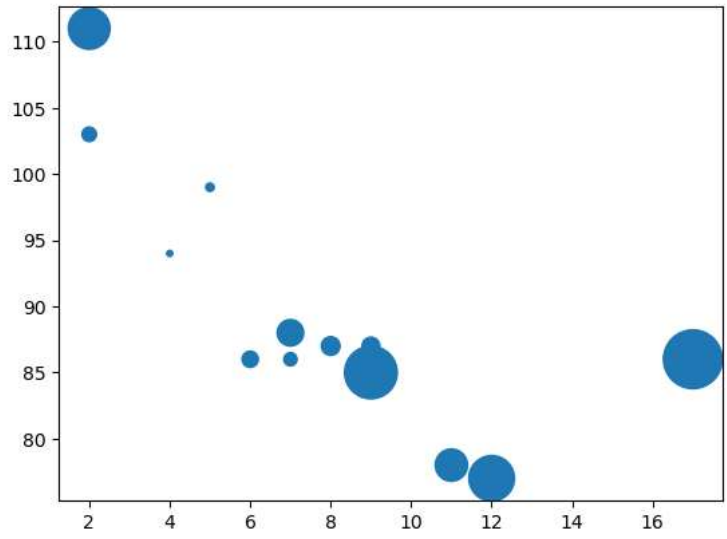
In [161]:
```
# Create a color array, and specify a colormap in the scatter plot:
import matplotlib.pyplot as plt
import numpy as np

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()
```



In [162]:
```
# Include the actual colormap:
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()
```



In [167]:
```
# Set your own size for the markers:
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()
```

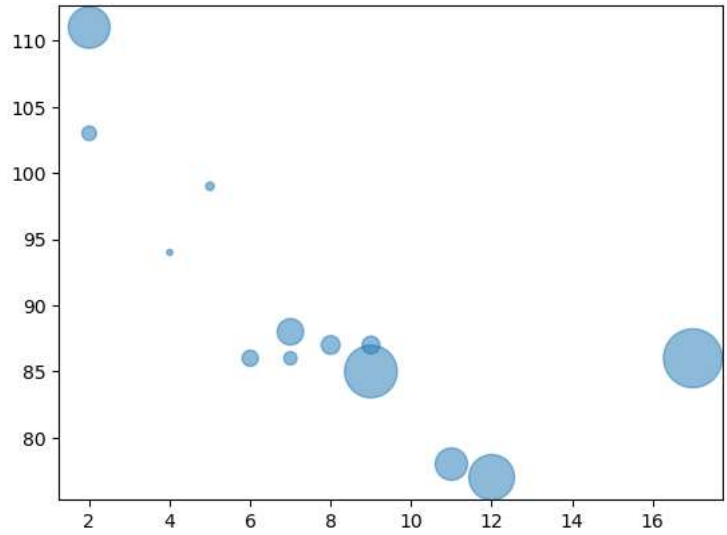


Alpha

You can adjust the transparency of the dots with the alpha argument.

Just like colors,make sure the array for sizes has the same lenght as the arrays for the x-and y-axis:

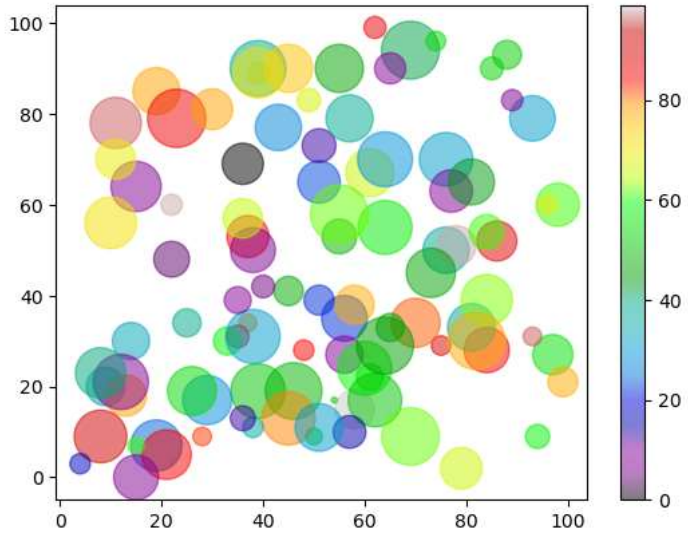
In [168]:
```
import matplotlib.pyplot as plt
import numpy as np


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 [169]:
```
# Create random arrays with 100 values for x-points, y-points, colors and size:
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()
```

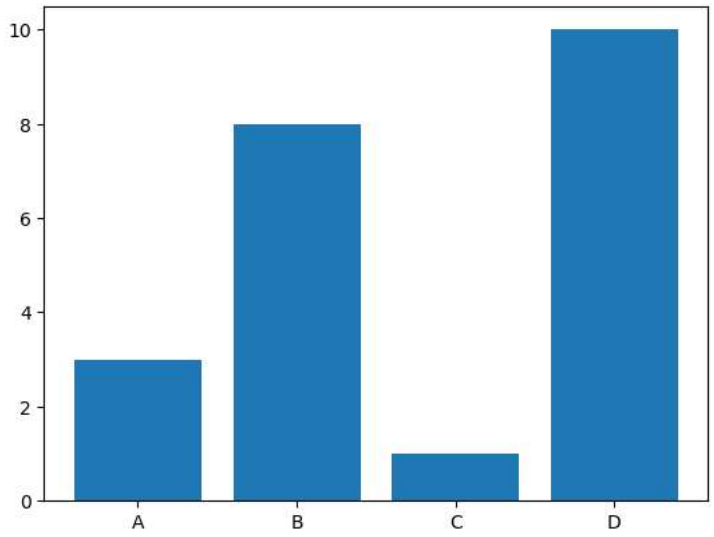


Matplotlib Bars

Creating Bars

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

In [12]:
```
# Draw 4 bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



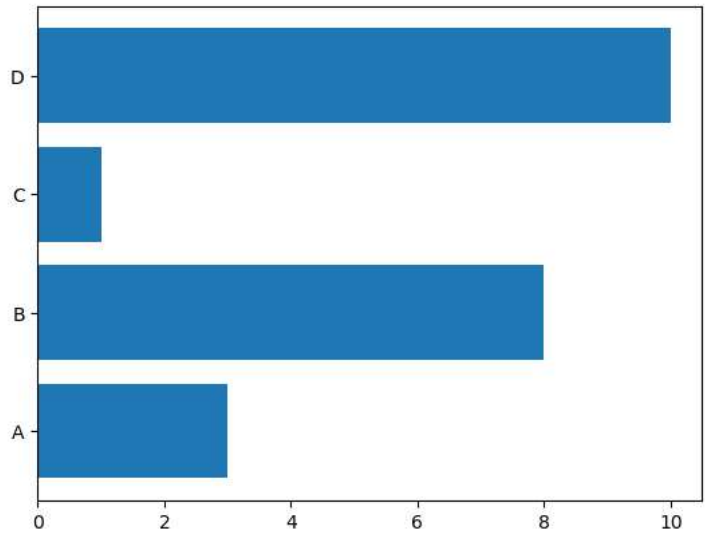
The bar() function takes arguments that describes the layout of the bars.

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

Horizontal Bars

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

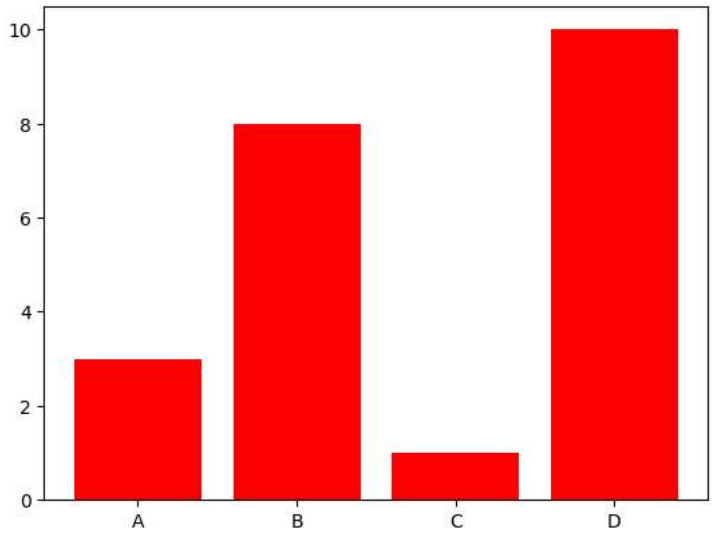
In [13]:
```
# Draw 4 horizontally bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



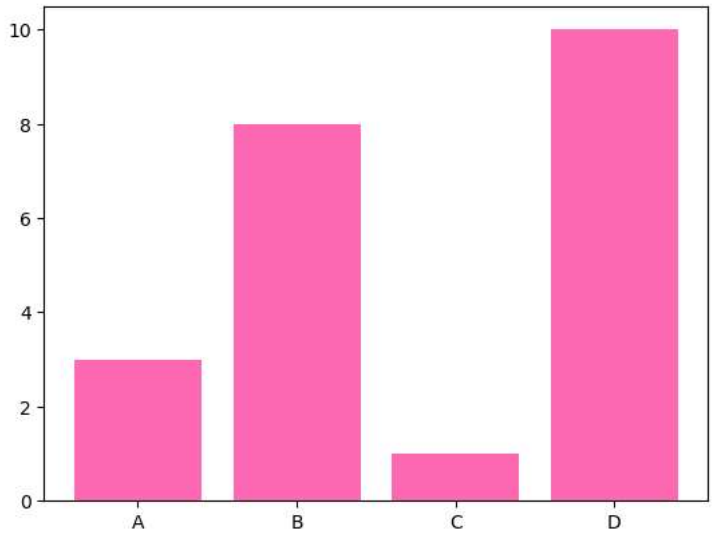
Bars Color

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

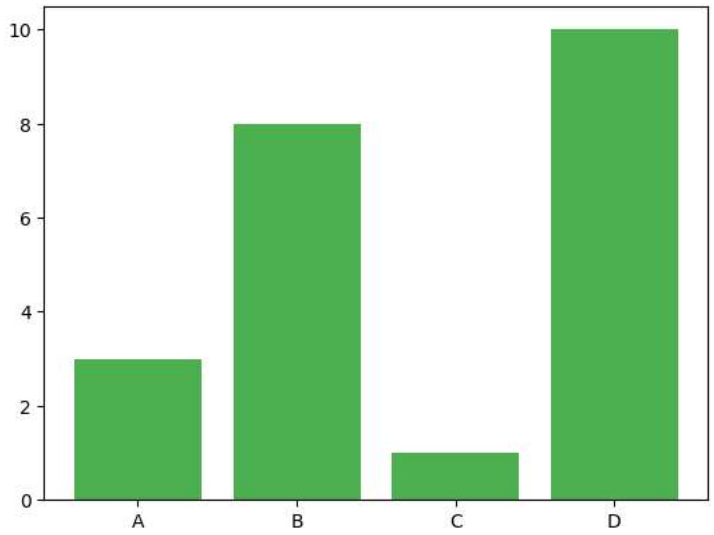
In [14]:
```
# Draw 4 red bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



In [16]:
```
# Draw 4 "hot pink" bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



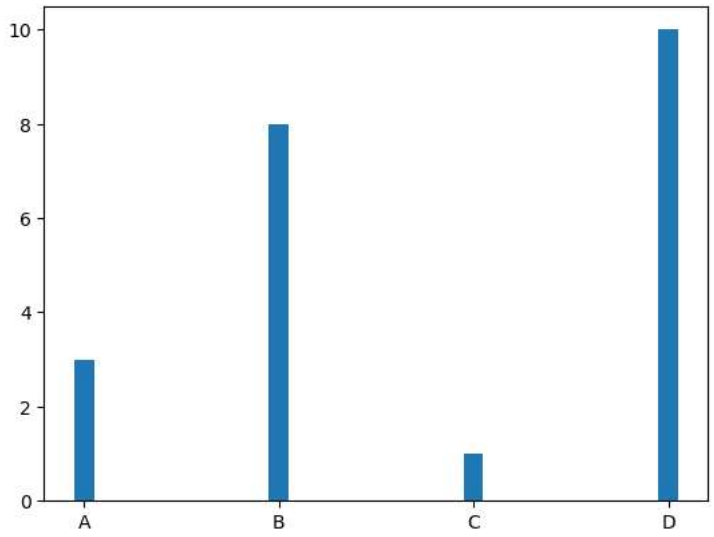
In [17]:
```
# Draw 4 bars with a beautiful green color:
import matplotlib.pyplot as plt
import numpy as np

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

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



Bar Width

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

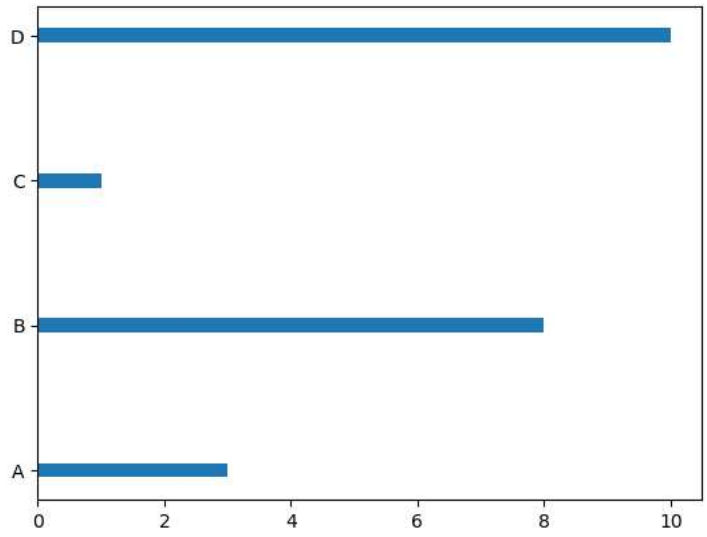
In [20]:
```
# Draw 4 very thin bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



Bar Height

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

In [23]:
```
# Draw 4 very thin bars:
import matplotlib.pyplot as plt
import numpy as np

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

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



Matplotlib Histograms

Histogram

A histogram is a graph showing frequency distributions.

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

Create Histogram

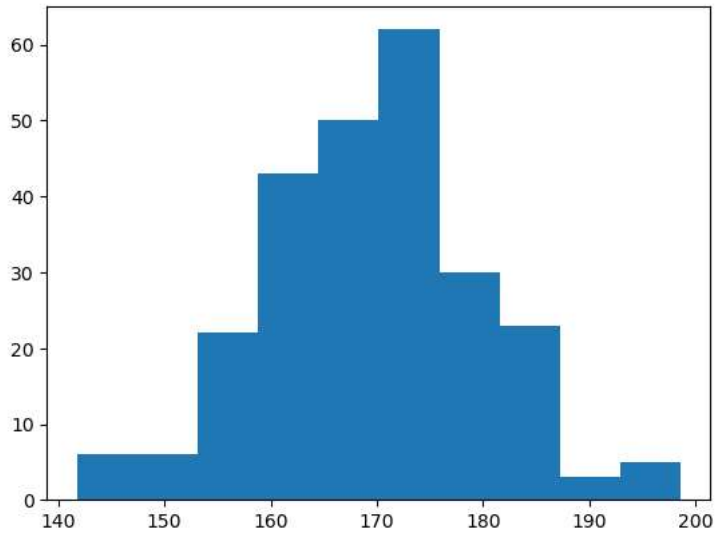
In Matplotlib, we use the hist() function to create histograms.

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

In [24]:
```
# A Normal Data Distribution by NumPy:
import numpy as np

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

print(x)
```

```
[191.95478251 158.86017605 164.83907536 182.01504866 172.41310645
 181.96376523 185.28836965 167.57242613 176.97362395 170.95435192
 149.68396486 167.19669106 171.47728472 173.96743272 189.33336489
 164.92271813 173.9948184  159.40646175 178.99558062 169.54673426
 161.01617758 169.54946188 171.72803132 174.60915955 156.63881329
 178.59000593 164.66306649 165.11059799 178.44955227 174.1603205
 173.56155087 169.78181467 160.28954415 159.19050926 179.91569402
 158.94144263 184.63227841 155.49032177 158.24077222 180.70493015
 147.9923401  160.53154009 170.35826314 176.89921947 174.97661067
 177.51108055 162.10697907 157.91134484 168.01065871 171.77571894
 170.38017579 160.82323463 180.77631172 163.82031023 163.36896765
 169.28773631 162.71049025 164.72773065 179.30467175 160.57238515
 157.30167273 169.30914869 162.55190767 190.15821008 161.0178416
 170.81175488 178.00198175 160.33664236 175.15871215 172.31217064
 174.19688385 167.29381149 167.47624931 181.10989825 185.32796835
 174.33535272 178.00073815 162.63884028 153.13776349 166.37108507
 163.28833349 177.66377478 164.80639113 173.56763422 173.45297427
 192.74620161 154.5714658  150.8537723  177.22732962 161.93599419
 177.82584634 182.56912449 172.13778782 193.1415103  174.36456919
 187.57278334 163.88317596 158.93096706 172.20336274 173.41251994
 169.03226619 188.71730744 175.26413507 181.24486843 176.50631096
 169.91899486 170.47929356 164.34162295 169.57700095 178.96749135
 179.33206354 187.1939063  163.79825969 172.48061172 187.63357473
 176.05062736 161.35108663 176.54806136 177.50754471 171.55991622
 165.93692104 166.51576163 185.81974267 166.17279212 161.63495633
 167.46148465 165.34067828 156.09267138 172.63337655 180.15101545
 165.26248465 146.08121859 148.23014619 161.2353774  169.79951482
 168.51751376 163.1948338  165.58430604 164.11998135 156.96261108
 168.82282776 166.48915646 178.50461853 154.10385574 161.81778298
 172.37912131 161.67697677 168.8770529  159.26699349 179.15295866
 181.51913461 170.27738358 173.53109406 175.37887349 183.07626095
 170.77620111 160.81100268 148.94555963 175.09123004 173.85252678
 174.13281945 160.31901552 172.19919032 171.00152367 173.63800134
 164.32265849 171.39650877 148.68081336 164.68556025 166.0709538
 170.49457427 160.0593053  172.83119215 177.36862559 163.99891788
 173.1440069  155.39522746 165.7645002  175.35803676 161.1681783
 165.96976406 158.66902711 167.26207881 167.65571554 175.47827889
 166.55286412 176.29538303 192.06591094 160.9884832  166.85600199
 190.41103902 163.18602063 158.27855883 169.86439756 175.40189795
 163.00818899 184.91698571 180.97290253 182.29286962 168.004567
 164.40306037 164.31434257 177.20360618 174.48243849 179.18552514
 171.13871865 183.99364612 168.31860306 178.58514549 181.52058492
 169.41679386 187.29179814 160.79527914 160.84427676 177.9409443
 177.99418001 170.93073378 174.31543463 180.6979835  144.63595095
 199.72295847 165.76115618 145.80937086 177.26729382 166.26788267
 167.99879062 169.79401631 170.9873108  159.92515493 180.53406092
 158.11828889 155.06507076 175.30122911 178.64260194 184.01437264
 157.58579455 160.63628985 169.47022354 173.74346356 169.77406393
 159.42776223 175.08066623 180.63827987 181.43890889 184.12736587
 170.83534774 151.14827852 166.14974591 173.60737864 178.5723919 ]
```

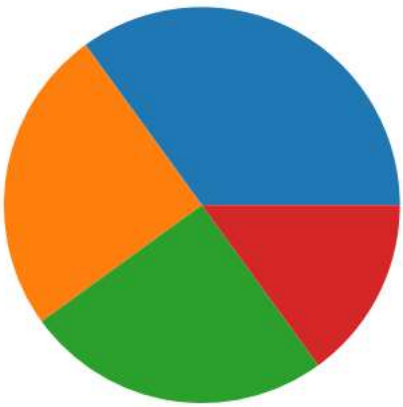
In [26]:
```
# The hist() function will read the array and produce a histogram:
#A simple histogram:
import matplotlib.pyplot as plt
import numpy as np

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

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



Matplotlib Pie Charts

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

In [27]:
```
# A simple pie chart:
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]).

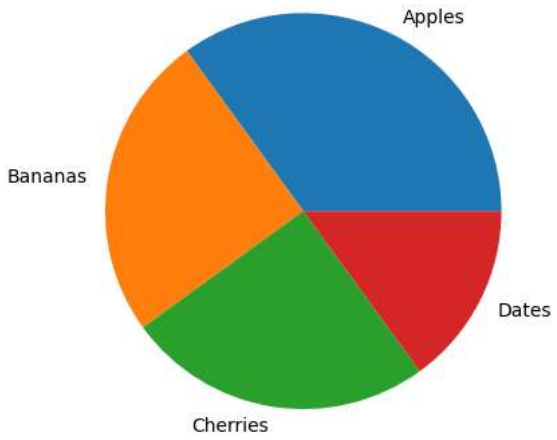
By dafault the plotting of the first wedge starts from the x-axis and moves counterclockwise:

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 [30]:
```
# A Simple pie chart:
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()
```

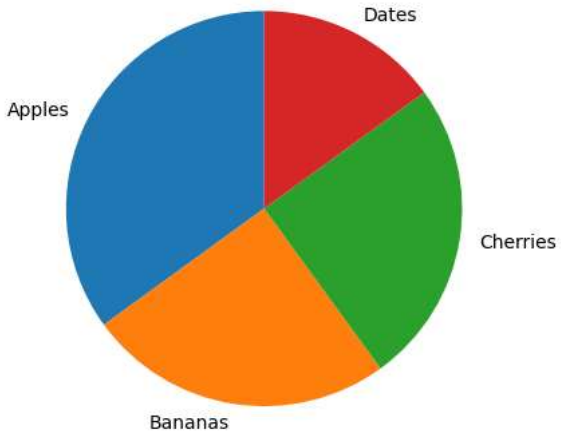


Start Angle

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 [4]:
```
# Start the first wedge at 90 degrees:
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()
```



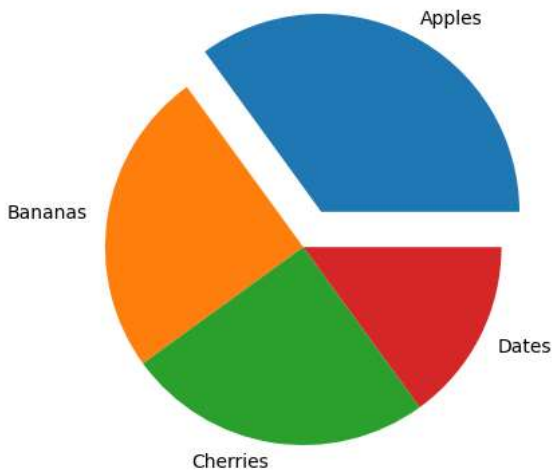
Explode

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 array with one value for each wedge.

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

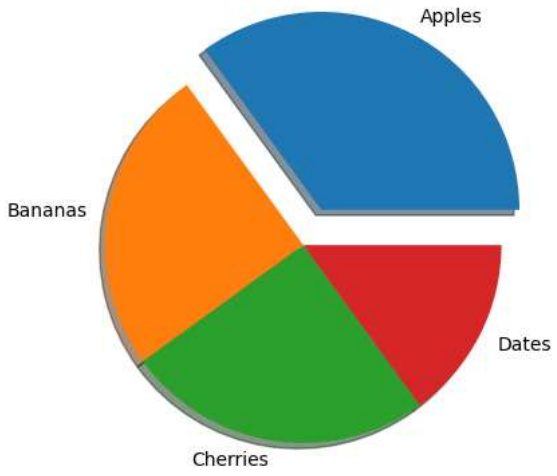
In [5]:
```
# Pill the "Apples" wedge 0.2 from the center of the pie:
import matplotlib.pyplot as plt
import numpy as np

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 [6]:
```
# Add a shadow:
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()
```

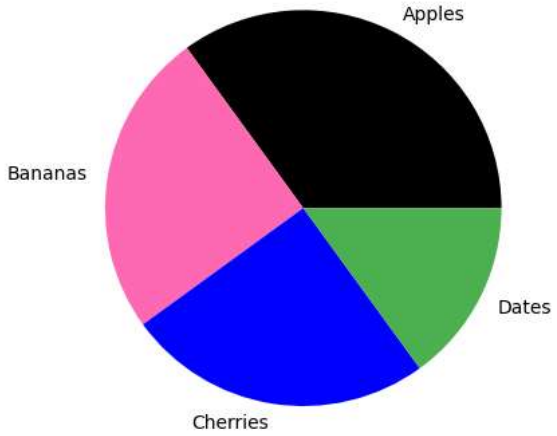


Colors

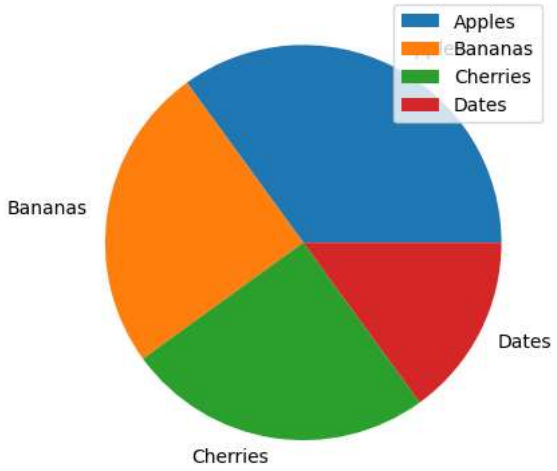
You can set the color of each wedge with the colors parameter.

The colors parameter, if specified, much be an array with one value for each wedge:

In [7]:
```
# Specify a new color for each wedge:
import matplotlib.pyplot as plt
import numpy as np


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

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



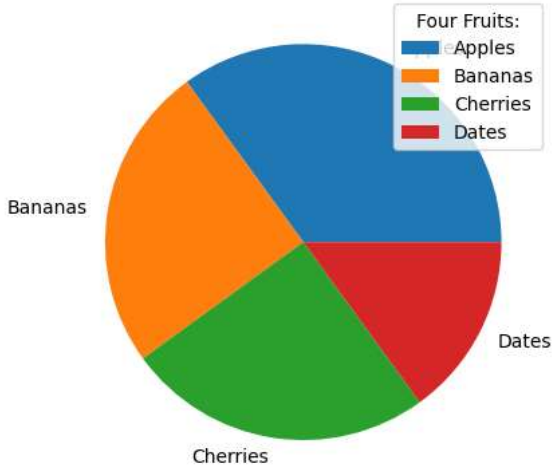
In [8]:
```
# Add a Legend:
import matplotlib.pyplot as plt
import numpy as np


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 ledgend function.

In [10]:
```
# Add a Legend with a header:
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 [ ]:
```

```