Numpy array vs List

consumes less memory.

fast as compared to the python List.

convenient to use.

```
# importing numpy package
import numpy as np

# importing system module
import sys

# declaring a list of 1000 elements
S= range(1000)

# printing size of each element of the list
print("Size of each element of list in bytes: ",sys.getsizeof(S))

# printing size of the whole list
print("Size of the whole list in bytes: ",sys.getsizeof(S)*len(S))

# declaring a Numpy array of 1000 elements
D= np.arange(1000)

# printing size of each element of the Numpy array
print("Size of each element of the Numpy array in bytes: ",D.itemsize)

# printing size of the whole Numpy array
print("Size of the whole Numpy array in bytes: ",D.size*D.itemsize)

```

```
Size of each element of list in bytes:  48
Size of the whole list in bytes:  48000
Size of each element of the Numpy array in bytes:  8
Size of the whole Numpy array in bytes:  8000
```

```
# importing required packages
import numpy
import time

# size of arrays and lists
size = 1000000

# declaring lists
list1 = range(size)
list2 = range(size)

# declaring arrays
array1 = numpy.arange(size)
```

```
array2 = numpy.arange(size)

# capturing time before the multiplication of Python lists
initialTime = time.time()

# multiplying elements of both the lists and stored in another list
resultantList = [(a * b) for a, b in zip(list1, list2)]

# calculating execution time
print("Time taken by Lists to perform multiplication:",
    (time.time() - initialTime),
    "seconds")

# capturing time before the multiplication of Numpy arrays
initialTime = time.time()

# multiplying elements of both the Numpy arrays and stored in another Numpy array
resultantArray = array1 * array2

# calculating execution time
print("Time taken by NumPy Arrays to perform multiplication:",
    (time.time() - initialTime),
    "seconds")

```

```
Time taken by Lists to perform multiplication: 0.1521763801574707 seconds
Time taken by NumPy Arrays to perform multiplication: 0.003917694091796875 seconds
```

Copies & Views

Contents are physically stored in another location, it is called Copy. A different view of the same memory content is provided, we call it as View.

No copy

it uses the same id() of the original array to access it. Any changes in either gets reflected in the other.

```
import numpy as np
a = np.arange(6)

print ('Our array is:',a )

print ('Applying id() function:',id(a))

print ('a is assigned to b:')
b = a
print (b)
print ('b has same id():')
print (id(b))
print ('Change shape of b:')
```

```
b.shape = 3,2
print (b)

print ('Shape of a also gets changed:')
print (a)
```

```
Our array is: [0 1 2 3 4 5]
Applying id() function: 139928273240208
a is assigned to b:
[0 1 2 3 4 5]
b has same id():
139928273240208
Change shape of b:
[[0 1]
 [2 3]
 [4 5]]
Shape of a also gets changed:
[[0 1]
 [2 3]
 [4 5]]
```

View or Shallow Copy a new array object that looks at the same data of the original array. Unlike the earlier case, change in dimensions of the new array doesn't change dimensions of the original.

```
import numpy as np
# To begin with, a is 3X2 array
a = np.arange(6).reshape(3,2)

print ('Array a:',a)
print ('Create view of a:')
b = a.view()
print (b)
print ('id() for both the arrays are different:')
print ('id() of a:')
print (id(a))
print ('id() of b:')
print (id(b))

# Change the shape of b. It does not change the shape of a
b.shape = 2,3
b[0,0]=100

print (b,b.shape)

print (a,a.shape)
```

```
Array a: [[0 1]
 [2 3]
 [4 5]]
Create view of a:
[[0 1]
 [2 3]
 [4 5]]
```

```
id() for both the arrays are different:
id() of a:
139928273188880
id() of b:
139928273005424
[[100   1   2]
 [  3   4   5]] (2, 3)
[[100   1]
 [  2   3]
 [  4   5]] (3, 2)
```

Deep Copy

It is a complete copy of the array and its data, and doesn't share with the original array.

```
import numpy as np
a = np.array([[10,10], [2,3], [4,5]])

print ('Array a is:' ,a)
print ('Create a deep copy of a:' )
b = a.copy()
print ('Array b is:',b)
#b does not share any memory of a
print ('Can we write b is a')
print (b is a)

print ('Change the contents of b:')
b[0,0] = 100

print ('Modified array b:')
print ('a remains unchanged:',a)
```

```
Array a is: [[10 10]
 [ 2  3]
 [ 4  5]]
Create a deep copy of a:
Array b is: [[10 10]
 [ 2  3]
 [ 4  5]]
Can we write b is a
False
Change the contents of b:
Modified array b:
a remains unchanged: [[10 10]
 [ 2  3]
 [ 4  5]]
```

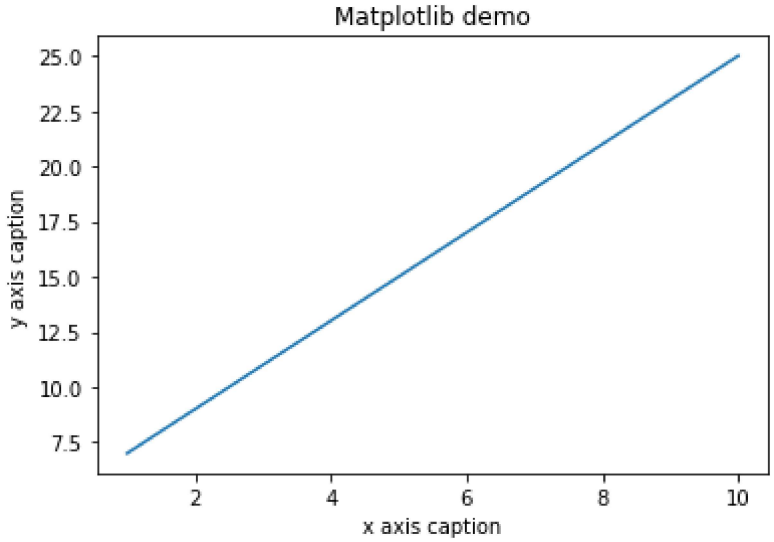
NumPy - Matplotlib

Matplotlib module was first written by John D. Hunter. Since 2012, Michael Droettboom is the principal developer. Currently, Matplotlib ver. 1.5.1 is the stable version available.

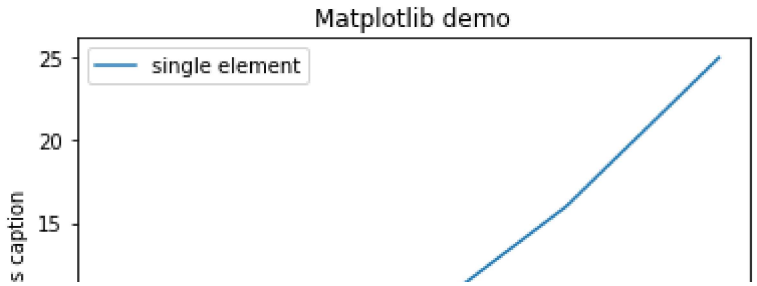
```
from matplotlib import pyplot as plt
```

```
x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
#plt.show()
```

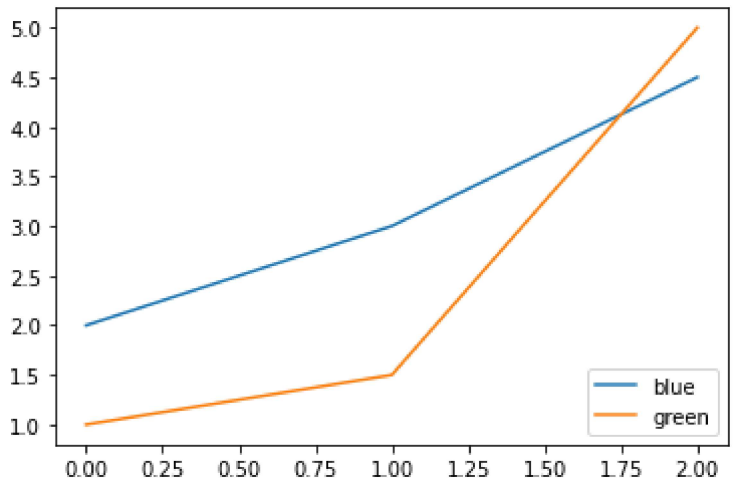
```
[<matplotlib.lines.Line2D at 0x7f43950d3490>]
```



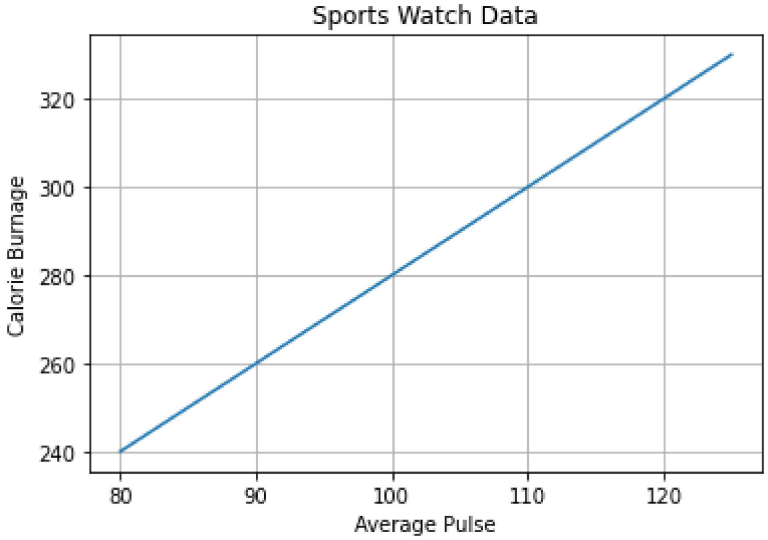

```
# X-axis values
x = [1, 2, 3, 4, 5]

# Y-axis values
y = [1, 4, 9, 16, 25]

# Function to plot
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x, y)

# Function add a legend
plt.legend(['single element'])

# function to show the plot
plt.show()
```



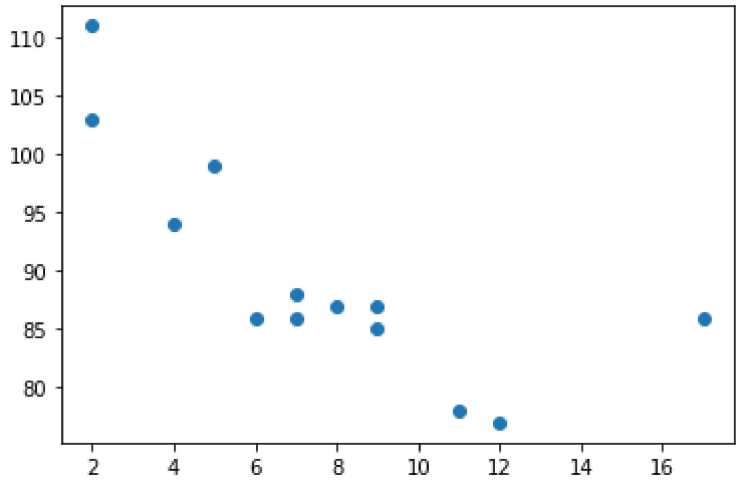

```
# Y-axis values
y1 = [2, 3, 4.5]

# Y-axis values
y2 = [1, 1.5, 5]

# Function to plot
plt.plot(y1)
plt.plot(y2)

# Function add a legend
plt.legend(["blue", "green"], loc ="lower right")

# function to show the plot
plt.show()
```



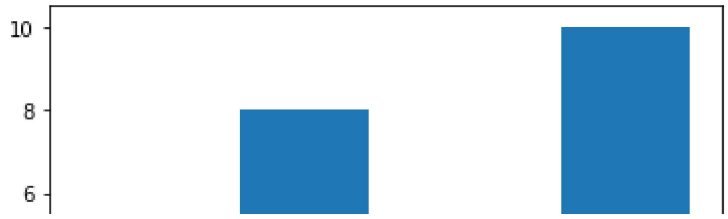

```
import numpy as np
import matplotlib.pyplot as plt

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()
```




```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
5 y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
6
7 plt.scatter(x, y)
8 plt.show()
```

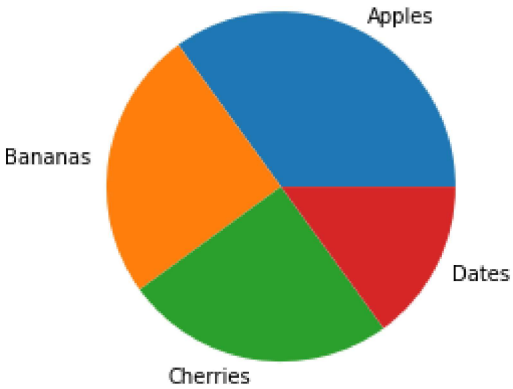



```
1
2 x = np.array(["A", "B", "C", "D"])
3 y = np.array([3, 8, 1, 10])
4
5 plt.bar(x,y)
6 plt.show()
```

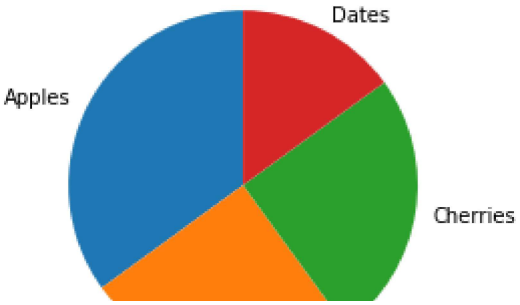



```
1 y = np.array([35, 25, 25, 15])
2
3 plt.pie(y)
4 plt.show()
```

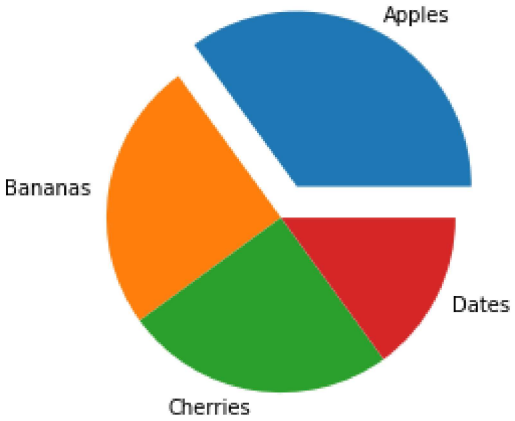
```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
3
4 plt.pie(y, labels = mylabels)
5 plt.show()
```



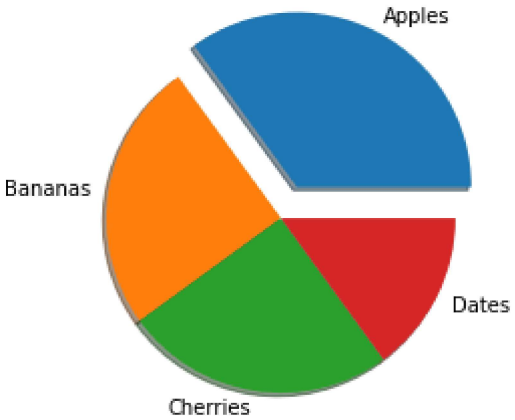

```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
3
4 plt.pie(y, labels = mylabels, startangle = 90)
5 plt.show()
```




```
1
2 y = np.array([35, 25, 25, 15])
3 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
4 myexplode = [0.2, 0, 0, 0]
5
6 plt.pie(y, labels = mylabels, explode = myexplode)
7 plt.show()
```

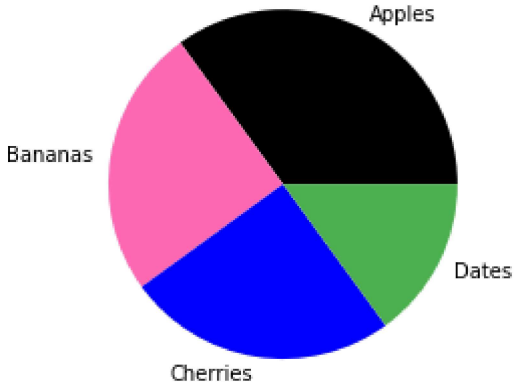



```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
3 myexplode = [0.2, 0, 0, 0]
4
5 plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
6 plt.show()
```

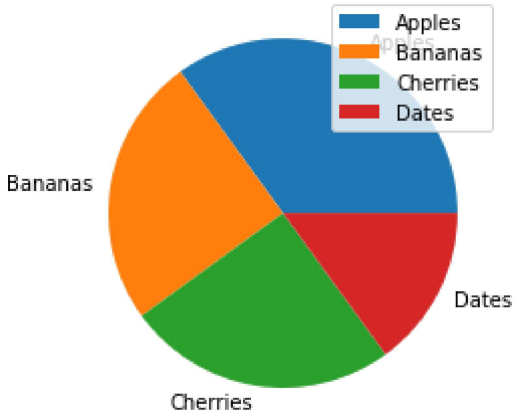



```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
```

```
3 mycolors = ["black", "hotpink", "b", "#4CAF50"]
4
5 plt.pie(y, labels = mylabels, colors = mycolors)
6 plt.show()
```




```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
3
4 plt.pie(y, labels = mylabels)
5 plt.legend()
6 plt.show()
```




```
1 y = np.array([35, 25, 25, 15])
2 mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
3
4 plt.pie(y, labels = mylabels)
5 plt.legend(title = "Four Fruits:")
6 plt.show()
```

1 . '-' Solid line style

2 '--' Dashed line style

3 '-.'Dash-dot line style

4 ':'Dotted line style

5 '.'Point marker

6 ','Pixel marker

7 'o'Circle marker

8 'v'Triangle_down marker

9 '^'Triangle_up marker

10 '<'Triangle_left marker

11 '>'Triangle_right marker

12 '1'Tri_down marker

13 '2'Tri_up marker

14 '3'Tri_left marker

15 '4'Tri_right marker

16 's'Square marker

17 'p' Pentagon marker

18 '*' Star marker

19 'h'Hexagon1 marker

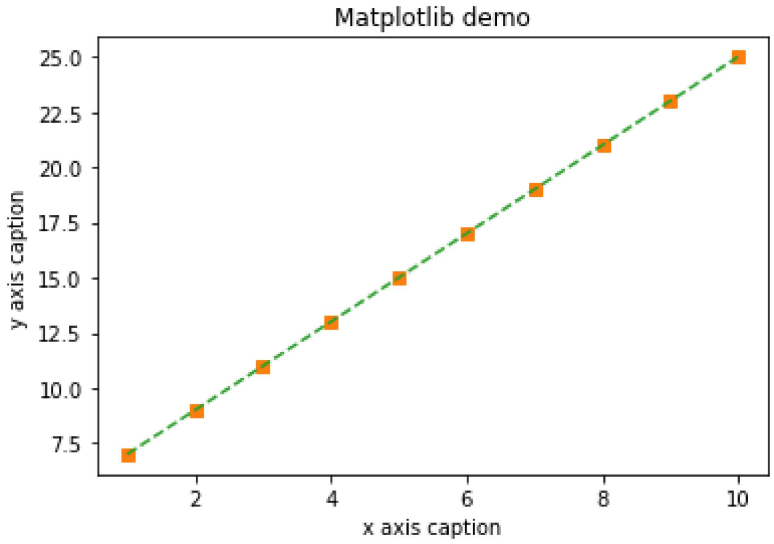
20 'H' Hexagon2 marker

21 '+'Plus marker

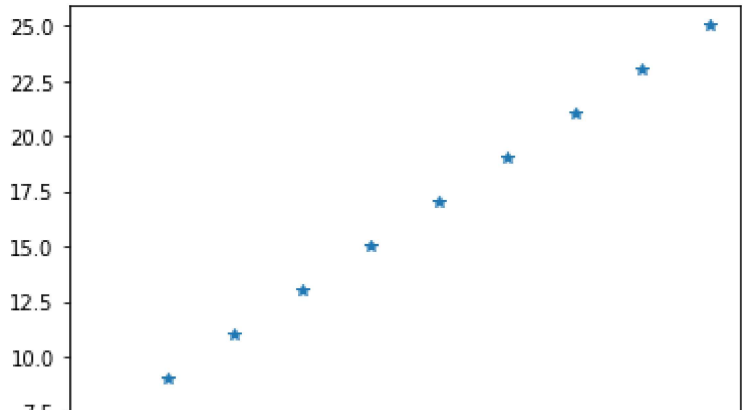
```
x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
# Function add a legend
plt.plot(x,y,"*")

plt.plot(x,y,"s")

```

```
11 plt.plot(x,y,"--")
12
```

```
[<matplotlib.lines.Line2D at 0x7f439918dd90>]
```




```
1 plt.plot(x,y,"*")
2 plt.show()
3 plt.plot(x,y,"s")
4 plt.show()
5 plt.plot(x,y,"--")
6 plt.show()
```



'b' Blue

'g' Green

'r' Red

'c' Cyan

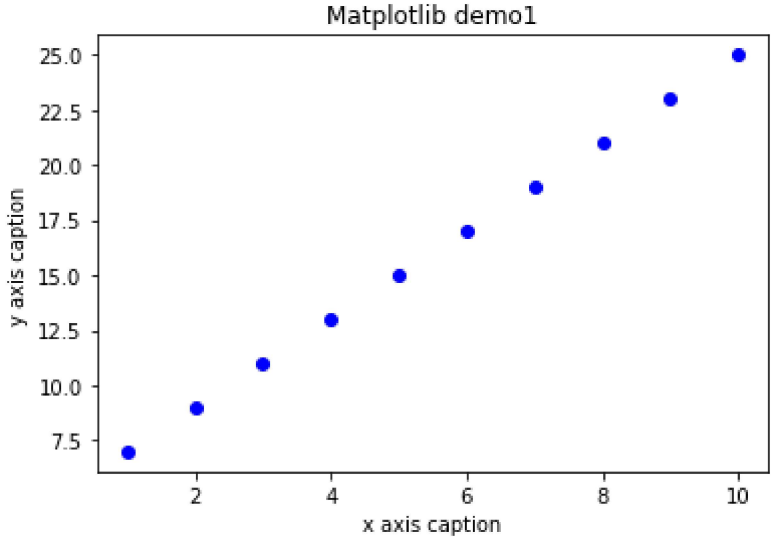
'm' Magenta

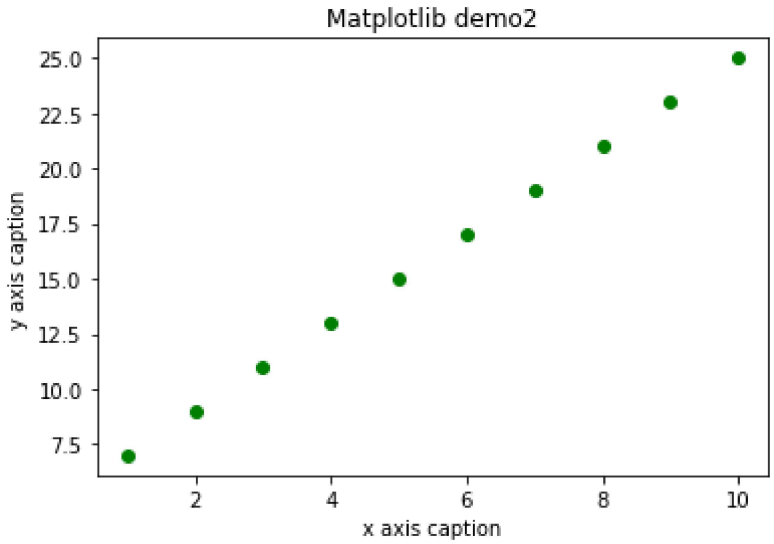
'y' Yellow

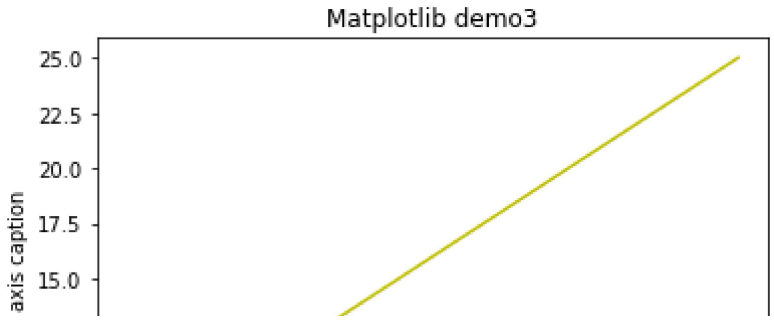
'k' Black

'w' White

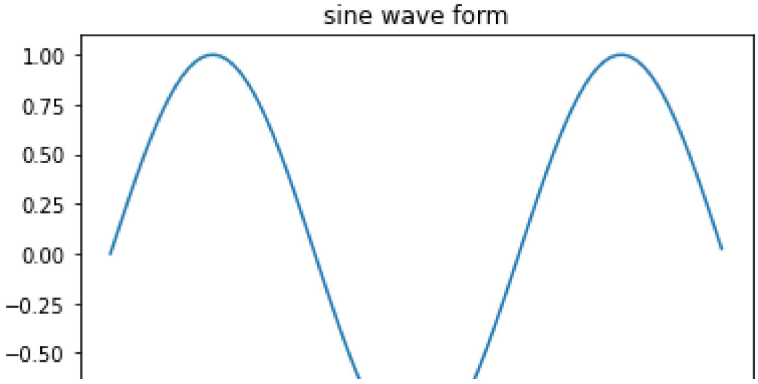
```
 1 import numpy as np
 2 from matplotlib import pyplot as plt
 3
 4 x = np.arange(1,11)
 5 y = 2 * x + 5
 6 plt.title("Matplotlib demo1")
 7 plt.xlabel("x axis caption")
 8 plt.ylabel("y axis caption")
 9 plt.plot(x,y,"ob")
10 plt.show()
11 plt.title("Matplotlib demo2")
12 plt.xlabel("x axis caption")
13 plt.ylabel("y axis caption")
14 plt.plot(x,y,"og")
15 plt.show()
16 plt.title("Matplotlib demo3")
17 plt.xlabel("x axis caption")
18 plt.ylabel("y axis caption")
19 plt.plot(x,y,"y")
20 plt.show()
```



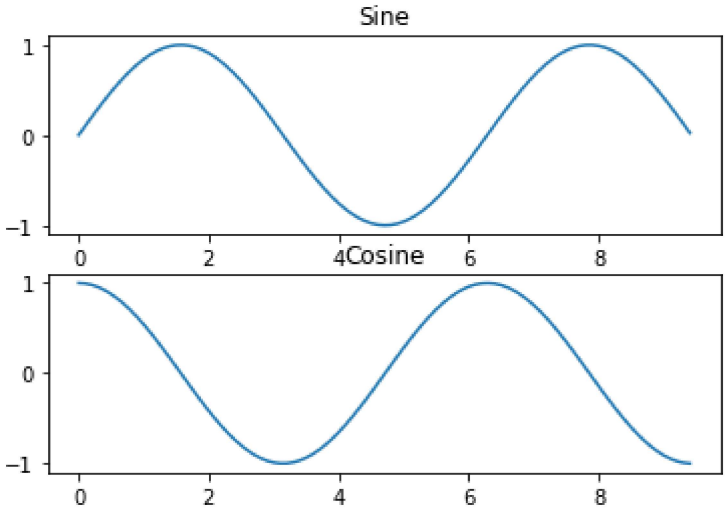





```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3
 4 # Compute the x and y coordinates for points on a sine curve
 5 x = np.arange(0, 3 * np.pi, 0.1)
 6 y = np.sin(x)
 7 plt.title("sine wave form")
 8
 9 # Plot the points using matplotlib
10 plt.plot(x, y)
11 plt.show()
```

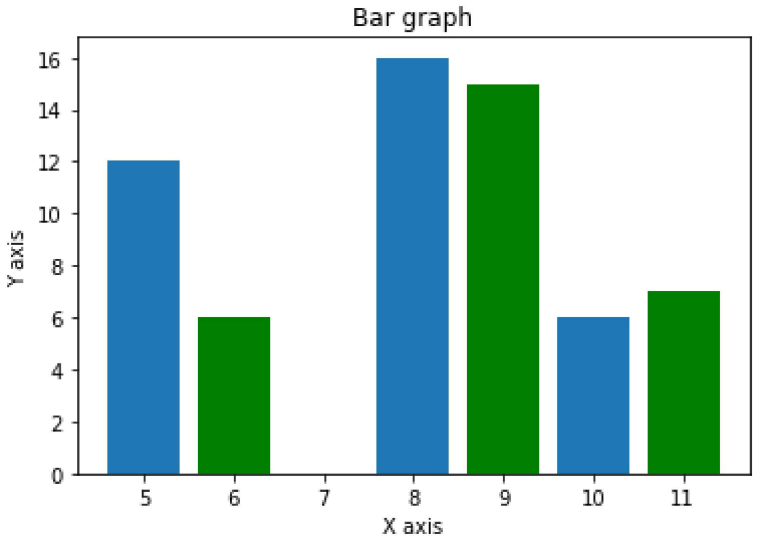



```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```




```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]

```

```
x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center')
plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')

plt.show()
```

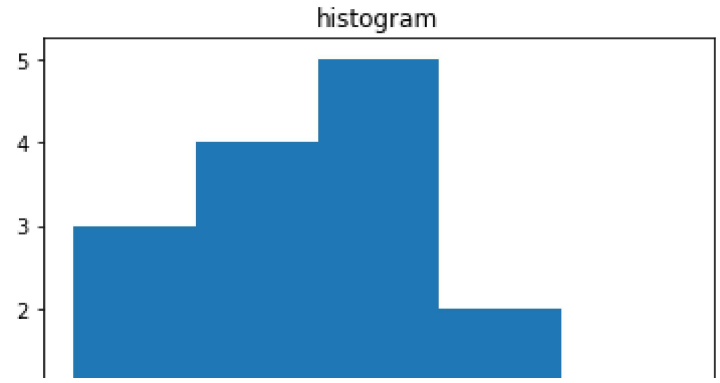


Histogram

```
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
np.histogram(a,bins = [0,20,40,60,80,100])
hist,bins = np.histogram(a,bins = [0,20,40,60,80,100])
print (hist)
print (bins)
```

```
[3 4 5 2 1]
[  0  20  40  60  80 100]
```

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

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("histogram")
plt.show()
```



Random value Generation

```
1 from numpy import random
2 x = random.randint(100)
3 print(x)
4 x = random.randint(100, size=(3, 5))
5 print(x)
```

```
72
[[14 87 98  0 81]
 [35 53 43 48 97]
 [73 86 10 75 89]]
```

```
1 x = random.choice([3, 5, 7, 9], size=(3, 5))
2 print(x)
```

```
[[5 5 9 5 7]
 [3 7 5 3 9]
 [3 7 7 3 5]]
```

```
1 x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(10))
2 print(x)
```

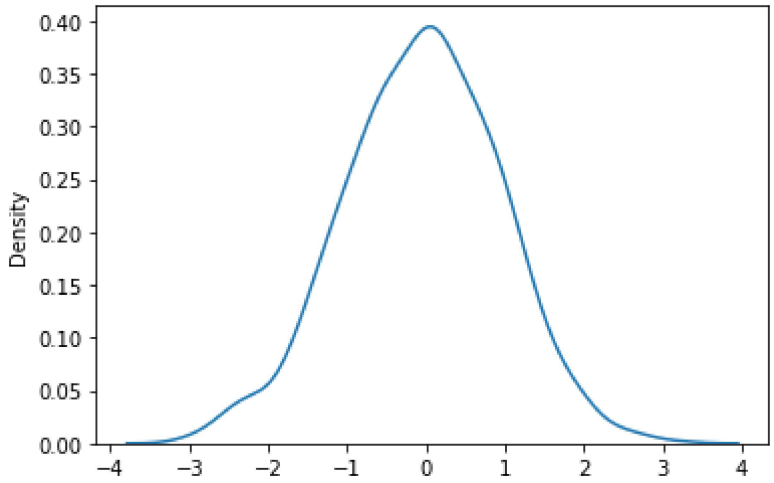
```
[3 7 5 3 7 7 5 7 5 7]
```

```
1 #Normal Distribution
2 x = random.normal(loc=1, scale=2, size=(2, 3))
3
4 print(x)
```

```
[[ 2.63419276  1.73322657  0.14622049]
 [ 1.44915431  1.78059274 -2.75022813]]
```

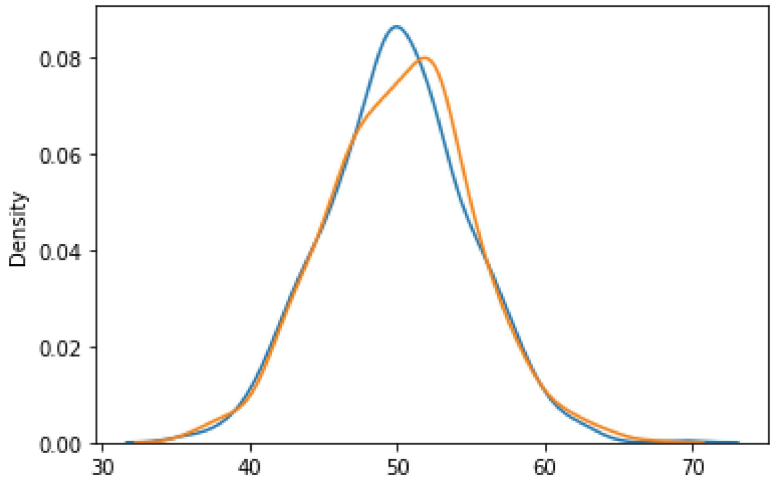
```
1 #Visualization of Normal Distribution
2 import matplotlib.pyplot as plt
3 import seaborn as sns #used to visualize random distributions.
4
5 sns.distplot(random.normal(size=1000), hist=False)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f439907f890>
```




```
1 sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')
2 sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')
3
4 plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```




```
1 from numpy import random
2
3 x = random.binomial(n=10, p=0.5, size=10)
4
5 print(x)
```

```
[6 3 6 4 4 5 6 4 4 5]
```

Colab paid products - Cancel contracts here

✓ 0s completed at 6:47 AM