Numpy

Numpy is integral package for Scientific Computing. It is a Library which provides multidimensional array object.

What is Numpy?

- Numpy is a scientific & Numerical Computing Library in pyhton.
- It provides high performance multi dimensional array object which helps to work with arrays.
- A Multi Dimensional Array is table of elements with same data type indexed by positive integres.
- In numpy, Dimensions are called as axes.

In [34]:
```
import numpy as np
a=np.array([1,2,3,4,5])
print(a)
print(a[0],a[1],a[-1])
print(a[2:],"\n",a[:])
```
```
[1 2 3 4 5]
1 2 5
[3 4 5]
 [1 2 3 4 5]
```

Numpy Array vs Python List

Numpy Array is Fast, Convenient & uses less memory over python list

In [24]:
```
import time
import sys
a = range(1000)
print(sys.getsizeof(1)*len(a))
b = np.arange(1000)
print(b.size * b.itemsize) #uses less memory
l1 = [1,2,3]
l2 = [3,4,5]
print(list(zip(l1,l2))) #zip function is to combine two or more iterables like list, tuples etc..
```
```
28000
4000
[(1, 3), (2, 4), (3, 5)]
```

In [36]:
```
a=100000
l1 = range(a)
l2 = range(a)
a1 = np.arange(a)
a2 = np.arange(a)
start = time.time()
res = [x + y for x, y in zip(l1,l2)]
print("time took by list in milli sec:", (time.time() - start)*1000)
start1 = time.time()
res1 = a1 + a2 #convenient
print("time took by array in milli sec:", (time.time() - start1)*1000) # fast
```
```
time took by list in milli sec: 12.88914680480957
time took by array in milli sec: 0.0
```

Basic Operations of Numpy

In [38]:
```
# Basic Operations of numpy
arr = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print(arr.ndim)
print(arr.size) # no.of ele in array
print(arr.itemsize) # Size of ele
print(arr.shape) # (no of rows or no of ele in outer most array, dimension or axes of array)
arr = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], dtype = np.float64)
print()
print(arr.size)
print(arr.itemsize)
```
```
2
12
4
(4, 3)

12
8
```

NumPy Data Types

Data Type	Description	Example
`int8`	Integer (8-bit)	`np.int8(127)`
`int16`	Integer (16-bit)	`np.int16(32767)`
`int32`	Integer (32-bit)	`np.int32(2**31 - 1)`
`int64`	Integer (64-bit)	`np.int64(2**63 - 1)`
`uint8`	Unsigned Integer (8-bit)	`np.uint8(255)`
`uint16`	Unsigned Integer (16-bit)	`np.uint16(65535)`
`uint32`	Unsigned Integer (32-bit)	`np.uint32(2**32 - 1)`
`uint64`	Unsigned Integer (64-bit)	`np.uint64(2**64 - 1)`
`float16`	Half precision float	`np.float16(3.14)`
`float32`	Single precision float	`np.float32(3.14)`
`float64`	Double precision float (default)	`np.float64(3.14)`
`complex64`	Complex number (2x float32)	`np.complex64(1 + 2j)`
`complex128`	Complex number (2x float64)	`np.complex128(1 + 2j)`
`bool_`	Boolean (True or False)	`np.bool_(True)`
`object_`	Python object	`np.array(['a', 123], dtype=object)`
`str_`	Fixed-length Unicode string	`np.str_('hello')`
`unicode_`	Alias for `str_`	`np.unicode_('hello')`

In [45]:
```
print(np.zeros((3,4)))
print()
print(np.ones((4,3)))
print()
print(np.arange(9))
```
```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[0 1 2 3 4 5 6 7 8]
```

Numpy String Operations

In [67]:
```
print(np.char.add(["jasmin"," shaik"], ["abc", "xyz"]))
print()
print(np.char.multiply("Hello  ", 3))
print()
print(np.char.center("Jasmin Shaik", 40, fillchar = '*'))
print()
print(np.char.capitalize("jasmin shaik"))
print()
print(np.char.title("jasmin shaik is doing python"))
print()
print(np.char.upper("jasmin"))
print()
print(np.char.lower(["JASMIN", "SHAIK"]))
```
```
['jasminabc' ' shaikxyz']

Hello  Hello  Hello

**************Jasmin Shaik**************

Jasmin shaik

Jasmin Shaik Is Doing Python

JASMIN

['jasmin' 'shaik']
```

In [76]:
```
print(np.char.split("practising from soo long"))
print()
print(np.char.splitlines("practising\nfrom soo long"))
print()
print(np.char.strip(["apractising", "afrom", "asoaaoa", "along"], 'a'))
print()
print(np.char.join([":","-"], ["Jasmin", "Shaik"]))
print()
print(np.char.replace("I can not do it", "not", " "))
```
```
['practising', 'from', 'soo', 'long']

['practising', 'from soo long']

['practising' 'from' 'soaao' 'long']

['J:a:s:m:i:n' 'S-h-a-i-k']

I can   do it
```

Array Manipulation

Changing Shape

In [113…
```
arr = np.arange(12)
print(arr)
print()
print(arr.reshape(3,4))
print()
print(arr.flatten())
print()
print(arr.flatten(order = "F"))
print()
print(np.arange(9).reshape(3,3))
```
```
[ 0  1  2  3  4  5  6  7  8  9 10 11]

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[ 0  1  2  3  4  5  6  7  8  9 10 11]

[ 0  1  2  3  4  5  6  7  8  9 10 11]

[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [114…
```
z = np.arange(15).reshape(3,5)
print(z,"\n")
print(np.transpose(z))
```
```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]

[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

In [115…
```
arr = np.arange(12).reshape(3,2,2)
arr
```
Out[115]:
```
array([[[ 0,  1],
        [ 2,  3]],

       [[ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11]]])
```

In [116…
```
np.rollaxis(arr, 2, 1)
```
Out[116]:
```
array([[[ 0,  2],
        [ 1,  3]],

       [[ 4,  6],
        [ 5,  7]],

       [[ 8, 10],
        [ 9, 11]]])
```

In [117…
```
np.rollaxis(arr, 1)
```
Out[117]:
```
array([[[ 0,  1],
        [ 4,  5],
        [ 8,  9]],

       [[ 2,  3],
        [ 6,  7],
        [10, 11]]])
```

In [118…
```
np.rollaxis(arr, 2)
```
Out[118]:
```
array([[[ 0,  2],
        [ 4,  6],
        [ 8, 10]],

       [[ 1,  3],
        [ 5,  7],
        [ 9, 11]]])
```

In [120…
```
np.swapaxes(arr,1,2)
```
Out[120]:
```
array([[[ 0,  2],
        [ 1,  3]],

       [[ 4,  6],
        [ 5,  7]],

       [[ 8, 10],
        [ 9, 11]]])
```

Numpy Arithematic Operations

In [129…
```
a = np.arange(9).reshape(3,3)
b = np.array([10,10,10])
print(a, "\n", b, "\n")
print(np.add(a,b), "\n")
print(np.subtract(a,b), "\n")
print(np.multiply(a,b), "\n")
print(np.divide(a,b), "\n")
```
```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
 [10 10 10]

[[10 11 12]
 [13 14 15]
 [16 17 18]]

[[-10  -9  -8]
 [ -7  -6  -5]
 [ -4  -3  -2]]

[[ 0 10 20]
 [30 40 50]
 [60 70 80]]

[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
```

Slicing

In [131…
```
a = np.arange(10)
print(a)
print(a[5:])
print(a[:3])
print(a[:])
print(a[5:])
print(a[:-4])
```
```
[0 1 2 3 4 5 6 7 8 9]
[5 6 7 8 9]
[0 1 2]
[0 1 2 3 4 5 6 7 8 9]
[5 6 7 8 9]
[0 1 2 3 4 5]
```

In [143…
```
print(a[slice(1,8,2)])
```
```
[1 3 5 7]
```

Iterating Over Array

In [146…
```
arr = np.arange(0, 50, 5)
print(arr)
arr = arr.reshape(2,5)
print(arr)

for x in np.nditer(arr):
    print(x)
```
```
[ 0  5 10 15 20 25 30 35 40 45]
[[ 0  5 10 15 20]
 [25 30 35 40 45]]
0
5
10
15
20
25
30
35
40
45
```

Iterating Over Array (C-style and F-style)

In [155…
```
arr = np.arange(0, 16).reshape(4,2,2)

print(arr)

for x in np.nditer(arr, order = "C"):
    print(x)
print()
for x in np.nditer(arr, order = "F"):
    print(x)
```
```
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]

 [[12 13]
  [14 15]]]
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

0
4
8
12
2
6
10
14
1
5
9
13
3
7
11
15
```

Joining Arrays

In [163…
```
# If all arrays are of same shape than only joining of arrays possible
a = np.array([[1,2],[3,4]])
b = np.array([[6,7],[10,11]])
print(a,"\n")
print(b,"\n")
print("after joining:\n", np.concatenate((a,b)))
print("after joining axis 0:\n", np.concatenate((a,b), axis = 0))
print("after joining axis 1:\n", np.concatenate((a,b), axis = 1))
```
```
[[1 2]
 [3 4]]

[[ 6  7]
 [10 11]]

after joining:
 [[ 1  2]
 [ 3  4]
 [ 6  7]
 [10 11]]
after joining axis 0:
 [[ 1  2]
 [ 3  4]
 [ 6  7]
 [10 11]]
after joining axis 1:
 [[ 1  2  6  7]
 [ 3  4 10 11]]
```

Spliting Array

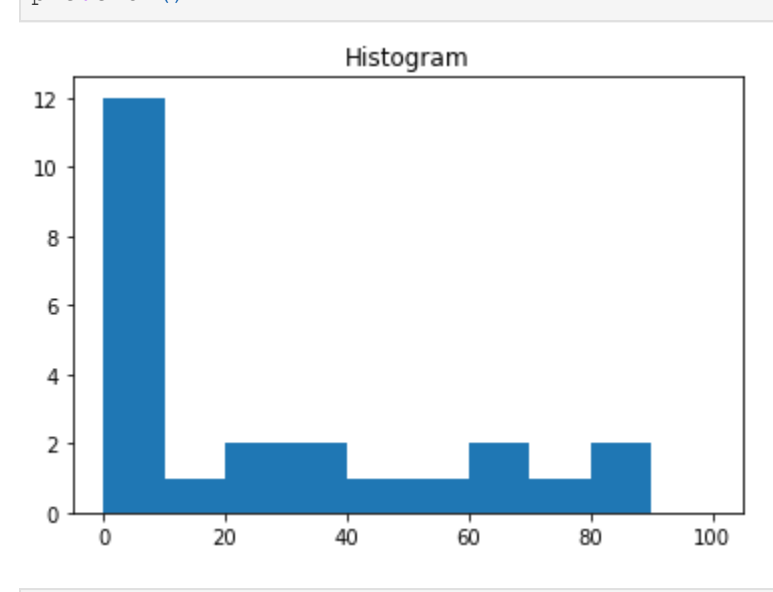
In [172…
```
a = np.arange(9)
print(a)
print(np.split(a,3))
print(np.split(a,[4,5]))
print(np.split(a,[4,8]))
```
```
[0 1 2 3 4 5 6 7 8]
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
[array([0, 1, 2, 3]), array([4]), array([5, 6, 7, 8])]
[array([0, 1, 2, 3]), array([4, 5, 6, 7]), array([8])]
```

Resizing an Array

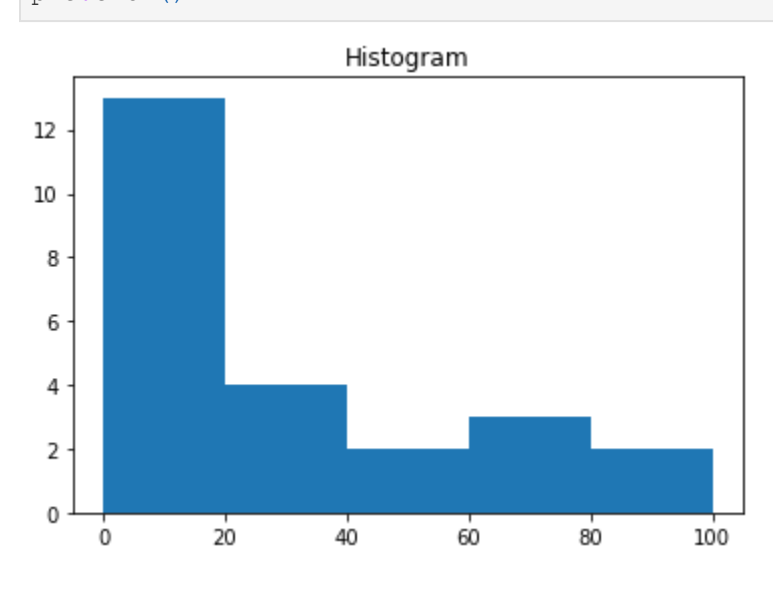
In [185…
```
a = np.array([[1,2,3],[0,9,8],[7,8,9],[0,9,6]])
print(a.shape)
print(a,"\n")
print(np.reshape(a,(3,4)))
print()
print(np.reshape(a,(3,2,2)))
```
```
(4, 3)
[[1 2 3]
 [0 9 8]
 [7 8 9]
 [0 9 6]]

[[1 2 3 0]
 [9 8 7 8]
 [9 0 9 6]]

[[[1 2]
  [3 0]]

 [[9 8]
  [7 8]]

 [[9 0]
  [9 6]]]
```

Numpy Histogram using matplotlib

In [191…
```
from matplotlib import pyplot as plt
a = np.array([1,3,6,10,7,45,6,7,89,78,65,89,35,25,34,24,67,56,4,5,6,7,0,1])
plt.hist(a, bins = [0,10,20,30,40,50,60,70,80,90,100])
plt.title("Histogram")
plt.show()
```



In [192…
```
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("Histogram")
plt.show()
```



Other Useful functions in Numpy

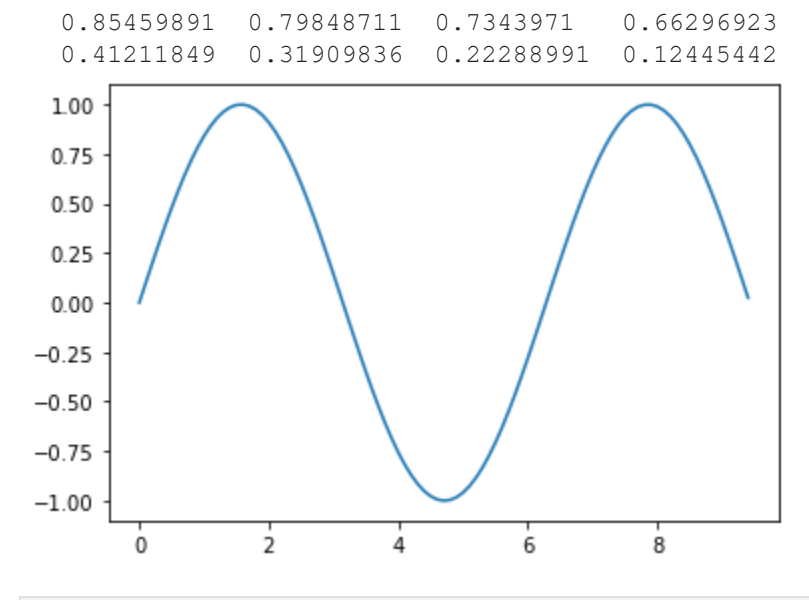
In [195…
```
a = np.linspace(1,3,15)
print(a)
```
```
[1.         1.14285714 1.28571429 1.42857143 1.57142857 1.71428571
 1.85714286 2.         2.14285714 2.28571429 2.42857143 2.57142857
 2.71428571 2.85714286 3.        ]
```

In [208…
```
a = np.array([[1,2],[29,0]])
print(np.sum(a, axis=0))
print(np.sum(a, axis=1))
```
```
[30  2]
[ 3 29]
```

In [237…
```
b = np.array([[1,2,3], [4,5,6]])
print(np.sqrt(b), "\n")
print(np.std(b), "\n")
print(np.mean(b), "\n")
print(np.median(b), "\n")
print(np.ravel(b), "\n") # Similar to flatten function
print(np.log10(b), "\n")
```
```
[[1.         1.41421356 1.73205081]
 [2.         2.23606798 2.44948974]]

1.707825127659933

3.5

3.5

[1 2 3 4 5 6]

[[0.         0.30103    0.47712125]
 [0.60205999 0.69897    0.77815125]]
```

Numpy Practice Example

In [217…
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,3*np.pi,0.1)
y = np.sin(x)
print(x)
print("\n", y)
plt.plot(x,y)
plt.show()
```
```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5
 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3
 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.  7.1
 7.2 7.3 7.4 7.5 7.6 7.7 7.8 7.9 8.  8.1 8.2 8.3 8.4 8.5 8.6 8.7 8.8 8.9
 9.  9.1 9.2 9.3 9.4]

 [ 0.          0.09983342  0.19866933  0.29552021  0.38941834  0.47942554
  0.56464247  0.64421769  0.71735609  0.78332691  0.84147098  0.89120736
  0.93203909  0.96355819  0.98544973  0.99749499  0.9995736   0.99166481
  0.97384763  0.94630009  0.90929743  0.86320937  0.8084964   0.74570521
  0.67546318  0.59847214  0.51550137  0.42737988  0.33498815  0.23924933
  0.14112001  0.04158066 -0.05837414 -0.15774569 -0.2555411  -0.35078323
 -0.44252044 -0.52983614 -0.61185789 -0.68776616 -0.7568025  -0.81827711
 -0.87157577 -0.91616594 -0.95160207 -0.97753012 -0.993691   -0.99992326
 -0.99616461 -0.98245261 -0.95892427 -0.92581468 -0.88345466 -0.83226744
 -0.77276449 -0.70554033 -0.63126664 -0.55068554 -0.46460218 -0.37387666
 -0.2794155  -0.1821625  -0.0830894   0.0168139   0.1165492   0.21511999
  0.31154136  0.40484992  0.49411335  0.57843976  0.6569866   0.72896904
  0.79366786  0.85043662  0.8987081   0.93799998  0.96791967  0.98816823
  0.99854335  0.99894134  0.98935825  0.96988981  0.94073056  0.90217183
  0.85459891  0.79848711  0.7343971   0.66296923  0.58491719  0.50102086
  0.41211849  0.31909836  0.22288991  0.12445442  0.02477543]
```



In [225…
```
# creating 6*6 array and replacing diagonals with 0 and 1.
z = np.zeros((6,6),dtype=int)
print(z, "\n")
z[1::2,::2] = 1
print(z, "\n")
z[::2,1::2] = 1
print(z)
```
```
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]

[[0 0 0 0 0 0]
 [1 0 1 0 1 0]
 [0 0 0 0 0 0]
 [1 0 1 0 1 0]
 [0 0 0 0 0 0]
 [1 0 1 0 1 0]]

[[0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]]
```

In [227…
```
# Finding total number and Locations of missing values in the array
z = np.random.rand(10,10)
z[np.random.randint(10, size=5), np.random.randint(10, size = 5)] = np.nan
z
```
Out[227]:
```
array([[0.95114793, 0.16744377, 0.54656145, 0.40099173, 0.11548118,
        0.54169642, 0.14658045, 0.88569239,        nan, 0.22756095],
       [0.09133778, 0.67641136, 0.27615108, 0.12338284, 0.40066751,
        0.44481626, 0.55190798, 0.13305462, 0.03037242, 0.60680655],
       [0.04728606, 0.13011515, 0.60732414, 0.66761425, 0.9388124 ,
        0.615344  , 0.05592245, 0.01190867, 0.84516607, 0.61527402],
       [0.42218138, 0.65700505, 0.766525  , 0.94234844, 0.2581955 ,
        0.9522338 , 0.2210144 , 0.69524361, 0.84113343, 0.06696265],
       [0.960341  , 0.06612879, 0.21974504, 0.03688613,        nan,
        0.50898697, 0.72138015, 0.6730198 , 0.11383271, 0.70115228],
       [0.37845922, 0.35248865, 0.75436711, 0.95415987, 0.23221162,
        0.80956866,        nan, 0.0393784 , 0.63476336, 0.1702071 ],
       [0.14173725, 0.6878504 , 0.4856067 , 0.58735863, 0.10455469,
        0.41360244, 0.01639311, 0.93764046, 0.767168  , 0.5008627 ],
       [0.75811539, 0.89282422, 0.67378395, 0.36522204, 0.42200514,
        0.28172333,        nan, 0.08154385, 0.63986731, 0.58932959],
       [0.65903251, 0.79221866, 0.27204628, 0.7157541 , 0.82500288,
        0.83525665, 0.57603187, 0.6624287 , 0.87546957, 0.2337397 ],
       [0.04168598, 0.52376442, 0.74572686, 0.29569106, 0.40328524,
        0.57244679, 0.60636607, 0.34068083, 0.07414366,        nan]])
```

In [233…
```
print("total missing values:", np.isnan(z).sum())
print("Indexes of missing values:\n", np.argwhere(np.isnan(z)))
print("indexes:\n", np.where(np.isnan(z)))
```
```
total missing values: 5
Indexes of missing values:
 [[0 8]
 [4 4]
 [5 6]
 [7 6]
 [9 9]]
indexes:
 (array([0, 4, 5, 7, 9], dtype=int64), array([8, 4, 6, 6, 9], dtype=int64))
```

In [ ]: