Milind Mali / Data Scientist / Data Analyst

Basic operation with NumPy

Numpy- Numerical Python

- 1.Built in function in numpy array
- 2.Creating one dimensional array
- 3.Creating two dimensional array
- 4.Three dimensional array
- 5.Higher dimensional array
- 6.Initial placeholder in numpy array
- 7.Analysis numpy array
- 8.Mathematical operation with numpy
- 9.Array manipulation
- 10.Numpy Indexing and selection
- 11.Grabbing single element
- 12.Broadcasting
- 13.Slicing
- 14.Indexing and selection in two dimensional array
- 15.Indexing in 3D array
- 16.Condition and selection

Advantage of Numpy Array

- Allow several mathematical Operation
- faster operations

although array list and tuple all are used to hold the any data but the process time takes for array operation is comparatively very less

List Vs Numpy Array--> Lets check the time taken

In [2]:

```
import numpy as np
```

In [3]:

```
from time import process_time
```

In [26]:

```
# time taken by list

list1=[i for i in range(10000000)]

start_time=process_time()

list1=[i+10 for i in list1]    #adding five to each number

end_time=process_time()

print(end_time-start_time)
```

```
1.578125
```

In [25]:

```
array1=np.array([i for i in range(10000000)])

start_time=process_time()

array1+=10

end_time=process_time()

print(end_time-start_time)
```

```
0.015625
```

Conclusion :--> required time for operation with array is very less thats why we prefer to array against list or tuple

In [31]:

```
list1=[1,2,3,4,5]

print(list1)

print(type(list1))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

numeber are seperated by comma

In [32]:

```
Array1=np.array([1,2,3,4,5])

print(Array1)

print(type(Array1))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

number of are not seperated by comma

Built in function in Numpy Array

creating one dimentional array

In [93]:

```
# arange function
x=np.arange(1,10,2)     # from 1 to 10 with step size 2
print(x)
```

```
[1 3 5 7 9]
```

In [94]:

```
# five random values from zero to one
x=np.random.rand(5)
print(x)
```

```
[0.28285425 0.90522102 0.13056326 0.30841338 0.10878923]
```

this is two dimentional array

In [99]:

```
list1=[[1,2,3],[4,5,6],[7,8,9]]

my_matrix=np.array(list1)

print(my_matrix)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [95]:

```
# 2*3 matrix contain from zero to one
x=np.random.rand(2,3)
print(x)
```

```
[[0.03096923 0.81105207 0.32229728]
 [0.44511911 0.17732007 0.52159635]]
```

In [96]:

```
# random values follows std normal distribution means whose mean is zero and std dev is 1
x=np.random.randn(5)
print(x)
```

```
[-0.74418913  0.43635588 -1.75289267  0.05025689  1.4464433 ]
```

In [97]:

```
# random values follows std normal distribution means whose mean is zero and std dev is 1
x=np.random.randn(3,4)
print(x)
```

```
[[-0.35284373  1.41559546  1.35191402  0.15987834]
 [ 0.82225866 -1.14712803 -0.50553481  0.75719029]
 [ 0.368709   -0.45620814 -0.20477501  0.4112788 ]]
```

In [98]:

```
#np.random.randint
x=np.random.randint(10,100,(3,4))
print(x)
```

```
[[93 25 84 86]
 [57 74 44 25]
 [97 46 47 29]]
```

In [40]:

```
# now create two dimentional array (with defining values at float values)
array2=np.array([(1,2,3,4),(5,6,7,8)],dtype=float)

print(array2)

array2.shape
```

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

Out[40]:

```
(2, 4)
```

Three dimentional array

In [155]:

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

higher Dimentional arrays

In [160]:

```
arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

intial placeholder in numpy array

- in some cases we want to intiate array with certain values

In [44]:

```
# create numpy array with all the values are zero
x=np.zeros((4,5))
print(x)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [45]:

```
# create numpy array with all the values are one
x=np.ones((4,5))
print(x)
```

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

In [46]:

```
# array of perticular values

x=np.full((4,4),5)
print(x)
```

```
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]]
```

In [48]:

```
# creating identity matrix- all the diogonal element will be one and rest all the values

x=np.eye(4)
print(x)
```

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

In [50]:

```
# create array with random values

x=np.random.random((3,4))
print(x)
```

```
[[0.73860796 0.10178962 0.98704855 0.23459453]
 [0.55524451 0.21111667 0.68869849 0.37894866]
 [0.61243223 0.1972518  0.38222988 0.44870634]]
```

all the values you will get here will be in the range of 0 to 1

In [52]:

```
# create random int values
x=np.random.randint(10,100,(3,4))
print(x)
```

```
[[37 47 35 38]
 [13 31 94 24]
 [75 93 49 38]]
```

In [54]:

```
# creating array of evenly spaced values
x=np.linspace(10,30,5)
print(x)
```

```
[10. 15. 20. 25. 30.]
```

In [55]:

```
# converting list to array
list1=[10,20,30,40,50]

array1=np.asarray(list1)

print(array1)
type(array1)
```

```
[10 20 30 40 50]
```

Out[55]:

```
numpy.ndarray
```

In [56]:

```
# same operation we do to convert tuple to array--> np.asarray()
```

Analysing numpy array

In [61]:

```
# shape

x=np.random.randint(10,99,(4,5))
print(x)
print("Shape of the array:\n ",x.shape)
```

```
[[74 11 89 32 93]
 [95 25 48 50 75]
 [47 52 63 36 52]
 [78 39 15 84 37]]
Shape of the array:
  (4, 5)
```

In [62]:

```
# dimenation of the array

x.ndim
```

Out[62]:

```
2
```

In [64]:

```
# number of the element present in the array

x.size
```

Out[64]:

```
20
```

In [65]:

```
#checking the datatype of the element in the array

x.dtype
```

Out[65]:

```
dtype('int32')
```

In [100]:

```
myarr=np.random.randint(10,20,10)
print(myarr)
```

```
[19 15 19 16 11 18 14 19 16 18]
```

In [101]:

```
myarr.max()
```

Out[101]:

```
19
```

In [102]:

```
# index location of max value
myarr.argmax()
```

Out[102]:

```
0
```

In [103]:

```
myarr.min()
```

Out[103]:

```
11
```

In [104]:

```
#index location of min value
myarr.argmin()
```

Out[104]:

```
4
```

Mathematical Opearation with NumPy array

In [66]:

```
list1=[1,2,3,4,5]

list2=[6,7,8,9,10]

list1+list2  #==> instead of adding this PLus operator will concatenate the two list
```

Out[66]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [80]:

```
a=np.random.randint(10,20,(3,3))
b=np.random.randint(20,30,(3,3))

print(a)
print(b)
```

```
[[16 18 11]
 [16 18 10]
 [15 13 10]]
[[25 29 29]
 [20 20 26]
 [26 29 27]]
```

In [81]:

```
# let do some basic Mathematic Operations
print(a+b)   #==> same result could be obtained by np.add(a,b)
```

```
[[41 47 40]
 [36 38 36]
 [41 42 37]]
```

In [82]:

```
print(a-b)    #==> same result could be obtained by np.substract(a,b)
```

```
[[ -9 -11 -18]
 [ -4  -2 -16]
 [-11 -16 -17]]
```

In [83]:

```
print(a*b)    #==> same result could be obtained by np.multiply(a,b)
```

```
[[400 522 319]
 [320 360 260]
 [390 377 270]]
```

In [84]:

```
print(a/b)    #==> same result could be obtained by np.division(a,b)
```

```
[[0.64       0.62068966 0.37931034]
 [0.8        0.9        0.38461538]
 [0.57692308 0.44827586 0.37037037]]
```

In [135]:

```
arr=np.arange(0,11)
arr
```

Out[135]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [137]:

```
arr+arr
```

Out[137]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [138]:

```
arr/arr
```

```
C:\Users\SSRVC\AppData\Local\Temp\ipykernel_18212\1862401812.py:1: Runtime
Warning: invalid value encountered in true_divide
  arr/arr
```

Out[138]:

```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

In [139]:

```
# since there is zero values at first place at both array which make no sence with math p
#in normal python it could be error where as in array function it shows nan values at tha
```

In [140]:

```
## there are some universal function also
```

In [141]:

```
arr
```

Out[141]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [142]:

```
np.sqrt(arr)
```

Out[142]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766])
```

In [143]:

```
np.sin(arr)
```

Out[143]:

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
       -0.54402111])
```

In [144]:

```
np.log(arr)
```

```
C:\Users\SSRVC\AppData\Local\Temp\ipykernel_18212\3120950136.py:1: Runtime
Warning: divide by zero encountered in log
  np.log(arr)
```

Out[144]:

```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,
       2.30258509])
```

In [145]:

```
# since log of zero is infinity but still it giving result and infity at its place which
```

In [146]:

```
arr
```

Out[146]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [147]:

```
arr.min()
```

Out[147]:

```
0
```

In [148]:

```
arr.max()
```

Out[148]:

```
10
```

In [149]:

```
arr.mean()
```

Out[149]:

```
5.0
```

In [150]:

```
arr.std()
```

Out[150]:

```
3.1622776601683795
```

In [151]:

```
arr.var()
```

Out[151]:

```
10.0
```

In [152]:

```
arr=np.random.randint(1,10,(5,5))
arr
```

Out[152]:

```
array([[1, 9, 4, 2, 2],
       [3, 1, 5, 1, 9],
       [6, 7, 4, 1, 4],
       [9, 6, 3, 3, 9],
       [6, 8, 9, 1, 9]])
```

In [153]:

```
# summation across the rows
arr.sum(axis=0)
```

Out[153]:

```
array([25, 31, 25,  8, 33])
```

In [154]:

```
# summation across the column
arr.sum(axis=1)
```

Out[154]:

```
array([18, 19, 22, 30, 33])
```

Array Manipulation

In [87]:

```
# transpose

array1=np.random.randint(10,20,(2,3))

print(array1)
array1.shape
```

```
[[14 12 19]
 [11 12 19]]
```

Out[87]:

```
(2, 3)
```

In [88]:

```
trans=np.transpose(array1)
print(trans)
trans.shape
```

```
[[14 11]
 [12 12]
 [19 19]]
```

Out[88]:

```
(3, 2)
```

In [90]:

```
# reshaping array
a=np.random.randint(10,30,(2,3))
print(a)
print(a.shape)
```

```
[[15 23 18]
 [25 18 13]]
(2, 3)
```

In [91]:

```
b=a.reshape(3,2)

print(b)
print(b.shape)
```

```
[[15 23]
 [18 25]
 [18 13]]
(3, 2)
```

Numpy Indexing and selection

- Grabbing single element
- Grabbing a slice of element
- Brodcasting selection
- Indexing and selection in two dimentions
- Condition Selection

Grabbing single element

In [105]:

```
myarr=np.arange(0,11)
print(myarr)
```

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

In [106]:

```
myarr[3]
```

Out[106]:

```
3
```

In [107]:

```
myarr[3:5]   #==> five is excluding
```

Out[107]:

```
array([3, 4])
```

In [108]:

```
myarr[:5]
```

Out[108]:

```
array([0, 1, 2, 3, 4])
```

In [109]:

```
myarr[5:]
```

Out[109]:

```
array([ 5,  6,  7,  8,  9, 10])
```

Brodcasting

In [110]:

```
myarr
```

Out[110]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [111]:

```
# it means you can reassign multiple values simultaneously

myarr[0:5]=100
```

In [112]:

```
myarr
```

Out[112]:

```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

In [113]:

```
arr=np.arange(0,11)
arr
```

Out[113]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

slicing

In [114]:

```
# slicing section of array and setting it to new variable
```

In [115]:

```
arr
```

Out[115]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [116]:

```
slice_of_array=arr[0:5]
slice_of_array
```

Out[116]:

```
array([0, 1, 2, 3, 4])
```

In [117]:

```
slice_of_array[:]=99
slice_of_array
```

Out[117]:

```
array([99, 99, 99, 99, 99])
```

In [118]:

```
arr
```

Out[118]:

```
array([99, 99, 99, 99, 99,  5,  6,  7,  8,  9, 10])
```

In [119]:

```
# so here it affect to its original array in order to not happen it we can do the copy of
```

In [120]:

```
arr_copy=arr.copy()
```

In [121]:

```
arr_copy[:]=100
```

In [122]:

```
arr_copy
```

Out[122]:

```
array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

In [123]:

```
arr
```

Out[123]:

```
array([99, 99, 99, 99, 99,  5,  6,  7,  8,  9, 10])
```

- slicing 2D array

In [159]:

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

```
[7 8 9]
```

- negative slicing

In [158]:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

```
[5 6]
```

indexing and selection in two dimentional array

In [124]:

```
arr_2d=np.random.randint(10,99,(3,3))
arr_2d
```

Out[124]:

```
array([[29, 93, 17],
       [52, 78, 60],
       [70, 19, 97]])
```

In [125]:

```
arr_2d.shape
```

Out[125]:

```
(3, 3)
```

In [126]:

```
# to grab first row
arr_2d[0]
```

Out[126]:

```
array([29, 93, 17])
```

In [127]:

```
# to grab third element in the first row which is 41
arr_2d[0][2]
```

Out[127]:

```
17
```

In [128]:

```
# to get some slice or subsection  of matrix
arr_2d
```

Out[128]:

```
array([[29, 93, 17],
       [52, 78, 60],
       [70, 19, 97]])
```

In [129]:

```
arr_2d[:2,1:]
```

Out[129]:

```
array([[93, 17],
       [78, 60]])
```

In [156]:

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

Indexing in 3D array

In [157]:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

```
6
```

Condition and selection

In [130]:

```
arr=np.arange(0,11)
arr
```

Out[130]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [131]:

```
arr>4
```

Out[131]:

```
array([False, False, False, False, False,  True,  True,  True,  True,
        True,  True])
```

In [132]:

```
bool_arr=arr>4
```

In [133]:

```
arr[bool_arr]   #==> filtering
```

Out[133]:

```
array([ 5,  6,  7,  8,  9, 10])
```

In [134]:

```
# same result can be obtained by following code
arr[arr>4]
```

Out[134]:

```
array([ 5,  6,  7,  8,  9, 10])
```

In [163]:

```
print("="*100)
```

```
====================================================================================================
```

In [ ]:

```

```