Python libraries

Numpy

What is Numpy?
Python library to work with arrays.
Short form of Numerical python.

Why Numpy?
Faster than lists
Improves the performances.
contigous memory allocation.

object
The numpy array is called ndarray.

Array Creation
To create array we use import to import library.

import numpy as np.

Here np is alias name.

```
1 import numpy as np
  arr = np.array ([1, 2, 3])
  print (arr)
```

Their are many ways to create arrays.

Dimension

Their can be n dimension of array.
We can use ndim method to define.

```
arr = np.array([1,2,3,4]) -> simple 1D

arr = np.array([1,2,3,4], ndim = 4) -> 4D array.
```

Note: slicing is same as simple python.

Different dimension

i) 0D
contain only single element
also called scalar.

```
arr = np.array(42)
print (arr, type(arr), arr.ndim)
=> 42, nd.array, D
```

type () is used to get type of object
ndim is used to get ndim of object.

ii) 1D
contain number of scalars.

```
ex  arr = np.array([1,2,3,4])
    print(arr, type(arr), ndim)

=> [1 2 3 4], 1
```

2D

contain number of vectors.

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

This is 2D array
It is called matrices.

```
[[1, 2 3]
 [2 3 4]]
```

To slice we use.

```
arr[0:1, 0:2]
```

This gives [1 2]

Data types

Numpy has some extra data types than python and they are referred by single character.

integer : i
boolean : b
float : f
unsigned int : u
timedelta : m
object : O
string : S
unicode string : U

complex float : c
datetime : M

Checking data type

To check datatype a property 'dtype' is used'.

```
arr= np.array([1,2,3,4])
print(arr.dtype)
=> integer.
```

Creating array with defined data type

The optional arguement 'dtype' is used.

```
arr = np.array([1,2,3,4], dtype = 'i')
print(arr.dtype)
=> int64
```

Note You can also give bytes of data type. For this.

```
=> arr = np.array([1,2,3], dtype = 'i4')
   print(arr.dtype)
   = int32.
```

Q What if value can not be converted?

A non integer like 'a' can not be converted to integer Thus it will raise a valueerror.

Eg
```
arr= np.array([1,'a',2], dtype ='i')  # error.
```

To change datatype we can use the function 'astype()' This function creates copy of array while allowing you to specify data type as parameter.

Note:- if axis is not passed explicitly then it is taken 0.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

```
=> [1, 2, 3 4 5 6]
```

Join along axis=1

```
arr1 = [[1, 2], [3, 4]]
arr2 = [[5, 6], [7, 8]]

arr = np.concatenate((arr1, arr2), axis=1)
```

```
    => [1 2 5 6]
       [3 4 7 8]
```

axis=1 means row wise
axis=0 means colums combine.

Join along stack functions
Same as concatenate by adds a new dimension.
Arrays must have exact shape.
Grouping arrays into higher dimension.

```
arr = np.stack((arr1, arr2), axis=1)
```

if axis not defined the considered 0.

stacking along rows & columns

For rows → hstack is used (no dimension change)
For columns → vstack is used.

stacking along height (depth)

dstack is used.

Splitting Arrays
reverse operation of joinning.
breaks one array into multiple.
array-split (array, no of splits).

```
arr = np.array([1,2,3,4])
newarr = np.array-split(arr,2)
print(newarr)

=> [array([1,2]), array([3,4])]
```

Note if elements are more or less than adjusted at the end.

We also have split() but it will not work properly as it does not adjust elements.

We can also use axis attribute here.

hsplit, vsplit, dsplit are available opposite to hstack, vstack, dstack.

Searching in arrays

We can search for eliments in array.
Return all the index that has the elments
where() is used.

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

=> 3, 6, 7
```

Note: returns a tuple of indices.

Q Search for indexes where values are even.

```
arr = np.array([1, 2, 3, 4, 5, 4, 9, 7, 8])
x = np.where(arr%2 ==0)
print(x)

=> array[(1, 3, 5, 8)]
```

Searchsorted()
This performs binary search in the array and return the index where value would be inserted to maintain order.

Note used on sorted arrays. (Asc)

```
arr = np.arr([1, 2, 4, 5, 6])
x = np.searchsorted(arr, 7)
print(x)
=> 5
```

Thus 5 in output indicates that 7 should inserted at 5th index to maintain order.

Multiple values

if you want search multiple values then pass through a array.

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

=> [1, 3 5]
```

returns an numpy.ndarray.

Sorting Arrays

putting elements into a sequence.
sort() is used.

```
arr = np.array([1, 5, 0, 2])
print(np.sort(arr))

=> [0 1 2 5]
```

Note returns a copy leaving original unchanged.

If you sort boolean array the all false come before.

```
np.array([[1, 5, 3], [0 ;6 ,2]])
print(np.sort(arr))

=> [1 3 5]
   [0 2 6]
```

both array will be sorted.

Filtering Arrays

getting some elements out of existing array and creating new array out of them is called filtering.

We filter an array using a boolean index list.

- if value at an index is True then it is contained else it is excluded.

```
filter = []
arr = np.array([1, 2, 4, 6, 9, 0, 3])
for x in arr:
    if x > 3:
        filter.append(True)
    else
        filter.append(False)

newarr = arr[filter]
print(newarr)

  => [4 6 9]
```

PAGE NO.:

Can filter directly from array

```
arr = np.array([1, 2, 3, 4, 5, 6])
filter = (arr%2 == 0)
newarr = arr[filter]
print(newarr)
```

=> [2 4 6]

Random

Random Number
A number that can not be predicted by logic.

Pseudo random
if there is a program to generate random number, it can be predicted, thus it is not truly random.

Note Numbers generated through generation algo are pseudo random.

To make Truly random number we need to get the random data from outside source.
Keystrokes, mouse movements etc.

Truely random
We do not really require random random number unless it comes to security.

Generate Random Number

Numpy offers random module to work with numbers.

“from numpy import random”.

i To generate integer

```
x = random.randint(100)
```

=> This generates random number from 0 to 100.

ii To generate float

rand() returns random float between 0 - 1.

```
x = random.rand()
```

Integer array

randint() takes size parameter.

```
x = random.randint(100, size=(5))
```

Note- For 2 D array.

```
x = random.randint(100, size=(3, 5))
```

3 -> no of rows.

5 -> no of elements in each row.

float array

rand() takes arguement for shape of array.

x= random.rand(5)
=> creates 1D array with 5 values 0-1 each.

x = random.rand (2,5)
=> creates 2D array with 2 rows having 5 elements.

Random Number from Array

choic() method helps to do it.
returns a random value from the array passed to it.

import numpy as np
from numpy import random as rm

arr= np.array([1, 3, 7, 0, 10])

x = rm.choice (arr)
print (x)

=> returs a random value from arr.

size operator works here also.

=> x = rm.choice (arr, size=(3,5))

=> return a 3 array with 5 values each

Data Distribution

lists of all possibles values. and how often these values oc

Radom Distribution

Set of random number that follow certain probability density function.

choice() allows us to specify probability for each value.

Note Probability is 0-1. Sum of all probabilities should be 1.

Eg

```
import numpy as np.
from numpy import random as rm

arr= np.array([1, 3, 5, 7, 9])
prob=np.array([0.2, 0.1, 0.4, 0.0, 0.3])

x= rm.choice(arr, p=prob, size=(100))
print(x)
```

p -> It is for probability

=> creates 100 integers from arr and with given probabilities.

Note size can also be used for 2D arrays.

Iterating with different step size

```
arr= np.array([[1,2,3,4],[5,6,7,8]])
for x in np.nditer(arr[:, ::2])
print(x, end=" ")
```

```
=> 1 3 5 7
```

Enumerated iteration

mean mentioning sequence of numbers.
It is used to get index of element white iterating.
ndenumerator()

```
arr = np.array([1,2,3,4])
for idx, x in np.ndenumerate(arr):
    print(idx, x)
```

```
=>    0, 1
      1, 2
      2, 3
      3, 4.
```

Joining Array

means putting contents of two or more arrays into a single array.
We join by 'axes'.

concatenate() is used.

This problem is solved using nditer.

To iterate one every element of 3D array. Instead of using 3 nested loops we use.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in np.nditer(arr):
        print(x, end = " ")
```

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

Change data type while iterating

we use 'op_types' argument in nditer to change data type to desired type.

Now this needs some space to perform this action this extra is called buffer.
To enable it in nditer() we pass "flags=['buffered'].

```
arr = np.array([1, 2 3])
for x in np.nditer(arr, flags=['buffered'], op_types=['S']):
        print(x)
```

The above code will convert values to string type.

Note Order of arguements does not affect.

Iterating Arrays

Going through elements one by one.
Can be done through simple for loop.

1D array

```
arr = np.array([1,2,3])
for x in arr:
    print(x, end=" ")
```

```
=> 1 2 3
```

2D Array

```
arr = np.array([[1,2],[3,4]])
for (x in arr:
    print(x)
```

```
=> [1 2]
   [3 4]
```

Note To print all elements one by one use nested loops.

```
for x in arr:
    for y in x:
        print(y, end=" ")
```

```
=> 1 2 3 4
```

Note For n dimension we need n nested loops.

Note function reshape() is used.

Can convert to any dimention but no of elements should match.

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

The above code is wrong.

Reshape function returns a view.

Note We are allowed to have one unknown dimention. pass -1 and it will convert automatically.

```
arr= np.array([1,2,3,4,5,6])
print(arr.reshape(2,-1).base)
newarr= arr.reshape(2,-1)

 => [[1, 2, 3]
     [4,5,6]]
```

Flattering Array

Any array can be converted to 1D array.

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

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

Note All raws must have same no of elements.

```
np.array([[1,2],[2,4,5]]) => Error.
```

or use dtype object.

create a 5 dimension array

```
np.array([1,2,3], ndim=5)
print(arr.shape)
=> (1,1,1,1,3)
```

Reshaping Arrays
changing the shape of arrays.

i. change 1D to 2D

```
arr = np.array([1,2,3,4,5,6])
print("current shape :", arr.shape, ndim)
newarr = arr.reshape(3,2)
print(newarr, newarr.ndim)

=> 6, 1
   [[1 2]  2
    [3, 4]
    [5. 6]]
```

Note copy should not be affected by change in original array while view should be.

Q. check if Array own it data?

Numpy offers a attribute to check if array owns its data or not.
It is called 'base'
Returns 'None' if array owns data.
Return original object otherwise.

```
arr = np.array([1, 2, 3])
x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)

=> None
   [1 2 3]
```

Shape

- Number of elements in each dimension.
- 'shape' attribute returns tuple with each index having number of corresponding elements

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

=>   (2, 3)
```

```
import numpy as np
arr = np.array([1.1, 1.2])
newarr = arr.astype('i')
print(newarr, dtype)
```

```
=>   [1.1 1.2], int64
```

Array copy

- copy is a new array.
- copy owns the data and changes to it does not effect original array and vice versa.

```
arr = np.array([1, 2, 3])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
=>  [42 2 3]
    [1 2 3]
```

Array View

- view is just view of original array.
- view does not owns the data, any changes to it affect original array.

```
x = arr.view()              =>  [42  2 3]
arr[0] = 42                     [42, 2 3]
print(arr)
print(x)
```