Akshanshofficial@gmail.com

Hi guys this is Akshansh(+91 8384891269, akshanshofficial@gmail.com (mailto:akshanshofficial@gmail.com)). It has been quite a long time with you. We have discussed Python basics and covered almost everything in those lecture notes (from lecture 1 to lecture 15). Since Python Basics has been covered, we are hopping into NumPy.

NumPy Library

NumPy is a basics package for scientific computing with Python and especially for the data analysis. In fact, this library is the basis of a large amount of mathematical and scientific Python packages and among them, you will learn Pandas too.

Knowledge of the NumPy library is a prerequisite in order to face, in the best way, all scientific Python Packages, and partiularly to use and understand more about the pandas librabry and how to get most out of it. Pandas library will be main focused in coming lecture notes and that is useful too, if you want to get yourself hired by tech companies.

NumPy : A little history (you can skip it too, if you don't want to know the history of it)

When Python language was at its begining phase, developer found it hard to to work on numerical computation. Python didn't do much in terms of mathematics and scientific community.

First attempt was Numeric, developed by Jum Hugunin in 1995, which was successively followed by an alternative package called Numarray. Both packages were specialized for the calculations of arrays, and each of them had strengths depending on which case they were being used. Thus, they were used differently depending on where they showed more efficient. This ambiguity led then to the idea of uniifying the two packages and therefore Travis Oliphant started to develop the NumPy library. It's first release occurred in 2006.

It is currently most widely used package for the calculation in scientific organization in the world for the calculation of multidimentional arrays and large arrays. This allows you to perform operation on arrays in very efficient way and also perform high level mathematical calculation. This is open source and licensed under BSD.

How can you download NumPy?

mostly it is pre downloaded in most of Python distribution; however, if not, you can install it later.

On Linux (Ubuntu and Debian)

sudo apt-get install python-numpy

On Linux (Fedora)

sudo yum install numpy scipy

On windows with Anaconda

conda install numpy

Once NumPy is installed in in your distribution, to import the NumPy module within your Python session, write: import numpy as np

I am importing numpy as np, it is pre downloaded in my system

In [1]:

```
import numpy as np
```

I've successfully imported it and now whole package is available to play with. Let's get started gus.

Ndarray: the hear of the library

The whole NumPy library is based on one main object : ndarray (which stands for N-Dimentional array). This object is a multidimentional homogeneous (same like) array with a predetermined number of items:

homogeneous becuase virtually all the items within it are of the same type and same size. In fact, the data type is specified by another NumPy object called dtype; each ndarray is associated with only one type dtype.

Shape of an array is defined by the help of dimension and items. Dimensions are also called axes.

Million dollor question- What is array?

array, array array, what the f is array? You might caught up by this word or question I would say. So waht is array, if you have been a mathematical guy like me, it is an arrangement of items. Items are placed in rows and collumns in an array.

an arrangement of items in rows and columns is called array. Rows are horizontal and columns are veritcal.

In [ ]:

```
a = [1 1 1]
    [2 2 2]
    [3 3 3]
```

In above cell I've defined an array named 'a'. This is something you've seen in mathematical books of class

Akshanshofficial@gmail.com

12th and BSc, right? There are 3 rows and 3 columns in array. First row consists 1,1,1 and second row have 2,2,2 and third row consists 3,3,3. Similarly first column has 1,2,3 and so do second and third.

there are 3 rows and 3 columns in this array so it will be called, 3x3 dimentional array.

note this and always remember, size of an array is fixed, once you create it, and it can't be changed then. This behavior is different from python lists, which can grow or shrink in size.

How we define an array or ndarray (n dimentional array)?

the easiest way to define an array is use the array() function, passing a python list containing the elements to be inculded in it as an argument.

use np.array() function and list will be passed into it as input

In [2]:

```
a=np.array([1,2,3])
```

In [3]:

```
#let's print array
a
```

Out[3]:

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

You can check the type of 'a', which will definitely an array

In [4]:

```
type(a)
```

Out[4]:

```
numpy.ndarray
```

In order to know the associated dtype to the just created ndarray, you have to use the dtype attribute.

In [5]:

```
a.dtype
```

Out[5]:

```
dtype('int64')
```

The just created array has one axis, and then its rank is 1, while its shape should be (3,1) than means 3

Akshanshofficial@gmail.com

columns and 1 row.

you can use ndim attribute to know the axes (rows) of an array

In [6]:

```
a.ndim #print 1 becuase there is only 1 row
```

Out[6]:

```
1
```

use size attribute to know the columns in your array (lenght)

In [7]:

```
a.size #prints 3 becuase of 3 items
```

Out[7]:

```
3
```

use shape attribute to get the shape of array

In [8]:

```
a.shape #how many col each row has?
```

Out[8]:

```
(3,)
```

'a' was just one dimentiional array becuase it has 1 row, right. So how you create a 2D array?

In order to create a 2D array, just pass two list in it. Let's create a 2D array

In [9]:

```
b=np.array([[1.3,2.4],[0.3,4.1]])
```

observatios :

two lists was passed to create a 2D array. Note that, both of lists were passed within [].

let's print our two dimentional array

Akshanshofficial@gmail.com

In [10]:

```
b
```

Out[10]:

```
array([[1.3, 2.4],
       [0.3, 4.1]])
```

In [11]:

```
b.dtype  #checking type
```

Out[11]:

```
dtype('float64')
```

In [12]:

```
b.ndim  #checking dimensions
```

Out[12]:

```
2
```

In [13]:

```
b.size  #checking size
```

Out[13]:

```
4
```

In [14]:

```
b.shape  #checking shape, that means 2 col in row 1, 2 col in row 2
```

Out[14]:

```
(2, 2)
```

another important attribute is itemsize, which can be used with ndarray objects, It defines the size of array in bytes of each item in the array, and data is the buffer containing the actual element of the array.

In [15]:

```
b.itemsize
```

Out[15]:

```
8
```

In [16]:

```
b.data
```

Out[16]:

```
<memory at 0x7fa1859c3c18>
```

Akshanshofficial@gmail.com

create an array by different ways

method#1 lists in list

In [17]:

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

In [18]:

```
c
```

Out[18]:

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

method#2 tuple() in tuple

In [19]:

```
d=np.array(((1,2,3),(4,5,6)))
```

In [20]:

```
d
```

Out[20]:

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

method#3 tuple() in list[]

In [21]:

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

In [22]:

```
e
```

Out[22]:

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

Types of Data

so far you have used only numeric values as simple integer and flaot, but NumPy arrays are designed to contain a wide variety of data types. For example, you can use the data type string

Akshanshofficial@gmail.com

In [23]:

```
g=np.array([['a','b'],['c','d']])
```

In [24]:

```
g
```

Out[24]:

```
array([['a', 'b'],
       ['c', 'd']], dtype='<U1')
```

In [25]:

```
g.dtype
```

Out[25]:

```
dtype('<U1')
```

In [26]:

```
g.dtype.name
```

Out[26]:

```
'str32'
```

data type supported by NumPy : bool, int, float, complex

the dtype Option

The array() function does not accept a single argument. You have seen that each ndarray object object is associated with a dtype object that uniquely defines the typw of data that will occupy each item in the array. For example, if you want ot define an array with complex values you can use the dtype option as follows:

In [27]:

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

In [28]:

```
f
```

Out[28]:

```
array([[1.+0.j, 2.+0.j, 3.+0.j],
       [4.+0.j, 5.+0.j, 6.+0.j]])
```

complex number is another mathematical numeric term, that consist, real part and imaginary part. For example, if you write 3, that means, it is written 3r+0i - that means 3 real part and 0 imaginary part. and it is defined by iota (i) in mathematics. Please refer mathematics books, for more information on it.

Intrinsic Creation of an Array

Akshanshofficial@gmail.com

t s c C eat o o a ay

The NumPy library provides a set of function that generate the ndarrays with an initial content, created with some different values depending on function.

let's create an array of zeros with 3 rows and 3 cols

In [29]:

```
np.zeros((3,3))
```

Out[29]:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

let's create an array of ones with 3 rows and 3 cols

In [30]:

```
np.ones((3,3))
```

Out[30]:

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

there is another arange() function, that arange values for you in arrays.

In [31]:

```
np.arange(0,10)
```

Out[31]:

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

it start from 0 and ends with, (10-1=9)

In [32]:

```
np.arange(4,10)
```

Out[32]:

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

Akshanshofficial@gmail.com

In [33]:

```
np.arange(0,12,3)
```

Out[33]:

```
array([0, 3, 6, 9])
```

it looks like, it behaves as range function. It started from 0 and ends with 11(12-1) and gap is 3 between each number.

arange() creates 1D arrays, you can create 2D arrays too - let's create it

In [34]:

```
np.arange(0,12).reshape(3,4)
```

Out[34]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Observe that array start from 0 and ends at 12-1 =11. reshape() gives you the shape, in rows and cols. reshape(3,4) means 3 rows and 4 cols.

there is another function similar to arange() and that is linspace(). It divides 1D array in equal parts.

In [35]:

```
np.linspace(0,10,5)
```

Out[35]:

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

this means, starting from 0 to 10, and divide it into 5 equal parts.

how to get an array with random number

to get the random numbers you've to look into random module. and thenn you've to access, random() method

In [36]:

```
np.random.random(3)
```

Out[36]:

```
array([0.90808386, 0.86705683, 0.74257682])
```

Akshanshofficial@gmail.com

In [37]:

```
np.random.random((3,3))
```

Out[37]:

```
array([[0.77341135, 0.5641283 , 0.98038082],
       [0.78492952, 0.36703246, 0.78270772],
       [0.09814195, 0.63389811, 0.55786254]])
```

Basic Operations on array

let's get an array

In [38]:

```
a=np.arange(4)
```

In [39]:

```
a
```

Out[39]:

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

let's add 4 to each element

In [40]:

```
a+4
```

Out[40]:

```
array([4, 5, 6, 7])
```

however, a still remains same, let's print it. (if you want to save a+4 value you can use b=a+4 to save it)

In [41]:

```
a
```

Out[41]:

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

In [42]:

```
b=np.arange(4,8)
```

Akshanshofficial@gmail.com

In [43]:

```
b
```

Out[43]:

```
array([4, 5, 6, 7])
```

let's add a and b together

In [44]:

```
a+b
```

Out[44]:

```
array([ 4,  6,  8, 10])
```

In [45]:

```
a-b
```

Out[45]:

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

In [46]:

```
a*b
```

Out[46]:

```
array([ 0,  5, 12, 21])
```

morever, these operators are also available for functions, provided that the value returned is a NumPy array. For example, you can multiply the array with the sine or the square root of the elements of the array b. Let's try this.

In [47]:

```
a*np.sin(b)
```

Out[47]:

```
array([-0.        , -0.95892427, -0.558831  ,  1.9709598 ])
```

In [48]:

```
a*np.sqrt(b)
```

Out[48]:

```
array([0.        , 2.23606798, 4.89897949, 7.93725393])
```

let's perform these aithmatic operations in multidimentional arrays

Akshanshofficial@gmail.com

In [49]:

```
A=np.arange(0,9).reshape(3,3)
```

In [50]:

```
A
```

Out[50]:

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

In [51]:

```
B=np.ones((3,3))
```

In [52]:

```
B
```

Out[52]:

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

In [53]:

```
A+B
```

Out[53]:

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

In [54]:

```
B-A
```

Out[54]:

```
array([[ 1.,  0., -1.],
       [-2., -3., -4.],
       [-5., -6., -7.]])
```

multiplication of two arrays, can be tricky for you if you haven't been a mathematics student for a while. I suggest you to watch a youtube tutorial if you don't know how arrays multiplication work.

```
Well, let me explain multiplication too. Let's suppose you have two arrays P
and Q.Element on row 1 and col1 of P will multiply with elelment of row1 and
col1 in array Q.
```

corresponding elements gets multiplied in arrays.

Akshanshofficial@gmail.com

In [65]:

```
A*B
```

Out[65]:

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

In [66]:

```
B*A
```

Out[66]:

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

In [67]:

```
p=np.arange(1,10).reshape(3,3)
```

In [68]:

```
p
```

Out[68]:

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

In [69]:

```
q=np.arange(10,19).reshape(3,3)
```

In [70]:

```
q
```

Out[70]:

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

In [71]:

```
p*q
```

Out[71]:

```
array([[ 10,  22,  36],
       [ 52,  70,  90],
       [112, 136, 162]])
```

Akshanshofficial@gmail.com

In [72]:

```
q*p
```

Out[72]:

```
array([[ 10,  22,  36],
       [ 52,  70,  90],
       [112, 136, 162]])
```

Matrix Product

In [77]:

```
p
```

Out[77]:

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

In [79]:

```
q
```

Out[79]:

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

You guys remember matrix product(multiplication)? Noh??? Watch a video on you tube on how matrix product works?

Suppose you have two matrices (plural of matrix, kind of array). YOu have to find A*B so what happens in this case, first row of A multiply with first column of B (each element multiple with corresponding element and then we add all) and this is how, first element of row 1 (in* AB) is found and then first row of A multiply with col2 of B and so on.

In [ ]:

```
let's multiply p*q
[(1*10+2*13+3*16) (1*11+2*14+3*17) (1*12+2*15+3*18)] #row1 of p with each col of
[(4*10+5*13+6*16) (4*11+5*14+6*17) (4*12+5*15+6*18)] #row2 of p wilh cols of q
[(7*10+8*13+9*16) (7*11+8*14+9*17) (7*12+8*15+9*18)] #row3 of p with cols of q
```

```
examine how rows multiply with cols, in matrix multiplication. Here in NumPy,
you don't have to do that shitty calculation. use dot() method for that.
```

Akshanshofficial@gmail.com

In [80]:

```
np.dot(p,q)
```

Out[80]:

```
array([[ 84,  90,  96],
       [201, 216, 231],
       [318, 342, 366]])
```

In [81]:

```
np.dot(q,p)
```

Out[81]:

```
array([[138, 171, 204],
       [174, 216, 258],
       [210, 261, 312]])
```

see in matrix p*q and q*p is not eqqual so does here too.

Universal functions (ufunc)

if a function act individually on each element of array, it is called universal for all or universal fucntion. Simply say ufunc.

there are many mathematical and trigonometric operations those meet this definition, for example the calculation of square root with sqrt() and logarithm with log() and sin with sin().

In [82]:

```
a=np.arange(1,5)
```

In [83]:

```
a
```

Out[83]:

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

In [84]:

```
np.sqrt(a)
```

Out[84]:

```
array([1.        , 1.41421356, 1.73205081, 2.        ])
```

Akshanshofficial@gmail.com

In [85]:

```
np.log(a)
```

Out[85]:

```
array([0.        , 0.69314718, 1.09861229, 1.38629436])
```

In [86]:

```
np.sin(a)
```

Out[86]:

```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

aggregate function

well universal function works on every individual element. There are many functions which consider whole array as a data set and produce a single result. These functions are called aggregate function.

In [87]:

```
a=np.arange(1,9)
```

In [88]:

```
a
```

Out[88]:

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

In [89]:

```
a.sum()
```

Out[89]:

```
36
```

In [90]:

```
a.min()
```

Out[90]:

```
1
```

In [92]:

```
a.max()
```

Out[92]:

```
8
```

Akshanshofficial@gmail.com

In [93]:

```
a.mean()
```

Out[93]:

```
4.5
```

In [94]:

```
a.std()
```

Out[94]:

```
2.29128784747792
```

Indexing, slicing and iterating

so far now we have learnt how to create an array and how to perform operation on arrays.

Now its time to manipulate (update/play) those arrays.

Indexing

arrays always refers to the use of square brackets ('[]')to index the elements of the array so that it can then be referred individually for various uses such as extracting a value, selecting items, or even assigning a new value.

In [96]:

```
a=np.arange(10,16)
```

In [97]:

```
a
```

Out[97]:

```
array([10, 11, 12, 13, 14, 15])
```

In [98]:

```
a[0]  #prints item at 0 index
```

Out[98]:

```
10
```

In [99]:

```
a[2]  #prints item at 2 index
```

Out[99]:

```
12
```

Akshanshofficial@gmail.com

In [101]:

```
a[5] #prints item at 5 index
```

Out[101]:

```
15
```

In [102]:

```
a[6] #error because max index here is 5
```

```
---------------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
 last)
<ipython-input-102-84c1e027215d> in <module>
----> 1 a[6]

IndexError: index 6 is out of bounds for axis 0 with size 6
```

In [103]:

```
a[-1] #prints 1st element from last
```

Out[103]:

```
15
```

In [104]:

```
a[-2] #prints 2nd element from last
```

Out[104]:

```
14
```

hope you got it now.

what is you want to know multiple items of the array?

In [105]:

```
a[[1,2,-1]]
```

Out[105]:

```
array([11, 12, 15])
```

see 1 index = 11, 2nd index =12, 1st index from last is =15

GOTCHAAA? I hope so.

accessing element in 2D arrays-

Akshanshofficial@gmail.com

In [106]:

```
A
```

Out[106]:

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

```
rows index start from ), so row1 has index 0, ro2 has index 1, row 3 has
index 2. Same goes for the cols too
```

In [109]:

```
A[0,0]  #0 index row, 0 index col
```

Out[109]:

```
0
```

In [110]:

```
A[0,1] #0 index row, 1 index col
```

Out[110]:

```
1
```

In [111]:

```
A[2,1]  #row index2, col index1
```

Out[111]:

```
7
```

slicing in arrays

you remember slicing? We started slicing things and stuff from lecture number 1 string. It cuts something. Slicing allows you to extract portion of array to generate new one.

In [112]:

```
a=np.arange(10,16)
```

In [113]:

```
a
```

Out[113]:

```
array([10, 11, 12, 13, 14, 15])
```

Akshanshofficial@gmail.com

In [114]:

```
a[1:5]  #prints from index 1 to 4 (5-1)
```

Out[114]:

```
array([11, 12, 13, 14])
```

In [115]:

```
a[0:3]  #prints from index 0 to 2 (3-1)
```

Out[115]:

```
array([10, 11, 12])
```

In [117]:

```
a[0:5:2]  #prints from index 0 to 5 with a gap of 2
```

Out[117]:

```
array([10, 12, 14])
```

lets play with 2 D array

In [118]:

```
A=np.arange(10,19).reshape((3,3))
```

In [119]:

```
A
```

Out[119]:

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

suppose you want to print row index 0

In [121]:

```
A[0]  #prints row 0
```

Out[121]:

```
array([10, 11, 12])
```

this is not a suitable way to print row, becuase you have to print cols too

Akshanshofficial@gmail.com

In [122]:

```
A[0, :] #prints row index 0, from index 0 to end
```

Out[122]:

```
array([10, 11, 12])
```

In [124]:

```
A[0, 1:3] #prints row index 0, from index 1 to 2 (3-1)
```

Out[124]:

```
array([11, 12])
```

In [126]:

```
A[:,0] #prints col0, from index 0 to end
```

Out[126]:

```
array([10, 13, 16])
```

In [128]:

```
A[1:3,0] #prints col0, from index 1 to 2 (3-1)
```

Out[128]:

```
array([13, 16])
```

What if you want to extract a small portion out of given array

Well if you want to make an small array from the bigger array. You can do it, syntax goes like this-

In [ ]:

```
array_name[rowsIndexSlicing, colIndexSlicing ]
```

suppose we want to extract an array, with rows index 0 to 1 (we have to write 2, because 2-1=1), and col index 0 to 1

In [129]:

```
A[0:2, 0:2]
```

Out[129]:

```
array([[10, 11],
       [13, 14]])
```

Observe that, in this small array which is extracted from bigger array A, rows index 0 and index1 is there so does for the columns

iterating in array

Akshanshofficial@gmail.com

In [130]:

```
a=np.arange(1,9)
```

In [131]:

```
a
```

Out[131]:

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

for iterating, we have to use "for" loop that will help us accessing each value/item of a

In [133]:

```
for element in a:
    print (element)
```

```
1
2
3
4
5
6
7
8
```

see each element was accessed by this.

Let's play with 2D array

In [134]:

```
A
```

Out[134]:

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

In [139]:

```
for row in A:
    print(row)
```

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

this didn't work as we though, because it is more complicated. A 2D array has multiple rows and multiple cols unlike 1D array.

in this we have to use two nested for loop, 1st loop will scan the rows and second loop will scan columns.

Akshanshofficial@gmail.com

If you want to make an iteration element by element you may use the following construct, using the for loop in A.flat

In [145]:

```
for item in A.flat:
    print(item)
```

```
10
11
12
13
14
15
16
17
18
```

it becomes difficult to map an array, so NumPy offers us an alternative and more elegant solution than the for loop.

most of the time, you have to apply function on specific rows or cols or individual item. Numpy offers apply_along_axis() function,

apply_along_axis() takes three arguments, the aggregate function, axis and last is Array.

If the option axis equal 0, then the iteration evaluates the elements columns by columns, if the axis equal 1, that means evaluate the elements rows by rows

axis 0, means columns by columns, axis 1 means rows by rows

In [146]:

```
A
```

Out[146]:

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

let's use apply_along_axis() function, on cols

In [147]:

```
np.apply_along_axis(np.mean, axis=0, arr=A)
```

Out[147]:

```
array([13., 14., 15.])
```

since axis was 0, it means, np.mean was applied on col by col

Akshansh official@gmail.com

let's use apply_along_axis() function, on rows

In [148]:

```
np.apply_along_axis(np.mean, axis=1, arr=A)
```

Out[148]:

```
array([11., 14., 17.])
```

since axis was 1, it means, np.mean was applied on row by row

watch this

In [149]:

```
def half(x):
    return x/2
```

In [150]:

```
np.apply_along_axis(half,axis=1,arr=A)
```

Out[150]:

```
array([[5. , 5.5, 6. ],
       [6.5, 7. , 7.5],
       [8. , 8.5, 9. ]])
```

In [151]:

```
np.apply_along_axis(half,axis=0,arr=A)
```

Out[151]:

```
array([[5. , 5.5, 6. ],
       [6.5, 7. , 7.5],
       [8. , 8.5, 9. ]])
```

i created a function named half, and passed into apply_along_axis

conditions and boolean arrays

So far we have used indexing and slicing to select a subset of the array. These methods make use of the indexes in a nimerical form. An alternative way to perform the selective extraction of the elements in an array is to use the conditions and boolean operators.

In [152]:

```
A=np.random.random((4,4))
```

Akshanshofficial@gmail.com

In [153]:

```
A
```

Out[153]:

```
array([[0.77823652, 0.48624403, 0.30008284, 0.05309543],
       [0.99160348, 0.35671566, 0.46585385, 0.34777192],
       [0.30643044, 0.94967883, 0.27016144, 0.28740148],
       [0.14074433, 0.0354593 , 0.3269285 , 0.19448532]])
```

In [160]:

```
A>0.5  #looking which one is greater than 0.5
```

Out[160]:

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

let's find all the numbers in arrays which is greater than 0.5

In [161]:

```
A[A>0.5]
```

Out[161]:

```
array([0.77823652, 0.99160348, 0.94967883])
```

In [162]:

```
type(A[A>0.5])  #checking type of it
```

Out[162]:

```
numpy.ndarray
```

Shape Manipulation

You have already seen during the creation of a two-dimentional array how it is possible to convert a one dimentional array into a matrix, thanks to reshape() function.

In [163]:

```
a=np.random.random(12)
```

Akshanshofficial@gmail.com

In [164]:

```
a
```

Out[164]:

```
array([0.38181056, 0.78692012, 0.15839377, 0.36105482, 0.79859204,
       0.41430448, 0.22449197, 0.01782456, 0.40484321, 0.71362944,
       0.47032253, 0.23990565])
```

In [167]:

```
A=a.reshape(3,4)  #12 elements are there 3*4=12
```

In [168]:

```
A
```

Out[168]:

```
array([[0.38181056, 0.78692012, 0.15839377, 0.36105482],
       [0.79859204, 0.41430448, 0.22449197, 0.01782456],
       [0.40484321, 0.71362944, 0.47032253, 0.23990565]])
```

The reshape() function returns a new array and therefore it is useful to create new objects. However if you want to modify the object by modifying the shape, you have to assign a tuple containing the new dimensions directly to its shape attribute. Let me show you how it is done.

In [169]:

```
a.shape=(3,4)
```

In [170]:

```
a
```

Out[170]:

```
array([[0.38181056, 0.78692012, 0.15839377, 0.36105482],
       [0.79859204, 0.41430448, 0.22449197, 0.01782456],
       [0.40484321, 0.71362944, 0.47032253, 0.23990565]])
```

as you can see, this time it is the starting array to change shape and there is no object returned. The inverse operation is possible, that is to convert a two-dimentional array into a one-dimentional array, through the ravel() function.

you can see that a is two dimentional array, let's create it again, 1D array again using ravel() function

In [171]:

```
a=a.ravel()
```

Akshanshofficial@gmail.com

In [175]:

```
a
```

Out[175]:

```
array([0.38181056, 0.78692012, 0.15839377, 0.36105482, 0.79859204,
       0.41430448, 0.22449197, 0.01782456, 0.40484321, 0.71362944,
       0.47032253, 0.23990565])
```

see that it has been again converted to one dimentional array again.

we can change the shape of array too

In [179]:

```
a.shape=12
```

In [180]:

```
a
```

Out[180]:

```
array([0.38181056, 0.78692012, 0.15839377, 0.36105482, 0.79859204,
       0.41430448, 0.22449197, 0.01782456, 0.40484321, 0.71362944,
       0.47032253, 0.23990565])
```

Transpose() method to change rows into cols and cols and rows

In [182]:

```
s=np.arange(12).reshape(3,4)
```

In [183]:

```
s
```

Out[183]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [184]:

```
s.transpose()
```

Out[184]:

```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

You can see that row1 has become col1 now. rows has made cols and cols has become rows

Akshanshofficial@gmail.com

Array Manipulation

Some times you have to create an array using already created array. In this section you will see how to create new arrays by joining or splitting arrays those are already created or defined.

Joining arrays

You can merge multiple arrays to form a new one that contains all of them togetger. This is called concept of stacking.

there are two types of stacking, vertical stacking vstack and horrizontal stacking hstack

In [185]:

```
A=np.ones((3,3))
```

In [186]:

```
A
```

Out[186]:

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

In [187]:

```
B=np.zeros((3,3))
```

In [188]:

```
B
```

Out[188]:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [189]:

```
np.vstack((A,B))
```

Out[189]:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

Akshahofficial@gmail.com

In [190]:

```
np.hstack((A,B))
```

Out[190]:

```
array([[1., 1., 1., 0., 0., 0.],
       [1., 1., 1., 0., 0., 0.],
       [1., 1., 1., 0., 0., 0.]])
```

use column_stack() or row_stack() for one dimentional array

In [191]:

```
a=np.array([0,1,2])
```

In [192]:

```
b=np.array([3,4,5])
```

In [193]:

```
c=np.array([6,7,8])
```

let's print them all

In [197]:

```
a,b,b
```

Out[197]:

```
(array([0, 1, 2]), array([3, 4, 5]), array([3, 4, 5]))
```

In [198]:

```
np.column_stack((a,b,c))
```

Out[198]:

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

In [199]:

```
np.row_stack((a,b,c))
```

Out[199]:

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

spliting arrays

Akshanshofficial@gmail.com

first you saw how to assemble multiple arrays to each other through the opweation of stacking. Now let's see how to split them.

In [200]:

```
A=np.arange(16).reshape(4,4)
```

In [201]:

```
A
```

Out[201]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

using hsplit()

if you want to horrizonatally split array, meaning that width of the array divided into two parts, A was like *44 and other parts will be 24*

In [202]:

```
[B,C]=np.hsplit(A,2)
```

In [203]:

```
B
```

Out[203]:

```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
```

In [204]:

```
C
```

Out[204]:

```
array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])
```

using vslipt()

if we vertically split,it means height will be divided into two parts.

Akshanshofficial@gmail.com

In [208]:

```
[D,E]=np.vsplit(A,2)
```

In [209]:

```
D
```

Out[209]:

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

In [210]:

```
E
```

Out[210]:

```
array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

split() function for more efficiency ¶

if you want to split the array into nonsymmetrical parts. In addition, passing the array as an argument, you have also to specify the indexes of the parts to be divide,

if you use option axis =1, then the indexes will be those of columns and if you use axis =0, row indexes will be impacted

In [214]:

```
A
```

Out[214]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

if you want to divide the matrix into three partsm the first of which will include the first column, second will include the second and thrird column, thrid one will include last column. Then you must specify three indexes in the following way.

In [221]:

```
[A1,A2,A3]=np.split(A,[1,3], axis=1)
```

Akshanshofficial@gmail.com

In [222]:

```
A1
```

Out[222]:

```
array([[ 0],
       [ 4],
       [ 8],
       [12]])
```

In [223]:

```
A2
```

Out[223]:

```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]])
```

In [224]:

```
A3
```

Out[224]:

```
array([[ 3],
       [ 7],
       [11],
       [15]])
```

you can do same thing by row.

In [225]:

```
[A1,A2,A3]=np.split(A,[1,3], axis=0)
```

In [226]:

```
A1
```

Out[226]:

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

In [227]:

```
A2
```

Out[227]:

```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Akshanshofficial@gmail.com

In [228]:

```
A3
```

Out[228]:

```
array([[12, 13, 14, 15]])
```

this feature also includes vsplit() and hsplit() functions.

Reading and Writing Array Data on Files

A very important aspect of NumPy that has not been taken into account yet is the reading of the data contained within a file. This procedure is very useful, especially when you have to deal with large amounts of data collected within arrays. This is a very common operation in data analysis, since the size of the dataset to be analyzed is almost always huge, and therefore it is not advisable or even possible to manage the transcription and subsequent reading of data by a computer to another manually, or from one session of the calculation to another.

Indeed NumPy provides in this regard a set of functions that allow the data analyst to save the results of his calculations in a text or binary file. Similarly, NumPy allows reading and conversion of written data within a file into an array.

Loading and Saving Data in Binary Files

In [229]:

```
A=np.arange(12).reshape(4,3)
```

In [230]:

```
A
```

Out[230]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

let's save data

In [231]:

```
np.save('saved_data', A)
```

let's load our saved data, make sure to use .npy at the last

Akshanshofficial@gmail.com

In [232]:

```
loaded_data=np.load('saved_data.npy')
```

In [233]:

```
loaded_data
```

Out[233]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

### Uffffff, finally it has been done. It already took 5+ hours me to sit down to make a proper lecture for you guys. It was hell of a time. I collected all the important features of NumPy at once. I hope you will find this helpful. My name is Akshansh (+91 8384891269, akshanshofficial@gmail.com) and my time is up for now. Keep pythoning guys. Stay safe and stay away from covid-19. Make your qurantine days count.

Akshanshofficial@gmail.com