Numpy Basics: Arrays and Vectorized Computation

Agenda:

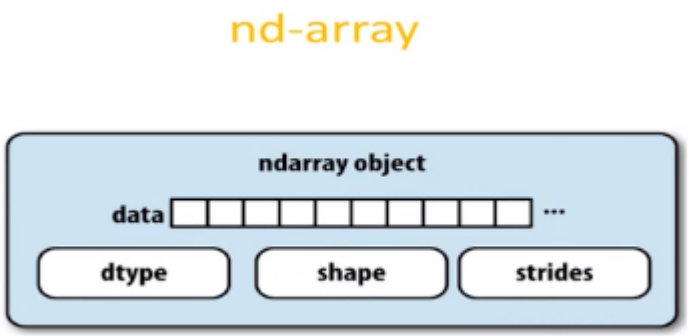
You will learn about `Numpy List` , `multidimensional array` , `arithmetic ndarray` , `indexing and slicing` , `element wise array` , `np.where`

Video#1. Numpy Introduction

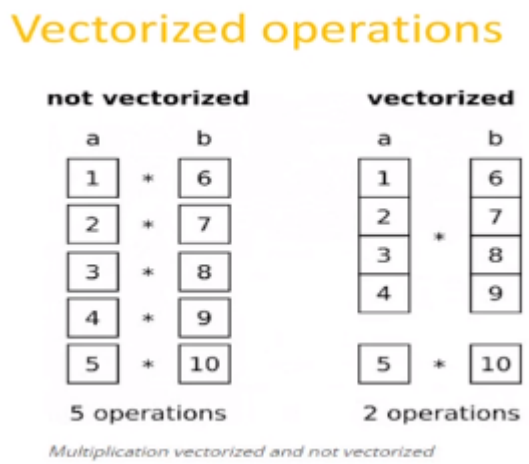
Introduction to Numpy

- Numerical Python
- Developed in 2005 by Travis Oliphant
- Lingua franca for data exchange
- ndarray – a n-dimensional array
- Fast operations on entire arrays
- Reading/writing array data
- Linear algebra operations

1. `lingua franca` (means medium of communication)



1. Yaani jobhi Numpy mai hum data store karte hain wo `n dimensional array (ndarray)` mai store hoga

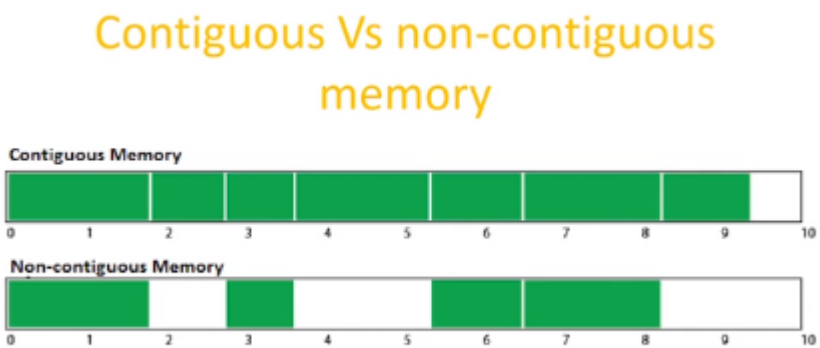


1. yaani agar hum python mai kaam karenge to humein loops ka sahaara lena parega, jabke yehi kaam hum Numpy mai karna chahen to hum ye single statement mai karsakte hain

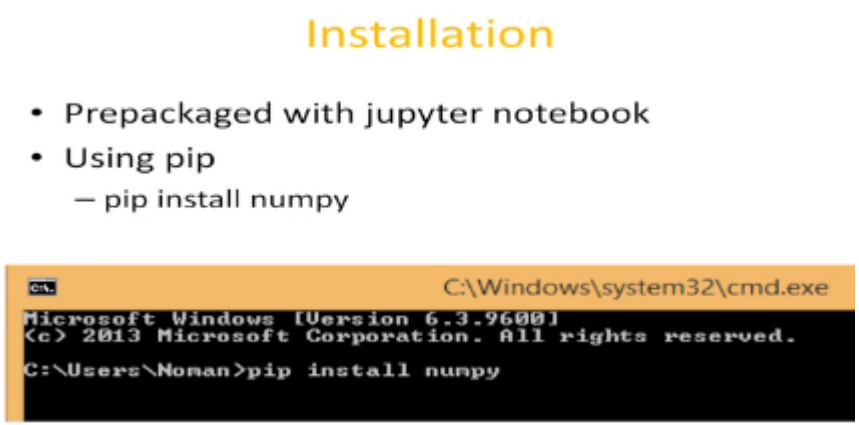
Why Numpy?

- NumPy internally stores data in a contiguous block of memory
- Complex computations on entire arrays

1. `contiguous` means sharing a common border; touching




```
 agar appne 100 by 100 ki array banayi hai to jab usko access karenge to yaani OS usko access karne jayega
to wo ubko 1 saath le ayega qk wo 1 saath parhe hue hain, usko memry mai 1 jaga sai doosri jaga move nahi
karna parhta,1 hee jaga sai usko data mil jata hai  jabke agar hum list ki baat karenge to list jo hai wo
dynamic arrays hote hain usmai kuch portion of data kisi jaga parh wa hai to kuch portion of data kahin
or parha wa hai to apne data doosri jaga wala access karna hai to apko doosri jaga jaana parega to ismai
time lagega
```



Video#2. Numpy VS Python List

In [4]:
```
list1 = range(1000000)
list1
```

Out[4]:
```
range(0, 1000000)
```

In [2]:
```
import numpy as np
```

In [6]:
```
arr1 = np.arange(1000000)
arr1
```

Out[6]:
```
array([     0,      1,      2, ..., 999997, 999998, 999999])
```

In [7]:
```
%timeit for i in range(1, 10): r = [x * 2 for x in list1]
```

```
4.55 s ± 1.45 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [8]:
```
%timeit for i in range(1, 10): r = arr1 * 2
```

```
47.4 ms ± 4.08 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Video#3. A Multidimensional Array Object

In [9]:
```
import numpy as np
```

Differnet Methods of Numpy:

1. np.zeros()

- is sai Numpy ki 1 array create hogi jismai saare elements zero honge, kis size ki array banani hai mujhe wo bhi batana parega
- e.g agar mai 4 by 4 ki array banana chahun to mai tuple dunga jiske through mai size dunga of the array

In [10]:
```
zerosArr = np.zeros((4, 4))
zerosArr
```

Out[10]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

2. np.ones()

- mai chahun to ones ki array bhi banasakta hun

In [11]:
```
onesArr = np.ones((5, 5))
onesArr
```

Out[11]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

3. np.empty()

- ismai desired shape kiarray create hojayegi or usmain data randomly kuch bhi hosakta hai
- ye kuch bhi randomly return karsakta hai, randomly kuch bhi data memory sai utha sakta hai, lekin desired shape ki array create hojayegi

In [16]:
```
emptyArr = np.empty((7,7))
emptyArr
```

Out[16]:
```
array([[1.86707675e-293, 8.09477154e-320, 2.47032823e-323,
        4.37186310e-297, 1.99090412e-316, 0.00000000e+000,
        1.97626258e-323],
       [8.50789222e-314, 0.00000000e+000, 0.00000000e+000,
        0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
        0.00000000e+000],
       [1.48539705e-313, 0.00000000e+000, 2.12199579e-314,
        1.48219694e-323, 0.00000000e+000, 1.81409836e-314,
        1.69759663e-313],
       [0.00000000e+000, 1.61862269e-316, 0.00000000e+000,
        4.37234008e-297, 0.00000000e+000, 2.12199579e-314,
        0.00000000e+000],
       [4.37244504e-297, 1.99091400e-316, 4.37248007e-297,
        0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
        0.00000000e+000],
       [0.00000000e+000, 4.37262006e-297, 0.00000000e+000,
        4.37266672e-297, 0.00000000e+000, 4.37271339e-297,
        1.99090708e-316],
       [0.00000000e+000, 1.17134168e-311, 5.14673969e-316,
        4.44933506e-308, 1.53188514e-305, 0.00000000e+000,
        0.00000000e+000]])
```

4. np.array(listName)

- is method ke zariye mai python list ko array mai convert karsakta hun

In [18]:
```
list1 = [1,2,3,4,5]
list1
```

Out[18]:
```
[1, 2, 3, 4, 5]
```

In [20]:
```
listToArr = np.array(list1)
listToArr
```

Out[20]:
```
array([1, 2, 3, 4, 5])
```

5. np.arange()

- same as range in python

```
In [21]: np.arange(1,100,10)
Out[21]: array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91])
```

Video#4. Arithmetic with ndarray

Vectorization

- Express batch operations on data without writing any for loops
- Any arithmetic operations between equal-size arrays applies the operation element-wise
- Arithmetic operations with scalars propagate the scalar argument to each element in the array

Operations on two metrics

```
In [51]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])

In [52]: arr
Out[52]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])

In [53]: arr * arr
Out[53]:
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])

In [54]: arr - arr
Out[54]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Operations on metrics and scalar

```
In [55]: 1 / arr
Out[55]:
array([[ 1.    ,  0.5   ,  0.3333],
       [ 0.25  ,  0.2   ,  0.1667]])

In [56]: arr ** 0.5
Out[56]:
array([[ 1.    ,  1.4142,  1.7321],
       [ 2.    ,  2.2361,  2.4495]])
```

- agar hum scalar add karayenge to wo element wise add hojayega

Comparison between metrics

```
In [57]: arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])

In [58]: arr2
Out[58]:
array([[  0.,   4.,   1.],
       [  7.,   2.,  12.]])

In [59]: arr2 > arr
Out[59]:
array([[False,  True, False],
       [ True, False,  True]], dtype=bool)
```

Practical

Generating random number array using `random.randn()`

In [2]:
```
randArr1 = np.random.randn((10))
randArr1
```

Out[2]:
```
array([ 0.23646443,  0.71764763, -0.80207094, -1.09564134, -0.81579061,
       -0.21666896,  1.81631404,  0.32526392,  0.77420252, -0.63961136])
```

In [3]:
```
randArr2 = np.random.randn((10))
randArr2
```

Out[3]:
```
array([ 0.56845388,  0.74420614, -0.35037112, -0.81533935, -0.58446462,
        0.80525824, -1.60845304, -0.58053357, -0.34321737,  1.57067003])
```

Creating 2 arrays to perform vectorization:

In [27]:
```
arr1 = np.arange(2, 10, 2)
arr1
```

Out[27]:
```
array([2, 4, 6, 8])
```

In [28]:
```
arr2 = np.arange(2, 10, 2)
arr2
```

Out[28]:
```
array([2, 4, 6, 8])
```

Adding two arrays:

In [29]:
```
add = arr1 + arr2
add
```

Out[29]:
```
array([ 4,  8, 12, 16])
```

Subtracting two arrays:

In [30]:
```
print(arr1)
print(arr2)
```

```
[2 4 6 8]
[2 4 6 8]
```

In [31]:
```
sub = arr1 - arr2
sub
```

Out[31]:
```
array([0, 0, 0, 0])
```

Multiplying two arrays:

In [32]:
```
mult = arr1 * arr2
mult
```

Out[32]:
```
array([ 4, 16, 36, 64])
```

Dividing two arrays:

In [33]:
```
div = arr1 / arr2
div
```

Out[33]:
```
array([1., 1., 1., 1.])
```

Multiplying an array by `scalar` value:

In [34]:
```
print(arr1)
print(arr2)
```

```
[2 4 6 8]
[2 4 6 8]
```

In [35]:
```
multByScalar1 = 2 * arr1
multByScalar1
```

Out[35]:
```
array([ 4,  8, 12, 16])
```

inverse a matrix using scalar:

In [36]:
```
print(arr1)
print(arr2)
```

```
[2 4 6 8]
[2 4 6 8]
```

In [37]:
```
inverseMatrix1 = 1 / arr1
inverseMatrix1
```

Out[37]:
```
array([0.5       , 0.25      , 0.16666667, 0.125     ])
```

Check which elements in array are greater than 0 :

In [40]:
```
# since all elements are greater than 0, it is returning true
arr1 > 0
```

Out[40]:
```
array([ True,  True,  True,  True])
```

Acces elements who are are greater than 0:

In [41]:
```
# agar numpy array ko as bool array pass kren to jo true hoti hain wo select hokar ajati hain

arr1[arr1 > 0]
```

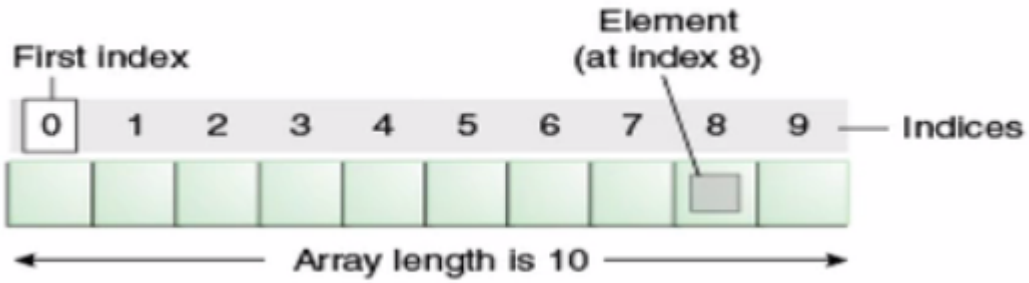
Out[41]:
```
array([2, 4, 6, 8])
```

Video#5. Indexing & Slicing

Indexing

- Acts similarly to Python lists
- Boolean indexing: specifying boolean arrays as index
- Fancy indexing: specifying arbitrary arrays as indexes

Array indexing

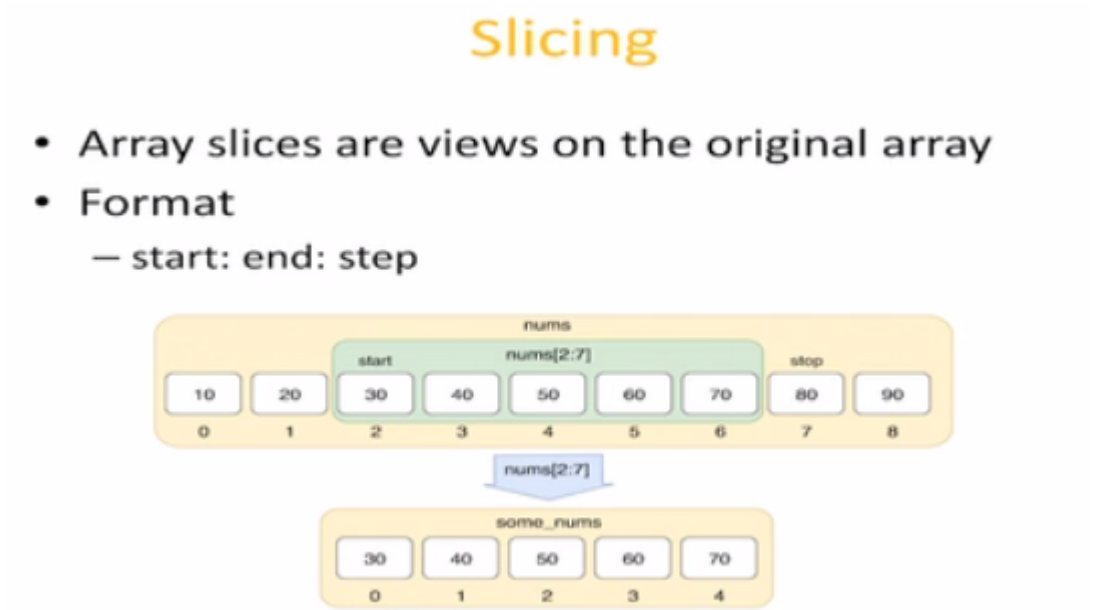


Boolean indexing

```
In [103]: data[names == 'Bob']
Out[103]:
array([[ 0.0929,  0.2817,  0.769 ,  1.2464],
       [ 1.669 , -0.4386, -0.5397,  0.477 ]])
```

Fancy indexing

```
In [120]: arr[[4, 3, 0, 6]]
```



Practical

In [42]:
```
arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
arr1
```

Out[42]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [43]:
```
arr1[4]
```

Out[43]:
```
5
```

Boolean Indexing:

In [45]:
```
# Accessing elements greater than 5
arr1[arr1 > 5]
```

Out[45]:
```
array([ 6,  7,  8,  9, 10])
```

Fancy Indexing:

In [46]:
```
# agar mai isko 1 list pass kardun

arr1[[2,1]]
```

Out[46]:
```
array([3, 2])
```

Accesing multidimensional `ndarray`:

In [47]:
```
# creating random array of 10 by 10

randArr = np.random.randn(10, 10)
randArr
```

Out[47]:
```
array([[-0.34647994,  1.47873681,  1.48648942,  0.48403995, -0.65743178,
        -1.87247429, -0.24304981,  0.62528388,  0.21060757,  1.31804202],
       [-0.27605532, -1.32816176, -1.12057406, -0.27557991,  1.77297   ,
         0.49613473, -0.68572188, -1.1733908 ,  1.27931125,  0.32895828],
       [ 0.47621513,  0.50018049,  0.48592307, -1.23050001, -2.30959656,
         2.09728241,  0.59949155, -2.00111817,  0.63526908, -1.0349937 ],
       [ 0.40574129,  0.03789466, -1.25293998,  0.05914265,  1.03218674,
         1.87752573, -0.17883648,  0.30673535,  0.19347169,  0.09450876],
       [ 0.36716884, -0.08360576,  0.30381209, -0.8661162 , -1.01435714,
        -0.66672709, -2.03759716,  0.2458695 , -1.25313254, -1.55349758],
       [ 0.90480491, -1.65448128, -0.75918036,  1.77589396,  0.41453769,
        -0.74658148, -0.0726056 ,  0.25405621,  0.2192515 ,  0.22484674],
       [-0.41390329,  0.41421575,  0.56691819, -0.01171048, -0.46741641,
        -0.88209894,  0.83835519, -1.22525662,  1.40636964, -0.3299354 ],
       [-0.46245022, -0.98868969,  0.63272828, -1.17187119, -0.095704  ,
        -0.39786988,  0.10235525, -0.31598198,  0.30828205, -0.90354403],
       [ 0.23171289,  1.59016037, -0.33746377, -1.15052456, -0.13950785,
        -0.67279502,  1.16355092,  1.01115322,  0.40091719,  0.67759967],
       [-0.43284755,  1.07843767, -1.61222049, -2.08203957,  0.51107326,
        -0.38028299, -0.23751739, -0.64413602, -0.07391223, -0.0529483 ]])
```

Accessing 1st row from ndarry:

In [48]:
```
# 1st row uth kar ajayegi

randArr[0]
```

Out[48]:
```
array([-0.34647994,  1.47873681,  1.48648942,  0.48403995, -0.65743178,
       -1.87247429, -0.24304981,  0.62528388,  0.21060757,  1.31804202])
```

Accessing 5th col from 1st row:

In [49]:
```
randArr[0][4]
```

Out[49]:
```
-0.6574317833888701
```

Accessing odd row:

In [50]:
```
# 1 sai lekar end tak rows chahiyen or 1 ko chor kar 1 chahiyen to step=2,
# or agar cols humein saare chahiyen to hum comma lagake, double colon lagake, starting, ending or step sixe
 skip karden
###########3   randArr[1:10:2,::] #################

randArr[1:10:2]
```

Out[50]:
```
array([[-0.27605532, -1.32816176, -1.12057406, -0.27557991,  1.77297   ,
         0.49613473, -0.68572188, -1.1733908 ,  1.27931125,  0.32895828],
       [ 0.40574129,  0.03789466, -1.25293998,  0.05914265,  1.03218674,
         1.87752573, -0.17883648,  0.30673535,  0.19347169,  0.09450876],
       [ 0.90480491, -1.65448128, -0.75918036,  1.77589396,  0.41453769,
        -0.74658148, -0.0726056 ,  0.25405621,  0.2192515 ,  0.22484674],
       [-0.46245022, -0.98868969,  0.63272828, -1.17187119, -0.095704  ,
        -0.39786988,  0.10235525, -0.31598198,  0.30828205, -0.90354403],
       [-0.43284755,  1.07843767, -1.61222049, -2.08203957,  0.51107326,
        -0.38028299, -0.23751739, -0.64413602, -0.07391223, -0.0529483 ]])
```

Accessing odd cols:

In [51]:
```
# agar mujhe tamaam rows chahiyen or shuru ke char cols chahiyen

randArr[::, 0:4]
```

Out[51]:
```
array([[-0.34647994,  1.47873681,  1.48648942,  0.48403995],
       [-0.27605532, -1.32816176, -1.12057406, -0.27557991],
       [ 0.47621513,  0.50018049,  0.48592307, -1.23050001],
       [ 0.40574129,  0.03789466, -1.25293998,  0.05914265],
       [ 0.36716884, -0.08360576,  0.30381209, -0.8661162 ],
       [ 0.90480491, -1.65448128, -0.75918036,  1.77589396],
       [-0.41390329,  0.41421575,  0.56691819, -0.01171048],
       [-0.46245022, -0.98868969,  0.63272828, -1.17187119],
       [ 0.23171289,  1.59016037, -0.33746377, -1.15052456],
       [-0.43284755,  1.07843767, -1.61222049, -2.08203957]])
```

In [52]:
```
# Creating ones array of order 5 by 5

onesArr = np.ones((5, 5))
onesArr
```

Out[52]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

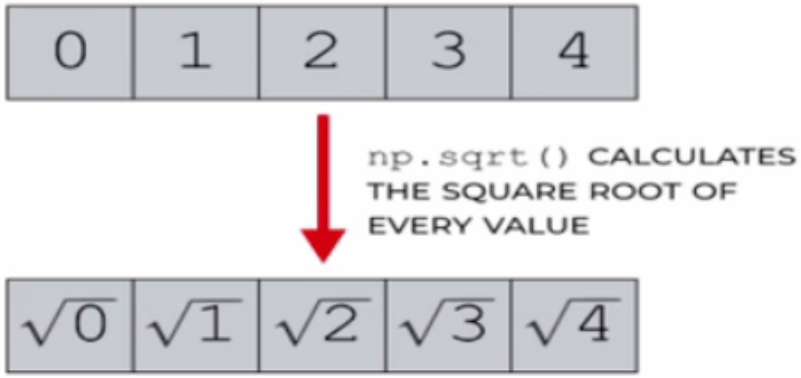
In [54]:
```
# 1st and last row and column ko skip karna hai
# to 1st and last row and 1st col and last col ko chor kar baqi ko hum zero (0) kardenge

onesArr[1:-1,1:-1] = 0
onesArr
```

Out[54]:
```
array([[1., 1., 1., 1., 1.],
       [1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 1.],
       [1., 1., 1., 1., 1.]])
```

Video#6. Fast Element-wise Array functions

Element-wise array functions

- Sqrt
- maximum



Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating-point, or complex values
`sqrt`	Compute the square root of each element (equivalent to `arr ** 0.5`)
`square`	Compute the square of each element (equivalent to `arr ** 2`)
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)
`ceil`	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
`floor`	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as a separate array
`isnan`	Return boolean array indicating whether each value is `NaN` (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-`NaN`) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise (equivalent to `~arr`).

practical

In [66]:
```
arr1 = np.array([5, 10, 15, 20])
arr1
```

Out[66]:
```
array([ 5, 10, 15, 20])
```

1. np.sqrt()

- square root calculate karega kisi bhi number ki

In [59]:
```
sqrtArr = np.sqrt(arr1)
sqrtArr
```

Out[59]:
```
array([2.23606798, 3.16227766, 3.87298335, 4.47213595])
```

2. np.power()

- is function mein mai 1st arg mai array dunga 2nd arg mai bataunga ke kitni power raise karni hai

In [60]:
```
powerArr = np.power(arr1, 2)
powerArr
```

Out[60]:
```
array([ 25, 100, 225, 400], dtype=int32)
```

3. np.maximum(1stArr, 2ndArr)

In [63]:
```
arr2 = np.array([50, 30, 50,-20])
arr2
```

Out[63]:
```
array([ 50,  30,  50, -20])
```

In [64]:
```
print(arr1)
print(arr2)
```

```
[ 5 10 15 20]
[ 50  30  50 -20]
```

In [65]:
```
# ye dono array ke har member ko alag alag compare karega or jo max hoga usse lelega

maxArr = np.maximum(arr1, arr2)
maxArr
```

Out[65]:
```
array([50, 30, 50, 20])
```

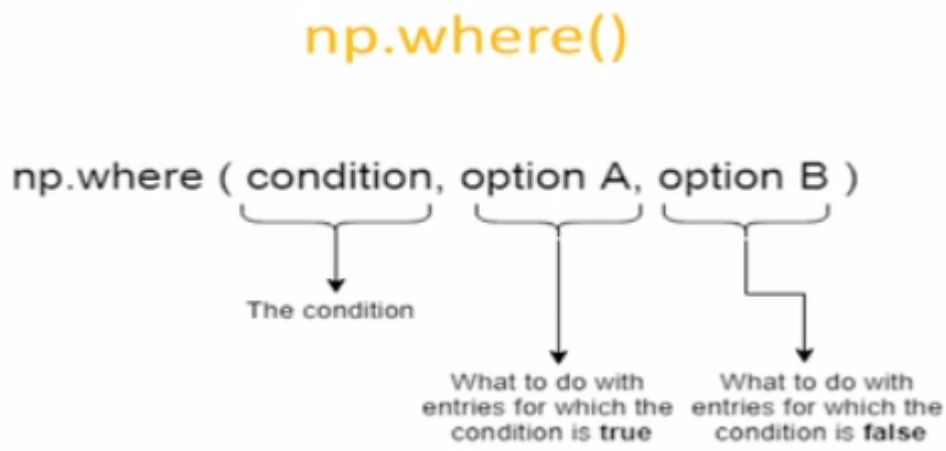
Video#7. np.where()

np.where()

- A vectorized version of the ternary expression x if condition else y

Ternary operator

#



- ternary operator ki tarah kaam np.where() main hota hai lekin differnce ye hai ke ye poore array pe apply hota hai yaani ke ye poori array ke har element ko check karega

- jab hum data science karte hain to data ata hai lekin baaz field missing hoti hain to un missing fields ke andar kuch default value daalna chahte hain to wahan pe ye `np.where` ka function bohot kaam ata hai
- for exapmle apne 1 survey conduct kia or usmai apne employees ki salaries poochin lekin kuch employees ne apni salaries batayin 0 yaani wo apni salaries disclose nhi karna chahte, kuch ne values daaldi -1, baaqiyon ne values sahi daali
- to ap ye karna chah rhe hain ke jahan pe salary 0 ya negatve arhi hai waha pe kuch default daalde e.g 25000 daalden

In [69]:
```
salariesArr = np.array([0, -1, 100000, 50000])
salariesArr
```

Out[69]:
```
array([     0,     -1, 100000,  50000])
```

In [71]:
```
# jahan pe salaries 0 hain wahan pe daalden 25000, otherwise salary ko hum as it is rehne den

analyzedSalariesArr = np.where(salariesArr <= 0, 25000, salariesArr)
analyzedSalariesArr
```

Out[71]:
```
array([ 25000,  25000, 100000,  50000])
```

In [72]:
```
# ya main yahan koi msg deskata hun us value ki jaga

msgSalariesArr = np.where(salariesArr <= 0, "Not OK", "OK")
msgSalariesArr
```

Out[72]:
```
array(['Not OK', 'Not OK', 'OK', 'OK'], dtype='<U6')
```

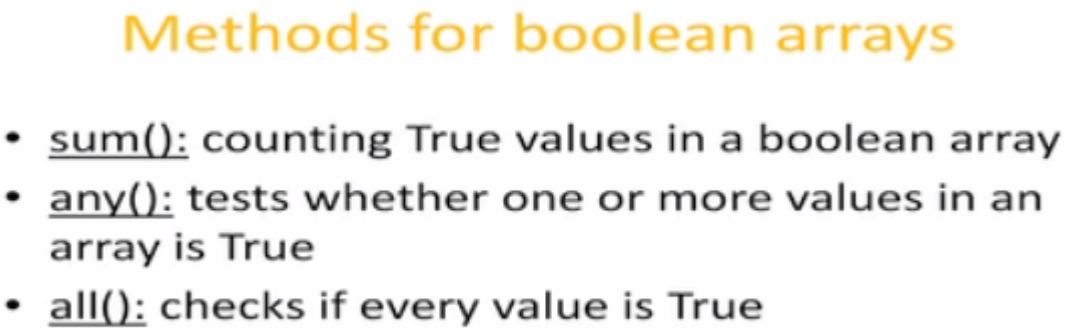
Video#8. Mathematical & Statistical Methods

Mathematical and Statistical Methods

- mean(): returns the mean value computed over the array
- cumsum(): returns the cumulative sum of array elements
- cumprod(): returns the cumulative product of array elements

#

- ese methods joke boolean arrays pe kaam karte hain



`###### mean()`

- e.g apke pass 1 bool array hai, mai ye dekhna chah rha hun ke usmai kitne elements hain joke true hain? to ap `sum()` ka method use karen to apko pata chal jayega ke usmai kitne elements mojood hain qk `True represents 1` jab hum sum karenge to total count ajayega `###### any()`
- `any()` method check karega ke array mai koi element True to nhi hai? koi 1 element True to nhi hai? `###### all()`
- `all()` ka method check karega ke tamaam elements True to nhi hain?

`##########################################################################################################################`

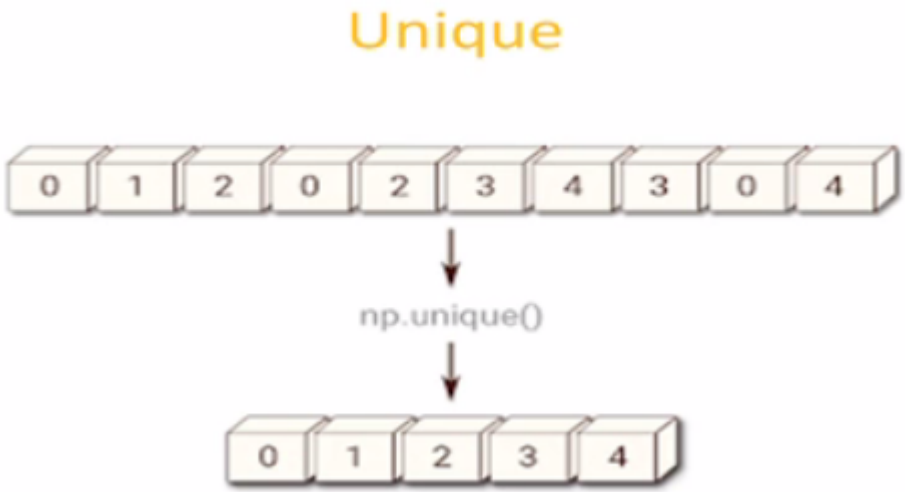
#

- iske elewah apke pass sorting ke methods bhi mojood hain

##

#

- numpy mai ese methods bhi mojood hain joke ye check karsakte hain ke unique elements konse hain array mai?



##

Practical

In [101]:
```
arr1 = np.array([10, 9, 7, 10, 7])
arr1
```

Out[101]:
```
array([10,  9,  7, 10,  7])
```

(1) np.mean(arrName) OR arrName.mean()

In [79]:
```
testArr1 = np.mean(arr1)
testArr2 = arr1.mean()

print(testArr1)
print(testArr2)
```

```
8.6
8.6
```

(2) np.cumsum(arrName) OR arrName.cumum()

In [80]:
```
testArr1 = np.cumsum(arr1)
testArr2 = arr1.cumsum()

print(testArr1)
print(testArr2)
```

```
[10 19 26 36 43]
[10 19 26 36 43]
```

(3) np.cumprod(arrName) OR arrName.cumprod()

In [81]:
```
testArr1 = np.cumprod(arr1)
testArr2 = arr1.cumprod()

print(testArr1)
print(testArr2)
```

```
[   10    90   630  6300 44100]
[   10    90   630  6300 44100]
```

In [83]:
```
# mai ye check akrta hun ke arr1 mai jo bhi element 6 sai bara ho wo ajaye as a bool or True = 1

arr2 = arr1 > 6
arr2
```

Out[83]:
```
array([ True,  True,  True,  True,  True])
```

In [85]:
```
# doing sum of bool array
# it is returning 5 bcoz eac True is 1 so five True makes 5

boolArrSum = arr2.sum()
boolArrSum
```

Out[85]:
```
5
```

(4) np.any(arrName) OR arrName.any()

In [86]:
```
# mai ye check karta hun ke koi element True hai bhi ya nhi using any()

anyTrueArr = arr2.any()
anyTrueArr
```

Out[86]:
```
True
```

(5) np.all(arrName) OR arrName.all()

In [87]:
```
# ye check akrta hun ke tamaam elements True hain bhi ya nhi hain ?

allTrueArr = arr2.all()
allTrueArr
```

Out[87]:
```
True
```

In [90]:
```
print(arr1)
```

```
[10  9  7 10  7]
```

In [91]:
```
# lets make some modification to arr2

arr2 = arr1 > 7
arr2
```

Out[91]:
```
array([ True,  True, False,  True, False])
```

(6) np.sum(arrName) OR arrName.sum()

In [92]:
```
# ab hum agar sum karte hain to kia hoga ?
# now it is returning 3 bcoz there is three True so it sums to 3 each of whic equals 1
# 3 element aise hai joke True hain

boolArrSum1 = arr2.sum()
boolArrSum1
```

Out[92]:
```
3
```

In [93]:
```
# kia koi element aisa hai joke true hai? g bilkul hai

boolArrAny1 = arr2.any()
boolArrAny1
```

Out[93]:
```
True
```

In [94]:
```
# kia tamaam elements True hain ? G nhi

boolArrAll1 = arr2.all()
boolArrAll1
```

Out[94]:
```
False
```

Sorting arr1 :

In [102]:
```
# first check that array is not sorted
print(arr1)
```

```
[10  9  7 10  7]
```

(7) np.sort(arrName) OR arrName.sort()

In [104]:
```
sortArr1 = arr1.sort()
print(sortArr1)
```

```
None
```

In [105]:
```
# arr1 sort hogyi

arr1.sort()
arr1
```

Out[105]:
```
array([ 7,  7,  9, 10, 10])
```

(8) np.unique(arrName)

In [106]:
```
# finding unique elements using unique() ?
# ismai apko np.unique ka method use karna parega qk error arha hai
uniqueArr = arr1.unique()
uniqueArr
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-106-dd91f2e32532> in <module>
      1 # finding unique elements using unique() ?
      2
----> 3 uniqueArr = arr1.unique()
      4 uniqueArr

AttributeError: 'numpy.ndarray' object has no attribute 'unique'
```

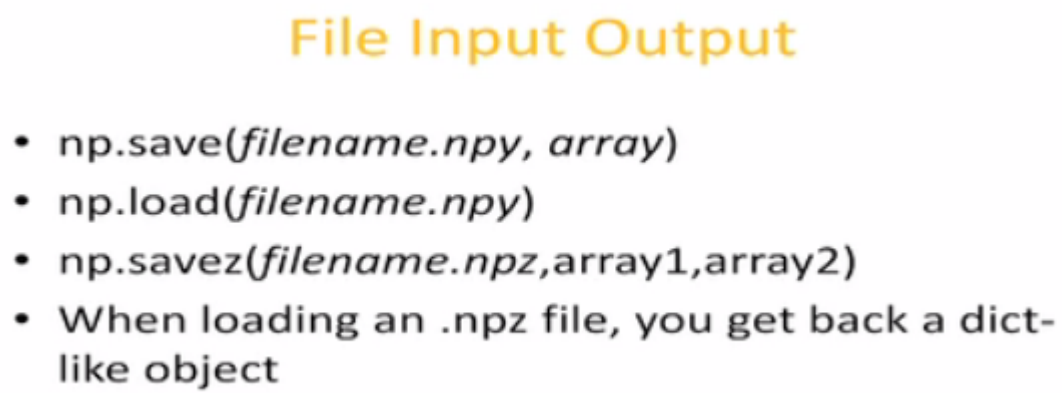
In [107]:
```
# now it works with np.unique()

uniqueArr1 = np.unique(arr1)
uniqueArr1
```

Out[107]:
```
array([ 7,  9, 10])
```

Video#9. File Input Output

#

- Numpy main ap complete array ko bhi save karakte hain naa sirf single array ko balke multiple array ko file mai save karsakte hain or load bhi karsakte hain file sai



- `np.save()` 1st arg filename mangta hai or uski extension bhi mangta hain agar nhi denge extension to wo khud dedega or 2nd arg mai wo array ka naam mangta hai jisko ap save karna chahte hain `lekin ye method sirf single array ko save karega`
- `np.savez()` `agar ap multiple arrays save karana chahte hain to ye method use karenge` 1st arg mai filename phir comma separated arrays names that you wanna save
- jab ap file load karenge to return mai apko 1 dictionary returun hogi jismai ap keys ke through un arrays ki values ko save karsakte hain ##################################

Practical

In [108]:
```
#Create two arrays

arr1 = np.array([1, 4, 5, 7])
arr2 = np.array([8, 7, 9])

#print each of them
print(arr1)
print(arr2)
```

```
[1 4 5 7]
[8 7 9]
```

saving single array `arr1` :

In [109]:
```
# mai arr1 ko save karunga in a file named "testArr1" ab qk mai yahan extension nhi derha to ye automatically
"npy" dedega

np.save("testArr1", arr1)
```

loading file named `testArr1` :

In [111]:
```
# laod karte hue mujhe file ki extension dena zaroori hai warna error ayega

loadArr1 = np.load("testArr1")
loadArr1
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-111-a74f3b73f430> in <module>
----> 1 loadArr1 = np.load("testArr1")
      2 loadArr1

C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\npyio.py in load(file, mmap_mode, allow_pickle, fix_impo
rts, encoding)
    426         own_fid = False
    427     else:
--> 428         fid = open(os_fspath(file), "rb")
    429         own_fid = True
    430

FileNotFoundError: [Errno 2] No such file or directory: 'testArr1'
```

In [112]:
```
# now it works as we prvide extension

loadArr1 = np.load("testArr1.npy")
loadArr1
```

Out[112]:
```
array([1, 4, 5, 7])
```

Saving multiple arrays `arr1` and `arr2`:

- ab mai yahan save() ka nhi balke `savez()` ka `method` `use` `karunga` qk mai yahan multiple arrays save kara rha hun or `extension` `bhi` `yahan` `npy` `nhi` `balke` `npz` `hogi`

In [124]:
```
np.savez("testArr1AndArr2", arr1=arr1, arr2=arr2)
```

In [125]:
```
# now it is returning dict
# is sai hum values retrive karsakte hain
loadArr1AndArr2 = np.load("testArr1AndArr2.npz")
loadArr1AndArr2
```

Out[125]:
```
<numpy.lib.npyio.NpzFile at 0xa03eeb0>
```

Retrieving value from dict:

In [127]:
```
# Jab uppar humne ye save ki thin to arr1 or arr2 diya tha key name nhi diya tha like arr1=arr1 and arr2=arr2
to ye cell..
# .. error derha tha now it works

loadArr1AndArr2["arr1"]
```

Out[127]:
```
array([1, 4, 5, 7])
```

Video#10. Linear Algebra functions

#

Function	Description
diag	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
dot	Matrix multiplication
trace	Compute the sum of the diagonal elements
det	Compute the matrix determinant
eig	Compute the eigenvalues and eigenvectors of a square matrix
inv	Compute the inverse of a square matrix
pinv	Compute the Moore-Penrose pseudo-inverse of a matrix
qr	Compute the QR decomposition
svd	Compute the singular value decomposition (SVD)
solve	Solve the linear system Ax = b for x, where A is a square matrix
lstsq	Compute the least-squares solution to Ax = b

- `arrName1.dot(arrName2)` product of two matrix calculate karega
- `arrName.inv()` inverse of marix
- `arrName.det()` determinant nikaalsakte hain ##

In [3]:
```
arr1 = np.array([
    [24, 45],
    [12, 56],
    [44, 78]
])
arr1
```

Out[3]:
```
array([[24, 45],
       [12, 56],
       [44, 78]])
```

In [4]:
```
# arr1 is 3 by 2 of order

np.shape(arr1)
```

Out[4]:
```
(3, 2)
```

In [5]:
```
arr2 = np.array([
    [34, 56, 90],
    [22, 12, 34],
])
arr2
```

Out[5]:
```
array([[34, 56, 90],
       [22, 12, 34]])
```

In [6]:
```
# arr2 is of 2 by 3 order

np.shape(arr2)
```

Out[6]:
```
(2, 3)
```

(2) arrName1.dot(arrName2):

In [7]:
```
dotArr = arr1.dot(arr2)
dotArr
```

Out[7]:
```
array([[1806, 1884, 3690],
       [1640, 1344, 2984],
       [3212, 3400, 6612]])
```

In [9]:
```
print(arr1)
print(arr2)
```

```
[[24 45]
 [12 56]
 [44 78]]
[[34 56 90]
 [22 12 34]]
```

(3) arrName.transpose()

In [11]:
```
# (3 by 2) ka matrix (2 by 3) mai convert hokar agya

transposeArr = arr1.transpose()
transposeArr
```

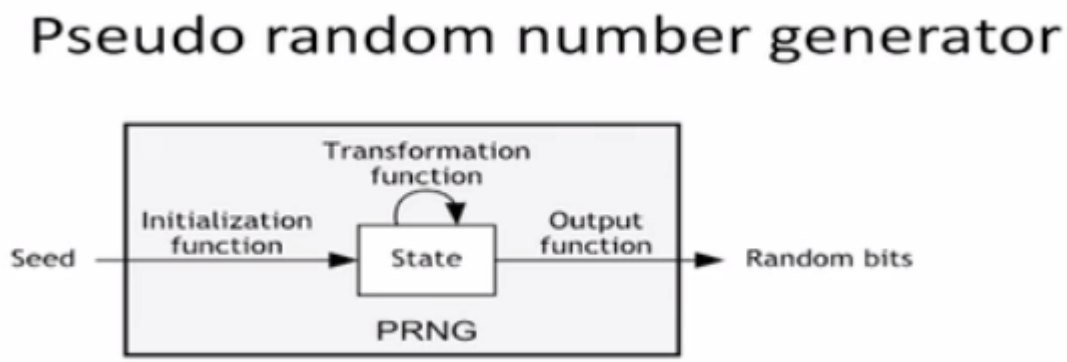
Out[11]:
```
array([[24, 12, 44],
       [45, 56, 78]])
```

In [15]:
```
traceArr = arr1.trace()
traceArr
```

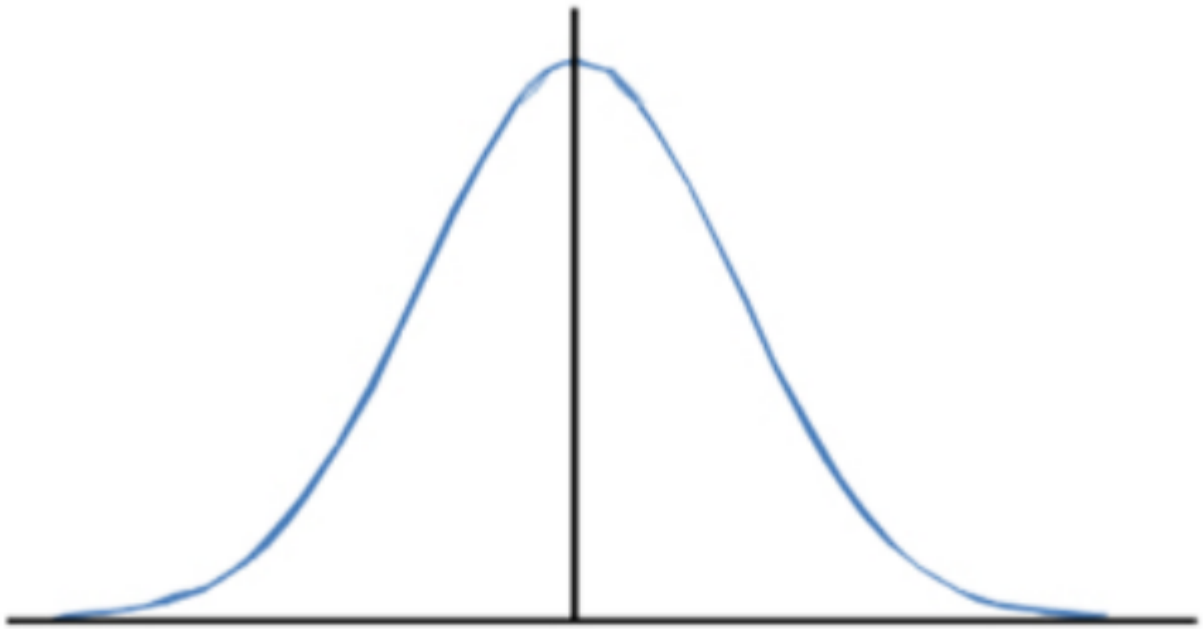
Out[15]:
```
80
```

Video#11. Pseudo Random Number Generator

- Random number ki zarurat q pesh ati hai? jab hum dataset load karte hain to datasets mai biasness ko khatam karne keliye aksar humne apne data ko randomize karna parhta hai, data ko shuffle karna parhta hai, taake kisi data ki specific sequence ki wajah sai biasness ko khatam karsaken



- truly random number generate karna poosible nhi hai computer keliye qk computer kisi formula ke taht random number generate karta hai isiliye hum kehte hain ke ye pseodo-random number generations modules hain



Normal distribution curve.

- computer jab random number generate karta hai to wo kisi distribution ke taht karta hai, normal distribution bhi hosakti hai, gamma distribution bhi hosakti hai, uniform distribution bhi hosakti hai

Pseudo-random number generation

- normal()
- seed()
- gamma()
- uniform()

- to ye tamaam distribution ke through random numbers generate karne ki facility mojood hai lekin 1 important baat ye hai ke jab hum random numbers use kar rhe hote hain to humne random numbers generator ko initialize karna parta hai, issiliye hum usko starting mai seed value dete hain, seed value ke through wo random numbers ko initilize karta hai

Practical

In [5]:
```
# seed value ap kuch bhi desakte hain
np.random.seed(7)
```

In [6]:
```
# ab mai normal distribution ke through 3 by 3 ka 1 matrix generate karwata hun randomly
# to ye eik random number generate hua normal distribution ke thorugh
np.random.normal(size=(3,3))
```

Out[6]:
```
array([[ 1.69052570e+00, -4.65937371e-01,  3.28201637e-02],
       [ 4.07516283e-01, -7.88923029e-01,  2.06557291e-03],
       [-8.90385858e-04, -1.75472431e+00,  1.01765801e+00]])
```

In [7]:
```
# unifom distribution ke through bhi random number generate karsakte hain

np.random.uniform(size=(3,4))
```

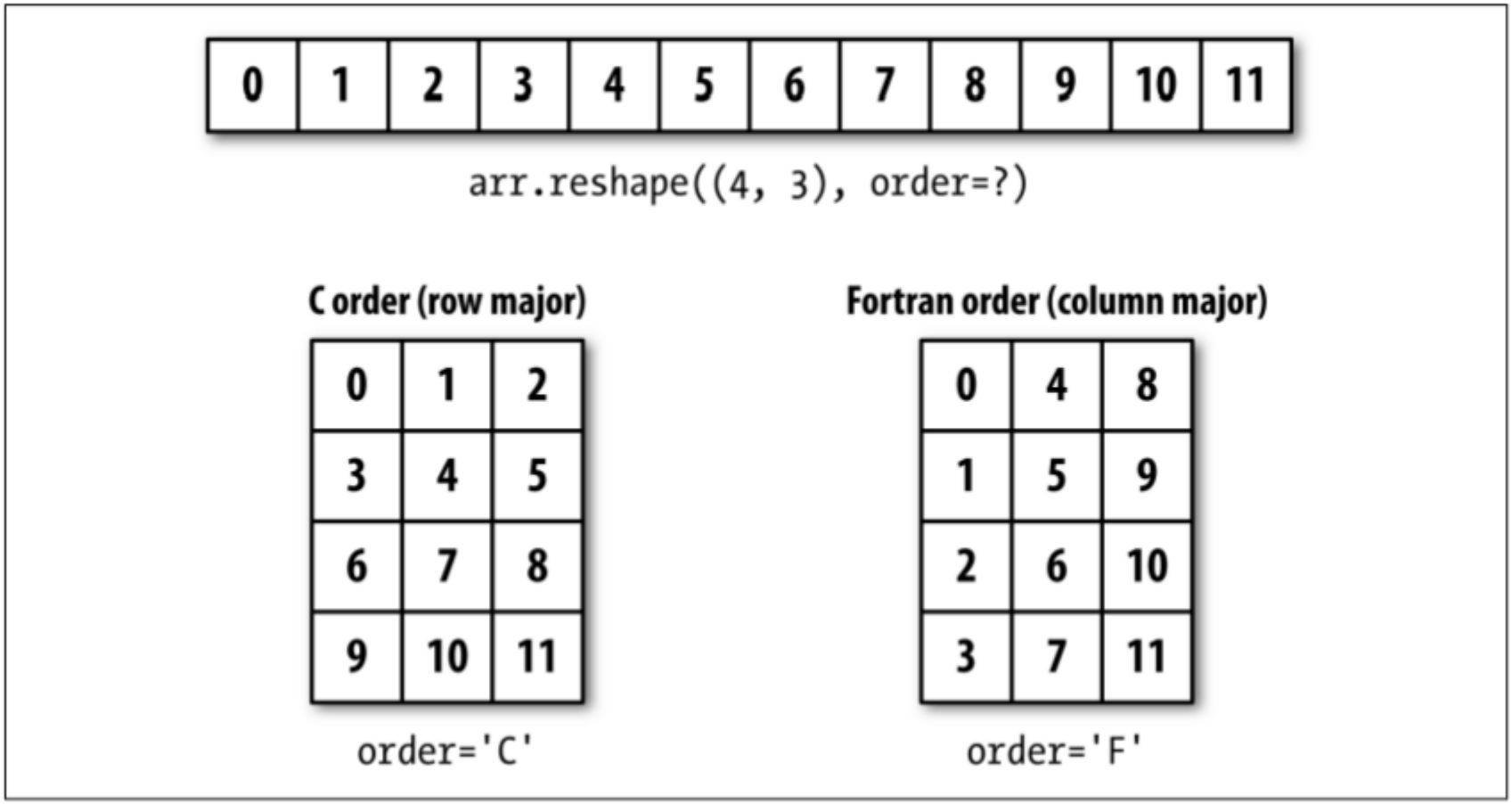
Out[7]:
```
array([[0.38094113, 0.06593635, 0.2881456 , 0.90959353],
       [0.21338535, 0.45212396, 0.93120602, 0.02489923],
       [0.60054892, 0.9501295 , 0.23030288, 0.54848992]])
```

12. Numpy Advanced Array Manipulations

13. Reshape

- row major order or column major order ye bunyaadi farq hai ke array kis tareeqe sai internally stored hai

Reshape



Reshaping in C (row major) or Fortran (column major) order

- reshape karte hue agar main order kehta hun C, to C mai, row major order main data store hota hai, e.g: agar mere pass 0 sai leakr 11 tak elements hain, agar ye C order main hain ya row major order main hain to ye banega 0,1,2, 1st row lekin agar ye F order main he to ye 0,1,2 column banega

reshape()

- Reshaping arrays
  - C order: Row major order
  - Fortran order: Column major order
  - Specifying -1 for reshape() means the value used for that dimension will be inferred from the data
  - ravel() or flatten() i.e. flattening to one dimension

Practical

In [19]:
```
arr1 = np.floor(np.abs(np.random.randn(4,5)))
arr1
```

Out[19]:
```
array([[0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [1., 0., 1., 0., 0.],
       [1., 1., 1., 0., 1.]])
```

In [20]:
```
# by default isne row major order mai save kia hai, ismai pehli dimension 2 hai second dimension bhi 2 hai or
last yaani 3rd..
# .. dimension 5 hai
arr1.reshape((2,2,5))
```

Out[20]:
```
array([[[0., 0., 0., 0., 1.],
        [1., 0., 0., 0., 0.]],

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

In [24]:
```
# reshape karte hue mai column major order bhi batasakta hun
orderC = arr1.reshape((2,2,5), order='C')
orderC
```

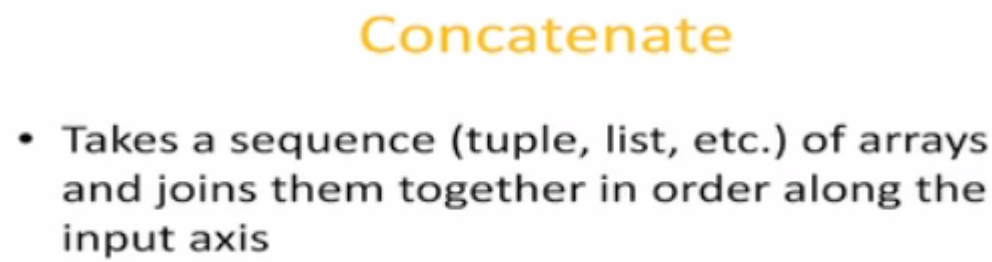
Out[24]:
```
array([[[0., 0., 0., 0., 1.],
        [1., 0., 0., 0., 0.]],

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

In [25]:
```
orderF = arr1.reshape((2,2,5), order='F')
orderF
```

Out[25]:
```
array([[[0., 0., 0., 0., 1.],
        [1., 0., 1., 0., 0.]],

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

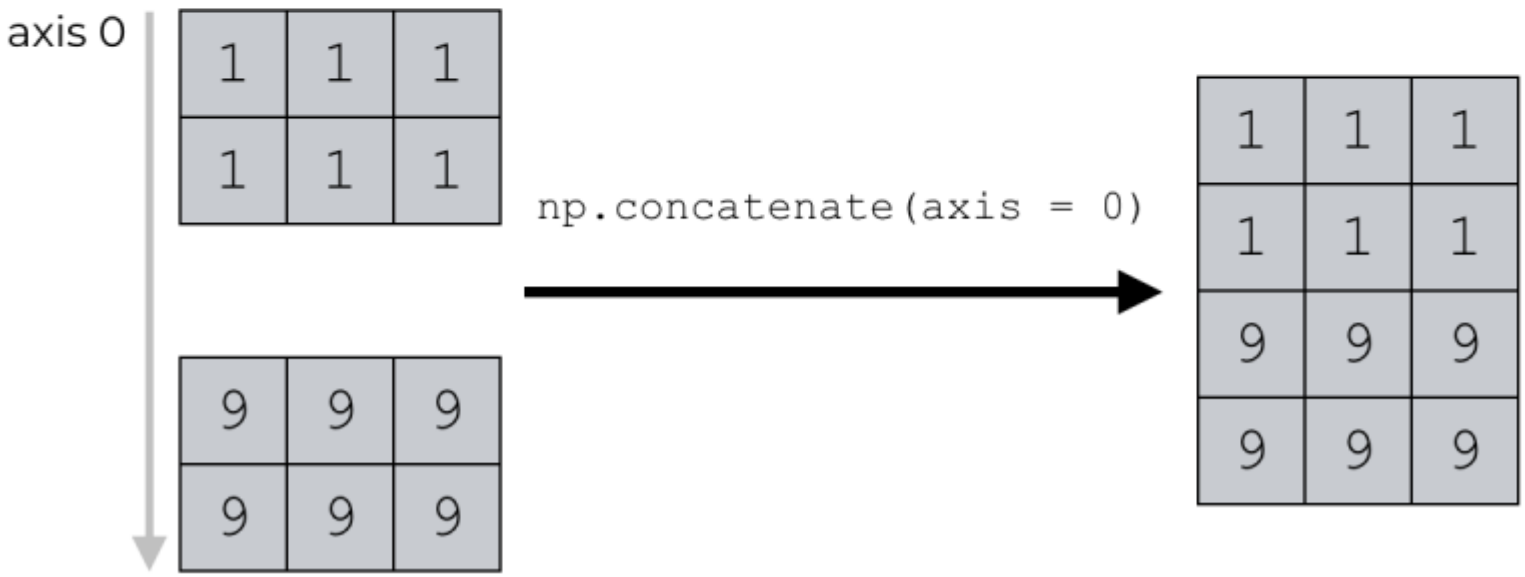
Video#14. Concatenate

- concatenate as an argument a set of arrays leta hai or unko concatenate karta hai either row wise ya col wise, iskeliye wo eik argument leta hai axis ka jo argument ye batata hai ke apne row wise concatenate karna hai ya col wise karni hai

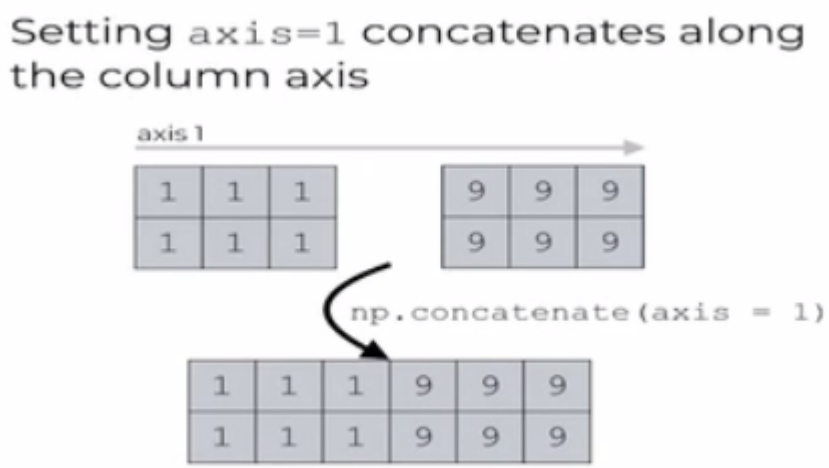


concatenate

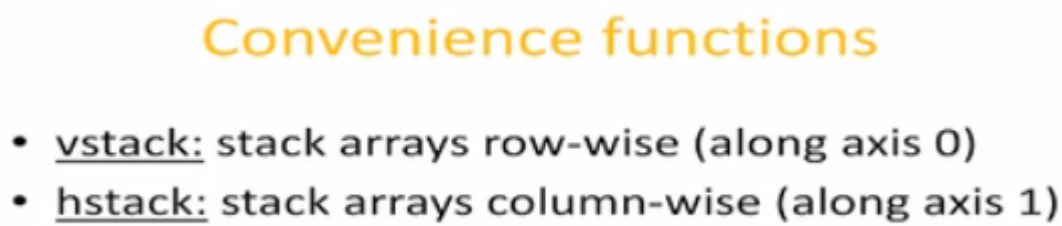
Setting `axis=0` concatenates along the row axis



- diagram ke through samjhte hain apke pass LHS par 2 arrays hain 1st array 2 by 3 ki hai jismain tamaam elements 1 hain or 2nd array 2 by 3 ki hai jismai tamaam elements 9 hain, agar hum in dono arrays ko concatenate karen, or arg hum pass karen axis ka 0, to iska matlab hai ke row wise concatenation karni hain, ismai hum dekhsakte hain ke cols ki tadaad 3 hee rhi rows ki tadaad 4 hogyi hai, jo cols they woh wohi rhe or jo rows thin wo stack down hote hue 2 sai rows 4 hogyi



- is doosri example main hum arg pass karte hain axis=1 or wohi same do arrays hain to ye col wise concatenate hogya



- concatenation mai ap axis ka arg pass karte hain 0 ya 1 to iske kuch conveninent functiins bhi hain `vstack` or `hstack`
- `vstack` matlab ap concatenation perform kar rhe hain or `axis` apne `0` diya hai, yaani row wise concatenate kar rhe hain
- `hstack` mai ap concatenation kar rhe hain along cols ya axis=1 kardia apne is function ke through

practical

In [29]:
```
# create list 1 & 2
list1 = [[1,3,5], [7,9,0]]

list2 = [[11,13,15], [17,19,10]]
```

In [30]:
```
# convert list 1 & 2 to np array
arr1 = np.array(list1)

arr2 = np.array(list2)
```

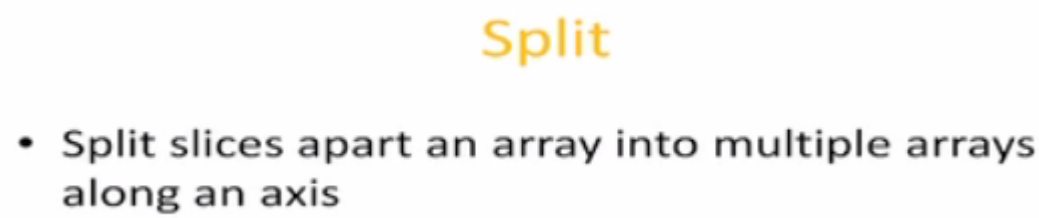
In [34]:
```
# concatenate two arrays

# as a tuple both arrays will be passed
# concatenated row wise, and col wise respectively

row_wise = np.concatenate((arr1,arr2), axis=0)
col_wise = np.concatenate((arr1,arr2), axis=1)

print(row_wise)
print("=======================")
print(col_wise)
```

```
[[ 1  3  5]
 [ 7  9  0]
 [11 13 15]
 [17 19 10]]
=======================
[[ 1  3  5 11 13 15]
 [ 7  9  0 17 19 10]]
```

In [35]:
```
# concatenate two arrays using functions
# dont need to specify axis

row_wise1 = np.vstack((arr1,arr2))  # vstack >>> row wise
col_wise1 = np.hstack((arr1,arr2))  # hstack >>> col wise

print(row_wise1)
print("===================")
print(col_wise1)
```

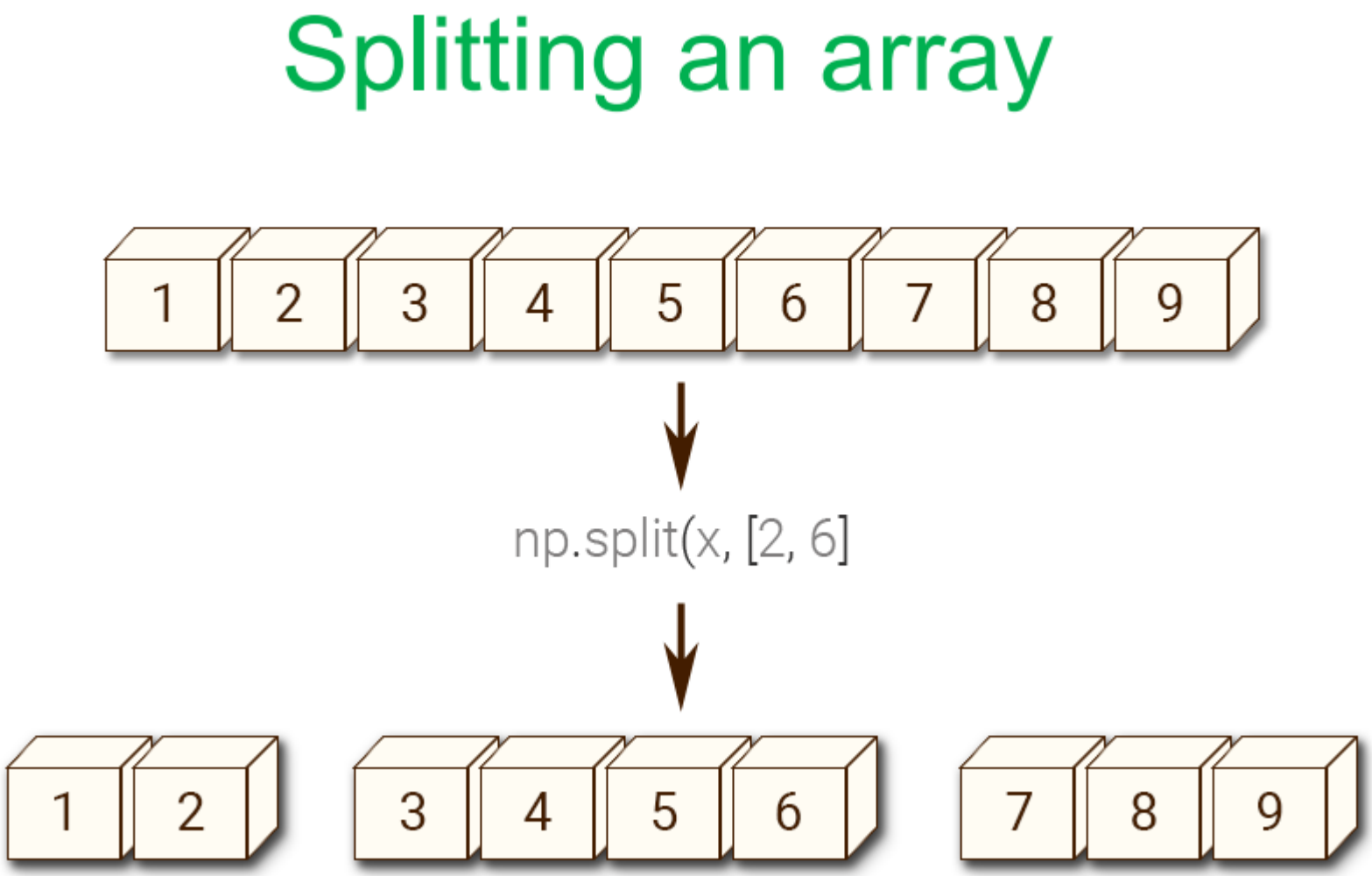
```
[[ 1  3  5]
 [ 7  9  0]
 [11 13 15]
 [17 19 10]]
===================
[[ 1  3  5 11 13 15]
 [ 7  9  0 17 19 10]]
```

Video#15 Split



- jese concatenate 2 arrays ko jorta hai split 1 array ko mukhtalif hisson mai torta hai

- isko hum is diagram ke through samjhte hain



- mere pass 1d array hai jismai 1 sai lekar 9 tak elements hain, mai ne isko split karte hue specify kia ke position 2 pe or position 6 pe, x array ko split karna hai, to ap dekh sakte hain ke hamare pass 3 arrays return hue
- pehla portion 0 sai start hote hue 2 pe end horha hai, doosra 2 sai start hote hue 6 pe end horha hai or teesra 6 sai start hote hue end tak jarha hai, to ye method specified positions pe apki array ko split kar rha hai

Practical

In [36]:
```
# create list

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

In [37]:
```
# convert list1 to array

arr1 = np.array(list1)
arr1
```

Out[37]:
```
array([1, 4, 5, 6, 7, 9, 1, 4])
```

In [39]:
```
# np.split() mai 1st arg mai us array ka naam dena hai jisko maine split karna hai, 2nd arg mai bataunga kin
 positions pe split
# ..karna hai, to ye 3 parts mai array break hogyi

# [1, 4, 5, 6, 7, 9, 1, 4]  >>>>>> array elements
#  0, 1, 2, 3, 4, 5, 6, 7   >>>>>> indexes od elements
#        |        |
# mai ne kaha ke 2 or 5 par split kardo to ye un specific index par jakar split kardega
np.split(arr1, [2,5])
```

Out[39]:
```
[array([1, 4]), array([5, 6, 7]), array([9, 1, 4])]
```

2d Array

In [42]:
```
# agar mere pass 2d array hoti
list1 = [[1, 4, 5, 6, 7, 9, 1, 4], [11, 14, 15, 16, 17, 19, 11, 14]]
```

In [43]:
```
# convert list to 2d array
arr2d = np.array(list1)
arr2d
```

Out[43]:
```
array([[ 1,  4,  5,  6,  7,  9,  1,  4],
       [11, 14, 15, 16, 17, 19, 11, 14]])
```

In [45]:
```
# splitting arr2d col wise(axis=1)
# [
#    [ 1,  4,  5,  6,  7,  9,  1,  4],
#    [11, 14, 15, 16, 17, 19, 11, 14]
# ]    0,  1,  2, 3,  4,  5,   6,  7
#                     |        |

np.split(arr2d, [4,6], axis=1)
```

Out[45]:
```
[array([[ 1,  4,  5,  6],
        [11, 14, 15, 16]]),
 array([[ 7,  9],
        [17, 19]]),
 array([[ 1,  4],
        [11, 14]])]
```

In [47]:
```
# splitting arr2d row wise(axis=0)
# [
#    [ 1,  4,  5,  6,  7,  9,  1,  4],
#    [11, 14, 15, 16, 17, 19, 11, 14]
# ]    0,  1,  2, 3,  4,  5,   6,  7
#          |
# rows 2 parts mai break hojayegi if split along 0 axis

np.split(arr2d, [1], axis=0)
```

Out[47]:
```
[array([[1, 4, 5, 6, 7, 9, 1, 4]]), array([[11, 14, 15, 16, 17, 19, 11, 14]])]
```

Array concatenation functions

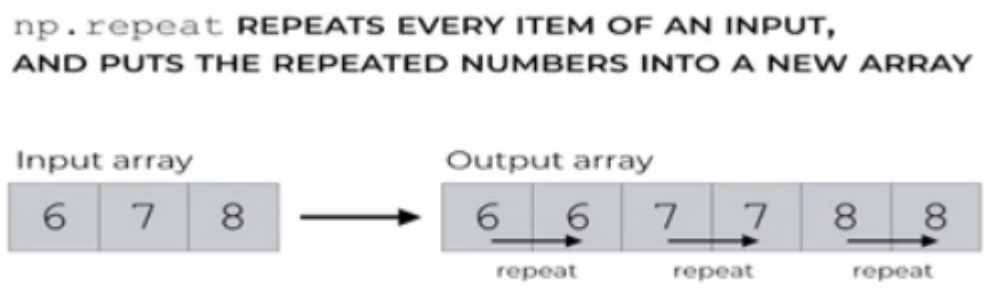
Function	Description
`concatenate`	Most general function, concatenates collection of arrays along one axis
`vstack, row_stack`	Stack arrays row-wise (along axis 0)
`hstack`	Stack arrays column-wise (along axis 1)
`column_stack`	Like `hstack`, but converts 1D arrays to 2D column vectors first
`dstack`	Stack arrays "depth"-wise (along axis 2)
`split`	Split array at passed locations along a particular axis
`hsplit/vsplit`	Convenience functions for splitting on axis 0 and 1, respectively

Video#16 Tile & Repeat

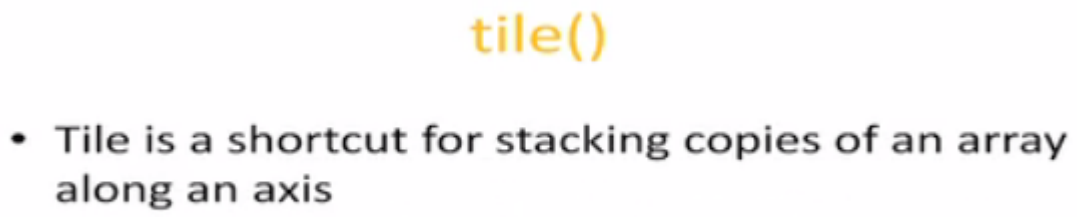
repeat()

- repeat replicates each element in an array some number of times, producing a larger array
- If you pass an array of integers, each element can be repeated a different number of times

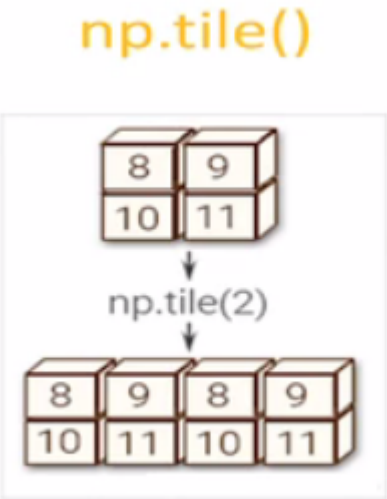
- repeat ka function as an arg eik array leta hai, or us array ke har elemnt ko repeat karta hai specified number of times, e.g: mere pass eik array hai jismai elements hain 6,7,8 or mai isko repeat karta hun 2 times, main arg deta hun ke mujhe isse 2 dafa repeat karna hai, to array ka har element 2 times repeat hoga



- jabke tile ka function poore array ko tile karega ya repeat karega



- e.g:



- mere pass 2 by 2 ki 1 array hai agar mai isko col wise tile kart hunto mujhe ye is tarah dikhega

Practical

In [48]:
```
# create an array

arr1 = np.array([1,5,7,8])
arr1
```

Out[48]:
```
array([1, 5, 7, 8])
```

In [50]:
```
# I call np.repeat() call karta hun or 1st arg mai main array ka naam pass karta hun jisko mai ne repeat karn
a hai , or 2nd ..
# .. arg main mai ye batata hun ke kitni baar repeat karna hai
# to har element 3 times repeat hoga

repeatArr1 = np.repeat(arr1,3)
repeatArr1
```

Out[50]:
```
array([1, 1, 1, 5, 5, 5, 7, 7, 7, 8, 8, 8])
```

In [51]:
```
# to np.tile() poori array ko repeat karta hai

tileArr1 = np.tile(arr1,2)
tileArr1
```

Out[51]:
```
array([1, 5, 7, 8, 1, 5, 7, 8])
```

Conclusion

- Covered ndarray: creating, indexing, slicing, I/O, linear algebra, random number generation
- Advanced concepts such as concatenation, split, repeat, tile
- Other advanced concepts such as broadcasting describes how arithmetic works between arrays of different shapes
- Sorting

- `brodcasting` : e.g agr hum 2 arrays par koi operation perform kar rhe hain or 1 array small zize ka hai, or 1 array large size ka hai to small size ka array apne ap ko khud hee expand karlega, is concept ko hum broadcating kehte hian

In [ ]:

```

```