For Download

1.Anaconda distribution search —> then download —> install Mac Anaconda —> launch Jupyter —>

(Terminal anaconda-navigator)

2. For vs code pip3 install numpy

3. Google Collabs

How to run / Install Jupyter —>
Jupyter Launch করার পর web browser এ new click করে Jupyter terminal open করতে হবে

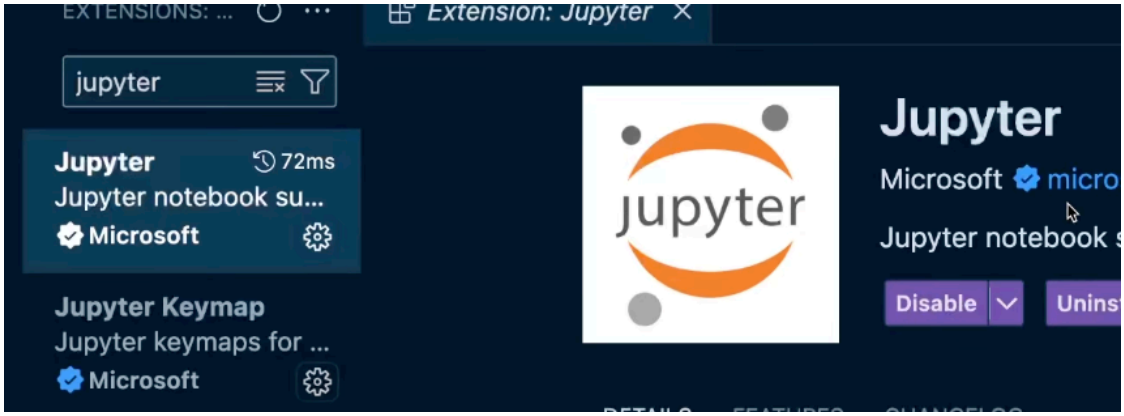
 তারপর cd" folder path copy " / folder Name (enter)
Then Jupyter notebook (enter) then file open

New (python condo click)

বা folder terminal open then write Jupyter Notebook লিখলেঈ হবে ।

Shift + enter (cell Execute হবে)

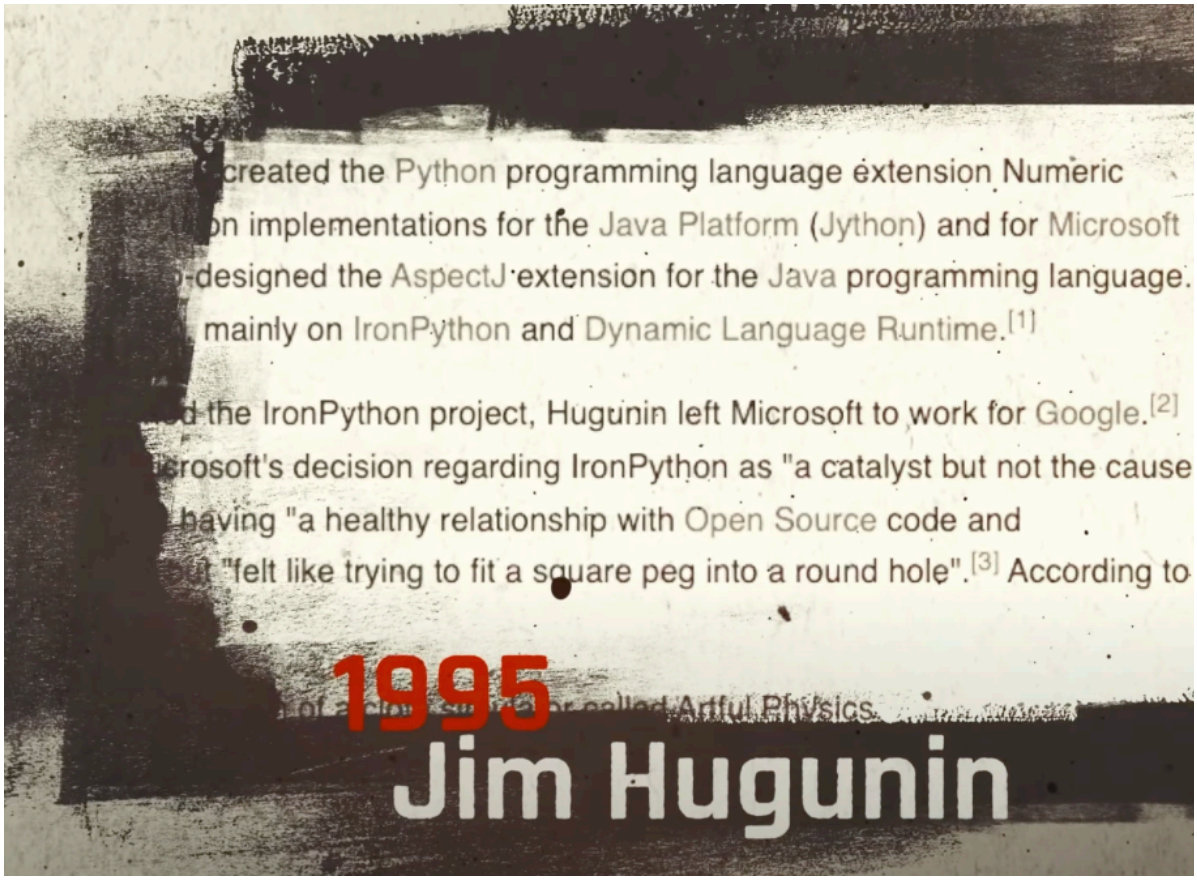
Second way VsCode —>
Extension Download Jupyter



Then click Select kernel —> then select python Environment

**********NOW START NUMPY **********

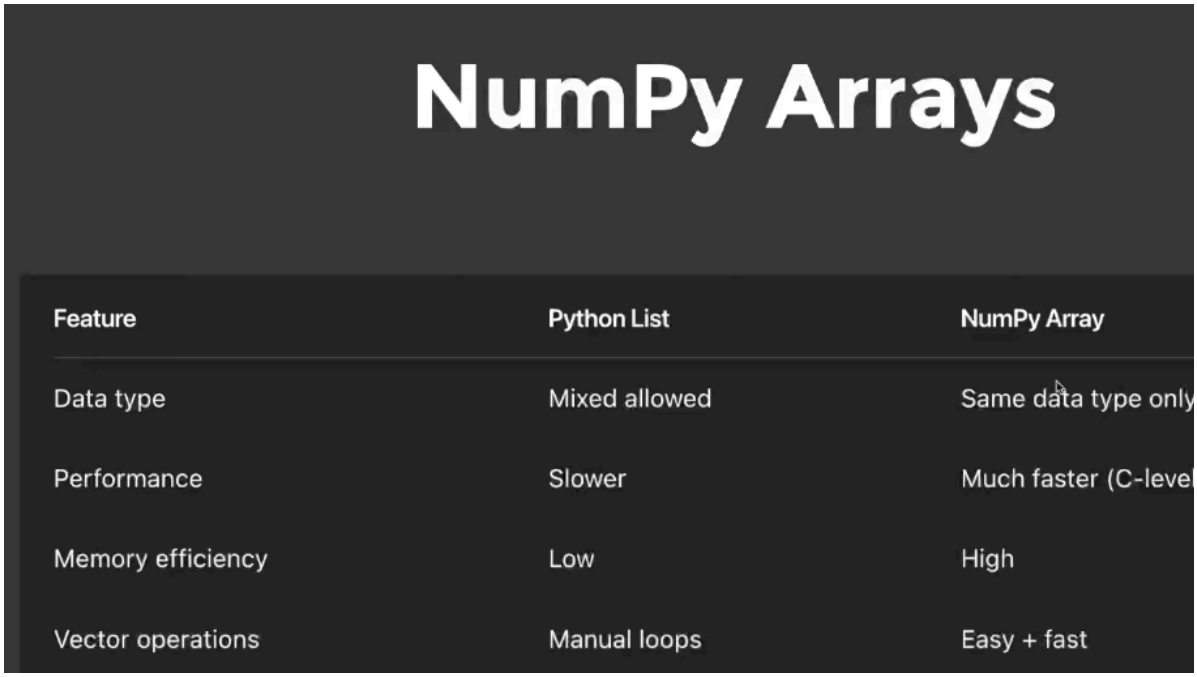
Python always math এর জন্য ভালো কিন্তু পায়থন এর array support ছিল না এর জন্য ১৯৯৫ এ jims hugunin created numeric array . But its not good



Then 2000 Numarray create করা হয় space telescopes science institute

Now still it's' not compatible but after that Travis oliphant Numeric and Numarray library combine করে Numpy create করে । Its' created by c programming and then its connected with python now its 50 times faster than list .

********** DIFFERENCE BETWEEN LIST AND ARRAY **********



List এর মধ্যে আমরা একইসাথে অনেক ধরনের data type element রাখতে পারি কিন্তু Numpy array তে আমরা শুধু এক্টাইএপের element রাখতে পারি ।
Slower because python interpreted language আর numpy created by c

********** Basic Array Creation **********

Import numpy as np
Np.array([]) একটা লিস্ট provides করে দিব ।

```
import numpy as np
```

Basic Array creation

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

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

```
a = [1,2,3.5,"hello"]
a
```

```
[1, 2, 3.5, 'hello']
```

আমরা যদি একটা list এ string রেখে সেইটা array এর মধ্যে রাখি তাহলে array সেগুলোকে একটা

Data type এ convert করে দিবে । যেমন এখানে সবগুলো কে string এ convert করে দিয়েছে ।

```
a = [1,2,3.5,"hello"]
a
```

```
[1, 2, 3.5, 'hello']
```

```
np.array(a)
```

```
array(['1', '2', '3.5', 'hello'], dtype='<U32')
```

যেমন এখানে float value তে convert করে দিয়েছে ।

```
a = [1,2,3.5]
a
```

```
[1, 2, 3.5]
```

```
np.array(a)
```

```
array([1. , 2. , 3.5])
```

**********Vector And Matrix **********

Vector — One dimetional List
Matrix — Multidimensional (list multidimensional support ঈ করে না)
Two dimensional এর উপরে যেগুলো সেগুলোকে বলা হয় tensar (3d , 4d, 5d)

Here two dimensional list it's looking like a normal list but when we convert a list into array it's looking like a vector .

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

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

```
np.array(l)
```

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

**********ARRAY GENERATION FUNCTION **********

If we generate array using arrange function
np.arrange(1,10) — এখানে ১ থেকে ১০ পর্যন্ত array create হয়ে যাবে ।

```
np.arange(1,11)

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

Np.arrange(1,10,2) step - 2 ঘর পর পর আসবে

```
arr = np.arange(1,11,2)
arr

array([1, 3, 5, 7, 9])
```

zeros(6) function .

```
arr = np.zeros(6)
arr

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

It's generated 4 rows & 8 columns .

```
arr = np.zeros((4,8))
arr

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

ONES(6) — function

```
arr = np.ones(6)
arr

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

```
arr = np.ones((6,6))
arr
```

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

If we generate default value not zeros or ones . Then we use full function np.full((shape) , value) ;

```
import numpy as np
filled_Array = np.full((2,2),7)
print(filled_Array)
```

```
oduction/creation/4.py"
[[7 7]
 [7 7]]
```

IDENTITY MATRIX - it's a square matrix and ones in the diagonal .

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

Output will be like that —>

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

আমরা যদি একটা function এ কাজ না জেনে থাকি তাহলে shift + tab দিবো তাহলে তা দেখতে পারবো কি করে এই ফাংশন ।

np.linspace(1,5,3) 1 থেকে ৫ পর্যন্ত প্রিন্ট করবে ২ টা element .

```
arr = np.linspace(1,5,2)
arr
```

```
array([1., 5.])
```

1 থেকে ৫ পর্যন্ত ৩ টা element প্রিন্ট করবে কিন্তু তাদের মধ্যে সমান space থাকবে।

```
arr = np.linspace(1,5,3)
arr
```

```
array([1., 3., 5.])
```

```
arr = np.linspace(1,5,10)
arr
```

```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

********** Random Generation function **********

np.random.rand(5)

```
np.random.rand(5)
```

```
array([0.5178548 , 0.09144116, 0.55729517, 0.78944859, 0.8888938 ])
```

10 random elements.

```
np.random.rand(10)
```

```
array([0.58291532, 0.17054358, 0.13068718, 0.32660627, 0.33849008,
       0.27173774, 0.7335558 , 0.0878656 , 0.24567008, 0.95931161])
```

Its provide vector

Random vector 1 থেকে ০ এর মধ্যে আসবে। Its a concept of normalisation

Random vector -3 থেকে ৩ এর মধ্যে আসবে। Its concept of standardisation

(np.random.randn(10) — its give 10 random element -3 theke 3

```
np.random.randn(10)
```

```
array([ 0.75630459, -1.89031619, -0.83065648,  1.34618169,  1.0496594
        1.32698836,  0.09977396, -1.2585357 ,  2.33771857, -0.4244752
```

আর যদি আমি যেকোনো ভ্যালু আনতে চাই তাহলে কিভাবে করবো তারজন্যে

np.random.randint(10) এর মানে ০ এর থেকে ১০ এর মধ্যে যেকোনো একটা নাম্বার প্রিন্ট করব।

```
np.random.randint(6)
```

```
0
```

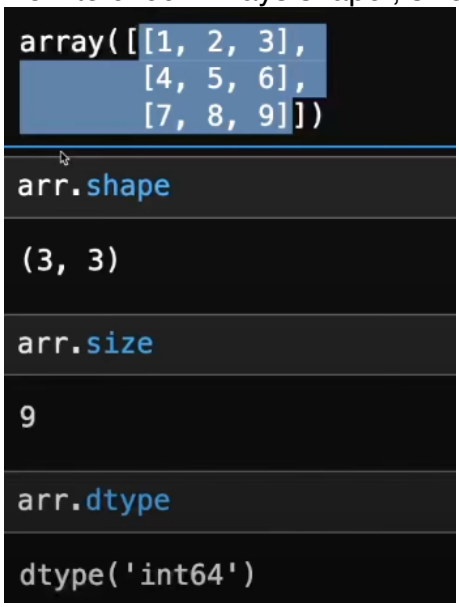
```
np.random.randint(10,20,10)
```

```
array([17, 13, 19, 16, 12, 13, 11, 18, 17, 13])
```

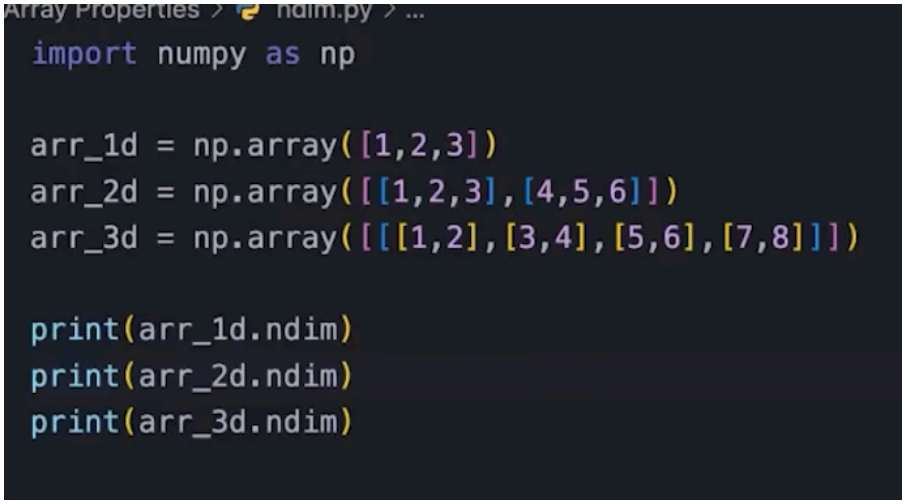
এখানে ১০ থেকে ২০ এর মধ্যে ১০ টা random number generate করে দিবে।

Attribute কে cll করে না এর আগে () লাগানো হয় না । আর মেথড কে call করা হয় এর আগে () লাগানো হয় ।

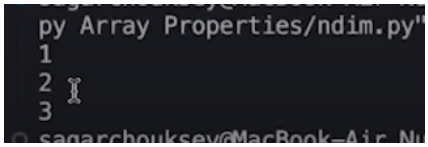
How to check Arrays shape , size , type ?



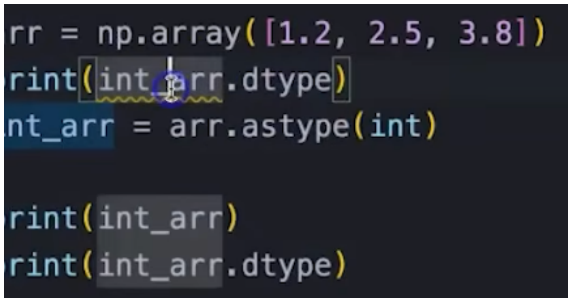
Ndim — number of dimension .
Array এর ডাইমেনশন জানা যায়।

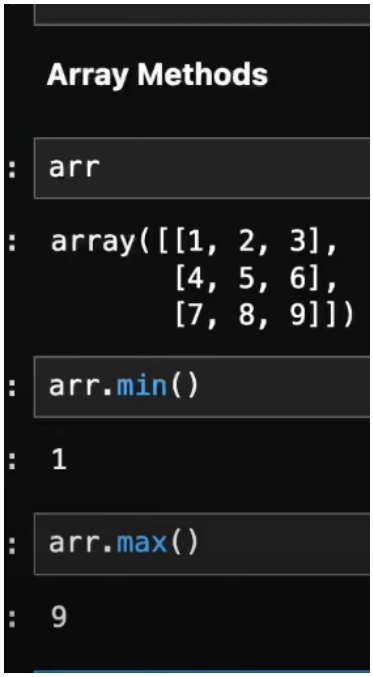


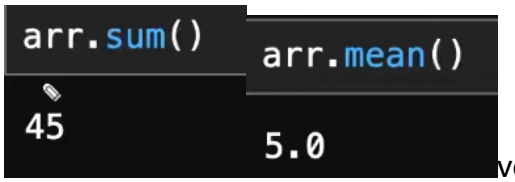
আউটপুট



Datatype কে change করার জন্য.







vector এর middle element value .

```
arr.argmax()
```

```
8
```

```
arr.argmin()
```

```
0
```

arr.argmax() —> maximum element এর ইনডেক্স , arr.argmin() —> minimum element এর index দিবে ।

it’s inbuild but np একটা দেয় আলাদা ভাবে ।

Column wise 0 axis and row wise 1-axis .

np.sum(arr, axis = 0) its provide column wise elements total sum
np.sum(arr, axis = 1) it’s provide row wise elements total sum .

```
np.sum(arr,axis = 0)
```

```
array([12, 15, 18])
```

```
np.sum(arr,axis = 1)
```

```
array([ 6, 15, 24])
```

**********Reshaping & Resizing **********

Reshaping means I have already an array 1-d Vector . Now I want to convert its 2d vector .
Reshape করতে হলে আমি যে array টা কে করবো সেটার মধ্যে সমান element থাকতে হবে আমি যে shpae এ কনভার্ট করবো ।
Here it is 25 element array now I want to 5*5 2d matrix .

```
arr = np.arange(1,31)
arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
```

Arr এর মধ্যে save করতেছি এই টা হলো resizing আর reshape করেছি reshape function লিখে ।

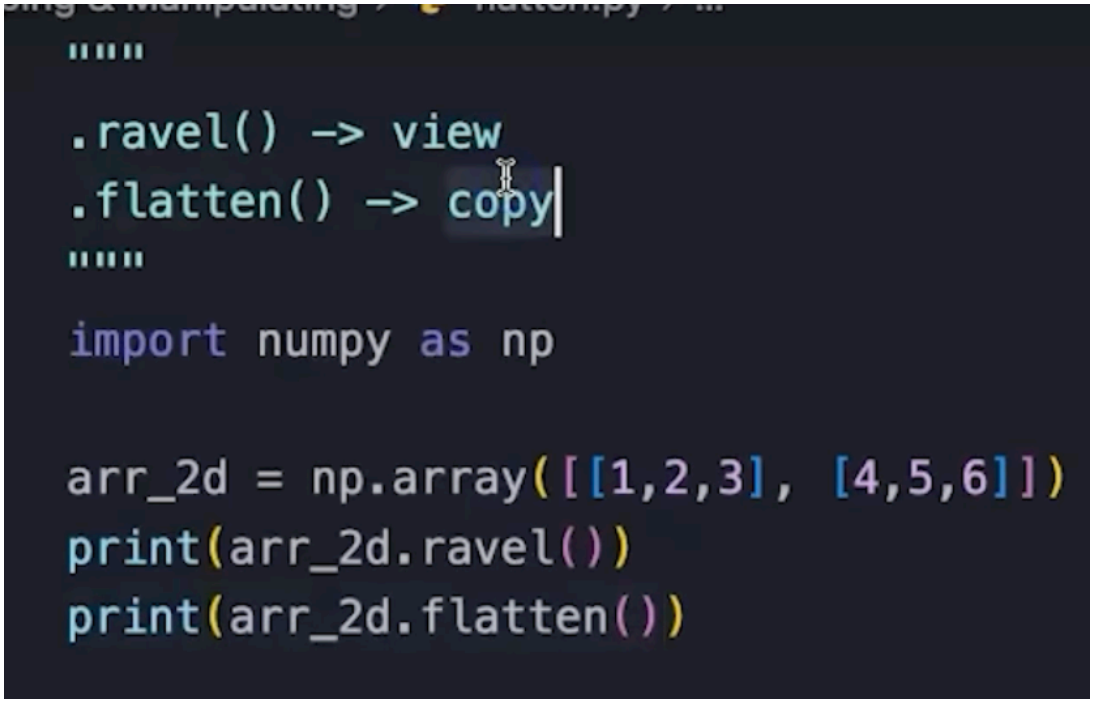
```
arr = arr.reshape(6,5)
```

```
arr
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25],
       [26, 27, 28, 29, 30]])
```

**********Ravel & flatten **********

আমরা যদি কোন 2-d array কে 1-d array তে কনভার্ট করি তাহলে flattering use করব।
এইটা দুই ধরনের হয়- ravel , flatten
Ravel — view (modification করলে main array তে modification হয়ে যায়।
Flatten — copy (modification করলে main array তে modification হয় নাই।



**********ARRAY INDEXING AND SLICING **********

Array Indexing :

```
import numpy as np
```

NUMPY INDEXING AND SLICING OF VECTORS

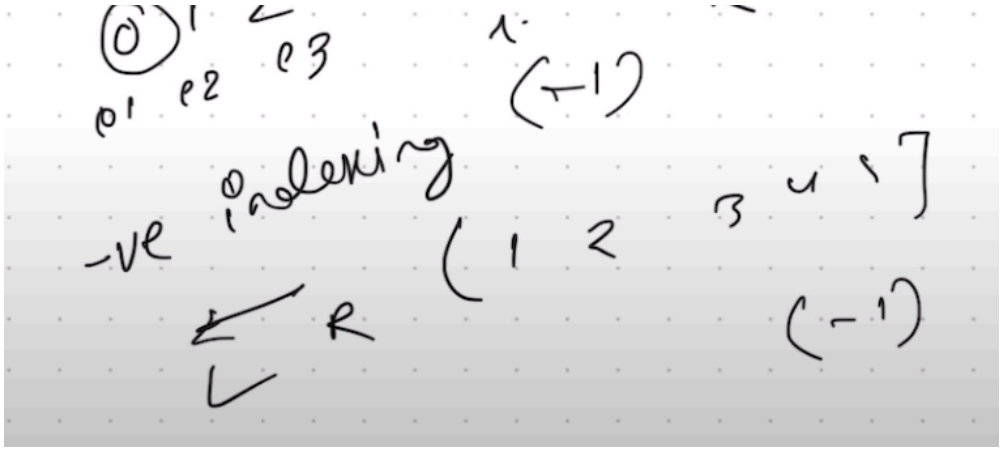
```
arr = np.arange(11,21)
arr
```

```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

```
arr[9]
```

```
20
```

Array indexing এর মধ্যে দুইটা জিনিস থাকে একটা positive indexing & negative Indexing .
Positive Indexing -> left to right value access করতে চাইলে।
Negative Indexing —> Right To Left value Access করতে চাইলে। Last element দেখতে চাইলে -1 দিতে হবে।



```
arr = np.array([10,20,30,40,50])

print(arr[0]) #first element
print(arr[2]) #2nd 0 based
print(arr[-1]) #last element
```

**********Array Slicing**********

Slice করতে চাইলে আমরা প্রথম index দিবো তারপর শেষ ইনডেক্স এ last Wanted slice Index +1 দিবো
arr[1:5] — 12 13 14 provide like that :
Array [start : stop : step] — start index : stop Index +1 : step (কিছু না দিলে default value 1 হবে।)
Negative step -1 দিলে reverse হবে।

```
arr = np.arange(11,21)
arr
```

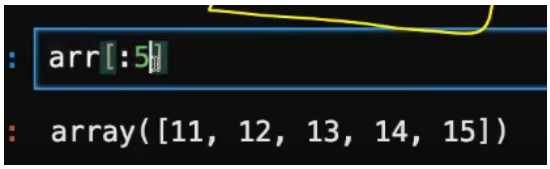
0 1 2 3 4 5

```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

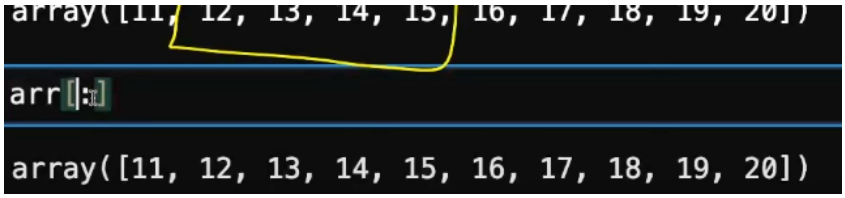
```
arr[1:5]
```

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

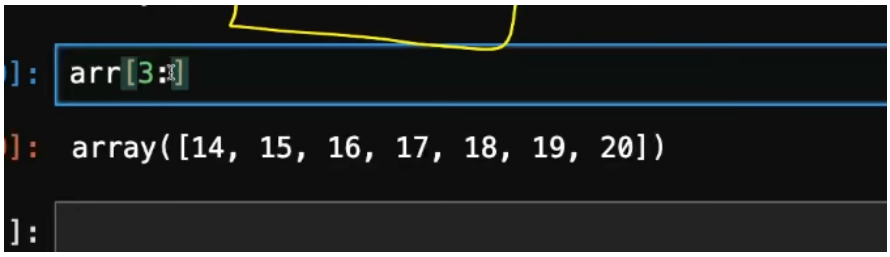
যদি starting index না দেই default প্রথম ইনডেক্স ০ ধরে নিবে ।



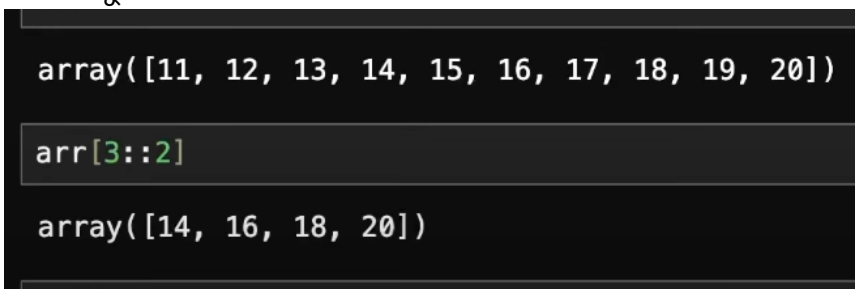
যদি লাস্ট এর ইনডেক্স ও না দেই তাহলে সব ভ্যালুই এসে পরবে ।



আর যদি last index না দেই তাহলে লাস্ট ইনডেক্স পর্যন্ত চলে যাবে ।



যদি ভ্যালু গুলো কয়েক ঘর পর পর পেতে চাই তাহলে ::want এভাবে দিবো ।



এই যে এখানে ২ ঘর পর পর আসবে ।

**********FANCY INDEXING**********

Fancy indexing হলো একসাথে কত গুলো ইনডেক্স নিতে চাইলে তখন এই টা ব্যবহার করব।

```
arr = np.array([10,20,30,40,50,60])

print(arr[[0, 2, 4]])
#10 30 50
```

আর এখানে shallow copy হয়। আমার মেইন array তে পরিবর্তন হবে না যদি নতুন array তে কোনো modification করি।

**********Boolean Masking **********

```
arr = np.array([10,20,30,40,50,60])

print(arr[arr > 25])
```

এখানে আমরা কোনো লুপ চালিয়ে কন্ডিশন না দিয়ে সরাসরি একটা array এর থেকে ২৫ থেকে বড় যে ভ্যালু গুলো সেগুলো পেতে পারি।

NUMPY INDEXING AND SLICING OF MATRIX

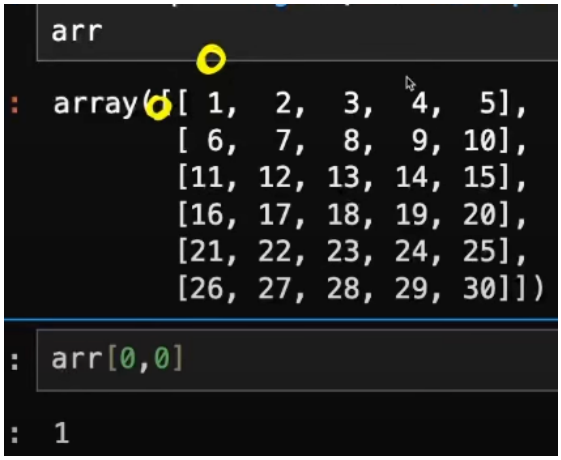
```
arr = np.arange(1,31).reshape(6,5)
arr
```

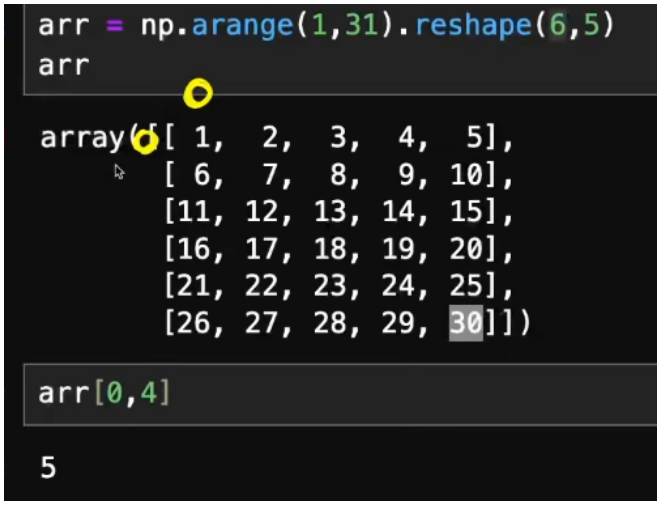
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25],
       [26, 27, 28, 29, 30]])
```

```
arr[5]
```

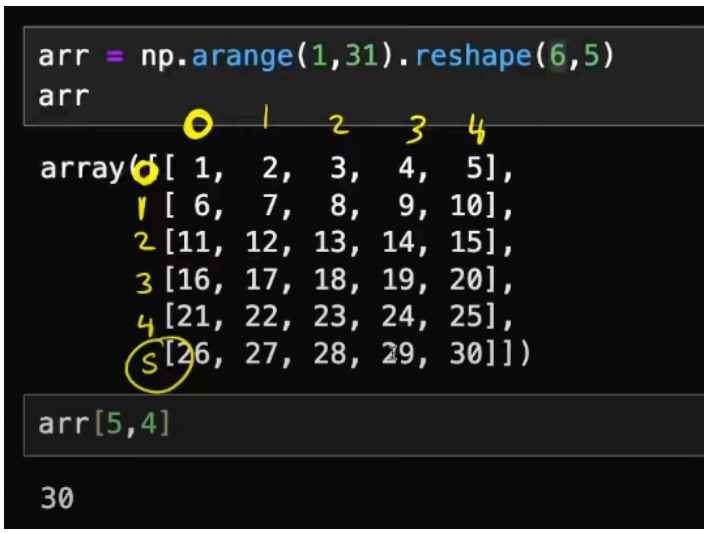
```
array([26, 27, 28, 29, 30])
```

ইন্ডেক্সিং that will target rows .





First targeting first row index then targeting col index if we find the value .



**********Slicing into the Matrix**********

```
[81]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10],
             [11, 12, 13, 14, 15],
             [16, 17, 18, 19, 20],
             [21, 22, 23, 24, 25],
             [26, 27, 28, 29, 30]])

[105]: slice = arr[0:2,1:3]
       slice

[105]: array([[2, 3],
              [7, 8]])
```

arr[3 : 6 , 3 : 5] it's wrong if we want to slicing last index it right code will be like that arr[3 : , 3 :] ;

```
]: array([[ 1,  2,  3,  4,  5],
          [ 6,  7,  8,  9, 10],
          [11, 12, 13, 14, 15],
          [16, 17, 18, 19, 20],
          [21, 22, 23, 24, 25],
          [26, 27, 28, 29, 30]])

]: slice = arr[3:,3:]
   slice

]: array([[19, 20],
          [24, 25],
          [29, 30]])
```

If we want to selecting full columns like we slice full row like arr[0] its give me full 0 row value like that if we want col value then first of all select full row first (0) to last (n)
We know if we not initialise value first value before (:) and not initialise value last value that will give us full array element . Hence we want full row value that why first of all we select row and then we select col value like that .
arr[:,3]

```
[81]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10],
             [11, 12, 13, 14, 15],
             [16, 17, 18, 19, 20],
             [21, 22, 23, 24, 25],
             [26, 27, 28, 29, 30]])

107]: slice = arr[3:,3:]
      slice

107]: array([[19, 20],
             [24, 25],
             [29, 30]])

109]: arr[:,2]

109]: array([ 3,  8, 13, 18, 23, 28])
```

If we want to save only even or odd elements of the array . আমরা আগে করতাম for loop চালাতাম তারপর check করতাম কোণটা odd আর কোনটা even .

```
[133]: array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])

[135]: bool_index = arr %2 == 0
       bool_index

[135]: array([False,  True, False,  True, False,  True, False,  Tr
              True])

[137]: arr = arr[bool_index]
       arr

[137]: array([12, 14, 16, 18, 20])
```

যখন আমরা array operation চালাই তখন আমরা ২ বা ২ ততোধিক array এর উপর চালাই ।

If we want to sum two array elements .
দুইটা array এর সাইজ সমান হতে হবে ।
Addition & Substation & multiplication & division :

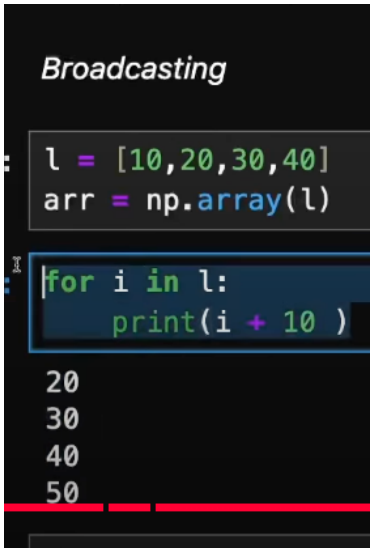
```
[141]: a1 = np.array([1,2,3,4,5])
       a2 = np.array([6,7,8,9,10])

[149]: a1 + a2

[149]: array([ 7,  9, 11, 13, 15])

[151]: a1 -a2

[151]: array([-5, -5, -5, -5, -5])

[153]: a1 * a2

[153]: array([ 6, 14, 24, 36, 50])

[155]: a1 / a2

[155]: array([0.16666667, 0.28571429, 0.375     , 0.44444444, 0.5
```

Power :

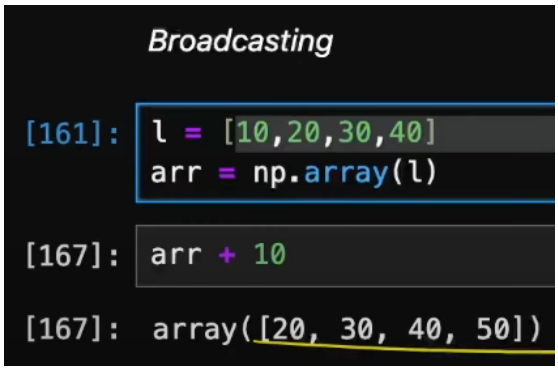
```
[159]: a1 ** a2

[159]: array([      1,     128,    6561,  262144, 9765625])
```

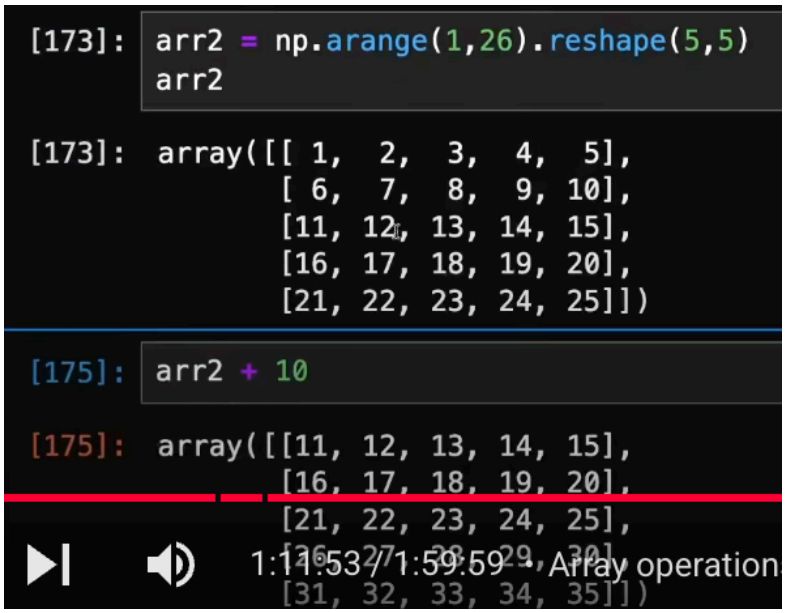
List এ প্রত্যেকটা element এর সাথে ১০ যোগ করতে চাইলে আমরা for loop চালাতে হতো ।

**********BROADCASTING**********



কিন্তু array তে ১০ সবগুলো array element এর সাথে যোগ করতে চাচ্ছি তাহলে array variable + 10 করলেই হবে ।





```
[185]: arr2 *2

[185]: array([[42, 44, 46, 48, 50],
              [52, 54, 56, 58, 60],
              [62, 64, 66, 68, 70],
              [72, 74, 76, 78, 80],
              [82, 84, 86, 88, 90]])
```

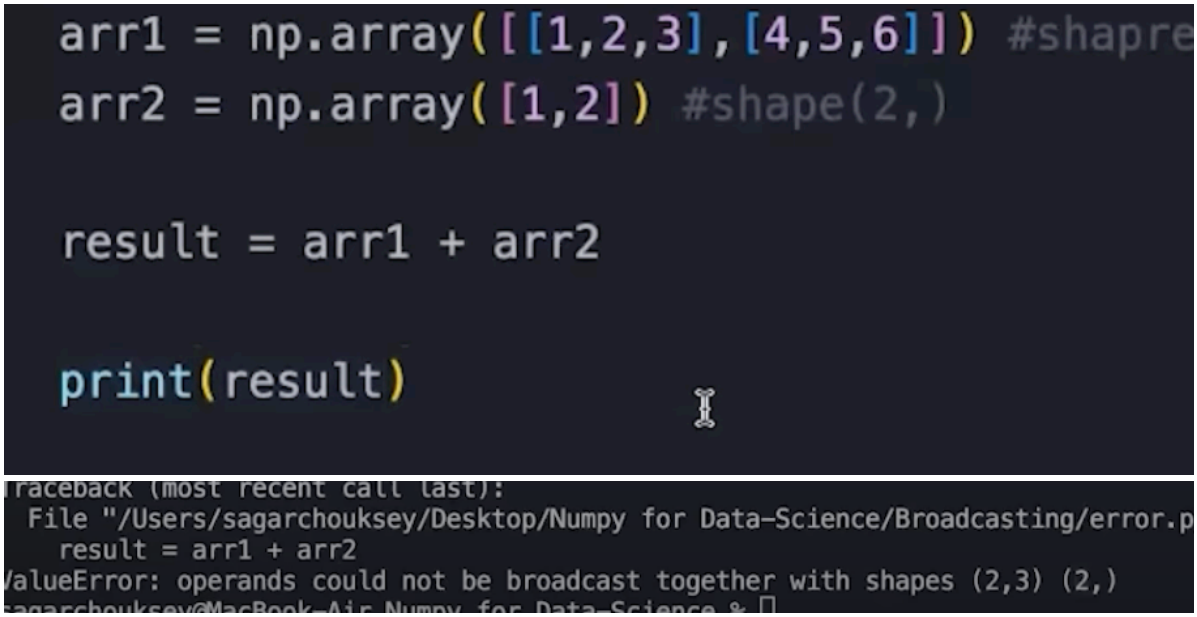
```
prices = np.array([100,200,300])
discount = 10 #
                (variable) prices: NDArray

final_prices = prices - (prices * discount/

print(final_prices)
```

```
[ 90. 180. 270.]
```

How NUMPY handle array of Different shape .

Broadcasting 3 rule :

1. Matching Dimension ([1,2,5] + [3,4,5] = [4 , 6 , 10]
2. Expanding Single Element ([1,2,3] + 10 = [11 ,12 , 13]
3. Incompatible Shape(shape match না করলে numpy error দিয়ে দিবে[১,৩,৫]+[১,৩] error)

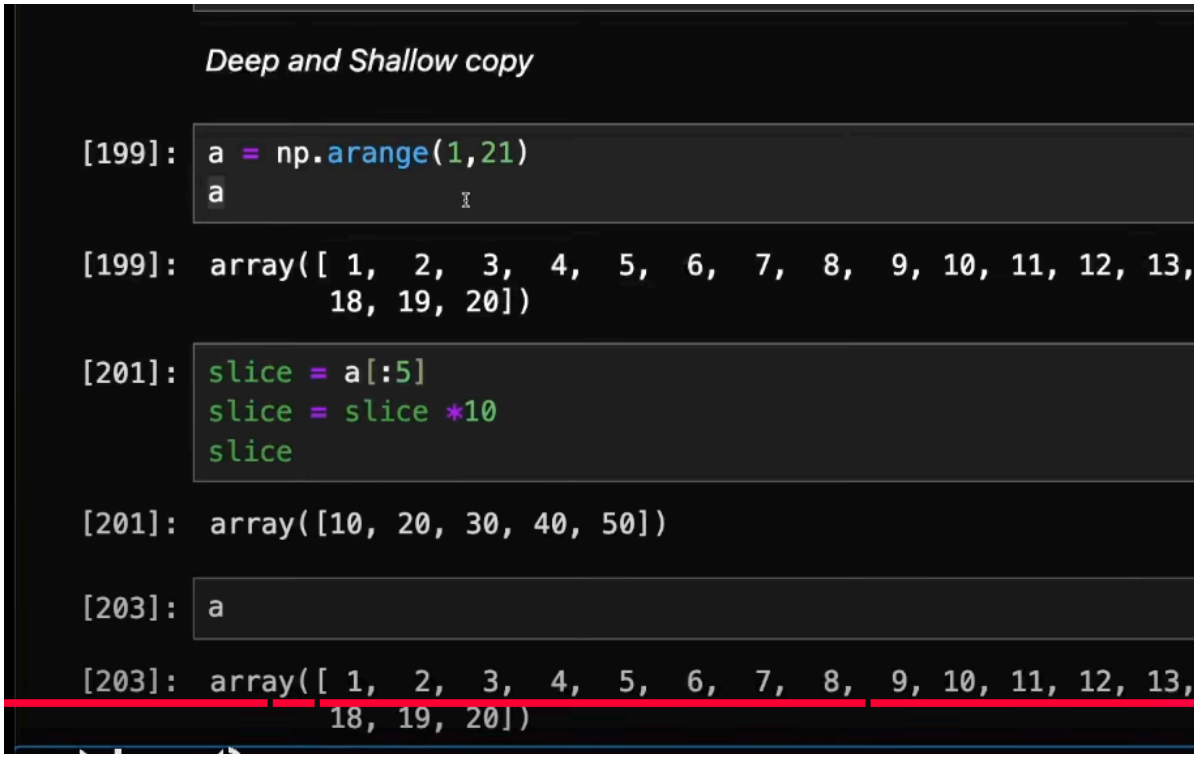


Give error .

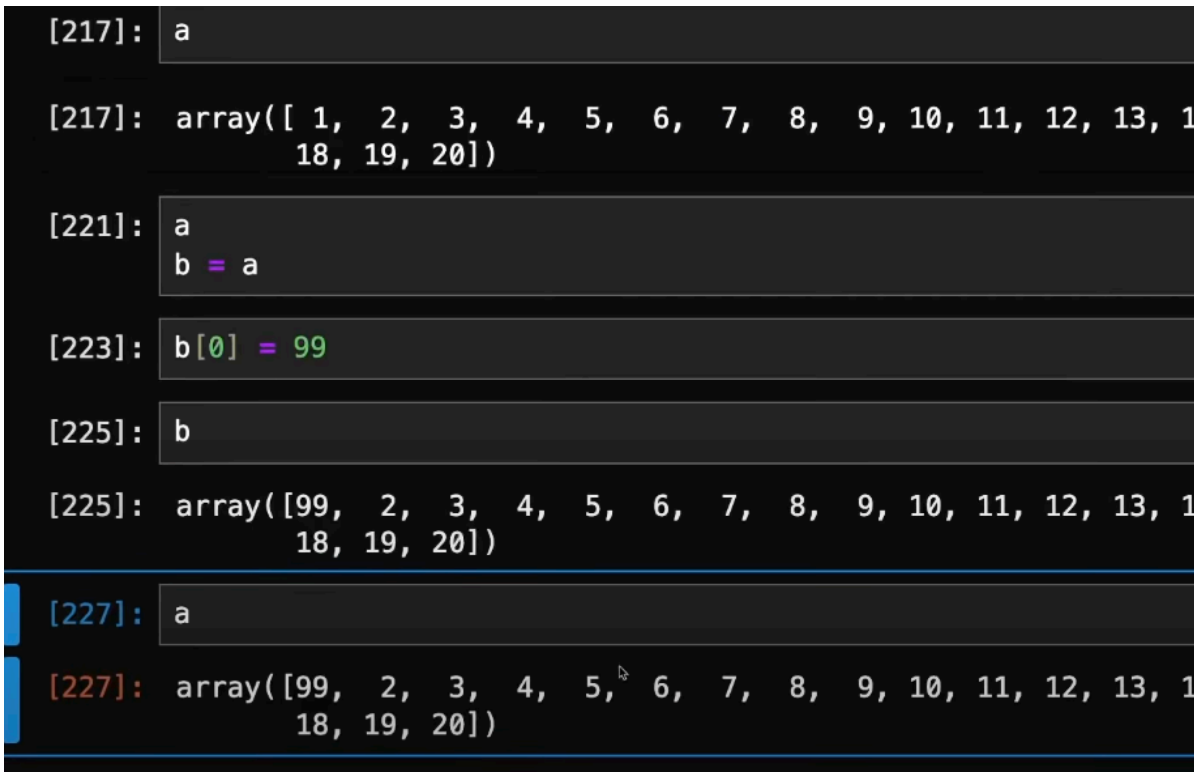
DEEP COPY AND SHALLOW COPY

Deep copy আর shallow copy

Deep copy হত list এর মধ্যে যেখানে আমরা slice করে broadcasting / arithmetic operation করতাম তখন main list এ change হয়ে যেত ।
But array এর মধ্যে হচ্ছে shallow copy এখানে আমরা slice করে নিলে আর সেখানে পরিবর্তন করলে আমাদের main array এর মধ্যে পরিবর্তন হবে না ।



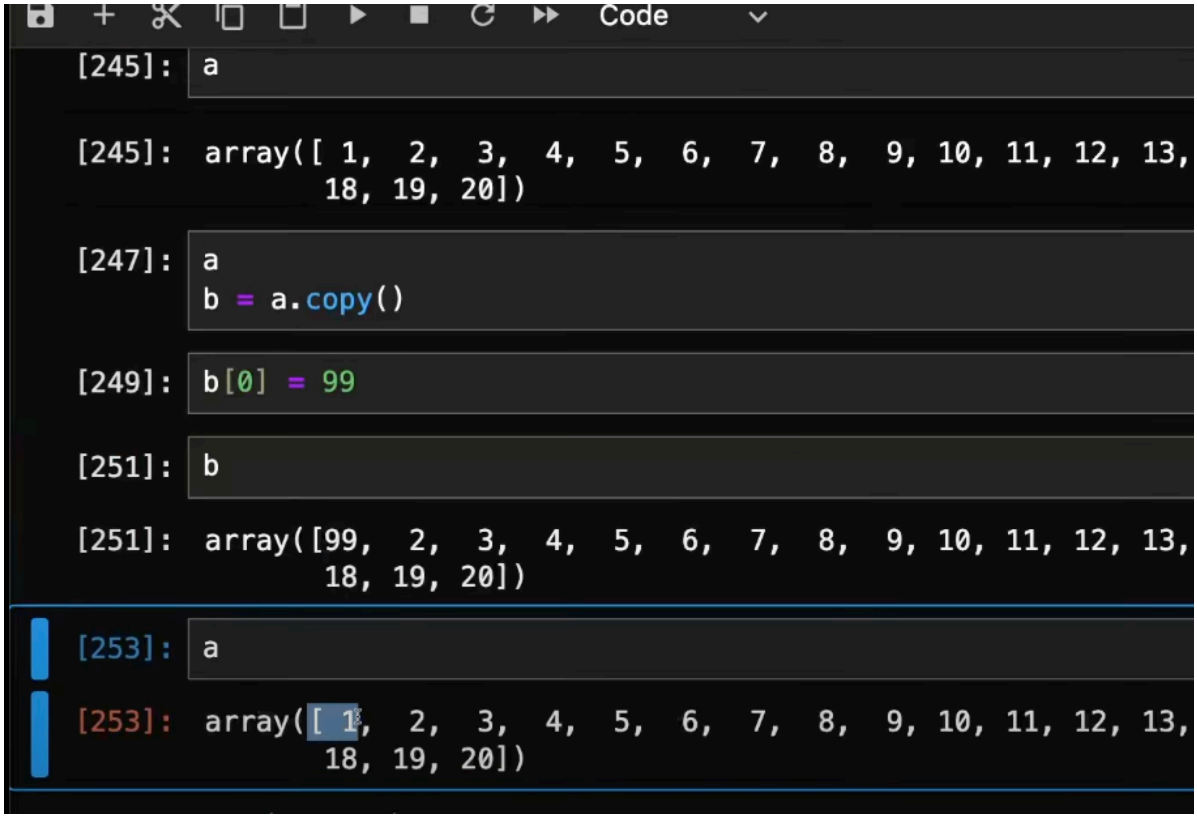
Slice করে নিলে টা অন্য আরেকটা location এ save হবে কিন্তু আমরা যখন পুরো array কে কপি করব তখন ডিপ কপি হয়ে মেইন array change হয়ে যাবে।

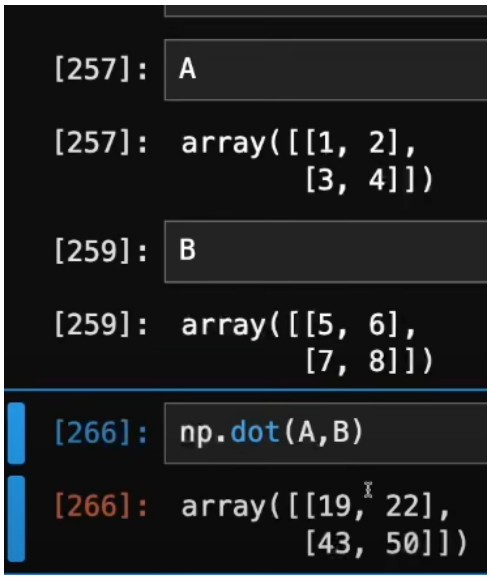


কিন্তু আমরা যদি deep copy না চাই তাহলে আমরা array a এর থেকে copy করে b তে রাখলে

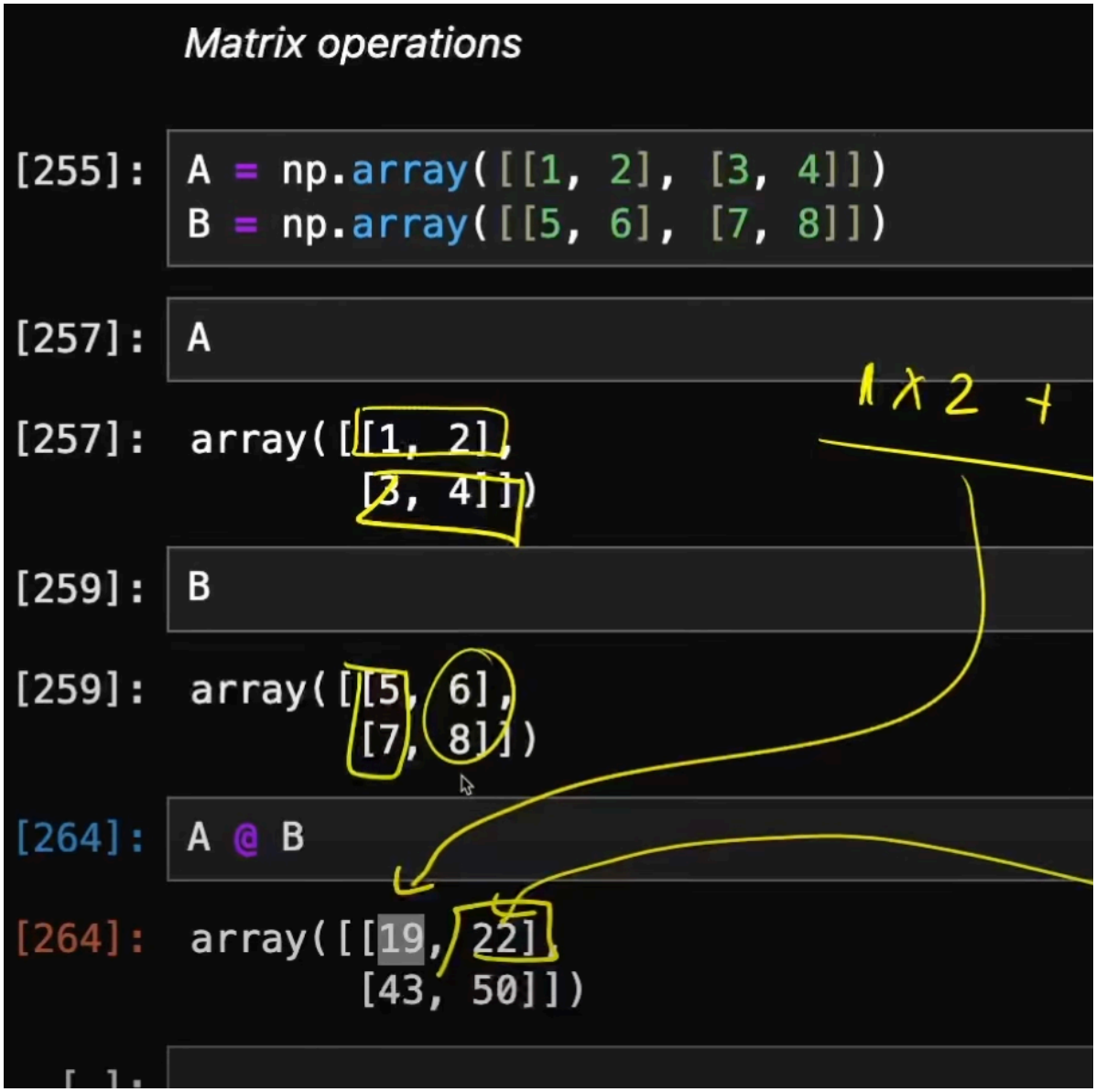
তাহলে b = a.copy() ফাঙ্কশন নিব।

Matrix Multiplication : using @ sign

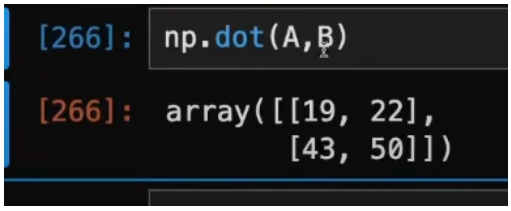




**********Matrix operation **********



Also like that we can a multiplication to b



Transpose matrix : row convert into column and column convert into row .

```
A
```

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

```
A.T
```

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

**********ARRAY MODIFICATION**********

**********Split & Stack **********

We can't stack two matrix we can stack two array / vector . Stack মনে দুইটা array / vector কে জোড়া লাগানো।
আমরা stack তিনভাবে করতে পারি vertical stack বা horizontal stack np.vstack((a,b)) বা horizontally np.vstack((a,b)), np.column_stack((a,b)) .

Stacking Array

```
[286]: a = np.array([1, 2, 3, 4])
       b = np.array([5, 6, 7, 8])
```

```
[294]: np.vstack((a,b))
```

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

```
[296]: np.hstack((a,b))
```

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

```
[298]: np.column_stack((a,b))
```

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

Split 2d array তে থাকে।

সমান পার্ট এ করতে হবে split . 3 টা করতে গেলে error আসবে।

```
[302]: c = np.arange(16).reshape(4, 4)
       c

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

[311]: np.hsplit(c,2)

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

```
np.hsplit(c,4)

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

4 টা split .

ভার্টিকাল স্প্লিট।

```
[7]: np.vsplit(c,2)

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

```
np.vsplit(c,4)

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

```
[323]: a = np.vsplit(c,4)

[325]: for i in a:
           print(i)

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

INSERT ———
————————————————

আমরা কোন array এর মধ্যে নতুন element insert করতে পারি না কারণ তার সাইজ ফিক্সড থাকে । আমরা যখন insert operation চালাই তখন মেইন array এর কপি তৈরি হয়।

```
arr = np.array([10,20,30,40,50,60])
print(arr)
new_arr = np.insert(arr, 2, 100)
print(new_arr)
```

২

Axis = 0 মানে row wise insert operation চলবে।

```
arr_2d = np.array([[1,2],[3,4]])
print(arr_2d)
#insert a new row at index 1
new_arr_2d = np.insert(arr_2d, 1, [5,6], axis=1)
print(new_arr_2d)
```

Axis = None মানে flatten copy করবে। 1d array তে insert করবে।

```
arr_2d = np.array([[1,2],[3,4]])
print(arr_2d)
#insert a new row at index 1
new_arr_2d = np.insert(arr_2d, 1, [5,6], axis=None)
print(new_arr_2d)
```

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

Append ———
————————————————

Append at the end of the array element add হবে। আর এখানেও নতুন array যেটা create হবে সেটা তে modification করলে main array তে modification হবে না।

```
arr = np.array([10,20,30])
new_arr = np.append(arr, [40,50,60])

print(new_arr)
```

CONCATENATE ————
————————————————

```
"""
np.concatenate((array1, array2), axis = 0)

axis 0 > vertical stacking
axis 1 > horizontal stacking
"""

import numpy as np

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

new_arr = np.concatenate((arr1, arr2))
print(new_arr)
```

DELETE ———>

————————————————

```
"""
np.delete(array, index, axis = None)
flattern array
"""
import numpy as np

arr = np.array([10,20,30,40,50,60])
print(arr)
new_arr = np.delete(arr, 0)
print(new_arr)
```

Axis - 0 মানে row wise

```
import numpy as np

arr_2d = np.array([[1,2,3],[4,5,6]])

new_arr_2d = np.delete(arr_2d, 0, axis=0)
print(new_arr_2d)
```

**********HANDLING MISSING VALUE**********

np.isnan(array) — detecting missing value . (Not a number) it's return boolean array [true for nun values)
np.ininf() -- infinite value detect করার জন্য ব্যবহার করি।
np.nan_to_num() — missing value replaced another value . Default value always 0 .

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

print(np.isnan(arr))

print(np.nan == np.nan)
```

Missing value এর জন্য true return করবে।

```
dling Missing Values/isnan.py"
 [False False  True False  True False
```

No we can't compare it directly .

Handle missing value . To replaced another value.

```
dling Missing Values > nan_to_num.py > ...
1   #np.nan_to_num(array, nan=value) default - 0
2   import numpy as np
3
4   arr = np.array([1,2,np.nan, 4, np.nan, 6])
5
6   cleaned_arr = np.nan_to_num(arr,nan=100)
7   print(cleaned_arr)
```

```
#np.isinf(array) 10^1000
#1/0

import numpy as np

arr = np.array([1,2,np.inf, 4, -np.inf, 6])

print(np.isinf(arr))
```

```
dling Missing Values/infinte.py"
[False False  True False  True False]
sagarchouksey@MacBook-Air Numpy for Data-S
```

যদি infinity value replaced করতে চাই।

```
import numpy as np

arr = np.array([1,2,np.inf, 4, -np.inf, 6])

print(np.isinf(arr))

cleaned_arr = np.nan_to_num(arr, posinf=1000, neginf=-1000)

print(cleaned_arr)
```