In [1]:
```
def squares(values):
    result = []
    for v in values:
        result.append(v * v)
        return result
```

In [2]:
```
to_square = range(100000)
```

time how long it takes to repeatedly square them all

In [3]:
```
%timeit squares(to_square)
```

```
1.51 µs ± 558 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Using NumPy and vectorized arrays:

The example can be rewritten as follows

In [6]:
```
import numpy as np
```

In [1]:
```
# now lets do this with a numpy array
array_to_square = np.arange(0, 100000)

# and time using a vectorized operation
%timeit array_to_square ** 2
```

```
236 µs ± 128 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Creating NumPy arrays and performing basic array operations

methods 1

A NumPy array can be created using multiple techniques. The following code `creates a new NumPy array object from a Python list` :

In [6]:
```
arr1 = np.array([11, 22, 33, 44, 55])
arr1
```

Out[6]:
```
array([11, 22, 33, 44, 55])
```

We can find the `type of array` , `size of array` , `shape of array` and `dtype of elements`

In [7]:
```
type(arr1)
```

Out[7]:
```
numpy.ndarray
```

In [8]:
```
np.size(arr1)
```

Out[8]:
```
5
```

In [10]:
```
arr1.dtype
```

Out[10]:
```
dtype('int32')
```

In [12]:
```
# 5 jo hai size bata rha hai ismai no of elements kitne hain, or comma (,) bata rha hai ismai sirf 1 row hai,
..
# ..jab bhi mai vector ki shape nikalunga to wo shape comma (,) ki shakal mai ayegi
################################
#  | 1, 2, 3, 4 |
#  | 3, 2, 1, 4 |
# it is one matrix(2d array) and we can calculate its order of matrix:
# Order = (rows, cols)
#       = (2, 4)
################################
###########################
#  | 1, 2, 3, 4 |  agar hum separately sirf isko dekhen to humein ye vector ke to par dikhti hain iska ordery
e hai = (4,)...
# .. dekhne mai 4 cols hain lekin yahan sirf row hai to ye sirf ye bata rha hota hain ke this a vector of 4 e
lements
#######################
np.shape(arr1)
```

Out[12]:
```
(5,)
```

In [13]:
```
# agar mai yahan dtype ka argument mai float32 pass kardun to ye saare elements ko array mai float type karde
ga

testArr1 = np.array([1, 2, 3, 4, 5], dtype="float32")
testArr1
```

Out[13]:
```
array([1., 2., 3., 4., 5.], dtype=float32)
```

In [14]:
```
# agar mai yahan dtype="float32" nhi likhta lekin agar koi 1 bhi number floating point hojata hai, to wo saar
e members ko..
# ..floating point mai convert kardega, uski dtype automatically float mai convert hojayegi

testArr2 = np.array([1, 2, 3, 4, 5.2])
testArr2
```

Out[14]:
```
array([1. , 2. , 3. , 4. , 5.2])
```

In [15]:
```
arr1.ndim  #returns the dimenssion of the array either it is 1d, 2d, 3d or nd
```

Out[15]:
```
1
```

- 1d array ko hum vector bhi kehte hain linear algebra mai, iski dimension 1 hogi
- `1d array =[1,2,3]  >>> vector, dimension>>1`

#

- 2d array mai 1,2,3 1st row and 2,3,4 2nd row, isko hum linear algebra mai `matrix` kehte hain or Machine learning mai `tensor` kehte hain, or iski dimension 2 hogi, opening ya closing brackets ki tadaad agar 2 hai to 2d hogi, 5 arhe hain to 5d hogi
- `2d array = [[1,2,3],[2,3,4]]  >>matrix/tensor >>2`

method 2

We can create a python range into numpyy array

In [16]:
```
np.array(range(10))
```

Out[16]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

method 3

make "a range" starting at 0 and with 10 values np.arange(0, 10)

> arange function same python ke range function ki tarah kaam karte hai

In [17]:
```
np.arange(10)
```

Out[17]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:
```
# 0 <= x < 10 , increment by two
np.arange(0, 10, 2) # starting, numberOfValues, steps
```

Out[18]:
```
array([0, 2, 4, 6, 8])
```

In [20]:
```
# 10 >= x > 0, counting down
np.arange(10, 0, -1)
```

Out[20]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

In [22]:
```
# linear spacing ye 0 sai leke 10 tak ki values create karega usko 20 parts mai divide karedga, evenly part h
onge, to 20..
#.. parts create honge or har 1 ke darmiyaan same differnce hoga, linear spacing ka matlab ye hai ke har 1 da
rmiyan same..
# .. differnce hoga, evenly distributed hogi, ya divided hogi
# evenly spaced #'s between two intervals
np.linspace(0, 10, 20)
```

Out[22]:
```
array([ 0.        ,  0.52631579,  1.05263158,  1.57894737,  2.10526316,
        2.63157895,  3.15789474,  3.68421053,  4.21052632,  4.73684211,
        5.26315789,  5.78947368,  6.31578947,  6.84210526,  7.36842105,
        7.89473684,  8.42105263,  8.94736842,  9.47368421, 10.        ])
```

Array creation functions

Function	Description
`array`	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
`asarray`	Convert input to ndarray, but do not copy if the input is already an ndarray
`arange`	Like the built-in `range` but returns an ndarray instead of a list
`ones`, `ones_like`	Produce an array of all 1s with the given shape and dtype; `ones_like` takes another array and produces a ones array of the same shape and dtype
`zeros`, `zeros_like`	Like `ones` and `ones_like` but producing arrays of 0s instead
`empty`, `empty_like`	Create new arrays by allocating new memory, but do not populate with any values like `ones` and `zeros`
`full`, `full_like`	Produce an array of the given shape and dtype with all values set to the indicated "fill value" `full_like` takes another array and produces a filled array of the same shape and dtype
`eye, identity`	Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)

Expicitly Convert /Cast Data type of ndarray

In [25]:
```
# currently the dtype of array is int32
int_arr = np.array([100, 200, 300, 400, 500])
int_arr.dtype
```

Out[25]:
```
dtype('int32')
```

In [26]:
```
# no we can change it using a function or through providing as string in argument
float_arr = int_arr.astype(np.float64)
float_arr.dtype
```

Out[26]:
```
dtype('float64')
```

Note:

Casting floating point into integer data type will `truncate the decimal part`

- `truncate` means to cut or short

In [27]:
```
arr3 = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr3
```

Out[27]:
```
array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
```

In [28]:
```
# now all floats have been converted to ints
arr4 = arr3.astype(np.int32)
arr4
```

Out[28]:
```
array([ 3, -1, -2,  0, 12, 10])
```

Arithmetic with NumPy Arrays

- NumPy arrays will vectorize many mathematical operators.
- The following example creates a 10-element array and then multiplies each element by a constant:

In [30]:
```
# multiply numpy array by 2
a1 = np.arange(0, 10)
a1 * 2
```

Out[30]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [31]:
```
# add two numpy arrays
a2 = np.arange(10, 20)
a1 + a2
```

Out[31]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

Accessing 1-D array elements:

In [34]:
```
print(a2)
```

```
[10 11 12 13 14 15 16 17 18 19]
```

In [32]:
```
a2[0]
```

Out[32]:
```
10
```

In [33]:
```
a2[-1]
```

Out[33]:
```
19
```

In [35]:
```
print(a1)
```

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

In [37]:
```
# select 0-based elements 0 and 2
a1[0], a1[2] # 2 values 1 saath utha li ye tuple ki form mai dikha rha hai
```

Out[37]:
```
(0, 2)
```

2d numpy Array

- abhi tak jo kaam humne kia tha wo saara ka saara 1d array mai kia tha ab wohi saara kaam hum 2d array mai karenge

Creating a 2d array by python list :

In [8]:
```
# ye list as a row add horhi hain
arr2d = np.array([[1,2,3,4,5,6], [11,22,33,44,55,66]])
arr2d
```

Out[8]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [11, 22, 33, 44, 55, 66]])
```

A more convenient and efficient means is to use the NumPy array's

```
>>> np.reshape() <<<<
reshape method ki madad sai kisi bhi dimension ka matrix create karna bara asaan hogya
```

method to reorganize a one-dimensional array into two dimensions.

In [9]:
```
# ye humne 1d array create kari isko hum 2d array mai convert karenge using reshape method
# yahan humne 1d array create kia jismia hamare pass 20 members hain
arr1d = np.arange(20)
arr1d
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [10]:
```
# ab jab 20 members ko 2 dimension mai create karna hai to sab sai pehle 1 cheez zehen mai rhe jo row or cols
ayenge..
# ..unka product bhi 20 hona chahiye 21 ya 19 nhi hoskta
# mai yahan keh rha hun ke arr1d ko reshape kardo 4 rows and 5 cols mai, 4 or 5 ka jo product hai wo hai 20 y
aani 20 elemnts..
# ..poore fit hone chahiyen is sai kam ya ziyada honge to ye apko error dega
# yahan 2d array jo hai wo reshape ke through create hogyi

arr2d = arr1d.reshape(4, 5)
arr2d
```

Out[10]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [11]:
```
# uppar humne 2 kaam kiye pehle 1d array banyi phir ussey reshape kia 2d array mai isko mai 1 single step mai
create karsakt hun

arr2d = np.arange(20).reshape(4,5)
arr2d
```

Out[11]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [12]:
```
# size of any dimensional array is the # of elements
# size humein ye batata hai ke array mai kitne members hain isne humein return kia ke ismai 20 elments hain
np.size(arr2d)
```

Out[12]:
```
20
```

In [13]:
```
# can ask the size along a given axis (0 is rows)
# 0 kia return karta hai? 0 ye batata hai ke row ka size kitna hai, rows kitni hain? to isne huemin 4 return
 kiya yaani 4 rows

np.size(arr2d, 0)
```

Out[13]: 4

In [14]:
```
# and 1 is the columns
# agar mai 0 ki jaga 1 pass karunga to 1 humein cols return karega

np.size(arr2d, 1)
```

Out[14]: 5

In [15]:
```
# order batayega ke 4 rows and 5 cols, ye vector ki soorat mai sirf (4,) dikhayega or 2d matrix ki soorat mai
rows and cols..
# .. dono batyega

arr2d.shape
```

Out[15]: (4, 5)

a 2d array again can be reshaped to 1d

In [16]:
```
# in this way we need to know how many elements are there in original array
# we are open to reshape to any dimension either 1d or other

arr1d = arr2d.reshape(20)
print(arr1d)
```

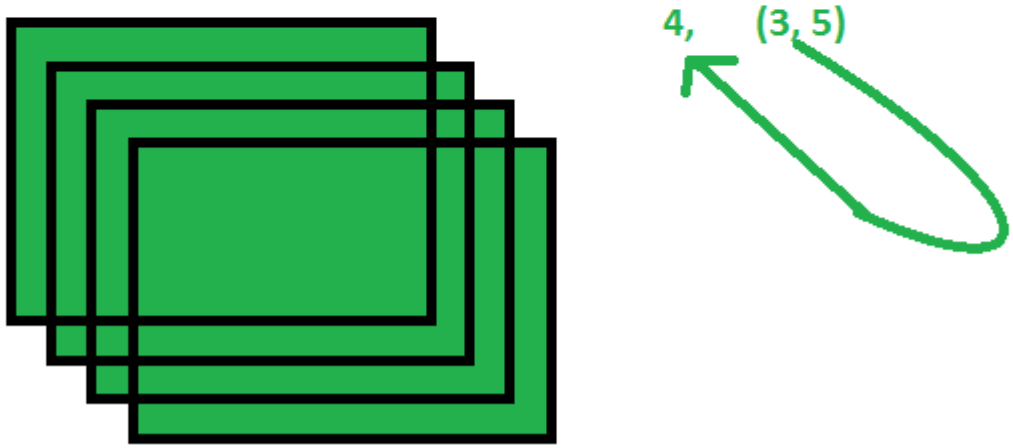
```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Pictorial Representataion of 3 dimensional Array (3d array)

> There are 4 matrx in this picture, and if one matrix is of (3, 5) order so all matrix will have same order of (3, 5)

> Now the shape will be read as "There are 4 matrices of 3 by 5",

> yaani mai ye keh sakta hun ke (3,5) ki 4 matrices hain jo eik doosre ke peechay hain



In [17]:
```
# reshaping a 2darray to 3darray
# mai yahan 64 elements ki 1 array bana rha hun jiske andar 8 rows or 8 cols hain

arr2d = np.arange(64).reshape(8,8)
print(arr2d)

print("==============================")

# again reshaping arr2d to arr3d
# yaani humne 2d array ko further 3 dimension mai tor diya to three dimension ko read kese karte hain matrix
 ke andar?
# ab kia # of rows and # of cols same hone chchiyen ? nhi mai kuch bhi rakhsakta hun
arr3d = arr2d.reshape(4,4,4)  #(depth, rows,columns)
arr3d
```

```
[[ 0  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 31]
 [32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47]
 [48 49 50 51 52 53 54 55]
 [56 57 58 59 60 61 62 63]]
==============================
```

Out[17]:

```
array([[[ 0,  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, 31]],

       [[32, 33, 34, 35],
        [36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59],
        [60, 61, 62, 63]]])
```

In [13]:
```
# to # of rows or # cols zaruri nhi ke same hon kuch bhi ap rakh sakte hain

arr3d = arr2d.reshape(2, 4, 8)
arr3d
```

Out[13]:
```
array([[[ 0,  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, 31]],

       [[32, 33, 34, 35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44, 45, 46, 47],
        [48, 49, 50, 51, 52, 53, 54, 55],
        [56, 57, 58, 59, 60, 61, 62, 63]]])
```

>>>>> np.ravel() >>>>>

- is another methods but it direclty converts the array to `1d only`
- ye method zize wagera kuch bhi nhi mangta qk iska kaam hee ye hai ke isko jo bhi matrix mile ussey 1d mai convert kardega, seedha seedha flat kardega jabke `np.reshape()` sai hume kisi bhi dimension mai convert karsakte hain matrix ko
- lekin `np.ravel()` resahpe sirf 1d mai hee karega

In [14]:
```
# isne seedha seedha 2d array ko 1d mai kardiya

raveled_arr1d = arr2d.ravel()
raveled_arr1d
```

Out[14]:
```
array([ 0,  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, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63])
```

>>>>>>>> `np.ravel()` and `np.reshape()` <<<<<<<<<

- `.reshape()` jab hum karte hain to ye copy nhi banata data ki balke view banata hai
- to original array ki shape change nhi hogi ye naya return karte haain matrix, nayi shape ke andar johumne dee hai

- Even though `.reshape()` and `.ravel()` do not change the shape of the original array or matrix, they do actually return a one-dimensional view into the specified array or matrix. If you change an element in this view, the value in the original array or matrix is changed.
- The following example demonstrates this ability to change items of the original matrix through the view:

In [15]:
```
# humne arr2d ko ravel kardiya to wo reshape hokar 1d mai convert hogyi
# ab mai ye kar rha hun ke jo raay mai ne ravel ki hai uski 0 index par ye 999 daaldo

raveled_arr1d[0] = 999
raveled_arr1d
```

Out[15]:
```
array([999,   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,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63])
```

In [16]:
```
# ab mai arr2d ko check karta hun jisko humne reshape kia tha ravel ke through
# to arr2d ko jab mai ne check kia to wo change jo hai wo arr2d mai bhi agya even though mai ne new variable
 banaya tha
# to ye pata lagi ke ye copy banake alag data nhi rakhta balke ye ussi data ka view change karke dikha rha ho
ta hai, or agar..
# .. view ke andar koi change hoga to original mai bhi change nazar ayega or agar original mai change hoga to
view mai bhi..
# .. change nazar ayega, to .revel() or .reshape() dono view banakar dete hain or agar view ke andar ap chang
e karenge to ..
# .. wo original ko bhi hit karega

arr2d   # array shows the effect of change in ravel
```

Out[16]:
```
array([[999,   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,  31],
       [ 32,  33,  34,  35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44,  45,  46,  47],
       [ 48,  49,  50,  51,  52,  53,  54,  55],
       [ 56,  57,  58,  59,  60,  61,  62,  63]])
```

In [17]:
```
# abhi humne .ravel() ka dekha ab hum .reshape() ka dekhenge
# yahan mai ne 2d array banyi through reshape()
array2d = np.arange(9).reshape(3, 3)
array2d
```

Out[17]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [18]:
```
# ab mai yahan isko 1d array banunga dobara
# 1d kese banega maine yahan rows and cols nhi bataye to ye 1d vector banega
# to ye 1 new vector nhi hai balke ussi ka view change karke dikhaya hai isne, yaani matrix wohi hai bas usne
apko dikhadiya..
# ..ke apko ye esa dikhega

array1d = array2d.reshape(9)
array1d
```

Out[18]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [19]:
```
# ab mai ne kaha ke iske 0 index par 777 daaldo

array1d[0] = 777
array1d
```

Out[19]:
```
array([777,   1,   2,   3,   4,   5,   6,   7,   8])
```

In [20]:
```
# ab mai us original 2darray ko check karta hun jisko reshape kiya tha
# to ye bhi pass by refernce hai

array2d   # show the change done in array1d coz array 1d is view of array2d
```

Out[20]:
```
array([[777,   1,   2],
       [  3,   4,   5],
       [  6,   7,   8]])
```

- ye bhi ravel ki tarah seedha array ko 1d mai flat kardega lekin ye copy banaleta hai, ye instead of view copy banaleta hai

>>>>>np.flatten()>>>>>

- is another methods but it direclty converts the array to 1d only.
- The .flatten() method functions `similarly to .ravel()` but `instead returns a new array with copied data instead of a view .`
- Changes to the result do not change the original matrix:

In [21]:
```
# ye mai ne 1 array create kari

array = np.arange(25).reshape(5, 5)
array
```

Out[21]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [22]:
```
# maine kaha array ko faltten kardo

flattened_array = array.flatten()
flattened_array
```

Out[22]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [23]:
```
# flattened_array ke index 0 pe 555 daaldo to isne daaldiya

flattened_array[0] = 555
flattened_array
```

Out[23]:
```
array([555,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24])
```

In [24]:
```
# mai ne kaha ke original array ko check karo, to original array mai change nhi hua
# to .flatten() copy banata hai lekin .ravel() or .reshape() view banate hain

array   # there is no effect in original
```

Out[24]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

- danger is liye likha hai ke ye original ko reset kardeta hai

np.resize() Danger!!!!

- The `.resize()` method functions `similarly to the .reshape()` method, except that `while reshaping returns a new array with data copied into it`, `.resize()` `performs an in-place reshaping of the array.` :

In [25]:
```
# mai ne newarray banyi usko reshape kia into 3 by 3
newarray = np.arange(0, 9).reshape(3,3)
newarray
```

Out[25]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [26]:
```
# ab mai newarray ko resize kar rha hun r resize ko nex line mai get karna chah  raha hun ke kia hai
# output mai kuch bhi nhi arha jab mai cell execute kar rha hun, to ye keh rha hai ke original ko change kard
iya hai jao or..
# jo dekhna hai original mai dekhlo,original mera seedha hogya 3 by 3 tha usne apna kaam wahan pe resize ka k
ardiya or kuch..
# .. apko return nhi kia

resized = newarray.resize(1, 9)
resized # it is returning none infact it changed the original data
```

In [27]:
```
newarray
```

Out[27]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

Basic Indexing and Slicing of Multidimensional Arrays

In [2]:
```
# create 2d array
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2d
```

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

Indexing:

It is a process in which we target(access)indvidual or group of contagious values of ndarrays.

In [3]:
```
# accessing a complete row
arr2d[2]
```

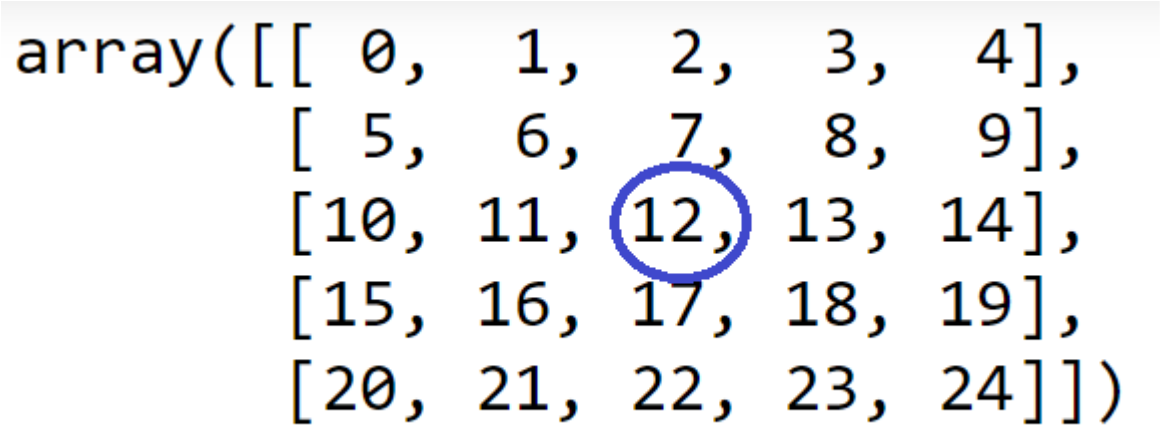
Out[3]:
```
array([7, 8, 9])
```

In [4]:
```
# accessing a complete columns
arr2d[:,1]
```

Out[4]:
```
array([2, 5, 8])
```

In [5]:
```
# accessing an element of a 2d array
array2d = np.arange(25).reshape(5,5)
array2d
```

Out[5]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



In [19]:
```
# array2d[2][2]    # isko hum ese bhi likh sakte hain

array2d[2,2] # row number 2 hai or col number bhi 2
```

Out[19]: 12

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [20]:
```
array2d[0,3], array2d[1,2],array2d[2,4],array2d[4,1]
```

Out[20]: (3, 7, 14, 21)

Structure for indexing and slicing in a 2darray

====2DARRAY =====

Shape>>> (Rows , Columns)

(__ : __ , __ : __)

(start of row : end of row , start of col , end of col)

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [22]:
```
#Light Blue
array2d[0:4, 1:2]
```

Out[22]:
```
array([[ 1],
       [ 6],
       [11],
       [16]])
```

In [23]:
```
#yellow
array2d[1:5, 0:3]
```

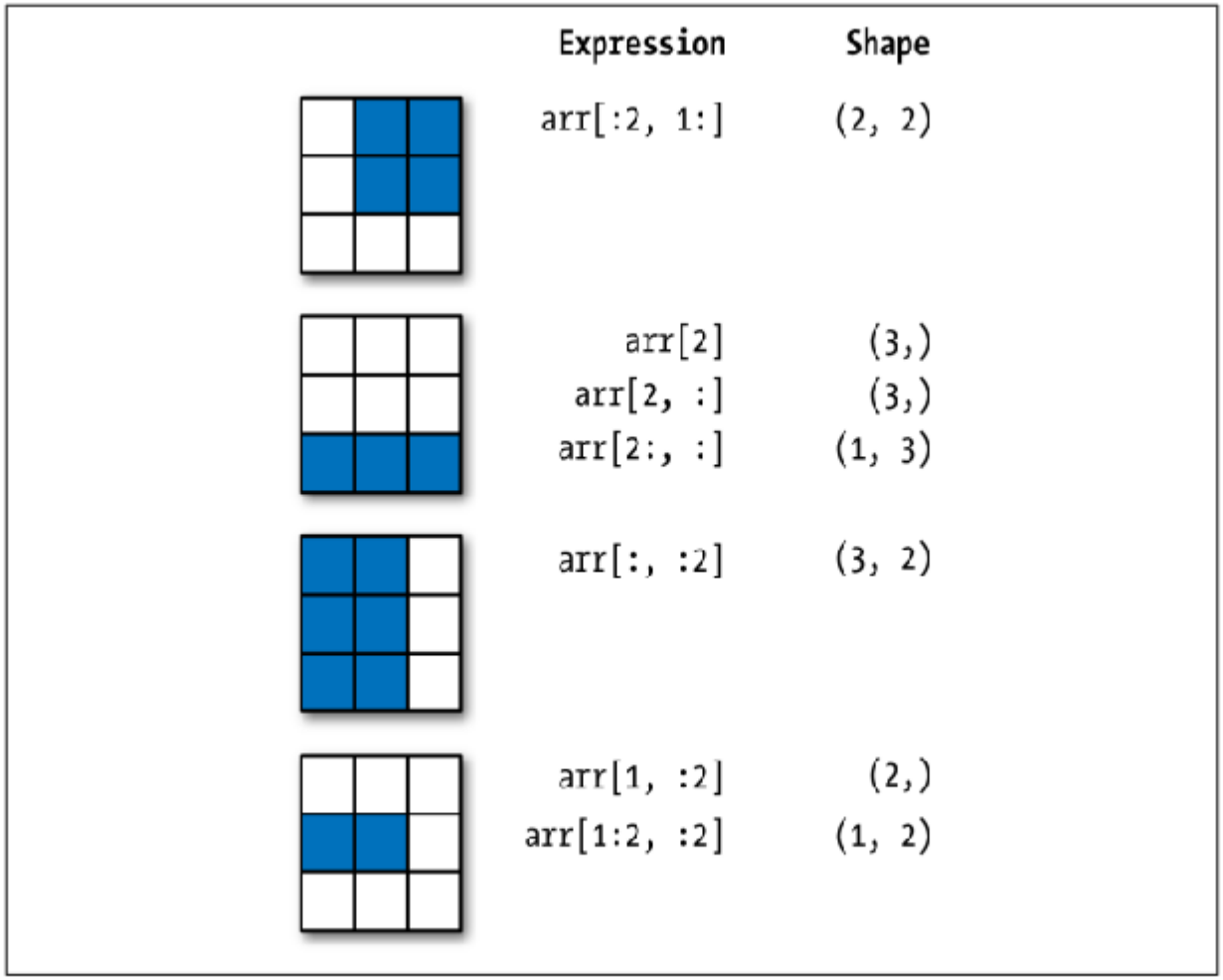
Out[23]:
```
array([[ 5,  6,  7],
       [10, 11, 12],
       [15, 16, 17],
       [20, 21, 22]])
```

In [24]:
```
#blue
# array2d[0:2,-2:]
array2d[0:2, 3:]
```

Out[24]:
```
array([[3, 4],
       [8, 9]])
```

In [25]:
```
#green
array2d[3:,2:]
```

Out[25]:
```
array([[17, 18, 19],
       [22, 23, 24]])
```



Two-dimensional array slicing

Question:

What is the difference between indexing and slicing ??

Indexing:

- Targets the element of the array. The returned is the `individual vlaue or group of values` having their `own data types`.

In [27]:
```
array2d
```

Out[27]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [29]:
```
# array2d mai sai mai ne indexing ki hai, 4th row or 2nd col ki, to ye return kar rha hai 22
# ab ye 22 kia hai? ye 1 number hai jiski dimension dimensionless hai yaani 0 hai, iska matlab ye hua ke jab
 mai indexing karta
# .. hun to mere pass element nikal ke ata hai, ab wo element apni individual capacity rakhta hai, wo apni ar
ray ko represent..
# .. nhi kar rha, ab uski apni 1 alag data type hai joke int hai
array2d[4][2]
```

Out[29]:
```
22
```

Slicing:

- Gives the part of the array that have the `same deimension(shape) as the full array has`.

In [30]:
```
# isne bhi wohi 22 diya lekin 1k slice kia hai to 22 ki dimension 2 hai qk uski array bhi 2d thi
# misaal: agar aap pizza kha rhe hain, or pizza ko slice karke nikaalenge 1 piece to jo piece niklega wo pizz
a hee kehlayega
# to ye 2d hai to 2d hee kehlayega
# lekin agar pizza mai sai agar eik mushroom uppar sai ap uthalen or usko bolen ke ye piza hai to ye piza nhi
hosakta..
# ..to wo individual mushroom ki apni value hogyi ab, ab wo piza ka part nhi rha
# bilkul ese hee jab mai slice karke nikaalte hun to wo original dimension jo hai wo maintained karta hai
# or jab mai indexing karta hun to indexing target karti hai element of array
# e.g mai nex cell mai indexing karta hun...

array2d[4:, 2:3]     # notice the brackets around 22 shows its not an element but a 2d array with one element
22.
```

Out[30]:
```
array([[22]])
```

In [60]:
```
# to chaaron values pass karna parenge bhale wo empty hon tabhi slicing hogi warna indexing kehlayegi
# agar mai iske chaaron parameters poore dunga to slicing hogi as in above cell

array2d[4,2:3]
```

Out[60]:
```
array([22])
```

In [62]:
```
# col number 4 mai ne get kia hai from row 2 to row 4
# to maine col select kia hai, mai ne kaha ke col number 2 lao to wo isko flat shakal mai dikha rha hai, row
 ki shakal mai..
# ..iska matlab ye hai ke values collect karke laake isne apko yahan dikhadin

array2d[1:4,2]
```

Out[62]:
```
array([ 7, 12, 17])
```

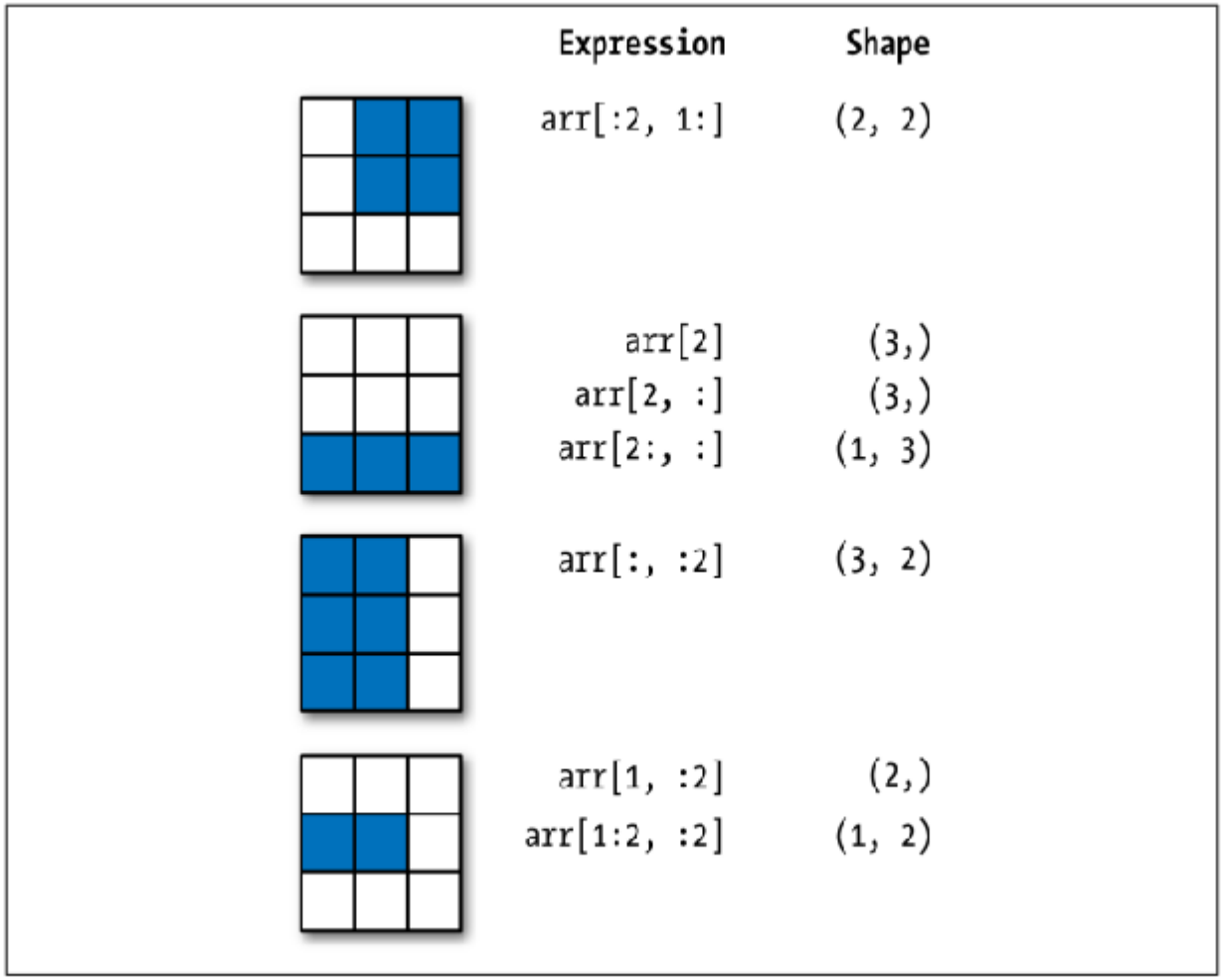
In [63]:
```
# lekin mujhe agar col chahiye lekin col ki shakal mai , verticallly hee nazar aye, to phir mujhe slice hee k
arna parega
#
array2d[1:4,2:3]
```

Out[63]:
```
array([[ 7],
       [12],
       [17]])
```

In [33]:
```
# maine kaha row number 2 utha lo tho uski dimension 1 arhi hai qk ye indexing hai agar ye slicing hoti to ye
2d hee ati
# or slicing sirf uswaqt slicing rehi hai jab ap rows and cols ke tamam parameters ko use karte hain e,g ap p
ic dehen:

array2d[2]
```

Out[33]:
```
array([10, 11, 12, 13, 14])
```



Two-dimensional array slicing

Box#1

In [35]:
```
testArr = np.arange(1,10).reshape(3, 3)
testArr
```

Out[35]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [42]:
```
testArr[:2, 1:]
```

Out[42]:
```
array([[2, 3],
       [5, 6]])
```

In [44]:
```
exp1 = testArr[:2, 1:]
np.shape(exp1)
```

Out[44]:
```
(2, 2)
```

Box#2

exp1:

In [45]:
```
testArr[2]
```

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

In [46]:
```
exp1 = testArr[2]
np.shape(exp1)
```

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

exp2:

In [47]:
```
testArr[2, :]
```

Out[47]:
```
array([7, 8, 9])
```

In [48]:
```
exp2 = testArr[2, :]
np.shape(exp2)
```

Out[48]:
```
(3,)
```

exp3:

In [49]:
```
testArr[2:, :]
```

Out[49]:
```
array([[7, 8, 9]])
```

In [50]:
```
exp3 = testArr[2:, :]
np.shape(exp3)
```

Out[50]:
```
(1, 3)
```

Box#3

exp1:

In [51]:
```
testArr[:, :2]
```

Out[51]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

In [52]:
```
exp1 = testArr[:, :2]
np.shape(exp1)
```

Out[52]:
```
(3, 2)
```

Box#4

exp1:

In [53]:
```
testArr[1, :2]
```

Out[53]:
```
array([4, 5])
```

In [54]:
```
exp1 = testArr[1, :2]
np.shape(exp1)
```

Out[54]:
```
(2,)
```

exp2

In [57]:
```
testArr[1:2, :2]
```

Out[57]:
```
array([[4, 5]])
```

In [59]:
```
exp2 = testArr[1:2, :2]
np.shape(exp2)
```

Out[59]:
```
(1, 2)
```

In [ ]:
```

```