NumPy

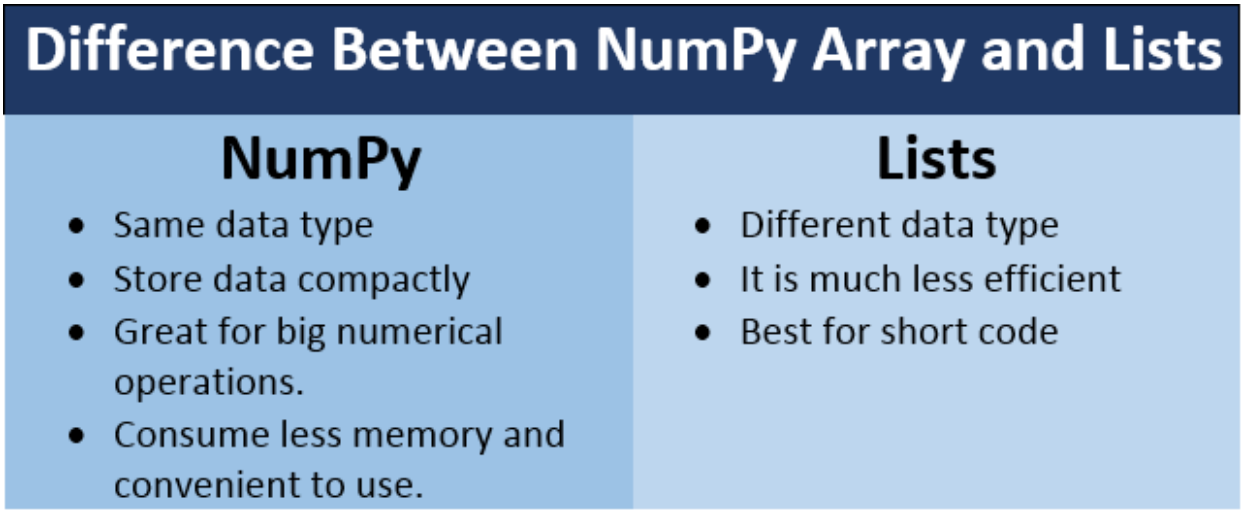
- **NumPy (Numerical Python) is a Python library.**
- **Used in almost every field of science.**
- **It is commonly used for working with numerical data in Python.**
- **It is core library of the Python.**

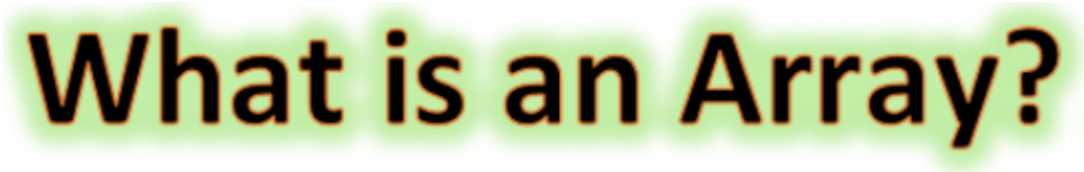
How to install NumPy is VsCode?

To install Numpy run this command **"pip install numpy"** in terminal window of VsCode. Not in OutPut window.\ ***NumPy is already installed Jupyter Notebook*.**

How to use NumPY

To Access NumPy and its functions import it in your python code like this. **"import numpy as np"**





- **An array is central data structure of the NumPy library.**

- **An array is a grid of values and it contains information about the raw data,how to locate an element and how to interpret an element.**
- **It has a grid of elements that can be indexed in various ways.**
- **The elements are all of the same type, referred to as the array dtype.**
- **An array can be indexed by a tuple of non-negative integers, by booleans, by another array or by integers.**

What is rank of an array?

The rank of the array is the number of dimensions.

What is the shap of an Array?

The shape of the array is a tuple of integers giving the size of the array along each dimension."

Tuple

tup=(90, "Chilla_version_2.0", True, 3.5)

Array Examples

In [2]:
```
import numpy as np
a = np.array([7,5,4,3,7])
a
```

Out[2]:
```
array([7, 5, 4, 3, 7])
```

In [5]:
```
#List of lists
import numpy as np
b = np.array([[10,11,12,20],[30,33,55,76],[90,78,65,34]])
b
```

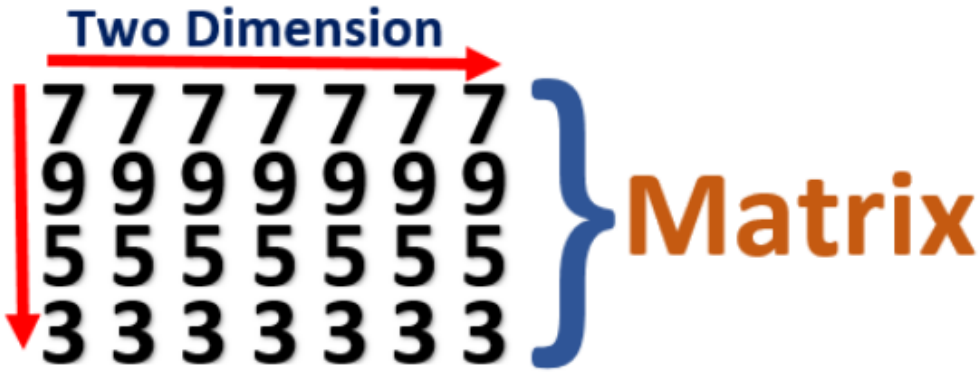
Out[5]:
```
array([[10, 11, 12, 20],
       [30, 33, 55, 76],
       [90, 78, 65, 34]])
```

Vector

A vector is an array with a single dimension(There is no difference between row and column vector).

Matrix

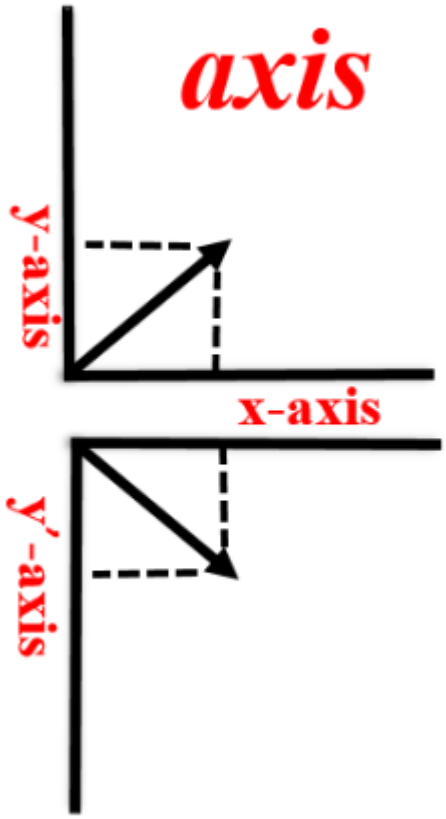
A matrix refer to an array with two dimensions.



Attributes of an array?

- **An array is a fixed-size container of items of the same type and size.**
- **The number of dimensions and items in an array is defined by its shape.**
- **The shape of the array is a tuple of integers giving the size of the array along each dimension.**
- **In NumPy, dimensions are called axes.**

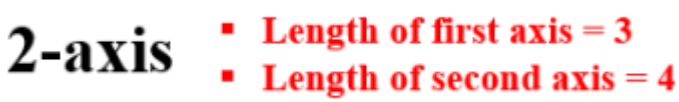
Axes



In [16]:
```
import numpy as np
c = np.array([[7,3,6,4],[5,9,3,3],[7,3,6,4]])
c
```

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

In above example array has 2 axes. The first axis has a length of 3 and the second axis has a length of 4.



Basic Arrays

We can use this function "*np.array()*" to create simple numpy array.

Import NumPy library

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

In [17]:
```
d=np.array([6, 7, 8])
d
```

Out[17]:
```
array([6, 7, 8])
```

We can also write it as:

In [14]:
```
np.array([6, 7, 8])
```

Out[14]:
```
array([6, 7, 8])
```

We can also create an array in other ways rather than creating from a sequence of elements.

With zeros() Fucntion

In [18]:
```
e=np.zeros(4)
e
```

Out[18]:
```
array([0., 0., 0., 0.])
```

With ones() Fucntion

In [22]:
```
f=np.ones(9)
f
```

Out[22]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

With empty() Fucntion

The function empty creates an array whose initial content is random and depends on the state of the memory.

The reason to use **empty** over **zeros** (or something similar) is speed - just make sure to fill every element afterwards.

In [29]:
```
# Create an empty array with 2 elements
# Result may vary.
g=np.empty(2)
g
```

Out[29]:
```
array([1., 1.])
```

By using range() Fucntion

اس میں ہم رینج کے طور ایک نمبر دیتے ہیں جہاں تک ہم ایرے پرنٹ کروانا چاہتے ہیں۔ جیسے اگلی مثال میں 9 رینج ہے

In [31]:
```
h=np.arange(9)
h
```

Out[31]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

6- Array having specific Range

You know what I mean.

For example 4 to 9.

9 is excluded because indexing start from "0".

In [32]:
```
i=np.arange(4,9)
i
```

Out[32]:
```
array([4, 5, 6, 7, 8])
```

7- With Specific Range and Interval

In [34]:
```
k=np.arange(3,30,3)
k
```

Out[34]:
```
array([ 3,  6,  9, 12, 15, 18, 21, 24, 27])
```

8- Line spaced Array

Mtlb ye k hm aik specific range provide krain gy k kaha se kaha tk array print

krni hai or ye bhi btain gy k us range k drmiyan kitni values ayen gi.

In [35]:
```
l=np.linspace(10,100,num=10)
l
# 10-100 k b/w aise 10 number print ho gy jin ka diffrence bilkl same ho gy.
```

Out[35]:
```
array([ 10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.])
```

9- Specific DataType in Array

In [36]:
```
m=np.zeros(7,dtype=np.int8)
m
```

Out[36]:
```
array([0, 0, 0, 0, 0, 0, 0], dtype=int8)
```

In [37]:
```
n=np.ones(8,dtype=np.float64)
n
```

Out[37]:
```
array([1., 1., 1., 1., 1., 1., 1., 1.])
```

In [2]:
```
m=np.ones(8)
m   #agr koi bhi data type use krain to as a default float use hoti...meri mrzi....
```

Out[2]:
```
array([1., 1., 1., 1., 1., 1., 1., 1.])
```

1-D Array

In [1]:
```
import numpy as np
c = np.array([89,45,53,34,3,3])
c
```

Out[1]:
```
array([89, 45, 53, 34,  3,  3])
```

In [3]:
```
# finding type
type(c)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
#Finding Length of array
len(c)
```

Out[4]:
```
6
```

In [5]:
```
#Finding Index of item.
c[0]
```

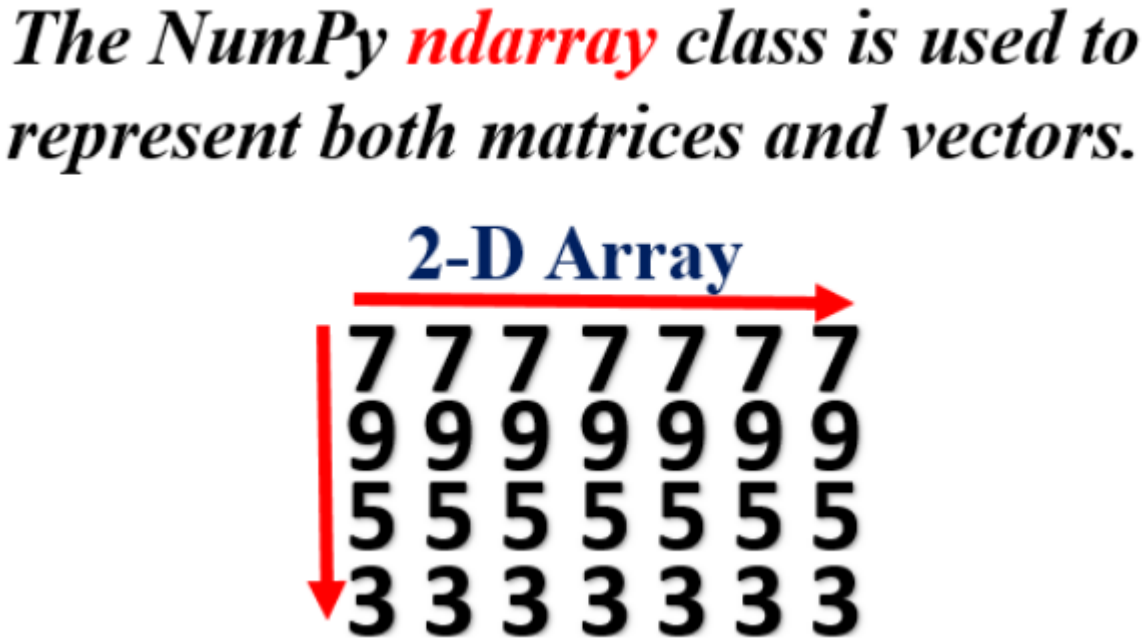
Out[5]:
```
89
```

In [6]:
```
#Is trah bhi array print kr skty hain
c[0:]
```

Out[6]:
```
array([89, 45, 53, 34,  3,  3])
```

2-D Array

A matrx refers to an array having two dimensions.



In [7]:
```
#List of lists
import numpy as np
d = np.array([[10,11,12,20],[30,33,55,76],[90,78,65,34],[35,44,89,55]])
d
```

Out[7]:
```
array([[10, 11, 12, 20],
       [30, 33, 55, 76],
       [90, 78, 65, 34],
       [35, 44, 89, 55]])
```

In [8]:
```
#Type
type(d)
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
#Length
len(d)
```

Out[9]:
```
4
```

In [10]:
```
#indexing
d[3]
```

Out[10]:
```
array([35, 44, 89, 55])
```

2-D with zeros() Function

In [12]:
```
np.zeros((3,4))
```

Out[12]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

2-D with ones() Function

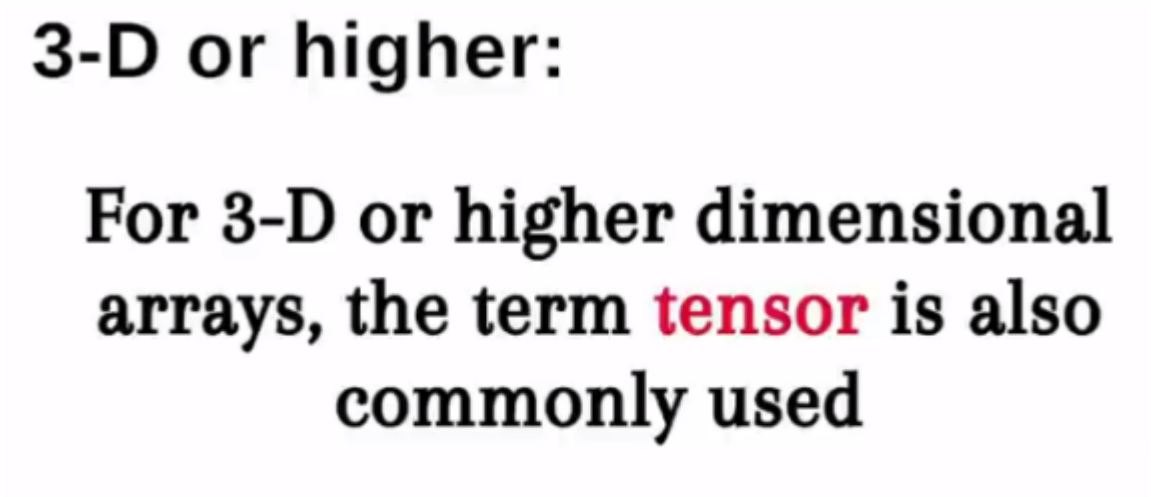
In [13]:
```
np.ones((7,8))
```

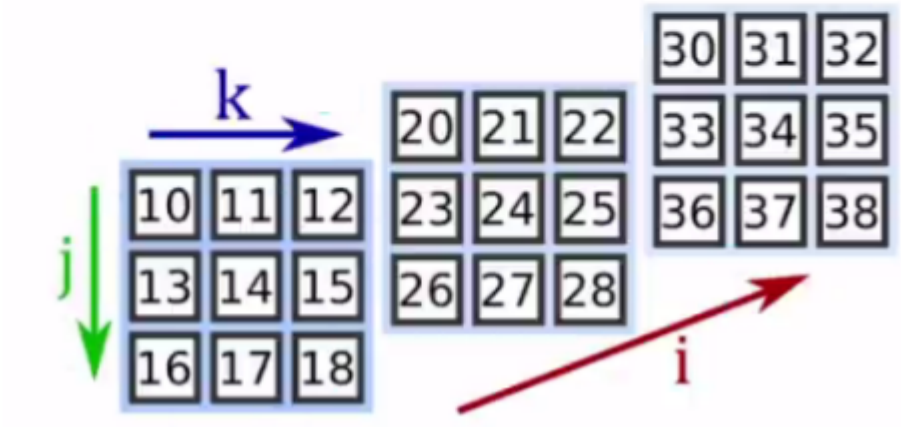
Out[13]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.]])
```

2-D with empty() Function

In [14]:
```
np.empty((3,4))
```

Out[14]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```





In [20]:
```
o = np.array([[[0, 1, 2, 3],
```

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

Out[20]:
```
array([[[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]],

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

In [17]:
```
s=np.arange(24).reshape(2,3,4)
s
```

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]]])
```

What is ndarray?

“ndarray,” is shorthand for “N-dimensional array.” An N-dimensional array is simply an array with any number of dimensions.

- Adding Elements in Array

In [2]:
```
import numpy as np
aa = np.array([99,3,4,34,42,23,53,34,23,44,56])
aa
```

Out[2]:
```
array([99,  3,  4, 34, 42, 23, 53, 34, 23, 44, 56])
```

USE numpy.append() to Add an Item To An Array

Use numpy.append(arr, values) to return a copy of arr with values added to the end

In [12]:
```
aa= np.append(aa,2)
aa
```

Out[12]:
```
array([99,  3,  4, 34, 42, 23, 53, 34, 23, 44, 56,  2,  2])
```

Use numpy.insert() to insert an elemnt to a specific indedex.

Use numpy.insert(arr, index, values) to return a copy of arr with values inserted at the index.

In [15]:
```
aa= np.insert(aa,2,990)
aa
```

Out[15]:
```
array([ 99,   3, 990,   4,  34,  42,  23,  53,  34,  23,  44,  56,   2,
         2])
```

- Sorting

In [5]:
```
import numpy as np
bb = np.array([89,45,53,34,3,3])
bb
```

Out[5]:
```
array([89, 45, 53, 34,  3,  3])
```

In [17]:
```
bb.sort()
#np.sort(bb)
```

In [18]:
```
bb
```

Out[18]:
```
array([ 3,  3, 34, 45, 53, 89])
```

- Concatenate

In [22]:
```
cc=np.concatenate((aa,bb))
cc
```

Out[22]:
```
array([99,  3,  4, 34, 42, 23, 53, 34, 23, 44, 56,  3,  3, 34, 45, 53, 89])
```

We can actually sort this array

In [24]:
```
cc.sort()
cc
```

Out[24]:
```
array([ 3,  3,  3,  4, 23, 23, 34, 34, 34, 42, 44, 45, 53, 53, 56, 89, 99])
```

Now with 2-D Array

Concatenation with Same Dimension

In [60]:
```
dd=np.array([[45,62,53,84],[94,63,32,61]])
```

In [ ]:
```
dd
```

Out[60]:
```
array([[45, 62, 53, 84],
       [94, 63, 32, 61]])
```

In [61]:
```
ee=np.array([[25,46,77,38],[55,44,33,22]])
ee
```

Out[61]:
```
array([[25, 46, 77, 38],
       [55, 44, 33, 22]])
```

In [62]:
```
np.concatenate((dd,ee),axis=1)
```

Out[62]:
```
array([[45, 62, 53, 84, 25, 46, 77, 38],
       [94, 63, 32, 61, 55, 44, 33, 22]])
```

In [63]:
```
np.concatenate((dd,ee),axis=0)
```

Out[63]:
```
array([[45, 62, 53, 84],
       [94, 63, 32, 61],
       [25, 46, 77, 38],
       [55, 44, 33, 22]])
```

Concatenation with Same Dimension

In [64]:
```
dd=np.array([[45,62,53,84],[94,63,32,61]])
dd
```

Out[64]:
```
array([[45, 62, 53, 84],
       [94, 63, 32, 61]])
```

In [66]:
```
ee=np.array([[25,46,77,38]])
ee
```

Out[66]:
```
array([[25, 46, 77, 38]])
```

In [67]:
```
np.concatenate((dd,ee),axis=0)
```

Out[67]:
```
array([[45, 62, 53, 84],
       [94, 63, 32, 61],
       [25, 46, 77, 38]])
```

- Shape and Size of an Array

- **ndarray.ndim** will tell you the number of axes, or dimensions, of the array.
- **ndarray.size** will tell you the total number of elements of the array. This is the product of the elements of the array's shape.
- **ndarray.shape** will display a tuple of integers that indicate the number of elements stored along each dimension of the array. If, for example, you have a 2-D array with 2 rows and 3 columns, the shape of your array is (2, 3).

In [74]:
```

```

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

Out[74]:
```
array([[[0, 1, 2, 3, 9],
        [4, 5, 6, 7, 2]],

       [[0, 1, 2, 3, 4],
        [4, 5, 6, 7, 6]],

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

Finding Dimension

In [75]:
```
ff.ndim
```

Out[75]:
```
3
```

Finding Size

In [76]:
```
ff.size
```

Out[76]:
```
30
```

Finding Shape

with the help of this function we can find total number of elements in array.

In [77]:
```
ff.shape
```

Out[77]:
```
(3, 2, 5)
```

Reshape an Array

You can use **reshape()** to reshape your array.

gg.reshape(4,3)

In above line **4** is number of **columns** and **3** is number of **rows**.

In [32]:
```
import numpy as np

gg = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

gg.reshape(4,3)
```

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

Out[32]:
```
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [8]:
```
gg.reshape(3,4)
```

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

In [9]:
```
gg.reshape(6,2)
```

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

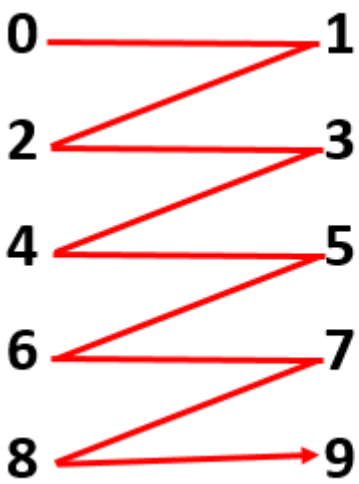
In [10]:
```
gg.reshape(2,6)
```

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

In [33]:
```
hh = np.arange(10)
hh
```

Out[33]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

=> C-like index ordering/Row major order

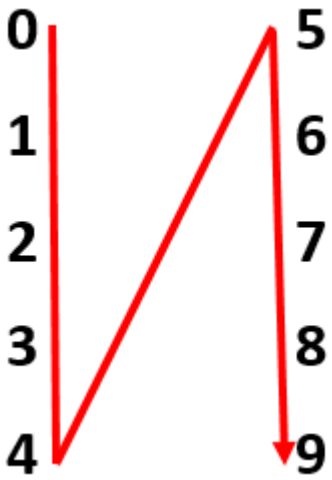


In [35]:
```
np.reshape(hh, newshape=(5,2), order='C')
```

Out[35]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
```

```
       [6, 7],
       [8, 9]])
```

=> Fortran-like index ordering



In [36]:
```
np.reshape(hh, newshape=(5,2), order='F')
```

Out[36]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

Convert a 1D array into a 2D array

You can use *np.newaxis* to add a new axis

In [37]:
```
jj=np.arange(10)
jj
```

Out[37]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [40]:
```
jj[np.newaxis, : ]
```

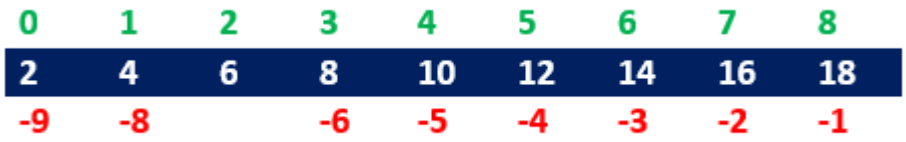
Out[40]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

In [41]:
```
jj[:,np.newaxis ]
```

Out[41]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
```

```
       [8],
       [9]])
```

Indexing and slicing



In [29]:
```
q=np.arange(2,20,2)
q
```

Out[29]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [44]:
```
q[6]
```

Out[44]:
```
14
```

In [46]:
```
# 3: ka mtlb hai k 3rd index se agy jtny bhi hain print kr do..
q[3:]
```

Out[46]:
```
array([ 8, 10, 12, 14, 16, 18])
```

In [49]:
```
q[4:8]
```

Out[49]:
```
array([10, 12, 14, 16])
```

In [60]:
```
q[-4]
```

Out[60]:
```
12
```

In [51]:
```
q[-4:]
```

Out[51]:
```
array([12, 14, 16, 18])
```

Some other Operations

- You can easily print all of the values in the array that are less than 14.

In [4]:
```
(q[q<14])
```

Out[4]:
```
array([ 2,  4,  6,  8, 10, 12])
```

- You can also select, for example, numbers that are equal to or greater than 8, and use that condition to index an array.

In [7]:
```
q1=(q[q>=8])
q1
```

Out[7]:
```
array([ 8, 10, 12, 14, 16, 18])
```

In [8]:
```
q1[0]
```

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

- You can select elements that are divisible by 3:

In [9]:
```
(q[q%3==0])
```

Out[9]:
```
array([ 6, 12, 18])
```

- you can select elements that satisfy two conditions using the & and | operators:

In [19]:
```
q[(q > 2) & (q < 16)]
```

Out[19]:
```
array([ 4,  6,  8, 10, 12, 14])
```

In [22]:
```
(q > 5) | (q == 15)
```

Out[22]:
```
array([False, False,  True,  True,  True,  True,  True,  True,  True])
```

In [35]:
```
 np.nonzero(q < 12)
```

Out[35]:
```
(array([0, 1, 2, 3, 4], dtype=int64),)
```

Create an Array from existing data

In [39]:
```
aa = np.arange(0,100,10)
aa
```

Out[39]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

- You can create a new array from a section of your array any time by specifying where you want to slice your array.

In [48]:
```
# Here, you grabbed a section of your array from index position 5 through index positio
a1=aa[5:9]
a1
```

Out[48]:
```
array([50, 60, 70, 80])
```

Vstack & Hstack

You can also stack two existing arrays, both vertically and horizontally.

Let's say you have two arrays, b1 and b2:

In [49]:
```
b1 = np.array([[1,2,3,4], [5,6,7,8]])
b1
```

Out[49]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [50]:
```
b2 = np.array([[4,3,2,1],[8,7,6,5]])
b2
```

Out[50]:
```
array([[4, 3, 2, 1],
       [8, 7, 6, 5]])
```

=> Vstack

vertically with vstack:

In [51]:
```
np.vstack((b1,b2))
```

Out[51]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8],
       [4, 3, 2, 1],
       [8, 7, 6, 5]])
```

=> Hstack

horizontally with hstack:

In [52]:
```
np.hstack((b1,b2))
```

Out[52]:
```
array([[1, 2, 3, 4, 4, 3, 2, 1],
       [5, 6, 7, 8, 8, 7, 6, 5]])
```

Split an Array

You can split an array into several smaller arrays using hsplit. You can specify

either the number of equally shaped arrays to return or the columns after which the division should occur.

In [73]:
```
c1 = np.arange(0, 30)#.reshape(2, 15)
c1
```

Out[73]:
```
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])
```

In [69]:
```
np.hsplit(c1, 6)
```

Out[69]:
```
[array([0, 1, 2, 3, 4]),
 array([5, 6, 7, 8, 9]),
 array([10, 11, 12, 13, 14]),
 array([15, 16, 17, 18, 19]),
 array([20, 21, 22, 23, 24]),
 array([25, 26, 27, 28, 29])]
```

In [70]:
```
c1.ndim
```

Out[70]:
```
1
```

In [77]:
```
c2=c1.reshape(2,15)
c2
```

Out[77]:
```
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]])
```

In [78]:
```
c2.ndim
```

Out[78]:
```
2
```

In [87]:
```
c1
```

Out[87]:
```
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])
```

=> If you wanted to split your array after the 4th and 5th column.

c1 aik 1-D array hia is lia pehly

- 4-idex/column ki aik array bni
- 5th index/column ki aik q k after 4th and 5th, is lia 5th ki alhdah bny gi.
- or aik arrya 5th index k baad waly elements ki bni hai

In [93]:
```
np.hsplit(c1, (4,5))
```

Out[93]:
```
[array([0, 1, 2, 3]),
 array([4]),
 array([ 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])]
```

Same operation with c2.

In [95]:
```
np.hsplit(c2, (6,8))
```

Out[95]:
```
[array([[ 0,  1,  2,  3,  4,  5],
        [15, 16, 17, 18, 19, 20]]),
 array([[ 6,  7],
        [21, 22]]),
 array([[ 8,  9, 10, 11, 12, 13, 14],
        [23, 24, 25, 26, 27, 28, 29]])]
```

Basic array operations

=> Addition

In [106…
```
c3=np.arange(50,56)

c3
```

Out[106…
```
array([50, 51, 52, 53, 54, 55])
```

In [101…
```
c4=np.arange(0,30,5)
c4
```

Out[101…
```
array([ 0,  5, 10, 15, 20, 25])
```

In [107…
```
c3+c4
```

Out[107…
```
array([50, 56, 62, 68, 74, 80])
```

=> Subtraction

In [109…
```
c3-c4
```

Out[109…
```
array([50, 46, 42, 38, 34, 30])
```

=> Multiplication

In [110…
```
c3*c4
```

Out[110…
```
array([   0,  255,  520,  795, 1080, 1375])
```

=> Division

In [113…
```
c3/c4
```

```
C:\Users\MUSHAR~1\AppData\Local\Temp/ipykernel_7064/2510859854.py:1: RuntimeWarning: div
```

```
ide by zero encountered in true_divide
  c3/c4
array([        inf, 10.2       ,  5.2       ,  3.53333333,  2.7       ,
        2.2       ])
```

=> Sum of Elements in Array

In [114…
```
c3.sum()
```

Out[114…
```
315
```

=> To add the rows or the columns in a 2D array,you would specify the axis

In [121…
```
s1 = np.arange(1,11).reshape(2,5)
s1
```

Out[121…
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [123…
```
# sum the rows with axis=0
s1.sum(axis=0)
```

Out[123…
```
array([ 7,  9, 11, 13, 15])
```

In [124…
```
# sum the rows with axis=0
s1.sum(axis=1)
```

Out[124…
```
array([15, 40])
```

=> Broadcasting

mzeed info k lia numpy search krna....

In [1]:
```
import numpy as np
s2 = np.array([1.0, 2.0])
s2 * 1.6
```

Out[1]:
```
array([1.6, 3.2])
```

Other useful array operations

In [7]:
```
s3 = np.arange(1.5,21.5)
s3
```

Out[7]:
```
array([ 1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5, 10.5, 11.5,
       12.5, 13.5, 14.5, 15.5, 16.5, 17.5, 18.5, 19.5, 20.5])
```

In [8]:
```
s3.size
```

Out[8]:
```
20
```

Maximum

In [9]:
```
s3.max()
```

Out[9]:
```
20.5
```

Minimum

In [10]:
```
s3.min()
```

Out[10]:
```
1.5
```

Sum

In [11]:
```
s3.sum()
```

Out[11]:
```
220.0
```

In [21]:
```
s4=s3.reshape(4,5)
s4
```

Out[21]:
```
array([[ 1.5,  2.5,  3.5,  4.5,  5.5],
       [ 6.5,  7.5,  8.5,  9.5, 10.5],
       [11.5, 12.5, 13.5, 14.5, 15.5],
       [16.5, 17.5, 18.5, 19.5, 20.5]])
```

In [22]:
```
s4.max()
```

Out[22]:
```
20.5
```

In [23]:
```
s4.min()
```

Out[23]:
```
1.5
```

Q: .....SMJH NHI AYEEE....

You can specify on which axis you want the aggregation function to be computed.

For example, you can find the minimum value within each column by specifying axis=0.

In [24]:
```
s4.min(axis=0)
```

Out[24]:
```
array([1.5, 2.5, 3.5, 4.5, 5.5])
```

The four values listed above correspond to the number of columns in your array.

With a four-column array, you will get four values as your result.

In [35]:
```
s4[2,4]
```

Out[35]:
```
15.5
```

In [33]:
```
s4[0:3]
```

Out[33]:
```
array([[ 1.5,  2.5,  3.5,  4.5,  5.5],
       [ 6.5,  7.5,  8.5,  9.5, 10.5],
       [11.5, 12.5, 13.5, 14.5, 15.5]])
```

In [38]:
```
s4[0:3,4]
```

Out[38]:
```
array([ 5.5, 10.5, 15.5])
```

Creating Matrices

We can pass Python lists of lists to create a 2-D array (or "matrix") to represent them in NumPy.

In [19]:
```
import numpy as np
s5=np.array([[1,2],[3,4]])
s5
```

Out[19]:
```
array([[1, 2],
       [3, 4]])
```

In [23]:
```
s5[0,1]
```

Out[23]:
```
2
```

In [32]:
```
s5[0:2]
```

Out[32]:
```
array([[1, 2],
       [3, 4]])
```