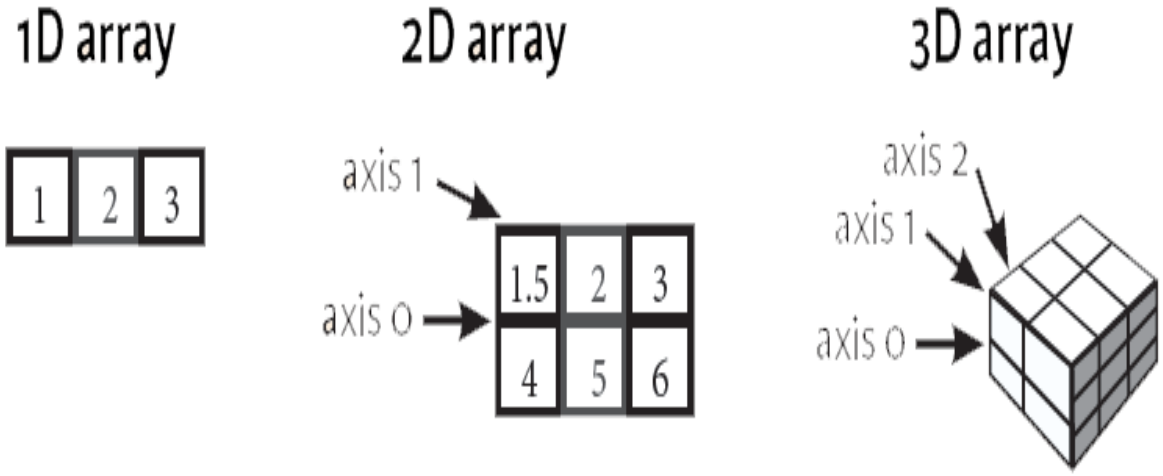
# Python Numpy

**Numpy is Python library that useful for scientific computing applications that is acronym for "Numeric Python" A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.**

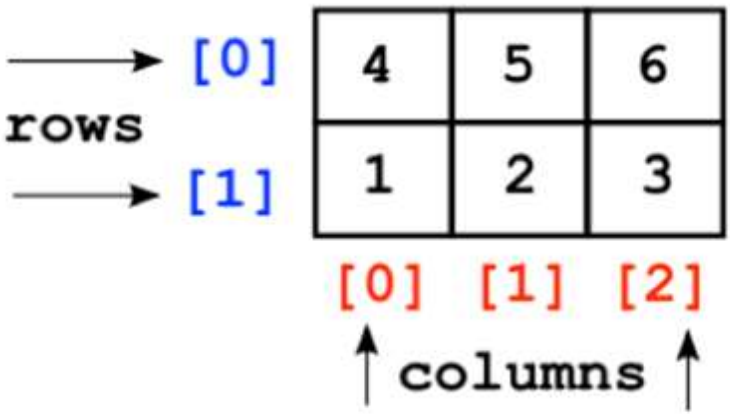
**We can initialize numpy arrays from nested Python lists, and access elements using square brackets:**

**import numpy as np**

- Numpy array is organized in 1-D, 2-D and n-D array.
-




-

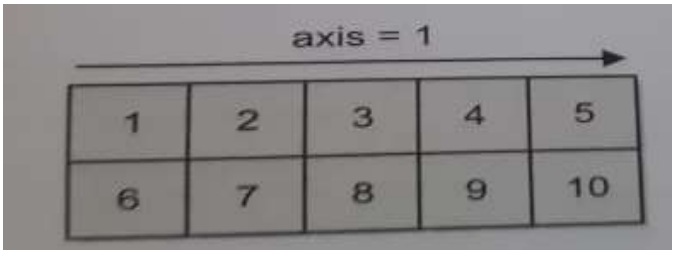


# Anatomy of Numpy Array

1. **Axes:** in Numpy axes refers to its dimension. There are 1-D, 2-d and Multi-Dimensional array in Numpy. The values of axes may be 0, 1 and 3.

- Axes 0 is first axes of array and it display along the rows.


- Axes 1 is second axes of array and it display along the columns.

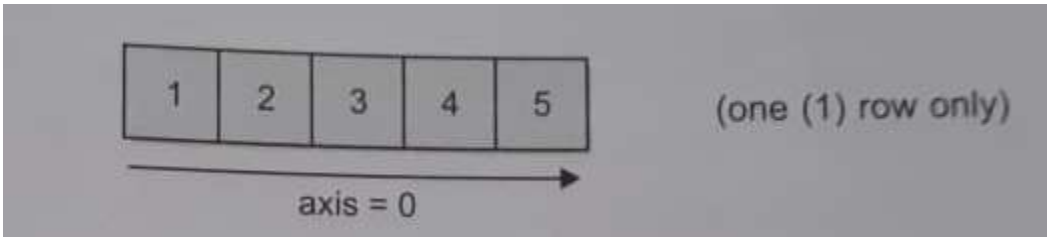



- As we add new dimension, the new axes used in array.

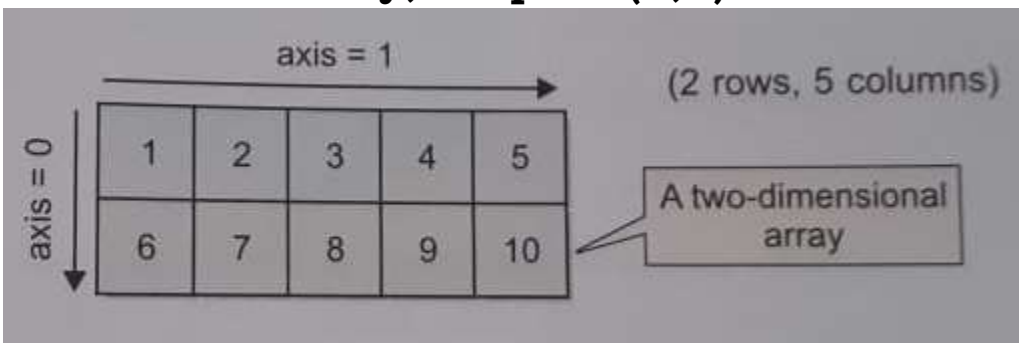
2. **Rank:** The numbers of axes in n-D Array is refers to its rank.
3. **Shape:** The shape of an n-D array tells about the number of element along with each axis of it. In other words, the shape of an array is a tuple of integers giving the size of the array along each dimension. The shape can be find by using following syntax.

   **Arrayname.shape**

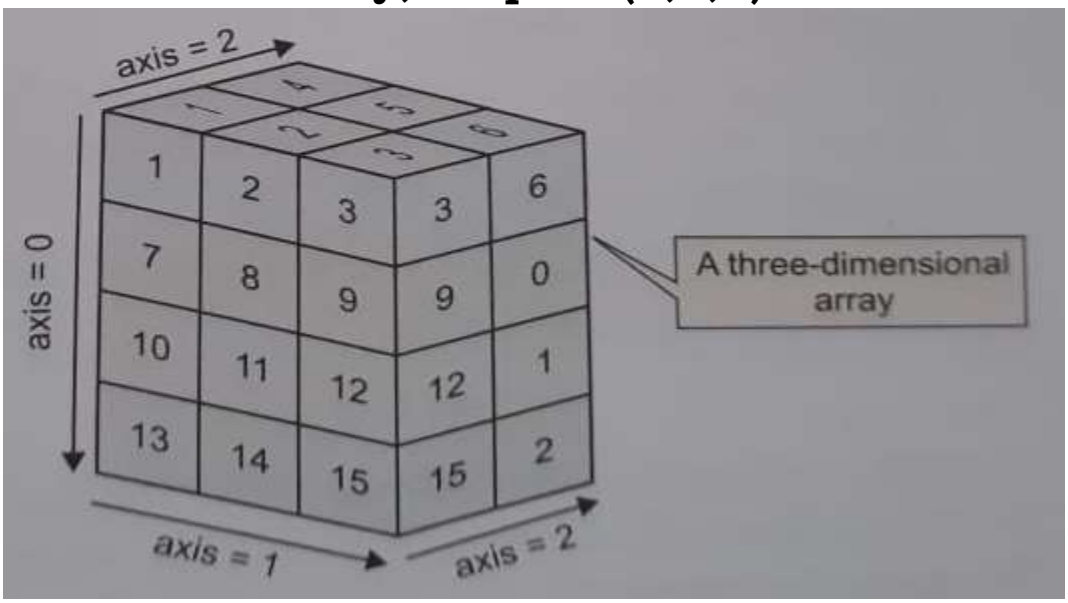
   - For 1-D array, shape is (5,)



- **For 2-D Array, shape is (2,5)**




- **For 3-D array, shape is (4,3,2)**




4. **Datatype: (dType) Numpy array stores homogeneous type of values that mean same type of values. By default the numpy creates float type array but we can create other numeric type array also. The type() function used to find the datatype of array.**

5. **ItemSize: It specify the size of each element in bytes. As we know that the numpy array stores same type of values so the size of each element. The shape can be find by using following syntax.**
   **Arrayname.itemsize**

**Example-1:**

```
import numpy as np
Ar1=np.array([2,4,6,8])      # 1-D array
print("Array-1\n",Ar1)
Ar2=np.array([[2,4,1,3],[5,7,6,9]])      # 2-D array
print("Array-2\n",Ar2)
print("Data type of Array-1: ",type(Ar1))
print("Data type of Array-2: ",type(Ar2))
print("Shape of Array-1: ",Ar1.shape)
print("Shape of Array-2: ",Ar2.shape)
print("ItemSize of Array1: ",Ar1.itemsize)
print("ItemSize of Array-2: ",Ar2.itemsize)
```

**Output:**

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

Data type of Array-1:  <class 'numpy.ndarray'>
Data type of Array-2:  <class 'numpy.ndarray'>
Shape of Array-1:  (4,)
Shape of Array-2:  (2, 4)
ItemSize of Array1:  4
ItemSize of Array-2:  4
```

# Numpy Array Vs Python List

♦ **Similarity**

- **Both numpy array and python list follow the same pattern of indexing. Index may be in positive (Forward) and in negative (Backward) format.**
- **Example: Arr = numpy.array([10,20,30,40])**

| | | | | |
|---|---|---|---|---|
| **Forward Indexing** | **0** | **1** | **2** | **3** |
| **Items** | **10** | **20** | **30** | **40** |
| **Backward Indexing** | **-4** | **-3** | **-2** | **-1** |

♦ **Difference**

- **List is mutable and Numpy array is immutable. The size of array cannot change once created. Can't support addition / removal of element**
- **The values of array can be changed with same type of existing type of values.**
- **List contains different type of values but numpy array can hold only one type of numeric values.**
- **We can create Jagged Array (list of Lists or nD list ) in python. But multi-dimension slicing is not possible in list. (Multi-dimension slicing means accessing a particular row or column or both). We can access a particular row in nd list, but accessing particular column is not possible.**
- **We can create an nd numpy array in python, and perform multidimensional slicing.**
- **An equivalent numpy array occupies much less space than Python list.**
- **Numpy array is faster than list.**
- **Python Array support Vectorized Operation, mean one operation can apply on all elements of array in one operation.**

**Example**

```
Ar1= numpy.array([2,4,6,8])
Ar1=Ar1+100
print("After Vectorization Array\n",Ar1)
lst=[2,4,6,8]
lst=lst+100
print("After Vectorization List\n",lst)
```

**Output:**

```
After Vectorization Array
 [102 104 106  108]
TypeError: lst=lst+100 , can only concatenate list (not "int") to list
```

## Creation of numpy array

**The array( ) function of numpy library used to create numpy array.**

**Syntax:**

**import numpy or import numpy as np**

**Array_name = np.array(array_convertable_object, dtype)**

**dtype is optional and default is float**

**array_convertable_object, it may be any sequence of elements that can convert into array.**

**List, tuple can used to convert into array. Set, String and dictionary create problem with array( ) to convert into array.**

## Create Array using fromstring( ) method.

**The fromstring() function is used to create a new 1-D array initialized from raw binary or text data in a string.**

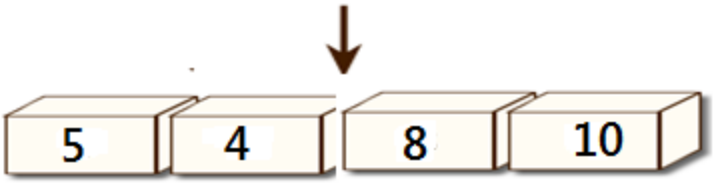
**Syntax:**

```
numpy.fromstring(string, dtype=float, count=-1, sep='')
```

| Name | Description | Required/ Optional |
|---|---|---|
| string | A string containing the data. | Required |
| Dtype | The data type of the array; default: float. For binary input data, the data must be in exactly this format | Optional |
| Count | Read this number of dtype elements from the data. If this is negative (the default), the count will be determined from the length of the data. | Optional |
| Sep | The string separating numbers in the data; extra whitespace between elements is also ignored | Optional |

```
import numpy as np
arr=np.fromstring("5  4  8  10", dtype=int, sep=" ")
```

```
np . fromstring (" 5  4  8  10", dtype = int, sep = "  "
```




## Create Array using fromiter( ) method.

**To create n-D array from all type of sequences (number, string, dictionary), the fromiter( ) can be used. It supports both numeric and non-numeric sequences but in general it is used with non-numeric sequences. The array ( ) work perfectly with numeric sequences)**

**Syntax:**

**numpy.fromiter ( iterable_sequence, datatype, count )**

**where,**

**iterable_sequence may be dictionary, string.**

**Datatype is mandatory and it should be int or Unicode type.**

**Count is the number of elements in array. If skipped, then it will read all elements.**

**Note: Array can create by using keys / Values of dictionary.**

**Example:** **Create array by using keys of dictionary.**

```
Import numpy as np
dict={100:"abc", 200:"pqr", 300:"xyz"}
ar=np.fromiter(dict.keys(), dtype=np.int32)
#ar=np.fromiter(dict, dtype=np.int32) It also works
print("Array elements: ",ar)
print("Shape of ar: ",ar.shape)
print("Size of ar: ",ar.size)
```

**Output:**

```
Array elements:  [200 300 100]
Shape of ar:  (3,)
Size of ar:  3
```

**Example:** **Create array by using values of dictionary.**

```
dict={100:"abc",200:"pqr",300:"xyz"}
ar=np.fromiter(dict.values(),dtype='U2')
print("Valuess of dict: ",ar)
print("Size of ar: ",ar.size)
print("Shape of ar: ",ar.shape)
```

**Output:**

```
Values of dict:  ['pq' 'xy' 'ab']
Size of ar:  3
Shape of ar:  (3,)
```

**Example:** **Create array by using string.**

```
import numpy as np
str="KV JJN"
ar=np.fromiter(str,dtype='U1', count=4)
print("Elements of Array: ",ar)
print("Size of ar: ",ar.size)
print("Shape of ar: ",ar.shape)
```

**Output:**

```
Elements of Array:  ['K' 'V' ' ' 'J']
Size of ar:  4
```

**Shape of ar: (4,)**

## Various Methods to create Arrays:

1. **empty():**

**numpy.empty(shape,[dtype=<datatype>], order='C' or'F']**
**C mean , arrangement of data as row wise.**
**F mean, arrangement of data as column wise.**
**Default datatype is float.**

```
import numpy as np

arr=np.empty([2,4],dtype=int)
print(arr)
output: [[123  6454  3541  5632]
         [2314  325  6578  415789]]

arr=np.empty([2,4],dtype=np.int64)
print(arr)
output: [[0  0  0  0]
         [0  0  0  0]]

arr=np.empty([2,4])
print(arr)
output: [[12.3  64.54  3.541  56.32]
         [2.314  3.25  65.78  415.789]]
```

## 2.zeros():

**numpy.zeros(shape,[dtype=<datatype>], order='C' or'F']**
**C mean , arrangement of data as row wise.**
**F mean, arrangement of data as column wise.**
**Default datatype is float.**

```
import numpy as np
arr=np.zeros(5)
print(arr)
output: [0 0 0 0 0]
```

```
arr=np.zeros([2,4],dtype=np.int64)
print(arr)
output: [[0 0 0 0]
         [0 0 0 0]]

arr=np.zeros([2,4])
print(arr)
output: [[0. 0. 0. 0.]
         [0. 0. 0. 0.]]
```

## 3. ones():

```
import numpy as np
arr=np.ones(5, dtype=np.int64)
print(arr)
output: [1 1 1 1 1]

arr=np.ones([2,4],dtype=np.int64)
print(arr)
output: [[1 1 1 1]
         [1 1 1 1]]

arr=np.ones(5)
print(arr)
output: [1. 1. 1. 1. 1.]
```

## 4. full():

Create n-D array with constant value

```
Import numpy as np
Arr=np.full([4,3],8)
Print(arr)
Output:
[[8 8 8]
[8 8 8]
[8 8 8]
[8 8 8]]
```

## 5. arange():

create array from a range.

(Start / First/ Lower limit Inclusive)
(Stop / End/ Upper limit Exclusive)

**Numpy.arange(start, stop, step, dtype)**

```
Import numpy as np
Arr=np.arange(2,9)
Print(Arr)
Output:  [2 3 4 5 6 7 8]

Arr=np.arane(6)
Print(Arr)
Output: [0 1 2 3 4 5]

Arr=np.arane(1,9,2 dtype=np.float32)
Print(Arr)
Output:  [1. 3. 5. 7.]

Arr=np.arane(1,2,0.2)
Print(Arr)
Output:  [1.0  1.2  1.4  1.6  1.8]
```

## 6. linspace():

Create array from a range. Create equally separated elements from start to end.
(Both limits inclusive)

**Numpy.linspace(start,stop,values, dtype)**

```
Arr=np.linspace(2,3,6)
Print(Arr)
Output:
[2. 2.2 2.4 2.6 2.8 3.]
```

## 7. copy():

It is used to create new copy of an existing array and new array store at new location.

```
Import numpy as np
```

**Arr1=np.array([2,4,6,8])**
**Arr2=np.copy(Arr1) # Use separate memory**
**Arr3=Arr1  # Arr1 and Arr3 will share common memory**
**Print(Arr1)  Output: [2 4 6 8]**
**Print(Arr2)  Output: [2 4 6 8]**
**Print(Arr3)  Output: [2 4 6 8]**

**Arr1[0]=100  # will work**
**Arr[4]=100 # will not work.**

**Print(Arr1)  Output: [100 4 6 8]**
**Print(Arr2)  Output: [2 4 6 8]**
**Print(Arr3)  Output: [100 4 6 8]**

## 8. reshape():

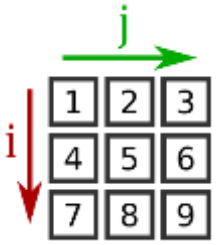
**Create 2-D array from 1-D array.**
**Import numpy as np**

**Arr1=np.array([2,4,6,8,10,12])**
**Arr2=np.reshape(Arr1,[3,2])**
**Print(Arr1)output: [2 4 6 8 10 12]**
**Print(Arr2)**
**Output:**
**[[2 4]**
**[6 8]**
**[10 12]]**
**Note: Number of elements should be as per dimension.**

9. **Slicing of Array elements:**

- **1-D array slicing: as same list slicing**
- **2-D array slicing: as same list slicing but with 2 slice patterns, first for row slicing and second for column slicing.**

```
import numpy as np
a2 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```




**We can index an element of the array using two indices,**
**i selects the row, and j selects the column**

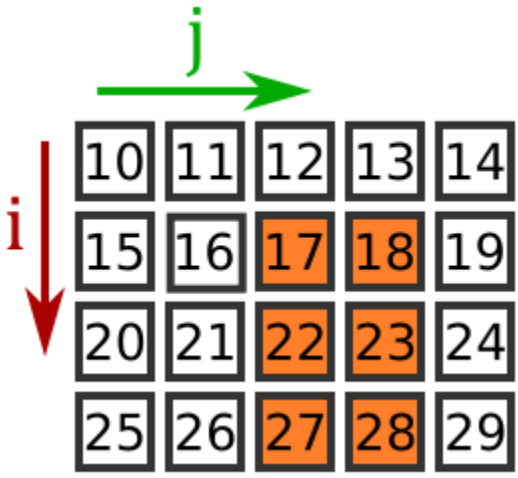
```
print(a2[2, 1])      output: 8
print(a2[2][1])      output: 8

print(a2[:, 1])              output:  [2, 5, 8]
```

```
import numpy as np

a2 = np.array([[10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19],
               [20, 21, 22, 23, 24],
               [25, 26, 27, 28, 29]])

print(a2[1:,2:4])  # [[17 18]
                   #  [22 23]
                   #  [27 28]]
```



# Joining or Concatenating Numpy Array

**To join two or more existing arrays, numpy provides following functions.**

1. **hstack( )**
2. **vstack( )**
3. **concatenate( )**

**hstack( ) function: This Function used to join two or more arrays horizontally. The number of rows in both arrays should be same. That mean all the input arrays must have same number of row dimensions but column dimension may be different in all arrays.**

**Example: Join Three 1-d arrays**

```
import numpy as np
Ar1=np.array([1,2,3,4])      # 4- Columns
Ar2=np.array([5,6,7])        # 3 Columns
Ar3=np.array([0,0)           # 2 Columns
Ar4=np.hstack((Ar1,Ar2,Ar3)) # hstack requires only one parameter.
print("Ar1: ",Ar1)
print("Ar2: ",Ar2)
print("Ar3: ",Ar3)
print("hstack() Ar4: ",Ar4)
```

**Output:**

```
Ar1:  [1 2 3 4]
Ar2:  [5 6 7 8]
Ar3:  [0 0 0 0]
hstack() Ar4:  [1 2 3 4 5 6 7 0 0]
```

**Example: Join Two 2-D arrays.**

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,5],[7,8]])
c=np.hstack((a,b))
print("A=\n",a)
print("B=\n",b)
print("C=\n",c)
```

**Output:**

```
A=
 [[1 2]
 [3 4]]
B=
 [[5 5]
```

[7 8]]
C=
[[1 2 5 5]
[3 4 7 8]]

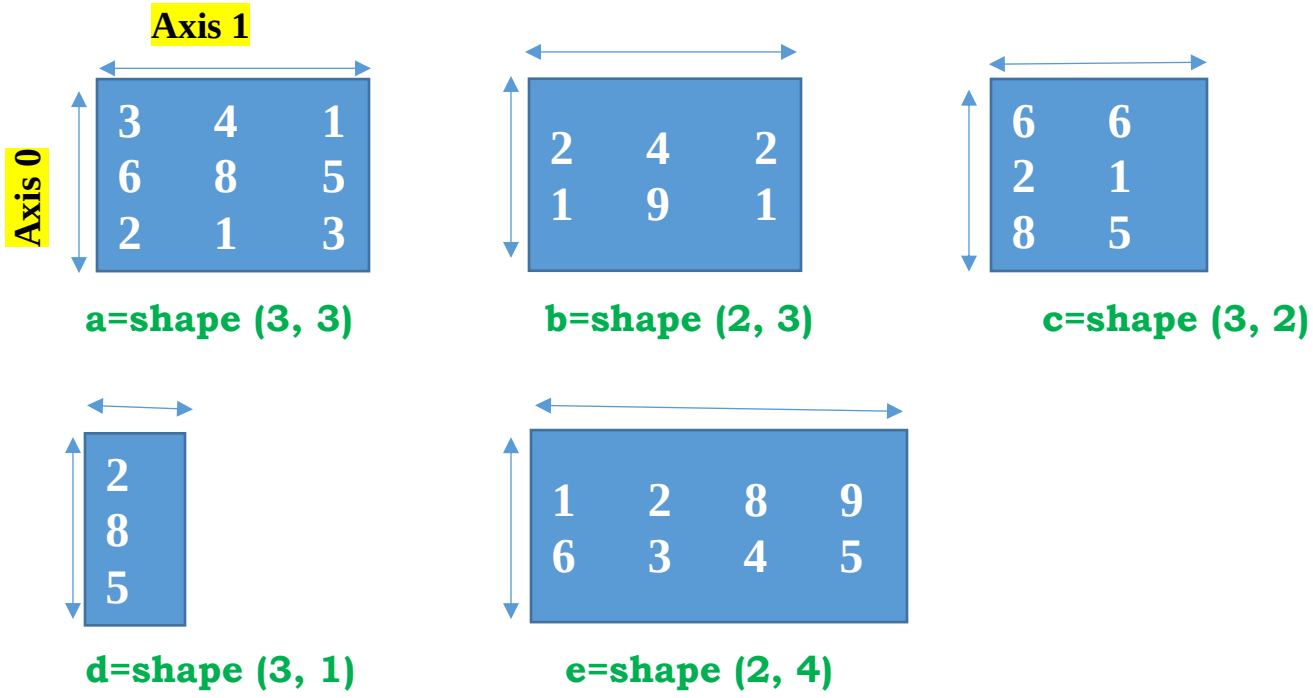
**vstack( ) function:** This Function used to join two or more arrays vertically. The number of columns in both arrays should be same. That mean all the input arrays must have same number of column dimensions

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,5],[7,8]])
c=np.vstack((a,b))          #vstack( ) requires one parameter only.
print("A=\n",a)
print("B=\n",b)
print("C=\n",c)
```

**output**

```
A=
      [[1 2]
      [3 4]]
B=
      [[5 5]
      [7 8]]
C=
      [[1 2]
      [3 4]
      [5 5]
     [7 8]]
```

**Concatenate ( ) function:** numpy provides concatenate( ) that used to join or concatenate two or more arrays on basis of their axis.




- Axis=0 ( or None) mean join arrays along with row
- Axis=1 mean join arrays along with Column
- When axis is 0, then column dimensions should be matched.
- When axis is 1, then row dimensions should be matched.
- If row dimensions are same then both array can joined on axis=1
- If column dimensions are same then both array can joined on axis=0

**Example:**

```
import numpy as np
a=np.array([[3,4,1],[6,8,5],[2,1,3]])          # Shape (3,3)
b=np.array([[2,4,2],[1,9,1]])                  # Shape (2,3)
c=np.array([[6,1],[2,1],[8,5]])                # Shape (3,2)
d=np.shape([3,1])                              # Shape (1,2)
d=np.array([[2],[8],[5]])
e=np.array([[1,2,8,9],[6,3,4,5]])              # Shape (2,4)
print("A=\n",a)
print("B=\n",b)
print("C=\n",c)
print("D=\n",d)
print("E=\n",e)
```

**Case-1: concatenate array “a” and “b”.**

**Array – a and b can only join as axis=0. Because No. of columns are equal.**

```
ab=np.concatenate((a,b),axis=0)
print("Concatenate a,b=\n",ab)
```

**Output:**

```
Concatenate a,b=
 [[3 4 1]
 [6 8 5]
 [2 1 3]
 [2 4 2]
 [1 9 1]]
```

**Think About it: is it possible to concatenate array “a” with others along with axis=0?**

**Case-2: concatenate array “a” and “c”.**

**Array – a and c can only join as axis=1. Because No. of rows are equal.**

```
ac=np.concatenate((a,c),axis=1)
print("Concatenate a,c=\n",ac)
```

**Output:**

```
Concatenate a,c=
 [[3 4 1 6 1]
 [6 8 5 2 1]
 [2 1 3 8 5]]
```

**Case-3: concatenate array “a” and “d”.**

**Array – a and d can only join as axis=1. Because No. of rows are equal.**

```
ad=np.concatenate((a,d),axis=1)
print("Concatenate a,d=\n",ad)
```

**Output:**

```
Concatenate a,d=
 [[3 4 1 2]
 [6 8 5 8]
 [2 1 3 5]]
```

**Case-4: concatenate array “b” and “e”.**

**Array – b and e can only join as axis=1. Because No. of rows are equal.**

```
be=np.concatenate((b,e),axis=1)
print("Concatenate b,e=\n",be)
```

**Output:**

```
Concatenate b,e=
 [[2 4 2 1 2 8 9]
 [1 9 1 6 3 4 5]]
```

## Splitting Array (Obtaining Subset of Array)

**Subset of n-D array is set of extracted elements from main array. A subset can be create by using the array slicing. Python provides some built in functions to create sub set.**

1. **split( )**
2. **hsplit( )**
3. **vsplit( )**

**hsplit( ) function:**

**This function extract the subset of numpy array after splitting it horizontally.**

**Syntax:**

**numpy.hspit(array,n)**

**Where "n" is number of subsets to be created. The value of n should be chosen in such a way that array columns should be divided in equal parts.**

**Ex: a=np.arange(24).reshape(4,6)  # 6 columns can split in 'n' sets**

**It will create the array as per given below shape.**

**Horizontal axis=1: hsplit( )**

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 |

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

```
Print(np.hsplit(a,3))

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

**vsplit( ) function:**

**This function extract the subset of numpy array after splitting it vertically.**

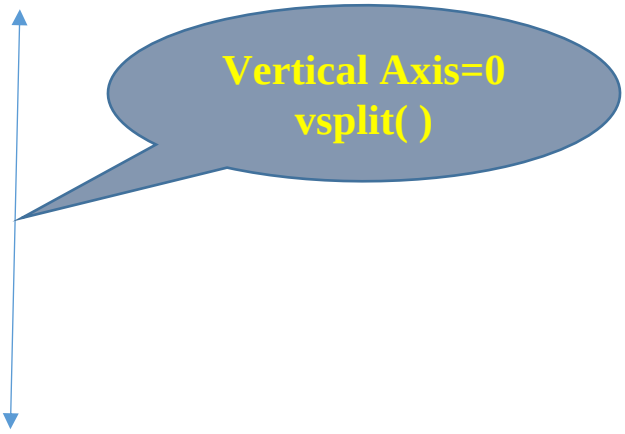
**Syntax:**

**numpy.vspit(array,n)**

**Where "n" is number of subsets to be created. The value of n should be chosen in such a way that array should be divided in equal parts.**

**Ex: a=np.arange(36).reshape(6,6)  # 6 rows can split in 'n' sets**

| | | | | | |
|---|---|---|---|---|---|
| 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 |




```
print(np.vsplit(a,2))

array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17]])
array([[18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

```
print(np.vsplit(a,2))

array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]]),
array([[24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

## split( ) function:

**split( ) function is general function that used to split the array on both horizontally and vertically. It can performed by using the axis. Axis=0 for horizontal division and axis=1 for vertical division. It can also divide array into equal or non-equal sub arrays.**

**Syntax:**
**numpy.split(array, n|[n,m], axis=0|1)**
**where**
**n is number of subsets for 1-D array**
**[n,m] are start,end index for 2-D array (in sorted order)**
**Axis=0, mean subset create based on rows**
**Axis=1, mean subset create based on columns**
**# if no axis given in 2-D then it will create based on vertical axis(Row wise).**
**Example:**

```
import numpy as np
a=np.arange(6)
print(a)
print("---np.split(a,2,axis=0)---\n",np.split(a,2,axis=0))
```

**Output:** [array([0, 1, 2]), array([3, 4, 5])]

```
#print("---np.split(a,2,axis=1)---\n",np.split(a,2,axis=1))
```

**Output:** IndexError: tuple index out of range

**---np.split(a,[4,5,6,7],axis=0)---**
**Output:** [array([0, 1, 2, 3]), array([4]), array([5]), array([], dtype=int32), array([], dtype=int32)]
**---np.split(a,[2,3,4,5],axis=0)---**
**Output:** [array([0, 1]), array([2]), array([3]), array([4]), array([5])]
**---np.split(a,[2,3],axis=0)---**
**Output:** [array([0, 1]), array([2]), array([3, 4, 5])]
**---np.split(a,[3,3],axis=0)---**
**Output:** [array([0, 1, 2]), array([], dtype=int32), array([3, 4, 5])]
**---np.split(a,[3,2],axis=0)---**
**Output:** [array([0, 1, 2]), array([], dtype=int32), array([2, 3, 4, 5])]
**---np.split(a,[3,1],axis=0)---**
**Output:** [array([0, 1, 2]), array([], dtype=int32), array([1, 2, 3, 4, 5])]

**Example:**

```
a=np.arange(36).reshape(6,6)
```

**If indices are in sorted order like [2,4], then display first 2 rows (index 0,1) in first set. Second Set starts from index 2 and display up to index 3 (less than 4). If there is Third set available then it will work like second set. Rest of rows display in last set.**
**If indices are in un-sorted order like [3,2], then display first 3 rows (index 0,1,2) in first set. Second Set starts from index 2 and goes up to either "next set" or "up to end of rows".**

```
import numpy as np
a=np.arange(36).reshape(6,6)
print(a)
Output: [[ 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]]
---np.split(a,[4,5,6,7],axis=0)---
 Output: [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]]), array([[24, 25, 26, 27, 28, 29]]), array([[30, 31,  32,  33,  34,  35]]),  array([],  shape=(0,  6),  dtype=int32),  array([], shape=(0, 6), dtype=int32)]
---np.split(a,[2,3,4,5],axis=0)---
 Output: [array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]]), array([[12, 13, 14, 15, 16, 17]]), array([[18, 19, 20, 21, 22, 23]]), array([[24, 25, 26, 27, 28, 29]]), array([[30, 31, 32, 33, 34, 35]])]
---np.split(a,[2,4],axis=0)---
 Output: [array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]]), array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]]), array([[24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
---np.split(a,[3,3],axis=0)---
Output:  [array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12,  13,  14,  15,  16,  17]]),  array([],  shape=(0,  6),  dtype=int32), array([[18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
---np.split(a,[3,2],axis=0)---
 Output: [array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12,  13,  14,  15,  16,  17]]),  array([],  shape=(0,  6),  dtype=int32), array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
Output: ---np.split(a,[3,1,2],axis=0)---
 [array([[ 0,  1,  2,  3,  4,  5],
```

[ 6, 7, 8, 9, 10, 11],
[12, 13, 14, 15, 16, 17]]), array([], shape=(0, 6), dtype=int32), array([[ 6, 7, 8, 9, 10, 11]]), array([[12, 13, 14, 15, 16, 17],
[18, 19, 20, 21, 22, 23],
[24, 25, 26, 27, 28, 29],
[30, 31, 32, 33, 34, 35]])]

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```
import numpy as np
a=np.arange(36).reshape(6,6)
print(a)
```

**Output:** [[ 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]]

---np.split(a,[4,5,6,7],axis=1)---

**Output:** [array([[ 0, 1, 2, 3],
[ 6, 7, 8, 9],
[12, 13, 14, 15],
[18, 19, 20, 21],
[24, 25, 26, 27],
[30, 31, 32, 33]]), array([[ 4],
[10],
[16],
[22],
[28],
[34]]), array([[ 5],
[11],
[17],
[23],
[29],
[35]]), array([], shape=(6, 0), dtype=int32), array([], shape=(6, 0), dtype=int32)]

---np.split(a,[2,3,4,5],axis=1)---

**Output:** [array([[ 0, 1],
[ 6, 7],
[12, 13],
[18, 19],
[24, 25],
[30, 31]]), array([[ 2],
[ 8],
[14],
[20],

[26],
[32]]), array([[ 3],
[ 9],
[15],
[21],
[27],
[33]]), array([[ 4],
[10],
[16],
[22],
[28],
[34]]), array([[ 5],
[11],
[17],
[23],
[29],
[35]])]
---np.split(a,[2,4],axis=1)---
Output: [array([[ 0, 1],
[ 6, 7],
[12, 13],
[18, 19],
[24, 25],
[30, 31]]), array([[ 2, 3],
[ 8, 9],
[14, 15],
[20, 21],
[26, 27],
[32, 33]]), array([[ 4, 5],
[10, 11],
[16, 17],
[22, 23],
[28, 29],
[34, 35]])]
---np.split(a,[3,3],axis=1)---
Output: [array([[ 0, 1, 2],
[ 6, 7, 8],
[12, 13, 14],
[18, 19, 20],
[24, 25, 26],
[30, 31, 32]]), array([], shape=(6, 0), dtype=int32), array([[ 3, 4, 5],
[ 9, 10, 11],
[15, 16, 17],
[21, 22, 23],

```
      [27, 28, 29],
      [33, 34, 35]])]
---np.split(a,[3,2],axis=1)---
 Output: [array([[ 0,  1,  2],
      [ 6,  7,  8],
      [12, 13, 14],
      [18, 19, 20],
      [24, 25, 26],
      [30, 31, 32]]), array([], shape=(6, 0), dtype=int32), array([[ 2,  3,  4,
5],
      [ 8,  9, 10, 11],
      [14, 15, 16, 17],
      [20, 21, 22, 23],
      [26, 27, 28, 29],
      [32, 33, 34, 35]])]
---np.split(a,[3,1,2],axis=1)---
 Output: [array([[ 0,  1,  2],
      [ 6,  7,  8],
      [12, 13, 14],
      [18, 19, 20],
      [24, 25, 26],
      [30, 31, 32]]), array([], shape=(6, 0), dtype=int32), array([[ 1],
      [ 7],
      [13],
      [19],
      [25],
      [31]]), array([[ 2,  3,  4,  5],
      [ 8,  9, 10, 11],
      [14, 15, 16, 17],
      [20, 21, 22, 23],
      [26, 27, 28, 29],
      [32, 33, 34, 35]])]
>>>
```

## Arithmetic Operation on 2-D Array (Vector Operation)

**Numpy provides a variety of arithmetical operations to perform on array. There are two ways for calculations.**

### 1. Using Operators

```
a=np.array([1,3,5,7,4,6,8])
print("A=",a)
print("A+2=",a+2)
print("A-2=",a-2)
print("A*2=",a*2)
print("A/2=",a/2)
print("A%2=",a%2)
print("A//2=",a//2)
print("A**2=",a**2)
print("A+A=",a+a)
print("A//A=",a//a)
```

```
A=      [1 3 5 7 4 6 8]
A+2=  [ 3  5  7  9  6  8 10]
A-2=   [-1  1  3  5  2  4  6]
A*2=   [ 2  6 10 14  8 12 16]
A/2=    [0.5 1.5 2.5 3.5 2.  3.  4. ]
A%2= [1 1 1 1 0 0 0]
A//2=   [0 1 2 3 2 3 4]
A**2=  [1  9 25 49 16 36 64]
A+A=   [ 2  6 10 14  8 12 16]
A//A=   [1 1 1 1 1 1 1]
```

### 2. Using Numpy Functions

```
a=np.array([1,3,5,7,4])
print("A=",a)
print("Addition:",np.add(a,2))
print("Subtraction:",np.subtract(a,2))
print("Division:",np.divide(a,2))
print("Multiplication:",np.multiply(a,2))
print("Modulus:",np.mod(a,2))
print("Reminder:",np.remainder(a,2))
print("Add 2 Array:",np.add(a,a))
```

```
A= [1 3 5 7 4]
Addition: [3 5 7 9 6]
Subtraction: [-1  1  3  5  2]
Division: [0.5 1.5 2.5 3.5 2. ]
Multiplication: [ 2  6 10 14  8]
Modulus: [1 1 1 1 0]
Reminder: [1 1 1 1 0]
```

```
a=np.array([2,4,6])
print("A      = ",a)          # Output: [2 4 6]
print("A*2    = ",a*2)        # Output: [4 8 12]
print("A      = ",a)          # Output: ..... ?
```

****Finish****