Numpy

Numpy and pandas are data science libraries or api(application programming interface).

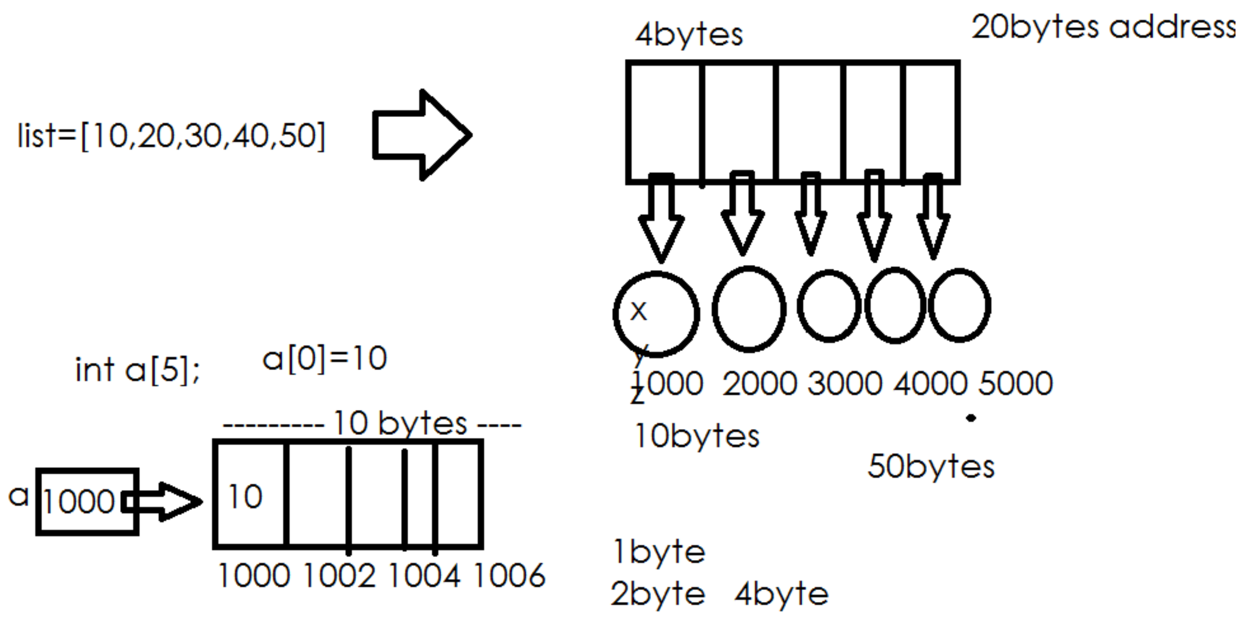
What is numpy?
Numpy stands for Numerical Python
Numpy is python library used for working with arrays (Vectors and Matrices).
Numpy provide functions to work with linear algebra, fourier transform and matrices.
Numpy is a python library is partially written in python and major part is wrriten in **C language.**

Array is collection of similar type data or elements.

Q: What is difference between Array and List?

List	**Array**
List is a collection of heterogeneous data elements.	Array is collection of homogeneous data elements.
List does not have axis or dimensions.	Array is having axis or dimensions
In List data will stored as objects.	In array data will stored as scalar value.
List is dynamic in size.	Array is fixed in size.
Because of list store data as objects it occupy more space.	Because of array store data as scalar values it occupy less space.
It is not efficient to process large sets of data.	It is efficient to process large set of data.
List hold only objects types.	

Array holds scalar types and objects types.



pip install numpy

(OR)

Download and install anaconda distribution

Anaconda is a python distribution which provides,

1. **Python softwrare**
2. **Data Science and ML Libraries**
    a. **Numpy**
    b. **Pandas**
    c. **Matplotlib**
    d. **Scipy**
    e. **TersonFlow**
    f. **ScikitLearn**
3. **IDE’s (Jupiter, Spyder)**

https://www.anaconda.com/products/individual

Jupyter notebook

Jupyter notebook is web application/internet application
The ***Jupyter Notebook*** is a web-based interactive computing platform. The notebook combines live code, equations, narrative text, visualizations, ...

Creating numpy array

Numpy is used to create arrays. The array object in numpy is called ndarray (OR) ndarray is data type or class which represent array object.
ndarray class is used to create array objects in python.
Numpy provide various functions for creating array.

1. array() : The function create ndarray object.

dtype: dtype is attribute of array, which define data type, the default type of array is object.
Numpy provide the following data types.

1. Integer data types.
    a. Int8 → 1byte
    b. Int16 → 2bytes
    c. Int32 → 4bytes
    d. Int64 → 8bytes
2. Float data types
    a. Float16 → 2bytes
    b. Float32 → 4bytes
    c. Float64 → 8bytes

3. Complex data types
    a. Complex64 → 8bytes
    b. Complex128 → 16bytes
4. Unsigned int data type
    a. Uint8 → 1byte
    b. Uint16 → 2bytes
    c. Uint32 → 4bytes
    d. Uint64 → 8bytes

This data types also represented using single characters.

1. i→ integer
2. f → float
3. u → unsigned int
4. s → string

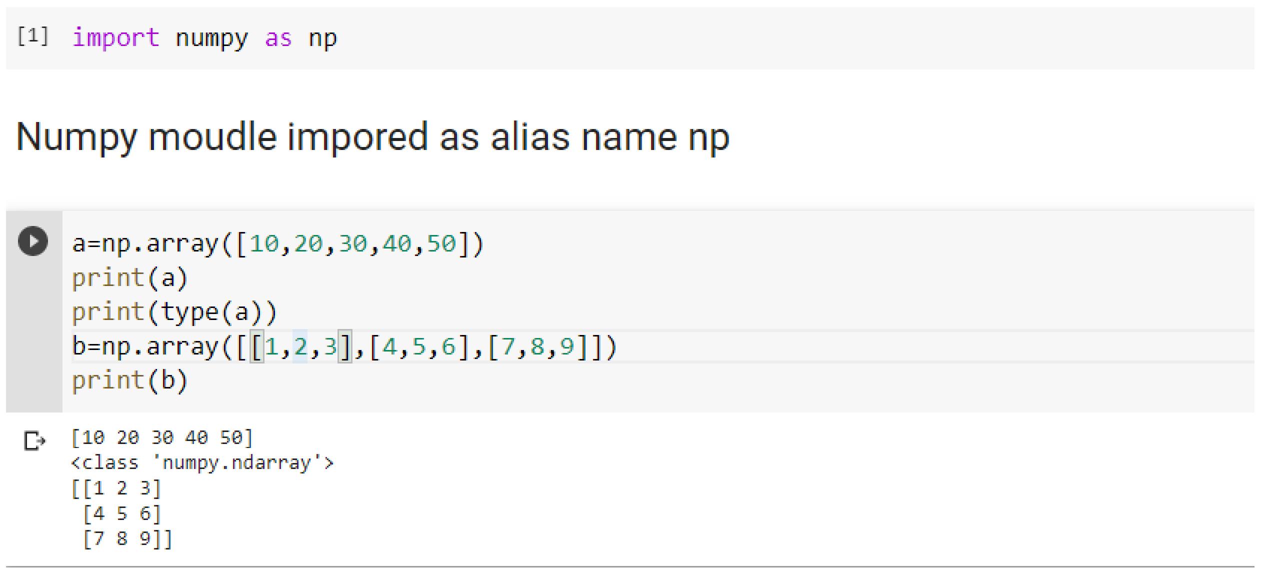
Syntax of array() function:
array(object,dtype,order,ndim)

object: object is an iterable/sequence which is used to construct array.
dtype: this indicates datatype of array
order: order can be C (row-major), F(column-major)
ndim: This specifies minimum number of dimensions of an output array.

Creating numpy array

Numpy is used to create arrays. The array object in numpy is called ndarray.
ndarray class is used to create array objects in python.
Numpy provide various functions for creating array.

2. array() : The function create ndarray object.



```
[1] import numpy as np
```

Numpy moudle impored as alias name np

```
a=np.array([10,20,30,40,50])
print(a)
print(type(a))
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(b)
```

```
[10 20 30 40 50]
<class 'numpy.ndarray'>
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

dtype: dtype is attribute of array, which define data type, the default type of array object.
Numpy provide the following data types.

5. Integer data types.
    a. Int8 → 1byte
    b. Int16 → 2bytes
    c. Int32 → 4bytes
    d. Int64 → 8bytes
6. Float data types
    a. Float16 → 2bytes
    b. Float32 → 4bytes

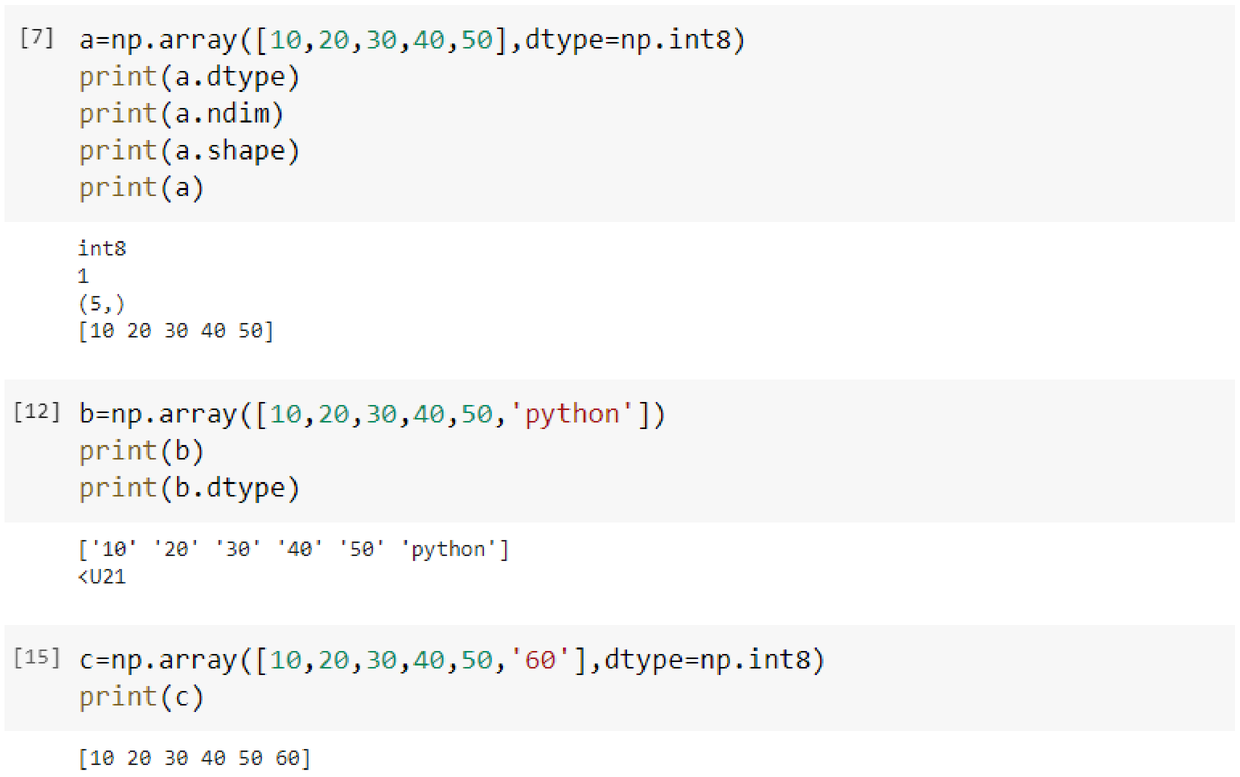
        c. Float64 → 8bytes
7. Complex data types
    a. Complex64 → 8bytes
    b. Complex128 → 16bytes
8. Unsigned int data type
    a. Uint8 → 1byte
    b. Uint16 → 2bytes
    c. Uint32 → 4bytes
    d. Uint64 → 8bytes

This data types also represented using single characters.

5. i→ integer
6. f → float
7. u → unsigned int
8. s → string

Syntax of array() function:
array(object,dtype,order,ndim)

object: object is an iterable/sequence which is used to construct array.
dtype: this indicates datatype of array
order: order can be C (row-major), F(column-major)
ndim: This specifies minimum number of dimensions of an output array.



```
[7] a=np.array([10,20,30,40,50],dtype=np.int8)
    print(a.dtype)
    print(a.ndim)
    print(a.shape)
    print(a)
```

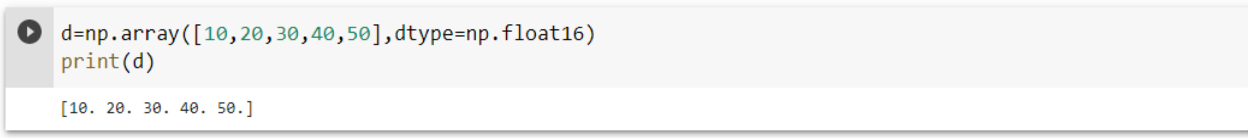
```
int8
1
(5,)
[10 20 30 40 50]
```

```
[12] b=np.array([10,20,30,40,50,'python'])
     print(b)
     print(b.dtype)
```

```
['10' '20' '30' '40' '50' 'python']
<U21
```

```
[15] c=np.array([10,20,30,40,50,'60'],dtype=np.int8)
     print(c)
```

```
[10 20 30 40 50 60]
```



```
d=np.array([10,20,30,40,50],dtype=np.float16)
print(d)
```

```
[10. 20. 30. 40. 50.]
```

Q: What is dimension?

Dimension define depth of array or nested.

Q: What is shape of the array/

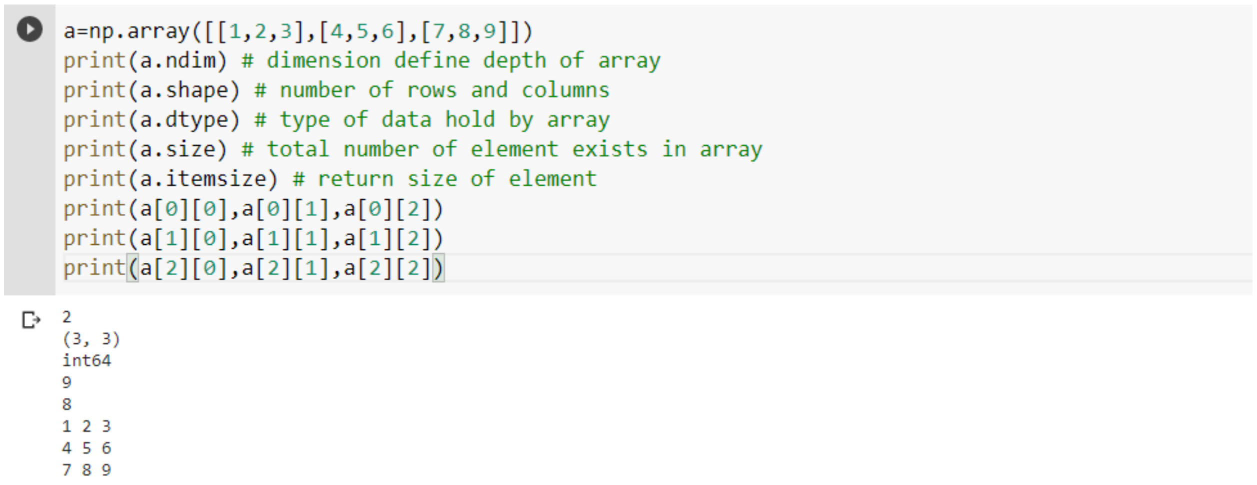
Shape define the number of rows and columns

Q: What is size?

The size will define total number elements exists within array

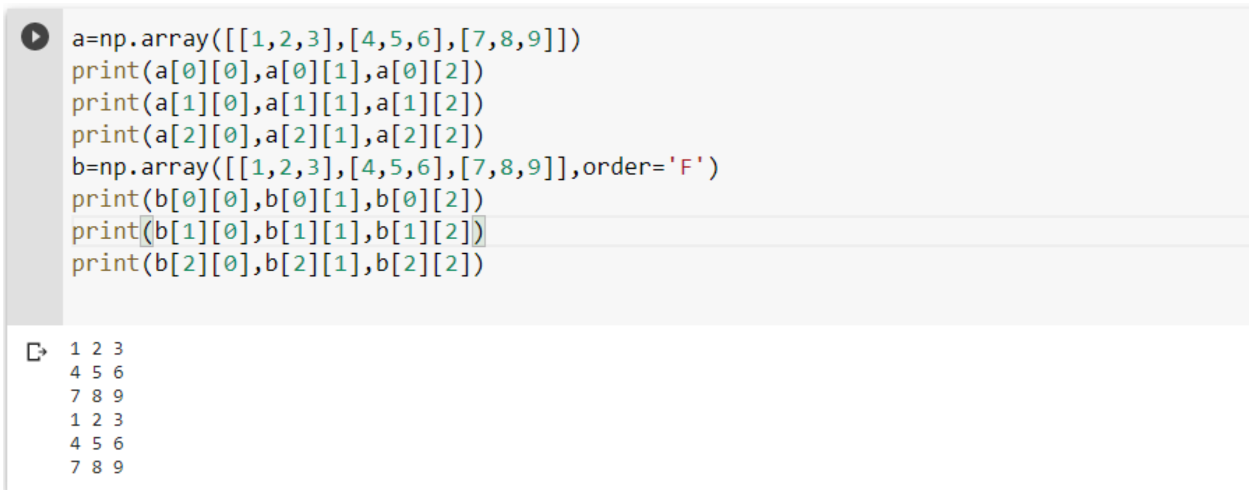
Q: What is dtype?

Type of elements hold by array



```python
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a.ndim) # dimension define depth of array
print(a.shape) # number of rows and columns
print(a.dtype) # type of data hold by array
print(a.size) # total number of element exists in array
print(a.itemsize) # return size of element
print(a[0][0],a[0][1],a[0][2])
print(a[1][0],a[1][1],a[1][2])
print(a[2][0],a[2][1],a[2][2])
```

```
2
(3, 3)
int64
9
8
1 2 3
4 5 6
7 8 9
```



```python
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a[0][0],a[0][1],a[0][2])
print(a[1][0],a[1][1],a[1][2])
print(a[2][0],a[2][1],a[2][2])
b=np.array([[1,2,3],[4,5,6],[7,8,9]],order='F')
print(b[0][0],b[0][1],b[0][2])
print(b[1][0],b[1][1],b[1][2])
print(b[2][0],b[2][1],b[2][2])
```

```
1 2 3
4 5 6
7 8 9
1 2 3
4 5 6
7 8 9
```

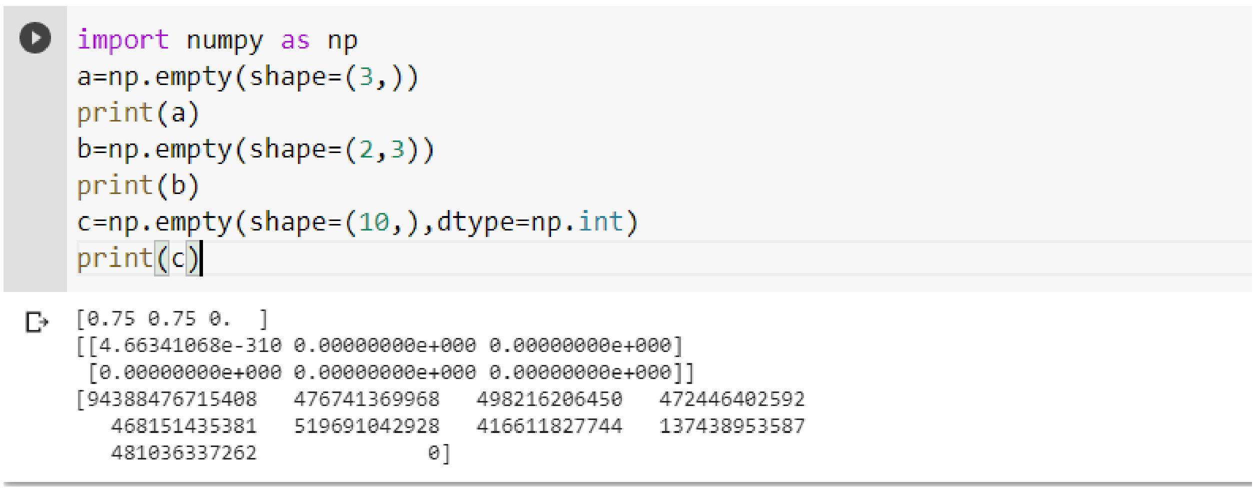
Other functions of creating numpy array

1. **empty()**

   This function return empty array.

   Syntax:

   empty(shape,dtype=float)

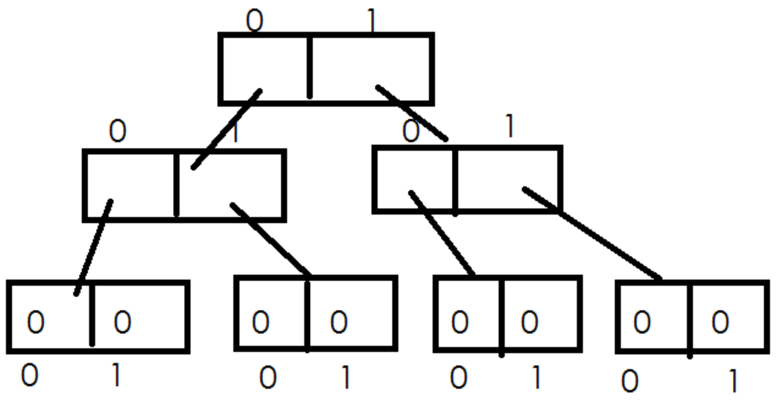


```python
import numpy as np
a=np.empty(shape=(3,))
print(a)
b=np.empty(shape=(2,3))
print(b)
c=np.empty(shape=(10,),dtype=np.int)
print(c)
```

```
[0.75 0.75 0.  ]
[[4.66341068e-310 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000]]
[94388476715408   476741369968   498216206450   472446402592
   468151435381   519691042928   416611827744   137438953587
   481036337262              0]
```

This will return ndarray with uninitialized values.

sales[5][5][5]



0,0,0 0,0,1 0,1,0 0,1,1   1,0,0 1,0,1 1,1,0 1,1,1

ndarray class or data type

This class or data type is used to represent array object. Numpy provides various functions for creating ndarray object

1. array()
2. arrange()
3. zeros()
4. ones()
5. full()

6. eye()
7. identity()

numpy provides the following data types for representing data of array.

Numpy provide the following data types.

1. Integer data types.
    a. Int8 → 1byte
    b. Int16 → 2bytes
    c. Int32 → 4bytes
    d. Int64 → 8bytes
2. Float data types
    a. Float16 → 2bytes
    b. Float32 → 4bytes
    c. Float64 → 8bytes
3. Complex data types
    a. Complex64 → 8bytes
    b. Complex128 → 16bytes
4. Unsigned int data type
    a. Uint8 → 1byte
    b. Uint16 → 2bytes
    c. Uint32 → 4bytes
    d. Uint64 → 8bytes

This data types also represented using single characters.

1. i→ integer

2. f → float
3. u → unsigned int
4. s → string

array() function

Syntax:
array(object,dtype,order,ndim)

array object is having following properties

1. dtype → return data type of array
2. order → organizing elements (C,F), row-major,col-major
3. ndim → number of dimensions
4. shape → number of rows and cols
5. size → total number of element of array

```
>>> import numpy as np
>>> a1=np.array([10,20,30,40,50])
>>> print(a1)
[10 20 30 40 50]
>>> type(a1)
<class 'numpy.ndarray'>
>>> a1.size
5
>>> a1.dtype
dtype('int32')
>>> a1.ndim
1
>>> a1.shape
```

```
(5,)
>>> a2=np.array((10,20,30,40,50),dtype=np.int8)
>>> print(a2)
[10 20 30 40 50]
>>> a3=np.array([10,20,30,40,50])
>>> print(a3.dtype)
int32
>>> print(a2.dtype)
int8
>>> a4=np.array([10,20,30,1.5])
>>> print(a4)
[10.  20.  30.   1.5]
>>> a4.dtype
dtype('float64')
>>> a5=np.array([10,20,30,40,'50'])
>>> print(a5)
['10' '20' '30' '40' '50']
>>> a5.dtype
dtype('<U11')
>>>
```

Note: array hold similar type of elements. If array created with multiple types of element, it converts to broader type.

```
>>>m1=np.array([[1,2,3],
            [4,5,6],
...             [7,8,9]])
>>> print(m1)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
>>> print(m1.ndim)
```

```
2
>>> print(m1.shape)
(3, 3)
>>> m2=np.array(((1,2),
...             (3,4),
...             (5,6)))
>>> print(m2)
[[1 2]
 [3 4]
 [5 6]]
>>> print(m2.shape)
(3, 2)
>>> print(m2.ndim)
2
>>>
```

arange()
This function return ndarray object.
There are various syntax of arange function

Syntax1: arange(stop)
Syntax2: arange(start,stop)
Syntax3: arange(start,stop,step)

Creating array using range of values.

Syntax1: arange(stop)
This syntax allows only one input stop
Default start=0,step=+1

```
>>> a2=np.arange(6)
>>> print(a2)
[0 1 2 3 4 5]
>>> a2.reshape((2,3))
array([[0, 1, 2],
       [3, 4, 5]])
>>> a2.shape=(2,3)
>>> print(a2)
[[0 1 2]
 [3 4 5]]
>>> a2.ndim
2
```

Syntax2: arange(start,stop)

```
>>> a3=np.arange(1,6)
>>> print(a3)
[1 2 3 4 5]
>>> a4=np.arange(1,7)
>>> print(a4)
[1 2 3 4 5 6]
a4.shape=(2,3)
print(a4)
[[1 2 3]
 [4 5 6]]
```

Syntax3: arange(start,stop,step)

```
>>> a5=np.arange(10,60,10)
>>> print(a5)
[10 20 30 40 50]
>>> a6=np.arange(-1,-6,-1)
>>> print(a6)
[-1 -2 -3 -4 -5]
```

```
>>> a7=np.arange(10,70,10)
>>> print(a7)
[10 20 30 40 50 60]
>>> a7.shape=(2,3)
>>> print(a7)
[[10 20 30]
 [40 50 60]]
>>> a8=np.arange(1,10)
>>> print(a8)
[1 2 3 4 5 6 7 8 9]
>>> a8.shape=(3,3)
>>> print(a8)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

zeros()
This function returns array with zero filled

```
>>> a1=np.zeros(shape=(5,))
>>> print(a1)
[0. 0. 0. 0. 0.]
>>> a2=np.zeros(shape=(3,3))
>>> print(a2)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
>>> a3=np.zeros(shape=(4,4),dtype=np.int8)
>>> print(a3)
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
>>>
```

ones()
this function returns array/ndarray object filled with ones
Syntax: ones(shape,dtype=np.float)

```
a1=np.zeros(shape=(5,))
print(a1)
[0. 0. 0. 0. 0.]
a2=np.zeros(shape=(3,3))
print(a2)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
a3=np.zeros(shape=(2,3,3))
print(a3)
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
a1=np.ones(shape=(5,))
print(a1)
[1. 1. 1. 1. 1.]
a2=np.ones(shape=(4,4))
print(a2)
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
a3=np.ones(shape=(3,3,3))
print(a3)
[[[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.]]]
>>> a1.ndim
1
>>> a2.ndim
2
>>> a3.ndim
3
>>> a1.shape
(5,)
>>> a2.shape
(4, 4)
>>> a3.shape
(3, 3, 3)
>>>
```

full()
this function return ndarray object using given value filled
we can use this function to create 1-d,2-d,..n-d array

full(shape,value,dtype)

```
>>> a1=np.full(shape=(5,),fill_value=10)
>>> print(a1)
[10 10 10 10 10]
>>> a2=np.full(shape=(3,3),fill_value=5)
>>> print(a2)
[[5 5 5]
 [5 5 5]
 [5 5 5]]
>>> a3=a2.reshape((9,))
>>> print(a3)
[5 5 5 5 5 5 5 5 5]
```

full()
this function return ndarray object using given value filled
we can use this function to create 1-d,2-d,..n-d array

full(shape,value,dtype)

```
>>> a1=np.full(shape=(5,),fill_value=10)
>>> print(a1)
[10 10 10 10 10]
>>> a2=np.full(shape=(3,3),fill_value=5)
>>> print(a2)
[[5 5 5]
 [5 5 5]
 [5 5 5]]
>>> a3=a2.reshape((9,))
>>> print(a3)
[5 5 5 5 5 5 5 5 5]
```

eye()

this function return ndarray object
this function is used to create eye matrix or identity matrix or unit matrix

A **unit matrix** can be defined as a scalar matrix in which all the diagonal elements are equal to 1 and all the other elements are zero.

Syntax: eye(N,M=None,k=0)

eye()

this function return ndarray object
this function is used to create eye matrix or identity matrix or unit matrix

A **unit matrix** can be defined as a scalar matrix in which all the diagonal elements are equal to 1 and all the other elements are zero.

Syntax: eye(N,M=None,k=0)

N represents number of rows
M represents number of columns
K represents diagonal principle

Example:

```
>>> import numpy as np
>>> m=np.eye(3,3)
>>> print(m)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
>>> m1=np.eye(4,4)
>>> print(m1)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
>>> m2=np.eye(4,4,1)
>>> print(m2)
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
>>> m3=np.eye(4,4,2)
>>> print(m3)
[[0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
>>> m4=np.eye(5,5,3)
>>> print(m4)
[[0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
>>> m5=np.eye(5,5,2,dtype=np.int8)
>>> print(m5)
```

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

Arithmetic Operations on numpy array

numpy provides various functions to perform arithmetic operations.
The following methods provided by ndarray.

1. add()
2. subtract()
3. multiply()
4. divide()
5. dot()
6. floor_divide()
7. mod()
8. power()

All above operations can be done using operators.

Example: add() -→ + , subtract() → -

add() : This function add two arrays

```
>>> a1=np.array([1,2,3])
>>> a2=np.array([4,5,6])
>>> a3=np.add(a1,a2)
>>> print(a1,a2,a3,sep="\n")
[1 2 3]
[4 5 6]
[5 7 9]
>>> m1=np.array([[1,2,3],
...             [4,5,6],
...              [7,8,9]])
>>> m2=np.array([[3,4,5],
...              [9,8,7],
...              [3,2,1]])
>>> m3=m1+m2
>>> print(m1,m2,m3,sep="\n")
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[3 4 5]
 [9 8 7]
 [3 2 1]]
[[ 4  6  8]
 [13 13 13]
 [10 10 10]]
>>>
```

numpy.subtract(x1,x2)
Subtract arguments, element-wise.

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> c=np.subtract(a,b)
>>> print(a,b,c,sep="\n")
[4 5 6]
[1 2 3]
[3 3 3]
>>> x=np.array([[4,5,6],
...             [8,9,10]])
>>> y=np.array([[1,2,3],
...             [4,5,6]])
>>> z=np.subtract(x,y)
>>> print(x,y,z,sep="\n")
[[ 4  5  6]
 [ 8  9 10]]
[[1 2 3]
 [4 5 6]]
[[3 3 3]
 [4 4 4]]
```

numpy.multiply(x1,x2)
Multiply arguments element-wise.

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> c=np.multiply(a,b)
>>> print(a,b,c,sep="\n")
[4 5 6]
[1 2 3]
[ 4 10 18]
>>> x=np.array([[4,5,6],
...             [8,9,10]])
>>> y=np.array([[1,2,3],
...             [4,5,6]])
>>> z=np.multiply(x,y)
>>> print(x,y,z,sep="\n")
[[ 4  5  6]
 [ 8  9 10]]
[[1 2 3]
 [4 5 6]]
[[ 4 10 18]
 [32 45 60]]
```

numpy.dot(a,b)
Dot product of two arrays

```
>>> m1=np.array([[1,2],
...             [3,4]])
>>> m2=np.array([[4,5],
...             [6,7]])
>>> m3=np.dot(m1,m2)
>>> print(m1,m2,m3,sep="\n")
[[1 2]
 [3 4]]
[[4 5]
 [6 7]]
[[16 19]
 [36 43]]
>>>
```

numpy.divide(x1,x2)
Divide arguments element-wise.

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> c=np.divide(a,b)
>>> print(a,b,c,sep="\n")
[4 5 6]
[1 2 3]
[4.  2.5 2. ]
```

numpy.floor_divide(x1,x2)

Divide arguments element-wise. Return result in integer

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> d=np.floor_divide(a,b)
>>> print(a,b,d,sep="\n")
[4 5 6]
[1 2 3]
[4 2 2]
>>>
```

numpy.mod(*x1, x2)*

Returns the element-wise remainder of division.

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> c=np.mod(a,b)
>>> print(a,b,c,sep="\n")
[4 5 6]
[1 2 3]
[0 1 0]
>>>
```

numpy.power(x1,x2)

First array elements raised to powers from second array, element-wise.

```
>>> a=np.array([4,5,6])
>>> b=np.array([1,2,3])
>>> c=np.power(a,b)
>>> print(a,b,c,sep="\n")
[4 5 6]
[1 2 3]
```

```
[  4  25 216]
```

Reading elements/values from numpy array
Reading elements from numpy array is done using different approaches

1. indexing
2. slicing

Indexing
Index is an integer value which is used to read one element from array.

```
>>> a=np.array([4,5,6])
>>> print(a[0],a[1],a[2])
4 5 6
>>> x=np.array([[4,5,6],
...             [8,9,10]])
>>> print(x[0])
[4 5 6]
>>> print(x[1])
[ 8  9 10]
>>> print(x[0][0],x[0][1],x[0][2])
4 5 6
>>> print(x[1][0],x[1][1],x[1][2])
8 9 10
```

Advanced Indexing
We can read elements based condition

```
>>> a=np.array([1,2,3,4,5,6,7,8,9,10])
>>> print(a)
[ 1  2  3  4  5  6  7  8  9 10]
```

```
>>> b=a[a%2==0]
>>> print(b)
[ 2  4  6  8 10]
>>> c=a[a%2!=0]
>>> print(c)
[1 3 5 7 9]
>>>
>>> d=a[a>=5]
>>> print(d)
[ 5  6  7  8  9 10]
```

We can read elements using multiple indexes. These multiple indexes are listed within list.

```
>>> a=np.array([1,2,3,4,5,6,7,8,9,10])
>>> index=[2,5,8]
>>> b=a[index]
>>> print(a)
[ 1  2  3  4  5  6  7  8  9 10]
>>> print(b)
[3 6 9]

>>> matrix=np.array([[1,2,3],
...                  [4,5,6],
...                  [7,8,9]])
>>> print(matrix)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
>>> import numpy as np
>>> a=np.array([[1,2,3],
          [4,5,6],
          [7,8,9]])
>>> print(a)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
>>> print(a[0])
[1 2 3]
>>> print(a[1])
[4 5 6]
>>> print(a[2])
[7 8 9]
>>> print(a[0][0],a[0][1],a[0][2])
1 2 3
>>> print(a[1][0],a[1][1],a[1][2])
4 5 6
>>> print(a[2][0],a[2][1],a[2][2])
7 8 9
>>> index=([0,1,2],[2,0,1])
>>> a[index]
array([3, 4, 8])
```

slicing
using slicing we can read more than one value

```
>>> a=np.array([10,20,30,40,50,60,70,80,90,100])
>>> print(a)
[ 10  20  30  40  50  60  70  80  90 100]
>>> b=a[0:5]
>>> print(b)
```

```
[10 20 30 40 50]
>>> c=a[5:]
>>> print(c)
[ 60  70  80  90 100]
>>> d=a[-1:-6:-1]
>>> print(d)
[100  90  80  70  60]
>>>
a=np.array([10,20,30,40,50,60,70,80,90,100])
print(a)
[ 10  20  30  40  50  60  70  80  90 100]
b=a[0:5]
print(b)
[10 20 30 40 50]
c=a[5:]
print(c)
[ 60  70  80  90 100]
d=a[-1:-6:-1]
>>> print(d)
[100  90  80  70  60]
>>> b=np.array([[1,2,3],
...            [4,5,6],
...            [7,8,9]])
>>> print(b)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
>>> c=b[0:2]
>>> print(c)
[[1 2 3]
 [4 5 6]]
>>> d=b[-1:-2:-1]
>>> print(d)
[[7 8 9]]
>>> e=b[0:2,0:2]
>>> print(e)
[[1 2]
 [4 5]]
>>>
```

Statistical Operations

Numpy library provides the functions to perform statistical operations on array.

1. amax()
2. amin()
3. mean()
4. median()
5. var()
6. std()

amax()

Return the maximum of an array or maximum along an axis.

numpy.amax**(***a***,** *axis=None)*

```
>>> a=np.array([10,20,30,40,50])
>>> np.amax(a)
50
>>> b=np.array([[1,2,3],
...             [4,5,6],
...             [7,8,9]])
>>> np.amax(b)
```

```
9
>>> np.amax(b,axis=0)
array([7, 8, 9])
>>> np.amax(b,axis=1)
array([3, 6, 9])
```

amin()

numpy.**amin(***a*, *axis=None)*

Return the minimum of an array or minimum along an axis.

```
>>> a=np.array([10,20,30,40,50])
>>> np.amin(a)
10
>>> b=np.array([[1,2,3],
...             [4,5,6],
...             [7,8,9]])
>>> np.amin(b)
1
>>> np.amin(b,axis=0)
array([1, 2, 3])
>>> np.amin(b,axis=1)
array([1, 4, 7])
>>>
```

mean()

numpy.**mean(***a*, *axis=None*)

Compute the arithmetic mean along the specified axis.
Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

```
a=np.array([1,2,3,4,5,6,7,8,9,10])
print(a)
[ 1  2  3  4  5  6  7  8  9 10]
>>> m=np.mean(a)
>>> print(m)
5.5
>>> b=np.array([[1,2,3],
...             [4,5,6],
...             [7,8,9]])
>>> print(b)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
>>> m=np.mean(b)
>>> print(m)
5.0
>>> m1=np.mean(b,axis=0)
>>> print(m1)
[4. 5. 6.]
>>> m2=np.mean(b,axis=1)
>>> print(m2)
[2. 5. 8.]
>>>
```

median()

numpy.**median(***a*, *axis=None*)

Compute the median along the specified axis.

Returns the median of the array elements.

This function returns middle element

Sort the elements of array in ascending order

If the number of elements in array is odd number, it returns middle element

If the number of elements in array is even number, it read two middle elements, add it and divide with 2

```
>>> a=np.array([1,2,3,4,5,6,7])
>>> print(a)
[1 2 3 4 5 6 7]
>>> np.median(a)
4.0
>>> b=np.array([1,2,3,4,5,6,7,8])
>>> print(b)
[1 2 3 4 5 6 7 8]
>>> np.median(b)
4.5
```

var()

numpy.**var(***a, axis=None)*

Compute the variance along the specified axis.

Returns the variance of the array elements, a measure of the spread of a distribution. The variance is computed for the flattened array by default, otherwise over the specified axis.

variance=sum(sqr(mean-xi))/total of number of elements

Xi is an each element of array

```
>>> a=np.array([1,2,3,4,5])
>>> print(a)
[1 2 3 4 5]
>>> np.var(a)
2.0
```

std()

numpy.**std(***a*, *axis=None)*

Compute the standard deviation along the specified axis.
Returns the standard deviation, a measure of the spread of a distribution, of the array elements. The standard deviation is computed for the flattened array by default, otherwise over the specified axis.

std=sqrt(var)

```
>>> a=np.array([1,2,3,4,5])
>>> print(a)
[1 2 3 4 5]
>>> np.var(a)
2.0
>>> np.std(a)
1.4142135623730951
>>> b=np.array([1,2,3,4,5,6,7,8,9,10])
>>> np.var(b)
8.25
>>> np.std(b)
2.8722813232690143
>>> import math
>>> math.sqrt(np.var(b))
2.8722813232690143
```