```
import numpy as np
print(np.__version__)
```

1) Array()

We can create a NumPy ndarray object by using the array() function.

Example:

```
import numpy as np
a = np.array([1, 2, 3, 4, 5])
print(a)
print(type(a))
```

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Eg:

```
import numpy as np
a = np.array((1, 2, 3, 4, 5))
print(a)
```

0-D Arrays

```
import numpy as np
a = np.array(42)
print(a)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

Eg:

```
import numpy as np
a = np.array([1, 2, 3, 4, 5])
print(a)
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

Eg:

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
```

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

2) Arrange()

```
numpy.arange([start, ]stop, [step, ], dtype=None)
```

```
>>> np.arange(start=1, stop=10, step=3)
array([1, 4, 7])

>>> np.arange(10)  # Stop is 10, start is 0, and step is 1!
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.arange(1, 10, 3)
array([1, 4, 7])

>>> np.arange(1, 10.1, 3)
array([ 1.,  4.,  7., 10.])

>>> np.arange(-5, -1)
array([-5, -4, -3, -2])
>>> np.arange(-8, -2, 2)
array([-8, -6, -4])
>>> np.arange(-5, 6, 4)
array([-5, -1,  3])

>>> x = np.arange(5, dtype=int)
>>> x
array([0, 1, 2, 3, 4])
>>> x.dtype
```

```
dtype('int64')

>>> x = np.arange(5, dtype=np.int32)
>>> x
array([0, 1, 2, 3, 4], dtype=int32)
>>> x.dtype
dtype('int32')
>>> x.itemsize  # In bytes
4
```

3. linespace()

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

```
>>> import numpy as np
>>> np.linspace(3.0, 4.0, num=7)
array([ 3.        ,  3.16666667,  3.33333333,  3.5       ,  3.66666667,
        3.83333333,  4.        ])
>>> np.linspace(3.0,4.0, num=7, endpoint=False)
array([ 3.        ,  3.14285714,  3.28571429,  3.42857143,  3.57142857,
        3.71428571,  3.85714286])
>>> np.linspace(3.0,4.0, num=7, retstep=True)
(array([ 3.        ,  3.16666667,  3.33333333,  3.5       ,  3.66666667,
        3.83333333,  4.        ]), 0.16666666666666666)

>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> A = 5
>>> x = np.zeros(A)
>>> a1 = np.linspace(0, 10, A, endpoint=True)
>>> a2 = np.linspace(0, 10, A, endpoint=False)
>>> plt.plot(a1, x, 'o')
[<matplotlib.lines.Line2D object at 0x7f3d13a48080>]
>>> plt.plot(a2, x + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x7f3d1b582438>]
>>> plt.ylim([-5.0, 1])
(-5.0, 1)
>>> plt.show()
```

4. zeros()

The zeros() function is used to get a new array of given shape and type, filled with zeros.
Syntax

```
numpy.zeros(shape, dtype=float, order='C')
```

Eg:

```
import numpy as np
a=np.zeros(6)
a

import numpy as np
a=np.zeros((6,), dtype=int)
a

import numpy as np
a=np.zeros((6,2))
a


Import numpy as np
s1=(3,2)
a=np.zeros(s1)
a
```

5. ones()

The ones() function is used to get a new array of given shape and type, filled with ones.
Syntax

```
numpy.ones(shape, dtype=None, order='C')
```

Example

```
>>> import numpy as np
>>> np.ones(7)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.])
>>> np.ones((2, 1))
array([[ 1.],
      [ 1.]])
>>> np.ones(7,)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.])
>>> x = (2, 3)
>>> np.ones(x)
array([[ 1.,  1.,  1.],
      [ 1.,  1.,  1.]])
>>> import numpy as np
>>> np.ones((2,2)) #default datatype float
```

```
array([[ 1.,  1.],
      [ 1.,  1.]])
>>> np.ones((2,2), dtype=int)
array([[1, 1],
      [1, 1]])
```

6. Full()

The full() function return a new array of given shape and type, filled with fill_value.

Syntax:

numpy.full(shape, fill_value, dtype=None, order='C')

Example

```
>>>import numpy as np
>>> np.full((3, 3), np.inf)
array([[ inf,  inf,  inf],
      [ inf,  inf,  inf],
      [ inf,  inf,  inf]])
>>> np.full((3, 3), 10.1)
array([[ 10.1,  10.1,  10.1],
      [ 10.1,  10.1,  10.1],
      [ 10.1,  10.1,  10.1]])
```

7. Eye()

The eye() function is used to create a 2-D array with ones on the diagonal and zeros elsewhere.

Syntax:

numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

where

N	Number of rows in the output. Required
M	Number of columns in the output. If None, defaults to N.
k	Index of the diagonal: 0 (the default) refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal.

Example

```
>>> import numpy as np
>>> np.eye(2)
array([[ 1.,  0.],
      [ 0.,  1.]])
>>> np.eye(2,3)
array([[ 1.,  0.,  0.],
      [ 0.,  1.,  0.]])
>>> np.eye(3, 3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>>
```

8. Identity()

The identity array is a square array with ones on the main diagonal. The identity() function return the identity array.

Syntax:

numpy.identity(n, dtype=None)

Example

```
>>> import numpy as np
>>> np.identity(2)
array([[ 1.,  0.],
       [ 0.,  1.]])

>>> import numpy as np
>>> a = np.arange(12).reshape((4,3))
>>> np.diag(a, k=1)
array([1, 5])

>>> import numpy as np
>>> a = np.arange(12).reshape((4,3))
>>> np.diag(a, k=-1)
array([ 3,  7, 11])
```

9. Diag()

The diag() function is used to extract a diagonal or construct a diagonal array.

Syntax:

numpy.diag(v, k=0)

Example

```
>>> import numpy as np
>>> a = np.arange(12).reshape((4,3))
>>> np.diag(a)
array([0, 4, 8])
```

10. Empty()

This function is used to create an array without initializing the entries of given shape and type. This function requires the user to set all the values in the array manually.

Syntax

numpy.empty(shape, dtype=float, order='C')

Example

```
import numpy as np
x = np.empty([3, 2])
x
```

```
import numpy as np
x = np.empty([3, 3], dtype=float)
x
```

random module

NumPy offers the random module to work with random numbers. This library contains several functions to create n-d arrays with random data.
i. randint() ==> To generate random int values in the given range.
ii. rand() ==> uniform distribution in the range [0,1)
iii. uniform() ==> uniform distribution in our provided range
iv. randn() ==> normal distribution values with mean 0 and variance is 1
v. normal() ==> normal distribution values with our mean and variance.

i) randint

To generate random int values in the given range
Synatax

randint(low, high=None, size=None, dtype = int)

- Return random integers from low (inclusive) to high (exclusive).
- It is represented as [low, high) ☾ >' [' means inclusive and ') ' means exclusive
- If high is None (by default), then results from [0, low)

Example
Program that will generate a single random int value in the range 10 to 19.

```
import numpy as np9
print(np9.random.randint(10,20))
```

O/P

```
C:\book>py h.py
15
```

Program to create 1-Dimensional ndarray of size 5 with random values from 1 to 8?

```
import numpy as np9
print(np9.random.randint(1, 9, size=5) )
```

O/P

```
C:\book>py h.py
[3 6 7 4 7]
```

ii) rand()

It will generates random float values in the range [0,1) from uniform distribution samples. [0 ==> means 0 is incluede and 1) ==> means 1 is excluded
Syntax

numpy.random.rand(d0, d1, d2,..., dn)

Example
Program to print random float number using uniform() function.
```
import numpy as np9
print(np9.random.uniform()) # it will acts same as np9.random. rand()
```
O/P
```
C:\book>py h.py
0.9404791837500721
```

Program to print random float value in the given customized range.
```
import numpy as np9
print(np9.random.uniform(10,20))
```
O/P
```
C:\book>py h.py
15.816574043742694
```

Program to print 1-D array with customized range of float values.
```
import numpy as np9
print(np9.random.uniform(10,20,size=10))
```
O/P
```
C:\book>py h.py
[18.1310384 14.10395587 13.3380287 15.79094315 17.20007353 14.47280544
19.88410912 11.42031482 16.94662155 11.44077793]
```

Program to print 2-D array with customized range of float values.
```
import numpy as np9
print(np9.random.uniform(10,20,size=(3,5)))
```
O/P
```
C:\book>py h.py
[[19.10672995 13.90164793 11.08141307 15.50829402 12.32057353]
[18.50717084 18.41184221 17.9648871 13.86784327 15.34135452]
[12.6092356 10.95898169 19.6283303 12.48246095 10.40802646]]
```

iii) uniform()
• rand() ☾ range is always [0,1)
• uniform() ☾ customize range [low, high)
Syntax:

uniform(low=0.0, high=1.0, size=None)

Example

Program to print random float number using uniform() function.

```
import numpy as np9
print(np9.random.uniform()) # it will acts same as np9.random. rand()
```

O/P

```
C:\book>py h.py
0.9404791837500721
```

iv) randn()

The values from normal distribution with mean 0 and variance(standard deviation) is 1.

Syntax

```
randn(d0, d1, d2 ..., dn)
```

Example

Program to find a single float value including -ve value.

```
import numpy as np9
a = np9.random.randn()
print(a)
```

O/P

```
C:\book>py h.py
0.7501336304615012
```

Program to find 1-D array with mean 0 and stand deviation 1.

```
import numpy as np9
a = np9.random.randn(10)
print(a)
print("Mean ", a.mean())
print("Variance ",a.var())
print("Standard deviation ",a.std())
```

O/P

```
C:\book>py h.py
[-0.09698142 0.83851884 -0.68160955 -1.66820751 1.50901544 0.96881446
0.72653551 -1.02128131 1.87966362 0.33067593]
Mean 0.27851440240634173
Variance 1.1613404701202303
Standard deviation 1.077655079383116
```

v) normal()

We can customize mean and variance

Syntax

```
normal(loc=0.0, scale=1.0, size=None)
```

- loc : float or array_like of floats
  Mean ("centre") of the distribution.
- scale : float or array_like of floats

Standard deviation (spread or "width") of the distribution. Must be non-negative.

- size : int or tuple of ints, optional

Example
Program to print random float number using normal() function.

```
import numpy as np9
print(np9.random.normal())
```

O/P

```
C:\book>py h.py
-0.7256715761343828
```

vi) shuffle()

Modify a sequence same-place by shuffling its contents. This function completely shuffles the given array along with the first axis of a multi-dimensional array, (axis-0). The order of sub-arrays is changed but their contents remain the same.
Syntax :

shuffle (lst)

Example
Program to print 1-Dimensional array using shuffle() function.

```
import numpy as np9
a = np9.arange(9)
print('a before shuffling ',a)
np9.random.shuffle(a)
print('a after shuffling ',a)
```

O/P

```
C:\book>py h.py
a before shuffling [0 1 2 3 4 5 6 7 8]
a after shuffling [8 5 4 6 1 2 3 0 7]
```

Program to print 2-Dimensional array using shuffle() function.