```
import numpy as np

lst = [1,2,3.4]
print(type(lst))

arr = np.array(lst, ndmin=2) #Converting a python list into an numpy
array and giving it dimensions
print(arr)
print(type(arr))
```

```
<class 'list'>
[[1.  2.  3.4]]
<class 'numpy.ndarray'>
```

Arrange Function

```
np.arange(1,10,2) #creates an array of numbers from starting number to
ending number with the number of steps after each number
for i in np.arange(1,10,2) :
    print(i)
```

```
1
3
5
7
9
```

Multidimensional Array

```
lst = [[1,2,3],[4,5,6],[7,8,9],[7,8,9]] #This is a two dimensional
list
print(lst)
arr = np.array(lst) #Converting two dimensional list to an np array
print(arr)
```

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

Size function

```
arr.size #This gives us the number of elements in the array
print('Total : ',arr.size)
print('Rows : ',np.size(arr, 0))            #This gives the number of
rows
print('Columns : ',np.size(arr,1))          #This gives the number of
columns
```

```
Total :  12
Rows :  4
Columns :  3
```

Shape Function

```
print(arr.shape) #This gives the order of the matrix
```

```
(4, 3)
```

Data Type

```
arr1 = np.array([1,2,3,1,2,3]) #Gives the type of data in the array
arr2 = np.array([1.2,3.1,2.3])
arr3 = np.array([1.2,3.1,3])


print(arr1.dtype)
print(arr2.dtype)
print(arr3.dtype)
```

```
int32
float64
float64
```

Ndim Function

```
print(arr.ndim) #How many dimensions the array is having

#we can also convert the dimension of the array
arr4 = np.array(arr, ndmin=4)
print(arr4)
print(arr4.ndim)
```

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

Zeroes Function

```
arr = np.zeros(shape=(3,5)) #Creates a numpy array/matrix with the
given order=shape, filled with zeroes in the places by default
print(arr.dtype)
print(arr)
```

```
float64
[[0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Ones Function

```
arr = np.ones(shape=(5,5)) #Creates a numpy array/matrix with the
given order=shape, filled with ones in the places by default
print(arr.dtype)
print(arr)
```

```
float64
[[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.]]
```

Eye Function

```
np.eye(5, dtype=int)   #Returns an identity matrix
```

```
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1]])
```

Empty Function

```
np.empty(shape=(5,5)) #This will create an empty matrix
```

```
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.]])
```

Random Functions

```
np.random.rand(3,4) #Creates an numpy array of random numbers with the
given order. The range of numbers will be from 0 to 1.
```

```
array([[0.6534109 , 0.55784076, 0.36156476, 0.2250545 ],
       [0.40651992, 0.46894025, 0.26923558, 0.29179277],
       [0.4576864 , 0.86053391, 0.5862529 , 0.28348786]])
```

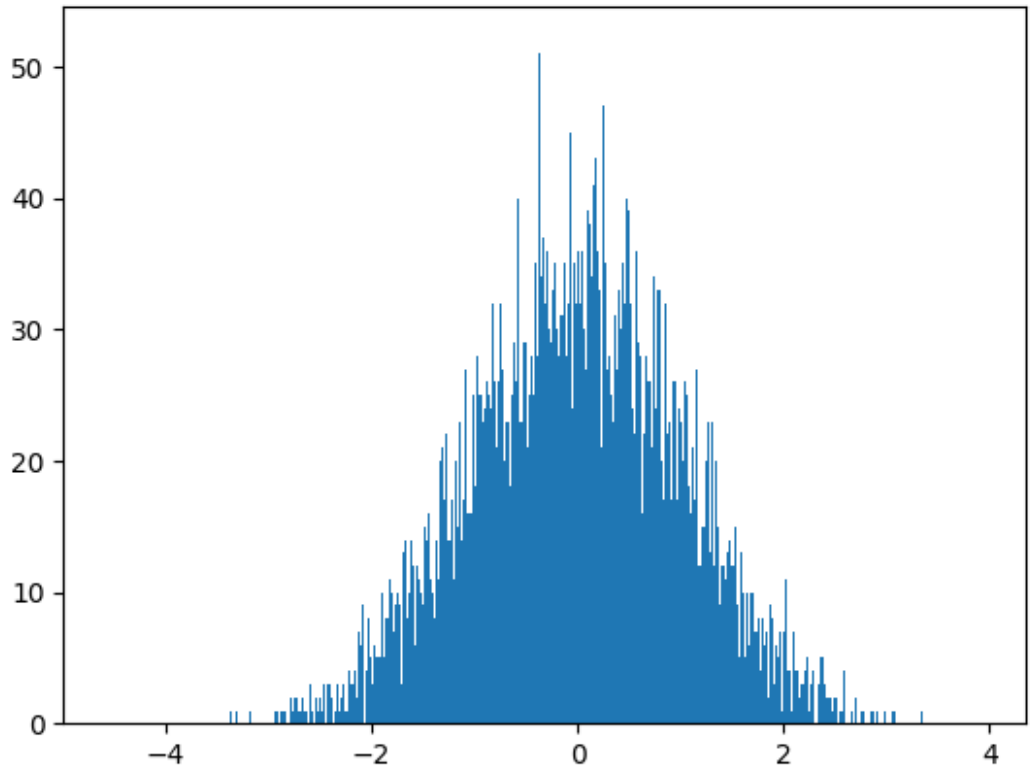
```
np.random.randint(low=1, high=20, size=100,) #Creates a numpy array of
random integers between the given low, high and of the given size
np.random.randint(low=1, high=15, size=(3,4)) #Method 2
np.random.randint(1,55,5) #Method 3
```

```
array([31, 46,  8, 34, 35])
```

```
arr = np.random.randn(100000) #generate random numbers according to a
specific probability distribution
print(arr)
import matplotlib.pyplot as plt
plt.hist(arr, bins=10000)
```

```
[ 1.34580752 -0.83644784 -0.41935506 ... -1.41711193 -0.63705021
 -0.18295412]

(array([1., 0., 0., ..., 0., 0., 1.]),
 array([-4.5696666 , -4.5688164 , -4.5679662 , ...,  3.93060944,
         3.93145963,  3.93230983]),
 <BarContainer object of 10000 artists>)
```



Reshape Function

```
arr = np.random.randint(low=1, high=15, size=(3,4))
print(arr.shape)
print(arr)
arr = arr.reshape(4,3) #Reshape the array into a different order
print(arr.shape)
print(arr)
#We can only reverse the shape into the same output size a*b = c , b*d
=c
```

```
(3, 4)
[[ 6 12  3  6]
 [11  7  9  8]
 [ 1  6  2  3]]
(4, 3)
[[ 6 12  3]
 [ 6 11  7]
 [ 9  8  1]
 [ 6  2  3]]
```

Linespace function

```
np.linspace(1,10,10) #Creates an numpy array of numbers which are
equidistant from each other

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

Flatten Function

```
arr = np.random.randint(1,15,(3,4))
print(arr.shape)
print(arr)
farr = arr.flatten() #Brings all the items of the arrays in one row
print(farr)

(3, 4)
[[13 12  7 13]
 [ 7  6  2 12]
 [ 3  1 14 11]]
[13 12  7 13  7  6  2 12  3  1 14 11]
```

Logspace Function

```
np.logspace(2,4,10, base=2) #Creates a numpy array of the log values
between the range, number of values and base

array([ 4.        ,  4.66611616,  5.44316   ,  6.34960421,
7.4069977 ,
        8.64047791, 10.0793684 , 11.75787594, 13.71590373,
16.        ])
```

Copy Function

```
arr = np.random.randint(1,15,10)
print(arr)
arr2 = arr  #This will copy the reference of the array not the items.
print(arr2)
arr3 = arr.copy() #This will not copy the reference only the values
print(arr3)
print(id(arr))
```

```
print(id(arr2))
print(id(arr3))
```

```
[ 2 11  4  8 14  2 13  3  5  7]
[ 2 11  4  8 14  2 13  3  5  7]
[ 2 11  4  8 14  2 13  3  5  7]
2156977934704
2156977934704
2156713326800
```

arr.max() , arr.min(), arr.sum() functions

```
arr = np.random.randint(1,15,(3,4))
print(arr)
print('Max:', arr.max())
print('Min:',arr.min())
print('Sum:',arr.sum())
```

```
[[11 13  5  3]
 [ 3 13  9 10]
 [13 11  1  6]]
Max: 13
Min: 1
Sum: 98
```

```
arr = np.random.randint(1,15,(3,4))
print(arr)
print('Max:', arr.max(axis=0)) #for specific column
print('Min:',arr.min(axis=1)) #for specific row
print('Sum:',arr.sum())
```

```
[[ 9  4  6  3]
 [ 3  3 11  1]
 [11 12  8  9]]
Max: [11 12 11  9]
Min: [3 1 8]
Sum: 80
```

Seed Function

```
np.random.seed(3) #Initialises a reference for the random values so
that everytime we call this seed(number) the same set of random
numbers is retured
#Like a label, the seed value helps identify a specific sequence of
random numbers generated by np.random.seed(). If you know the seed
value, you can predict the sequence of random numbers produced.
arr = np.random.randint(1,10,(3,5))
arr
```

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

Sorting Functions

```
np.random.seed(3)
arr = np.random.randint(1,10,(3,5))
print(arr)
print('-'*15)
print(np.sort(arr)) #By default, it is sorting the rows
print('-'*15)
print(np.sort(arr, axis=0)) #Here it is sorting the columns
print('-'*15)
arr.shape
np.reshape(np.sort(arr.flatten()), arr.shape)
```

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

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

```
np.reshape(np.sort(arr.flatten(), kind='mergesort'), arr.shape) #We
can also tell the type of Sorting Algorithm we want
```

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

Mathematical operations

```
np.random.seed(3)
arr = np.random.randint(1,10,(3,5))
print(arr)
print('-'*25)
print(arr+1)    # 1 is added to each item of the numpy array
print('-'*25)
print(arr-1)    # 1 is subtracted from each item of the numpy array
print('-'*25)
```

```
print(arr*3)    # 3 is multiplied by each item of the numpy array
print('-'*25)
print(arr//3)    # 3 is divided(quotient) by each item of the numpy
array
```

```
[[9 4 9 9 1]
 [6 4 6 8 7]
 [1 5 8 9 2]]
-------------------------
[[10  5 10 10  2]
 [ 7  5  7  9  8]
 [ 2  6  9 10  3]]
-------------------------
[[8 3 8 8 0]
 [5 3 5 7 6]
 [0 4 7 8 1]]
-------------------------
[[27 12 27 27  3]
 [18 12 18 24 21]
 [ 3 15 24 27  6]]
-------------------------
[[3 1 3 3 0]
 [2 1 2 2 2]
 [0 1 2 3 0]]
```

Matrix Multiplication

```
np.random.seed(3)
arr = np.random.randint(1,10,(2,2))
np.random.seed(2)
arr1 = np.random.randint(1,10,(2,2))
print(arr1, '\n \n' ,arr)
print('\n', arr1.dot(arr))      #To multiply two matrices
```

```
[[9 9]
 [7 3]]

 [[9 4]
 [9 9]]

 [[162 117]
 [ 90  55]]
```

Power

```
print(arr ** 2) #ALl the elements of matrix is raised to the given
power
```

```
[[81 16]
 [81 81]]
```

Percentile

```
np.random.seed(3)
arr = np.random.randint(1,10,10)
print(arr)
arrsort = np.sort(arr)
print(arrsort)
print(np.percentile(arrsort, 50)) #gives the percentile value from the
sorted array
```

```
[9 4 9 9 1 6 4 6 8 7]
[1 4 4 6 6 7 8 9 9 9]
6.5
```

Mean, Variance and Standard deviation

```
np.random.seed(3)
arr = np.random.randint(1,10,10)
print(arr)
print('mean : ',arr.mean())
print('median : ',arr.var())
print('standard deviation : ',arr.std())
```

```
[9 4 9 9 1 6 4 6 8 7]
mean :  6.3
median :  6.410000000000001
standard deviation :  2.5317977802344327
```

Filtering an numpy array

```
arr = np.arange(1,15)
print(arr)
print(arr>10)   #shows true for all the values which are greater than
10
print(arr[arr>5])   #shows all the values which are greater than 5 as
a seperate array
arr2 = arr[arr>11]
print(arr2)
arr2[arr2>5]=10 #Replaces all the values greater than with 5
print(arr2)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[False False False False False False False False False False  True
True
  True  True]
[ 6  7  8  9 10 11 12 13 14]
[12 13 14]
[10 10 10]
```

Transpose of an array - Convert rows into columns and vice versa

```
arr = np.random.randint(1,20,(3,3))
print(arr)
print('*'*15)
print(arr.T)        #Gives the transpose of the array
```

```
[[ 1 13  8]
 [15 18  3]
 [ 3  2  6]]
***************
[[ 1 15  3]
 [13 18  2]
 [ 8  3  6]]
```

Where Function

```
arr = np.random.randint(1,100,(3,3))
print(arr)
arr%2 == 0
```

```
[[98 30 25]
 [63  8 44]
 [34 80 49]]
```

```
array([[ True,  True, False],
       [False,  True,  True],
       [ True,  True, False]])
```

```
newarr = np.where(arr%2 == 0, 'even',arr) #Replaces the given values
in the array with the given value when the given condition comes true
for that particular element of the array
newarr = np.where(arr%2 !=0, 'odd',newarr)
print(newarr)
```

```
[['even' 'even' 'odd']
 ['odd' 'even' 'even']
 ['even' 'even' 'odd']]
```

Merging Arrays

a) Concatenate

```
print(arr)
print(newarr)
print(np.concatenate((arr,newarr)))    #Brings both of the elements of
the array together
print(np.concatenate((arr,newarr),axis=1))  #axis = 1 means add more
rows and axis =0 means add more columns
#Matrixes should be of same order
```

```
[[98 30 25]
 [63  8 44]
 [34 80 49]]
[['even' 'even' 'odd']
 ['odd' 'even' 'even']
 ['even' 'even' 'odd']]
[['98' '30' '25']
 ['63' '8' '44']
 ['34' '80' '49']
 ['even' 'even' 'odd']
 ['odd' 'even' 'even']
 ['even' 'even' 'odd']]
[['98' '30' '25' 'even' 'even' 'odd']
 ['63' '8' '44' 'odd' 'even' 'even']
 ['34' '80' '49' 'even' 'even' 'odd']]
```

V-stack and H-Stack

```
print(arr)
print(newarr)
print(np.hstack((arr, newarr))) #Horizontal concatenation
print(np.vstack((arr, newarr))) #Vertical concatenation
```

```
[[98 30 25]
 [63  8 44]
 [34 80 49]]
[['even' 'even' 'odd']
 ['odd' 'even' 'even']
 ['even' 'even' 'odd']]
[['98' '30' '25' 'even' 'even' 'odd']
 ['63' '8' '44' 'odd' 'even' 'even']
 ['34' '80' '49' 'even' 'even' 'odd']]
[['98' '30' '25']
 ['63' '8' '44']
 ['34' '80' '49']
 ['even' 'even' 'odd']
 ['odd' 'even' 'even']
 ['even' 'even' 'odd']]
```

Splitting Arrays

```
nextarr = np.hstack((arr, newarr))
print(nextarr)
print(np.hsplit(nextarr,2)) #Horizontally splits in two equal parts
print(np.vsplit(nextarr,3)) #Vertically splits in three equal parts
```

```
[['98' '30' '25' 'even' 'even' 'odd']
 ['63' '8' '44' 'odd' 'even' 'even']
 ['34' '80' '49' 'even' 'even' 'odd']]
[array([['98', '30', '25'],
```

```
       ['63', '8', '44'],
       ['34', '80', '49']], dtype='<U11'), array([['even', 'even',
'odd'],
       ['odd', 'even', 'even'],
       ['even', 'even', 'odd']], dtype='<U11')]
[array([['98', '30', '25', 'even', 'even', 'odd']], dtype='<U11'),
array([['63', '8', '44', 'odd', 'even', 'even']], dtype='<U11'),
array([['34', '80', '49', 'even', 'even', 'odd']], dtype='<U11')]
```

How to put images on a numpy array

```
from matplotlib.image import imread

img = imread('../Assets/batman-8510022_640.jpg')
print(img.shape)    #Gives the number of pixels in height and width
and channel
print(img)  #Gives the pixels of image in the matrix format


(640, 640, 3)
[[[24 26 27]
  [30 31 31]
  [32 33 35]
  ...
  [16 13 12]
  [10  8  8]
  [11  8  8]]

 [[26 26 27]
  [30 31 32]
  [32 33 34]
  ...
  [12  9 10]
  [12 11 10]
  [19 17 15]]

 [[27 27 27]
  [31 32 32]
  [33 33 34]
  ...
  [21 17 15]
  [37 29 27]
  [29 24 21]]

 ...

 [[ 4  3  2]
  [ 4  3  2]
  [ 4  3  2]
  ...
```

```
  [33 28 25]
  [31 26 23]
  [31 26 23]]

 [[ 3  3  2]
  [ 4  3  2]
  [ 3  3  2]
  ...
  [36 31 27]
  [36 31 28]
  [33 29 26]]

 [[ 3  3  2]
  [ 3  3  3]
  [ 3  3  2]
  ...
  [30 25 23]
  [31 25 23]
  [26 22 20]]]
```