**NumPy:**

NumPy is a general-purpose array processing Python Package which stands for ‘Numerical Python’.

It is the Core Library for Scientific Computing, which contains a high performance multi-dimensional array object which is in the form of Rows and Columns.

Example 1: Single Dimensional numpy Array:

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

OUTPUT:

[1 2 3]

Example 2: Multi Dimensional numpy Array:

**import** numpy **as** np

a = np.array([(4,5,6),(7,8,9)])  
print(a)

OUTPUT:

[[4 5 6]

[7 8 9]]

**Python NumPy Array v/s List:**

1. Python numpy array occupies **Less Memory** as compared to List

2. Python numpy array is **Faster** in terms of Execution as compared to List.

3. It is very **Convenient** to work with numpy as compared to List.

**Applications of NumPy:**

1. Mathematics ( MATLAB replacement)

2. Plotting (Matplotlib)

3. Backend (Pandas, Digital Photography)

4. Machine Learning

**Python NumPy Operations:**

1. **ndim**:

We can find the axes (dimension) of the array, whether it is a single-dimensional array or a two-dimensional array.

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.ndim)

print(b.ndim)

OUTPUT:

1

2

2. **shape**:

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension.

For a matrix with n rows and m columns, **shape** will be (n,m).

The length of the shape tuple is therefore the (number of axes, ndim).

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.shape)

print(b.shape)

OUTPUT:

(3,)

(2,3)

3. **dtype:**

We can find the data type of the elements that are stored in an array

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.dtype)

print(b.dtype)

OUTPUT:

int32

float64

4. **size:**

We can find the total number of elements of an array which is equal to the **product of the elements of shape**.

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.size)

print(b.size)

OUTPUT:

3

6

5. **itemsize:**

We can find the size, in bytes, of each element of an array.

For example, an array of elements of type **float64** has **itemsize** 8 (=64/8), while one of type **complex32** has **itemsize** 4 (=32/8).

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.size)

print(b.size)

OUTPUT:

4

8

6. **nbytes:**

We can find the total size, in bytes, of an array.

**nbytes = size \* itemsize**

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.nbytes)

print(b.nbytes)

OUTPUT:

12

48

**7. reshape:**

We can change number of rows and columns of an array which gives a new view to an object.

Example:

**import** numpy **as** np

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(b)  
print(**"Shape of b -"**,b.shape)  
print(b.reshape(3,2))

OUTPUT:

[[4. 5. 6.]

[7. 8. 9.]]

Shape of b - (2, 3)

[[4. 5.]

[6. 7.]

[8. 9.]]

**8. linspace:**

This operation in Python numpy returns evenly spaced numbers over a specified interval.

Example:

**import** numpy **as** np

c = np.linspace(1,3,10)  
print(c)

OUTPUT:

[1. 1.22222222 1.44444444 1.66666667 1.88888889 2.11111111

2.33333333 2.55555556 2.77777778 3. ]

In the above output, it has printed 10 values between 1 and 3.

**9. max/min/sum:**

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

print(a.max())   
print(b.min())  
print(a+b)

OUTPUT:

3

4.0

[[ 5. 7. 9.]

[ 8. 10. 12.]]

# 'N' largest Values from a NumPy arrayc = np.array([2,6,45,100,34,69])  
print(c[np.argsort(c)[-2:][::-1]])

OUTPUT:

[69,100]

**10. Square Root and Standard Deviation:**

Example:

**import** numpy **as** np

a=np.array([1,2,3])

b=np.array([(4.0,5.0,6.0),(7.0,8.0,9.0)])

# Square Root  
print(**"Square Root of a - "**,np.sqrt(a))  
# Standard Deviation  
print(**"Standard Deviation of a - "**,np.std(a))

OUTPUT:

Square Root of a - [1. 1.41421356 1.73205081]

Standard Deviation of a - 0.81649658092772

**11. Vertical and Horizontal Stacking:**

If we want to concatenate two arrays and not just add them, we can perform it in 2 ways – Vertical Stacking and Horizontal Stacking.

Example:

**import** numpy **as** np

x = np.array([(1,2,3),(4,5,6)])  
y = np.array([(1,2,3),(4,5,6)])  
print(**"Vertical Stacking"**)  
print(np.vstack((x,y)))  
print(**"Horizontal Stacking"**)  
print( np.hstack((x,y)))

OUTPUT:

Vertical Stacking

[[1 2 3]

[4 5 6]

[1 2 3]

[4 5 6]]

Horizontal Stacking

[[1 2 3 1 2 3]

[4 5 6 4 5 6]]

12. ravel:

This operation is used to convert an array into single column.

Example:

**import** numpy **as** np

arr = np.array([(1,2,3),(4,5,6),(7,8,9)])  
print(arr)  
print(**"ravel - "**,np.ravel(arr))

OUTPUT:

[[1 2 3]

[4 5 6]

[7 8 9]]

ravel – [1 2 3 4 5 6 7 8 9]

**Accessing/Changing specific elements, rows, columns etc:**

**import** numpy **as** np

# 2-D Array:  
a = np.array([(1,2,3,4,5,6,7),(8,9,10,11,12,13,14)])  
print(a) # OUTPUT: [[ 1 2 3 4 5 6 7]  
 [ 8 9 10 11 12 13 14]]  
  
# Get a specific Element [r,c]  
print(a.shape) # OUTPUT: (2,7)  
print(a[0,-2]) # OUTPUT: 6  
print(a[1,5]) # OUTPUT: 13  
  
# Get a specific row  
print(a[0,]) # OUTPUT: [1 2 3 4 5 6 7]  
print(a[1,]) # OUTPUT: [ 8 9 10 11 12 13 14]  
  
# Get a specific column  
print(a[:,3]) # OUTPUT: [ 4 11]  
print(a[:,6]) # OUTPUT: [ 7 14]  
  
# Slicing [startindex:endindex:stepsize]  
print(a[0,1:6:2]) # OUTPUT: [2 4 6]  
print(a[1,0:5:3]) # OUTPUT: [ 8 11]  
  
# Replace  
a[1,5] = 20  
a[:,2] = [99,99]  
print(a) # OUTPUT: [[ 1 2 99 4 5 6 7]  
 [ 8 9 99 11 12 20 14]]

# 3-D Array  
b = np.array([[[1,2,1],[3,4,2]],[[6,7,3],[9,10,4]]])  
print(b) # OUTPUT: [[[ 1 2 1]  
 [ 3 4 2]]  
  
 [[ 6 7 3]  
 [ 9 10 4]]]

# Get specific element  
print(b[0,1,1]) # OUTPUT: 4  
print(b[:,1,:]) # OUTPUT: [[ 3 4 2]  
 # [ 9 10 4]]  
# Replace  
b[:,0,:]=[[5,5,5],[5,5,5]]  
print(b) # OUTPUT: [[[ 5 5 5]  
 [ 3 4 2]]  
  
 [[ 5 5 5]  
 [ 9 10 4]]]

**Initializing Different Types of Arrays:**

**import** numpy **as** np

# All 0s Matrix  
print(np.zeros(5)) # OUTPUT: [0. 0. 0. 0. 0.]  
Zeroes = np.zeros((2,3))  
print(Zeroes) # OUTPUT: [[0. 0. 0.]  
 [0. 0. 0.]]  
# All 1s Matrix  
print(np.ones((3,2),dtype=**'int32'**)) # OUTPUT: [[1 1]  
 [1 1]  
 [1 1]]  
# Any other number Syntax: np.full(shape,value)  
print(np.full((2,2),44)) # OUTPUT: [[44 44]  
 [44 44]]  
print(np.full\_like(Zeroes,5,dtype=**'int32'**)) # OUTPUT: [[5 5 5]  
 [5 5 5]]  
print(np.full(Zeroes.shape,8,dtype=**'int32'**)) # OUTPUT: [[8 8 8]  
 [8 8 8]]  
# Random Decimal Numbers  
print(np.random.rand(3,2)) # OUTPUT: [[0.11835874 0.18091045]  
 [0.30973103 0.81601807]  
 [0.72002592 0.09790571]]  
# Random Integer Values  
print(np.random.randint(7,size=(3,3))) # OUTPUT: [[5 4 0]  
 [6 5 6]  
 [0 4 1]]  
print(np.random.randint(5,9,size=(2,4))) # OUTPUT: [[8 5 7 8]  
 [7 6 5 7]]  
# The Identity Matrix  
print(np.identity(4,dtype=**'int32'**)) # OUTPUT: [[1 0 0 0]  
 [0 1 0 0]  
 [0 0 1 0]  
 [0 0 0 1]]  
# Print Output as below Matrix  
**'''  
[[1 1 1 1 1]  
 [1 0 0 0 1]  
 [1 0 9 0 1]  
 [1 0 0 0 1]  
 [1 1 1 1 1]]  
'''**Matrix = np.ones((5,5),dtype=**'int32'**)  
z = np.zeros((3,3),dtype=**'int32'**)  
z[1,1]=9  
Matrix[1:-1,1:-1] = z  
print(Matrix)

# Copying arrays  
arr1 = np.array([1,2,3])  
arr2 = arr1.copy()  
arr2[1]=55  
# Results after Copying Arrays **with copy()** Method  
print(**'arr1 - '**,arr1) # OUTPUT : arr1 - [1 2 3]  
print(**'arr2 - '**,arr2) # OUTPUT : arr2 - [1 55 3]  
arr2=arr1  
arr2[1]=99  
# Results after Copying Arrays **without copy()** Method  
print(**'arr1 - '**,arr1) # OUTPUT : arr1 - [ 1 99 3]

print(**'arr2 - '**,arr2) # OUTPUT : arr2 - [ 1 99 3]