In [1]:

```
a=[1,2,3,4,5]
print(a)
type(a)
```

```
[1, 2, 3, 4, 5]
```

Out[1]:

```
list
```

NumPy

(**Num**erical **Py**thon)

- used for working with arrays.
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy is incredible library to perform mathematical and statistical operations due to it's fast and memory efficient as it is optimized to work with latest CPU architectures.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.So processes can access and manipulate them very efficiently.

In [2]:

```
import numpy as np
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.21.5'
```

ndarray

The ndarray is created using an array function in NumPy:

```
numpy.array(object)
```

In [4]:

```
a=[1,2,3,4,5]

np_a = np.array(a)
print(np_a)
type(np_a)
```

```
[1 2 3 4 5]
```

Out[4]:

```
numpy.ndarray
```

In [5]:

```
np_a
```

Out[5]:

```
array([1, 2, 3, 4, 5])
```

Creating NumPy Arrays

0-D

In [6]:

```
arr = np.array(42)
print(arr)
type(arr)
```

```
42
```

Out[6]:

```
numpy.ndarray
```

1-D

In [7]:

```
a = np.array([0, 1, 2, 3])
print(a)
```

```
[0 1 2 3]
```

In [8]:

```
a = np.array((2,3,4,5))
a
```

Out[8]:

```
array([2, 3, 4, 5])
```

In [9]:

```
#print dimensions
a.ndim
```

Out[9]:

```
1
```

In [10]:

```
#shape
a.shape
```

Out[10]:

```
(4,)
```

Functions for creating arrays

arange

- arange is an array-valued version of the built-in Python range function
- **Syntax:** np.arange(start,end,step)

In [11]:

```
a = np.arange(10, 101, 10)
print(a)
```

```
[ 10  20  30  40  50  60  70  80  90 100]
```

linspace

- Return evenly spaced numbers over a specified interval.
- **Syntax:** np.linspace(start, end, num_of_samples)

In [12]:

```
np.linspace(0, 100, 11)
```

Out[12]:

```
array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.])
```

zeros

- Generate arrays of zeros
- **Syntax:** np.zeros(shape)

In [13]:

```
np.zeros(3)
```

Out[13]:

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

ones

- Generate arrays of ones
- **Syntax:** np.ones(shape)

In [14]:

```
np.ones(2)
```

Out[14]:

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

random.randint

- Return random integers from start (inclusive) to end (exclusive)
- **Syntax:** np.random.randint(start, end, num_of_samples)

In [15]:

```
np.random.randint(1,41,5)
```

Out[15]:

```
array([26, 23, 40, 10, 33])
```

Accessing elements

1. Accessing a single value through Indexing
2. Accessing a multiple values based on slicing
3. Accessing a multiple values based on indexing
4. Accessing a multiple values based on condition

Accessing a single value through Indexing

In [16]:

```
a = np.arange(10, 101, 10)
print(a)

a[2]
```

```
[ 10  20  30  40  50  60  70  80  90 100]
```

Out[16]:

```
30
```

Accessing a multiple values based on slicing

In [17]:

```
a[1:8:2]
```

Out[17]:

```
array([20, 40, 60, 80])
```

Accessing a multiple values based on indexing

Syntax : array[list of index numbers]

In [18]:

```
a[[0,4,4,5,7]]
```

Out[18]:

```
array([10, 50, 50, 60, 80])
```

Accessing a multiple values based on condition (Mask Indexing or Fancy Indexing)

In [19]:

```
a%2==0
```

Out[19]:

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

In [20]:

```
a[a%2==0]
```

Out[20]:

```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

Numpy is mutuable

1. Replace
2. Add
3. Remove

Replacing a single value in a array

In [21]:

```
a[5]=50
print(a)
```

```
[ 10  20  30  40  50  50  70  80  90 100]
```

Replacing a multiple values in a array

In [22]:

```
a[[1,5]] = 80
a
```

Out[22]:

```
array([ 10,  80,  30,  40,  50,  80,  70,  80,  90, 100])
```

Add values in a array

In [23]:

```
b = np.append(a,[110,120])
b
```

Out[23]:

```
array([ 10,  80,  30,  40,  50,  80,  70,  80,  90, 100, 110, 120])
```

Delete a single value in a array

In [24]:

```
b = np.delete(a,8)
print('Numpy array after deletion of one element:',a)
```

```
Numpy array after deletion of one element: [ 10  80  30  40  50  80  70  80  90 100]
```

Delete a multiple values in a array

In [25]:

```
b = np.delete(a,[1,2])
b
```

Out[25]:

```
array([ 10,  40,  50,  80,  70,  80,  90, 100])
```

sort

In [26]:

```
np.sort([4,6,8,2,1])
```

Out[26]:

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

Copy array

In [27]:

```
a = np.array([1,2,3])
b = a.copy()
b[0] = 100
print(a)
print(b)
```

```
[1 2 3]
[100   2   3]
```

around()

In [28]:

```
arr=np.array([1.432, 3.087, 4.56])
np.around(arr)
```

Out[28]:

```
array([1., 3., 5.])
```

In [29]:

```
np.around(arr,2)
```

Out[29]:

```
array([1.43, 3.09, 4.56])
```

Operations

Arthimetic Oprations

In [30]:

```
a = np.array([1, 2, 3, 4])
a
```

Out[30]:

```
array([1, 2, 3, 4])
```

In [31]:

```
a+1
```

Out[31]:

```
array([2, 3, 4, 5])
```

In [32]:

```
a**2
```

Out[32]:

```
array([ 1,  4,  9, 16], dtype=int32)
```

In [33]:

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

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

In [34]:

```
a+b
```

Out[34]:

```
array([ 6,  8, 10, 12])
```

In [35]:

```
a-b
```

Out[35]:

```
array([-4, -4, -4, -4])
```

In [36]:

```
a*b
```

Out[36]:

```
array([ 5, 12, 21, 32])
```

Comparision operations

In [37]:

```
a == b
```

Out[37]:

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

In [38]:

```
a > b
```

Out[38]:

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

In [39]:

```
np.array_equal(a, b)
```

Out[39]:

```
False
```

In [40]:

```
np.array_equal(a, c)
```

Out[40]:

```
True
```

Mathematical operations

In [41]:

```
a = np.arange(0,6)
a
```

Out[41]:

```
array([0, 1, 2, 3, 4, 5])
```

In [42]:

```
np.sin(a)
```

Out[42]:

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427])
```

In [43]:

```
np.cos(a)
```

Out[43]:

```
array([ 1.        ,  0.54030231, -0.41614684, -0.9899925 , -0.65364362,
        0.28366219])
```

In [44]:

```
np.log(a)
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_6920\176755284.py:1: RuntimeWarning: divide by zero encountered in log
  np.log(a)
```

Out[44]:

```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791])
```

In [45]:

```
np.exp(a)
```

Out[45]:

```
array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,
        54.59815003, 148.4131591 ])
```

Statistical Operations

In [46]:

```
np.mean(a)
```

Out[46]:

```
2.5
```

In [47]:

```
np.median(a)
```

Out[47]:

```
2.5
```

In [48]:

```
np.std(a)
```

Out[48]:

```
1.707825127659933
```

2D Numpy Array

In [49]:

```
b = np.array([[0, 1, 2], [3, 4, 5]])
print(b)
```

```
[[0 1 2]
 [3 4 5]]
```

In [50]:

```
b.ndim
```

Out[50]:

```
2
```

In [51]:

```
b.shape
```

Out[51]:

```
(2, 3)
```

In [52]:

```
np.zeros((2,2))
```

Out[52]:

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

In [53]:

```
np.ones((2,2))
```

Out[53]:

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

In [54]:

```
np.eye(2,2)
```

Out[54]:

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

In [55]:

```
np.diag([1, 2, 3])
```

Out[55]:

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [56]:

```
a = np.array([[10,20],[30,40]])
print(a)
print(a[1,1])
```

```
[[10 20]
 [30 40]]
40
```

In [57]:

```
#assigning value in 2d array
a[1,1] = 25
a
```

Out[57]:

```
array([[10, 20],
       [30, 25]])
```

Flattening : Converting nd array to 1D array

In [58]:

```
a = np.array([[0, 1, 2], [3, 4, 5]])
a.ravel()
```

Out[58]:

```
array([0, 1, 2, 3, 4, 5])
```

Reshape array

In [59]:

```
arr = np.arange(6)
print(arr)
print(arr.shape)
```

```
[0 1 2 3 4 5]
(6,)
```

In [60]:

```
a1=arr.reshape(2,3)
a1
```

Out[60]:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

Transpose array

In [61]:

```
c = a.T
print(c)
```

```
[[0 3]
 [1 4]
 [2 5]]
```

Sort

In [62]:

```
#Sorting along an axis: axis=1 --> row and axis=0 -->column
a = np.array([[5, 4, 6], [2, 8, 2]])
b = np.sort(a,axis=0)
b
```

Out[62]:

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

In [63]:

```
a = np.array([[1,2],[3,4],[5,6]])
b = np.array([[4,4],[4,4],[4,4]])

# adding a and b
print(a+b)
```

```
[[ 5  6]
 [ 7  8]
 [ 9 10]]
```

In [64]:

```
# multiply a and b - elementwise multiplication
print(a*b)
```

```
[[ 4  8]
 [12 16]
 [20 24]]
```

In [65]:

```
# matrix multiplication
a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
np.matmul(a,b)
```

Out[65]:

```
array([[19, 22],
       [43, 50]])
```