Numerical Python - Numpy

1. To provide high performance multidimensional array

2. Numpy vs List
   1. Numpy occupies less memory compared to List
   2. Numpy is Faster than List
   3. Numpy is easier than List

3. Numpy Array programs
   1. Initialization of Array
      - Syntax - Single Dimension

```
import numpy as np
a = np.array([7,9,9,2])
print(a)
```

```
[7 9 9 2]
```

      - Syntax-Multi Dimension

```
import numpy as np
b = np.array([(7,9,9,2),(16,11,9,6)])
print(b)
```

```
[[ 7  9  9  2]
 [16 11  9  6]]
```

      - Range of Values in array

```
c = np.arange(10) #range of values in array
d = np.linspace(1,5,6) #set of values by Interval [start,end,no.ofvaluesEqualInterval]
print(c)
print(d)
```

```
[0 1 2 3 4 5 6 7 8 9]
[1.  1.8 2.6 3.4 4.2 5. ]
```

   2. Properties of Array
      - Syntax

```
a = np.array([7,9,9,2])
b = np.array([(7,9,9,2),(16,11,9,6)])
print(a.ndim) #Dimension of Array
print(a.itemsize) #Byte Size of each Element
print(a.dtype) #Datatype of array
print(a.size*a.itemsize) #memory occupied - space occupied by 1 element * length of numpy array
print(a.size) #number of elements in array
print(b.shape) #shape of array (rows and columns)
```

```
1
8
int64
32
4
(2, 4)
```

   3. Array Reshape
      - Syntax

```
b = np.array([(7,9,9,2),(16,11,9,6)])
print(b)
b = b.reshape(4,2)
print(b)
```

```
[[ 7  9  9  2]
 [16 11  9  6]]
[[ 7  9]
 [ 9  2]
 [16 11]
 [ 9  6]]
```

   4. Array Slicing
      - Syntax

```
b = np.array([(7,9,9,2),(16,11,9,6)])
print(b[0,2]) #Slicing the Element from certain row and certain column b[rowIndex,ColumnIndex]
print(b[0:,2]) #Slicing Element from all rows and certain column
```

```
9
[9 9]
```

   5. Array Max,Min&Sum Finding
      - Syntax

```
e = np.array([7,9,9,2,16,11,96])
print(e.min()) #Minimum value in array
print(e.max()) #Maximum value in array
print(e.sum()) #Sum of array
```

```
2
96
150
```

   6. Arithmetic Operation
      - Syntax

```
a = np.array([7,9,9,2])
c = np.array([1,9,9,6])
print(a+c)
print(a-c)
print(a*c)
print(a/c)
```

```
[ 8 18 18  8]
[ 6  0  0 -4]
[ 7 81 81 12]
[7.         1.         1.         0.33333333]
```

      - Syntax

```
a = np.array([7,9,9,2])
b = np.array([(7,9,9,2),(16,11,9,6)])
print(b.sum(axis=0)) #Each element of 1st dim array with same index with another dim element
print(b.sum(axis=1)) #sum of all element is 1st dim & sum of all element is 2nd dim
print(np.sqrt(a)) #Squareroot of array elements
print(np.std(a)) #Standard Deviation of array
print(np.exp(a)) #Exponential of array
print(np.log(a)) #Natural Log - ln
print(np.log10(a)) #Log base 10
```

```
[23 20 18  8]
[27 42]
[2.64575131 3.         3.         1.41421356]
2.8613807855648994
[1.09663316e+03 8.10308393e+03 8.10308393e+03 7.38905610e+00]
[1.94591015 2.19722458 2.19722458 0.69314718]
[0.84509804 0.95424251 0.95424251 0.30103   ]
```

   7. Array Stacking
      - Syntax

```
b = np.array([(7,9,9,2),(16,11,9,6)])
f = np.array([(1,2,3,4),(5,6,7,8)])
print(np.vstack((b,f))) #Vertical Stack
print(np.hstack((b,f))) #Horizontal Stack
print(b.ravel()) #Merging as single
```

```
[[ 7  9  9  2]
 [16 11  9  6]
 [ 1  2  3  4]
 [ 5  6  7  8]]
[[ 7  9  9  2  1  2  3  4]
 [16 11  9  6  5  6  7  8]]
[ 7  9  9  2 16 11  9  6]
```

      - Syntax