Introduction to NumPy

- NumPy is a Python package and it stands for numerical python
- Fundamental package for numerical computations in Python
- Supports N-dimensional array objects that can be used for processing multidimensional data
- Supports different data-types

Array

- An array is a data structure that stores values of same data type
- Lists can contain values corresponding to different data types,
- Arrays in python can only contain values corresponding to same data type

NumPy Array

- A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers
- The number of dimensions is the rank of the array
- The shape of an array is a tuple of integers giving the size of the array along each dimension

Creation of array

In [1]:

```
my_list = [1,2,3,4,5,6]
print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

To create numpy array, we first need to import the numpy package:

In [3]:

```
import numpy as np
```

In [4]:

```
array   = np.array(my_list, dtype = int)
print(array)
```

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

In [5]:

```
print(type(array))
print(len(array))
print(array.ndim)
print(array.shape)
```

```
<class 'numpy.ndarray'>
6
1
(6,)
```

In [6]:

```
array2 = array.reshape(3,2)
print(array2)
array2.shape
```

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

Out[6]:

```
(3, 2)
```

In [7]:

```
array3 = array.reshape(3,-1)
print(array3)

print(array3.ndim)
```

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

In [8]:

```
## Initializing numpy arrays from nested Python lists

my_list2 = [1,2,3,4,5]
my_list3 = [2,3,4,5,6]
my_list4 = [9,7,6,8,9]

mul_arr  = np.array([my_list2, my_list3,my_list4])
print(mul_arr)

print(mul_arr.shape)
```

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

In [9]:

```
mul_arr.reshape(1,15)
```

Out[9]:

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

NumPy-Attributes

In [10]:

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

```
(2, 3)
```

In [35]:

```
# reshaping the ndarray
a.shape = (3,2)
print(a)
```

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

In [11]:

```
# Reshape function to resize an array
b = a.reshape(3,2)
print(b)
```

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

In [13]:

```
r = range(24)
```

In [14]:

```
print(r)
```

```
range(0, 24)
```

In [15]:

```
# an array of evenly spaced numbers
a = np.arange(24)
print(a)
print(a.ndim)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
1
```

In [16]:

```
# Reshaping the array 'a'
b = a.reshape(6,4,1)
print(b)
```

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

 [[ 4]
  [ 5]
  [ 6]
  [ 7]]

 [[ 8]
  [ 9]
  [10]
  [11]]

 [[12]
  [13]
  [14]
  [15]]

 [[16]
  [17]
  [18]
  [19]]

 [[20]
  [21]
  [22]
  [23]]]
```

numpy.itemsize

This array attribute returns the length of each element of array in bytes.

In [17]:

```
# dtype of array is int8 (1 byte)
x = np.array([1,2,3,4,5], dtype = np.int8)
print(x.itemsize)
```

```
1
```

In [18]:

```
# dtype of array is now float32 (4 bytes)
x = np.array([1,2,3,4,5], dtype = np.float32)
print(x.itemsize)
```

```
4
```

NumPy Arithmetic operations

In [19]:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
print(x)
print(y)
```

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

In [20]:

```
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [21]:

```
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

In [22]:

```
print(x * y)
print(np.multiply(x, y))
print(x.dot(y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[19. 22.]
 [43. 50.]]
```

In [23]:

```
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19. 22.]
 [43. 50.]]
[[19. 22.]
 [43. 50.]]
```

In [24]:

```
print(x / y)
print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

In [26]:

```
print(np.sum(x))          # Compute sum of all elements
print(np.sum(x, axis=0))  # Compute sum of each column
print(np.sum(x, axis=1))  # Compute sum of each row
```

```
10.0
[4. 6.]
[3. 7.]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: