Working with NumPy

Outcomes

At the end of this chapter, you will be able to

- Define NumPy data types
- Create and work with NumPy Arrays
- Understand the applications of NumpPy

What is Numpy?

- A Python package for scientific computations
- NumPy = ***Numerical Python***
- Numpy Library = Multi-dimensional array objects & Functions for processing it.
- Multi-dimensional object --> ***ndarray*** object
- Variaous operations supported: Arithmetic, Logical, Shape manipulation, Sort, DFT, Statistics, Random simulation etc.
- Importing NumPy: ***import numpy as np***

In [2]:

```
# import numpy
import numpy as np
```

Creating and Accessing ndarray object

- ndarray: Collection of items of the same type
- Items can be accessed using zero-based index.
- array() Function: ***numpy.array(object)***

```
--> Creates an ndarray from an object.
--> object: Any array or sequence.
```

One Dimensional Array (Vector):

In [2]:

```
# Create an one dimensional ndarray object from a list, display its values and type
a1 = np.array([1, 2, 3])
print(a1)
print(type(a1))
```

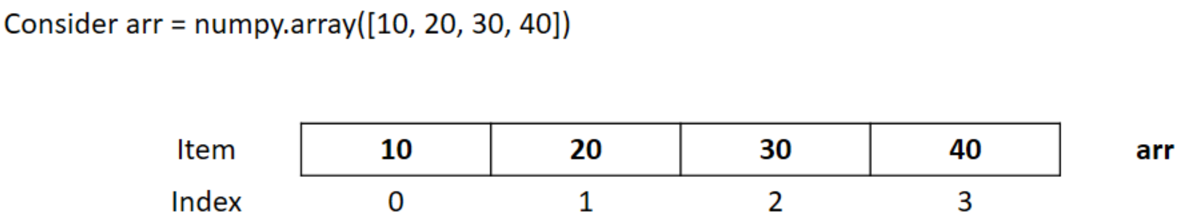
```
[1 2 3]
<class 'numpy.ndarray'>
```

In [3]:

```
# Create an one dimensional ndarray object from a tuple, display its values and type
a1 = np.array((1, 2, 3))
print(a1)
print(type(a1))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

Indexing & Slicing:



Consider arr = numpy.array([10, 20, 30, 40])

Item	**10**	**20**	**30**	**40**	**arr**
Index	0	1	2	3	

In [5]:

```
# Create above one dimensional ndarray object and access its elements individually. And also slice some of its elements.
arr = np.array([10, 20, 30, 40])
print(arr[0])
print(arr[1])
print(arr[2])
print(arr[3])
print(arr[-1])
print(arr[1:3])
```

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

Two-Dimensional Arrays(Matrices)

- Each element is itself 1D array
- arr2 = numpy.array([[row1], [row2], ..., [rowM]])
- Accessing an element: arr2[row_index, column_index]

In [6]:

```
# Create a two dimensional array and access their rows and columns individually.
# Access the individual elements.
arr2 = np.array([[10, 20], [30, 40], [50, 60]])
print(arr2)
print("Row 1:",arr2[0])
print("Row 2:",arr2[1])
print("Row 3:",arr2[2,:])
print("Column 1:",arr2[:,0])
print("Column 2:",arr2[:,1])
print("Fisrst two rows:\n",arr2[0:2,:])
print(arr2[2,1])
```

```
[[10 20]
 [30 40]
 [50 60]]
Row 1: [10 20]
Row 2: [30 40]
Row 3: [50 60]
Column 1: [10 30 50]
Column 2: [20 40 60]
Fisrst two rows:
 [[10 20]
 [30 40]]
60
```

ndarray Attribtes

- ndarray.ndim: Number of dimensions
- ndarry.shape: Array dimensions as a tuple.
- ndarry.size: Total number of elements
- ndarry.dtype: Type of the array elements.
- ndarray.itemsize: The size of each element in bytes

In [7]:

```
# Create an one dimensional ndarray object and diplay all its attribute values.
a = np.array([1, 2, 3])
print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype)
print(a.itemsize)
```

```
1
(3,)
3
int32
4
```

In [8]:

```
# Create a two dimensional ndarray object and diplay all its attribute values.
b = np.array([[1, 2, 3],[4, 5, 6]])
print(b.ndim)
print(b.shape)
print(b.size)
print(b.dtype)
print(b.itemsize)
```

```
2
(2, 3)
6
int32
4
```

Other NumPy array creation functions

zeros(shape)

- Returns an array of zeros with specified shape
- Dafalut dtype is float

ones(shape)

- Returns an array of ones with specified shape
- Dafalut dtype is float

arange(start, stop, step, dtype)

- Returns an ndarray object with elements decided by start, stop and step arguments.
- Note that stop value is not including in the array.
- Default dtype is type is determined by type of start or stop values.

linspace(start, stop, N, dtype)

- Returns an ndarray object with N elements in the range from start value to stop stop .
- Note that stop value is inclusive in the array.
- Defaul type is float64

In [9]:

```
# Create ndarray objects with all zeros using different dimensions and dtypes.
z1 = np.zeros(5)
print(z1)

z2 = np.zeros((2,3))
print(z2)

z3 = np.zeros((2,3), dtype = np.uint8)
print(z3)
```

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

In [10]:

```
# Create ndarray objects with all ones using different dimensions and dtypes.
o1 = np.ones(5)
print(o1)

o2 = np.ones((2,3))
print(o2)

o3 = np.ones((2,3), dtype = np.int64)
print(o3)
```

```
[1. 1. 1. 1. 1.]
[[1. 1. 1.]
 [1. 1. 1.]]
[[1 1 1]
 [1 1 1]]
```

In [11]:

```
# Create one dimensional ndarray objects with various range of values and dtypes.
r1 = np.arange(10)
print(r1)
print(r1.dtype)

r2 = np.arange(10.0)
print(r2)
print(r2.dtype)

r3 = np.arange(10, 101, 10)
print(r3)

r4 = np.arange(0, 11, 2)
print(r4)
```

```
[0 1 2 3 4 5 6 7 8 9]
int32
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
float64
[ 10  20  30  40  50  60  70  80  90 100]
[ 0  2  4  6  8 10]
```

In [8]:

```
# Create one dimensional ndarray objects with 25 elements between 1 and 100 of different dtypes.
l1 = np.linspace(1, 100, 25, dtype = np.uint8)
print(l1)
```

```
[  1   5   9  13  17  21  25  29  34  38  42  46  50  54  58  62  67  71
  75  79  83  87  91  95 100]
```