PYTHON PROGRAMMING

UNIT – III

Arrays: Advantages of Arrays, Creating an Array, Importing the Array Module, Indexing and Slicing on Arrays, Types of arrays, working with arrays using numpy.

Arrays:

- An array is defined as collection of similar data items that are stored at contiguous locations. In Python, they are containers which are able to store more than one item at the same time.

Differences between Python Lists and Python Arrays

- Lists are one of the most common data structures in Python, and a core part of the language.
- Lists and arrays behave similarly.
- Just like arrays, lists are an ordered sequence of elements.
- They are also mutable and not fixed in size, which means they can grow and shrink throughout the life of the program. Items can be added and removed, making them very flexible to work with.

- However, lists and arrays are not the same thing. Lists store items that are of various data types. That is not the case with arrays.
- Arrays store only items that are of the same data type

Advantages of Arrays:

- Arrays can store only one type of data
- In python, the size of array is not fixed. Array can increase or decrease their size dynamically
- Array uses less memory than List

Creating an Array :

- Lists are built into the Python programming language, whereas arrays aren't.

- Arrays are not a built-in data structure, and therefore need to be imported via the array module in order to be used

There are three ways you can import the array module:

1. By using import array at the top of the file. This includes the module array.

```
import array
array.array()              #creating an array
```

2. Instead of typing array.array() all the time, you could use import array as arr at the top of the file, arr acts like alias name

```
import array as arr
arr.array()                #creating an array
```

3. Lastly, you could also use from array import * , with * importing all the functionalities available.

```
from array import *
array()                    #creating an array
```

Once the array module is imported, we can then go onto <u>define a python array</u>

- The general syntax for creating an array is

  variable_name = array(typecode,[elements])

- variable_name would be the name of the array.

- The typecode specifies what kind of elements would be stored in the array. Whether it would be an array of integers, an array of floats or an array of any other Python data type

- Inside square brackets you mention the elements that would be stored in the array

The different type codes that can be used with the different data types when defining Python arrays:

TYPECODE	C TYPE	PYTHON TYPE	SIZE
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	wchar_t	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

Example of how to define an array in Python:

```
import array as arr
numbers = arr.array('i',[10,20,30])
print(numbers)
```

output : array('i', [10, 20, 30])

Example of how to create an array numbers of float data type.

```
from array import *
numbers = array('d',[10.0,20.0,30.0])
print(numbers)
```

output : array('d', [10.0, 20.0, 30.0])

Array Indexing

- Each item in an array has a specific address. Individual items are accessed by referencing their index number.

- Indexing in Python, and in all programming languages and computing in general, starts at 0.

- The index value of the last element of an array is always one less than the length of the array. Where n is the length of the array, n - 1 will be the index value of the last item

Syntax:

```
array_name[index_value_of_item]
```

```
import array as arr
numbers = arr.array('i',[10,20,30])
print(numbers[0])          # gets the 1st element
print(numbers[1])          # gets the 2nd element
print(numbers[2])          # gets the 3rd element
```

output

10
20
30

We can also access each individual element using negative indexing. With negative indexing, the last element would have an index of -1, the second to last element would have an index of -2, and so on.

```
import array as arr
numbers = arr.array('i',[10,20,30])
print(numbers[-1])          #gets last item
print(numbers[-2])          #gets second to last item
print(numbers[-3])          #gets first item
```

output

30
20
10

How to Slice an Array in Python

- To access a specific range of values inside the array, use the slicing operator, which is a colon :

- When using the slicing operator and you only include one value, the counting starts from 0 by default. It gets the first item, and goes up to but not including the index number you specify

```
import array as arr
numbers = arr.array('i',[10,20,30])
print(numbers[:2])
```

Output : array('i', [10, 20])

```
import array as arr
numbers = arr.array('i',[10,20,30])
print(numbers[1:3])
```

Output : array('i', [20, 30])

Array Types in Python

- When talking about arrays, any programming language like C or Java offers two types of arrays. They are:

- **Single dimensional arrays:** These arrays represent only one row or one column of elements. For example, marks obtained by a student in 5 subjects can be written as 'marks' array, as:

  marks = array('i', [50, 60, 70, 66, 72])

- The above array contains only one row of elements. Hence it is called single dimensional array or one dimensional array.

- **Multi-dimensional arrays:** These arrays represent more than one row and more than one column of elements. For example, marks obtained by 3 students each one in 5 subjects can be written as 'marks' array as:

- marks = [[50, 60, 70, 66, 72], [60, 62, 71, 56, 70], [55, 59, 80, 68, 65]]

- In Python, we can create and work with single dimensional arrays only. So far, the examples and methods discussed by us are applicable to single dimensional arrays.

- But, Python does not support multi-dimensional arrays. We can construct multidimensional arrays using third party packages like **numpy (numerical python)**

Why Use NumPy?

- In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists and occupies less memory when compared to python list..

- Arrays are very frequently used in data science, where speed and resources are very important.

numpy

- Numpy is the core library for scientific and numerical computing in Python. It provides high performance multi dimensional array object and tools for working with arrays

- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

- We can create a NumPy ndarray object by using the array() function.

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

Output : [1 2 3 4 5]
<class 'numpy.ndarray'>

- NumPy is usually imported under the np alias.

  import numpy as np

- Now the NumPy package can be referred to as np instead of numpy.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Output :

[1 2 3 4 5]

To install numpy

```
pip  install numpy
```

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Use a tuple to create a NumPy array:

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

Output :

[1 2 3 4 5]

Dimensions in Arrays:

- In Numpy dimensions are called as axes.

1-D Arrays : These are the most common and basic arrays.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Output: [1 2 3 4 5]

2-D Arrays : An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

Output : [[1 2 3]
[4 5 6]]

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.

#Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

Output :

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

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

Attributes of an Array

ndim - Check Number of Dimensions

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Output :

```
0
1
2
3
```

shape - Shape of an Array

- The shape of an array returns the number of elements in each dimension.

- NumPy arrays have an attribute called **shape** that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
b = np.array([[1, 2, 3], [4, 5, 6],[7,8,9]])

print(a.shape)
Print(b.shape)
```

Output : (2, 4)
(3, 3)

Size Attribute

The size attribute gives the total number of elements in the array.

Example:

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a.size)
```

Output:

6

itemsize Attribute

The itemsize attribute gives the memory size of the array elements in bytes

Example:

```
import numpy as np
a=np.array([1,2,3,4,5])
print(a.itemsize)
```

Output : 4

```
from numpy import *
a=array([1.1,2.2,3.3,4.4])
print(a.itemsize)
```

Output : 8

dtype - Checking the Data Type of an Array

The NumPy array object has a property called **dtype** that returns the data type of the array:

```
import numpy as np

a = np.array([[1, 2, 3],[4,5,6]])
b = np.array([1.7,6.4,9.6])

print(a.dtype)
print(b.dtype)
```

Output :

```
Int64
float64
```

Methods on Numpy Arrays:

numpy.arrange()

It creates an array by using the evenly spaced values over the given interval. The interval mentioned is half opened i.e. [Start, Stop]).

Syntax

numpy.arrange(start, stop, step, dtype)

```
import numpy as np
a = np.arange(0,10)
print(a)
```

Output: [0 1 2 3 4 5 6 7 8 9]

```
import numpy as np
arr = np.arange(0,10,2)
print(arr)
```

Output : [0 2 4 6 8]

```
import numpy as np
arr = np.arange(0,10,2,float)
print(arr)
```

Output : [0. 2. 4. 6. 8.]

```
import numpy  as  np
arr = np.arange(1.5,5.5)
print(arr)
```

Output : [1.5 2.5 3.5 4.5]

Reshaping arrays

- Reshaping means changing the shape of an array. By reshaping we can add or remove dimensions or change number of elements in each dimension.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Output :

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Reshape From 1-D to 3-D

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

b = a.reshape(2, 3, 2)

print(b)
```

Output:

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
b = a.flatten()
print(b)
```

Output : [1 2 3 4 5 6]

We can also use reshape(-1) to do this.

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
b = a.reshape(-1)
print(b)
```

The ones() and Zeros() Functions

- The ones() function is useful to create 2D array with several rows and c columns where all the lements will be taken as 1.

Syntax: ones((r,c),dtype)

Example : a = ones((3,4),float)
print(a)

- In the above example we can see the array as

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

Zeros()

- Just like ones() function, we can also use the zeros() function to create a 2D array with zero elements

Syntax: **zeros**((r,c),dtype)

Example : b = zeros((3,4) , int)
print(b)

- In the above example we can see the array as

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

```
In [63]: np.ones((3,2,2),dtype = int)

Out[63]: array([[[1, 1],
                 [1, 1]],

                [[1, 1],
                 [1, 1]],

                [[1, 1],
                 [1, 1]]])

In [65]: np.zeros((2,3,1),object)

Out[65]: array([[[0],
                 [0],
                 [0]],

                [[0],
                 [0],
                 [0]]], dtype=object)
```

Sorting

The NumPy ndarray object has a function called sort(), that will sort a specified array.

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

Output : [0 1 2 3]

Indexing

```
import numpy as np
a = np.array([[14, 23, 31], [48, 55, 67],[71,82,93]])
print(a[1,2])
print(a[0,1])
```

Output : 67
23

concatenate

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

Output :

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

Transpose

```
import numpy as np
a = np.array([[1,2,7],[3,4,6]])
print(a.T)
```

Output :

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

Math (Arithmetic Functions)

```
import numpy as np

a = np.array([[4,3],[7,5]])

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

print(a+b)

print(a-b)

Print(a%b)

Print(a/b)

Print(a//b)
```

Output:

```
[[ 5  5]
 [10  9]]

[[3 1]
 [4 1]]



[[0 1]
 [1 1]]

[[4.         1.5       ]
 [2.33333333 1.25      ]]

[[4 1]
 [2 1]]
```

Random Concept

- A random number is a number that cannot be guessed by anyone. When a random number is generated, we do not know which number we are going to get.

- numpy has a sub module called random that is equipped with **rand()** function that is useful to get random numbers.To call this function we use **random.rand()**

- When we call rand() function, it will generate a random number between 0.0 to 1.0

Example :

```
from numpy import random
a = random.rand()
print(a)
```

Output : 0.0388975984272591

```
a = random.rand(5)
print(a)
```

- This will create a 1D array ‘a’ with 5 elements which are random numbers between 0.0 to 1.0 as

```
[0.98742525  0.95504435  0.85397834  0.54731165  0.70156094]
```

- We can also create 2D array of random numbers

```
from numpy import random
a=random.rand(2,3)
print(a)
```

Output:

```
[[0.33374606  0.97211924  0.61379747]
 [0.07321675  0.96438621  0.22315058]]
```

Examples on random concept

```
In [109]: from numpy import random

In [132]: random.randint(100) # generate a random integer from 0 to 100
Out[132]: 32

In [120]: random.randint(10,100,size = 11)
Out[120]: array([11, 37, 74, 15, 32, 19, 62, 24, 77, 88, 39])

In [133]: random.rand() # 0 to 1
Out[133]: 0.07132085658417153

In [121]: a = random.rand(10)
          a
Out[121]: array([0.31442079, 0.42479701, 0.46666873, 0.91499173, 0.34568887,
                 0.16933889, 0.14767002, 0.18532065, 0.89948857, 0.60589583])
```

In [121]:
```
a = random.rand(10)
a
```

Out[121]:
```
array([0.31442079, 0.42479701, 0.46666873, 0.91499173, 0.34568887,
       0.16933889, 0.14767002, 0.18532065, 0.89948857, 0.60589583])
```

In [122]:
```
a*10
```

Out[122]:
```
array([3.14420792, 4.24797012, 4.66668729, 9.14991733, 3.45688868,
       1.69338887, 1.47670016, 1.85320646, 8.99488568, 6.05895833])
```

In [123]:
```
a+10
```

Out[123]:
```
array([10.31442079, 10.42479701, 10.46666873, 10.91499173, 10.34568887,
       10.16933889, 10.14767002, 10.18532065, 10.89948857, 10.60589583])
```

In [124]:
```
a+a
```

Out[124]:
```
array([0.62884158, 0.84959402, 0.93333746, 1.82998347, 0.69137774,
       0.33867777, 0.29534003, 0.37064129, 1.79897714, 1.21179167])
```