Class XII (Informatics Practices)

Numpy

Array

We have learnt about various data types like list, tuple, and dictionary. In this chapter we will discuss another datatype 'Array'. An array is a data type used to store multiple values using a single identifier (variable name). An array contains an ordered collection of data elements
where each element is of the same type and can be referenced by its index (position).
The important characteristics of an array are:

- Each element of the array is of same data type, though the values stored in them may be different.
- The entire array is stored contiguously in memory. This makes operations on array fast.
- Each element of the array is identified or referred using the name of the Array along with the index of that element, which is unique for each element. The index of an element is an integral value associated with the element, based on the element's position in the array.

For example consider an array with 5 numbers: [10, 9, 99, 71, 90]
Here, the 1st value in the array is 10 and has the index value [0] associated with it; the 2nd value in the array is 9 and has the index value [1] associated with it, and so on. The last value (in this case the 5th value) in this array has an index [4]. This is called zero based indexing. This is very similar to the indexing of lists in Python. The idea of arrays is so important that almost
all programming languages support it in one form or another.

NumPy Array NumPy arrays are used to store lists of numerical data,vectors and matrices. The NumPy library has a large set of routines (built-in functions) for creating, manipulating, and transforming NumPy arrays. Python language also has an array data structure, but it is not as versatile,
efficient and useful as the NumPy array. The NumPy Contiguous memory allocation:
The memory space must be divided into the fined sized position and each position is allocated to a single data only.
Now Contiguous Memory Allocation:
Divide the data into several blocks and place in different parts of the memory according to the availability of memory.
array is officially called ndarray but commonly known as array.
NumPy array whenever we use "array". following are few differences between list and Array.

Difference Between List and Array

List can have elements of different data types for example, [1,3.4, 'hello', 'a@']
All elements of an array are of same data type for example, an array of floats may be: [1.2, 5.4, 2.7]
Elements of a list are not stored contiguously in memory. Array elements are stored in contiguous memory locations. This makes operations on arrays faster than lists.

Lists do not support element wise operations, for example, addition, multiplication, etc. because elements may not be of same type.
Arrays support element wise operations. For example, if A1 is an array, it is possible to say A1/3 to divide each element of the array by 3.
Lists can contain objects of different datatype that Python must store the type information for every element along with its
element value. Thus lists take more space in memory and are less efficient.
NumPy array takes up less space in memory as
compared to a list because arrays do not require to store datatype of each element separately.
List is a part of core Python. Array (ndarray) is a part of NumPy library.

Creation of NumPy Arrays from List

There are several ways to create arrays. To create an array and to use its methods, first we need to import the NumPy library.

```
#NumPy is loaded as np (we can assign any
#name), numpy must be written in lowercase
>>> import numpy as np
```

The NumPy's array() function converts a given list
into an array. For example,

```
#Create an array called array1 from the
#given list.
>>> array1 = np.array([10,20,30])
#Display the contents of the array
>>> array1
array([10, 20, 30])
```

• Creating a 1-D Array

An array with only single row of elements is called 1-D array. Let us try to create a 1-D array from a list which contains numbers as well as strings.

```
>>> array2 = np.array([5,-7.4,'a',7.2])
>>> array2

array(['5', '-7.4', 'a', '7.2'], dtype='<U32')
```

Observe that since there is a string value in the list, all integer and float values have been promoted to string, while converting the list to array.

Note: U32 means Unicode-32 data type.

• Creating a 2-D Array

We can create a two dimensional (2-D) arrays by passing nested lists to the array() function.

Example

```
>>> array3 = np.array([[2.4,3],
> > > a r r a y 3 [4.91,7],[0,-1]]) a r r a y ( [[[ 24..49 1,, 37.. ]],, [ 0. , -1. ]])
```

Observe that the integers 3, 7, 0 and -1 have been promoted to floats.

Attributes of NumPy Array

Some important attributes of a NumPy ndarray object are:

i) ndarray.ndim: gives the number of dimensions of the array as an integer value. Arrays can be 1-D, 2-D or n-D. In this chapter, we shall focus on 1-D and 2-D arrays only. NumPy calls the dimensions as axes (plural of axis). Thus, a 2-D array has two axes. The row-axis is called axis-0 and the column-axis is called axis-1. The number of axes is also called the array's rank.

Example

```
>>> array1.ndim
1
>>> array3.ndim
2
```

ii) ndarray.shape: It gives the sequence of integers indicating the size of the array for each dimension.

Example

```
# array1 is 1D-array, there is nothing
# after , in sequence
>>> array1.shape
(3,)
>>> array2.shape
(4,)
>>> array3.shape(3, 2)
```

A common mistake occurs while passing argument to array() if we forget to put square brackets. Make sure only a single argument containing list of values is passed.

```
#incorrect way
>>> a =
np.array(1,2,3,4)
#correct way
>>> a =
np.array([1,2,3,4])
```

A list is called nested list when each element is a list itself.

The output (3, 2) means array3 has 3 rows and 2 columns.

iii) ndarray.size: It gives the total number of elements of the array. This is equal to the product of the elements of shape.

Example

```
>>> array1.size
3
>>> array3.size
6
```

iv) ndarray.dtype: is the data type of the elements of the array. All the elements of an array are of same data type. Common data types are int32, int64, float32, float64, U32, etc.

Example

```
>>> array1.dtype
dtype('int32')
>>> array2.dtype
dtype('<U32>')
>>> array3.dtype
dtype('float64')
```

v) ndarray.itemsize: It specifies the size in bytes of each element of the array. Data type int32 and float32 means each element of the array occupies 32 bits in memory. 8 bits form a byte. Thus, an array of elements of type int32 has itemsize 32/8=4 bytes. Likewise, int64/float64 means each item has itemsize 64/8=8 bytes.

Example

```
>>> array1.itemsize
4 # memory allocated to integer
```

```
>>> array2.itemsize
128 # memory allocated to string
>>> array3.itemsize
8 #memory allocated to float type
```

Other Ways of Creating NumPy Arrays

1. We can specify data type (integer, float, etc.) while creating array using dtype as an argument to array(). This will convert the data automatically to the mentioned type. In the following example, nested list of integers are passed to the array function. Since data type has been declared as float, the integers are converted to floating point numbers.

```
>>> array4 = np.array( [ [1,2], [3,4] ],dtype=float)
>>> array4
array([[1., 2.],[3., 4.]])
```

2. We can create an array with all elements initialized to 0 using the function zeros(). By default, the data type of the array created by zeros() is float. The following code will create an array with 3 rows and 4 columns with each element set to 0.

```
>>> array5 = np.zeros((3,4))
>>> array5
array([[0., 0., 0., 0.],
[0., 0., 0., 0.],
[0., 0., 0., 0.]])
```

3. We can create an array with all elements initialized to 1 using the function ones(). By default, the data type of the array created by ones() is float. The following code will create an array with 3 rows and 2 columns.

```
>>> array6 = np.ones((3,2))
>>> array6
array([[1., 1.],[1., 1.],[1., 1.]])
```

4. We can create an array with numbers in a given range and sequence using the arange() function. This function is analogous to the range() function of Python.

```
>>> array7 = np.arange(6)
# an array of 6 elements is created with
start value 5 and step size 1
>>> array7
array([0, 1, 2, 3, 4, 5])
# Creating an array with start value -2, end
# value 24 and step size 4
>>> array8 = np.arange( -2, 24, 4 )
>>> array8
array([-2, 2, 6, 10, 14, 18, 22])
```

1. What is NumPy ?
2. What is an array and how is it different from a list? What is the name of the built-in array class in NumPy ?
3. Create the following NumPy arrays:

a) A 1-D array called zeros having 10 elements and all the elements are set to zero.
b) A 1-D array called vowels having the elements ‘a’,‘e’, ‘i’, ‘o’ and ‘u’.
c) A 2-D array called ones having 2 rows and 5 columns and all the elements are set to 1 and dtype as int.

d) Use nested Python lists to create a 2-D array called myarray1 having 3 rows and 3 columns and store the following data:
2.7, -2, -19
0, 3.4, 99.9
10.6, 0, 13
e) A 2-D array called myarray2 using arange() having 3 rows and 5 columns with start value = 4,step size 4 and dtype as float.

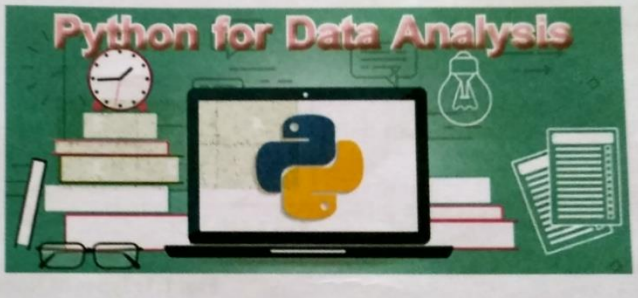
For Reference :

NumPy

1.1 INTRODUCTION

Computer Science has always been a field of continuous evolution and regular advancements in terms of software efficiency, programming methodologies, user interface and applications. With the advent of data sciences or data analytics, it has become easier and efficient to handle big data, often called huge data.

Data science or data analytics is a process of analyzing a large set of data points to get answers to questions related to that dataset.



Python provides a powerful interface for scientific computing using its popular libraries and is being extensively used for data sciences.

NumPy and Pandas are the most extensively used libraries supported by Python. This chapter focuses on NumPy, which is a good tool and a complete library offered by Python for data analysis. Most of the statistical analysis which needs data to be stored in memory uses NumPy.

We have discussed NumPy concepts in detail in Class XI. In this chapter, we will be revising the concepts learnt in the previous class with practical implementations.

1.2 WHAT IS NumPy

The NumPy library is a popular Python library used for scientific computing applications and is an acronym for "Numerical Python".

NumPy ("Numerical Python" or "Numeric Python") is an open-source module of Python that provides functions for fast mathematical computation on arrays and matrices. Arrays, in general, refer to a named group of homogeneous (same data type) elements. NumPy provides excellent ndarray—n-dimensional array—objects.

2020-4-6 12:09

In an 'ndarray' object, we can store multiple items of the same ... facility around the array object that makes NumPy convenient for performing math and data manipulation.

Let us first revise the concept of arrays.

What is an array

An array is a container which can hold a fixed number of items and these items should be of the same type. It is a contiguous (one after the other) memory location holding elements of the same type. An array consists of:

- **Element**—Each item stored in an array is called an element.
- **Index**—Each location of an element in an array has a numerical index, which is used to identify the element.

Anatomy of an Array

We will first discuss the components of a NumPy array. The various terms associated with NumPy arrays are as follows:

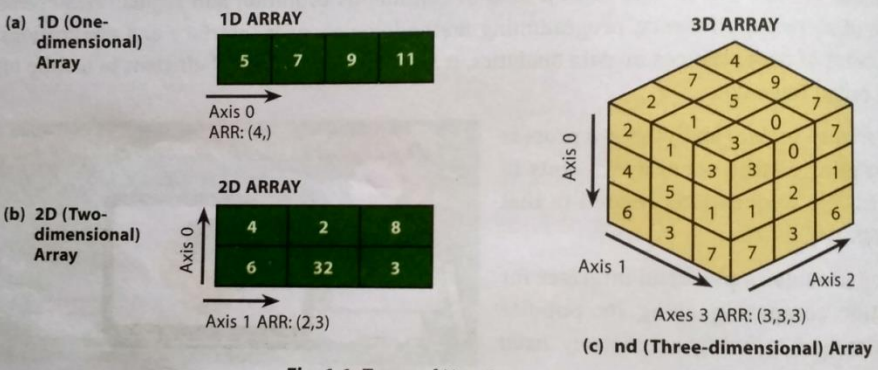


Fig. 1.1: Types of NumPy Arrays

1. **Axes:** An array can be classified as single-dimensional (1D) array, two-dimensional (2D) array and multi-dimensional (3D or 4D and so forth) array as shown in Fig. 1.1.

   The dimensions of an array are described as its axes. The axes of an array describe the order of indexing in multidimensional arrays.

   As evident from the figure,

   (a) **axis=0** defines 1D array running across only one row-wise with size as (4,) which indicates the number of columns as 4.

   (b) The next figure shows a 2D array with **axis=1**, indicating elements across the columns. In a 2D array, we have both rows as well as columns and size is (2,3) with 2 rows and 3 columns.

   (c) For a multidimensional array like a 3D array, there are three dimensions running across more than 2 axes, with size as (3,3,3).

   It must be remembered that axes are always numbered starting 0 onwards for ndarray.

say that rank is equal to the number of axes.

3. **Shape:** The shape of an ndarray is defined by the number of elements it contains along each of its axis. The shape() method describes the shape or, we may say, the size of an ndarray and returns the size in the form of a tuple.

Array Representation

Arrays can be declared in various ways in different languages. Given below is an illustration.

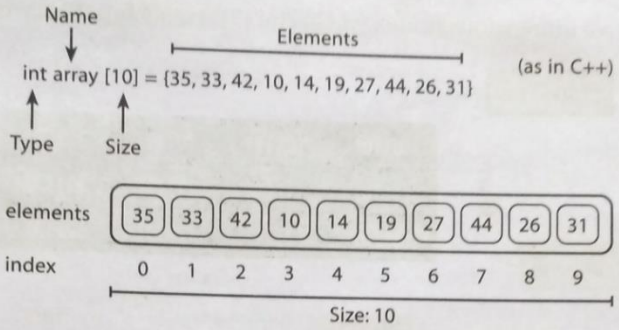


Fig. 1.2: Array Representation

As per the above illustration (Fig. 1.2), following are the important points to be remembered:

- Index starts with 0.
- Array length is 10, which means that it can store 10 elements.
- Each element can be accessed via its index. *For example*, we can fetch an element at index 6 as 27.

Thus, NumPy is used to implement all operations that are performed using arrays.

1.3 WORKING WITH NumPy

Before we start writing programs using NumPy, we need to import it. The syntax to import NumPy is:

```
>>>import numpy as np
```

(here, np is an alias for NumPy, which is optional)

☞ **NumPy Arrays:** NumPy arrays are the building blocks of most of the NumPy operations. The NumPy arrays can be handled and divided into two types:

- **One-dimensional (1D) array**—It is also known as a Vector. The number of subscript/index determines the dimensions of the array. An array of one dimension is known as a one-dimensional array or 1D array.

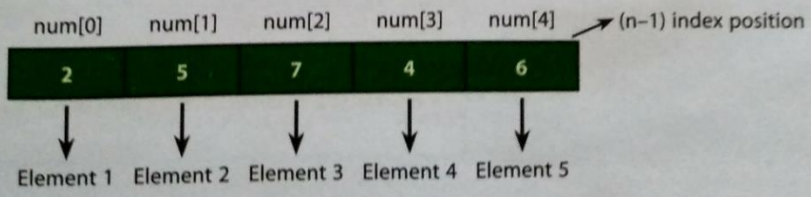


Fig. 1.3: 1D Representation in Memory

In Fig. 1.3, num is an array; its first element is at 0 index position, next element is at 1 and so on, till the last element at n–1 index position. At 0 index position, the value is 2 and at 1 index position, the value is 5.

- **Multidimensional arrays or ndarrays**—The most extensively used multidimensional arrays are two-dimensional (2D) arrays, also known as Matrices.

  We can initialize NumPy arrays from nested Python lists and access their elements. Their main data object is the **ndarray**, an n-dimensional array type that describes a collection of 'items' of the same type.

  An array of two dimensions/index/subscript is termed as a 2D (two-dimensional) array.

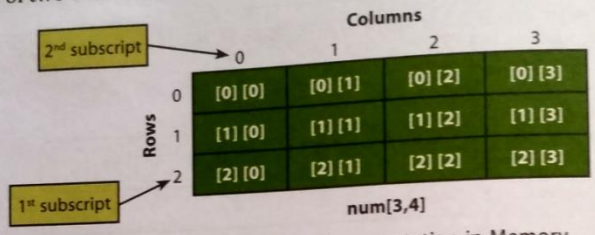


Fig. 1.4: 2D Representation in Memory

In Fig. 1.4, num is an array of two dimensions with 3 rows and 4 columns. Subscript of rows is from 0 to 2 and columns are from 0 to 3.

Table 1.1: Difference between NumPy Array and List

NumPy Array	List
NumPy array works on homogeneous (same) types.	Python lists are made up of heterogeneous (different) types.
NumPy array does not support addition and removal of elements.	Python list supports adding and removal of elements.
Can't contain elements of different types	Can contain elements of different types
Less memory consumption	More memory consumption
Faster runtime execution	Runtime execution is comparatively slower than arrays

1.4 HOW TO CREATE A NumPy ARRAY

There are several ways to create a NumPy array. Let us see a few of them in brief.

S.No.	Method	Description	Examples

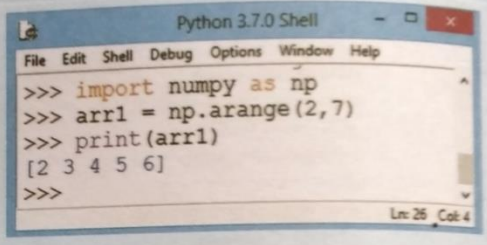
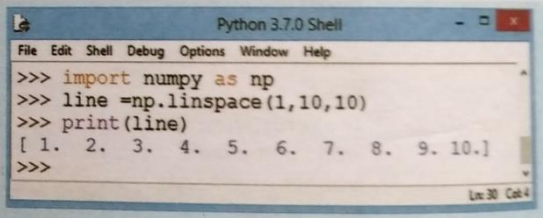
1.	array()	To create a one-dimensional/two-dimensional NumPy array, we can simply pass a Python list to the **array** method.	`#To create a one-dimensional array`
			
			`import numpy as np`
			`l1 = [2, 3, 4, 5, 6]`
			`arr1 = np.array(l1)`
			`print(arr1)`
			
			`#To create a two-dimensional array`
			
			`import numpy as np`
			`A = np.array([[10,11,12,13],[21,22,23,24]]) #Multiple lists inside list`
			`print(A)`

		one-dimensional array from String.	`import numpy as np`
			`data =np.fromstring('1 2 10 12', dtype=int, sep=' ')`
			`print(data)`

3.	empty()	**empty()** function can be used to create empty array or an uninitialized array of specified shape and dtype. Syntax is:
		`numpy.empty (Shape, [dtype= <datatype>,] [order = 'C' or 'F'])`
		where, dtype: is a data type of Python or NumPy to set initial values. Shape: is dimension meaning row-wise arrangement of data. C means C-like. Order :'C' means column-wise arrangement of data. 'F' means Fortran-like. Also, the parameters dtype and order are optional. If you do not specify dtype, then, by default, dtype is taken as float. Similarly, default order is taken as 'C'.
			`import numpy as np`
			`arr = np.empty([3,2], dtype=int)`
			`print(arr)`
			`>>>`
			`RESTART: C:/Users/preeti/AppData/Local/_arr.py`
			`[[6357069 6815843]`
			`[7209065 4653157]`
			`[6881397 100]]`
			`>>>`

4.	zeros()	The **zeros()** function works in the same manner as empty(), but the only difference is that it creates and returns a new array with all zeroes as its elements as per the specified size and type. Its syntax is:
		`numpy.zeros(shape, dtype=float, order='C')`
		Here, shape refers to shape of an empty array in int or sequence of integer values. dtype is the desired output data type and is optional.
			`# array of five zeros. Default dtype is float`
			`import numpy as np`
			`arr1 = np.zeros(5)`
			`print(arr1)`
			`>>>`
			`RESTART: C:/Users/preeti/AppData/Local/oes.py`
			`[0. 0. 0. 0. 0.]`
			`>>>`

5.	ones()	The **ones()** function works in the same manner as empty(), creates and returns a new array with all ones as its elements, as per the specified size and type. The syntax is:
		`numpy.ones(shape, dtype=None, order='C')`
			`# array of five ones. Default dtype is float`
			`import numpy as np`
			`arr1 = np.ones(5)`
			`print(arr1)`
			
			`>>>`
			`RESTART: C:/Users/preeti/AppData/Local`
			`s.py`
			`[1. 1. 1. 1. 1.]`

6.	arange()	The **arange()** function is used to create array from a range. Its syntax is: `<arrayname> = numpy.arange([start],stop,[step],[dtype])` Another commonly-used method for creating a NumPy array is the arange method. This method takes the start index of the array, the end index and the step size (which is optional).	
7.	linspace()	Another very useful method to create NumPy array is the **linspace** method. This function can be used to prepare array of range. Its syntax is: `<arrayname> = numpy.linspace([start],stop,[dtype])` This method takes three arguments: a start index, end index and the number of linearly-spaced numbers that we want between the specified range. For instance, if the first index is 1, the last index is 10, and we need 10 equally-spaced elements within this range, we can use the linspace method.	

2020-4-6 12:11

8.	copy()	The **copy()** function is used to create a copy of an existing array. If we just assign a portion of an array to another array, the new array we just created refers to the parent array in memory. Its syntax is:
		`<ndarray>.copy(order='C')`
			`#Copy function is used to create the copy of the existing array.`
			`import numpy as np`
			`x = np.array([1, 2, 3])`
			`y = x`
			`z = np.copy(x)`
			`x[0] = 10`
			`print(x)`
			`print(y)`
			`print(z)`
			
			`>>>`
			` RESTART: C:/Users/preeti/AppData/Local`
			`.py`
			`[10  2  3]`
			`[10  2  3]`
			`[1 2 3]`
			`>>>`

9.	reshape()	We can create 2D array from 1D array using **reshape()** function. Reshaping means changing the arrangement of items so that the shape of the array changes while maintaining the same number of dimensions. Its syntax is:
		`<ndarray>.reshape(<shape tuple>)`
			prog_reshape.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_re...
			File Edit Format Run Options Window Help
			`#Creation of 2D array from 1D array using reshape() function.`
			
			`import numpy as np`
			`A = np.array([1,2,3,4,5,6])`
			`B = np.reshape(A, (2, 3)) #reshape shall convert 1D array into`
			`# 2D array with 2 rows and 3 columns`
			`print(B)`
			Ln: 10 Col: 0
			
			`>>>`
			` RESTART: C:/Users/preeti/AppData/Local`
			`.py`
			`[[1 2 3]`
			` [4 5 6]]`
			`>>>`

10.	eye()/ identity()	The **eye** method can be used to create an identity matrix, which can be very useful to perform a variety of operations using arrays. An identity matrix is a matrix with zeroes across rows and columns except the diagonal. It contains all 1s as its diagonal elements. Its syntax is:
		`numpy.identity(n,dtype=None)`
		or
		`numpy.eye(n,dtype=<class'float'>`
			`#create a 4x4 identity matrix using the eye method`
			`import numpy as np`
			
			`# 4x4 matrix with 1's as all diagonal elements`
			`mat1 = np.identity(4)  Alternatively, np.eye(4)`
			`print("\nMatrix 1 : \n", mat1)`
			
			`#Alternatively`
			`mat2 = np.identity(4, dtype = float)`
			`print("Matrix 2 : \n", mat2)`
			
			` RESTART: C:/Users/preeti`
			`y1.py`
			
			`Matrix 1 :`
			` [[1. 0. 0. 0.]`
			` [0. 1. 0. 0.]`
			` [0. 0. 1. 0.]`
			` [0. 0. 0. 1.]]`
			`Matrix 2 :`
			` [[1. 0. 0. 0.]`
			` [0. 1. 0. 0.]`
			` [0. 0. 1. 0.]`
			` [0. 0. 0. 1.]]

1.5 OPERATIONS ON NumPy ARRAY

NumPy allows several operations to be performed on arrays for accessing and retrieving its elements. The major operations include slicing, joins and subsets. We will now discuss these operations in detail.

1.5.1 Array Slicing

Structures like lists and NumPy arrays can be sliced. Slicing of NumPy array elements is similar to slicing of list elements. This means that a sub-sequence of the structure can be indexed and retrieved.

Slicing is specified by using the colon operator ':' with a 'from' and 'to' index before and after the column respectively. The slice extends from the 'from' index and ends one item before the 'to' index. The syntax is:

```
data[from:to]
```

Practical Implementation–1

To illustrate slicing in 1D array.

prog_slice1D.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_slice1D.py (3.7.0)

File Edit Format Run Options Window Help

```
#Slicing in 1D array

import numpy as np
data = np.array([5,2,7,3,9])
print (data[:])  #Shall extract slice from start to end
print(data[1:3]) #Shall extract slice from 1st position to 4th, excluding 4th
                 #position element
print(data[:2])  #Shall extract slice from start to 2nd index value
                 #excluding the last element
print(data[-2:]) #Shall extract slice from negative indices
```

Ln: 16 Col: 0

```
>>>
 RESTART: C:/Users/preeti/AppData/Local
.py
[5 2 7 3 9]
[2 7]
[5 2]
[3 9]
>>>
```

Practical Implementation–2

To illustrate slicing in 2D array.

Example 1:

prog_2dslicing.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_2dslicing.py (3.7_

File Edit Format Run Options Window Help

```
#Slicing of numpy 2D-arrays

import numpy as np
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
b = a[1:2,0:4] # Slice consists of second row all columns
print('b',b)
b = a[1:2,1:2] # Slice consists of second row second column
print('b',b)
b = a[:2,0:2] # Slice consists of two rows first[1,2] and second[5,6]
print('b',b)
b = a[:2,1:3] # Slice consists of two rows first[2,3] and second[6,7]
print('b',b)
b = a[1:3,2:4] # Slice consists of two rows first[7,8] and second[11,12]
print('b',b)
```

```
>>>
 RESTART: C:/Users/preeti/AppData
ng.py
b [[5 6 7 8]]
b [[6]]
b [[1 2]
  [5 6]]
b [[2 3]
  [6 7]]
b [[ 7  8]
  [11 12]]
>>>
```

Example 2:

prog_slice2D.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_slice2D.py (3.7.0)

File Edit Format Run Options Window Help

```
#2D array Slicing

import numpy as np
A = np.array([[7,5,9,4],[7,6,8,8],[1,6,7,7]])
print(A[:2,:3])      #print elements of 0,1 rows and 0,1,2 columns
print(A[:3,::2])     #print elements of 0,1,2 rows and alternate column position
print(A[::-1,::-1])  #print elements in reverse order
print(A[:,0])        #print all elements of 0 column
print(A[0,:])        #print all elements of 0 rows
print(A[0])          #print all elements of 0 row
```

Ln: 13 Col: 0

```
>>>
 RESTART: C:/Users/preeti/AppData/Local
.py
[[7 5 9]
 [7 6 8]]
[[7 9]
 [7 8]
 [1 7]]
[[7 7 6 1]
 [8 8 6 7]
 [4 9 5 7]]
[7 7 1]
[7 5 9 4]
[7 5 9 4]
```

We can even change the values of the elements that are extracted using slicing.

For example,

prog_slice_changevalue.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyt...

File Edit Format Run Options Window Help

```
#Changing the values in an array resulting from slice

import numpy as np
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
b = a[0:2,1:3] #Sliced array
print(b)
b[0,0]=20 # Changing the value in the slice
print(b)
print(a) # The change gets reflected in the original array
```

Ln: 13 Col: 0

```
>>>
 RESTART: C:/Users/preeti/
hangevalue.py
[[2 3]
 [6 7]]
[[20  3]
 [ 6  7]]
[[ 1 20  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>>
```

Value of b[0,0] gets changed from 2 to 20

Value of a[0,1] gets changed from 2 to 20

2020-4-6 12:12

1.5.2 Joins in Arrays

Joining of two arrays in NumPy is done using **concatenate()** function. This function is used to join two or more arrays of the same shape. In case of a 2D array, this function concatenates two arrays either by rows or by columns. Its syntax is:

```
numpy.concatenate((a1, a2, ...), axis)
```

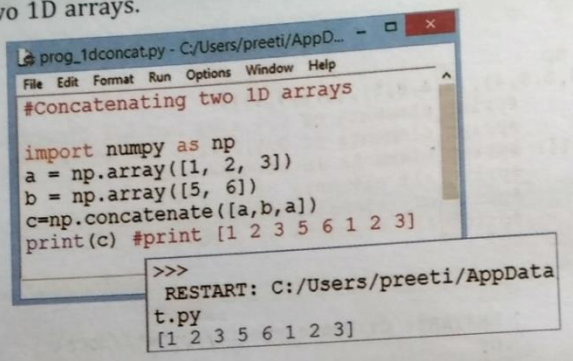
Here,

a1, a2... is the sequence of arrays of the same type.

axis—refers to the axis along which arrays must be joined. Default is 0.

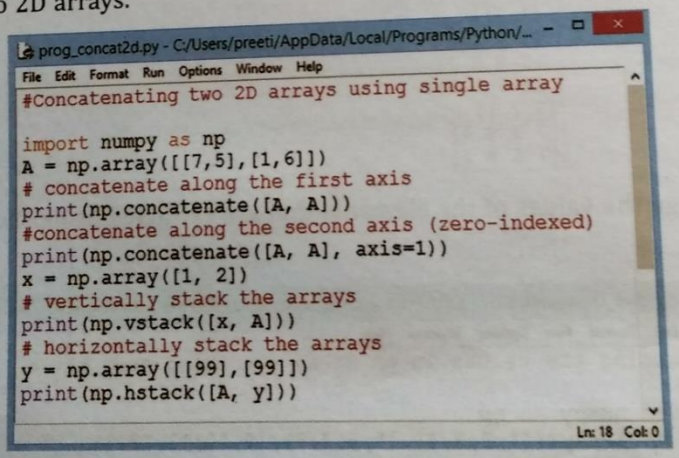
Practical Implementation–3

To concatenate/join two 1D arrays.



Practical Implementation–4

To concatenate two 2D arrays.



In addition to the concatenate function, NumPy also offers two convenient functions hstack and vstack to stack/combine arrays horizontally or vertically. hstack performs horizontal and vstack performs vertical concatenation of two arrays and hence the output is obtained as shown alongside.

```
>>>
 RESTART: C:/Users/preeti
d.py
[[7 5]
 [1 6]
 [7 5]
 [1 6]]
[[7 5 7 5]
 [1 6 1 6]]
[[1 2]
 [7 5]
 [1 6]]
[[ 7  5 99]
```

(g) (i) (h) (iii) (i) (i) (j) (i) (k) (ii) (l) (i

SOLVED QUESTIONS

1. Why are NumPy arrays used over lists?

Ans. (a) NumPy arrays have contiguous memory allocation. Thus, the same elements stored as list will re more space as compared to arrays.
(b) They are speedier to work with and hence, are more efficient than the lists.
(c) They are more convenient to deal with.

2. Write a NumPy program to get the NumPy version.

Ans.
```
import numpy as np
print(np.__version__)
```

3. Find the output of the following program: [CBSE Sample Paper

```
import numpy as np
d=np.array([10,20,30,40,50,60,70])
print(d[-4:])
```

Ans. `[40 50 60 70]`

4. Fill in the blank with appropriate NumPy method to calculate and print the variance of an array.

```
import numpy as np
data=np.array([1,2,3,4,5,6])
print(np.___(data,ddof=0)
```

Ans. `print(np.var(data,ddof=0))`

5. Write the output of the following code:

```
import numpy as np
array1=np.array([10,12,14,16,18,20,22])
array2=np.array([10,12,15,16,12,20,12])
a=(np.where(array1==array2))
print(array1[a])
```

Ans. `[10 12 16 20]`

6. Write a NumPy program to test whether none of the elements of a given array is zero.

Ans.
```
import numpy as np
x = np.array([1, 2, 3, 4])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
x = np.array([0, 1, 2, 3])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
```

Output:

```
Original array:
[1 2 3 4]
Test if none of the elements of the said array is zero: True
Original array:
[0 1 2 3]
Test if none of the elements of the said array is zero: False
```

7. Write a NumPy program to create a 3×3 identity matrix, i.e., diagonal elements are 1, the rest are 0. Replace all 0 to a random number from 10 to 20.

Ans.
```
import numpy as np
array1=np.identity(3)
print(array1)
x=np.where(array1==0)
for i in x:
  array1[x]=np.random.randint(low=10,high=20)
print(array1)
```

8. Write a NumPy program to create a 3x3 matrix with the elements between 0 and 9. Check for even numbers in the matrix and replace them with a random number between 10 and 20.

Ans.
```
import numpy as np
Z = np.arange(9).reshape(3,3)
print(Z)
x=np.where((Z%2)==0)
for i in x:
    Z[x]=np.random.randint(low=10,high=20)
print(Z)
```

9. Write a NumPy program to create a 10x10 matrix in which the elements on the borders will be equal to 1, and inside 0.

Ans.
```
import numpy as np
x = np.ones((10, 10))
x[1:-1, 1:-1] = 0
print(x)
```

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

10. Write a NumPy program to find the number of rows and columns of the given matrix.

Output:

```
Original matrix:
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
Number of rows and columns of the said matrix:
(3, 4)
```

Ans.

```
import numpy as np
m= np.arange(10,22).reshape((3, 4))
print("Original matrix:")
print(m)
print("Number of rows and columns of the said matrix:")
print(m.shape)
```

11. Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

Output:

```
Original array:
[[0 1]
 [2 3]]
Sum of all elements:
6
Sum of each column:
[2 4]
Sum of each row:
[1 5]
```

Ans.

```
import numpy as np
x = np.array([[0,1],[2,3]])
print("Original array:")
print(x)
print("Sum of all elements:")
print(np.sum(x))
print("Sum of each column:")
print(np.sum(x, axis=0))
print("Sum of each row:")
print(np.sum(x, axis=1))
```

12. Write a NumPy program to convert a given array into a list and then convert it into a list again.

Ans.

```
import numpy as np
a = [[1, 2], [3, 4]]
x = np.array(a)
a2 = x.tolist()
print(a == a2)
```

Output:

```
True
```

2020-4-6 12:13

20. Write a NumPy program to create a random array with 1000 elements and compute the average, variance, standard deviation of the array elements.

The sample is:

```
Average of the array elements:
-0.0255137240796
Standard deviation of the array elements:
0.984398282476
Variance of the array elements:
0.969039978542
```

Note: The output may vary as random() shall generate a different number every time the code gets execu

Ans.
```
import numpy as np
x = np.random.randn(1000)
print("Average of the array elements:")
mean = x.mean()
print(mean)
print("Standard deviation of the array elements:")
std = x.std()
print(std)
print("Variance of the array elements:")
var = x.var()
print(var)
```

21. Write a NumPy program to compute the covariance matrix of two given arrays.

Required output should be:

```
Original array1:
[0 1 2]
Original array2:
[2 1 0]
Covariance matrix of the said arrays:
[[ 1. -1.]
 [-1. 1.]]
```

Ans.
```
import numpy as np
x = np.array([0, 1, 2])
y = np.array([2, 1, 0])
print("\nOriginal array1:")
print(x)
print("\nOriginal array2:")
print(y)
print("\nCovariance matrix of the said arrays:\n",np.cov(x, y))
```

22. Write a NumPy program to compute cross-correlation of the two given arrays.

```
Original array1:
[0 1 3]
Original array2:
[2 4 5]
Cross-correlation of the said arrays:
[[2.33333333 2.16666667]
[2.16666667 2.33333333]]
```