Experiment 05

Write a python program to understand Basic Array operations on 1-D and 2D andMultidimensional arrays using Array, Numpy.
Also explain the attributes of the array function with examples and demonstrate the concept of matrix creation using matrix and perform different matrix operations.
Also show the different ways to create an array and slow slicing operator on 2D array.
Implement few builtin functions related to array.

Roll No.	01
Name	Aamir Ansari
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO2: Interpret advanced data types and functions in python

Aim:

Write a python program to understand Basic Array operations on 1-D and 2D and Multidimensional arrays using Array, Numpy

Introduction:

NumPy

Numpy is one of the most fundamental libraries for using Python in the field of Machine Learning. NumPy is a numerical library for Python that allows extremely fast data handling and generation. It utilises arrays that can store data much more efficiently than a regular Python list. NumPy is extremely important to learn before learning about the Pandas library

Environment Setup: **pip install numpy**

NumPy array (ndarray class) is the most used construct of NumPy in Machine Learning and Deep learning. Let us look into some important attributes of this NumPy array.

Let us create a Numpy array, array_A.

Pass the above list to array() function of NumPy

```
array_A = np.array([ [3,4,6], [0,8,1] ])
```

Some attributes of array are:

(1) ndarray.ndim

ndim represents the number of dimensions (axes) of the ndarray.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of ndim will be 2. This ndarray has two dimensions (axes) - rows (axis=0) and columns (axis=1)

(2) ndarray.shape

shape is a tuple of integers representing the size of the ndarray in each dimension.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of shape will be (2,3) because this ndarray has two dimensions - rows and columns - and the number of rows is 2 and the number of columns is 3

(3) ndarray.size

size is the total number of elements in the ndarray. It is equal to the product of elements of the shape.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], shape is (2,3), size will be product (multiplication) of 2 and 3 i.e. (2*3) = 6. Hence, the size is 6.

(4) ndarray.dtype

dtype tells the data type of the elements of a NumPy array. In NumPy array, all the elements have the same data type.

e.g. for this NumPy array [[3,4,6], [0,8,1]], dtype will be int64

(5) ndarray.itemsize

itemsize returns the size (in bytes) of each element of a NumPy array.
e.g. for this NumPy array [[3,4,6], [0,8,1]], itemsize will be 8, because this array consists of integers and size of integer (in bytes) is 8 bytes.

Python Matrix

A Python matrix is a specialized two-dimensional rectangular array of data stored in rows and columns. The data in a matrix can be numbers, strings, expressions, symbols, etc. Matrix is one of the important data structures that can be used in mathematical and scientific calculations

The data inside the two-dimensional array in matrix format looks as follows

2	3	4	=> row 1
5	6	7	=> row 2
‖ V col 1	‖ V col 2	‖ V col 3	

The above matrix shows a 2x3 matrix. It has two rows and three columns. The data inside the first row, i.e., row1, has values 2,3,4, and row2 has values 5,6,7. The columns col1 has values 2,5, col2 has values 3,6, and col3 has values 4,7.
So similarly, you can have your data stored inside the nxn matrix in Python. A lot of operations can be done on a matrix-like addition, subtraction, multiplication, etc.
Python does not have a straightforward way to implement a matrix data type.
The python matrix makes use of arrays, and the same can be implemented.

1. Create a Python Matrix using the nested list data type
2. Create Python Matrix using Arrays from Python Numpy package

Creation of matrix using list

In Python, the arrays are represented using the list data type. So now will make use of the list to create a python matrix.
We will create a 3x3 matrix, as shown below

8	14	-6
12	7	4
-11	3	21

1. The matrix has 3 rows and 3 columns.
2. The first row in a list format will be as follows: [8,14,-6]
3. The second row in a list will be: [12,7,4]
4. The third row in a list will be: [-11,3,21]

The matrix inside a list with all the rows and columns is as shown below:

```
List = [[Row1],
        [Row2],
        [Row3]
        ...
        [RowN]]
```

So as per the matrix listed above the list type with matrix data is as follows

M1 = [[8, 14, -6], [12,7,4], [-11,3,21]]

We will make use of the matrix defined above. The example will read the data, print the matrix, display the last element from each row

<u>To print the whole matrix:</u>

```
        M1 = [[8, 14, -6],
        [12,7,4],
        [-11,3,21]]
        #To print the matrix
        print(M1)
```

output

The Matrix M1 = **[[8, 14, -6], [12, 7, 4], [-11, 3, 21]]**

<u>To print the last element from each row</u>

```
        M1 = [[8, 14, -6],
                [12,7,4],
                [-11,3,21]]
        matrix_length = len(M1)

        #To read the last element from each row.
        for i in range(matrix_length):
           print(M1[i][-1])
```

```
# output
        -6
        4
        21
```

To print the rows in the Matrix

```
        M1 = [[8, 14, -6],
              [12,7,4],
              [-11,3,21]]
        matrix_length = len(M1)

        #To print the rows in the Matrix
        for i in range(matrix_length):
           print(M1[i])
```

```
# output
        [8, 14, -6]
        [12, 7, 4]
        [-11, 3, 21]
```

Adding Matrices Using Nested List

We can easily add two given matrices. The matrices here will be in the list form. Let us work on an example that will take care to add the given matrices.

Matrix 1:

```
        M1 = [[8, 14, -6],
              [12,7,4],
              [-11,3,21]]
```

Matrix 2 :

```
        M2 = [[3, 16, -6],
              [9,7,-4],
              [-1,3,13]]
```

Last will initialize a matrix that will store the result of M1 + M2.

Matrix 3 :

```
        M3  = [[0,0,0],
              [0,0,0],
              [0,0,0]]
```

To add, the matrices will make use of a for-loop that will loop through both the matrices given.

```
M1 = [[8, 14, -6],
     [12,7,4],
     [-11,3,21]]
M2 = [[3, 16, -6],
          [9,7,-4],
          [-1,3,13]]
M3  = [[0,0,0],
       [0,0,0],
       [0,0,0]]
matrix_length = len(M1)

#To Add M1 and M2 matrices
for i in range(len(M1)):
for k in range(len(M2)):
       M3[i][k] = M1[i][k] + M2[i][k]

#To Print the matrix
print("The sum of Matrix M1 and M2 = ", M3)
```

output

The sum of Matrix M1 and M2 = [[11, 30, -12], [21, 14, 0], [-12, 6, 34]]

Multiplication of Matrices using Nested List

To multiply the matrices, we can use the for-loop on both the matrices as shown in the code below:

```
M1 = [[8, 14, -6],
     [12,7,4],
     [-11,3,21]]

M2 = [[3, 16, -6],
          [9,7,-4],
          [-1,3,13]]

M3  = [[0,0,0],
       [0,0,0],
       [0,0,0]]

matrix_length = len(M1)
```

```
#To Multiply M1 and M2 matrices
for i in range(len(M1)):
for k in range(len(M2)):
    M3[i][k] = M1[i][k] * M2[i][k]

#To Print the matrix
print("The multiplication of Matrix M1 and M2 = ", M3)
```

Output:

The multiplication of Matrix M1 and M2 = [[24, 224, 36], [108, 49, -16], [11, 9, 273]]

Create Python Matrix using Arrays from Python Numpy package

The python library Numpy helps to deal with arrays. Numpy processes an array a little faster in comparison to the list.

The command to install Numpy is

pip install NumPy

Then we import it

import NumPy

We can import NumPy as an alias

import NumPy as np

We are going to make use of array() method from Numpy to create a python matrix

Creation of matrix using numpy

```
import numpy as np
M1 = np.array([[5, -10, 15], [3, -6, 9], [-4, 8, 12]])
print(M1)
```

#Output:

```
[[  5 -10  15]
 [  3  -6   9]
 [ -4   8  12]]
```

Matrix Operation using Numpy.Array()

The matrix operation that can be done is addition, subtraction, multiplication, transpose, reading the rows, columns of a matrix, slicing the matrix, etc. In all the examples, we are going to make use of an array() method.

Matrix Addition

To perform addition on the matrix, we will create two matrices using numpy.array() and add them using the (+) operator.

Example:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
M2 = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
M3 = M1 + M2
print(M3)
```

Output:

```
[[ 12 -12  36]
 [ 16  12  48]
 [  6 -12  60]]
```

Matrix Subtraction

To perform subtraction on the matrix, we will create two matrices using numpy.array() and subtract them using the (-) operator.

Example:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
M2 = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
M3 = M1 - M2
print(M3)
```

Output:

```
[[ -6  24 -18]
 [ -6 -32 -18]
 [-20  40 -18]]
```

Matrix Multiplication

First will create two matrices using numpy.arary(). To multiply them will, you can make use of numpy dot() method. Numpy.dot() is the dot product of matrix M1 and M2. Numpy.dot() handles the 2D arrays and perform matrix multiplications.

Example:

```
import numpy as np
M1 = np.array([[3, 6], [5, -10]])
M2 = np.array([[9, -18], [11, 22]])
M3 = M1.dot(M2)
print(M3)
```

Output:

```
[[  93   78]
 [ -65 -310]]
```

Matrix Transpose

The transpose of a matrix is calculated, by changing the rows as columns and columns as rows. The transpose() function from Numpy can be used to calculate the transpose of a matrix.

Example:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
M2 = M1.transpose()
print(M2)
```

Output:

```
[[  3   5   4]
 [  6 -10   8]
 [  9  15  12]]
```

Slicing of a Matrix

Slicing will return you the elements from the matrix based on the start /end index given.

The syntax for slicing is - [start:end]

If the start index is not given, it is considered as 0. For example [:5], it means as [0:5].

If the end is not passed, it will take as the length of the array.

If the start/end has negative values, it will the slicing will be done from the end of the array.

Ex.

```
import numpy as np
arr = np.array([2,4,6,8,10,12,14,16])
print(arr[3:6])          # will print the elements from 3 to 5
print(arr[:5])           # will print the elements from 0 to 4
print(arr[2:])           # will print the elements from 2 to length of the array.
print(arr[-5:-1])        # will print from the end i.e. -5 to -2
print(arr[:-1])          # will print from end i.e. 0 to -2
```

Output:

```
[ 8 10 12]
[ 2  4  6  8 10]
[ 6  8 10 12 14 16]
[ 8 10 12 14]
[ 2  4  6  8 10 12 14]
```

Now let us implement slicing on the matrix . To perform slicing on a matrix the syntax will be M1[row_start:row_end, col_start:col_end]

1. The first start/end will be for the row, i.e to select the rows of the matrix.
2. The second start/end will be for the column, i.e to select the columns of the matrix.

The matrix M1 that we are going to use is as follows:

```
M1 = np.array([[2, 4, 6, 8, 10],
               [3, 6, 9, -12, -15],
               [4, 8, 12, 16, -20],
               [5, -10, 15, -20, 25]])
```

There are a total of 4 rows. The index starts from 0 to 3. The 0th row is the [2,4,6,8,10], 1st row is [3,6,9,-12,-15] followed by 2nd and 3rd.

The matrix M1 has 5 columns. The index starts from 0 to 4.The 0th column has values [2,3,4,5], 1st columns have values [4,6,8,-10] followed by 2nd, 3rd, 4th, and 5th.

Here is an example showing how to get the rows and columns data from the matrix using slicing. In the example, we are printing the 1st and 2nd row, and for columns, we want the first, second, and third column. To get that output we have used: M1[1:3, 1:4]

Example:

```
import numpy as np
M1 = np.array([[2, 4, 6, 8, 10],
   [3, 6, 9, -12, -15],
   [4, 8, 12, 16, -20],
   [5, -10, 15, -20, 25]])
print(M1[1:3, 1:4])     # For 1:3, it will give first and second row.
                        #The columns will be taken from first to third.
```

Output:

```
[[  6   9 -12]
 [  8  12  16]]
```

Example : <u>To print all rows and third columns</u>

```
import numpy as np
M1 = np.array([[2, 4, 6, 8, 10],
               [3, 6, 9, -12, -15],
               [4, 8, 12, 16, -20],
               [5, -10, 15, -20, 25]])
print(M1[:,3]) # This will print all rows and the third column data.
```

Output:

[8 -12 16 -20]

Example: To print the first row and all columns

```
import numpy as np
M1 = np.array([[2, 4, 6, 8, 10],
   [3, 6, 9, -12, -15],
   [4, 8, 12, 16, -20],
   [5, -10, 15, -20, 25]])
print(M1[:1,]) # This will print first row and all columns
```

Output:

[[2 4 6 8 10]]

Example: To print the first three rows and first 2 columns

```
import numpy as np
M1 = np.array([[2, 4, 6, 8, 10],
   [3, 6, 9, -12, -15],
   [4, 8, 12, 16, -20],
   [5, -10, 15, -20, 25]])
print(M1[:3,:2])
```

Output:

[[2 4]
[3 6]
[4 8]]

Accessing NumPy Matrix

We have seen how slicing works. Taking that into consideration, we will how to get the rows and columns from the matrix.

Example: To print the rows of the matrix

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
print(M1[0])  #first row
print(M1[1]) # the second row
print(M1[-1]) # -1 will print the last row
```

Output:

[3 6 9]
[5 -10 15]
[4 8 12]

To get the last row, you can make use of the index or -1. For example, the matrix has 3 rows,

1. M1[0] will give you the first row,
2. M1[1] will give you second row
3. M1[2] or M1[-1] will give you the third row or last row.

To print the columns of the matrix

```
import numpy as np
M1 = np.array([[2, 4, 6, 8, 10],
               [3, 6, 9, -12, -15],
               [4, 8, 12, 16, -20],
               [5, -10, 15, -20, 25]])
print(M1[:,0]) # Will print the first Column
print(M1[:,3]) # Will  print the third Column
print(M1[:,-1]) # -1 will give you the last column
```

Output:

```
[2 3 4 5]
[  8 -12  16 -20]
[ 10 -15 -20  25]
```

Results:

```
import numpy as np
import random

# Creating 1D Array
arr_1D = np.array([1, 2, 3])
print("Array One dimension :  \n",arr_1D)

# Creating 2D Array
arr_2D = np.array([[1, 2, 3], [4, 5, 6]])
print("\nArray with Two dimension :  \n", arr_2D)

# Creating nD array
n = int(input("\nEnter the dimension of array to create :  "))
nDarray = np.zeros([n, random.randint(1,4)])
print("\nExample of empty array of dimension",n,"is : \n",nDarray)

# Creating an array from tuple
```

```
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_05>python creation.py
Array One dimension :
 [1 2 3]

Array with Two dimension :
 [[1 2 3]
 [4 5 6]]

Enter the dimension of array to create :  4

Example of empty array of dimension 4 is :
 [[0.]
 [0.]
 [0.]
 [0.]]

Array created using passed tuple:
 [1 3 2]

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_05>python creation.py
Array One dimension :
 [1 2 3]

Array with Two dimension :
 [[1 2 3]
 [4 5 6]]

Enter the dimension of array to create :  3

Example of empty array of dimension 3 is :
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

Array created using passed tuple:
 [1 3 2]
```

```
import numpy as np

# Defining Array 1
a = np.array([[1, 2], [3, 4]])

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

print("First Array:\n", a)
print("\nSecond Array:\n", b)

# Adding 1 to every element
print("\n>---------------------------< Adding >---------------------------<")
print ("\nAdding 1 to every element:\n", a + 1)

# Subtracting 2 from each element
print("\n>---------------------------< Subtracting >---------------------------<")
print ("\nSubtracting 2 from each element:\n", b - 2)

# sum of array elements
print("\n>---------------------------< Array sum >---------------------------<")
print ("\nSum of all elements of "
       "First array: ", a.sum())

# Adding two arrays
# Performing Binary operations
print ("\nArray sum:\n", a + b)

# transpose
print("\n>---------------------------< Transpose >---------------------------<")
print("\n Transpose of First Matrix:\n", a.transpose())

# reciprocal
arr = np.array([25, 1.33, 1, 1, 100])
print('\nOur array is:')
print(arr)
print('\nReciprocal of array:')
print(np.reciprocal(arr))

# power
```

```
print("\n>--------------------------< Power >--------------------------<")
arr = np.array([5, 10, 15])
print('\nFirst array is:')
print(arr)
print('\nApplying power function:')
print(np.power(arr, 2))

# mod
print("\n>--------------------------< Mod >--------------------------<")
arr = np.array([5, 15, 20])
arr1 = np.array([2, 5, 9])

print('\nFirst array:')
print(arr)

print('\nSecond array:')
print(arr1)

print('\nApplying mod() function:')
print(np.mod(arr, arr1))

print('\nApplying remainder() function:')
print(np.remainder(arr, arr1))
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_05>python operations.py
First Array:
 [[1 2]
 [3 4]]

Second Array:
 [[4 3]
 [2 1]]

>---------------------------< Adding >---------------------------<

Adding 1 to every element:
 [[2 3]
 [4 5]]

>---------------------------< Subtracting >---------------------------<

Subtracting 2 from each element:
 [[ 2  1]
 [ 0 -1]]
```

```
>----------------------------< Array sum >----------------------------<

Sum of all elements of First array:  10

Array sum:
 [[5 5]
 [5 5]]

>----------------------------< Transpose >----------------------------<

 Transpose of First Matrix:
 [[1 3]
 [2 4]]
```

```
Our array is:
[ 25.     1.33   1.     1.   100.  ]

Reciprocal of array:
[0.04      0.7518797 1.        1.        0.01     ]

>----------------------------< Power >----------------------------<

First array is:
[ 5 10 15]

Applying power function:
[ 25 100 225]

>----------------------------< Mod >----------------------------<

First array:
[ 5 15 20]

Second array:
[2 5 9]

Applying mod() function:
[1 0 2]

Applying remainder() function:
[1 0 2]
```

```
import numpy

# Two matrices are initialized by value
x = numpy.array([[12, 6], [4, 5]])
y = numpy.array([[7, 4], [2, 9]])

#  add()is used to add matrices
print ("\nAddition of two matrices: ")
print (numpy.add(x,y))

# subtract()is used to subtract matrices
print ("\nSubtraction of two matrices : ")
print (numpy.subtract(x,y))

# divide()is used to divide matrices
print ("\nMatrix Division : ")
print (numpy.divide(x,y))

print ("\nMultiplication of two matrices: ")
print (numpy.multiply(x,y))

print ("\nThe product of two matrices : ")
print (numpy.dot(x,y))

print ("\nsquare root is : ")
print (numpy.sqrt(x))

print ("\nThe summation of elements : ")
print (numpy.sum(y))

print ("\nThe column wise summation  : ")
print (numpy.sum(y,axis=0))

print ("\nThe row wise summation: ")
print (numpy.sum(y,axis=1))

# using "T" to transpose the matrix
print ("\nMatrix transposition : ")
print (x.T)
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_05\Code>python opertaions2.py

Addition of two matrices:
[[19 10]
 [ 6 14]]

Subtraction of two matrices :
[[ 5  2]
 [ 2 -4]]

Matrix Division :
[[1.71428571 1.5       ]
 [2.         0.55555556]]

Multiplication of two matrices:
[[84 24]
 [ 8 45]]

The product of two matrices :
[[ 96 102]
 [ 38  61]]

square root is :
[[3.46410162 2.44948974]
 [2.         2.23606798]]

The summation of elements :
22

The column wise summation  :
[ 9 13]

The row wise summation:
[11 11]

Matrix transposition :
[[12  4]
 [ 6  5]]
```

Conclusion:

Hence we have successfully learned to use Arrays from NumPy and matrix