Numpy Library.

* contains powerful tools for manipulation of numerical data in python.
* Has several functions that are useful for performing mathematical and logical operations on arrays with various dimensions.
* Grid-like data structure containing value of the same data type.
* One-dimensional numpy arrays are considered vectors.
* Multi-dimensional numpy arrays are considered matrices.

Code :-

```
import numpy as np
```

Numpy's Array function :-

* Takes in an array-like object such as a python list as input.
* Returns a numpy array containing the items from the specified input.

Code :-

```
arrayo = np.array([])   #empty numpy array
print(arrayo)
```

1-D Array

MITS

```
list1 = [1, 2, 3, 4, 5, 6]
array1 = np.array(list1)
print(array1)
```

o/p :: array([1, 2, 3, 4, 5, 6])

2-D Array :-

```
list2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
array2 = np.array(list2)
print(array2)
```

o/p

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

Numpy arange function :-

* Allows you to create a numpy array that contains numbers from an interval that you specify.
* Can take in 3 numerical inputs: start, stop, step.
* Start is an optional input, indicates the start of the interval, and is included in the interval. If start is not provided, the intervals starts at 0 by default.
* Stop is a required input, indicates the end of the interval, and e is excluded from the interval.
* Step is an optional input and indicates the spacing between the numbers in the interval. If step is not provided, the numbers in the interval are spaced 1 apart from each other by default. Note that this function works best when step is an integer.

Code :

```
print (np.arange(5))
```

o/p :- array([0, 1, 2, 3, 4])

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

o/p :- array([0, 2, 4, 6, 8, 10])

Numpy's zeros Function :-

MITS

* Takes in shape as input and returns a numpy array that has the specified shape and contains all zeros.
* The shape provided can be either an integer or a tuple of two positive integers.
* If the shape is given as an integer, this function will return a one-dimensional array containing as many zeros as specified by the shape.
* If the shape is given as a tuple, the first number in the tuple indicates the number of rows, the second number indicates the number of columns, and this function will return a multiple-dimensional numpy array that has many rows and columns specified by the shape and contains all zeros.

Code :-

```
print(np.zeros(5))
```

O/P :- array([0., 0., 0., 0., 0.])

```
print(np.zeros((3,5)))
```

O/P :- array([[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.]])

Expt.No. ..... Page No. ..... Date .....

Numpy's Ones function.

MITS

* Takes in Shape as input and returns a numpy array that has the specified shape and contains all ones.
* The shape provided can be either an integer or a tuple of two positive integers.
* If the shape is given as an integer, this function will return a one-dimensional array containing as many Ones as specified by the shape.
* If the shape is given as tuple, the first number in the tuple indicates the number of rows, the second number indicates the number of columns, and this function will return a multi-dimensional Numpy array that has many rows and columns specified by the Shape and contains all ones.

Code :-

```
print (np.Ones(6))
```

O/P :- array([1., 1., 1., 1., 1., 1.])

```
print (np.ones((4,6)))
```

O/P :- array([[1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1.]])

Expt.No.	Page No.	Date .....

Numpy's linspace function :-

* Can take 3 numerical inputs : Start, Stop and num.
* Start indicates the Start of the interval.
* Stop is a required input, indicates the end of the interval.
* Both start and Stop are included in the interval.
* Num indicates how many numbers to generate, and the function returns a numpy array containing that many evenly spaced numbers over the specified interval.

Code :-

```
#Create a numpy array of 9 evenly spaced numbers from
1 to 2, including 1 and 2.
print (np.linspace (1, 2, 9))
```

O/P

```
array ([1.    , 1.125 , 1.375, 1.5  , 1.625, 1.75 ,
        1.875, 2.   ])
```

Numpy's random.randint function.

MITS

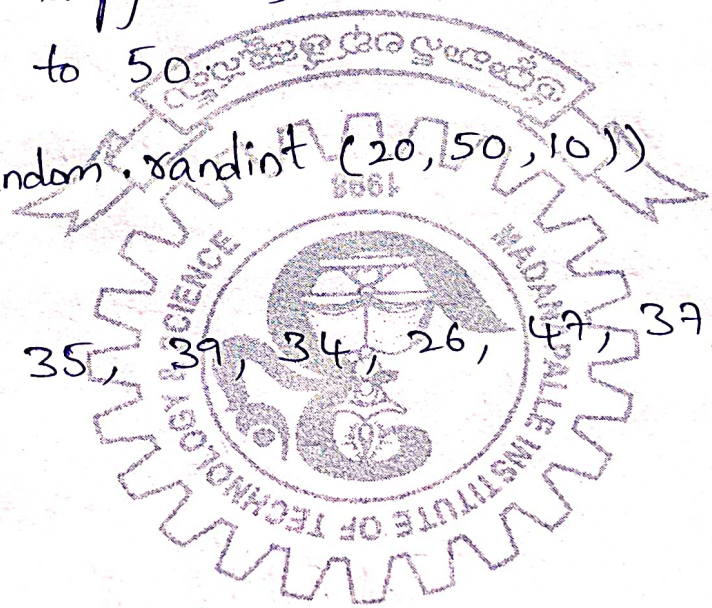
* Takes in 3 inputs: low, high and size.
* Returns a numpy array containing random integers from low to high, inclusive of low and exclusive of high, where the number of random integers is given by size.

Code

```
# create a numpy array of 10 random integers
  from 20 to 50
print (np.random.randint (20,50,10))
```

o/p:-

```
array([47, 35, 39, 34, 26, 47, 37, 41, 32, 43])
```



Find minimum & maximum values in Numpy Arrays

MITS

Code:-

```
import numpy as np
arrayrandom= np.random.randint(1, 50, 20)
print (arrayrandom)
```

O/P:- array([27, 3, 45, 30, 13, 30, 18, 18, 32, 26,
39, 22, 4, 37, 25, 5, 42, 8, 15, 13])

```
# Find minimum value in array - random
print(arrayrandom.min())
```

O/P:- 3

```
# Find maximum value in array - random
print (arrayrandom.max())
```

O/P:- 45

Find Indices of Minimum & maximum values in Numpy Arrays

Code :-

```
import numpy as np
array_random = np.random.randint(1, 50, 20)
print (array_random)
```

o/p :- array([30, 13, 18, 8, 8, 22, 5, 44, 18, 40, 45,
22, 41, 2, 26, 23, 12, 12, 27, 49])

(indices: 0 1 2 3 4 5 6 7 8 9 10 / 11 12 13 14 15 16 17 18 19)

```
# Find index of minimum value in array-random
print(array-random.argmin())
```

o/p :- 13

```
# Find index of maximum value in array-random
print (array-random.argmax())
```

o/p :- 19

Find Shapes of Numpy Arrays and Reshape

MITS

Code

```
① import numpy as np.
  array0 = np.array([])
  print(array0.shape)
#o/p => (0,)

② array1 = np.array([1,2,3,4,5])
  print(array1.shape)
#o/p => (5,)

③ array2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
  print(array2.shape)
#o/p => (3,3)

④ array_zeros = np.zeros((4,5))
  print(array_zeros.shape)
# o/p => (4,5)

⑤ array_ones = np.ones((3,6))
  print(array_ones.shape)
#o/p => (3,6)
```

6. array_ro = np.random.randint(1, 50, 25)
print(array_ro)
#o/p:- array([9, 49, 21, 39, 16, 2, 29, 6, 46, 35, 9, 24, 30,
24, 31, 15, 10, 34, 16, 45, 16, 2, 37, 11, 17])

print(array_ro.shape)
#o/p => (25,)

Reshape

7. print(array_ro.reshape((5,5)))
#o/p :- array([[9, 49, 21, 39, 16],
[2, 29, 6, 46, 35],
[9, 24, 30, 24, 31],
[15, 10, 34, 16, 45],
[16, 2, 37, 11, 17]])

8. print_r1 = np.random.randint(1, 50, 10)
print(print_r1)
#o/p:- array([9, 20, 31, 4, 42, 17, 4, 8, 9, 13])
print(print_r1.reshap((5, 2))
#o/p:- array([[9, 20, 31, 4, 42],
[17, 4, 8, 9, 13]])

Select Items & Group of Items from Numpy Arrays

MITS

Code:-

```
import numpy as np.
① array_0 = np.arange(5)
print(array_0)
```

o/p :- array([0, 1, 2, 3, 4])

```
② # select the item at Index 1 from array_0
array_0[1]  ⇒ o/p: 1
③ # select the items at Indices 1 to 3 inclusive from array_0
array_0[1:4] ⇒ o/p : array([1, 2, 3])
④ # select the items at indices 0 to 3 inclusive from array_0
array_0[:4] # o/p : array([0, 1, 2, 3])
⑤ # select the items starting at index 3 until the end
  of array_0.
array_0[3:]  # o/p : array([3, 4])
```

⑥ array1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(array1)

#o/p:- array([[1,2,3],
[4,5,6],
[7,8,9]])

⑦ # select the item in row 0 and column 0 from array1 using double bracket notation.

array1[0][0] #o/p => 1

Or

⑧ # select the item in row 0 and column 0 from array1 using single bracket notation.

array1[0,0]

⑨ # select first two items from row 0 of array1

print1[:1,:2]

#o/p:- array([[1,2]])

⑩ # select first two items from every row until row 1 inclusive from array1.

print(array1[:2,:2])

#o/p:- array([[1,2],
[4,5]])

(11) # Select first two items from each row of array1

```
array1[:, :2]
```

#o/p:- array([[1, 2],
[4, 5],
[7, 8]])

(12) # Select row 0 from array1

```
array1[0]
```

#o/p:- array([1, 2, 3])

(13) # select every thing before row 2 from array1

```
array1[0:2]        #array1[:2]
```

#o/p:- array([[1, 2, 3],
[4, 5, 6]])

(14)

```
array2 = np.arange(0, 21, 2)
print(array2)
```

#o/p:- array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20])

MITS

(15) # Select the items from array2 that are greater than or equal to 12.

```
print(array2[array2 >=12])
```

#o/p:- array([12, 14, 16, 18, 20])

(16) # select the items from array2 that are greater than 7 and less than 13.

```
print(array2[(array2>7) & (array2<13)])
```

#o/p:- array([8, 10, 12])

Arithmetic operations on Numpy Arrays

MITS

Code :-

```
import numpy as np.
#Initialize arrayA as a one-dimensional numpy Array
#Containing the odd integers between 1 & 20 inclusive.
arrayA = np.arange(1, 21, 2)
print (arrayA)
#O/P : array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])
#Initialize arrayB as a one-Dimensional numpy Array
#Containing integers 1 to 10 inclusive.
arrayB = np.arange(1, 11)
print (arrayB)
#O/P :- array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# Addition of ArrayA & ArrayB.
print (arrayA + arrayB)
#O/P :- array([2, 5, 8, 11, 14, 17, 20, 23, 26, 29])
```

④ # Substraction of arrayA & arrayB.

print (arrayA - arrayB)

#o/p :- array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

⑤ # Multiplication of arrayA & arrayB.

print (arrayA * arrayB)

#o/p :- array([1, 6, 15, 28, 45, 66, 91, 120, 153, 190])

⑥ # Division of ArrayA & arrayB.

print (arrayA / arrayB)

#o/p :- array([1. , 1.5 , 1.666667, 1.75, 1.8 ,

1.83333333, 1.85714286, 1.875 , 1.888889, 1.9])

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

print (arrayC)

#o/p :- array ([[1, 2, 3],
[4, 5, 6]])

⑧ arrayD = np.array([[7, 8, 9], [10, 11, 12]])

print (arrayD)

#o/p :- array ([[7, 8, 9],
[10, 11, 12]])

MITS

⑨ # Addition of arrayC & arrayD

print (arrayC + arrayD)

#o/p:- array ([[8, 10, 12],
[14, 16, 18]])

⑩ # Substraction of arrayC & arrayD

print (arrayC - arrayD)

#o/p:- array([[-6, -6, -6],
[-6, -6, -6]])

⑪ # multiplication of arrayC & arrayD

print (arrayC * arrayD)

#o/p:- array ([[7, 16, 27],
[40, 55, 72]])

⑫ # Division of arrayC & arrayD

print (arrayC / arrayD)

#o/p:- array ([[0.14285714, 0.25 , 0.33333333],
[0.4 , 0.45454545, 0.5]])

Scalar Operations on Numpy Arrays

MITS

Code:-

```
import numpy as np
① arrayA = np.arange(2, 21, 2)
print(arrayA)
#O/P:- array([2,4,6,8,10,12,14,16,18,20])
② # add 3 to each element of Array A, Creating a
  new numpy array.
print(arrayA+3)
#o/p:- array([5,7,9,11,13,15,17,19,21,23])
③ # substract 4 from each element of array A.
print (arrayA-4)
#O/P:- array([-2,0,2,4,6,8,10,12,14,16])
④ # Multiplication each element of arrayA by 5.
print (arrayA *5)
#O/P:- array([10,20,30,40,50,60,70,80,90,100])
⑤ # Divide each element of arrayA by 2
print (arrayA /2)
#O/P:- array([1.,2.,3.,4.,5.,6.,7.,8.,9.,10.])
```

⑥
```
arrayB = np.array([[1,2,3],[4,5,6]])
print(arrayB)
#o/p:- array([[1,2,3],
              [4,5,6]])
```

⑦
```
# addition 3 to each element of arrayB
print(arrayB +3)
#o/p:- array([[4,5,6],
              [7,8,9]])
```

⑧
```
# substraction 4 to each element of arrayB
print(arrayB - 4)
#o/p:- array([[-3,-2,-1]
              [0,1,2]])
```

⑨
```
# Multiplication each element of arrayB by 5
print(arrayB *5)
#o/p:- array([[5, 10, 15],
              [20, 25, 30]])
```

# division each element of arrayB by 2.
print(arrayB/2)
#o/p:- array([[0.5, 1. , 1.5],
[2. , 2.5, 3.]])

Statistical Operations on Numpy Arrays

Code

import numpy as np

① scores = np.random.randint(50, 101, 200)

#Compute the median of scores.

② print(np.median(scores))

#o/p:- 73.5

#Compute the mean of scores

③ print(np.mean(scores))

#o/p:- 74.97

④ # compute the variance of scores

print(np.var(scores))

#o/p:- 223.60910000000004

⑤ # compute the standard deviation of scores.

```
print (np.std(scores))
```

#O/P:- 14.953564792383121

Other operations on Numpy Arrays

Code.

```
import numpy as np.
```

①
```
arrayA = np.arange(6)
print(arrayA)
```

#O/p:- array([0, 1, 2, 3, 4, 5])

②
```
print (np.square(arrayA))
```

#O/P:- array([0, 1, 4, 9, 16, 25])

③
```
arrayB = np.array([36, 49, 64, 81, 100, 121, 144,
                   169, 196, 225])
# To find square root of an ellch element in array B
print (np.sqrt (arrayB))
```

#op:- array([6., 7., 8., 9., 10., 11., 12., 13., 14., 15.])

Expt.No.	Page No.	Date .....

④ # compute the exponential of each item in arrayA
print (np.exp(arrayA))
#o/p:- array([1. , 2.71828183, 7.3890561,
20.08553692, 54.59815003, 148.4131591])

⑤ arrayC = np.arange(1,11)
print (arrayC)
#o/p:- array([1,2,3,4,5,6,7,8,9,10])

⑥ # compute the natural logarithmic of each item in arrayC.
print (np.log(arrayC))
#o/p:- array([0., 0.69314718, 1.09861229, 1.38629436,
1.60943791, 1.79175947, 1.94591015, 2.07944154,
2.19722458, 2.30258509])

⑦ # initialize arrayD as a one-Dimensional numpy array.
# representing some angles in radians..
arrayD = np.array([0, np.pi/6, np.pi/4, np.pi/3,
np.pi/2, 2*np.pi/3, 3*np.pi/4, 5*np.pi/6,
np.pi/2])

```
# compute the sine of each item in arrayD
print (np.sin(arrayD))
# o/p:- array([0., 0.5, 0.70710678, 0.8660254, 1.,
                0.8660254, 0.70710678, 0.5, 1.])
```

⑧
```
arrayE = np.arange(-10, 11)
Print (arrayE)
# o/p:- array([-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0,
                1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

⑨
```
# compute absolute value of each item in arrayE
print(np.abs(arrayE))
# o/p:- array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2,
               3, 4, 5, 6, 7, 8, 9, 10])
```

⑩
```
arrayf = np.arange(21)
print(arrayf)
# o/p:- array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
              14, 15, 16, 17, 18, 19, 20])
```

```
# compute sum of all the items in arrayf.
print (np.sum(arrayf))
# 210.
```

(11)

```
arrayG = np.array([[1,2,3],[4,5,6],[7,8,9]])
Print(arrayG)
#o/p:-
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
print (np.sum(arrayG))
#o/p: 45
# Sum of each column of arrayG.
print(np.sum(arrayG, axis=0)
# o/p: array([12, 15, 18])
# sum of each row of arrayG
print (np.sum(arrayG, axis=1)
# o/p: array([6, 15, 24])
```

Expt.No.	Page No.	Date .....