=============================================

Numpy Module---Data Scienece

=============================================

Index

----------

=>Purpose of Numpy

=>History of Numpy

=>Advantages of Numpy

=>Similarities and Differences Between List and Numpy

=>Importance of "ndarray"

=>Number of Approaches ro create an object of nd array

=>Attributes of ndarray

=>Types of Ndarrays (1-D, 2-D and n-Dimesional Arrays)

=>Basic Indexing and Slicing Operations on ndarray objects

=>Advanced Indexing and Slicing Operations on ndarray objects

=>Slecting or filtering the values of ndarray objects

=>deleting, updating and adding the values of ndrray

=>Arithmetic or Matrix Operations on ndarray

=>Statistical Operations on ndarray

=>copy and view of ndarray

=>Sorting the values of ndarray

==========================================

Numpy

==========================================

Introduction to Numpy:

-------------------------------------

=>Numpy stands for Numerical Python.

=>Numpy is one of the pre-defined third party module / Library and numpy module is not a pre-defined module in Python Language.

=>Syntax for installing any module:

pip install module-name

=>Example: Install numpy module

pip install numpy

=>To use numpy as part of our program, we must import numpy module.

=>A Numpy module is a collection of Variables, Functions and Classes.

===============================================================

History of Numpy:

------------------------------

=>Numpy was developed by studying existing module called "Numeric Library"(origin for development of numpy module)

=>The Numeric Library was developed by JIM HUNGUNIAN

=>The Numeric Library was not able to solve complex maths calculations.

=>Numpy module developed by TRAVIS OLIPHANT for solving complex maths

calculations and array oraganization.

=>Numpy Module developed in the year 2005

=>Numpy Module developed in C and PYTHON languages.

==================================================== Advantages of using NumPy

====================================================

-------------------------------------

Need of NumPy:

-------------------------------------

=>With the revolution of data science, data analysis libraries like NumPy, SciPy, Scikit, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.

=>NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix Operations and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

-----------------------------------------------------------------------

The advantages of Numpy Programming are:

-----------------------------------------------------------------------

1) With Numpy Programming, we can deal with Arrays such 1-D, 2-D and Multi-Dimensional Arrays.

2) NumPy maintains minimal memory for large sets of data:

3) Numpy provides Fast in Performing Operations bcoz internally its data is available at same address.

4) NumPy performs array-oriented computing.

5) It efficiently implements the multidimensional arrays.

6) It performs scientific computations.

7) It is capable of performing reshaping the data stored in multidimensional arrays.

8) NumPy provides Many in-built functions for Various Complex Mathematical Operations such as statistical , financial, trigonometric Operations etc.

===================================================

Python Traditional List VS Numpy Module

===================================================

Similarities of python Traditional List VS Numpy Module:

-----------------------------------------------------------

=>An object of list used to store multiple values of same type or different type and both types (unique +duplicates) in single object.

=>In Numpy Programming, the data is organized in the object of "ndarray", which is one of the pre-defined class in numpy module. Hence an object of ndarray can store same type or different type and both types (unique +duplicates) in single object.

=>The objects of ndarray and list are mutable (changes can takes place)

-----------------------------------------------------------------------

Differences between Python Traditional List and ndarray object of Numpy Module:

----------------------------------------------------------------------

=>An object of list contains both homogeneous and hetrogeneous values where as an object of ndarray of numpy can store only similar type of values(even we store different values, internally they are treated as similar type by treating all values of type "object" ).

=>On the object of list, we can't perform Vector Based Operations. Where as on the object of ndarray, we can perform Vector based operations.

=>In large sampling of data, List based applications takes more memory space where ndarray object takes less memory space.

=>List based applications are not effiecient bcoz list object values takes more time to extract or retrive ( they are available at different Address) where as numpy based applications are efficient bcoz of ndarray object values takes less to time to extract or retrive( they are available at same Address / clustered).

=>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.

=================================================

Number of approaches to create an object of ndarray

=================================================

=>"ndarray" is one of the pre-defined class of numpy module and whose object is used for storing the data in numpy programming in the form of 1-D, 2-D and n-Dimensional Arrays.

=>In numpy programming, we have the following essential approaches to create an object of ndarray.

1. array()

2. arange()

3. zeros()

4. ones()

5. full()

6. identity()

7.hstack()

8.vstack()

-----------------------------------------------------------------------

1) array ():

----------------------------------

=>This Function is used for converting Traditional Python Objects into ndarray object.

=>Syntax:- varname=numpy.array( Object,dtype )

=>Here var name is an object of <class,ndarray>

=>here array() is pre-defined function of numpy module used for converting Traditional Python Objects into ndrray object.

=>object represents any Traditional Python Objects

=>dtype represents any numpy data type such as int8,int16,int32,float16, float 32, float64,....etc (Internal dfata types of C lang)

Examples:

------------------

>>> import numpy as np

>>> l1=[10,20,30,40,50,60]

>>> print(l1,type(l1))-----------------[10, 20, 30, 40, 50, 60] <class 'list'>

>>> a=np.array(l1)

>>> print(a,type(a))----------------[10 20 30 40 50 60] <class 'numpy.ndarray'>

>>> t=(10,20,30,40,50,60,70)

>>> print(t,type(t))--------------(10, 20, 30, 40, 50, 60, 70) <class 'tuple'>

>>> a=np.array(t)

>>> print(a,type(a ))--------------[10 20 30 40 50 60 70] <class 'numpy.ndarray'>

>>> d1={10:1.2,20:4.5,30:6.7}

>>> a=np.array(d1)

>>> a----array({10: 1.2, 20: 4.5, 30: 6.7}, dtype=object)

---------------------------------------------------------------------------------------

>>> t=(10,20,30,40,50,60)

>>> a=np.array(t)

>>> a--------------array([10, 20, 30, 40, 50, 60])

>>> a.ndim------------1

>>> a.dtype----------dtype('int32')

>>> a.shape-------------(6,)

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

>>> c=a.reshape(2,3)

>>> b--------------

array([[10, 20],

[30, 40],

[50, 60]])

>>> c

array([[10, 20, 30],

[40, 50, 60]])

>>> print(b,type(b))

[[10 20]

[30 40]

[50 60]] <class 'numpy.ndarray'>

>>> print(c,type(c))

[[10 20 30]

[40 50 60]] <class 'numpy.ndarray'>

>>> b.ndim-------------2

>>> c.ndim------------2

>>> b.shape---------------(3, 2)

>>> c.shape-------------(2, 3)

>>> d=a.reshape(3,3)-------ValueError: cannot reshape array of size 6 into shape (3,3)

----------------------------------------------------------------------------------------------------------------------------

>>> t1=((10,20),(30,40))

>>> print(t1,type(t1))--------------((10, 20), (30, 40)) <class 'tuple'>

>>> a=np.array(t1)

>>> a

array([[10, 20],

[30, 40]])

>>> a.ndim----------2

>>> a.shape----------(2, 2)

------------------------------------------------------------------------------------------------------------

>>> t1=( ((10,20,15),(30,40,25)),( (50,60,18),(70,80,35) ))

>>> print(t1,type(t1))

(((10, 20, 15), (30, 40, 25)), ((50, 60, 18), (70, 80, 35))) <class 'tuple'>

>>> a=np.array(t1)

>>> a

array([[[10, 20, 15],

[30, 40, 25]],

[[50, 60, 18],

[70, 80, 35]]])

>>> print(a)

[[[10 20 15]

[30 40 25]]

[[50 60 18]

[70 80 35]]]

>>> a.ndim

3

>>> a.shape

(2, 2, 3)

>>> b=a.reshape(4,3)

>>> b

array([[10, 20, 15],

[30, 40, 25],

[50, 60, 18],

[70, 80, 35]])

>>> c=a.reshape(3,4)

>>> c

array([[10, 20, 15, 30],

[40, 25, 50, 60],

[18, 70, 80, 35]])

>>> d=a.reshape(3,2,2)

>>> d

array([[[10, 20],

[15, 30]],

[[40, 25],

[50, 60]],

[[18, 70],

[80, 35]]])

>>> d[0]

array([[10, 20],

[15, 30]])

>>> d[1]

array([[40, 25],

[50, 60]])

>>> d[2]

array([[18, 70],

[80, 35]])

-----------------------------------------------------------------------

2. arange ():

-----------------------------------------------------------------------

Syntax1:- varname=numpy.arange(Value)

Syntax2:- varname=numpy.arange(Start,Stop)

Syntax3:- varname=numpy.arange(Start,Stop,Step)

=>Here var name is an object of <class,ndarray>

=>Syntax-1 creates an object of ndarray with the values from 0 to value-1

=>Syntax-2 creates an object of ndarray with the values from Start to Stop-1

=>Syntax-3 creates an object of ndarray with the values from Start to Stop-1 with equal

Interval of Value-----step

=>arange() always create an object of ndarray in 1-D array only but not Possible to create directly 2-D and Multi Dimesional Arrays.

=>To create 2-D and Multi Dimesional Arrays, we must use reshape() or shape attribute

Examples:

-----------------

>>> import numpy as np

>>> a=np.arange(10)

>>> a-----------array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> a.ndim--------1

>>> a=np.arange(50,62)

>>> print(a,type(a))---[50 51 52 53 54 55 56 57 58 59 60 61] <class 'numpy.ndarray'>

>>> a.ndim------1

>>> a=np.arange(10,23,2)

>>> a-----array([10, 12, 14, 16, 18, 20, 22])

>>> a=np.arange(10,22,2)

>>> a--------array([10, 12, 14, 16, 18, 20])

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

>>> c=a.reshape(3,2)

>>> b-----

array([[10, 12, 14],

[16, 18, 20]])

>>> c

array([[10, 12],

[14, 16],

[18, 20]])

>>> b.ndim------ 2

>>> c.ndim------- 2

>>> b.shape-----(2, 3)

>>> c.shape-----(3, 2)

>>> l1=[ [[10,20],[30,40]], [[15,25],[35,45]] ]

>>> l1----------[[[10, 20], [30, 40]], [[15, 25], [35, 45]]]

>>> a=np.arange(l1)----------TypeError: unsupported operand type(s) for -: 'list' and 'int'

==================================================================

3. zeros():

------------------------

=>This Function is used for building ZERO matrix either with 1-D or 2-D or n-D

=>Syntax: varname=numpy.zeros(shape,dtype)

=>Here Shape can be 1-D(number of Zeros) or 2-D(Rows,Cols) or n-D( Number of Matrices,Number of Rows, Number of Columns)

-----------------------

Examples:

--------------------

>>> import numpy as np

>>> a=np.zeros(12)

>>> a-----------array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

>>> a=np.zeros(12,dtype=int)

>>> a------------array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

>>> a.reshape(3,4)

array([[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]])

>>> a.reshape(4,3)

array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0],

[0, 0, 0]])

>>> a.reshape(6,2)

array([[0, 0],

[0, 0],

[0, 0],

[0, 0],

[0, 0],

[0, 0]])

>>> a.reshape(2,6)

array([[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]])

>>> a.reshape(2,3,2)

array([[[0, 0],

[0, 0],

[0, 0]],

[[0, 0],

[0, 0],

[0, 0]]])

>>> a.reshape(2,2,2,2)------ValueError: cannot reshape array of size 12 into shape (2,2,2,2)

>>> a.reshape(3,2,2)

array([[[0, 0],

[0, 0]],

[[0, 0],

[0, 0]],

[[0, 0],

[0, 0]]])

>>> a.reshape(2,3,2)

array([[[0, 0],

[0, 0],

[0, 0]],

[[0, 0],

[0, 0],

[0, 0]]])

>>> a.reshape(2,2,3)

array([[[0, 0, 0],

[0, 0, 0]],

[[0, 0, 0],

[0, 0, 0]]])

----------------------------------------------------------------------------

>>> import numpy as np

>>> a=np.zeros((3,3),dtype=int)

>>> a

array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]])

>>> a=np.zeros((2,3))

>>> a

array([[0., 0., 0.],

[0., 0., 0.]])

>>> a=np.zeros((2,3),int)

>>> a

array([[0, 0, 0],

[0, 0, 0]])

>>> a=np.zeros((3,2,3),dtype=int)

>>> a

array([[[0, 0, 0],

[0, 0, 0]],

[[0, 0, 0],

[0, 0, 0]],

[[0, 0, 0],

[0, 0, 0]]])

>>> print(a,type(a))

[[[0 0 0]

[0 0 0]]

[[0 0 0]

[0 0 0]]

[[0 0 0]

[0 0 0]]] <class 'numpy.ndarray'>

-------------------------------------------------------------------------------------------------------------------------------

4. ones()

---------------------------------------

=>This Function is used for building ONEs matrix either with 1-D or 2-D or n-D

=>Syntax: varname=numpy.ones(shape,dtype)

=>Here Shape can be 1-D(number of ones) or 2-D(Rows,Cols) or n-D( Number of Matrices,Number of Rows, Number of Columns)

Examples:

-----------------------------

>>> import numpy as np

>>> a=np.ones(10)

>>> print(a,type(a))----------[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] <class 'numpy.ndarray'>

>>> a=np.ones(10,dtype=int)

>>> print(a,type(a))-------------[1 1 1 1 1 1 1 1 1 1] <class 'numpy.ndarray'>

>>> a.shape-----------(10,)

>>> a.shape=(5,2)

>>> a

array([[1, 1],

[1, 1],

[1, 1],

[1, 1],

[1, 1]])

>>> a.ndim-------------- 2

>>> a.shape------------ (5, 2)

>>> a.shape=(2,5)

>>> a

array([[1, 1, 1, 1, 1],

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

>>> a.shape----------------------(2, 5)

>>>

>>> a=np.ones((3,4),dtype=int)

>>> a

array([[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]])

>>> a=np.ones((4,3),dtype=int)

>>> print(a,type(a))

[[1 1 1]

[1 1 1]

[1 1 1]

[1 1 1]] <class 'numpy.ndarray'>

>>> a.shape----------(4, 3)

>>> a.shape=(3,2,2)

>>> a

array([[[1, 1],

[1, 1]],

[[1, 1],

[1, 1]],

[[1, 1],

[1, 1]]])

>>> a=np.ones((4,3,3),dtype=int)

>>> a

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],

[1, 1, 1]],

[[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]])

================================================================

5) full()

-------------------------

=>This is function is used for building a matrix by specifying fill value either 1-D or 2-D or n-D

=>Syntax:-

varname=numpy.full(shape,fill\_value,dtype)

=>varname is an obejct of <class, numpy.ndarray>

=>Here Shape can be 1-D(number of Fill\_Value) or 2-D(Rows,Cols) or n-D( Number of Matrices,Number of Rows, Number of Columns)

=>fill\_value can be any number of programmer choice

Examples:

-------------------

>>> a=np.full(3,1)

>>> a---------array([1, 1, 1])

>>>print(type(a))--------<class,numpy.ndarray>

>>> a=np.full(3,9)

>>> a------------array([9, 9, 9])

>>> a=np.full(6,8)

>>> a-------------array([8, 8, 8, 8, 8, 8])

>>> a.shape=(3,2)

>>> a

array([[8, 8],

[8, 8],

[8, 8]])

>>> a=np.full(6,9)

>>> a----------array([9, 9, 9, 9, 9, 9])

>>> a.reshape(2,3)

array([[9, 9, 9],

[9, 9, 9]])

>>> a=np.full((3,3),9)

>>> a

array([[9, 9, 9],

[9, 9, 9],

[9, 9, 9]])

>>> a=np.full((2,3),6)

>>> a

array([[6, 6, 6],

[6, 6, 6]])

>>> a.reshape(3,2)

array([[6, 6],

[6, 6],

[6, 6]])

>>> a=np.full((3,3,3),7)

>>> a

array([[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]],

[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]],

[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]]])

==================================================================

6) identity():

----------------------------------

=>This function always bulid Identity or unit matrix

=>Syntax:- varname=numpy.identity(N,dtype)

=>Here N represents Either we can take Rows or Columns and PVM takes as NXN Matrix (Square Matrix--Unit or Identity)

Examples:

--------------------------

>>> import numpy as np

>>> a=np.identity(3,dtype=int)

>>> print(a,type(a))-------------

[[1 0 0]

[0 1 0]

[0 0 1]] <class 'numpy.ndarray'>

>>> a=np.identity(5,dtype=int)

>>> print(a,type(a))

[[1 0 0 0 0]

[0 1 0 0 0]

[0 0 1 0 0]

[0 0 0 1 0]

[0 0 0 0 1]] <class 'numpy.ndarray'>

===================================X====================================

========================================

Numpy---Basic Indexing

=================================================

==>If we want to access Single element of 1D,2D and N-D arrays we must use the concept of Basic Indexing.

--------------------------------------------------------------------

=>Accessing Single Element 1D-Array :

---------------------------------------------------------------------

=>Syntax:- ndarrayname [ Index ]

=>Here 'index' can be either either +ve or -ve indexing

----------------

Examples:

------------------

>>> a=np.array([10,20,30,40,50,60])

>>> a

array([10, 20, 30, 40, 50, 60])

>>> a[0]

10

>>> a[3]

40

-----------------------------------------------------------------------------------------------------------

=>Accessing single Element of 2D :

----------------------------------------------------------------------------------------------------------

=>Syntax:- ndarrayobj[ row index , column index ]

----------------

Examples:-

---------------

>>>import numpy as np

>>> a=np.array([10,20,30,40,50,60])

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

>>> b

array([[10, 20, 30],

[40, 50, 60]])

>>> b[0,0]

10

>>> b[0,1]

20

>>> b[1,2]

60

======================================================================

=>Accessing single Element of 3D :

----------------------------------------------------------------------------------------------------------------------------

Syntax:- ndarrayobj[ Index of matrix , row index , column index ]

-------------

Examples:

---------------

>>> a=np.array([10,20,30,40,50,60,70,80])

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

>>> b

array([[[10, 20],

[30, 40]],

[[50, 60],

[70, 80]]])

>>> b[0,0,0]-----------10

>>> b[-1,0,0]---------50

>>> b[-2,1,1]---------40

=======================================================

Numpy---Indexing and Slicing Operations of 1D,2D and 3D array

=======================================================

------------------------------------

1D Arrays Slicing:

------------------------------------

Syntax:- 1dndrrayobj[begin:end:step]

-----------------------

Examples:

-----------------------

>>> a=np.array([10,20,30,40,50,60,70])

>>> a------------array([10, 20, 30, 40, 50, 60, 70])

>>> a[::-1]-----------array([70, 60, 50, 40, 30, 20, 10])

>>> a[::]-----------array([10, 20, 30, 40, 50, 60, 70])

------------------------------------

2D Arrays Slicing:

------------------------------------

Syntax:- ndrrayobj[ i , j ]

here 'i' represents Row Index

here 'j' represents Column Index

(OR)

Syntax:- 2dndrrayobj[Row Index, Column Index]

Syntax:- 2dndrrayobj[begin:end:step, begin:end:step]

--------------------------------------------------------------------

Examples:

--------------------------------------------------------------------

>>> a=np.array([[10,20,30],[40,50,60]])

>>> a

array([[10, 20, 30],

[40, 50, 60]])

>>> a[0,0]

10

>>> a[0:,0:1]

array([[10],

[40]])

>>> a[0:,1:2]

array([[20],

[50]])

>>> a[1:,:]

array([[40, 50, 60]])

===============================================================

3D Arrays Slicing

-----------------------------

Syntax:- 3dndrrayobj[i,j,k]

here 'i' represents Which 2D matrix ( Matrix Number-->0 1 2 3 4 5...... )

here 'j' represents which Rows in that 2D matrix

here 'k' represents which Columns in that 2D matrix

(OR)

Syntax:- 3dndrrayobj[ Matrix Index, Row Index, Column Index ]

(OR)

Syntax:- 3dndrrayobj[begin:end:step, begin:end:step, begin:end:step ]

----------------------

Examples:

----------------------

>>> lst=[ [ [1,2,3],[4,5,6],[7,8,9] ],[ [13,14,15],[16,17,18],[19,20,21] ] ]

>>> print(lst)

[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[13, 14, 15], [16, 17, 18], [19, 20, 21]]]

>>> arr2=np.array(lst)

>>> print(arr2)

[[[ 1 2 3]

[ 4 5 6]

[ 7 8 9]]

[[13 14 15]

[16 17 18]

[19 20 21]]]

>>> arr2.ndim

3

>>> arr2.shape

(2, 3, 3)

>>> arr2[:,:,0:1]

array([[[ 1],

[ 4],

[ 7]],

[[13],

[16],

[19]]])

>>> arr2[:,:,:1]

array([[[ 1],

[ 4],

[ 7]],

[[13],

[16],

[19]]])

>>> arr2[: , 0:2, 1:3]

array([[[ 2, 3],

[ 5, 6]],

[[14, 15],

[17, 18]]])

>>> arr2[: , :2, 1:]

array([[[ 2, 3],

[ 5, 6]],

[[14, 15],

[17, 18]]])

=================================================

NumPy---Advanced Indexing

=================================================

==>If we want to access multiple elements, which are not in order (arbitrary elements) of 1D,2D and N-D arrays we must use the concept of Advanced Indexing.

=>If we want access the elements based on some condition then we can't use basic indexing and Basic Slicing Operations. To fullfill such type of requirements we must use advanced Indexing.

--------------------------------------------------------------------

=>Accessing Multiple Arbitrary Elements ---1D :

---------------------------------------------------------------------

=>Syntax:- ndarrayname [ x ]

=>Here 'x' can be either ndarray or list which represents required indexes of arbitrary elements.

----------------

Examples:

------------------

>>> lst=[10,20,30,40,50,60,70,80,90]

>>> a=np.array(lst)

>>> print(a)----------------[10 20 30 40 50 60 70 80 90]

#access 10 30 and 80 elements

# here indexes of 10 30 and 80 are 0 2 7

>>>lst=[0,2,7] here [0,2,7] are indexes of 10 30 and 80

>>> indexes=np.array(lst) # here lst converted into ndarray object

>>> print(indexes)---------[0 2 7]

>>> print(a[indexes])--------------[10 30 80]

(OR)

>>> ind=[0,2,7] # prepare the list of indexes of arbitray elements(10,30,80) of ndarray and pass to ndarray

>>> print(a[ind]) -----------[10 30 80]

OR

>>> print(a[[0,2,7] ]) -----------[10 30 80]

Examples:

---------------------

Q1-->Access 20 30 80 10 10 30

>>> lst=[10,20,30,40,50,60,70,80,90]

>>> a=np.array(lst)

>>> print(a)----------------[10 20 30 40 50 60 70 80 90]

>>> ind=[1,2,7,0,0,2] # [1,2,7,0,0,2] are the indexes of 20 30 80 10 10 30

>>> print(a[ind])----------------[20 30 80 10 10 30]

-----------------------------------------------------------------------------------------------------------

=>Accessing Multiple Arbitrary Elements ---2D :

----------------------------------------------------------------------------------------------------------

=>Syntax:- ndarrayobj[ [row indexes],[column indexes] ]

Examples:-

---------------

>>>import numpy as np

>>>mat=np.array([ [1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16] ] )

>>> print(mat)

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16]]

Q1) Access the principle diagnal elements 1 6 11 16

Ans:- mat[ [0,1,2,3],[0,1,2,3] ]

=>When the above statement is executed, The PVM takes internally as

mat[ (0,0), (1,1), (2,2),(3,3) ]-------- 1 6 11 16

>>> mat[ [0,1,2,3],[0,1,2,3] ]-----------array([ 1, 6, 11, 16])

Q2) Access the elements 6 14

Ans: mat[ [1,3] , [1,1] ]

=>When the above statement is executed, The PVM takes internally as

mat[ (1,1),(3,1) ]

>>> mat[[1,3],[1,1]]-----------array([ 6, 14])

======================================================================

=>Accessing Multiple Arbitrary Elements ---3D :

----------------------------------------------------------------------------------------------------------------------------

Syntax:- ndarray[ [Indexes of 2Dmatrix],[row indexes],[column indexes] ]

-------------

----------------

Examples:

---------------

>>>import numpy as np

>>>l1=[ [ [1,2,3,4],[5,6,7,8],[9,10,11,12] ],[ [13,14,15,16],[17,18,19,20],[21,22,23,24] ] ]

>>>mat3d=np.array(l1)

>>>print(mat3d)

>>> print(mat3d)

[[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

[[13 14 15 16]

[17 18 19 20]

[21 22 23 24]]]

>>> mat3d.ndim

3

>>> mat3d.shape

(2, 3, 4)

----------------------------------------

Q1) Access the elements 1 14 24

Ans:- mat3d[ [0,1,1], [0,0,2], [0,1,3] ]

When the above statement is executed, Internally PVM takes as follows.

=>mat3d[ (0,0,0),(1,0,1),(1,2,3) ]-Gives-->1 14 24

Q1) Access the elements 10 16

>>> mat3d[[-2,-1],[-1,-3],[-3,-1]]----------array([10, 16])

====================================================

OR

========

>>> l1=[ [ [1,2,3,4],[5,6,7,8],[9,10,11,12] ],[ [13,14,15,16],[17,18,19,20],[21,22,23,24] ] ]

>>> a=np.array(l1)

>>> a

array([[[ 1, 2, 3, 4],

[ 5, 6, 7, 8],

[ 9, 10, 11, 12]],

[[13, 14, 15, 16],

[17, 18, 19, 20],

[21, 22, 23, 24]]])

>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]---Syntax

>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]

>>> #access 1,8,13,20

>>> matind=(0,0,1,1)

>>> rowind=(0,1,0,1)

>>> colind=(0,3,0,3)

>>> a[matind,rowind,colind]

array([ 1, 8, 13, 20])

>>> a[ [0,0,0,1,1,1],[0,1,2,0,1,2],[0,1,2,0,1,2] ]

array([ 1, 6, 11, 13, 18, 23])

===========================X==========================================

a=np.array([10,20,30,40,50,60,70,80,15,25,35,45,55,65,75,85])

print(a)

a.shape=(2,2,2,2)

print(a)

[[[[10 20]

[30 40]]

[[50 60]

[70 80]]]

[[[15 25]

[35 45]]

[[55 65]

[75 85]]]]

#access 10 from a---4-D

a[0][0][0][0]------------10

# access 10 and 40 from a---4D

a[[0,0],[0,0],[0,1],[0,1]]----array([10, 40])

# access 60,55 and 15 from a---4D

a[ [0,1,1],[1,1,0],[0,0,0],[1,0,0] ]----array([60, 55, 15])

=========================================

Numpy--Arithmetic Operations (OR) Matrix Operations

=========================================

=>On the objects of ndarray, we can apply all types of Arithmetic Operators.

=>To perform Arithmetic Operations on the objects of ndarray in numpy programming, we use the following functions.

a) add()

b) subtract()

c) multiply()

d) dot() or matmul()

e) divide()

f) floor\_divide()

g) mod()

h) power()

=>All the arithmetic Functions can also be perfomed w.r.t Arithmetic Operators.

=>All these Arithmetic Operations are called Matrix Operations.

---------------

a) add():

--------------

Syntax:- varname=numpy.add(ndarrayobj1, ndarrayobj2)

=>This function is used for adding elements of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

-----------------

>>> l1=[ [10,20],[30,40] ]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=np.add(a,b)

>>> c

array([[11, 22],

[33, 44]])

--------------------------------------------------------------------------------

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=a+b # we used operator + instead of add()

>>> c

array([[11, 22],

[33, 44]])

=================================

b) subtract()

------------------------------

Syntax:- varname=numpy.subtract(ndarrayobj1, ndarrayobj2)

=>This function is used for subtracting elements of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

------------------

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=np.subtract(a,b)

>>> c

array([[ 9, 18],

[27, 36]])

-----------------------------------

>>> d=a-b # we used operator - instead of subtract()

>>> d

array([[ 9, 18],

[27, 36]])

===================================

c) multiply():

-----------------------

Syntax:- varname=numpy.multiply(ndarrayobj1, ndarrayobj2)

=>This function is used for performing element-wise multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

>>> l1=[[1,2],[3,4]]

>>> l2=[[5,6],[4,3]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[1, 2],

[3, 4]])

>>> b

array([[5, 6],

[4, 3]])

>>> c=np.multiply(a,b)

>>> c

array([[ 5, 12],

[12, 12]])

-----------------------------------------------

>>> e=a\*b # we used operator \* instead of multiply()

>>> e

array([[ 5, 12],

[12, 12]])

------------------------------------------

d) dot() (or) matmul()

=>To perform Matrix Multiplication, we use dot(), matmul()

Syntax:- varname=numpy.dot(ndarrayobj1, ndarrayobj2)

Syntax:- varname=numpy.matmul(ndarrayobj1, ndarrayobj2)

=>These functions is used for performing actual matrix multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

-----------------

Examples:

>>> l1=[[1,2],[3,4]]

>>> l2=[[5,6],[4,3]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[1, 2],

[3, 4]])

>>> b

array([[5, 6],

[4, 3]])

>>> d=np.dot(a,b)

>>> d

array([[13, 12],

[31, 30]])

>>> e=np.matmul(a,b)

>>> e

array([[13, 12],

[31, 30]])

-----------------------------------------------------------------------

e) divide()

-----------------------------------

Syntax:- varname=numpy.divide(ndarray1,ndarry2)

=>This function is used for performing element-wise division of ndarrayobj1, ndarrayobj2 and result can be displayed

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=np.divide(a,b)

>>> c

array([[10., 10.],

[10., 10.]])

---------------------------------------------------------------

>>> d=a/b # we used operator / instead of divide()

>>> d

array([[10., 10.],

[10., 10.]])

---------------------------------------------------------------------------------------------

f) floor\_divide()

-----------------------------------

Syntax:- varname=numpy.floor\_divide(ndarray1,ndarry2)

=>This function is used for performing element-wise floor division of ndarrayobj1, ndarrayobj2 and result can be displayed

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=np.floor\_divide(a,b)

>>> c

array([[10, 10],

[10, 10]])

---------------------------------------------------------------

>>> d=a//b # we used operator // instead of floor\_divide()

>>> d

array([[10, 10],

[10, 10]])

--------------------------------------------------------------------------------------------------------------------

g) mod()

-------------------------------

Syntax:- varname=numpy.mod(ndarray1,ndarry2)

=>This function is used for performing element-wise modulo division of ndarrayobj1, ndarrayobj2 and result can be displayed

--------------------

Examples:

---------------------

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>> c=np.mod(a,b)

>>> c

array([[0., 0.],

[0., 0.]])

------------------------------------------------------------------------

=>We can also do with operator %

>>> e=a%b

>>> e

array([[0, 0],

[0, 0]], dtype=int32)

-----------------------------------------------------------------------------------------------------

h) power():

---------------------------------------

Syntax:- varname=numpy.power(ndarray1,ndarry2)

=>This function is used for performing element-wise exponential of ndarrayobj1, ndarrayobj2 and result can be displayed

---------------------------------------

>>> l1=[[10,20],[30,40]]

>>> l2=[[1,2],[3,4]]

>>> a=np.array(l1)

>>> b=np.array(l2)

>>> a

array([[10, 20],

[30, 40]])

>>> b

array([[1, 2],

[3, 4]])

>>>c=np.power(a,b)

>>>print(c)

array([[ 10, 400],

[ 27000, 2560000]],

--------------------------------------------

>>> f=a\*\*b # Instead of using power() we can use \*\* operator

>>> f

array([[ 10, 400],

[ 27000, 2560000]], dtype=int32)

==================================X=======================

===========================================

Numpy--Statistical Operations

===========================================

=>On the object of ndarray, we can the following Statistical Operations .

a) amax()

b) amin()

c) mean()

d) median()

e) var()

f) std()

=>These operation we can perform on the entire matrix and we can also peform on columnwise (axis=0) and Rowwise (axis=1)

a) amax():

-------------------

=>This function obtains maximum element of the entire matrix.

=>Syntax1:- varname=numpy.amax(ndarrayobject)

=>Syntax2:- varname=numpy.amax(ndarrayobject,axis=0)--->obtains max

elements on the basis of columns.

=>Syntax3:- varname=numpy.amax(ndarrayobject,axis=1)--->obtains max

elements on the basis of Rows.

Examples:

-------------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> max=np.amax(A)

>>> cmax=np.amax(A,axis=0)

>>> rmax=np.amax(A,axis=1)

>>> print ("Max element=",max)-----------Max eleemnt= 4

>>> print("Column Max eleemnts=",cmax)---Column Max eleemnts= [4 4 3]

>>> print("Row Max eleemnts=",rmax)---Row Max eleemnts= [3 4 4]

-----------------------------------------------------------------------------------------------------

b) amin ():

-------------------

=>This function obtains minmum element of the entire matrix.

=>Syntax1:- varname=numpy.amin(ndarrayobject)

=>Syntax2:- varname=numpy.amin(ndarrayobject,axis=0)--->obtains min elements on the basis of columns.

=>Syntax3:- varname=numpy.amin(ndarrayobject,axis=1)--->obtains min elements on the basis of Rows.

Examples:

-------------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> min=np.amin(A)

>>> cmin=np.amin(A,axis=0)

>>> rmin=np.amin(A,axis=1)

>>> print("Min eleemnt=",min)---Min eleemnt= 1

>>> print("Column Min eleemnts=",cmin)---Column Min eleemnts= [1 2 1]

>>> print("Row Min eleemnts=",rmin)---Row Min eleemnts= [1 1 2]

---------------------------------------------------------------------------------------------------

c) mean():

-----------------

=>This is used for cal mean of the total matrix elements.

=>The formula for mean=(sum of all elements of matrix) / total number of elements.

Syntax1:- varname=numpy.mean(ndarrayobject)

Syntax2:- varname=numpy.mean(ndarrayobject,axis=0)--->Columnwise Mean

Syntax3:- varname=numpy.mean(ndarrayobject,axis=1)--->Rowwise Mean

Examples:

-----------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> m=np.mean(A)

>>> cm=np.mean(A,axis=0)

>>> rm=np.mean(A,axis=1)

>>> print("Mean=",m)-------Mean= 2.4444444444444446

>>> print("Column Mean=",cm)--Column Mean= [2.66666667 2.66666667 2. ]

>>> print("Row Mean=",rm)---Row Mean= [ 2. 2.33333333 3. ]

------------------------------------------------------------------------------------------------------

d) median()

---------------------

=>This is used for calculating / obtaining median of entire matrix elements.

=>Median is nothing but sorting the given data in ascending order and select middle element.

=>If the number of sorted elements are odd then center or middle element becomes median.

=>If the number sorted elements are even then select center or middle of two elements, add them and divided by 2 and that result becomes median.

Syntax1:- varname=numpy.median(ndarrayobject)

Syntax2:- varname=numpy.median(ndarrayobject,axis=0)

Syntax3:- varname=numpy.median(ndarrayobject,axis=1)

Examples:

--------------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> md=np.median(A)

>>> cmd=np.median(A,axis=0)

>>> rmd=np.median(A,axis=1)

>>> print("Median=",md)----Median= 2.0

>>> print("Column Median=",cmd)---Column Median= [3. 2. 2.]

>>> print("Row Median=",rmd)------Row Median= [2. 2. 3.]

>>> l1=[[2,3],[4,1]]

>>> A=np.array(l1)

>>> print(A)

[[2 3]

[4 1]]

>>> md=np.median(A)

>>> cmd=np.median(A,axis=0)

>>> rmd=np.median(A,axis=1)

>>> print("Median=",md)---Median= 2.5

>>> print("Column Median=",cmd)---Column Median= [3. 2.]

>>> print("Row Median=",rmd)---Row Median= [2.5 2.5]

-----------------------------------------------------------------------------------------

e) var():

-------------

Variance= sqr(mean-xi) / total number of elements

here 'xi' represents each element of matrix.

------------------

Syntax1:- varname=numpy.var(ndarrayobject)

Syntax2:- varname=numpy.var(ndarrayobject,axis=0)

Syntax3:- varname=numpy.var(ndarrayobject,axis=1)

--------------------

Examples:

--------------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> vr=np.var(A)

>>> cvr=np.var(A,axis=0)

>>> rvr=np.var(A,axis=1)

>>> print("Variance=",vr)----Variance= 1.1358024691358024

>>> print("Column Variance=",cvr)---Column Variance= [1.55555556 0.88888889 0.66666667]

>>> print("Row Variance=",rvr)---Row Variance=[0.66666667 1.55555556 0.66666667]

--------------------------------------------------------------

f) std()

------------------

standard deviation=sqrt(var)

Syntax1:- varname=numpy.std(ndarrayobject)

Syntax2:- varname=numpy.std(ndarrayobject,axis=0)

Syntax3:- varname=numpy.std(ndarrayobject,axis=1)

-------------------------------

Examples:

----------------

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]

>>> A=np.array(l1)

>>> print(A)

[[1 2 3]

[4 2 1]

[3 4 2]]

>>> vr=np.var(A)

>>> cvr=np.var(A,axis=0)

>>> rvr=np.var(A,axis=1)

>>> print("Variance=",vr)---Variance= 1.1358024691358024

>>> print("Column Variance=",cvr)---Column Variance= [1.55555556 0.88888889 0.66666667]

>>> print("Row Variance=",rvr)---Row Variance= [0.66666667 1.55555556 0.66666667]

--------------------------------------------------------------------------------------------------

>>> sd=np.std(A)

>>> csd=np.std(A,axis=0)

>>> rsd=np.std(A,axis=1)

>>> print("std=",sd)---std= 1.0657403385139377

>>> print(" column std=",csd)--- column std= [1.24721913 0.94280904 0.81649658]

>>> print("Row std=",rsd)--Row std= [0.81649658 1.24721913 0.81649658]

==========================X=====================================

Note: numpy module does not contain mode().

mode() present in statistics module of Python

mode() gives Highest Frequent Element in given object

Examples:

----------------

>>> import statistics as s

>>> l1=[10,20,30,10,20,40,10]

>>> s.mode(l1)-------------10

>>> l1=[10,20,30,10,20,40,10,20]

>>> s.mode(l1)-------------10

>>> l1=[20,10,30,10,20,40,10,20]

>>> s.mode(l1)------------------20

>>> s.multimode(l1)--------------[20, 10]

-------------------------------

>>> a=np.array(l1)

>>> s.mode(a)----------------20

>>> s.multimode(a)-----------[20, 10]

-------------------------------------------------

NumPy Sorting Arrays

--------------------------------------------------

=>Sorting is nothing arranging the elements in an ordered sequence.

=>Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

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

Examples:

-------------------

import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr)) # [0 1 2 3]

----------------------------------------------------

import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr)) # ['apple' 'banana' 'cherry']

-----------------------------------------------------

import numpy as np

arr = np.array([True, False, True])

print(np.sort(arr)) # [False True True]

-------------------------------------------------------

Sorting a 2-D Array

--------------------------------------------------------

If you use the sort() method on a 2-D array, both columns and Rows of nd array will be sorted.

-------------------

Examples:

-------------------

import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))

#output

[[2 3 4]

[0 1 5]]

--------------------------------------------------------------------------------------------------

a=np.array([110, 20, -30, 40, 50, 160, 7, 8, 90])

print(a)

np.sort(a)-----------array([-30, 7, 8, 20, 40, 50, 90, 110, 160])

np.sort(a)[::-1]-----array([160, 110, 90, 50, 40, 20, 8, 7, -30])

a.shape=(3,3)

a-------------------------------array([[110, 20, -30],

[ 40, 50, 160],

[ 7, 8, 90]])

np.sort(a,axis=0) # ColumnWise

array([[ 7, 8, -30],

[ 40, 20, 90],

[110, 50, 160]])

-----------------------------------------------------------

print(a)

array([[110, 20, -30],

[ 40, 50, 160],

[ 7, 8, 90]])

np.sort(a,axis=1) # Row Wise

array([[-30, 20, 110],

[ 40, 50, 160],

[ 7, 8, 90]])

---------------------------

np.delete()

-------------------

Python’s Numpy library provides a method to delete elements from a numpy array based on index position i.e.

numpy.delete(arr, obj, axis=None)

---------------------------------------------------------------

arr : Numpy ndarray from which elements needs to be deleted.

obj : Index position or list of index positions of items to be deleted from numpy ndarray arr.

axis : Axis along which we want to delete.

If 1 then delete columns.

If 0 then delete rows.

Examples:

--------------------

# Create a Numpy array from list of numbers

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

Now let’s delete an element at index position 2 in the above created numpy array,

# Delete element at index position 2

arr = np.delete(arr, 2)

print('Modified Numpy Array by deleting element at index position 2')

print(arr)

Output:----Modified Numpy Array by deleting element at index position 2

[ 4 5 7 8 9 10 11]

----------------------------------------------------

To delete multiple elements from a numpy array by index positions, pass the numpy array and list of index positions to be deleted to np.delete() i.e.

# Create a Numpy array from list of numbers

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

# Delete element at index positions 1,2 and 3

arr = np.delete(arr, [1,2,3])

print('Modified Numpy Array by deleting element at index position 1, 2 &amp; 3')

print(arr)

Output:---------Modified Numpy Array by deleting element at index position 1, 2 &amp; 3

[ 4 8 9 10 11]

----------------------------------------------------------------------------------------

Delete rows & columns from a 2D Numpy Array

----------------------------------------------------------------------------------------

Suppose we have a 2D numpy array i.e.

# Create a 2D numpy array from list of list

arr2D = np.array([[11 ,12, 13, 11],

[21, 22, 23, 24],

[31, 32, 33, 34]])

print(arr2D)

Output:

--------------

[[11 12 13 11]

[21 22 23 24]

[31 32 33 34]]

---------------------------------------------

=>Now let’s see how to delete rows and columns from it based on index positions.

=>Delete a column in 2D Numpy Array by column number

=>To delete a column from a 2D numpy array using np.delete() we need to pass the axis=1

along with numpy array and index of column i.e.

# Delete column at index 1

arr2D = np.delete(arr2D, 1, axis=1)

print('Modified 2D Numpy Array by removing columns at index 1')

print(arr2D)

Output:

-------------------

Modified 2D Numpy Array by removing columns at index 1

[[11 13 11]

[21 23 24]

[31 33 34]]

=>It will delete the column at index position 1 from the above created 2D numpy array.

=>Delete multiple columns in 2D Numpy Array by column number

=>Pass axis=1 and list of column numbers to be deleted along with numpy array to np.delete() i.e.

# Create a 2D numpy array from list of list

arr2D = np.array([[11 ,12, 13, 11],

[21, 22, 23, 24],

[31, 32, 33, 34]])

# Delete column at index 2 &amp; 3

arr2D = np.delete(arr2D, [2,3], axis=1)

print('Modified 2D Numpy Array by removing columns at index 2 &amp; 3')

print(arr2D)

Output:

Modified 2D Numpy Array by removing columns at index 2 &amp; 3

[[11 12]

[21 22]

[31 32]]

It deleted the columns at index positions 2 and 3 from the above created 2D numpy array.

----------------------------------------------------------------

Delete a row in 2D Numpy Array by row number

Our original 2D numpy array arr2D is,

[[11 12 13 11]

[21 22 23 24]

[31 32 33 34]]

To delete a row from a 2D numpy array using np.delete() we need to pass the axis=0 along with numpy array and index of row i.e. row number,

--------------------------------------------------------------------

# Delete row at index 0 i.e. first row

arr2D = np.delete(arr2D, 0, axis=0)

print('Modified 2D Numpy Array by removing rows at index 0')

print(arr2D)

Output:

[[21 22 23 24]

[31 32 33 34]]

It will delete the row at index position 0 from the above created 2D numpy array.

Delete multiple rows in 2D Numpy Array by row number

Our original 2D numpy array arr2D is,

[[11 12 13 11]

[21 22 23 24]

[31 32 33 34]]

Pass axis=0 and list of row numbers to be deleted along with numpy array to np.delete() i.e.

# Delete rows at ro1 1 &amp; 2

arr2D = np.delete(arr2D, [1, 2], axis=0)

print('Modified 2D Numpy Array by removing rows at index 1 &amp; 2')

print(arr2D)

Output:

Modified 2D Numpy Array by removing rows at index 1 &amp; 2

[[11 12 13 11]]

It deleted the row at index position 1 and 2 from the above created 2D numpy array.

-----------------------------------------------------------------------------------------------------------

Delete specific elements in 2D Numpy Array by index position

-----------------------------------------------------------------------------------------------------------

Our original 2D numpy array arr2D is,

[[11 12 13 11]

[21 22 23 24]

[31 32 33 34]]

When we don’t pass axis argument to np.delete() then it’s default value is None, which means 2D numpy array will be flattened for deleting elements at given index position. Let’s use np.delete() to delete element at row number 0 and column 2 from our 2D numpy array,

# Delete element in row 0 and column 2 from 2D numpy array

modArr = np.delete(arr2D, 2)

print('Modified 2D Numpy Array by removing element at row 0 &amp; column 2')

print(modArr)

Output:

Modified 2D Numpy Array by removing element at row 0 &amp; column 2

[11 12 11 21 22 23 24 31 32 33 34]

-----------------------------------------------