**Working with Arrays using numpy**

numpy is a package that contains several classes, functions, variables etc., to deal with scientific calculations in python. numpy is useful to create and also process single and multiple-dimensional arrays. In addition, numpy contains a large library of mathematical functions like linear algebra functions and Fourier transforms.

The arrays which are created using numpy are called n dimensional arrays where n can be any integer. If n=1, it represents a one dimensional array. If n=2, it is a two dimensional array. Similarly if n=3, it is a three dimensional array. The arrays created in numpy can accept only one type of elements. We cannot store different data types into same array.

**# prog-1 : creating an array with characters**

from numpy import \*

arr=array(['a','b','c','d']) # create array

print(arr) #display array

To create a string type array where can store a group of strings. We should use additional attribute ‘dtype=str’ in the array() function as:

arr=array(['Delhi','Hyderabad','Mumbai','Ahamadabad'],dtype=str)

Alternatively, we can omit the ‘dtype=str’ in the above statement and write it as:

arr=array(['Delhi','Hyderabad','Mumbai','Ahmedabad'])

**# Prog-2 :A python program to create a string type array using numpy**

from numpy import \*

# create array

arr=array(['Delhi','Hyderabad','Mumbai','Ahamadabad'],dtype=str)

print(arr) #display array

**# prog-3: A python program to create an array from another array**

from numpy import \*

a=array([1,2,3,4,5]) #original array

b=array(a) #create b from a using array() function

c=a # create c by assigning a to c

#Display the arrays

print("a=",a)

print("b=",b)

print("c=",c)

**Creating Arrays using linspace**

The linspace() function is used to create an array with evenly spaced points between a starting point and ending point.

**Linspace(start,stop,n)**

**‘start’** represents the starting element and ‘stop’ represents the ending element. ‘n’ is an integer that represents the number of parts the elements should be divided. If ‘n’ is omitted, then it is taken as 50.

a=linspace(0,10,5)

In the above statement, we are creating an array ‘a’ with starting element 0 and ending element 10. The range is divided into 5 equal parts and hence the points will be 0, 2.5, 5, 7.5 and 10. These statements are stored into ‘a’.

**# Prog-4:creating an array using linspace()**

from numpy import \*

# divide 0 to 10 into 5 parts and take those points in the array

a=linspace(0,10,5)

print('a=',a)

**Creating Arrays using logspace**

The logspace() function is similar to linspace(). The linspace() function produces the evenly spaced points. Similarly, logspace() produces evenly spaced points on a logarithamically spaced scale.

logspace(start,stop,n)

The logspace() function starts at a value which is ‘start’ to the power of 10 and ends at a value which is ‘stop’ to the power of 10. If ‘n’ is not specified, then the value is taken as 50.

a=logspace(1,4,5)

the function represents values starting from 101 to 104  These values are divided into 5 equal parts and those points are stored into the array ‘a’.

**# Prog-5: Creating an array using logspace()**

from numpy import \*

# divide 1 power 10 to 4 power 10 into 5 equal parts and take those

# points in the array

a=logspace(1,4,5)

# find no.of elements in a

n=len(a)

# repeat from 0 to n-1 times

for i in range(n):

print('%.1f' % a[i],end=' ') # display 1 digit after decimal point

**Creating arrays using arange() function**

The arange() function in numpy is same as range() function in python.

arange(start,stop,stepsize)

arange(10)

arange(5,10)

arange(1,10,3)

arange(10,1,-1)

arange(0,10,1.5)

a=arange(2,11,2)

**Creating arrays using zeros() and ones() functions**

We can use the zeros() function to create an array with all zeros. The ones() function is useful to create an array with all 1’s

zeros(n,datatype)

ones(n,datatype)

Where ‘n’ represents the number of elements. We can eliminate the ‘datatype’ argument. If the we do not specify the ‘datatype’, then the default data type used by numpy is ‘float’.

Zeros(5)

Zeros(5,int)

Ones(5,float)

**# Prog-6: A python program to perform some mathematical operations on a numpy array**

# mathematical operations on arrays

# import all from numpy module

from numpy import \*

# create a numpy array using array() function

arr=array([10,20,30.5,-40])

print("original array : ",arr)

# do arithemtic operations on the elements of the array

print("after adding 5 :",arr+5)

print("after subtracting 5 :",arr-5)

print("after multipling 5 :",arr\*5)

print("after dividing 5 :",arr/5)

print("after modulas with 5 :",arr%5)

# we can use the arrays in expressing also

print("Expression value :",(arr+5)\*\*2-10)

# do some math functions

print("sin values ",sin(arr))

print("cos values ",cos(arr))

print("Tan values ",sin(arr))

print("biggest element ",max(arr))

print("smallest element ",min(arr))

print("sum of all emenets ",sum(arr))

print("Average of all elements",mean(arr))

**Comparing arrays**

We can use the relational operators >,>=,<,<= , == and != to compare the arrays of same size. These operators compare the corresponding elements of the arrays and return another array with Boolean type values. It means the resultant array contains elements which are True or False.

# Prog-7: To know the result of comparing two arrays

from numpy import \*

a=array([1,2,3,0])

b=array([0,2,3,1])

c=a==b

print("result of a==b",c)

c=a>b

print("result of a>b",c)

**# Prog-8:using any() and all() functions**

from numpy import \*

a=array([1,2,3,0])

b=array([0,2,3,1])

c=a>b

print("The result of a>b",c)

print("check if any one element is true :",any(c))

print("check if all elements are true :",all(c))

if(any(a>b)):

print("a contains at least one element greater than those of b")

#Prog-9: A python program to compare the corresponding elements of two arrays # and retrieve the biggest elements

# use where() function

from numpy import \*

a=array([10,20,30,40,50],int)

b=array([1,21,3,40,51],int)

# if a>b then take element from a else from b

c=where(a>b,a,b)

print(c)

# Prog-10: a python program to retrieve non zero elements from an array

from numpy import \*

a=array([1,2,0,-1,0,6],int)

# retrieve indexes of non-zero elements from a

c=nonzero(a)

#display indexes

for i in c:

print(i)

#display the elements

print(a[c])

**Aliasing the arrays**

**# Prog-11: A python program to alias an array and understand the affect of aliasing.**

**# aliasing an array**

**from numpy import \***

**a=arange(1,6) # create a with elements 1 to 5**

**b=a # give another name b to a**

**print("original array:",a)**

**print("alias array :",b)**

**b[0]=99 # modify 0th element of b**

**print("after modification :")**

**print("original array :",a)**

**print("alias array :",b)**

a

4

b

5

3

3

2

1

**viewing and copying arrays**

we can create another array that is same as an existing array. This is done by the view() method. This method creates a copy of an existing array such that the new array will also contain the same elements found in the existing array. The original array and the newly created arrays will share different memory locations. If the newly created array is modified, the original array will also be modified since the elements in both the arrays will be like mirror images.

a

4

5

3

2

1

b

5

4

3

2

1

**#Prog-12:A python program to create a view of an existing array**

**from numpy import \***

**a=arange(1,6) #create a with elements 1 to 5**

**b=a.view() #create a view of a and call it b**

**print("original array",a)**

**print("new array :",b)**

**b[0]=99 #modify 0th element of b**

**print("after modification ")**

**print("original array ",a)**

**print("new array ",b)**

# **Prog-12:A python program to copy an array as another array**

from numpy import \*

a=arange(1,6) #create a with elements 1 to 5

b=a # give another name b to a

print("Original array :",a)

print("alias array:",b)

b[0]=99 # modify 0th element of b

print("After modification")

print("Original array :",a)

print("alias array:",b)

Viewing and Copying arrays

We can create another array that is same as an existing array. This is done by the view() method. This method creates a copy of an existing array such that the new array will also contain the same elements found in the existing array. The original array and the newly created array is modified, the original array will also be modified. Since the elements in both the arrays will be like mirror images.

# Prog-13: Pyhton program to create a view of an existing array

# creating view for an array

from numpy import \*

a=arange(1,6) #create a with elements 1 to 5

b=a.view() # create a view of a and call it b

print("original array",a)

print("New array",b)

b[0]=99 # modify 0th element of b

print("After modification")

print("Original array",a)

print("New array",b)

Viewing is nothing but copying only. It is called ‘**shallow copying’** as the elements in the view when modified will also modify the elements in the original array. So, both the arrays will act as one and the same. Suppose we want both the arrays to be independent and modifying one array should not affect another array. We should go for ‘**deep copying’**. This is done with the help of copy() method. This method makes a computer copy of an existing array and its elements. When the newly created arrays is modified, it will not affect the existing array or vice versa. There will not be any connection between the contents of the two arrays.

# Prog-14:A python program to copy an array as another array

from numpy import \*

a=arange(1,6) #create a with elements 1 to 5

b=a.copy() # create a copy a and call it b

print("Original array:",a)

print("new array:",b)

b[0]=99 # modify 0th element of b

print("After modification:")

print("Original array:",a)

print("new array:",b)

**Slicing and Indexing in numpy Arrays**

Slicing refers to extracting a range of elements from the array.

arrayname[start:stop:stepsize]

A python program to understand slicing operations on arrays

from numpy import \*

#create array a with elements 10 to 15

a=arange(10,16)

print(a)

#retrieve from 1st to one element prior to 6th element in steps of 2

b=a[1:6:2]

print(b)

#retrieve all elements from a

b=a[::]

print(b)

#retrieve from 6-2=4th to one element prior to 2nd element in decreasing step size

b=a[-2:2:-1]

print(b)

#retrieve from 0th to one element prior to 4th element(6-2=4th)

b=a[:-2:]

print(b)

**Indexing refers to the locations of the elements. By specifying the location numbers from 0 onwards till n-1, we can refer to all elements as a[0],a[1],…a[n-1].so, in general we can refer to the elements of an array as a[i] where I can change from 0 to n-1.**

**# Prog-15:** **A python program to retrieve and display elements of numpy array using indexing**

**from numpy import \***

**a=arange(10,16)**

**print(a)**

**# retrieve from 1st to one element prior to 6th element in steps of 2**

**a=a[1:6:2]**

**print(a)**

**# display elements using indexing**

**i=0**

**while(i<len(a)):**

**print(a[i])**

**i+=1**

**Dimensions of Arrays**

The dimension of an array represents the arrangement of elements in the array. If the elements are arranged horizontally, it is called a row and if the elements are arranged vertically, then it is called a column. When an array contains only 1 row or only 1 column of elements, it is called single dimensional array or one dimensional array.

**# aray with 1 row**

**arr1=array([1,2,3,4,5])**

**print(arr1) #display [1 2 3 4 5]**

**# array with 1 column**

**arr2=array([10,**

**20,**

**30,**

**40])**

**print(arr2)**

**#create a 2D array with 2 rows and 3 cols in each row**

**arr2=array([[1,2,3],**

**[4,5,6]])**

**print(arr2)**

**Attributes of an array**

**Numpy’s array class is called ndarray. It is also known as by alias name array. Let’s remember that there is another class ‘array’ in python that is different from numpy’s ‘array’ class. This class contains the following important attributes ( or variables).**

**The ndim Attribute**

**The ‘ndim’ attribute represents the number of dimensions or axes of the array. The number of dimensions is also referred to as ‘rank’. For a single dimension array, it is 1 and for a two dimensional array, it is 2.**

**arr1=array([1,2,3,4,5]) # 1D array**

**print(arr1.ndim)**

**The shape Attribute**

**The ‘shape’ attribute gives the shape of an array. The shape is a tuple listing the number of elements along each dimension. A dimension is called an axis. For a 1D array, shape elements in the row. For 2D array, it specifies the number of rows and columns in each row. We can also change the shape using ‘shape’ attribute.**

**Ex-1 :arr1=array([1,2,3,4,5])**

**print(arr1.shape)**

**o/p**

**(5,)**

**Wx-2:arr2=array([[1,2,3],[4,5,6]])**

**print(arr2.shape)**

**o/p (2,3) #2 rows and 3 cols**

**Ex:-3 : arr2.shape=(3,2) #change shape of arr2 to 3 rows and 2 cols**

**print(arr2)**

**o/p: [ [1 2]**

**[ 3 4]**

**[ 5 6] ]**

**The size attribute:**

**The size attribute gives the total number of elements in the array. For example, consider the following :**

**Ex-1: arr1=array([1,2,3,4,5])**

**print(arr1.size)**

**o/p 5**

**Ex-2:**

**arr2=array([[1,2,3],[4,5,6]])**

**print(arr2.size)**

**o/p : 2**

**The itemsize attribute**

**The ‘itemsize’ attribute gives the memory size of the array elements in bytes. As we know 1 byte is equal to 8 bits.**

**Ex-1: arr1=array([1,2,3,4,5])**

**print(arr1.itemsize)**

**o/p : 4(size of each element)**

**Ex-2:**

**arr2=array([1.1,2.1,3.5,4,5.0])**

**o/p: 8**

**The dtype Attribute**

**The ‘dtype’ attribute gives the data type of the elements in the array.**

**arr1=array([1,2,3,4,5]) # integer type array**

**print(arr1.dtype)**

**o/p : int32**

**Ex-2: arr2=array([1.1,2.1,3.5,4,5.0])**

**print(arr2.dtype)**

**o/p: float64**

**The nbytes Attribute**

**The ‘nbytes’ attribute gives the total number of bytes occupied by an array. The total number of bytes=sizeof the array \* item size of each element in the array.**

**arr2=array([[1,2,3],[4,5,6]])**

**print(arr2.nbytes)**

**o/p**

**The reshape() method**

**The ‘reshape() method is useful to change the shape of an array. The new array should have the same number of elements as in the original array.**

**Ex:-1 : arr1=arange(10) # 1D array with 10 elements**

**print(arr1)**

**o/p : [0 1 2 3 4 5 6 7 8 9]**

**print(arr1.reshape(2,5))**

**o/p**

**[[0 1 2 3 4]**

**[5 6 7 8 9]]**

**Ex:-2**

**arr1=arr1.reshape(5,2) #change the shape as 5 rows, 2 cols**

**print(arr1)**

**o/p**

**[[ 0 1]**

**[2 3]**

**[4 5]**

**[6 7]**

**[8 9] ]**

**The flatten() Method**

**the flatten() method is useful to return a copy of the array collapsed into one dimension.**

Convert from 2D to 1D or 3D to 1d(nD to 1D)

**Ex-1:**

**arr1=array([[1,2,3],[4,5,6]])**

**print(arr1)**

**o/p**

**[[1 2 3]**

**[4 5 6 ]]**

**By using the flatten() method, we can convert this array into 1D array as:**

**arr1=arr1.flatten()**

**print(arr1)**

**o/p: [ 1 2 3 4 5 6]**

**Working with Multi-dimensional Arrays**

**The 2D arrays, 3D arrays etc. are called multi dimensional arrays. A 2D array contains more than 1 row and 1 column and it can be treated as a combination of several 1D arrays. A 2D array is also considered as a matrix. For example, a 2D array with ‘m rows and ‘n’ columns is called m x n matix. As we know in Mathematics, a matrix contains elements arranged in several rows and columns. Hence, we can take a matix as a 2D array and vice versa.**

**We can create mult-dimensional arrays in the following ways.**

1. **Using array() function**
2. **Using ones() and zeros() functions**
3. **Using eye() function**
4. **Using reshape() function**

**The array() function**

**The numpy’s array() function can be used to create a multidimensional array. Usually, we pass lists of elements to this function. If we pass one list of elements to this function, then it will create a 1D array. If we pass two lists of elements, then this function creates a 2D array.**

**a=array([1,2,3,4]) #this creates a 1D array with 1 row**

**a=array([[1,2,3,4],[5,6,7,8]]) #creates 2D array with 2 rows and 4 cols**

**The ones() and zeros() functions**

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

**ones((r,c),dtype)**

**ex: ones((3,4),float)**

**b=zeros((3,4),int)**

**The eye() function**

**The eye() function creates a 2D array and fills the elements in the diagonal with 1’s.**

**eye(n,dtype=datatype)**

**This will create an array with ‘n’ rows and ‘n’ columns. The default datatype is ‘float’. For example eye(3) will create an array with ‘n’ rows and ‘n’ columns. The default datatype is ‘float’. For example, eye(3) will create a 3x3 array and fills the diagonal elements with 1s .**

**The reshape() function**

**The reshape() function is useful to convert a 1D array into a multidimensional(2D or 3D) array.**

**reshape(arrayname,(n,r,c))**

**Here, ‘arrayname’ represents the name of the array whose elements to be converted. ‘n’ indicates the number of arrays in the resultant array.’r’,’c’ indicates the number of rows and columns.**

**Ex:**

**a=array([1,2,3,4,5,6])**

**To convert ‘a’ into a 2D array using reshape() function**

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

**o/p**

**[[1 2 3]**

**[4 5 6] ]**

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

**It is possible to use the reshape() function to convert a 1D array into a 3D array.**

**A=arrange(12)**

**B=reshape(a,(2,3,2))**

**#Prog-16: To retrieve the elements from a 2D array and display them using loops**

**from numpy import \***

**#create a 2D array with 3 rows and 3 cols**

**a=[[1,2,3],[4,5,6],[7,8,9]]**

**#display only rows**

**for i in range(len(a)):**

**print(a[i])**

**#display element by element**

**for i in range(len(a)):**

**for j in range(len(a[i])):**

**print(a[i][j],end=' ')**

**print()**

**#Prog-17: To retrieve the elements from a 3D array using indexing**

**from numpy import \***

**#create a 3D array with size 2x2x3**

**a=[[[1,2,3],**

**[4,5,6]],**

**[[7,8,9],**

**[10,11,12]]]**

**#display element by element**

**for i in range(len(a)):**

**for j in range(len(a[i])):**

**for k in range(len(a[i][j])):**

**print(a[i][j][k],end=' ')**

**print()**

**print()**

**Matrices in numpy**

**In Mathematics, matrix represents a rectangular array of elements arranged in rows and columns. It means elements are available in a matrix in the form of several rows and columns. If a matrix has only 1 row , it is called a ‘row matrix’. If a matrix has only 1 column, then it is called a ‘column matrix’ i.e row matrix and column matrix are nothing but 1D arrays.**

**Getting Diagonal Elements of a Matrix**

**To retrieve the diagonal elements of a matrix, we can use diagonal() function as:**

**The diagonal() function returns a 1D array that contains diagonal elements of the original matrix.**

**a=matrix(‘1 2 3;4 5 6; 7 8 9’) # to create a 3x3 matrix**

**print(a)**

**d=diagonal(a)**

**print(d)**

Finding Maximum and Minimum Elements

To know the maximum element, we can use max() method and to know the minimum element. We can use min() methods. These methods should be called using the matrix name.

big=a.max()

print(big)

small=a.min()

print(small)

**Finding sum and average of lements**

**a.sum()**

**a.mean()**

**Products of elements**

**It is possible to know the products of elements in a matrix. For this purpose, numpy provides prod() method. It returns a matrix that contains the products of elements in each column of the original matrix. prod(1) returns a matrix that contains products of elements in each row. These methods should be called using the matrix name.**

**Ex:**

**m=reshape(matrix(arange(12)),(3,4))**

**print(m)**

**b=m.prod(1)**

**print(b)**

**Sorting the matrix**

**Numpy provides sort() function that stores the matrix elements into ascending order.**

**Sort(matrixname,axis)**

**If we use , axis=1, it sorts the elements in each rows into ascending order. If we use, axis=0, then it sorts the elements in each column into ascending order. The default value of axis is 1. It means, if we do not mention ‘axis’, then it is value will be taken as 1.**

**m=matrix([[5,4,1],[2,7,0]])**

**print(m)**

**To sort the elements in the rows, we can use sort() function as:**

**a=sort(m)**

**print(a)**

**b=sort(m,axis=0)**

**print(b)**

**Transpose of a matrix**

**m=matrix(‘1 2 3 ; 4 5 6; 7 8 9’) #create a matrix of 3 x 3 size**

**print(m)**

**t=m.transpose() #find the transpose matrix**

**print(t)**

**also, consider the following**

**t1=m.getT() #find transpose using get()**

**print(t1)**

**# prog-18: A python program to accept a matrix from the keyboard and display its transpose matrix**

**from numpy import \***

**#accept number of rows and cols into r,c**

**r,c=[int(a) for a in input("Enter rows ,cols :").split()]**

**#accept matrix elements as a string into str**

**str=input("Enter matrix elements ")**

**# convert the string into a matrix with size rxc**

**x=reshape(matrix(str),(r,c))**

**print("the original matrix ")**

**print(x)**

**print("The tanspose matrix")**

**y=x.transpose()**

**print(y)**

**Matrix addition and multiplication**

**We can use arithmetic operations like +,- and /to perform addition, subtraction and division operations on 2 matrices.**

**a=matrix(‘1 2 3; 4 5 6’) #create a matrix a with 2x3 matrix**

**b=matrix(‘2 2 2; 1 -1 2’)**

**print(a)**

**print(b)**

**c=a+b # add a and b**

**print(c)**

**d=a/b**

**print(d)**

**#prog-19: To accept two matrices and find their product**

**import sys**

**from numpy import \***

**#accept number of rows and cols of first matrix into r1,c1**

**r1,c1=[int(a) for a in input("first matrix rows , cols : ").split()]**

**#accept number of rows and cols of first matrix into r2,c2**

**r2,c2=[int(b) for b in input("second matrix rows , cols : ").split()]**

**#Test the condition if c1!=r2, then multiplication is not possible**

**if c1!=r2:**

**print("Multiplication is not possible")**

**sys.exit() # terminate the program**

**#accept the first matrix elements as a string into str1**

**str1=input("Enter first matrix elements ")**

**#convert str1 into a matrix with size r1xc1**

**x=reshape(matrix(str1),(r1,c1))**

**#accept the second matrix elements as a string into str2**

**str2=input("Enter second matrix elements ")**

**#convert str2 into a matrix with size r2xc2**

**y=reshape(matrix(str2),(r2,c2))**

**z=x\*y**

**print(z)**

Random numbers

A random number is a number that cannot be guessed by any one. When a random number is generated, we do not know which number we are going to get. numpy has a sub module called random that is equipped with the rand() function that is useful to create random numbers. To call this function, we should use: random.rand(). Since it belongs to the ‘random ‘ sub module.  
 When we simply call the rand() function, it will generate a random number b/n 0.0 and 1.0 and every time we may have a different number.

a=random.rand(5)

print(a)

This will create a 1D array ‘a’ with 5 elements which are random numbers b/n 0.0 and 1.0

We can also create 2D array of random numbers by passing the size of the 2D array to the random() function as.

b=random.rand(2,3)

print(b)

**Advanced Indexing:**

It is possible to make a selection from ndarray that is a non-tuple sequence, ndarray object of integer or Boolean data type, or a tuple with at least one item being a sequence object. Advanced indexing always returns a copy of the data. As against this, the slicing only presents a view.

There are two types of advanced indexing.

1. Integer 2. Boolean

Integer Indexing

This mechanism helps in selecting any arbitrary item in an array based on its Ndimensional index. Each integer array represents the number of indexes into that dimension. When the index consists of as many integer arrays as the dimensions of the target array,it becomes straight forward.

In the following example, one element of specified column for each row of ndarray object is selected. Hence, the row index contains all row numbers, and the column index specifies the element to be selected.

Example-1:

import numpy as np

x=np.array([[1,2],[3,4],[5,6]])

y=x[[0,1,2],[0,1,0]]

print(y)

The selection includes elements at (0,0),(1,1) and (2,0) from the first array.

In the following example, elements placed at corners of 4x3 array are selected. The row indices of selection are [0,0] and [3,3] where as the column indices are [0,2] and [0,2]

Example-2:

import numpy as np

x=np.array([[0,1,2],[3,4,5,],[6,7,8],[9,10,11]])

print("our array is")

print(x)

print()

rows=np.array([[0,0],[3,3]])

cols=np.array([[0,2],[0,2]])

y=x[rows,cols]

print("the corner elements of this array are")

print(y)

The resultant selection is an ndarray object containing corner elements.

Advanced and basic indexing can be combined by using one slice(:) or ellipsis(…) with an index array. The following example uses slice for row and advanced index for column. The result is the same when slice is used for both. But advanced index results in copy and may have different memory layout.

Example-3:

import numpy as np

x=np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])

print("our array is")

print(x)

print()

#slicing

z=x[1:4,1:3]

print("after slicing, our array becomes")

print(z)

print()

#using advanced index for column

y=x[1:4,[1,2]]

print(y)

**Boolean Array Indexing**

This type of advanced indexing is used when the resultant object is meant to be the result of Boolean operations, such as comparison operators.

In the following example, items greater than 5 are returned as a result of Boolean indexing

Example-1:

import numpy as np

x=np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])

print ("our array is")

print(x)

print()

print("The items > 5 are:")

print(x[x>5])

example :2

In this example, NaN(Not a Number) elements are omitted by using ~(complement operator)

import numpy as np

a=np.array([np.nan,1,2,np.nan,3,4,5])

print(a[~np.isnan(a)])

Example-3:

The following example shows that how filter out the non-complex elements from an array.

import numpy as np

a=np.array([1,2+6j,5,3.5+5j])

print(a[np.iscomplex(a)])

**Iterating Over Array**

Numpypackage contains an iterator object **numpy.nditer.** It is an efficient multidimensional iterator object using which it is possible to iterate over an array. Each element of an array is visited using python’s standard iterator interface.

To create a 3x4 array using arrange() function and iterate over it using **nditer**

Wxample-1

import numpy as np

a=np.arange(0,60,5)

a=a.reshape(3,4)

print("original array is")

print(a)

print()

print("modified array is")

for x in np.nditer(a):

print(x,end=' ')

example-2:

The order of iteration is chosen to match the memory layout of an array without considering a particular ordering. This can be seen by iterating over the transpose of the above array.

import numpy as np

a=np.arange(0,60,5)

a=a.reshape(3,4)

print("the original array is")

print(a)

print("Transpose of the original array is ")

b=a.T

print(b)

print()

print("modified array is")

for x in np.nditer(b):

print(x,end=' ')

**Iteration Order**

If the same elements are stored using F-style order, the iterator chooses the more efficient way of iterating over an array.

import numpy as np

a=np.arange(0,60,5)

a=a.reshape(3,4)

print("the original array is")

print(a)

print()

print("Transpose of the original array is")

b=a.T

print(b)

print("sorted in C-style order")

c=b.copy(order='C')

print(c)

for x in np.nditer(c):

print(x,end=' ')

**Modifying Array Values**

The **nditer** object has another optional parameter called **op\_flags.** Its default value is read-only, but can be set to read-write or write-only mode. This will enable modifying array elements using this iterator.

import numpy as np

a=np.arange(0,60,5)

a=a.reshape(3,4)

print("the original array is")

print(a)

print()

for x in np.nditer(a,op\_flags=['readwrite']):

x[...]=2\*x

print('modified array is')

print(a)

**External loop**

The nditer class constructor has a ‘flags’ parameter, which can take the following values –

c\_index => C-order index can be tracked.

f\_index => Fortran\_order index is tracked

multi\_index => Type of indexes with one per iteration can be tracked.

external\_loop => causes values given to be one-dimensional arrays with

multiple values instead of zero dimensional array.

**Example**

In the following example, one-dimensional arrays corresponding to each column is traversed by the iterator.

import numpy as np

a=np.arange(0,60,5)

a=a.reshape(3,4)

print("the original array is")

print(a)

print()

print('modified array is')

for x in np.nditer(a,flags=['external\_loop'],order='F'):

print(x,end=' ')

**Brodcasting Iteration**

If two arrays are broadcastable, a combined nditer object is able to iterate

############################################

End

#############################################

**What’s numpy**

**numpy is a python extension to add support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions.**

**Ex: 1**

**import sys**

**x=20**

**sys.getsizeof(x)**

**o/p : 24**

**Ex: 2**

**from array import array  
 a=array(‘H’,[2,3,4,5])**

**a[2:4]**

**o/p : array(‘H’,[4,5])**

**Ex:3**

**Import numpy as np**

**a=np.array([2,3,4,5,7])**

**print(a.shape) # (5,)**

**print(a.dtype) # dtype(int64)**

**print(a.ndim) # 1**

**print(a.size) # 5**

**print(a.bytes) # 40**

**print(np.array(range(10)) # array([0,1,2,3,4,5,6,7,8,9])**