1. **What makes Numpy.shape() different from Numpy.size()**

Numpy.shape() – It will give you the size of n-dim array like if we have 2x4(mxn) matrix then it will give output as tuple: (2,4)

Numpy.size() – It will gives us the count of elements present in the array.

from numpy import \*  
arr=array([[1,2,3,4],[2,3,4,5,]])  
print("arr shape:",arr.shape)  
print("arr size:",arr.size)

o/p:

arr shape: (2, 4)

arr size: 8

Numpy.shape() – it is related to size of the dimensions of an N-dim array

Numpy.size() – It is related to the number of elements present in the array.

1. **In NumPy, describe the idea of broadcasting**

When we want to perform arithmetic operations on arrays which are having different dimensions or different size or different shape then only will use broadcasting. And broadcasting will not be applicable in all cases. It has some rules, if the rules are satisfied then only broadcasting will be performed internally while performing arithmetic operations

Note: If both the arrays have same dimension, same shape and same size then broadcasting is not required.

Rules:

1. If the two arrays of different dimensions, we should make both the arrays to equal dimensions.

Have to add 1’s in the shape of lesser dimension array on the left side.

A:(4,3)

B:(3,)

Broadcasting will make these two arrays as 2-D arrays :

A:(4,3)

B:(1,3)

1. If the rows of A doesn’t match with the rows of B, similarly if the columns of A doesn’t match with the columns of B, then the array with size equal to 1 in that dimension is expanded to match other size of the same dimension.

Before:

A:(4,3)

B:(1,3)

After :

A:(4,3)

B:(4,3)

Now both the arrays are of same dimension and same shape, then the arithmetic operation will be performed normally.

**Before broadcasting, If we have matrix like below**:

A:(4,3)

B:(2,3) // rule 2 fails then will get ValueError and broadcasting won’t be performed

* The result is always of higher dimension of input arrays.

Eg : inputs:3-D, 1-D

Output : 3-D

**Note** **: In any dimension if the sizes are not matched and neither equal to 1, then we will get error as below mentioned:**

**Eg :1**

import numpy as np  
a=np.array([10,20,30,40])  
b=np.array([1,2,3])  
print("shape of a:",a.shape)  
print("shape of b:",b.shape)  
c=a\*b

o/p:

Traceback (most recent call last):

File "C:\Users\Kranthi Ranadheer\PycharmProjects\MyProj\numpy\_broadcasting.py", line 6, in <module>

c=a\*b

ValueError: operands could not be broadcast together with shapes (4,) (3,)

shape of a: (4,)

shape of b: (3,)

**Eg:2**

A=np.array([10,20,30])

B=np.array([40])

Rule1: both are 1 D arrays

Rule2 :

Before : shape of a:(3,) and b’s shape (1,)

After : a’s shape(3,) and b’s shape(3,)

Now: A=np.array([10,20,30])

B=np.array([40,40,40])

import numpy as np  
a=np.array([10,20,30])  
b=np.array([40])  
print("shape of a:",a.shape)  
print("shape of b:",b.shape)  
c=a\*b  
print("c:",c)

o/p:

shape of a: (3,)

shape of b: (1,)

c: [ 400 800 1200]

**Eg 3:**

import numpy as np  
a=np.array([[10,20],[30,40],[50,60]])  
b=np.array([10,40])  
print("shape of a:",a.shape)  
print("shape of b:",b.shape)  
c=a\*b  
print("c:",c)

o/p:

shape of a: (3, 2)

shape of b: (2,)

c: [[ 100 800]

[ 300 1600]

[ 500 2400]]

1. **What makes python better than other libraries for numerical computation?**

Python has a numpy package, by using this we can deal with multi dimensional arrays easily and it has another package which is pandas : it can be used for data structures. And python is easy to learn.

1. **How does NumPy deal with files?**

By using numpy package we can read and write the files.

If we want to load txt file :

with no missing values: numpy.loadtxt

with missing values : numpy.genfromtxt

and it also return a masked array masking out missing values (if usemask=True)

np.gemfromtxt(“test.txt”, delimiter=”,”, usemask=True)

🡪 numpy.save()

The **numpy.save()** file stores the input array in a disk file with **npy** extension.

import numpy as np  
a = np.array([1,2,3,4,5])  
np.save('kee\_file',a)

To reconstruct array from kee\_file.npy, use load() function

import numpy as np  
k = np.load('kee\_file.npy')  
print(k)

savetxt(),loadtxt() – the storage and retrieval of array data in simple text format will use this savetxt() and loadtxt()

1. **Mention the importance of NumPy.empty() ?**

**numpy.empty(shape, dtype = float, order = ‘C’) :**Return a new array of given shape and type, with random values.

This function doesn’t set values to zero and it is quite faster than the numpy.zeros().This function requires the user to set all the values in the array manually.

**dtype: data-type(optional)**

This parameter defines the data type, which is desired for the output array.

**order: {'C', 'F'}(optional)**

This parameter defines the order in which the multi-dimensional array is going to be stored either in **row-major** or **column-major**. By default, the order parameter is set to **'C'**.

import numpy as np  
x = np.empty([3, 2])  
print("x:",x)

o/p:

x: [[1.78022206e-306 1.78019082e-306]

[1.02360935e-306 1.78020848e-306]

[2.78150360e-307 6.01346930e-154]]