1.what makes numpy.shape()different from numpy.size?

Ans:I noticed that some numpy operations take an argument called shape, such as np.zeros, whereas some others take an argument called size, such as np.random.randint. To me, those arguments have the same function and the fact that they have different names is a bit confusing. Actually, size seems a bit off since it really specifies the .shape of the output.

Is there a reason for having different names, do they convey a different meaning even though they both end up being equal to the .shape of the output

I don't think so. Its community developed. Its not always easy to ensure consistency. I agree with you. Shape (in the numpy context) seems to me the better option for an argument name. The actual relation between the two is size = np.prod(shape) so the distinction should indeed be a bit more obvious in the arguments

randint uses the size parameter name, but uses shape in the explanation. This is not to be confused with a real size attribute.

Shape relates to the size of the dimensions of an N-dimensional array.

Size regarding arrays, relates to the amount (or count) of elements that are contained in the array (or sometimes, at the top dimension of the array - when used as length).

For example, let a be a matrix

1 2 3 4 5 6 7 8 9 10 11 12

the shape of a is (3, 4), the size of a is 12 and the size of a[1] is 4

Because you are working with a numpy array, which was seen as a C array, size refers to how big your array will be. Moreover, if you can pass np.zeros(10) or np.zeros((10)). While the difference is subtle, size passed this way will create you a 1D array. You can give size=(n1, n2, ..., nn) which will create an nD array.

However, because python users want multi-dimensional arrays, array.reshape allows you to get from 1D to an nD array. So, when you call shape, you get the N dimension shape of the array, so you can see exactly how your array looks like.

In essence, size is equal to the product of the elements of shape.

EDIT: The difference in name can be attributed to 2 parts: firstly, you can initialise your array with a size. However, you do not know the shape of it. So size is only for total number of elements. Secondly, how numpy was developed, different people worked on different parts of the code, giving different names to roughly the same element, depending on their personal vision for the code.

2. In numpy, describe the idea of board casting?

Ans:The term broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations. Arithmetic operations on arrays are usually done on corresponding elements. If two arrays are of exactly the same shape, then these operations are smoothly performed.

Example 1

Live Demo

import numpy as np a = np.array([1,2,3,4]) b = np.array([10,20,30,40]) c = a \* b print c

Its output is as follows −

[10 40 90 160]

If the dimensions of two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability. The smaller array is broadcast to the size of the larger array so that they have compatible shapes.

Broadcasting is possible if the following rules are satisfied −

Array with smaller ndim than the other is prepended with '1' in its shape.

Size in each dimension of the output shape is maximum of the input sizes in that dimension.

An input can be used in calculation, if its size in a particular dimension matches the output size or its value is exactly 1.

If an input has a dimension size of 1, the first data entry in that dimension is used for all calculations along that dimension.

A set of arrays is said to be broadcastable if the above rules produce a valid result and one of the following is true −

Arrays have exactly the same shape.

Arrays have the same number of dimensions and the length of each dimension is either a common length or 1.

Array having too few dimensions can have its shape prepended with a dimension of length 1, so that the above stated property is true.

The following program shows an example of broadcasting.

Example 2

Live Demo

import numpy as np a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20.0],[30.0,30.0,30.0]]) b = np.array([1.0,2.0,3.0]) print 'First array:' print a print '\n' print 'Second array:' print b print '\n' print 'First Array + Second Array' print a + b

The output of this program would be as follows −

First array: [[ 0. 0. 0.] [ 10. 10. 10.] [ 20. 20. 20.] [ 30. 30. 30.]] Second array: [ 1. 2. 3.] First Array + Second Array [[ 1. 2. 3.] [ 11. 12. 13.] [ 21. 22. 23.] [ 31. 32. 33.]]

3.what makes python better than other libraries for numerical computation?

Ans:Python is the most widely used programming language today. When it comes to solving data science tasks and challenges, Python never ceases to surprise its users. Most data scientists are already leveraging the power of Python programming every day. Python is an easy-to-learn, easy-to-debug, widely used, object-oriented, open-source, high-performance language, and there are many more benefits to Python programming. Python has been built with extraordinary Python libraries for data science that are used by programmers every day in solving problems. Here’s the top 10 Python libraries for data science:

Top 10 Python Libraries for Data Science

TensorFlow

NumPy

SciPy

Pandas

Matplotlib

Keras

SciKit-Learn

PyTorch

Scrapy

BeautifulSoup

1.TensorFlow

The first in the list of python libraries for data science is TensorFlow. TensorFlow is a library for high-performance numerical computations with around 35,000 comments and a vibrant community of around 1,500 contributors. It’s used across various scientific fields. TensorFlow is basically a framework for defining and running computations that involve tensors, which are partially defined computational objects that eventually produce a value

SciPy

SciPy (Scientific Python) is another free and open-source Python library for data science that is extensively used for high-level computations. SciPy has around 19,000 comments on GitHub and an active community of about 600 contributors. It’s extensively used for scientific and technical computations, because it extends NumPy and provides many user-friendly and efficient routines for scientific calculations

. NumPy

NumPy (Numerical Python) is the fundamental package for numerical computation in Python; it contains a powerful N-dimensional array object. It has around 18,000 comments on GitHub and an active community of 700 contributors. It’s a general-purpose array-processing package that provides high-performance multidimensional objects called arrays and tools for working with them. NumPy also addresses the slowness problem partly by providing these multidimensional arrays as well as providing functions and operators that operate efficiently on these arrays

4.How does NumPy deal with files ?

Ans:

The ndarray objects can be saved to and loaded from the disk files. The IO functions available are −

load() and save() functions handle /numPy binary files (with npy extension)

loadtxt() and savetxt() functions handle normal text files

NumPy introduces a simple file format for ndarray objects. This .npy file stores data, shape, dtype and other information required to reconstruct the ndarray in a disk file such that the array is correctly retrieved even if the file is on another machine with different architecture.

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('outfile',a)

To reconstruct array from outfile.npy, use load() function.

import numpy as np b = np.load('outfile.npy') print b

It will produce the following output −

array([1, 2, 3, 4, 5])

The save() and load() functions accept an additional Boolean parameter allow\_pickles. A pickle in Python is used to serialize and de-serialize objects before saving to or reading from a disk file.

savetxt()

The storage and retrieval of array data in simple text file format is done with savetxt() and loadtxt() functions.

Example

import numpy as np a = np.array([1,2,3,4,5]) np.savetxt('out.txt',a) b = np.loadtxt('out.txt') print b

It will produce the following output −

[ 1. 2. 3. 4. 5.]

The savetxt() and loadtxt() functions accept additional optional parameters such as header, footer, and delimiter

5. Mention the important of Numpy.empty()?

Ans:The numpy module of Python provides a function called numpy. empty(). This function is used to create an array without initializing the entries of given shape and type

Just like numpy.zeros(), the numpy.empty() function doesn't set the array 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 and should be used with caution.

Syntax

numpy.empty(shape, dtype=float, order='C')

Parameters:

shape: int or tuple of ints

This parameter defines the shape of the empty array, such as (3, 2) or (3, 3).

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'.

Returns:

This function returns the array of uninitialized data that have the shape, dtype, and order defined in the function.

Example 1:

import numpy as np

x = np.empty([3, 2])

x

Output:

array([[7.56544226e-316, 2.07617768e-316], [2.02322570e-316, 1.93432036e-316], [1.93431918e-316, 1.93431799e-316