MIT World Peace University

Python Programming
Second Year B. Tech, Semester 4

Learning Basics of the *Numpy library*

Assignment No. 8

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 25, 2023

Contents

1 Aim

Write a python code to use a Numpy module create an array and check the following:

1. Type of array.
2. Axes of array.
3. Shape of array.
4. Type of elements in array.

2 Objectives

1. To learn and implement functions of the Numpy module.

3 Theory

3.1 Numpy Library

The Numpy library is a Python library that is used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

It also has a large collection of high-level mathematical functions to operate on these arrays. It is also used for working in domain of machine learning, deep learning, and artificial intelligence.

The reason it was created is because the Python language does not have built-in support for Arrays, and they are very useful in data science and scientific computing.

It has since grown in popularity and is now used in a wide range of fields including finance, engineering, and medicine.

3.2 Numpy Functions

The numpy Library supports a large number of functions that can be used to work with arrays. They make our job easier and help us to write code faster.

3.2.1 arrange()

Definition:
The arrange() function returns evenly spaced values within a given interval. The format of the arrange() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x)

> [0 1 2 3 4 5 6 7 8 9]
```

3.2.2 shape()

Definition:
The shape() function returns the shape of an array. The format of the shape() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x.shape)

> (10,)
```

3.2.3 reshape()

Definition:
The reshape() function gives a new shape to an array without changing its data. The format of the reshape() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x.reshape(2, 5))

> [[0 1 2 3 4]
> [5 6 7 8 9]]
```

3.2.4 size()

Definition:
The size() function returns the total number of elements of the array. The format of the size() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x.size)

> 10
```

3.2.5 ndim()

Definition:
The ndim() function returns the number of array dimensions. The format of the ndim() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x.ndim)
```

```
> 1
```

3.2.6 eyes()

Definition:
The eyes() function returns a 2-D array with ones on the diagonal and zeros elsewhere. The format of the eyes() function is:

Example and Use:

```
import numpy as np
x = np.arrange(10)
print(x.reshape(2, 5))

> [[0 1 2 3 4]
> [5 6 7 8 9]]
```

3.2.7 ones()

Definition:
The ones() function returns a new array of given shape and type, filled with ones. The format of the ones() function is:

Example and Use:

```
import numpy as np
x = np.ones(10)
print(x)

> [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

3.2.8 zeros()

Definition:
The zeros() function returns a new array of given shape and type, filled with zeros. The format of the zeros() function is:

Example and Use:

```
import numpy as np
x = np.zeros(10)
print(x)

> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

3.2.9 dtype()

Definition:
The dtype() function returns the data type of the array

Example and Use:

```
import numpy as np
x = np.zeros(10)
print(x.dtype)

> float64
```

4 Input and Output

4.1 Input

Creating array using numpy library

4.2 Output

ndarray is created and shape and data type are displayed.

5 Requirements

1. Python 3.7 or above
2. Numpy

6 Code

7 Threading

[1]:
```
import threading

import time

def calc_square(numbers):
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.2)
        print('square:', n*n)

def cal_cube(numbers):
    print("calculate cube of numbers")
    for n in numbers:
        time.sleep(0.2)
        print('cube:', n*n*n)

arr = [2,3,8,9]
```

7.1 Let us try to run it without threads

[4]:
```
%%time
calc_square(arr)
cal_cube(arr)
```

```
calculate square numbers
square: 4
square: 9
square: 64
square: 81
calculate cube of numbers
cube: 8
cube: 27
cube: 512
cube: 729
CPU times: user 11.3 ms, sys: 222 µs, total: 11.5 ms
Wall time: 1.6 s
```

7.2 Now let us try to use threads

[15]:
```
thread_1 = threading.Thread(target=calc_square, args=(arr,))
thread_2 = threading.Thread(target=cal_cube, args=(arr,))
```

[16]:
```
%%time
thread_1.start()
thread_2.start()
thread_1.join()
print("Done with thread 1")
thread_2.join()
print("Done with thread 2")
```

```
calculate square numbers
calculate cube of numbers
square: 4
cube: 8
square: 9
cube: 27
square: 64
cube: 512
square: 81
Done with thread 1
cube: 729
Done with thread 2
CPU times: user 1.99 ms, sys: 8.22 ms, total: 10.2 ms
Wall time: 804 ms
```

[ ]:
```

```

8 Conclusion

The Numpy was studied and understood. The functions of the Numpy library were also studied and implemented.

9 FAQ

1. **What is the difference between Numpy array and List?**

   Numpy array:

   - Numpy arrays are faster than lists.
   - Numpy arrays are more compact than lists.
   - Numpy arrays are more convenient than lists.

   List:

   - Lists are slower than Numpy arrays.
   - Lists are more flexible than Numpy arrays.
   - Lists are more convenient than Numpy arrays.

2. **Explain type and dtype with example.**

   type() is a built-in function in Python that returns the type of the object passed to it. It is used to check the type of an object. dtype(), however, is an attribute of the ndarray object in Numpy. It returns the data type of the array. The difference between the two is that type() is a function and dtype() is an attribute.

   (a) **Type:**

   ```
   import numpy as np
   x = np.zeros(10)
   print(type(x))

   > <class 'numpy.ndarray'>
   ```

   (b) **Dtype:**

   ```
   import numpy as np
   x = np.zeros(10)
   print(x.dtype)

   > float64
   ```

3. **Explain axis in numpy arrays.**

   Axis is a dimension of an array. In a 2-D array, there are two axes, the first running vertically downwards across rows (axis 0), and the second running horizontally across columns (axis 1). In a 3-D array, there are three axes, the first running downwards along the depth of the array,

the second running horizontally across rows (axis 1), and the third running vertically across columns (axis 2).

Example:

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print(x.sum(axis=0))

> [5 7 9]
```