

Manipulating NumPy Arrays

Abdulaziz M. Alqumayzi

(DS-540) – Intro to python for computer science and data science: Learning to program with

AI, Big Data and The Cloud

Colorado State University – Global Campus

Dr. Ernest Bonat

October 2, 2020

Manipulating NumPy Arrays

Introduction

In this critical thinking activity, we manipulate Python-language NumPy arrays. One of the several open-source libraries that the Anaconda Python distribution installs is the NumPy (Numerical Python) library. NumPy is based on over 450 Python libraries. Many common data science libraries are based on or rely on NumPy, such as Pandas, SciPy (Scientific Python), and Keras (for deep learning). The purpose of this critical thinking is to write a program for a 1-D NumPy array to provide the following tasks in the debate:

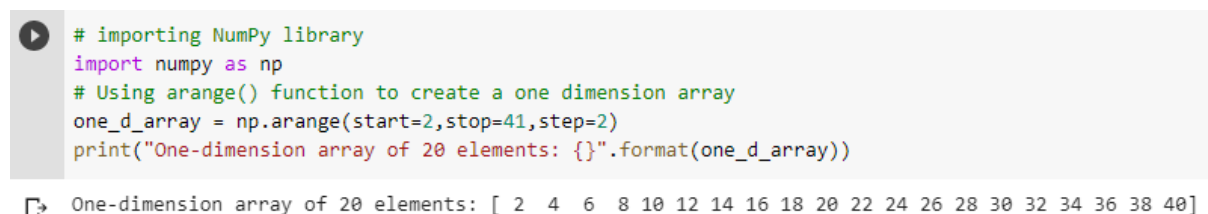
Discussion

The first task is to create a one-dimensional (1-D) with 20 numerical items and print.

Creating a one-dimension array in Python using the NumPy library. The NumPy library was imported and assigned as np. The function arange() used to create the array. This array has 20 elements; started from value 2 ended at value 40, 2 steps between the values.

Figure 1

create a one-dimension array code and output



```
# importing NumPy library
import numpy as np
# Using arange() function to create a one dimension array
one_d_array = np.arange(start=2,stop=41,step=2)
print("One-dimension array of 20 elements: {}".format(one_d_array))
```

One-dimension array of 20 elements: [2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40]

Note. In figure 1, the code is next to the circle that has a triangle inside it and the output is next to the rectangle with the arrow that coming from it.

The second task is to print the dimension of the array.

The function ndim() is used to show the dimensions of the array, which is 1.

Figure 2

print dimension of the array code and output

```
# ndim() function shows the number of dimension
print("The dimension of the array is: {}".format(np.ndim(one_d_array)))
```

```
➤ The dimension of the array is: 1
```

Note. In figure 2, the code shows the dimension of the array.

The third task is to print the array value for index equal 10.

The value in the index 10 was accessed by its index number 10.

Figure 3

print the value in the index 10 codes and outputs

```
print("Value in index 10 is: {}".format(one_d_array[10]))
```

```
➤ Value in index 10 is: 22
```

Note. In figure 3, the values in array can be accessed by its indexes.

The fourth task is to slice and print the array between 5 and 15 indexes. Include both of the arrays.

The new array was created that has the slice from the original array. The new array has the values from index 5 to 15.

Figure 4

slice an array code and output

```
# indexes 5 and 15 included in the array
sliced_array = one_d_array[5:16]
print("The original array before slicing is: {}".format(one_d_array))
print("Sliced array from 5 to 15 indexes are: {}".format(sliced_array))
```

```
➤ The original array before slicing is: [ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40]
   Sliced array from 5 to 15 indexes are: [12 14 16 18 20 22 24 26 28 30 32]
```

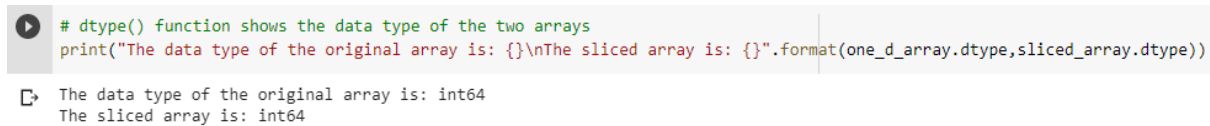
Note. In figure 4, the number 16 does not mean the index 16 is included. In Python, the start value is included, and the stop value does not.

The fifth task is to print the data type of the array.

The function (dtype) coming after the array name is to identify the data type of the two arrays. The original array and the sliced array were printed.

Figure 5

print data type of the array code and output



```
# dtype() function shows the data type of the two arrays
print("The data type of the original array is: {}\nThe sliced array is: {}".format(one_d_array.dtype,sliced_array.dtype))
```

The data type of the original array is: int64
The sliced array is: int64

Note. In figure 5, the code shows the data type of the two arrays.

The sixth task is to make a copy of the array and print it.

The function copy() created a copy from the original array.

Figure 6

create and print a copy code and output



```
array_copy = one_d_array.copy()
print("This is a copy array of the original array: {}".format(array_copy))
```

This is a copy array of the original array: [2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40]

Note. In figure 6, the code shows the way to make a copy from an original array.

The seventh task is to make a view of the array and print it.

The function view() created a view from the original array.

Figure 7

create and print a view code and output



```
array_view = one_d_array.view()
print("This is a view array of the original array: {}".format(array_view))
```

This is a view array of the original array: [2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40]

Note. In figure 7, the code shows the way to make a view from an original array.

The eighth task is to print the shape of the array.

The function (shape) coming after the array name was used to identify the array shape. The number in the first place indicates that the array has 20 elements. The second place is empty because the array is a one-dimension array.

Figure 8

print the shape of the array code and output

```
# shape() function shows the shape of the array
print("The shape of the original array is: {}".format(one_d_array.shape))
```

The shape of the original array is: (20,)

Note. In figure 8, the code shows the shape of an array.

The last task is to reshape the array and print it.

The array was reshaped using the function reshape() and reshaped as a two-dimension array that has 5 rows and 4 columns.

Figure 9

reshape and print code and output

```
# reshape() function used to make a 5x4 array
print("Reshaping the array into 5 rows and 4 columns: \n{}".format(one_d_array.reshape(5,4)))
```

Reshaping the array into 5 rows and 4 columns:

```
[[ 2  4  6  8]
 [10 12 14 16]
 [18 20 22 24]
 [26 28 30 32]
 [34 36 38 40]]
```

Note. In figure 9, the code shows the way to reshape a one-dimension array to a two-dimension array.

References

- Deitel, P. J., & Deitel, H. M. (2020). *Intro to Python for computer Science and data science learning to program with AI, big data and the cloud*. Hudson Street, NY: Pearson Education.
- Kalb, I. (2016). *Learn to program with Python*. Berkeley, CA: Apress.
- Mueller, J., & Emid, A. (2018). *Beginning programming with Python® for dummies®*. Hoboken, NJ: For Dummies, a Wiley brand.