PY0101EN-5-2-Numpy2D

November 17, 2020

```
<a href="https://cocl.us/PY0101EN_edx_add_top">
     <img src="https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClas
</a>
```

2D Numpy in Python

Welcome! This notebook will teach you about using Numpy in the Python Programming Language. By the end of this lab, you'll know what Numpy is and the Numpy operations.

Table of Contents

```
<ul>
    <li><a href="create">Create a 2D Numpy Array</a></li>
    <li><a href="access">Accessing different elements of a Numpy Array</a></li>
    <li><a href="op">Basic Operations</a></li>
</ul>
<p>
    Estimated time needed: <strong>20 min</strong>
</p>
```

Create a 2D Numpy Array

[ ]:
```python
# Import the libraries

import numpy as np
import matplotlib.pyplot as plt
```

Consider the list a, the list contains three nested lists **each of equal size**.

[ ]:
```python
# Create a list

a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
a
```

We can cast the list to a Numpy Array as follow

[ ]:
```python
# Convert list to Numpy Array
# Every element is the same type

A = np.array(a)
A
```

We can use the attribute ndim to obtain the number of axes or dimensions referred to as the rank.

```
[ ]: # Show the numpy array dimensions

     A.ndim
```

Attribute shape returns a tuple corresponding to the size or number of each dimension.

```
[ ]: # Show the numpy array shape

     A.shape
```

The total number of elements in the array is given by the attribute size.

```
[ ]: # Show the numpy array size

     A.size
```

Accessing different elements of a Numpy Array

We can use rectangular brackets to access the different elements of the array. The correspondence between the rectangular brackets and the list and the rectangular representation is shown in the following figure for a 3x3 array:

We can access the 2nd-row 3rd column as shown in the following figure:

We simply use the square brackets and the indices corresponding to the element we would like:

```
[ ]: # Access the element on the second row and third column

     A[1, 2]
```

We can also use the following notation to obtain the elements:

```
[ ]: # Access the element on the second row and third column

     A[1][2]
```

Consider the elements shown in the following figure

We can access the element as follows

```
[ ]: # Access the element on the first row and first column

     A[0][0]
```

We can also use slicing in numpy arrays. Consider the following figure. We would like to obtain the first two columns in the first row

This can be done with the following syntax

[ ]:
```
# Access the element on the first row and first and second columns

A[0][0:2]
```

Similarly, we can obtain the first two rows of the 3rd column as follows:

[ ]:
```
# Access the element on the first and second rows and third column

A[0:2, 2]
```

Corresponding to the following figure:

Basic Operations

We can also add arrays. The process is identical to matrix addition. Matrix addition of X and Y is shown in the following figure:

The numpy array is given by X and Y

[ ]:
```
# Create a numpy array X

X = np.array([[1, 0], [0, 1]])
X
```

[ ]:
```
# Create a numpy array Y

Y = np.array([[2, 1], [1, 2]])
Y
```

We can add the numpy arrays as follows.

[ ]:
```
# Add X and Y

Z = X + Y
Z
```

Multiplying a numpy array by a scaler is identical to multiplying a matrix by a scaler. If we multiply the matrix Y by the scaler 2, we simply multiply every element in the matrix by 2 as shown in the figure.

We can perform the same operation in numpy as follows

[ ]:
```
# Create a numpy array Y

Y = np.array([[2, 1], [1, 2]])
Y
```

[ ]:
```
# Multiply Y with 2

Z = 2 * Y
```

```
Z
```

Multiplication of two arrays corresponds to an element-wise product or Hadamard product. Consider matrix X and Y. The Hadamard product corresponds to multiplying each of the elements in the same position, i.e. multiplying elements contained in the same color boxes together. The result is a new matrix that is the same size as matrix Y or X, as shown in the following figure.

We can perform element-wise product of the array X and Y as follows:

```
[ ]: # Create a numpy array Y

     Y = np.array([[2, 1], [1, 2]])
     Y
```

```
[ ]: # Create a numpy array X

     X = np.array([[1, 0], [0, 1]])
     X
```

```
[ ]: # Multiply X with Y

     Z = X * Y
     Z
```

We can also perform matrix multiplication with the numpy arrays A and B as follows:

First, we define matrix A and B:

```
[ ]: # Create a matrix A

     A = np.array([[0, 1, 1], [1, 0, 1]])
     A
```

```
[ ]: # Create a matrix B

     B = np.array([[1, 1], [1, 1], [-1, 1]])
     B
```

We use the numpy function dot to multiply the arrays together.

```
[ ]: # Calculate the dot product

     Z = np.dot(A,B)
     Z
```

```
[ ]: # Calculate the sine of Z

     np.sin(Z)
```

We use the numpy attribute T to calculate the transposed matrix

```
[ ]: # Create a matrix C

     C = np.array([[1,1],[2,2],[3,3]])
     C
```

```
[ ]: # Get the transposed of C

     C.T
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow this article to learn how to share your work.

Get IBM Watson Studio free of charge!

```
<p><a href="https://cocl.us/PY0101EN_edx_add_bbottom"><img src="https://s3-api.us-geo.objectst
```

About the Authors:

Joseph Santarcangelo is a Data Scientist at IBM, and holds a PhD in Electrical Engineering. His research focused on using Machine Learning, Signal Processing, and Computer Vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Mavis Zhou