2D Numpy in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- Operate comfortably with `numpy`
- Perform complex operations with `numpy`

Table of Contents

Create a 2D Numpy Array

In [1]:
```
# Import the libraries

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

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

In [2]:
```
# Create a list

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

Out[2]:
```
[[11, 12, 13], [21, 22, 23], [31, 32, 33]]
```

We can cast the list to a Numpy Array as follows

In [3]:
```
# Convert list to Numpy Array
# Every element is the same type

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

Out[3]:
```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

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

In [4]:
```
# Show the numpy array dimensions

A.ndim
```

Out[4]:
```
2
```

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

In [5]:
```
# Show the numpy array shape

A.shape
```

Out[5]:
```
(3, 3)
```

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

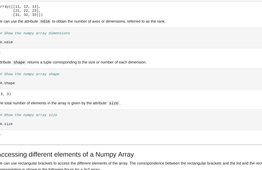
In [6]:
```
# Show the numpy array size

A.size
```

Out[6]:
```
9
```

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:

$$A = \begin{bmatrix} A[0,0] & A[0,1] & A[0,2] \\ A[1,0] & A[1,1] & A[1,2] \\ A[2,0] & A[2,1] & A[2,2] \end{bmatrix}$$



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:

In [7]:
```
# Access the element on the second row and third column

A[1, 2]
```

Out[7]:
```
23
```

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

In [8]:
```
# Access the element on the second row and third column

A[1][2]
```

Out[8]:
```
23
```

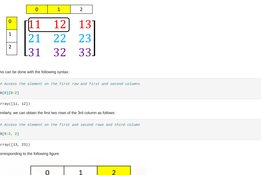
Consider the elements shown in the following figure

We can access the element as follows:

In [9]:
```
# Access the element on the first row and first column

A[0][0]
```

Out[9]:
```
11
```

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

In [10]:
```
# Access the element on the first row and first and second columns

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

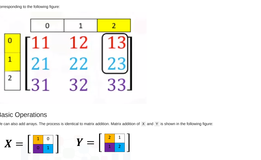
Out[10]:
```
array([11, 12])
```

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

In [11]:
```
# Access the element on the first and second rows and third column

A[0:2, 2]
```

Out[11]:
```
array([13, 23])
```

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:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X + Y = \begin{bmatrix} 1+1 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

The numpy array is given by X and Y:

In [12]:
```
# Create a numpy array X

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

Out[12]:
```
array([[1, 0],
       [0, 1]])
```

In [13]:
```
# Create a numpy array Y

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

Out[13]:
```
array([[1, 1],
       [1, 2]])
```

We can add the numpy arrays as follows.

In [14]:
```
# Add X and Y

Z = X + Y
Z
```

Out[14]:
```
array([[2, 1],
       [1, 3]])
```

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

$$Y = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$2Y = \begin{bmatrix} 2 \times 1 & 2 \times 1 \\ 2 \times 1 & 2 \times 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

We can perform the same operation in numpy as follows:

In [15]:
```
# Create a numpy array Y

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

Out[15]:
```
array([[1, 1],
       [1, 2]])
```

In [16]:
```
# Multiply Y with 2

Z = 2 * Y
Z
```

Out[16]:
```
array([[2, 2],
       [2, 4]])
```

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.

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X \circ Y = \begin{bmatrix} 1 \times 1 & 0 \times 1 \\ 0 \times 1 & 1 \times 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

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

In [17]:
```
# Create a numpy array X

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

Out[17]:
```
array([[1, 0],
       [0, 1]])
```

In [18]:
```
# Create a numpy array Y

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

Out[18]:
```
array([[1, 1],
       [1, 2]])
```

In [19]:
```
# Multiply X with Y

Z = X * Y
Z
```

Out[19]:
```
array([[1, 0],
       [0, 2]])
```

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

First, we define matrix A and B:

In [20]:
```
# Create a matrix A

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

Out[20]:
```
array([[0, 1, 1],
       [1, 0, 1]])
```

In [21]:
```
# Create a matrix B

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

Out[21]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

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

In [22]:
```
# Calculate the dot product

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

Out[22]:
```
array([[2, 2],
       [2, 2]])
```

In [23]:
```
# Calculate the sine of Z

np.sin(Z)
```

Out[23]:
```
array([[0.        , 0.84147098],
       [0.        , 0.84147098]])
```

We use the numpy attribute `T` to calculate the transposed matrix

In [24]:
```
# Create a matrix C

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

Out[24]:
```
array([[1, 1, 1],
       [2, 2, 2],
       [3, 3]])
```

In [25]:
```
# Get the transpose of C

C.T
```

Out[25]:
```
array([[2, 2],
       [2, 2]])
```

Quiz on 2D Numpy Array

Consider the following list `a`, convert it to Numpy Array

In [26]:
```
# Write your code below and press Shift+Enter to execute

a = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
A = np.array(a)
A
```

Out[26]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

► Click here for the solution

Calculate the numpy array size.

In [27]:
```
# Write your code below and press Shift+Enter to execute

A.size
```

Out[27]:
```
12
```

► Click here for the solution

Access the element on the first row and first second columns.

In [28]:
```
# Write your code below and press Shift+Enter to execute

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

Out[28]:
```
array([1, 2])
```

► Click here for the solution

Perform matrix multiplication with the numpy arrays A and B.

In [29]:
```
# Write your code below and press Shift+Enter to execute

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

Out[29]:
```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]])
```

► Click here for the solution

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-01-10	3.1	Malika	Removed the readme for GitView
2021-01-05	3.2	Malika	Updated the solution for 2d multiplication
2020-08-09	3.1	Malika	Updated the screenshots for the two rows of the 3rd column
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab