Subject: Artificial intelligence

This notebook will teach you about the libraries, matrices, operators, arrays in the Python Programming Language. By the end of this lab, you'll know the basic concepts about libraries, matrices, operators, arrays, and how to use these functions

What is NumPy?

NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

```
import numpy as np
```

What is Panda?

Pandas is one of the tools in Machine Learning which is used for data cleaning and analysis. It has features which are used for exploring, cleaning, transforming and visualizing from data. ... It provides fast, flexible, and expressive data structures.

```
import pandas as pd
```

Relational Operators:

```
x = np.array([1, 2, 3, 4, 5])

x < 3  # less than
        ➢ array([ True,  True, False, False, False])
x > 3  # greater than
        ➢ array([False, False, False,  True,  True])
x <= 3  # less than or equal
        ➢ array([ True,  True,  True, False, False])
x >= 3  # greater than or equal
        ➢ array([False, False,  True,  True,  True])
x != 3  # not equal
        ➢ array([ True,  True, False,  True,  True])
x == 3  # equal
        ➢ array([False, False,  True, False, False])
(2 * x) == (x ** 2)
```

- ➢ array([False, True, False, False, False])

print(x)

- ➢ [1 2 3 4 5]

How to Print Tables in Colab?

```
a=int(input("enter table number"))
b=int(input("enter the number to which table is to printed"))
i=1
while i<=b:
   print(a,"x",i,"=",a*i)
   i=i+1
```

```
enter table number3
enter the number to which table is to printed6
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
```

NumPy Array Attributes:

First let's discuss some useful array attributes. We'll start by defining three random arrays, a one-dimensional, two-dimensional, and three-dimensional array. We'll use NumPy's random number generator, which we will *seed* with a set value in order to ensure that the same random arrays are generated each time this code is run:

- ❖ import numpy as np

```
np.random.seed(0)  # seed for reproducibility

x1 = np.random.randint(10, size=6)  # One-dimensional array
x2 = np.random.randint(10, size=(3, 4))  # Two-dimensional array
x3 = np.random.randint(10, size=(3, 4, 5))  # Three-dimensional arra
```

Each array has attributes ndim (the number of dimensions), shape (the size of each dimension), and size (the total size of the array):

```
print("x3 ndim: ", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)

x3 ndim:  3
x3 shape: (3, 4, 5)
x3 size:  60
```

Array Slicing: Accessing Subarrays:

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the *slice* notation, marked by the colon (:) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array x, use this:

```
x[start:stop:step]
```

If any of these are unspecified, they default to the values start=0, stop=*size of dimension*, step=1. We'll take a look at accessing sub-arrays in one dimension and in multiple dimensions.

One-dimensional subarrays

```
x = np.arange(10)
x
        ➢ array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[:5]  # first five elements
        ➢ array([0, 1, 2, 3, 4])
x[5:]  # elements after index 5
        ➢ array([5, 6, 7, 8, 9])
x[4:7]  # middle sub-array
        ➢ array([4, 5, 6])
```

A potentially confusing case is when the step value is negative. In this case, the defaults for start and stop are swapped. This becomes a convenient way to reverse an array:

```
x[::-1]  # all elements, reversed
        ➢ array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
x[5::-2]  # reversed every other from index 5
        ➢ array([5, 3, 1])
```

Splitting of Arrays:

The opposite of concatenation is splitting, which is implemented by the functions np.split, np.hsplit, and np.vsplit. For each of these, we can pass a list of indices giving the split points:

```
x = [1, 2, 3, 99, 99, 3, 2, 1]
x1, x2, x3 = np.split(x, [3, 5])
print(x1, x2, x3)
```

- [1 2 3] [99 99] [3 2 1]

Matrices:

```
A = np.matrix([[5, 6, 2],
               [4, 7, 19],
               [0, 3, 1    2]])
A
```

- matrix([[5, 6, 2],

```
        [ 4,  7, 19],
        [ 0,  3, 12]])
B = np.matrix([[14, -2, 12],
               [4, 4, 5],
               [5, 5, 1]])
B
```

- matrix([[14, -2, 12],

```
        [ 4,  4,  5],
        [ 5,  5,  1]])
5 * A
```

- matrix([[25, 30, 10],

```
        [20, 35, 95],
        [ 0, 15, 60]])
A ** 3
matrix([[ 557, 1284, 3356],
        [ 760, 2305, 6994],
        [ 288, 1074, 3519]])
```