Python3 And OpenCV.

1) Getting Started with NumPy.

NumPy: Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, an tools for working with these arrays.

To import numpy ---->

In [1]:
```
import numpy as np
```

1. 1) Arrays.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

In [11]:

```python
## Creating a simple one dimentional (a rank 1) ndArray
x = np.array([1, 2, 3], np.int16)

## Printing the type of x.
print('1-----------\n')
print(type(x))
print('\n------------\n')


## Printing the shape of x.
print('\n2-----------\n')
print(x.shape)
print('\n------------\n')

## Adressing elements.
print('\n3-----------\n')
print(x[0], x[1], x[2])
print('\n------------\n')


# Changing an element of the array.
x[0] = 5
print('\n4-----------\n')
print(x)
print('\n------------\n')
```

```
1-----------

<class 'numpy.ndarray'>

------------


2-----------

(3,)

------------


3-----------
```

```
1 2 3

------------


4-----------

[5 2 3]

------------
```

In [14]:

```python
## Creating a Two  dimentional (a rank 2) ndArray
x = np.array([[1, 2, 3], [4, 5, 6]], np.int16)

## Printing x.
print('1-----------\n')
print(x)
print('\n------------\n')

## Printing the shape of x.
print('\n2-----------\n')
print(x.shape)
print('\n------------\n')

## Adressing elements.
print('\n3-----------\n')
print(x[0, 0], x[0, 1], x[0, 2])
print('\n------------\n')

# Changing an element of the array.
x[0, 0] = 5
print('\n4-----------\n')
print(x)
print('\n------------\n')
```

```
1-----------

[[1 2 3]
 [4 5 6]]

------------


2-----------

(2, 3)

------------


3-----------

1 2 3
```

```
------------


4-----------

[[5 2 3]
 [4 5 6]]

------------
```

In [15]:
```
## Out of bound Exeption.
print(x[3][2])
```

```
---------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-a1bcb3dc75d8> in <module>
      1 ## Out of bound Exeption.
----> 2 print(x[3][2])

IndexError: index 3 is out of bounds for axis 0 with size 2
```

Array Slicing.

Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimensio the array.

In [19]:

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

print('\n----------------\n')

## Pulling out the subarray consisting of the first 2 rows and columns 1 and 2.
b = a[:2, 1:3]

print(b)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

----------------

[[2 3]
 [6 7]]
```

Note: A slice of an array is a view into the same data, so modifying it will modify the original array.

In [20]:

```
print(a[0, 1])

print('\n------\n')

b[0, 0] = 77
print(a[0, 1])
```

```
2

------

77
```

Note: Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array:

In [23]:
```
row_r1 = a[1, :]
row_r2 = a[1:2, :]

print('Array', row_r1, 'Rank: ', row_r1.ndim, 'Shape: ', row_r1.shape)
print('Array', row_r2, 'Rank: ', row_r2.ndim, 'Shape: ', row_r2.shape)
```

```
Array [5 6 7 8] Rank:  1 Shape:  (4,)
Array [[5 6 7 8]] Rank:  2 Shape:  (1, 4)
```

1. 2) NumPy Ndarray Properties.

In [27]:

```
# To show the structure of the array (the size of the array along each dimension).
print('1-----------\n')
print(x)
print('\n------------\n')

print('2-----------\n')
print(x.shape)
print('\n------------\n')

# To show the number of dimensions (the rank of the array).
print('3-----------\n')
print(x.ndim)
print('\n------------\n')

# To show the datatype.
print('4-----------\n')
print(x.dtype)
print('\n------------\n')

# To show the size (the total number of elements inside the array).
print('5-----------\n')
print(x.size)
print('\n------------\n')

# To show the size in terms of bytes.
print('6-----------\n')
print(x.nbytes)
print('\n------------\n')

# To show the transpose of the given array.
print('7-----------\n')
print(x.T)
print('\n------------\n')
```

```
1-----------

[[5 2 3]
 [4 5 6]]

------------

2-----------
```

```
(2, 3)

------------

3-----------

2

------------

4-----------

int16

------------

5-----------

6

------------

6-----------

12

------------

7-----------

[[5 4]
 [2 5]
 [3 6]]

------------
```

1. 3) NumPy Ndarray Datatypes.

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to const arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

In [30]:

```
# Let numpy choose the datatype
y = np.array([1, 2])
print('1-----------\n')
print(y.dtype)
print('\n------------\n')

# Force a particular datatype.
y = np.array([1, 2], dtype=np.int64)
print('2-----------\n')
print(y.dtype)
print('\n------------\n')
```

```
1-----------

int32

------------

2-----------

int64

------------
```