Welcome to the tutorial on NumPy

In this document, we'll talk about the following:

- NumPy

NumPy

NumPy (https://numpy.org/) is an open source project used to enable numerical computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
We generally import NumPy with the following line:

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

Some frequently used functions are:

`np.array(array,[dtype=None, order='K', ndmin=0])`

We use this function to create a numpy array.

Parameters

array: Python array which should be made into numpy array
dtype(Optional): Data type of the elements in the array
order(Optional): *{'K','C', 'F'}* Specify the memory layout of the array Row major `C` or Column major `F`. Use `K` to keep the order same as the source array. **ndmin(Optional)**: Specifies the minimum number of dimensions that the resulting array should have

Returns

Returns the numpy array

In [2]:
```
np.array([1,2,3])
```

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

Every numpy array has the following properties:

Propery Name	Example
Dimensions	`array_name.ndim`
Shape	`array_name.shape`
Number of elements	`array_name.size`
Data type	`array_name.dtype`
Size of each element in bytes	`array_name.itemsize`

Let's see some examples

In [3]:
```
a = np.array([1,2,3])
print(a.shape)
print(a.ndim)
```

```
(3,)
1
```

`np.zeros(shape, [dtype=None, order='C'])`

We use this function to create a numpy array filled with zeros

Parameters

shape: Shape of the array
dtype(Optional): Data type of the array
order(Optional): *{'K','C', 'F'}* Specify the memory layout of the array Row major `C` or Column major `F`.

Returns

Returns a array filled with **zeros** of the given shape and type

In [4]:
```
np.zeros((3,2), dtype=np.int32)
```

Out[4]:
```
array([[0, 0],
       [0, 0],
       [0, 0]], dtype=int32)
```

np.ones(shape, [dtype=None, order='C'])

We use this function to create a numpy array filled with ones

Parameters

shape: Shape of the array
dtype(Optional): Data type of the array
order(Optional): *{'K','C', 'F'}* Specify the memory layout of the array Row major `C` or Column major `F` .

Returns

Returns a array filled with **ones** of the given shape and type

In [5]:
```
np.ones((1,5))
```

Out[5]:
```
array([[1., 1., 1., 1., 1.]])
```

np.empty(shape, [dtype])

Parameters

shape: Shape of the array
dtype(Optional): Data type of the array
order(Optional): *{'K','C', 'F'}* Specify the memory layout of the array Row major `C` or Column major `F` .

Returns

Return a new array of given shape and type, without initializing entries.

In [6]:
```
np.empty((2,3))
```

Out[6]:
```
array([[4.6879644e-310, 0.0000000e+000, 0.0000000e+000],
       [0.0000000e+000, 0.0000000e+000, 0.0000000e+000]])
```

Note: The values shown above are junk values

numpy.arange([start,]stop, [step, dtype=None])

This function gives us an 1D array with evenly spaced values within a given interval.

Parameters

stop: Stop of interval
start(Optional): Start of interval
step(Optional): Spacing between values
dtype(Optional): Data type of the array

Returns

Array of evenly spaced values

In [7]:
```
np.arange(10, 20, 2)
```

Out[7]:
```
array([10, 12, 14, 16, 18])
```

Operations on Numpy arrays

Most operation which are possible on normal Python variables are also available on numpy array.

In [8]:
```
a = np.array([1,2,3])
b = np.array([4,5,6])
```

In [9]:
```
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

```
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

Note: The operations performed here (especially multiplication) are performed element wise

There are also some additional operations which are exclusive to matrices which are simple to implement via the Numpy package

`np.resize(old_array, new_shape)`

We use this function to resize an existing numpy array

Parameters

old_array: Original array to be reshaped
new_shape: int or tuple of int of the new shape

Returns

The new array is formed from the data in the old array. *Data is repeated if necessary to fill out the required number of elements*

In [10]:
```
a = np.arange(11, 21)
print("a : \n",a)
print("a's shape = ",a.shape)
a = np.resize(a, (2,5))
print("New a : \n",a)
print("New a's shape = ",a.shape)
```

```
a :
 [11 12 13 14 15 16 17 18 19 20]
a's shape =  (10,)
New a :
 [[11 12 13 14 15]
 [16 17 18 19 20]]
New a's shape =  (2, 5)
```

Array slicing

Slicing means extracting elements from one given index to another given index Some key points to remember when slicing are:

- Slicing is done by passing indexing like this: `[start:end]`
- We can also define the step, like this: `[start:end:step]`
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

Let's look at some examples

In [11]:
```
a = np.arange(11, 21)
print("a      = \t",a)
print("a[2:5] = \t",a[2:5])
print("a[:5]  = \t",a[:5])
print("a[7:]  = \t",a[7:])
print("a[::3] = \t",a[::3])
```

```
a      =        [11 12 13 14 15 16 17 18 19 20]
a[2:5] =        [13 14 15]
a[:5]  =        [11 12 13 14 15]
a[7:]  =        [18 19 20]
a[::3] =        [11 14 17 20]
```

`np.dot(a, b)`

Parameters

a: First array argument
b: Second array argument

Returns

Returns the following:

- If both a and b are 1-D arrays, it is inner product of vectors

- If both a and b are 2-D arrays, it is matrix multiplication(cross product)
- If either a or b is 0-D (scalar), it is equivalent to multiply and using numpy.multiply(a, b)

In [12]:
```
a = np.array([1,2,3])
b = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("a = \n",a)
print("b = \n",b)
print("np.dot(a,b) = \n",np.dot(a,b))
```

```
a =
 [1 2 3]
b =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
np.dot(a,b) =
 [30 36 42]
```

Putting it all together

Let's make a program which uses some of the functions mentioned above. We will be making program which takes pictures from the webcam.

NOTE: Though you may not be using NumPy functions directly, OpenCV uses them for a lot of process.

Please run this locally on your system and not on the Jupyter Notebook(NB).

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert Color to Grayscale fram
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```