Venkatesh. K.H Date 18/11/2019

Numpy

When you are learning data Science, you can't miss Numpy. Numpy is a module for python. The name is an acronym for "Numeric python" & "Numerical python" Numpy is core library for Scientific computing in Python. It provides a high-performance multi-dimensional array object.

Below is command to import the Numpy module.

```
import numpy
```

But, the better way of importing the Numpy is given below

Author: Travis Oliphant

```
import numpy as np
```

A lightweight alternative is to install Numpy using popular Python package installer, pip.

```
pip install numpy
```

How are lists different from Numpy?

- Lists, they are very slow meanwhile numpy is very faster.

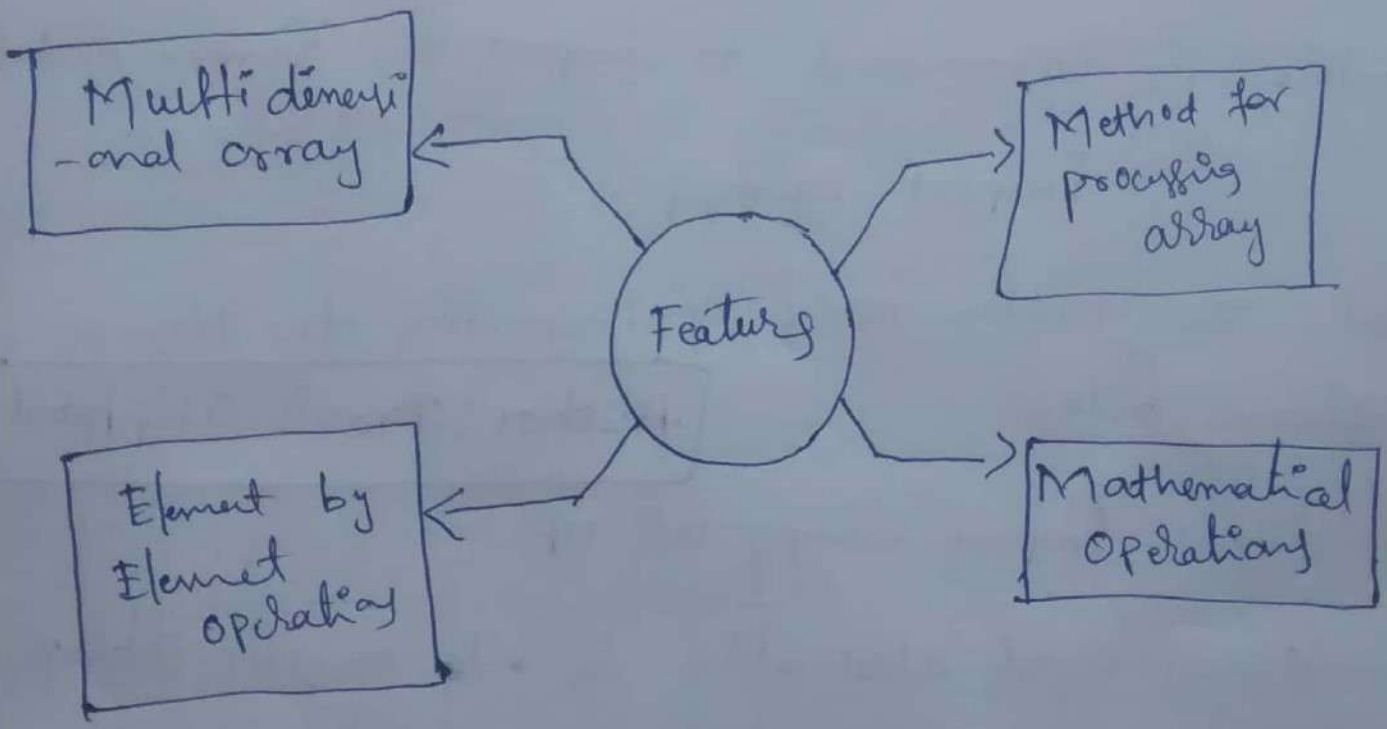
Memory

The main benfits of using Numpy arrays should

be Smaller memory consumption and better runtime behavior.

Applications of Numpy ?

- Mathematics
- Plotting (Matplotlib)
- Machine learning



Single dimensional array

```
import numpy as np
Hello = np.array([4,8,9])
print(Hello)
```

→ Single dimensional array.

Output:- [4 8 9]

Multi Dimensional Array

```
import numpy as np
hi = np.array([[3,1],[8,6]])
print(hi)
```

→ Multi dimensional array.

output: [[3,1],[8,6]].

Numpy

```
import numpy as np
x = np.array([1,2,3], np.int16)
print (x)
print (type(x))
```

→ List

16 bit integer is the data type of the elements within the nd array

output:- [1 2 3]

<class 'numpy.ndarray'>

One of the most important data Structure. ndarray → stands for 'n' dimension array.

```
print(x[0]); print(x[1]); print(x[2])
```

o/p: 1 2 3 → Accessing index value.

```
import numpy as np
x = np.array([1,2,3], np.int16)
print(x[3])
```

Throws an Error

Output:- Index Error. Traceback
--> print(x[3])

Index Error: index 3 is out of bounds for axis 0 with size 3.

Two dimensional array Example

```
import numpy as np
x = np.array([[1,2,3],[4,5,6]], np.int16)
print(x)
```

O/P:- [[1 2 3] [4 5 6]]

```
print(x[0,0]); print(x[0,1]); print(x[0,2])
```

O/P:-
1
2
3

Slice the data Structure around two axis

Let us see now, how to slice this particular ndarray

```
import numpy as np
x = np.array([[1,2,3],[4,5,6]], np.int16)
print(x)
```

o/p: [1 2 3] [4 5 6].

```
print(:,0)
```

o/p: invalid Syntax

```
print(x[:,0])
```
→ which print 1st column

o/p: [1 4]

```
print(x[:,1])
```
→ which print 2nd column

o/p: [2 5]

```
print(x[1,:])
```
→ which print Entire Second row

o/p: 4 5 6

Numpy Ndarray properties

```
x = np.array([[[1,2,3],[4,5,6]],[[0,-1,-2],
    [-3,-4,-5]]], np.int16)
print(x)
```

O/P:- `[[[1 2 3] [4 5 6]] [[0 -1 -2] [-3 -4 -5]]]`

```
print(x.shape)
```

O/P:- (2, 2, 3) -> 3 Entries in the Shape

There is another way to check that.

```
print(x.ndim) -> number of dimensions (ndim)
```

O/P:- 3

How to check data type?

```
print(x.dtype)
```

O/P:- int16

How to check size of the ndarray?

```
print(x.size)
```

O/P:- 12 -> Total 12 Elements present in the array.

print (x.T) → o/p: $\begin{bmatrix}\begin{bmatrix}1 & 0\\ 4 & -3\end{bmatrix} & \begin{bmatrix}2 & -1\\ 5 & -4\end{bmatrix} & \begin{bmatrix}3 & -2\\ 6 & -5\end{bmatrix}\end{bmatrix}$

↓

which is used for transpose matrix.

Numpy Constants → inf

print (np.inf) → +ve infinity

print (np.NAN) → Not a Number → o/p: nan

print (np.NINF) → -ve infinity → o/p: -inf

print (np.NZERO) → -ve zero → o/p: -0.0

print (np.PZERO) → +ve zero → o/p: 0.0

Scientific Constants of Numpy

print (np.e) → 2.718281828459045

print (np.Euler-gamma) → 0.5772156649015329

print (np.pi) → 3.141592653589793.

```
# numpy.Eye method
import numpy as np
x = np.eye(3, dtype=np.uint8)
print (x)
```

((reaty)

o/p: $\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ → which prints an array where all the diagonal Elements are 1.

```
x = np.identity(3, dtype = np.uint8)
```

O/P :-

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} [1\ 0\ 0] \\ [0\ 1\ 0] \\ [0\ 0\ 1] \end{bmatrix}$$

We can Even create a matrix of One's

```
x = np.ones((2,5,5), dtype = np.int16)
print (x)
```

O/P :-

```
[[[[1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]]

  [[1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]
   [1 1 1 1 1]]]]
```

Attributes

* arange
* linspace * Split
* logspace * resize
* reshape * append
* copy * insert
* nditer * delete
* flat * unique
* flatten * capitaliz
* ravel * strip()
* rollaxis * join(
* swapaxes * decode
* broadcast
* Empty * Encode
* hSplit * sin
* VSplit * cos
* tan.

Even we can create zero's

```
x = np.zeros((2,3,3), dtype = np.int16)
print (x)
```

```
[[[0 0 0]  [0 0 0]
  [0 0 0]  [0 0 0]
  [0 0 0]  [0 0 0]].
```