NUMPY-LIBRARY--NUMERIC PYTHON :

1. NUMPY :

PAGE 170

In [131]:
```
# NOTE : NUMPY

# 1. The 'NUMPY' is also called as 'NUMERIC PYTHON'.
# 2. In 'NUMPY' - We are going to handle data -'Technically' -
#    With the help of 'LINEAR ALZEBRIC EXPRESSION' (OR) 'LINEAR ALZEBRA'
# 3. 'NUMPY' - Specially designed to 'Handle the NUMERIC DATA'
# 4. NUMERIC DATA - Which includes -> INTEGERS,FLOAT,DOUBLE DATA,BIG INTEGERS AND SMALL INTEGERS.
# 5. Any DATA We call, by using an a 'NUMPY' - Going to be called into an a 'MATRIX PATTERN'.

# 6. The Entire 'MACHINE LEARNING' (OR) 'DATA SCIENCE' WORKING ENVIRONMENT ->
#    depends on 'NUMPY' to 'Handle the DATA EFFECTIVELY'.

# 7. Whenever we want to Work with 'NUMPY'. First, We have to Call 'NUMPY' is the most imp thing.
#    -->  []: import numpy as np
```

In [2]:
```
# MATRIX PATTERN :

# [1,2,3,4,5] - Called as 'ONE DIMENSIONAL' (OR) 'VECTOR DATA' (OR) 'SINGLE LINE OBJECT'

# [3][5] - Called as 'SCALER OBJECTS' (OR) 'SINGLE OBJECT'.

# [1,2
#  3,4] - Matrix Size (2 * 2), 'TWO DIMENSIONAL MATRIX'.


# 'ABOVE TWO DIMENSION DATA' - WE CALL IT AS " TENSOR DATA "

# We have an a 'Individual Library' as an a 'TENSOR FLOW' in 'DEEP LEARNING AREA'.

# TENSOR = means,'MULTI DIMENSION'.
```

In [40]:
```
# TensorFlow is an open-source library developed by Google primarily for deep learning applications.
# It also supports traditional machine learning.
# TensorFlow was originally developed for large numerical computations
# without keeping deep learning in mind.

# What TensorFlow is used for?
# TensorFlow is an end-to-end open source platform for machine learning.
# TensorFlow is a rich system for managing all aspects of a machine learning system;
# however, this class focuses on using a particular 'TensorFlow API' to develop and train
# machine learning models.
# TensorFlow is an open-source machine learning framework,
# and Python is a popular computer programming language.
# It's one of the languages used in TensorFlow.
# Python is the recommended language for TensorFlow, although it also uses C++ and JavaScript.

# An API, or 'Application Programming Interface', is a server that you can use to retrieve and
# send data to using code.
# APIs are most commonly used to retrieve data, and that will be the focus of this beginner tutorial.
# When we want to receive data from an API, we need to make a request.
# Requests are used all over the web.

# 'array' is 'Mandatory' - 'np.array'
# Whatever data we call in 'numpy' - The Set of data is Called as -> 'Array of Data'.

# Even 'IMAGES' or 'Any IMAGE' Can be Called in the 'np.array'
# 'PIXELS' are going to be Readed in the 'MATRIX FORMAT'.
# 'PIXELS' - IMAGES like, HD,UHD,STANDARD e.t.c those pixels are technically readed in 'MATRIX FORMAT'
# Each and Every Image (or) Frame pixels are going to be readed in the 'MATRIX FORMAT'.
```

In [36]:
```
# When we have 'Single line data' - Then We Call it as an a 'VECTOR' - '1' Dimensional representation.

# The Matrix pattern Can be readed in the '2' dimensional representation of 'x' and 'y'.

# 'ROW' - We Call it as an a 'x' pattern.
# 'COLUMN' - We Call it as an a 'y' pattern.

# The '3rd' Dimension - is Called as an a 'z' Dimension
```

In [52]:
```
# Whenever the Data in the 'Matrix Pattern', it will be very flexible to fetch -
# not only the 'ROWS & COLUMNS', We Can fletch the Data in-between of 'ROWS & COLUMNS'
```

In 'NUMPY' - All the Objects Called as - "numpy.array"

In [7]:
```
# NOW WE NEED TO CALL -> 'NUMPY' :

import numpy as np
```

In [8]:
```
# Here, my_list is showing -> 'List of Objects'

my_list = [1,2,3]
```

In [9]:
```
type(my_list)
```

Out[9]: list

In [10]:
```
len(my_list)
```

Out[10]: 3

In [11]:
```
# 'n' dimensional means, 'NUMBER OF DIMENSIONS ARRAY'
# Now We are 'Converting' - 'LIST DATA into numpy.array'

arr = np.array(my_list)
```

In [16]:
```
# output : numpy.ndarray - means, 'numpy 'n' dimensional array'(nd)

type(arr)
```

Out[16]:
```
numpy.ndarray
```

In [18]:
```
# Here, it is Showing 'array of Objects'
# Because, We have Converted Objects into 'list to array''

arr
```

Out[18]:
```
array([1, 2, 3])
```

In [19]:
```
# Difference of 'list to array':

# (a) The list of Objects - [1,2,3]
# (b) numpy.array Objects - array([1,2,3])
```

The 'SHAPE OF arr': Identifying 'SHAPE'

PAGE 173

In [21]:
```
# Here, The OUTPUT (3,) -> '1' dimension (or) 'VECTOR'
# means, Only '3' objects it have, single line data
# '2' dimensional means, 'MATRIX'
# Above '3' dimensional, 'TENSOR'

arr.shape
```

Out[21]: (3,)

Identifying DIMENSIONS :

PAGE 174

In [23]:
```
# ndim - means, Number of Dimensions :

arr.ndim
```

Out[23]: 1

In [24]:
```
# Now another example :

b  = np.array([[0,1,2],
               [3,4,5]])
```

In [26]:
```
# (2, 3) - 2 by 3 means, 2 rows ans 3 columns

b.shape
```

Out[26]: (2, 3)

In [31]:
```
b.ndim
```

Out[31]: 2

In [32]:
```
# Another example of '2' dimensional data :

c = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

In [33]:
```
c
```

Out[33]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [34]:
```
c.shape
```

Out[34]:
```
(3, 3)
```

In [35]:
```
c.ndim
```

Out[35]:
```
2
```

'3' Dimensional Representation :

page 178

In [37]:
```
x = np.array([[[0,1],[2,3]],[[4,5],[6,7]]])
```

In [39]:
```
# This is Called as '3' Dimensional Representation :

x
```

Out[39]:
```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

In [50]:
```
x.shape
```

Out[50]:
```
(2, 2, 2)
```

In [51]:
```
x.ndim
```

Out[51]:
```
3
```

In [41]:
```
# Now i want to Generate something like :
```

In [44]:
```
# Here, c = np.array (or) We can call 'np.zeros' directly
# Here, i required '3' zeros.

c = np.zeros(3)
```

In [53]:
```
# Here, The Zeros are 'float data type'

c
```

Out[53]:
```
array([0., 0., 0.])
```

In [57]:
```
# It will throw Error, Because We need to Declare the '2' dimensional (()) to the values.

c2 = np.zeros(3,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20532\3773320116.py in <module>
----> 1 c2 = np.zeros(3,3)

TypeError: Cannot interpret '3' as a data type
```

In [58]:
```
# Default Data Type of 'NUMPY OBJECT' is 'FLOAT'
# Generating the Set of Zero'0' Objects.
# (3,3) 3 by 3 means, Matrix only
# Whenever We declare Matrix Data, Mandatorily we need to Declare '2' dimensional (()) brackets.

c2 = np.zeros((3,3))
```

In [59]:
```
c2
```

Out[59]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [60]:
```
c2.ndim
```

Out[60]:
```
2
```

In [61]:
```
c2.shape
```

Out[61]:
```
(3, 3)
```

2. LINSPACE ( ) : NUMPY

PAGE 180

In [72]:
```
# Now Here, Wherever We are Working with an a
#'PANDAS' and 'MATPLOTLIB(Plotting Data)-> means plotting the points' ->
# We go with an a 'x' and 'y' axis.

# When we call the Data into 'plotting points', First we need to have an a 'x' and 'y' plotting points

# eg: whenever we take Graph paper, First we need to have the differenciation points of 'x' and 'y'.
# Based on this 'x' and 'y' only the data points are going to be plotted on 'Graph' in general.

# Here, Manually i don't want to declare this 'x' and 'y' points:
# for that, We have an a Some Technical method(), Which We Call -> 'LINSPACE METHOD( )'

# In 'Matplotlib' - We need to plot this particular Graphical Points 'Manually'.
# But Here, 'Manually' We 'no need' to declare, We have an a Method Called 'LINSPACE( )'
```

In [73]:
```
# 'LINSPACE METHOD( )' -
#     Will helps us to Point Equal Number Of Points On 'x' axis (or) 'y' axis (or) 'z' axis.
```

In [74]:
```
# Here, '0' is Starting no./ '1' is Ending no. / '6' is required Points.

a = np.linspace(0,1,6)
a
```

Out[74]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

In [76]:
```
# In 'numpy' We Can Generate 'Ones' also :

c3 = np.ones((3,4))
c3
```

Out[76]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [78]:
```
# Here, 'NUMPY' has 'no attribute' of 'twos', that's why it has thrown 'Error' :

c5 = np.twos((3,3))
c5
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20532\1945999028.py in <module>
      1 # Here, 'NUMPY' has 'no attribute' of 'twos', that's why it has thrown 'Error' :
      2
----> 3 c5 = np.twos((3,3))
      4 c5

~\anaconda3\lib\site-packages\numpy\__init__.py in __getattr__(attr)
    311                 return Tester
    312
--> 313             raise AttributeError("module {!r} has no attribute "
    314                                  "{!r}".format(__name__, attr))
    315

AttributeError: module 'numpy' has no attribute 'twos'
```

3. IDENTICAL MATRIX : NUMPY

PAGE 182

In [80]:
```
np.eye(3)   # Means, (3,3) 3 by 3 'Default Matrix' We are getting :
```

Out[80]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [85]:
```
np.eye(4)   # Means, (4,4) 4 by 4 'Default Matrix' We are getting :
```

Out[85]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

4. DIAGONAL MATRIX : NUMPY

PAGE 183

In [98]:
```
# Diagnal work only for 'MATRIX' generally in real time :
# Even 'vector' also it won't work without double brackets :
# Diagnally it is printing :

c6 = np.diag((2,3))
c6
```

Out[98]:
```
array([[2, 0],
       [0, 3]])
```

In [99]:
```
c7 = np.diag((1,2,3,4))
```

In [100]:
```
c7
```

Out[100]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

In [97]:
```
c7 = np.diag((1))
c7
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20532\1713242199.py in <module>
----> 1 c7 = np.diag((1))
      2 c7

<__array_function__ internals> in diag(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\lib\twodim_base.py in diag(v, k)
    301         return diagonal(v, k)
    302     else:
--> 303         raise ValueError("Input must be 1- or 2-d.")
    304
    305

ValueError: Input must be 1- or 2-d.
```

In [103]:
```
# Diagnal Work only for 'Matrix' :

np.diag((2,2))
```

Out[103]:
```
array([[2, 0],
       [0, 2]])
```

5. RANDOM NUMBERS : NUMPY

PAGE 184

In [104]:
```
# Now i want to Generate a Random Numbers from Numpy :
```

Only Positive Numbers : Random : Numpy :

In [106]:
```
# Here, We can also write it like -> 'np.random.randn', means we can work it with adding 'n' to 'rand'

a = np.random.rand(4)
a
```

Out[106]:
```
array([0.61176864, 0.57255689, 0.26197236, 0.25521162])
```

Includes with NEGATIVE Numbers : Random : Numpy :

In [108]:
```
# Here, We can also write it like -> 'np.random.randn', means we can work it with adding 'n' to 'rand'

b = np.random.randn(4)
b
```

Out[108]:
```
array([-1.86957025,  0.97059933, -1.09654578,  1.8811768 ])
```

In [ ]:
```

```

In [114]:
```
# Inside the Matrix also we can Generate Random :

c = np.random.rand(6)
c
```

Out[114]:
```
array([0.36346269, 0.56875543, 0.94526326, 0.16287396, 0.62926251,
       0.10819326])
```

6. Defaultly we get 'int' Data Type : NUMPY

PAGE 185

In [115]:
```
d = np.arange(10)
d
```

Out[115]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [119]:
```
# Now In this particular 'arange' method also,we can do some particular changes like :
# Here, We are giving float64 data type. The 'OUTPUT' we are going to get in 'float data'

e = np.arange(10,dtype = 'float64')
e
```

Out[119]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

7. VIEWS : NUMPY

PAGE 185

In [120]:
```
# 'VIEW' is nothing but Whenever we  want to change some particular value inside.
# The Highend advantage of a 'Python' is -> "We can SHARE THE MEMORY".
# Instead of declaring, directly we can Share the Memorey by using 'VIEWS'.

# VIEWS : By using 'VIEWS' We can Share the Memory (or) We can Identify the Memory Shared or not.
```

In [121]:
```
a = np.arange(10)
a
```

Out[121]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [126]:
```
# Here, We mentioned b = a [::2],
# Here [::2] 'colon of colon of 2' means,
# Starting Number : Ending Number : Step of 2.

# Here, We are getting 'Even Numbers' in 'b' :

b = a [::2]
b
```

Out[126]:
```
array([0, 2, 4, 6, 8])
```

In [127]:
```
# Memory Shared :

np.shares_memory(a,b)
```

Out[127]:
```
True
```

In [139]:
```
c = np.arange(10)
c
```

Out[139]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [141]:
```
# Now 'a,b,c' we have:Now i'm trying to find,Wheather the Memory of 'a' & 'c' has been shared (or) not
# It is giving Simply 'FALSE', Because, it is not identified like b = a[::2]

np.shares_memory(a,c)
```

Out[141]:
```
False
```

8. randint : Random Numbers : NUMPY

PAGE 188

In [152]:
```
# Here, From '0' to '20' and i want to get Number of Objects '15' :

a = np.random.randint(0,20,15)
a
```

Out[152]:
```
array([ 7,  9,  0,  6,  1, 16, 13,  4,  6,  0, 17,  1,  6, 15,  5])
```

9. MASKING : We Can Change the Numbers : Based on our requirement : NUMPY :

PAGE 189

In [153]:
```
mask = (a % 2 == 0)   # 'a' divided by 2 == 0
extract_a_mask = a[mask]
```

In [154]:
```
# Here, Masking Particular 'Even' Numbers :

extract_a_mask
```

Out[154]:
```
array([ 0,  6, 16,  4,  6,  0,  6])
```

Changing 'Even' Numbers to '-1' :

In [155]:
```
a [mask] = -1   # Here, 'a' means Entire Data.
```

In [156]:
```
# 143rd sum values, previously we have generated set of objects by 'randint()'
# Now, These 'Even' Numbers will be changed to '-1'
# page 189

a
```

Out[156]:
```
array([ 7,  9, -1, -1,  1, -1, 13, -1, -1, -1, 17,  1, -1, 15,  5])
```

In [157]:
```
# Now another example:

b = np.arange(0,100,10)
b
```

Out[157]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [161]:
```
# Now above 'b' values printed Based on Index :
# Here, The Positions we arranged to the above 'b' values :
# page 190

b [[2,3,1,3,4]]
```

Out[161]:
```
array([20, 30, 10, 30, 40])
```

10. Complex Numbers : Imaginary values... i (or) j :

page 190

In [166]:
```
# Complex Numbers - means, 'Where it includes imaginary values' - Which represented by 'i' (or) 'j' :
# These Particular Complex NUmbers very easily we can declare (or) work in python.
# No Other Language support with an a Complex Numbers.

# Complex data type - we can easily work it out. we can do an a 'SUMMATION'.
# And Whenever we call default data type, we will get an a 2.0,1.0,,, something like,
# it will 'Converted' to an a 'FLOAT DATA' :
```

In [167]:
```
# Here, it is printing '1.0' or something like...
# Means, it will Converted to 'float' data.
#

v = np.array([1 + 2j, 3 + 6j])
v
```

Out[167]:
```
array([1.+2.j, 3.+6.j])
```

In [168]:
```
print(v.dtype)  # above 'v' data type it will print :
```

```
complex128
```

In [169]:
```
# Before we have seen diagonal data also :

c = np.diag([1,2,3])
c
```

Out[169]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [179]:
```
# Now i'm declaring some data type :
# Here, Matrix 0, 1st row, 2nd row.
# page 191

d = np.array([[1,2,3],[4,5,6],[7,8,9]])
d
```

Out[179]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [180]:
```
d.ndim
```

Out[180]:
```
2
```

In [181]:
```
d.shape
```

Out[181]:
```
(3, 3)
```

In [182]:
```
print(d.dtype)
```

```
int32
```

11. Calling Objects Based On Random Index : (Matrix) NUMPY

page 192

In [186]:
```
# There will be a 'Step Size' in 'ARRANGE' :
# 'Random Numbers' always it will be 'No.of Objects'. How many Objects we required, that's it.
# In 'Random' - Calling Objects based on Index :
# We Can use 'Step' in 'Arrange' and 'Masking'
```

In [187]:
```
# Here, i want to 'fetch' an a particular value of above'd' values - sum-179 :
# Explanation in BLUE NOTE BOOK :
# To be Clearly - 2nd Row, 1st Position of 'd' values.

d[2,1]
```

Out[187]: 8

In [188]:
```
# Now again, 0 Row, 2nd Position of 'd' values :

d[0,2]
```

Out[188]: 3

In [189]:
```
# Now i'm calling 'd' :

d
```

Out[189]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

12. ASSIGNING NEW VALUES BASED ON INDEX :

PAGE 193

In [191]:
```
d[2,2]
```

Out[191]: 9

In [192]:
```
# Now the 2nd Row, 2nd Position - Index value '9' Will be Changed to '56' :

d [2,2] = 56
```

In [193]:
```
d
```

Out[193]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8, 56]])
```

In [195]:
```
f = np.arange(10)
f
```

Out[195]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

13. ASSIGNING in ARRANGE :

PAGE 194

In [196]:
```
f
```

Out[196]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [204]:
```
# Here, 'f' of ':(Colon)' i'm taking '5' and assigning as an a '10'

f [5:] = 10
```

In [205]:
```
f
```

Out[205]:
```
array([ 0,  1,  2,  3,  4, 10, 10, 10, 10, 10])
```

In [206]:
```
f [5:] == 10
```

Out[206]:
```
array([ True,  True,  True,  True,  True])
```

14. SIMPLE ARTHEMATIC OPERATIONS :

PAGE 194

In [207]:
```
# Here, 'Not Arange' - 'Array' We need to give :
# 'Arange' means, We need to give only 'Single Number'
```

In [208]:
```
# Here, for ([1,2,3,4]), i have given 'a + 1', So, it has been added and Changed to -> ([2,3,4,5])

a = np.array([1,2,3,4])
a + 1
```

Out[208]:
```
array([2, 3, 4, 5])
```

In [211]:
```
# Now i'm taking Simple Operation like :
# Here i required 4 Ones and i was added '+ 1' to each iteration.
# '4' Ones + 1 = 2.,2.,2.,2.

b = np.ones(4) + 1
b
```

Out[211]:
```
array([2., 2., 2., 2.])
```

15. SIMPLE MULTIPLICATION :

PAGE 195

In [219]:
```
# Here, print(c.dot(c)) also used as Multiplication :

c = np.diag([1,2,3,4])
print(c * c)
print("*****************")
print(c.dot(c))
```

```
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
*****************
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
```

16. SUMMATION ON MATRIX : BASIC OPERATIONS

PAGE 196

(A) VECTOR SUMMATION :

In [220]:
```
# Here,  1+2+3+4 = 10 ;

y = np.array([1,2,3,4])
np.sum(y)
```

Out[220]:
```
10
```

(B) 'x' is an a 'MATRIX' :

In [221]:
```
x = np.array([[1,1],[2,2]])
x
```

Out[221]:
```
array([[1, 1],
       [2, 2]])
```

(C) COLUMN BASED SUMMATION :

In [225]:
```
x
```

Out[225]:
```
array([[1, 1],
       [2, 2]])
```

In [229]:
```
# Here, The COLUMN BASED - means, 'x' values '1st column- 1+2', and '2nd column- 1+2', Total (3 by 3)
# axis = 0 means, COLUMN;
# axis = 1 means, ROW;

x.sum(axis = 0)
```

Out[229]:
```
array([3, 3])
```

(D) ROW BASED SUMMATION :

In [231]:
```
# Here, The ROW BASED - means, 'x' values '1st ROW- 1+1', and '2nd ROW- 2+2', Total (2 by 4)
# axis = 0 means, COLUMN;
# axis = 1 means, ROW;

x.sum(axis = 1)
```

Out[231]:
```
array([2, 4])
```

(E) x.min, x.max, x.argmax, x.argmin :

page 197

In [232]:
```
x = np.array([4,3,5,2])
```

In [233]:
```
x
```

Out[233]:
```
array([4, 3, 5, 2])
```

In [234]:
```
# x.min (minimum)

x.min()
```

Out[234]:
```
2
```

In [236]:
```
# x.max (maximum)
x.max()
```

Out[236]:
```
5
```

x.argmax : 'argmax' Represent the Value of Index :

In [244]:
```
# What is argmax () in Python?

# Argmax is an operation that finds the argument that gives the maximum value from a target function.
# Argmax is most commonly used in machine learning for finding the class -
# with the largest predicted probability
```

In [245]:
```
x.argmax()
```

Out[245]:
```
2
```

In [246]:
```
x.argmax( axis = None, out = None)
```

Out[246]: 2

x.argmin : 'argmin' Represent the Value of Index :

In [247]:
```
# Numpy argmin is a function in python which returns the index of the minimum element
# from a given array along the given axis.
# The function takes an array as the input and outputs the index of the minimum element.
# The input array can be a single-dimensional array as well as a multi-dimensional array.
```

In [248]:
```
x.argmin()
```

Out[248]: 3

In [249]:
```
x.argmin( axis = None, out = None)
```

Out[249]: 3

(F) x.mean, median, standard diviasion : statistics

In [251]:
```
x.mean()
```

Out[251]: 3.5

In [256]:
```
# Here, 'np.median()' is something different :

np.median(x)
```

Out[256]: 3.5

In [258]:
```
# STANDARD DIVIASION :

x.std()
```

Out[258]:
```
1.118033988749895
```

In [ ]:
```

```

In [259]:
```
# Here, Whatever they designed we are Utilizing and we are Learning :
# 'LOGICS' - Whatever we 'build' that depends on us.
# functions, methods, objects e.t.c and build the logics. That is in our hands.
```

17. LOADING THE DATA SET : READ THE DATA SET : NUMPY

PAGE 198

In [282]:
```
# Now this 'Dataset' actually, Whenever we 'Call' into an a 'NUMPY(np.)',
# It will automatically 'CONVERTED' to an a " numpy.matrix "

data = np.loadtxt("DataS/marks.txt")
data
```

Out[282]:
```
array([[  1.,  45., 100.,  76.,  88.,  65.],
       [  2.,  96.,  98.,  91.,  56.,  45.],
       [  3.,  78.,  97.,  82.,  88.,  83.],
       [  4.,  67.,  67.,  76.,  70.,  90.],
       [  5.,  90.,  79.,  78.,  90.,  67.]])
```

In [274]:
```
data.shape
```

Out[274]:
```
(5, 6)
```

In [275]:
```
data.ndim
```

Out[275]:
```
2
```

18. DUMP ALL DATA (ROW :, COLUMN :) :

page 199 most imp

Before ' , ' comma - Affected on the 'Rows' and

After ' , ' comma - Affected on the 'Columns'

Before ' : ' colon - Skip the 1st 'Rows' or 'Columns'

After ' : ' colon - Accept the 1st 'Rows' or 'Columns'

In [277]:
```
# popular is a left hand variable: also anything we can give :

popular = data[ : , : ]
popular
```

Out[277]:
```
array([[  1.,  45., 100.,  76.,  88.,  65.],
       [  2.,  96.,  98.,  91.,  56.,  45.],
       [  3.,  78.,  97.,  82.,  88.,  83.],
       [  4.,  67.,  67.,  76.,  70.,  90.],
       [  5.,  90.,  79.,  78.,  90.,  67.]])
```

19. After ' , ' After ' : ' -> Accept Number (OR) Accept 1st Columns

page 200

In [286]:
```
# Here, Accept first '2' 'Columns':

popular2 = data [ : , :2  ]
popular2
```

Out[286]:
```
array([[ 1., 45.],
       [ 2., 96.],
       [ 3., 78.],
       [ 4., 67.],
       [ 5., 90.]])
```

In [284]:
```
# Here, All 'Rows' and Only '3' 'columns' have been printed:

popular3 = data[ : , : 3 ]
popular3
```

Out[284]:
```
array([[  1.,  45., 100.],
       [  2.,  96.,  98.],
       [  3.,  78.,  97.],
       [  4.,  67.,  67.],
       [  5.,  90.,  79.]])
```

20. After ' , ' and Before ' : ' -> SKIP FIRST COLUMNS

In [292]:
```
# Here, Skip first 'columns' :

popular3 = data[ : , 3:  ]
popular3
```

Out[292]:
```
array([[76., 88., 65.],
       [91., 56., 45.],
       [82., 88., 83.],
       [76., 70., 90.],
       [78., 90., 67.]])
```

21. Before ' , ' and Before ' : ' Skip 1st 'Rows'

In [288]:
```
# Here, Skip first 'Rows' and  All 'columns' have been printed:

popular3 = data[ 2: , :  ]
popular3
```

Out[288]:
```
array([[ 3., 78., 97., 82., 88., 83.],
       [ 4., 67., 67., 76., 70., 90.],
       [ 5., 90., 79., 78., 90., 67.]])
```

22. Before ' , ' and After ' : ' Accept 1st 'Rows'

page 202

In [289]:
```
# Here, first 2 'Rows' Accepted and All 'columns' have been printed:

popular3 = data[ :2 , :  ]
popular3
```

Out[289]:
```
array([[  1.,  45., 100.,  76.,  88.,  65.],
       [  2.,  96.,  98.,  91.,  56.,  45.]])
```

In [329]:
```
# skip 1st row ok
# Accept 1st '4' rows ok
# skip 1st '2' cloumns ok
# Accept 1st '5' columns ok

popular7 = data [1 : 4, 2 : 5]
popular7
```

Out[329]:
```
array([[98., 91., 56.],
       [97., 82., 88.],
       [67., 76., 70.]])
```

In [298]:
```
# Directly with python also we can work it out with the Numeric Numbers :

popular
```

Out[298]:
```
array([[  1.,  45., 100.,  76.,  88.,  65.],
       [  2.,  96.,  98.,  91.,  56.,  45.],
       [  3.,  78.,  97.,  82.,  88.,  83.],
       [  4.,  67.,  67.,  76.,  70.,  90.],
       [  5.,  90.,  79.,  78.,  90.,  67.]])
```

23. CALICULATE EXECUTION TIME :

PAGE 203

In [299]:
```
# if we go indepth of our regular working environment :
# The Simple form of Numeric Numbers we use very less :
# We have used an a Complex Numbers 90%.
# We used an a Huge Highend of an a Complex Numbers - 'Huge Caliculations'
# At the 'TIME' Normal Python won't give any Proper Support.
# That's why We go for 'NUMPY'
# There is a Method Which is Called 'timeit( )'
# timeit( ) -> Helps us to 'Caliculate' how much time it is taking for the Execution.
# Now let us take an a Simple Caliculation.
```

(A) timeit( ) : In Normal Python :

page 205

In [302]:
```
# Space must between %timeit and [i ** 2 for i in L]
L = range(1000)
%timeit [ i ** 2 for i in L]
```

```
491 µs ± 6.07 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

(B) timeit( ) : In Numpy :

page 205

In [307]:
```
# The Speed of an a 'NUMPY' is more than 'Normal Python' :

a = np.arange(1000)
%timeit a ** 2
```

```
2.8 µs ± 66.7 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

24. ' 3D MATRIX ' :

PAGE 206

In [308]:
```
# We have seen, how the 3D Matrix look like, again We will build one particular 3D Matrix
```

In [311]:
```
x = np.array([[[1,2,3],[4,5,6]],[[0,-1,-2],[-9,-3,-2]]])
x
```

Out[311]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 0, -1, -2],
        [-9, -3, -2]]])
```

In [312]:
```
x.shape
```

Out[312]:
```
(2, 2, 3)
```

In [313]:
```
x.ndim
```

Out[313]: 3

In [314]:
```
# Now Print Everything of above sum : 311
```

In [315]:
```
print(x.shape)
print(x.ndim)
print(x.dtype)
print(x.size)
print(x.nbytes)
```

```
(2, 2, 3)
3
int32
12
48
```

25. TRANSPOSE :

PAGE 207

In [319]:
```
# And also we can do 'Transpose' - we have use Capital 'T'

# (a) Rows will become Columns,
# (b) Columns will become Rows.
# This is In-generally we see in the 'Matrix'
```

In [320]:
```
print(x.T)
```

```
[[[ 1  0]
  [ 4 -9]]

 [[ 2 -1]
  [ 5 -3]]

 [[ 3 -2]
  [ 6 -2]]]
```

In [321]:
```
x.shape
```

Out[321]:
```
(2, 2, 3)
```

26. Random Numbers : numpy(np.)

page 208

In [322]:
```
# Before In Random Numbers, We have seen how to generate the Random Numbers, means -
# Direct Declaration.   Now What we are going to see here means - We Can Declare the Parameters,
# like, What is the Lowest Value, Highest Value, and Size.
```

In [327]:
```
n = np.random.randint(low = 0, high = 9, size = 10)
n
```

Out[327]:
```
array([2, 6, 0, 0, 5, 4, 1, 2, 2, 4])
```

27. Generating RANDOM NUMBERS in MATRIX : NUMPY

PAGE 208

In [328]:
```
x = np.random.rand(3,3)
x
```

Out[328]:
```
array([[0.07223169, 0.55483923, 0.11273817],
       [0.71083652, 0.2010567 , 0.82683943],
       [0.41443282, 0.09626717, 0.67417266]])
```

In [ ]:
```

```