Notes

__/__/____

Numpy

Install Numpy:

```
# Install a pip package in the current Jupyter kernel.
! pip install numpy.

# Install a pip package by command line
  pip install numpy.

# python list :
lst = [1, 2, 3, 4, 5, 6]
```

- Import numpy : import numpy as np.

Create Numpy array :

```
arr = np.array( lst )

print ( type(arr) )          # <class 'numpy.ndarray'>
print ( arr )                # [1 2 3 4 5 6]
```

- Why Numpy ?
- Most powerful numerical processing library in python.
  Array oriented computing.
- provides extension package to python for multi dimensional array.
- very efficient.
- Scientific computation.

__/__/____

- properties of Numpy assay :
- Assay : ass
- Shape : ass.shape
- Data Type : ass.dtype
- Itemsize : ass.itemsize
- Dimensionality : ass.ndim

Functions for creating Numpy assay :

→ ass = np.asange (start, end, step)

→ ass = np.ones((3,3)) #
Rows ↑ ↑ colums.

$$\begin{bmatrix} 1. & 1. & 1. \\ 1. & 1. & 1. \\ 1. & 1. & 1. \end{bmatrix}$$

→ ass = np.zeso((3,3)) # $\begin{bmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$

→ ass = np.eye(3) #[[[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]]

(or)

ass = np.eye(3,2) #[[1. 0.]
[0. 1.]
[0. 0.]]

__/__/____

Datatypes in Numpy :-

Below is a list of all data types in Numpy and the characters used to represent them.

- i - integer
- b - boolean
- str - string
- f - float
- m - timedelta
- M - datetime
- O - Object
- u - unsigned integer
- c - complex float
- U - unicode string
- V - fixed chunk of memory for other type (void)

Example :

```
import numpy as np
lst = [1, 2, 3]

arr = np.array(lst, dtype='f')    # [1. 2. 3.]
arr = np.array(lst, dtype='O')    # [1 2 3]
arr = np.array(lst, dtype='str')  # ['1' '2' '3']
```

__/__/____

Numpy Array - Indexing, Slicing and updating :-

* Since data in numpy array is stored sequentially, It is possible to access the data with the help of Indexing and Slicing operation.
* Numpy array's are Mutable. i.e; Data stored in numpy array can be updated / changed.

Indexing :

Randomly generating 1-D array :

```
arr = np.random.randint(100, size=(5,))
arr          # [1 98 59 28 12]
```

Accessing single Index :

⇒ `arr[2]` # 59

Accessing Multiple Indexes :

⇒ `arr[[2, 4]]` # [59, 12]

Randomly generated 2-D array :

```
arr = np.random.randint(100, size=(5,4))
arr
```

#
```
[[43 32  9 42]
 [39 51 78 75]
 [49 59 87 69]
 [11 78 46 44]
 [40 43 20 42]]
```

__/__/____

Accessing 2nd index :

```
arr[2]          # [49 59 87 69]
```

Accessing (2,1) index :

Syntax-1 `arr[2][1]` # [59]

Syntax-2 `arr[2, 1]` # [59]

Row Index — column Index

Accessing Multiple values :

```
arr[[2,1],[1,1]]          # [59 51]
```

Row Indexes — column Indexes

```
arr[[1,3],[1,2]]          # [51,46]
```

51 (1,1)

46 (3,2)

Indexing with Boolean Arrays :

```
arr = np.random.randint(100, size=(6,))
arr             # [1 80 68 22 54 68]

idx = [True, False, True, False, False, True]
arr[idx]
# [1 68 68]
```

__/__/____

Slicing:

Eg:
```
arr = np.random.randint(100, size=(10,))
arr   # [6 40 86 3 50 33 80 18 84 47]

arr[1:4]   # [40 86 3]
arr[0:-4]  # [6 40 86 3 50 33]
arr[::2]   # [6 86 50 80 84]
```

Eg:
```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr   # [[1 2 3]
      #  [4 5 6]
      #  [7 8 9]]
```

Remember Syntax-2 for accessing the data in Numpy array:

```
arr[:2, 1:3]   # [[2,3],
               #  [5,6]]
```

Row Indexes from 0 to 2. (2 not included upto 1 only)

Column Indexes from 1 to 3 (upto 2 only 3 not included)

```
arr[1:2, :]   #[[4,5,6]]
```

__/__/____

Updating Data in Numpy Array :

```
arr = np.random.randint (100, size = (5,4)).
print ('Original Array: \n', arr)

arr [1, 1] = 99
print ('updated Array:\n', arr )

# Original Array :
  [[94  7  65  88]
   [23  72  58  57]
   [48  20  33  52]
   [2   90  5   63]]
```

updated Array :

```
[[94  7  65  88]
 [23  99  58  57]
 [48  20  30  52]
 [2   90  5   63]]
```

Numpy Flatten and Ravel :

Example :

```
arr = np.random.randint (10,40, size =(2,2))
arr
# [[12  19]
   [34  27]]
flatten_array = arr.flatten( )  # [12  19  34  27]

ravel_array = arr.ravel( )  # [12  19  34  27]
```

__/__/____

Note: flatten() and ravel() look's like both are performing flatten operation on the array data.

⇒ Is there any difference blw flatten() & ravel():

update value in the array.

arr[1, 1] = 15.

Original Array → [[12 19]
[34 27]]

updated Array → [[12 19]
[34 15]]

flatten_array → [12 19 34 27]
ravel_array → [12 19 34 15]

Note: * The main difference blw flatten and ravel is the flatten creates the copy of the original array and flattens to 1-D array, where as, ravel creates a view of the Original array and changes to 1-D array.

* Whenever we perform update on original Array ravel will take care of updates, where as flatten doesn't.

__/__/____

Numpy Reshape :

```
arr = np.random.randint(10,40, size=(3,2))
arr
# [[15  21]
   [11  39]
   [27  32]]

arr_reshaped = arr.reshape(2,3)
arr_reshaped

# [[15  21  11]
   [39  27  32]]
```

Iterating over Numpy Array :

Iterating over a 1-D Array :

```
arr = np.random.randint(10, 40, size=(3,0))
arr  # [15, 27 35]

# Looping over items in the array.

for item in arr :
    print(item, end=' ')

# 15 27 35.
```

__/__/____

Iterating over a 2-D array :

```
arr = np.random.randint(10, 78, size = (3,2))
arr
# [[12 19]
   [65 70]
   [51 43]]
```

Looping over rows in the 2-D array :

```
for row in arr:
    print(row)          # [12 19]
                        [65 70]
                        [51 43]
```

Looping over items in the 2-D array :

```
for item in arr.ravel():
    print(item, end=' ')    # 12 19 65 70 51 43
```

Iterating using np.nditer( ) :

```
for item in np.nditer(arr):
    print(item, end=' ')
                        # 12 19 65 70 51 43.
```

using Ellipsis :

```
for item in np.nditer(arr, op_flags=['readwrite']):
    if item > 20:
        item[...] = (item*0)
arr
    # [[12, 19],
       [0, 0],
       [0, 0]]
```

__/__/____

Python Operators on Numpy array :

```
x = np.array([[1,2,3],[4,5,6]])
x    # [[1 2 3]
       [4 5 6]]

x+5      #[[6 7 8]
          [9 10 11]]

x%2      #[[1 0 1]
          [0 1 0]]

x%2==0   # [[False True False]
            [True False True]]
```

Numpy Maths :

- Square Root — np.sqrt(4)
- Exponent — np.exp( )
- Trigonometric sin — np.sin( )
- Trigonometric cos — np.cos( )

- Elementwise operations :

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

- Addition : np.add(x,y) # [[6 8]
  [10 12]]

- Subtraction : np.subtract(x,y) # [[-4 -4]
  [-4 -4]]

__/__/__

- Multiply : np.multiply(x, y) # [[5 12]
  [21 32]]

- Division : np.divide(x, y) # [[0.2 0.33333333]
  [0.42857143 0.5]]

- Matrix Multiplication :

way-1 : np.matmul(x, y)

way-2 : np.dot(x, y)

way-3 : x @ y

[[19 22]
[43 50]]

Diagonal Elements :

```
X = np.array([[1,2,3],[4,5,6]])
X                    # [[1 2 3]
                       [4 5 6]]
np.diag(X)           # [1 5]
```

Transpose of an array :

```
X = np.array([[1,2,3],[4,5,6]])
X                    # [[1 2 3]
                       [4 5 6]]
```

- Transpose : X.T # [[1 4]
[2 5]
[3 6]]

__/__/____

Numpy Statistics :

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

```
x      # [[1 2]
         [3 4]]
```

```
np.sum(x)          # 10
np.sum(x, axis=0)  # Columnwise : [4 6]
np.sum(x, axis=1)  # Rowwise    : [3 7]
```

Example → `x = np.array([[1,2,3],[4,5,6]])`

```
x      # [[1 2 3]
         [4 5 6]]
```

```
np.min(x)          # 1
np.min(x, axis=0)  # Columnwise : [1 2 3]
np.max(x, axis=1)  # Rowwise    : [1 4]
```

- Minimum : `np.min()`
- Maximum : `np.max()`
- Mean : `np.mean()`
- Median : `np.median()`
- Variance : `np.var()`
- Std Dev : `np.std()`

__/__/____

Miscellaneous Topics :

- Linspace :

Syntax → np.linspace (begin, end, No. of. elements)

Example : np.linspace (1, 5, 9)

[1. 1.5 2. 2.5 3. 3.5 4. 4.5 5.]

- Sorting :

syntax → np.sort ( )

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

y # [[7 4 3]
[1 9 8]
[2 5 6]]

np.sort (y) # [[3 4 7]
(OR) [1 8 9]
np.sort (y, axis = 1) [2 5 6]]

Note : By default sorting in ascending order and rowwise.

np.sort (y, axis = 0) # [[1 4 3]
[2 5 6]
[7 9 8]]

__/__/____

Stacking :

Example → arr_1 = np.arange(5,15).reshape(2,5)

arr_1

```
# [[5 6 7 8 9]
   [10 11 12 13 14]]
```

arr_2 = np.arange(25,35).reshape(2,5)

arr_2

```
# [[25 26 27 28 29]
   [30 31 32 33 34]]
```

np.vstack([arr_1, arr_2])

```
# [[5, 6, 7, 8, 9],
   [10, 11, 12, 11, 12],
   [25, 26, 27, 28, 29],
   [30, 31, 32, 33, 34]]
```

np.hstack([arr_1, arr_2])

```
# [[5, 6, 7, 8, 9, 25, 26, 27, 28, 29],
   [10, 11, 12, 13, 14, 30, 31, 32, 33, 34]]
```

Concatenate :

np.concatenate([arr_1, arr_2], axis=0) # concatenating (vertical stacking)

```
# [[5, 6, 7, 8, 9],
   [10, 11, 12, 13, 14],
   [25, 26, 27, 28, 29],
   [30, 31, 32, 33, 34]]
```

np.concatenate([arr_1, arr_2], axis=1)

```
# [[5, 6, 7, 8, 9, 25, 26, 27, 28, 29],
   [10, 11, 12, 13, 14, 30, 31, 32, 33, 34]]
```

__/__/____

Append :

```
np.append(arr-1, arr-2, axis=0)      # Appending - Vertical stacking

#[[5, 6, 7, 8, 9],
  [10, 11, 12, 13, 14],
  [25, 26, 27, 28, 29],
  [30, 31, 32, 33, 34]]

np.append(arr-1, arr-2, axis=1)      # Appending - Horizontal stacking.

#[[5, 6, 7, 8, 9, 25, 26, 27, 28, 29],
  [10, 11, 13, 14, 30, 31, 32, 33, 34 ]]
```

Where - process Array Elements conditionally :-

```
numpy.where(condition, [x, y])   # where True, yield x, otherwise y
                                 # values to choose from.
```

Eg:

```
arr = np.array([1,2,3],[4,5,6])
arr
      #[[1 2 3]
        [4 5 6]]

np.where(arr%2==0, 'Even', 'odd')

#[['odd Even odd']
  ['Even odd Even']]
```

__/__/____

Argsort :

```
arr = np.array([10, -5, 6, -1])
```

Indexes : `arr.argsort()` # [1 3 2 0]

Sorted Array : `arr[arr.argsort()]` # [-5 -1 6 10].

Numpy Broadcasting :

* Start matching the dimensions backward (Right to Left)
  - Compatiable - If same numbers appears (or) If one of them is 1
  - Incompatiable - Otherwise.

Examples :

1	2	3			-1	0	1			-1	0	3
4	5	6	*		-1	0	1		=	-4	0	6
7	8	9			-1	0	1			-7	0	9
(3,3)					(3,) (or) (1,3)					(3,3)		

1	2	3			3	3	3			.3	.7	1.
4	5	6	/		6	6	6		=	.6	.8	1.
7	8	9			9	9	9			.8	.9	1.
(3,3)					(3,1)					(3,3)		

1	2	3			1	1	1			1	2	3
1	2	3	*		2	2	2		=	2	4	6
1	2	3			3	3	3			3	6	9
(3,)	(1,3)				(3,1)	(3,)				(3,3)		

__/__/____

Reading CSV into Numpy Array :

```
import numpy as np
#using loadtxt()

arr = np.loadtxt('data/nyc-weather.csv', delimiter=",",
                                         dtype='str')
print(arr)

#[['EST' 'Temperature' 'DewPoint' 'Humidity']
 ['1/1/2016' '38' '23' '52']
 ['1/2/2016' '36' '18' '46']
 ['1/3/2016' '40' '21' '47']
 ['1/4/2016' '25' '9' '44']
 ['1/5/2016' '20' '-3' '41']]
```

path