DATE :

Note:-

→ To know the present working directory (pwd)

→ To change directory cd (path)

V23:-

python for datascience

Numpy:-

→ It stands for Numerical python.
→ used for mathematical ops.
→ It can create upto 0d, 1d, 2d, 3d - - - - - nd arrays.

Ex:-
```
a = 94                          #0D array.
temp = np.array(a)
print(type(temp)) # class <array>
print(temp.ndim) # 0
Print(temp.shape) # ()
```

Ex:-
```
a = [1, 2, 3, 4]
temp = np.array(a)
print(type(temp))   # class <array>
print(temp.ndim)    # 1
print(temp.shape)   # (4,) } → no.of items in tuple is
                       ↓
                    no.of items
```

DATE :

Note:- shape

(1) 0 dim array → ( )
(2) 1 dim array → (no. of items,)
(3) 2 dim array → (no. of rows, no. of cols)

*** functions for creating 1D array:-**

(1) arange():-

→ arange() is a array-valued version of the built in python range function.

→ Syntax:- ~~np.array~~ np.arange(start, end, step)

Ex:-
```
a = np.arange(1, 10, 2)
print(a) # [1, 3, 5, 7, 9]
```

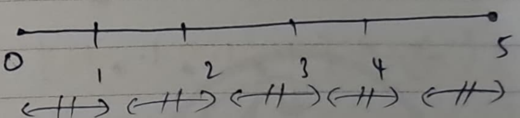

DATE :

(2) linspace():-

→ Returns evenly spaced numbers over a specified interval.

→ Syntax:- np.linspace(start, end, no_of_samples)

Ex:- np.linspace(0, 5, 6) # [0., 1., 2., 3., 4., 5.]



Note:- start, end both are inclusive.

(3) zeroes:-

→ Generates arrays of zeroes.
→ Syntax:- np.zeroes(shape)

Ex:-
```
a = np.zeroes((4,)) # (4,) → 1D array with '4' items.
print(a) # [0.0, 0.0, 0.0, 0.0]
```
→ [0, 0, 0, 0]

Note:- by default datatype = float, so if we want int →

DATE :

```
a = np.zeroes((4,), dtype=int)
print(a) # [0 0 0 0]
```

④ **Ones:-**

→ Here it all values filled with ones.
→ Syntax:- np.ones(shape)

Ex:- a = np.ones((4,))
print(a) # [1., 1., 1., 1.]

⑤ **random.randint:-**

→ Return random integers from start(inclusive) and end (exclusive)

→ Syntax:-
np.random.randint(start, end, noof samples)

Ex:- a = np.random.randint(1, 1000, 5)
print(a) # [294 803 690 292 901]
↓
type # class <array>

DATE :

*** Accessing values from array:-**

```
       -5   -4   -3   -2   -1
Ex:- l = [30, 40, 50, 60, 70]
          0    1    2    3    4
a = np.array(l)
print(a) # [30 40 50 60 70]
print(a[0]) # 30
print(a[0:4:2]) # [30 50]
```

Note:-
→ indexing and slicing operations are same as list and as well as same in '1D' array.

→ indexing and slicing can't be applied to '0d' array.

→ Accessing a multiple values based on condition (Mask indexing or Fancy indexing):-

Syntax:- array[condition]

→ first it will check the ~~item which~~ condition, the item which satisfies the condition that item will get selected.

Ex:- a = np.array([45, 46, 47])
print(a[a>45]) # [46 47]

DATE :

-> insertion. l_1=np.append(l_1, 900) -> np.where(l_1==9..
-> deletion (l_1=np.delete(l_1, index) } del l_1 } To del from memory.
-> updation.
-> Arithemetic ops.

-> np.array_equal(arr1, arr2)

checks if both numpy arrays contains same ele or not.

√24

* 2D array :- (list of 1D arrays ☺)

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

so, 'arr' is a (2x3) array.

-> To fill a values as 'Nan' then —

arr = np.array([[1, 2, np.nan],
[4, np.nan, 5]])

so, [[1 2 nan]
[4 nan 5]]

DATE :

-> To Create a array of (3x3) with value '4' in every item then—

arr => np.zeroes((3,3), dtype=int)+4

(or)

=> np.ones((3,3), dtype=int)+3

-> To create identity matrix (generally it is a square matrix) -> (diagonals will be '1')

i.e $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$ (2x2), $\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$ (3x3)

so, to create above identity matrix =>

arr1 => np.eye(2,2)
arr2 => np.eye(3,3)

-> Now to have our own values in diagonal then—

arr => np.diag([1, 2, 4]) => $\begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 4\end{bmatrix}$

sabar
Since 1975 Pumps, Motors & Cables

DATE :

* indexing in 2D array:-

-> To extract a single value from 2D.

Syntax:-

arr_name[row_index, col_index]

Ex:-
$$\begin{array}{c|c} & \begin{matrix} 0 & 1 \end{matrix} \\ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \end{array}$$

to get '4' so position is (1, 1)

so,

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

-> To extract a single row-

```
print(arr[1]) # [3,4]
```

-> To extract multiple rows-

```
print(arr[0:3]) # row 0,1,2 returned.
```

-> To replace a value -

```
a[2,2] = 500
```

Sabarplast
Pipes & Fittings

DATE :

-> To replace a entire row -

```
a[1] = [4,6]
```

* flatening:- Converting ND array to 1D array

We use '.ravel()' method

i.e

Consider a 2D array

$$\begin{bmatrix} 3, & 4, & 5 \\ 9, & 6 & 8 \\ 10, & 9, & 6 \end{bmatrix} \xrightarrow[\text{into 1D}]{\text{to convert}} [3,4,5,9,6,8,10,9,6]$$

So,

```
arr1 => np.array([[3,4,5],[9,6,8],[10,9,6]])
arr2 => arr1.ravel()
print(arr2) # [3,4,5,9,6,8,10,9,6]
```

* To convert rows <=> cols & vice versa Then it is called Transpose matrix.

i.e

a => a.T

$$\underset{a}{\begin{bmatrix} 3, & 4, & 5 \\ 1, & 4 & 3 \end{bmatrix}} \longleftrightarrow \underset{a}{\begin{bmatrix} 3 & 1 \\ 4 & 2 \\ 5 & 3 \end{bmatrix}}$$

* In 2D arrays, we have two ways of sorting.

1. Row wise sorting
2. Column wise sorting.

Ex:- Consider a matrix 'A'

$$\begin{bmatrix} 4, & 3, & 1 \\ 9, & 6, & 4 \\ 10, & 12 & 1 \end{bmatrix} \xrightarrow{\text{Row wise sort}} \begin{bmatrix} 1 & 3 & 4 \\ 4 & 6 & 9 \\ 1 & 10 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 4, & 3, & 1 \\ 9, & 6, & 4 \\ 10, & 12, & 1 \end{bmatrix} \xrightarrow{\text{colwise sort}} \begin{bmatrix} 4 & 3 & 1 \\ 9 & 6 & 1 \\ 10 & 12 & 4 \end{bmatrix}$$

So,

```
arr1 => np.array([[4,3,1],[9,6,4],[10,12,1]])
arr1 => np.sort(arr1, axis=1) # Row wise sort
arr1 => np.sort(arr1, axis=0) # colwise sort.
```


*Arthemetic ops:-

→ Addition:-

arr1 + arr2

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 4 & 5 \\ 6 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 5 & 8 \\ 10 & 9 \end{bmatrix}$$

→ multiplication:- Two ways

1. dot product
2. matrix multiplication.

1:- $\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 4 & 5 \\ 6 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 15 \\ 24 & 20 \end{bmatrix}$

print(a*b)

2:- $\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 4 & 5 \\ 6 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1\times4+3\times6 & 1\times5+3\times4 \\ 4\times4+5\times6 & 4\times5+5\times4 \end{bmatrix}$

c = np.matmul(a, b)

DATE :

* Concatenation :-

We can concatenate two arrays in two ways:-
1. vestically
2. horizantally.

1. vestically :-

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 6 & 7 \\ 9 & 8 \end{bmatrix}$$

$$\Rightarrow C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & 7 \\ 9 & 8 \end{bmatrix}$$

Ex:- arr1 => np.array([[1,2],[3,4]])
arr2 => np.array([[6,7],[9,8]])
arr3 => np.vstack((arr1, arr2))

vice versa
to do horizantal ([A][B] => [A B])
do,
arr4 => np.hstack((arr1, arr2))