Numpy:-

numpy:- (numerical python)

A python library to perform fast operations on large data.

why numpy over list:?

* faster - contiguous memory
* memory efficient
* convenience
* multi dimensional arrays
* Rich mathematical and statistical functions

* creating of arrays:-

```
import numpy as np   } module importer
n = np.array([3,4,5,6])   // one dimensional
print(n)
n1 = np.array([[3,4,2,1],[1,2,3,4]])   // two dimensional
print(n1)
```

op:-

```
[3 4 5 6]    // one D
[[3 4 2 1]   // TWO D
 [1 2 3 4]]
```

* creating array with only zeros:-

```
import numpy as py
n1 = np.zeros([3, 5])
print(n1)

n2 = np.zeros([3,5], dtype='int32')
print(n2)
```

column
row

OP:-(n1)

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

OP:-(n2)

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

* creating a array with ones:- (3 dimensional array)

```
n1 = np.ones([2,3,3], dtype='int32')
print(n1)
```

row
column
faces.

OP:-(n1

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

* creating a empty array:-

```
n5 = np.empty([1,3])
print(n5)
```

row, column

op:-

```
[[1.0432  0.00e
  =4.9406]]
```

Junck values

* create array with range of values:- [np.arrange(ST, SP, step)]

```
n6 = np.arange(10)
print(n6)
```

excluded

op:-

```
[0 1 2 3 4 5 6 7 8
 9 ]
```

* creating linearly spaced entries:-

```
n7 = np.linspace(0, 20, num = 5)
print(n7)
```

included

```
n8 = np.linspace(0, 20, num = 8)
print(n8)
```

included

op:- (n7)

```
[0.  5.  10.  15.  20.]
```

op:- (n8)

```
[0.  2.85714 2  5.7142
 8.57142  11.42857  14....
 17....   20.... ]
```

* creating array with same value:-

```
n9 = np.full([2,5],99)
print(n9)
```

common value.

op:-

```
[[99 99 99 99 99
  99 99 99 99 99]]
```

* filling value to the empty array:-

```
n11 = np.empty([2,2])
n11.fill(9)
print(n11)
```

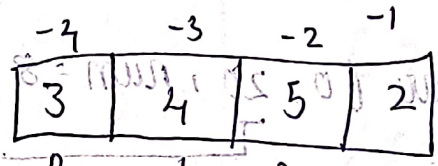
OP:-
[[9 9]
 [9 9]]

* Accessing arrays:- [through index]

ex:-

```
import numpy as py
n1 = py.array([3,4,5,2])
print(n1[0])
print(n1[-1])
```

OP:-
n1[0] 3
n1[-1] 2



// one dimensional

also, n1=

	0	1	2
0	8	4	3
1	1	2	3

// two dimensional

ex:- n1[0,1] 4

row → row
column → column

-) slicing:-

n1[1,:]

n1[:,1]

* updating of array :-

n1 :- [[[1 2 3]
[3 2 1]]
[[4 5 6]
[6 5 4]]]

n1[1, 0, 0] = 10

print(n1):

print(n1.ndim) = (find dimension of array)

op :- 3 = // 3 dimension array

[[[1 2 3]
[3 2 1]]] → change, 1st face, 0th row, 0th column

[[[10 5 6]
[6 5 4]]]

also print(n1.shape)

op :- (2, 2, 3)

print(n1.size)

op :- 12 → total elements

print(n1.dtype) → Int32 → structure of the array

print(n1.itemsize) → 4 → datatype of element

print(n1.nbytes) → 72 → size of one bytes

→ Total space occupied

* Sorting of array:-

```
n10 = np.array([3,4,1,9,5,10,8,7])
np.sort(n10)
```

op:-

```
array([1,3,4,5,7,8,9,10])
```

saved in n10,

-> If for 2 dimensional:-

```
n11 = np.array([[6,2,7],[5,10,1]])
np.sort(n11)
```

each row is sorted. [default axis=1]

op:-

```
array([[2,6,7],
       [1,5,10]])
```

= n11

-> for column sort:-

```
np.sort(n11, axis=0)
```

└> for column,

op:-

```
array([[5,2,1],
       [6,10,7]])
```

* reshaping and array conversions

ex:-

```
n12 = n10.reshape(4,2)
print(n12)
```

n10 :- [contain 8 elem]

[3, 4, 1, 9, 5, 10, 8, 7]

↓ 8 elements

```
[[ 3  4]
 [ 1  9]
 [ 5 10]
 [ 8  7]]
```

-) Joining of array using concatenate.

```
n13 = np.arange(4)
print(n13)

n14 = np.arrange(5, 8, 9)
print(n14)

n15 = np.concatenate((n13, n14))
print(n15)
```

n13

[0 1 2 3]

n14

[5 6 7 8]

n15

[0 1 2 3 5 6 7 8]

-) converting 1d array to 2d array

```
n16 = np.arange(8)
print(n16)

n17 = n16[np.newaxis, :]
print(n17)
print(n17.shape())
```

[0 1 2 3 4 5 6 7]

[[0 1 2 3 4 5 6 7]]

(1, 8)

↓ row └→ column

```
n18 = n16[:, np.newaxis]
print(n18)
print(n18.shape)
```

```
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
(8, 1)
```

(row, column)

* Indexing & slicing :-

```
n21 = np.arange(10)
print(n21)

print(n21[n21<5])   // [0 1 2 3 4]
```

↳ element less than five

```
print(n21[(n21>3) & (n21<7)])   // [4 5 6]
```

↓

combination of conditions,

also:

```
five_up = n21 > 5
print(five_up)
print(n21[five_up])
```

---

op:- [False False False False False False True True True True] // five_up

[6 7 8 9] // n21[five_up]

print (np. nonzero (n21 > 5))

op:- (array ([6, 7, 8, 9], dtype = int64),)

↓
indices.

also,

print (n21[3:7]) ← not included

[3 4 5 100]

---

* stack, split:

n22 = np. array ([[1, 2], [3, 4]])

n23 = np. array ([[5, 6], [7, 8]])

array np. hstack ((n22, n23))
↳ horizontal stack

op: array ([[1, 2, 5, 6]
[3, 4, 7, 8]])

np. vstack ((n22, n23))
↳ vertical stack.

op:- array ([[1 2],
[3, 4],
[5, 6],
[7, 8]])

```
n24 = np.arange(20)
print(n24)
np.hsplit(n24, 4)
```

(which array, split into 4 array..)

hsplit → horizontal split,

OP:-

```
[0 1 2 3 4 5 6 7 8 9 ... 19]

[array([0, 1, 2, 3, 4]),
 array([5, 6, 7, 8, 9]),
 array([10, 11, 12, 13, 14]),
 array([15, 16, 17, 18, 19])]
```

---

* copying (array):-

```
n25 = n24  # shallow copy
```

if we change one array,
then, it reflect to other because it copy only reference address.

deep copy:-

```
n26 = n24.copy()
n24[1] = 200
print(n24)
print(n26)
```

```
[100 200 2 3 4 5 ..... 19] //n24
[100 1 2 3 4 5 .... 19] //n26
```

* math operations:-

```
n22 = [[1 2]
       [3 4]]

n23 = [[5 6]
       [7 8]]
```

print (n22 + n23)	[[6 8] [10 12]]
print (n23 * n23)	[[25 36] [49 64]]

also,

```
n26 = [100 200 2 3 4 .. 19]
```

print(n26.sum())	489
print(n26.max())	200
print(n26.min())	2

---

* Random numbers:

```
np.random.randn(3,3)
```
(3 = row, 3 = column)

OP:- array([[0.... , 0.5... , 0.6]
[0.48 , 0.25 , 0.85]
[0.65 , 0.58 , 0.57]])

```
np.random.randint(200, size=(4,2))
```
(200 = not inclusive; 4 = row, 2 = column)

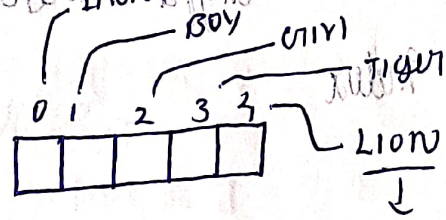
OP:- array([[1 , 8],
[7 , 11],
[14 , 8],
[0 , 9]])

PANDAS
[Panel data -or- python data analysis]

* Two data structures:-
  - → pandas series : // one dimensional array
    - ↳ contain same data type
      - or
      - different data type
      - or
      - different data structures.

note:- we can customize the index in pandas series.

example:-



↳ This is known as "name the labels" ← "name the index"

- → pandas data frames:- // two Dimensional arrays.

same we can label the index like we did in ~~Auto~~ pandas series",

* " pip install pandas "
  - ↳ command prompt.
    - to download pandas Library.

* creating of series:-

```
import pandas as pd
L = ['IAM', 'The', 'Best']
L_series = pd.series(L)
print(L_series)
```

OP:-

```
0   IAM
1   The
2   Best
dtype: object
```

index

```
L_series = pd.series(L, index=['1ST', '2nd', '3rd'])
print(L_series)
```

OP:-

```
1ST   IAM
2nd   The
3rd   Best
dtype : object
```

labeling index

also,

```
r_dict = {'red': 'tomato', 'grun': 'kovakkai'}
r_series = pd.series(r_dict)
```

OP:-

```
red    tomato
grun   kovakkai
dtype : object
```

[index labelled by dictionary]

* inserting elements in series :-

ex:- v_dict = {'red' : 'tomato', 'grun' : 'kovakkai'}
v_series = pd.series(v_dict)
v_series['orangi'] = 'carrot'
print(v_series)

op:-

red tomato
grun kovakkai
orangi carrot
dtype : object.

* deleting elements from series :-

v_series.drop(['orangi'])

red tomato
grun kovakkai
dtype : object

↓
it delete orangi index & value in only v_series.

also,

v_series.drop(['orangi'], inplace = True)

↓
by using this it delete in v_series and as well as v_dict

Note:-

series is an a heterogenous because it allow elements of different data type and data structures.

* creating of data frame:-

```
import pandas as pd
data = { 'vegetables' : ['beetroot', 'carrot', 'tomato', 'cowpea'],
         'color' : ['purple', 'orange', 'tomato', 'korakkai'],
         'unit' : ['1kg', '1kg', '1kg', '1kg', '1piece'],
         'min' : [15, 25, 53, 'seventeen'],
         'max' : [40, 25, 90, 45], }

data_fr = pd.dataframe(data)
```

op:-

	vegetables	color	unit	min	max
0	beetroot	purple	1kg	15	40
1	carrot	orange	1kg	25	45
2	tomato	tomato	1kg	35	90
3	korakkai	korakkai	1kg	seventeen	45

-> data_fr : columns

op:-

Index(['vegetables', 'color', 'unit', 'min', 'max'], dtype='object')

-> data_fr.index

op:-

RangeIndex(start=0, stop=5, step=1)

if we want to change Index values,

then,

```
data_fr = pd.DataFrame(data, Index=
          ['a', 'b', 'c', 'd', 'e'])
```

so,

[0 1 2 3] change to [a b c d]

* accessing through columns:-

```
data_Fr['unit']
```

OP:-

a	1kg
b	1kg
c	1kg

Name: unit, dtype = object

* accessing row wise:-

```
data_fr.loc['a']
```

OP:-

vegetable	beetroot
color	purple
unit	1kg
max	15
min	40

Name: a, dtype = object

* accessing by both column & row:-

```
data_fr.loc['a', 'unit']
```

(row, columns)

OP:-

'1kg'

* Slicing in data frames:-

data_fr.iloc[0:3, 0:3]

rows [from 0 -> 2]

columns [from 0 -> 2]

op:-

	vegitables	color	unit
a	beetroot	purple	1kg
b	carrot	orange	1kg
c	tomato	tomato	1kg

* adding new rows:-

```
data_fr.loc['f'] = {'vegitables' : 'possinakai',
                    'color' : 'yellow',
                    'unit' : 'unieco',
                    'min' : 10,
                    'max' : 30}
```

new row will add :- in the index of 'f'

* adding new columns:-

data_fr['average'] = [20, 35, 75, 85]

new, columns will added in the name of 'average'

* removing of columns or rows

```
data_fr.drop(['feature'], axis=0)
```

axis = 0 → row (~~column~~)
axis = 1 → column
default

```
data_fr.drop(['average'], axis=1)
```

* finding a non-value or missing value in the data frame:-

```
data_fr.isnull()
```

	column1	column2	column3
a	false	false	false
b	false	false	false
c	false	false	false

```
data_fr.isnull().sum()
```

column1	0
column2	0
column3	0

```
data_fr['season'] = [True, True, False, False, np.nan]
```

np.nan → pandas.nan (non-value)

So,

```
data_fr.isnull().sum()
```

OP:-

column1	0
column2	0
season	1 → one non.value

* filling a common value in the place of nan (non-value) :-

data_fr.fillna(false)

↓

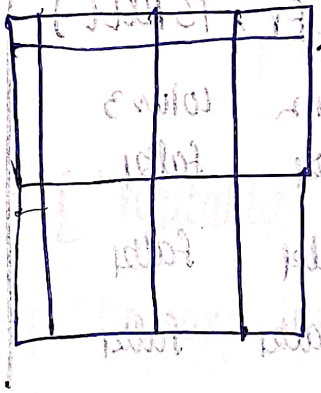
replaced in every non-value.

* adding two data frame:- [default axis=0] ↳ row join

Pd.concat([data_fr1, data_fr2])

So

OP:-



Pd.concat([data_fr1, data_fr2], axis=1)

↓

column join

OP:-