(7-2-24)

DATE :

V6 :- 22/02/24

deleting a Variable :-

Using Keyword 'del'

```
a=55          a (55)
del a          ↓
print()#error.  x -> deleted.
```

del a,b,c #to del multiples variables.

V7 :-

id( ) => gives the address

type( ) => gives the type of object.

To create str => ' ' (or) " " (or) ''' '''

To multiline str => ''' ''' (or) " " " " " "

Ex: s= """ pavan
aditha """

DATE :

```
name = "pavan"
age = 22.
print("my name is:", name, "age:", age);
print(f"my name is :{name} age is: {age}")
print("my name is :{} age is {}".format(name, age));
```

all three output is :-

my name is pavan age is: 22 //.

Note: int("12.67") X (:' is considered as char)

float("12.67") ✓

V8 :-

indexing and slicing.

V9 :-

DATE :

interview ②

l = [49, 60, [2, 3, "hello", 3], 4, 6]
① ② ③ ④

Now, get the "hello"

so,

⇒ l[2][2]

To get 'e' in hello ⇒ l[2][2][1]

list concatenation ⇒ $l_1 = [1,2,3]$

$l_2 = [4]$

$l = l_1 + l_2 \Rightarrow [1,2,3,4]$

both should be the list type.

③ Diff b/w method / functions?

method:- It is a functions which will be applied on particular datatype.

functions:- It is set of instructions used to perform particular task and will be applied to all datatypes.

Ex:- len(), type()

DATE :

③ what is the dif b/w append Vs extend.

append:- add only one item to the end.

extend:- add at the end but multiple items

Ex:- $l_1 = [1,2,3]$

$l_2 = [4,5,6]$

$l = l_1.append(l_2)$ | $l = l_1.extend(l_2)$

[1,2,3,[4,5,6]] | # [1,2,3,4,5,6]

has one item | has multiple items

(extend() == concatenation)

So, $l_1.extend(l_2)$ is equal to $l_1 + l_2$.

DATE :

V10 :-

→ .insert(index, ele) → ele replace avvadhu just 1 step forward ithadhi!

→ .clear() ⇒ will delete all items in a object. but object itself will not be removed.

→ del l ⇒ Now same as clear but object will also get deleted.

→ .count(item) ⇒ returns how many times the item got repeated.

→ .pop() ⇒ last item will deleted.
.pop(i) ⇒ index 'i' item will be removed.

→ .reverse()

ex:- l=[1,1,[3,4,5]]
l.reverse() # [[3,4,5],2,1]
but, i want only nested element to be reversed
i.e [1,2,[5,4,3]]

So,
l[-1].reverse();
(or)
l[2].reverse();

DATE :

ex:- count '5' in list?

l=[1,1,1,2,[2,3,4,5,5],6,6,7,8,7,66,7]

c=l.count(5)
l[4].count(5);

→ .remove(item) → Throws error if not found!
removes the specified item. only first occurring ele from left

→ .sort() # homogeneous type should only be allowed.

→ .sort(reverse=True) # desc order.

→ diff b/w .sort() & sorted(l)

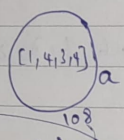
.sort()	sorted(l)
→ It is a method	→ It is a function.
→ Automatically stores in called variable	→ Not stores in a original var but we need to store in other var.
ex:- l=[1,2,4,3] l.sort(l) print(l) [1,2,3,4]	sorted(l) → print(l) # [1,2,4,3] so, temp=sorted(l) ↳ [1,2,3,4].

DATE :

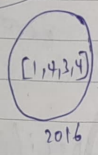
Shallow copy

(an entirely new copy will be created using .copy())

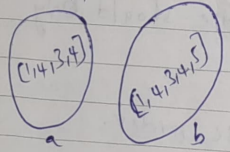
→ a = [1,4,3,4]



→ b = a.copy()



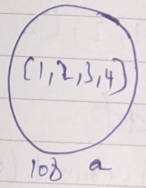
→ b.append([5])



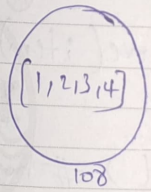
→ Print(a) # [1,4,3,4]
print(b) # [1,4,3,4,5]

deep copy

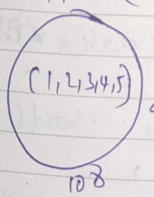
a = [1,2,3,4]



b = a



b.append([5])



print(a) # [1,2,3,4,5]
print(b) # [1,2,3,4,5]

DATE :

(VII)

Tuple:-

Same as list, but it is immutable.

→ Using ()

Ex:- l = (1,2,3)

Nested tuple:

t = (1,2,3,[4,5],(6,7),8)
print(t[3]) # [4,5]

So, it is since list type all the methods of list will be able to applied on it

i.e

t[3].append(60) # [4,5,60]
t[3].insert(0,99) # [0,4,5,60]
print(t) # (1,2,3,[0,4,5,60],(6,7),8)

but, it is applied on tuples (or) nested tuple

DATE :

concatenation:-

```
t1 = (1,2,3)
t2 = (4,5,6)
t3 = t1+t2
```

Note:-
in list →
i) .extend() will be change the items in existing obj.

i,e

```
l1 = (2,3,4) #id=234
l1.extend([4,5])
id(l1)# id=234
```

but
if l1+[4,5,6] will create entirely new obj of 'l1'

i,e

```
id(l1)#234
l1 = l1+[4,5,6]
id(l1)#536
```


To convert a list to tuple → Use tuple() function

→ Ex:- l1 = [1,2,3]
= l1 = tuple(l1)

DATE :

→ Vice versa from tuple to list

```
t1 = (1,2,3)
l1 = list(t1)
print(type(l1)) # list
```

*Interview Questions:-

Q) Select a tuple of your choice, where 1 item should be a list, Now inside that list can i add 1 value of my choice?

A) Yes.

Ex:- tup = (1,2,3,[4,5],"pavan")
=

```
tup[3].insert(0,"adithya")
print(tup)
# (1,2,3,["adithya",4,5],"pavan")
```

↑ added

Q) Can we replace a value in a list item inside a tuple?

A) yes.

Ex:- tup[3][0] = 400
=
tup # (1,2,3,[400,5],"pavan")

DATE :

V12 -

Sets:-

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc,

empty list => l = [ ] (or) l = list( )
tuple => t = ( ) (or) t = tuple( )
set => s = set( )

→ set is a unordered collection of item.
→ duplicates are not allowed
→ defined by { }.
→ set can only store immutable items
i,e it can store only – int, float, bool, str, tuple.

→ Nested set is not allowed since set is immutable.

DATE :

→ .add(item)
adds single item to set if not present if already present it will not add.

→ To add multiple item we use .update( ) method, but we need to add items inside that list.
i,e Ex:- $S_1 = \{1,2,3\}$

```
S1.update([4,5,"true"])
print(S1) #all items seperately added.
{1,2,3,4,5,"true"}
```

Simple,
To add one item => use .add( ) → as one item
To multiple items => use .update → as seperate item.
i,e

$S_1 = \{1,2,3\}$

S1.add((40,50))	S1.update((40,50))
#{1,2,3,(40,50)}	#{1,2,3,40,50}

Note:- Use tuple brackets only for you to avoid confusion.

DATE :

→ · discard(item)

removes specified item if available
if not available nothing happens.

→ · remove(item)

if available well and good it will remove.

if not available it will throw error.

→ · clear( )
→ del s
} Same concept as list

→ · copy( ) ⇒ to create shallow copy.

Note:- In set 'replace' is not possible bcuz we dont know index right 😊.

Set operations:-

DATE :

$S_A = \{1, 2, 3\}$

$S_B = \{2, 3, 4\}$

$S_3 = S_A.union(S_B)$

print (S_3)

⇒ S_A, S_B remains same.

→ · intersection( )

S_A.intersection(S_B)

→ · isdisjoint( )

S_A.isdisjoint(S_B) → True (if no common ele)
→ False (if common ele ✓)

→ $S_A - S_B$

S_A lo nunchi S_B ele del chey.

i.e print($S_A - S_B$) ⇒ {1}

print($S_B - S_A$) ⇒ {4}

→ · symmetric_difference( ) ⇔ $(A \cup B - A \cap B)$

S_A.( )(S_B) ⇒ # {1,4}

DATE :

→ .issubset(.) → True / False.

ie S_A.issubset(S_B)

(S_B contains all ele of S_A.)

Then,

~~→ .issuperset~~ A → subset

B → superset.

* dictionaries :-

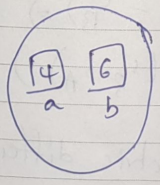
empty dict ⇒ dict() (or) {}

→ key value pair

Key : value

V13 :-

d1 = { "a": 4,
"b": 6 }



print(d1["a"]) # 4

→ Keys can be of fundamental datatypes Generally immutable

→ values can be of any data type.

DATE :

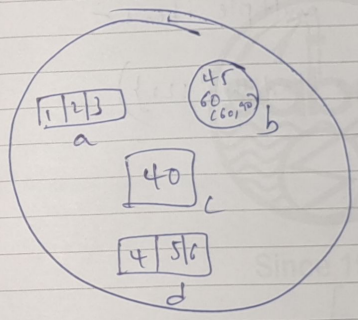
Ex:

d = { "a" = [1, 2, 3]

"b" = { 45, 60, (60, 90) }

"c" = 40

"d" = (4, 5, 6) }



to access (4, 5, 6) we do,

print(d["d"])

→ len(d) ⇒ 4.

→ print(d["d"][1]) # 5

like these also possible.

DATE :

To gets items (key-value pair)

→ .items( ) ⇒ returns list of tuple
[( )]

To get keys ⇒ .keys( ) ⇒ returns list. [ ]
To get values ⇒ .values( ) ⇒ returns list [ ]

→ To add single (or) multiple items –

.update({'pavan':22, 'adithya':22})

→ To delete a item.

~~del[key]~~

del d["keyname"]

Ex:-

= del d["pavan"]

(or)

d.pop("pavan")

→ To replace a value of a key.

i) Ex:- d = {"Hindi": 65,
"pavan": 72,
}

Now to replace '72' with 22 ⇒

i) d["pavan"] = 22
(but it will give key error if 'key' not present)

ii) d.get("pavan") = 22.

DATE :

→ To retrive a value of respective key.

Ex:- d = {"maths": 90, "Hindi": 65}

Now to retrive "maths" value –

i) print(d["maths"])
(but it may give error if 'key' is not present)

ii) print(d.get("maths"))
('None' will be returned if 'key' is not present)

DATE :

→ To replace a key in dict.

```
d = {"mathses": 90,
     "hindi": 65}
```

to replace "mathses" to "maths"

```
d["maths"] = d.pop("mathses");
print(d) # {"maths": 90, "hindi": 65}
```

(V14) :-

String methods:-

Note: String is immutable

→ .upper()
will make all alphabets to uppercase and does nothing to special char and nums.

Ex: S = "True....123"
S = S.upper()
print(S) # TRUE...123

→ s.lower()

→ s.swapcase() # up → low, low → up

→ s.capitalize() # (entire string first word letter will be capital)

DATE :

→ s.title() # Each word first word will be cap

→ s = s.strip()

→ s = s.replace(from, to) ⇒ replaces all, no need of loop.

→ l = s.split(" ") # split and returns list. based on space as delimeter.

→ list to str
i) s1 + " " + s2 + " " + s3
ii) " ".join(listname)

→ .index("item") # returns starting index number.
if not found → error.

→ .find("item") # Same as index but if not found error will not be raised but '-1' returned

DATE :

→ .count("item")

→ .startswith("str") → True
→ False

→ .endswith("str") → True
→ False

→ .isalnum() → True
→ False.

→ .isdigit()

→ .isalpha()

→ .islower()

→ .isupper()

→ .isspace() # only spaces ?

→ .istitle()

~~→ .isalnum()~~

5+18-0/9
23.0

DATE :

Interview Q) s = "Hello"
s[0] = "g"
whether it will be replaced ?

~~Yes it can be replaced but entirely new obj.~~

No, since strings are immutable.

Vis :-

/ → will always gives float

// → (trunc) part is returned of quotient.

BODMAS
() ↓
exponent

Ex :- 5+2*9-4//6**2/9
=
⇒ 23.0 ✓ (not) 23 X

()	high priority
**	
//	
*, /, %	} → left to right
+, -	low

Ex :- 5+5*5-5/5 ⇒ 29.0
=
5+5*5-0.0
5+25-0.0
30-0.0
⇒ 30.0

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,)
```

(4,) -> no. of items in tuple is no. of items

DATE :

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():-

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

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

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

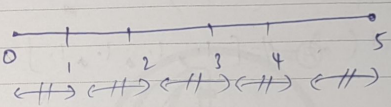
DATE :

(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. → [0, 0, 0, 0]

print(a) # [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, noofsamples)

Ex!- a = np.random.randint(1, 1000, 5)
print(a) # [294 803 670 292 901]
↓
type # class <array>

* Accessing values from array!-

DATE :

Ex!- l = [30, 40, 50, 60, 70]
(indexes: 0, 1, 2, 3, 4 / -5, -4, -3, -2, -1)
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>47]) # [46 47]

DATE :

→ insertion. (l1 = np.append(l1, 900)) → np.where(l1 > 5)
→ deletion (l1 = np.delete(l1, index)) } del l1
→ updation. → To del from memory.
→ Arithmetic 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.zeros((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 [[1 0] [0 1]] (2x2), [[1 0 0] [0 1 0] [0 0 1]] (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]) => [[1 0 0] [0 2 0] [0 0 4]]

DATE :

* indexing in 2D array:-

→ To extract a single value from 2D.

Syntax:-

arr_name[row_index, col_index]

Ex:-

	0	1
0	1	2
1	3	4

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
```


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()
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

$$a = \begin{bmatrix} 3 & 4 & 5 \\ 1 & 4 & 3 \end{bmatrix} \leftrightarrow a = \begin{bmatrix} 3 & 1 \\ 4 & 2 \\ 5 & 3 \end{bmatrix}$$

DATE :

* 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
```


DATE :

* 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 :-

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 4 & 5 \\ 6 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1\times4+3\times6 & 1\times5+3\times4 \\ 4\times4+5\times6 & 4\times5+5\times4 \end{bmatrix} \Rightarrow \begin{bmatrix} 22 & 17 \\ 46 & 40 \end{bmatrix}$$

c = np.matmul(a, b)

DATE :

* Concatenation :-

We can concatenate two arrays in two ways.
① vertically
② horizantally.

① Vertically :-

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

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

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] ⇒ [AB])
do,
arr4 = np.hstack((arr1, arr2))

(V25) :-

* pandas *

DATE :

Pandas
- Series
- dataframe.

Pandas Series :-

⇒ It is 1D array holding data of any type.
⇒ A pandas series is like a column in a table.

Ex :- import numpy as np
import pandas as pd

```
l = [10, 20, 30]
arr = np.array(l) # [10 20 30]
arr2 = pd.Series(l) #  0  10
                       1  20
                       2  30
```

⇒ A series can be created using '3' ways -
① list
② 1D array
③ Dict

DATE :

① List:-

Row name (or) index names.

l = [10, 20, 30]
arr => pd.Series(l) #

0	10
1	20
2	30

index names:- by default it will be index values.

Ex:- arr = pd.Series(l, index=["a", "b", "c"])

a	10
b	20
c	30

② Using 1D array:-

Same as above.

③ Using dict:-

d => {'a': 10, 'b': 20, 'c': [30, 40]}
arr => pd.Series(d)

a	10
b	20
c	[30, 40]

Note:- Key values of dictionary will be the index names when we use dictionary. and index name should be immutable.

DATE :

* Dataframe:-

=> It is a combination of multiple series.

② ~~difference b/w Series, dataframe~~

Series

l = [10, 20, 30]
t => pd.Series(l)

0	10
1	20
2	30

Dataframe

l = {'a': 10, 'b': 20, 'c': 30}
t => pd.DataFrame(l)

	a	b	c .....
0	10	20	30

Here

Ex:- d = {'a': [10, 20], 'b': [30, 40], 'c': [10, 30, 40]}
t => pd.DataFrame(d)

	a	b	c
0	10	30	10
1	20	40	30
2	-	-	40

DATE :

Note:-

1) A normal list can't be a DataFrame.
2) A normal list can be a series.
3) A ~~should~~ list which can be converted to 2D array those list can be a DataFrame.

List	Nested list
↓	↓ → which can be converted to
1D	2D
↓	↓ → that can be a
Series	DataFrame

-> Dataframe can be created using two ways

① Using 2D arrays
② Using dict type.

DATE :

① :

Ex:-

```
l = np.array([[10, 20, 30],
              [40, 50, 60],
              [60, 30, 40]])
d = pd.DataFrame(l, index=["a", "b", "c"],
                 columns=["i", "j", "k"])
print(d)
```

print(d)

	i	j	k
a	10	20	30
b	40	50	60
c	60	30	40

i, j, k → colnames.
a, b, c → Row names

② Using dict :- (Row names will become column names only in dict)

```
di = {'a': [10, 20, 30], 'b': [40, 50, 60], 'c': [60, 30, 40]}
df = pd.DataFrame(di)
print(df)
```

	a	b	c
0	10	40	60
1	20	50	30
2	30	60	40

Observe

DATE :

* How to add new column in a data frame

Ex:- d => {"Age": [10, 20, 30, 40, 50],
"Gender": [M, F, M, F, M]}

Now i want to add 'S.no' column.

So,

df => pd.DataFrame(d)

df["S.No"] = [a, b, c, d, e]

print (df)

S.No	Age	Gender
a	10	M
b	20	F
c	30	M
d	40	F
e	50	M

To set index as "S.No"

df = df.set_index("S.No")

DATE :

To retrieve the data we have two ways

1. loc
2. iloc

1:- Based on rownames, colnames

Syntax:- df[rownames, colnames] # To retrieve single row, single colum

Ex:- df[a, "Age"] # 10

To retrive multiple rows/colums

df[["a", "c", "e"], "Gender"]

	S.No	Gender
a		M
c		M
e		M

To retrive all rows of a single column

df.loc[[:], "Age"]

	S.No	Age
a		10
b		20
c		30
d		40
e		50

DATE :

(2) iloc:- Based on index numbers.

	-3	-2	-1
	0	1	2
-5	0	Age	Sal	Gender
-4	1	22	17000	M
-3	2	27	18000	F
-2	3	29	19000	M
-1	4	30	40000	M
		35	60000	F

→ col index numbers

→ index names.

row index numbers

Syntax:- df.iloc[row index number, column index num]

→ To retrieve all the rows of 'Age' col,

```
df.iloc[[:], 0]
```

(or)

```
df.iloc[[:], -3]
```


inplace = True.

DATE :

→ To retrive 1st and last row of Age, gender column-

```
df.iloc[[0,4], [0,2]]
```

(V26):

→ Reset index:-

```
df.reset_index()
```

→ How to drop a row:-

① Single row (or) multiple → index = row name.

Syntax:- df.drop(index=[,,,,])

→ How to drop column:- → column names.

df.drop(columns=[,,,])

270 74 430

3 5 7

DATE :

→ Sorting:-

sorting is done by considering any particular column.

```
df.sort_values(by="Age") #AESC
df.sort_values(by="Age") //
df.sort_values(by="Age", ascending=False) #DES
```

→ Combining multiple dataframes:-

df1

	city	temp
0	mumbai	32
1	delhi	45
2	benglore	40
3	hyd	36

df2

	city	humidity
0	delhi	68
1	mumbai	65
2	banglore	77
3	chennai	70

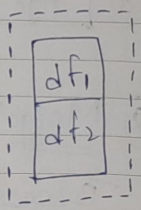
- there are two different ways to combine dataframes—
  1. Concat
  2. merge.

DATE :

1. Concat:-

Two dataframes will be concatenated no matter the duplicates, it will be added again.

i.e a) df1 / df2 (or) b) df1 | df2



a) pd.concat([df1, df2], axis=0)

	city	temp	humidity
0	Mumbai	32	NAN
1	delhi	45	NAN
2	benglore	40	NAN
3	hyd	36	NAN
0	delhi	NAN	68
1	mumbai	NAN	65
2	benglore	NAN	75
3	chennai	NAN	70

↓

To make (0, 1, 2, 3, 4, 5, 6, 7, 8 ....)

pd.concat([df1, df2], axis=0, ignore_index=True)

- and, To make df1 | df2

DATE :

pd.concat([df1, df2], axis=1)

	city	temp	city	Humidity
0	mumbai	32	delhi	68
1	~~benglore~~ delhi	45	mumbai	65
2	~~hyd~~ benglore	40	benglore	75
3	hyd	36	chennai	70

② Merge:-

~~To merge two dataframes~~

As we can see if we consider city column we can observe 'mumbai', ~~hyd~~ 'benglore' repeted two times since they are same places.
so, to avoid it we use merge.
pd.merge(df1, df2, on="city", how="~~inner~~ left")

	city	temp	humidity
0	mumbai	32	65
1	delhi	45	68
2	benglore	40	75
3	hyd	36	NAN

how => will take four Arg

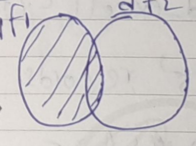
they are how = { inner(default)
outer
left
right

DATE :

Note:- { SQL Joins = merge (pandas) }

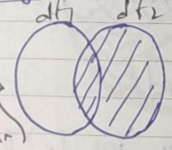
i) Left merge:-

{Records of df1 & common records of df1 & df2 based on col}

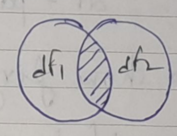


ii) Right merge:-

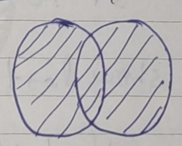
{Records of df2 & common records of df1 & df2 based on common column}



iii) inner:- {common Records in both df1 & df2 based on col}



iv) outer:- {All the records}



{df1 → df1

error

Use → if no column is common then no merge is possible

merge => when you want to combine two df based on common column.

concat => when you want to attach two df no matter the common columns.

DATE :

* importing datasets:-

① pd.read_excel("path")

path should be \\ (or) /

Ex:- df = pd.read_excel("c:/Home/test.xlsx")

→ To show first '5' records.

df.head() // def => 5

df.head(n=10) # '10' records from first

→ To show last records.

df.tail() // def = '5' last records.

df.tail(n=10) # Last '10' records.

→ To know the dimension of df →

df.shape

Ex:- (4,3)

→ To know total no. of values.

df.size

Ex:- 4x3 => 12 values.

DATE :

→ To know the column names:-

df.columns

(or)

df.keys()

V.27 :-

→ to rename column names:-

df.rename(columns={'aGE': "AGE",
'sum': "SUM"})

→ datatypes of each column:-

df.dtypes → object (if one value is str in col then it is)
→ float (if one float is present)
→ int &

~~if (object)~~

DATE :

i.e if (column has atleast one string)
{ output => object;
}
else if (column has atleast one float)
{ output => float;
else { output => int;
}

priority maximum → object
float
priority minimum → int

→ To give Overview of dataset:-
df.info()
⇒ shape + dtype + columns.

→ check for missing values:-

DATE :

df.isnull().sum() ⇒ int

→ How to convert panda series to 1D array:-

df["colname"].values
(or)
df[" "].to_numpy()

* if it is a series then it will be converted to '1D' array.
* if it is dataframe then it will be converted to '2D' array.

and to convert 2D → 1D we use ravel().

i.e
df[" "].to_numpy().ravel()

DATE :

*filtering:-

→ Selecting/extracting the data based on condition.

Ex:-
Consider a dataframe.

	X	Y
0	9	5
1	3	6
2	4	6
3	5	2
4	7	4
5	2	2

→ To get only records which are $X \geq 3$

So,
```
df[df["x"]>=3]
```

→ To get only record where $X \geq 3$ and $Y \leq 4$

So,
```
df[(df["x"]>=3) & (df["y"]<=4)]
```


Note:- Pandas
core
and ⟺ &
or ⟺ |

DATE :

→ To get records where $X == 3$

So,
```
df[df["x"]==3]
```

———— x ————

→ To get what categories of data present in a column.

```
df["Gender"].unique()
#(M, F)
```

	Gender
0	M
1	M
2	F
3	M
4	F

→ To get how many count that each category has —

```
df["Gender"].value_counts()
```

→ To get ~~the~~ total how ~~records unto~~

DATE :

*Discretization:-

→ converting numerical data into non numerical data.

Ex:-

	Marks
0	70
1	60
2	50
3	90
4	95
5	72

Now marks < 70 are 'fail' to be marked
marks > 90 are 'exellent'
→ 70 < 90 are 'moderate'

So,

```
df["status"] = pd.cut(df["marks"],
                      bins=[0,70,90,100],
                      labels=["fail","moderate","exellent"])
```

output:-

	Marks	status
0	70	fail
1	60	fail
2	50	fail
3	90	moderate
4	95	exellent
5	72	moderate

*crosstab:-

→ applied only on '2' discrete categorical variables.

DATE :

Ex:- pd.crosstab(df['time'], df['gender'])

time \ gender	female	male
Dinner	52	124
lunch	35	33

(128) :-

statistics

*Analytics :- Complete analysis of data.

Types of Analytics:-

1. Descriptive Analytics:- Describe the data. (statistic
2. Diagnatic Analytics:- check the reason behind that.
3. Predictive Analytics:- Predict (ML & AI)
4. Prescriptive Analytics:- What we have todo.