Python Pandas

Series में सिर्फ row use होगा ↳ 1D
Data frame में row & col दोनों use होंगे ↳ 2D

Date __/__/____

def → Pandas is an open source python libratary providing high performance data manipulation and data analysis using its powerful data structure

→

Property	Series	Data Frame
Dimensions	1D	2D
Types of Data	Homogeneous (all elements must be same type)	Heterogeneous (elements can have diff type)
mutability	a) Value Mutable b) Size mutable (size of series object once created cannot change)	(a) Value mutable (b) Size mutable

Series data Structure

index	data
0	24
1	28
2	33
3	44
4	51

→ A series data structure has 2 main components -
a) index
b) data

→ Both components are one dimentional array with same length

→ The index is used to access individual data value
Eg → $S_2[2]$ → 33

Creating series object MP

Creating empty series object

```
import pandas as pd
S2 = pd.Series()
```
S2 → series object
↳ खाली है क्यूँकी empty object

```
print(S2) → Series([], float64)
```

Date __/__/____

Creating Non empty ~~Series~~ Series objects

Syntax

<Series object> = pandas.~~Series~~Series(data, index=idx, dtype=dtype of data)

data: group of homogeneous data

idx: numpy

data can be one of the following

i) List
ii) ndarray
iii) Dictionary
iv) Scalar value

① Specify data ~~as~~ as python sequence(List)

To create a series object using List

Q) Write a python code to create a series by using python sequence/list [10, 20, 30, 40, 50]

→
```
import pandas as pd.
(capital 'S')
s1 = pd.Series([10,20,30,40,50])
print(s1)
```

output

index	data
0	10
1	20
2	30
3	40
4	50

~~Q2) To cr~~

Date __/__/____

To create a series object with explicitly defined index value

```
import pandas as pd
s2 = pd.Series([10,20,30,40,50], index=['a','b','c','d','e'])
print(s2)
```

output

index	data
a	10
b	20
c	30
d	40
e	50

handling floating point value for generating series object

```
import pandas as pd
s3 = pd.Series([2,4,6,7.5])
print(s3)
```

output

0	2.0
1	4.0
2	6.0
3	7.5

Since 7.5 is one of the element in the list, is a float value it shall convert result of int value into float value & hence the result display a float value.

numpy ✓ ~~numphy~~

Date ___/___/_____

(ii) ndarray

Specify ~~data~~ as ndarray

→ numpy array use होता
↳ सारा data same type का

- functions

numpy.arrange()
numpy.linspace()
numpy.tile()

a) numpy.~~arrange~~ arangi()

→ behave like python ~~behave~~ range function

Syntax numpy.arrange(start, stop, step)

→ return an array whose stock argument is not inclusive

Eg →

```
import panda as pd
import numpy as ~~numphy~~ np

a1 = np.arrange(3, 13, 3.5)
S1 = pd.Series(a1)
```

output

0	3.0
1	6.5
2	10.0

→ last value exclusive होता है

Jb n.0 होगा, तो बस n. आएगा '0' नहीं लिखा जाएगा

Date __/__/____

b) numpy.linspace()

Syntax → numpy.linspace(start, stop, num, dtype)
num → no of item to generate with in the range

d.f → return evenly spaced number over a specified interval

Eg → numpy.linspace(start=0, stop=100, num=5) | output [0 25 50 75 100]

Q) Write a python code To print array of 6 values b/w 2 & 3.

→ import numpy as np

val = np.linspace(2, 3, 6)
print(val) → [2. 2.2 2.4 2.6 2.8 3.]

Q) Write a py code to create a series object using ndarray that has 5 elements in the range 24 – 64.

→ import pandas as pd
import numpy as ny

arr = ny.linspace(24, 64, 5)
sl = pd.Series(arr)

print(sl) → (series में is like 24.0)

0	24.0
1	34.0
2	44.0
3	54.0
4	64.0

Date ___/___/_____

(c) ~~numpy.type()~~ tile()

def → Construct a new array by repeating array

syntax → numpy.tile(arr, rep)

a) write a python code to create a series object using nd array created by ~~titi~~ tiling a list [3,5] twice

→

```
import numpy as ny
import pandas as pd.

s1 = pd.Series(ny.tile([3,5], 2))
print (s2)
```

output

0	3
1	5
2	3
3	5

(iii) ~~*~~ Dictionary

def → using a dictionary for creating a series object, key of the dict become ~~to~~ index of the series object & value of dictionary ~~&~~ become data of series object.

c) Write a python code to create a series object ~~not~~ using the dictionary that store no of students in each section.

→ d = {'a': 50, 'b': 40, 'c': 55, 'd': 65}

```
import pandas as pd
s1 = pd.Series(d)
print(s1)
```

→

a	50
b	40
c	55
d	65

Date ___/___/_____

Naming a Series

→ index ला col heading दे शकतो

```
import pandas as pd
import ~~n~~
d = {A: 45, B: 65, C: 52, D: 60}
S1 = pd.Series(d)
```

Out[ ]

Section	
A	45
B	65
C	52
D	60

```
S1.~~print~~ index.name = ~~student~~ 'section'
print(S1)
```

You can give a name to column index of a series object by using name property

iv) Specify data as a scalar value (fixed value)

```
import panda as pd
S1 = pd.Series(data = 50000, index = ['qtr1', 'qtr2', 'qtr3', 'qtr4'])
print(S1)
```

→

qtr1	50000
qtr2	50000
qtr3	50000
qtr4	50000

→ data can be scalar or constant value
→ If data is a scalar value, then index arg to series function must be provided. The scalar value shal be repeated to match the length of the index

Page No.

Date __/__/____

Q) Write a python code to create a series object that store initial budget allocated 50 000 each for the 4 quaters of the year

→ import pandas as pd

sl = pd. Series(50 000, index=['q1', 'q2', 'q3', 'q4'])
print (sl)

Additional functionality < NaN / None

→ Nan numpy class ~~का~~ ~~है~~ → np. NaN ~~होता~~

In certain situations, we need to create a series object for which size is defined but sum element or data are missing. This is ~~to~~ handled by NaN value, which is attribute of numpy library

Eg→ import pandas as pd
import numpy as ~~#~~np

sl = pd. Series([7.5, 5.6, np.NaN, 6.2])
print (sl)

→

0	7.5
1	5.6
2	NaN
3	6.2

Date __/__/____

We can also use None to add missing data

Eg →
```
import pandas as pd
s1 = pd.Series([7.5, 5.6, None, 6.2])
print(s1)
```
→
```
0   7.5
1   5.6
2   NaN
3   6.2
```

→ isme at NaN aata hai

Eg → ④ Specify index as well as data with series

```
import pandas as pd
mon = ['Jan', 'Feb', 'Mar', 'Apr']
d = [31, 28, 31, 30]
s2 = pd.Series(d, mon)
s2.index.name = 'month'
print(s2)
```
→
```
month
Jan   31
Feb   28
Mar   31
Apr   30
```

a) Find the output
```
import pandas as pd
s4 = pd.Series(range(1, 15, 3), index = [x for x in 'abcde'])
print(s4)
```
→
```
a   1
b   4
c   7
d   10
e   13
```

Date ___/___/_____

a) iii) Specify datas type along with data & index find output

→ import pandas as pd
import numphy as np

mon = ['Jan', 'Feb', 'Mar', 'Apr']
d = [31, 28, 31, 30]

S5 = pd.Series(data = d, index = mon, dtype = np.float64)
s5.index.name = 'month'
print(s5)

Output →

month	
Jan	31.0
Feb	28.0
Mar	31.0
Apr	30.0

1D array	List
no commas [2 4 6 8 10]	[2, 4, 6, 8, 10]

Date __/__/____

iv) Using a Mathematical function / expression to create a data array in Series()

Syntax → <Series object> = pandas.Series(data=<function/expr>, index)

data array: 1 D array

numpy.arange() → it behave like python range function
→ return the ~~retu~~ array whose stop arg are exclusive

a) import numpy as ny
import pandas as pd

```
a = np.arrange(3, 15, 3)
print(a) ——→ [3 6 9 12]

S = pd.Series(data = a*2, index = a)
print(s)
↳
3   6
6   12
9   18
12  24
```

numpy array support vectorized operations. This operation is applied on every element of numpy array and stored as a part of series object. But if you apply a similar operation on py list, the output will be completely differed. & no when multiple with a list if replicate the list those many times

Date ___/___/_____

important point

indices need not be unique in a panda series

```
print(s1['b'])
    ↳ 6
print(s1['a'])
  ↳ a  3
    a  9
    a  12
    a  18
```

(s1) Series

a	3
b	6
a	9
a	12
c	15
a	18

while creating a series object. when you give a index array as a sequence, then ~~three~~ there is no compulsion for the uniqueness of the ~~comp~~ indexes. You can have duplicate entry in the index & python could not raise any error

~~Series~~

1 byte = 8 bits | 1 int को store करने के लिए 8 bytes लगते
~~तो~~ 8 ⇒ 8 byte ⇒ 8 × 8 = 64 bits

Date ___/___/_____

Series object attribute
↳ property
↳ features

S= series obj	
a	1
b	4
c	7
d	10
e	13

① S. index → return index of the series object

out[]
index(['a', 'b', 'c', 'd', 'e']) data type → object
↳ अगर 1 diff character हो जाये तो obj होगा

Pra ⓒ What is the datatype of series obj Sl given below

sl = pd.series([11, 12.5, "ok"])

a) int 64
b) Float 64
c) object
d) object 64
↳ exist नहीं करता
ऐसा confusion के लिए दिया है।

② S. values → 1d array होगा

print (s.values) → [1 4 7 10 13]

③ S.dtype → return type of data value

④ S. shape → ~~s size~~ ~~return tuple~~ as
→ return how many element it contain, including NaN, in the form of tuple

out[] → (5,)
↳ comma इसलिए आया है

~~S.Size~~

Date ___/___/_____

(VI) S.nbytes → return total no. of bytes. ~~reqno~~
→ 8 × no of elements

(VII) S.ndim → Series 1d होगा
→ diamention हमा

(VIII) S.size → return total no. of elements
and [] → 5

~~(VIII) S.itemtype~~

(VIII) S.itemsize → 1 item (element) को store करने का size (bits)
→ (8)

S.hasnans

(IX) ~~S.hasNans~~ → True या false

(X) S.empty → return true if series elm ∅ empty

Accessing a Series object and its elements

print (s1['c']) → 7

print (s1['b']) → b 4
b 10

s1	
a	1
b	4
c	7
b	10
e	13

position always starts from ~~zero~~ '0'

Slicing में last value inclusive होती है range में ~~arrange~~ arange में exclusive होती है

Date ___/___/_____

① Accessing individual elements

To access individual elements of a series object, you can give ~~it~~ its index in square brackets along with its name

② Extracting slice from series object (More than 1 element को लेना)

→ Slicing takes place position wise and not ~~the~~ index wise

→ Multiple value जहाँ index आ गया होगा, वहाँ use होगा।

Syntax → var[start : stop : step]

eg → s[1:4:2] out[] → 1, 10

⑤

position	index	data
0	a	1
1	b	4
2	c	7
3	d	10
4	e	13

→ internally, there is a position ~~also~~ associated with element — 1st element get the position '0' and so on.

When you extract slices, you have to need to specify slice as : Series object [start : stop : step]
But 'start' & 'stop' signify the position of the element. ~~But~~ not the index

PYQ) Find output

```
import pandas as pd

s1 = pd.Series(data = 2*[3, 2, -6])
print(s1)
```

out[]

0	3
1	2
2	-6
3	3
4	2
5	-6

Lab Ass - 6

Date ___/___/_____

Create a series object 's2' that store the no. of students in each section of class 12 is given below -

A	39
B	45
C	48
D	50

Q) 1st 2 sections have been given a task of selling tickets @₹100/each as part of social experiment.
Write the code to display how much they have collected (sectionwise).

→ import pandas as pd

```
stud = [ 39, 45, 48, 50]
sec = [ 'A', 'B', 'C', 'D']

s2 = pd.Series(data = stud, index = sec)

print (s2[ : 2] * 100)
```

→ A 3900
B 4500
dtype: int64

Date __/__/____

Operation on series object

1) Modifying element of series obj

→ The data value of a series obj can easily modified by the following statement:

syntax → <series obj>[<index>] = <new value> (for 1 elm)

↳ <series obj>[start: stop: step] = <new value> (for multiple elm)

Renaming Index ~~(index object)~~

You can even change or rename index of the series obj by assigning new index array to its index attributes

syntax → <series obj>.index = <new index>

→ changing index

Eg ↳ From

0		a
1	To →	b
2		c
3		d

Date __/__/____

2) Retrieving values from a series obj by using head() and tail()

→ The head() function is used to fetch 1st n rows from ~~the~~ a Series object

→ Tail() return last n rows from the series object

Syntax → <series obj>.head([n])

(series objects).tail([n])

'n' ~~is~~ by def value '5'

→ if you donot provide any value for 'n'. The head() & tail() will return first '5' & last '5' rows ~~for~~ respectively from a series obj.

3) Vector operation on series object

Def → Vector operation means that if you apply a function then it is individually applied ~~to~~ on each item of the object

Eg →

S5	
0	11
1	12
2	13
3	14
4	15

print (S5 + 2)

0	13
1	14
2	15
3	16
4	17

Jitna index match होगा उतना में arithmatic होगा, बाकी में NaN देगा

Date __/__/____

4) Arithmatic operation on Series object

```
>>> s                    >>> s1                   >>> s3
1   11                   1   21                   101  21
2   12                   2   22                   102  22
3   13                   3   23                   103  23
4   14                   4   24                   104  24
```

```
s + s1 →  1   34
          2   34      } index same है, तो add हो जाएगा
          3   36
          4   38
```

```
s1 + s3 →  1    NaN
           2    NaN
           3    NaN
           4    NaN
           101  NaN   } index same नहीं हैं, तो NaN आएगा
           102  NaN
           103  NaN
           104  NaN
```

You can perform arithmatic operation like addition, substraction etc. with series obj, element by element. based on indexes

The arithmatic operation is possible on object of the same index. Otherwise, it will return NaN.

Boolean exp → जहाँ relational exp का होता है।

Date ___ / ___ / _____

5) Retrieving values using condition (filtering)

↳ relational operator use होता (=, >, <, ...)
↳ output true/false आउटपुट (Boolean)

Giving condition to retrive value from a series object

Syntax <series obj>[bool exp on series]

Eg → ③ = 0 1.0 print (s<2)
1 1.4 ↳ True
2 1.8 True
3 1.5 True
4 2.0 True
false

6) Sorting Series value

a) Sorting on basis of values

Syntax - <series obj>.sort_values ascending = True ⇒ def value

Eg - S1 ⇒ A 19
B 21
C 06
D 13

S1.sort_values( )
↳ C 06
D 13
A 19
B 21

S1.sort_values(ascending = False)
↳ B 21
A 19
D 13
C 06

Date __/__/____

→ To sort a series obj on the basis of values, you may use sort_value().

→ The arg assending is optional and if skipped, it take the value assending = True by default

b) Sorting on the basis of indexes

Syntax → <Series obj>.sort.index()

Eg → S1.sort_index() →

A	19
B	21
C	06
D	13

S1.sort_index(assending = False) →

D	13
C	06
B	21
A	19

7) Deleting element from a series obj [drop()]

We can delete an element from a series obj by using drop() by passing index of the element to be deleted as the argument to it

Syntax → S1.drop('B')

Q) Find the output

```
import pandas as pd

sl = pd.Series(data = 2*[31, 2, -6])
print(sl)
```

↳

0	31
1	2
2	-6
3	31
4	2
5	-6

Q) `sl = pd.Series(data = range(31, 2, -6), index = [x for x in 'aeiou'])`
`print(sl)`

↳

a	31
e	25
i	19
o	13
u	7

Q) `sl = pd.series([11, 12.5, 'ok'])`
datatype? => object

Q) ⑤

0	0.43
1	0.61
2	0.26
3	0.83

① print s*100

↳

0	43
1	61
2	26
3	83

④ print (s>0)

~~0 True~~ ↳ Error

~~1 True~~

~~2 True~~

~~3 True~~