Series Data Structure: Series is a pandas DS that represents a one-dimensional array like object containing an array of data* (of any NP data type) and an associated array of data labels, called index.

Creating Series Object:-

1) **Creating empty series:**

```
<Series Object> = pandas.Series()
```

2) **Creating non-empty Series Object:**

```
<Series Object> = pd.Series(data, index=idx)
```

Where data could be of any following kind:

- A Python sequence
- An ndarray
- A Python dictionary
- A scalar value.

Eg:

```
ndar1 = np.arange(3, 13, 3.5)
ser1 = pd.Series(ndar1)
ser1
```

O/P:

```
0    3.0
1    6.5
2   10.0
dtype: float64
```

* data could also be a Panda object (Series * not DF).

Date
Page 5

e.g: medalwon = pd.Series (10, index = range(0,1))
medals2 = pd.Series (15, index = range (1, 6, 2))
ser2 = pd.Series ('Yet to Start', \
index = ['Indore', 'Delhi', 'Shimla'])
ser2

o/p: Indore Yet to Start
Delhi "
Shimla "
dtype: Object

e.g: obj3 = pd.Series ([6.5, np.NaN, 2.34])
obj3

o/p:
0 6.5
1 NaN
2 2.34
dtype: float64

Note: (I) V.imp If data is NP then length of index and data must be identical.
If data is a Pandas Series Object then additional index would be filled automatically with NaN.
(P321 IP. py)

(II) Loop could be ~~defined~~ used to define index Note
e.g: S1 = pd.Series (range(1,15,3)), \
index = [x for x in 'abcde'])

* bcoz DF is 2d and series is 1d.

o/p:
```
a    1
b    4
c    7
d    10
e    13
dtype: int64
```

3) Using a mathematical function / expression to create data array in Series():

e.g.: a = np.arange(9, 13)
obj7 = pd.Series(index = a, data = a*2)

o/p:
```
9    18
10   20
11   22
12   24
```

e.g.: lst = [9, 10, 11, 12]
obj8 = pd.Series(data = (2*lst))

o/p:
```
0    9
1    10
2    11
3    12
4    9
5    10
6    11
7    12
```

Note List is replicated. Here index is skipped that's why replicated, otherwise it would have been an error, in case of mismatch in index and data.

Series Object Attributes:

V – Values
I – Index
D – dtype
I – itemsize
S – size (No. of elements)
H – hasnans

N – ndim
E – empty
S – shape
N – nbytes

Accessing a Series Object and its Elements:

(1) Accessing individual elements

e.g.: obj6[3]
obj5['Feb']

(2) Extracting slices from Series Object:

Slicing takes place position wise and not the index wise in a series object.

obj7

position	index	data
0	9	18
1	10	20
2	11	22
3	12	24

When you have to extract slices, then you need to specify slices as [start : end : step].

eg: obj7[0 : : 2]

o/p: 9 18
11 22

e.g: obj7[: : -1]

o/p 12 24
11 22
10 20
9 18

Operations on Series Object:-

① Modifying Elements of Series Object:-

Series objects are value mutable as well as size-immutable.

Condition	obj
0	1.5
1	12.75
2	24.00
3	35.25
4	46.50

	obj2
a	1.5
b	12.75
c	24.00
d	35.25
e	46.50

* obj[5] = 44.4
del obj[2]

obj[2:4] = -15.75

o/p:

0	1.5
1	12.75
2	-15.75
3	-15.75
4	46.50

obj2[1:5:2] = 380

o/p:

a	1.5
b	380.0
c	24.00
d	380.0
e	46.50

Indexes of a Series object can also be changed by assigning new index array.

e.g. obj2.index = ['v', 'w', 'x', 'y', 'z']

o/p:

v	1.5
w	380.0
x	24.0
y	380.0
z	46.50

(2) The head() and tail()

head() is used to fetch first 'n' rows.
tail() " " last " .

If you do not provide any value for n, then head() and tail() will return first 5 and last 5 rows.

③ Vector operations on Series object:

Since series objects are built upon ndarrays, they also support vectorized operations - just like ndarrays.

e.g: ob2 + 2, ob2 * 3, ob2 ** 2
ob2 > 15 etc. np.sqrt(ob1)

④ Arithmetic on Series Object:

ob1 + ob3
ob1 * ob3
ob1 / ob3
ob2 + ob5

Ex:

	ob1
0	1.35
1	12.75
2	-15.75
3	-15.75
4	46.50

	ob4
0	1.25
1	5.53
2	9.80
3	14.08
4	18.35
5	22.63
6	26.90
7	31.18

ob1 + ob4

0	3.10
1	18.28
2	-5.94
3	-1.67
4	64.85
5	NaN
6	"
7	"

Note
Objects that have some or all non-matching indexes of both the objects,

it will return NaN.

When you perform arithmetic operations on two series type objects, the data is aligned on the basis of matching indices. For non overlapping indices the arithmetic operations result as a NaN.

Result of object arithmetic operations can be stored in another ~~existing~~ object. Note

e.g: ob6 = ob1 + ob3

Result of vector operation can also be stored in another object.

e.g: ob8 = ob2 ** 2

(5) Filtering Entries:- Entries can be filtered from series objects using expressions of type boolean.

e.g:

	ob5
a	1.25
b	5.53
c	9.80
d	14.08
e	18.35

Now ob5 > 5 will evaluate as

a	False
b	True
c	"
d	"
e	"

But obs[obs > 5] filtered the entries from series.

obj

b	5.53
c	9.8
d	14.08
e	18.35

Difference between Numpy Array & Series Objects:-

(i) In case of ndarray vectorized operations can only be performed if shapes of two ndarray match. But with series object, data is aligned as per matching of indices and for non matching indices NaN returned.

(ii) In ndarray indexes are always numeric, but series objects can have any type of indexes.