In [1]:
```
# VIVEK-CHAUHAN-ADVANCED-DATA-ANALYTICS-PANDAS-1
```

In [2]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
empty = pd.Series()
print(empty) #without attribute it will give an error may be.
```

```
Series([], dtype: object)
```

In [11]:
```
a = pd.Series([1,2,3,4])
print(a)
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

In [27]:
```
sequence = pd.Series(range(5)) # range is used to store values upto and series is u
print(sequence)
```

```
0    0
1    1
2    2
3    3
4    4
dtype: int64
```

In [39]:
```
index = pd.Series([1,2,3,4,5],index = [5,6,7,8,9]) #index is assign the starting or
print(index)
```

```
5    1
6    2
7    3
8    4
9    5
dtype: int64
```

In [46]:
```
s1 = pd.Series([4,6,8,10])
print("series object")
print(s1)
```

```
series object
0     4
1     6
2     8
3    10
dtype: int64
```

In [44]:
```
s2 = pd.Series([11,21,31,41])
print("object series-2")
print(s2)
```

```
object series-2
0    11
1    21
2    31
3    41
dtype: int64
```

In [48]:
```
s3 = pd.Series(['O','H','O'])
print(s3)
```

```
0    O
1    H
2    O
dtype: object
```

In [52]:
```
s4 = pd.Series("SO FUNNY")
print(s4)
```

```
0    SO FUNNY
dtype: object
```

In [62]:
```
s5 = pd.Series(np.arange(3,13,3.5)) #it will arange means start from 3 and end from
print(s5)
```

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

In [68]:
```
s6 = pd.Series(np.linspace(24,64,5)) #linspace is used like a tuple.start,end,and g
print(s6)
```

```
0    24.0
1    34.0
2    44.0
3    54.0
4    64.0
dtype: float64
```

In [75]:
```
s7 = pd.Series(np.tile([3,5],2)) #tile is used for repeating the data.
print(s7)
```

```
0    3
1    5
2    3
3    5
dtype: int32
```

In [79]:
```
s8 = ({'jan':31,'feb':28}) #here it will return as a disctionary like key and value
print(s8)
```

```
{'jan': 31, 'feb': 28}
```

In [81]:
```
s8 = pd.Series({'jan':31,'feb':28}) #here it will return the index and value cause
print(s8)     # series assign the values in the sequence format.
```

```
jan    31
feb    28
dtype: int64
```

In [83]:
```
section = {'A':39,'B':41,'C':42,'D':44} # you can declare disctionary first and the
student = pd.Series(section)
print(student)
```

```
A    39
B    41
C    42
D    44
dtype: int64
```

In [85]:
```
# now we will use it like a scalar value.

s9 = pd.Series(10,index = range(0,1))
print(s9)
```

```
0    10
dtype: int64
```

In [87]:
```
# now we will use it like a scalar value.

s9 = pd.Series(15,index = range(1,6,2)) # it will store the same data from start to
print(s9)
```

```
1    15
3    15
5    15
dtype: int64
```

In [9]:
```
s10 = pd.Series(200,index = range(2020,2029,2)) #alter year that's why the gap is 2
print(s10)
```

```
2020    200
2022    200
2024    200
2026    200
2028    200
dtype: int64
```

In [11]:
```
#explicitly used the data and the index values.

section = ['a','b','c','d']
contri = [5000,2525,2020,2024]
s11 = pd.Series(data = contri,index = section)
print(s11)
```

```
a    5000
b    2525
c    2020
d    2024
dtype: int64
```

In [19]:
```
# specify the data type along with the data as well as the index

s12 = pd.Series(data = None,index = None,dtype = None) #none must be a camelcase.
print(s12)
```

```
Series([], dtype: object)
```

In [29]:
```
#mathematical function/expression to create array in the series.

a = np.arange(9,13) #it will create an series array.
print(a)
```

```
[ 9 10 11 12]
```

In [31]:
```
b = pd.Series(index = a,data = a*2) # now the mathematical expression comes in the
print(b)
```

```
9     18
10    20
11    22
12    24
dtype: int32
```

In [37]:
```
c = pd.Series(index = a,data = a**2) #double * indicates the square value.
print(c)
```

```
9      81
10    100
11    121
12    144
dtype: int32
```

In [43]:
```
list = [2020,2022,2024,2025]
d = pd.Series(data = 2*list) #if you create the list then the * operator repeat the
print(d)
```

```
0    2020
1    2022
2    2024
3    2025
4    2020
5    2022
6    2024
7    2025
dtype: int64
```

In [53]:
```
list = pd.Series(data = [10,20,30,40])#if you create the list then the * operator r
print(list) #now it will become series.
print(list*2) #now it wil become ans in the array.
```

```
0    10
1    20
2    30
3    40
dtype: int64
0    20
1    40
2    60
3    80
dtype: int64
```

In [57]:
```
print(list.index) #it will return the index of the array.
```

```
RangeIndex(start=0, stop=4, step=1)
```

In [61]:
```
print(list.values) #it will return the values of the array.
```

```
[10 20 30 40]
```

In [65]:
```
print(list.ndim) #it wil return the number of dimensions.
```

```
1
```

In [69]:
```
print(list.name) #if there is any assign or series object.
```

```
None
```

In [8]:
```
series1 = pd.Series(data = [1,2,3,4,5],index = ['a','b','c','d','e'])
print(series1)

series1.index.name = 'newseries' # here we declare the new name of the index.
print(series1)
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
newseries
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [77]:
```
print(series1) # new name of the series or object is permanent.
```

```
newseries
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [85]:
```
print(series1.name) #it will return the new name of the series.
```

```
None
```

In [91]:
```
series1.name = "name of series" #now we declare the name of the series.
print(series1)
```

```
newseries
a    1
b    2
c    3
d    4
e    5
Name: name of series, dtype: int64
```

In [95]:
```
print(series1.name) #it will return the name of the series.
```

name of series

In [113…
```
# retrieve the data type and size of the type.

print(series1.dtype) # it will return the data type of the series.
```

int64

In [115…
```
print(series1.itemsize)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28628\4233166749.py in ?()
----> 1 print(series1.itemsize)

C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'Series' object has no attribute 'itemsize'
```

In [111…
```
print(type(series1)) # it will return the type of the series / array.
```

<class 'pandas.core.series.Series'>

In [119…
```
# retriving the shape of the object

print(series1.shape) #shape count the how many elements it ontains including missio
```

(5,)

In [125…
```
#retriving dimensions of the object.

print(series1.nbytes) #it will return the element * 8 ex:- 5 element * 8 byte per e
```

40

In [127…
```
# checking the empty attributes or not.

obj = pd.Series()
```

In [131…
```
print(obj.empty) # it will return the true cause the obj is empty.
```

True

In [135…
```
print(series1.empty) # it will return false cause the series1. is not empty.
```

False

In [141…
```
print(series1.len())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28628\2994100963.py in ?()
----> 1 print(series1.len())

C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'Series' object has no attribute 'len'
```

In [145…
```
print(series1.count()) # it will count the total number of elements.
```

```
5
```

In [149…
```
# accessing the individual elements from a series object.

print(series1[2])
```

```
3
C:\Users\fv8.DESKTOP-N5HA3AQ\AppData\Local\Temp\ipykernel_28628\2217711831.py:3: Fut
ureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with DataFrame be
havior). To access a value by position, use `ser.iloc[pos]`
  print(series1[2])
```

In [159…
```
# extracting slices from the series object.

print(series1[1:]) #the index name is just for our understanding computer memory is
```

```
newseries
b    2
c    3
d    4
e    5
Name: name of series, dtype: int64
```

In [161…
```
# extracting slices from the series object.

print(series1[1::2]) #the index name is just for our understanding computer memory
```

```
newseries
b    2
d    4
Name: name of series, dtype: int64
```

In [165…
```
print(series1[:2]*100) # it will multiply by 100 till the 2 index is not come.
```

```
newseries
a    100
b    200
Name: name of series, dtype: int64
```

In [12]:
```
# modifying elelment on the series object
```

```
print(series1)
series1[1]
```

```
newseries
a    1
b    2
c    3
d    4
e    5
dtype: int64
C:\Users\fv8.DESKTOP-N5HA3AQ\AppData\Local\Temp\ipykernel_37000\4253546821.py:4: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  series1[1]
```

Out[12]: 2

In [18]:
```
series1[2:4] = 15 #update the series element.
```

In [20]:
```
print(series1)
```

```
newseries
a     1
b     2
c    15
d    15
e     5
dtype: int64
```

In [24]:
```
# renaming the indexes

series1.index = [11,12,13,14,15] #declare the new indexes for the array.
print(series1)
```

```
11     1
12     2
13    15
14    15
15     5
dtype: int64
```

In [35]:
```
#modifying the index values.

series1[11] = 13
series1[2:4] = 26

print(series1)
```

```
11    13
12     2
13    26
14    26
15     5
dtype: int64
```

In [37]:
```
#head and the tail functions

series1.head() # without passing the number in the head function by default it will
```

Out[37]:
```
11    13
12     2
13    26
14    26
15     5
dtype: int64
```

In [43]:
```
series1.head(2) #with the number it will return the first 2 rows from the object.
```

Out[43]:
```
11    13
12     2
dtype: int64
```

In [45]:
```
series1.tail() # without passing the number in the tail function by default it will
```

Out[45]:
```
11    13
12     2
13    26
14    26
15     5
dtype: int64
```

In [47]:
```
series1.tail(2) #with the number it will return the last 2 rows from the object.
```

Out[47]:
```
14    26
15     5
dtype: int64
```

In [51]:
```
# vector operations on a series object.

series1 + 2 # it will add 2 on each and every element which are present in the seri
```

Out[51]:
```
11    15
12     4
13    28
14    28
15     7
dtype: int64
```

In [55]:
```
series1 > 10 # by defult it will return the boolean values.
```

Out[55]:
```
11     True
12    False
13     True
14     True
15    False
dtype: bool
```

In [67]:
```
#arithmetic operation on series object.make sure both the objects has same index na
#if size is not match again it will return nan.

school1 = pd.Series(data = [10,20,30],index = ["science","commerce","arts"])
```

```
school2 = pd.Series(data = [40,50,60],index = ["science","commerce","arts"])

print(school1)
print(school2)

total_students = school1 + school2

print("total number of students is:",total_students)
```

```
science     10
commerce    20
arts        30
dtype: int64
science     40
commerce    50
arts        60
dtype: int64
total number of students is: science     50
commerce    70
arts        90
dtype: int64
```

In [73]:
```
# for the average student in each school.

average_students = total_students / 2
print(average_students)
```

```
science     25.0
commerce    35.0
arts        45.0
dtype: float64
```

In [81]:
```
# filtering entries in series object.

series1>13 # without bracket it will return the boolean values only.
```

Out[81]:
```
11    False
12    False
13     True
14     True
15    False
dtype: bool
```

In [85]:
```
# filtering entries in series object.

series1[series1>13] #with bracket it will return the actual values from the series
```

Out[85]:
```
13    26
14    26
dtype: int64
```

In [109…
```
info = pd.Series(data = [25,15,30])
print(info)
print(info>15) # vectorized operation result
print(info[info>10]) # filtering result
```

```
0    25
1    15
2    30
dtype: int64
0     True
1    False
2     True
dtype: bool
0    25
1    15
2    30
dtype: int64
```

In [111…
```
# sorting series on the basis of the values. by default it will return ascending or

info.sort_values()
```

Out[111…
```
1    15
0    25
2    30
dtype: int64
```

In [119…
```
info.sort_values(ascending = False) # false is capital for the return descending va
```

Out[119…
```
2    30
0    25
1    15
dtype: int64
```

In [121…
```
# sorting the values on the basis of the indexes.

sindex = pd.Series(data = [10,20,30],index = [35,45,25])
print(sindex)
```

```
35    10
45    20
25    30
dtype: int64
```

In [125…
```
sindex.sort_index() # sorting the series on the basis of the indexes. by default it
```

Out[125…
```
25    30
35    10
45    20
dtype: int64
```

In [127…
```
sindex.sort_index(ascending = False)  # false is capital for the return descending
```

Out[127…
```
45    20
35    10
25    30
dtype: int64
```

In [ ]: