pandas.Series

**Pandas Series** is a One Dimensional indexed array. It is most similar to the NumPy array. **[pandas.Series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html" \t "_blank)** is a method to create a series.

Here practically explanation about **Series**.  
For using pandas library in Jupyter Notebook IDE or any Python IDE or IDLE, we need to import Pandas, using the [**import**](https://docs.python.org/3/reference/import.html)keyword

|  |  |
| --- | --- |
| 1 | import pandas as pd |

Here we are using [**as**](https://www.python.org/dev/peps/pep-0221/)keyword to short pandas name as “**pd**“

The latest version of Pandas Library is 0.24.2 released on 12 March 2019. To know the version of Jupyter Notebook IDE

|  |  |
| --- | --- |
| 1 | pd.\_\_version\_\_ |
| 1 | Output >>>  '0.24.2' |

Series is similar to [**python list**](https://docs.python.org/3/tutorial/introduction.html#lists) but series have additional functionality, methods, and operators, because of these series is advanced than a list.

**Methods of Creating a Series**

**1. Creating series from list**

but first, we are creating a list

|  |  |
| --- | --- |
| 1  2 | list\_1 = [1, 2, -3, 4.5, 'indian']  print(list\_1) |
| 1 | Output >>>   [1, 2, -3, 4.5, 'indian'] |

Python list stores int, float, string data types

**Creating series using the above list**

|  |  |
| --- | --- |
| 1  2 | series1 = pd.Series(list\_1)  print(series1) |
| 1  2  3  4  5  6  7 | Output >>>            0         1            1         2            2        -3            3       4.5            4    indian            dtype: object |

Here it is showing **0 1 2 3 4** is index and **1 2 -3 4.5** Indian are data values.

|  |  |
| --- | --- |
| 1 | type(series1) |
| 1 | Output >>>   pandas.core.series.Series |

**pandas.core.series.Series** means series is a one-dimensional array, which can store indexed data

**2. Creating Empty Series**

Empty series is like an empty list, we can create empty series using an empty list

|  |  |
| --- | --- |
| 1  2 | empty\_s = pd.Series([])  print(empty\_s) |
| 1 | Output >>>   Series([], dtype: float64) |

**3. Creating Series using a different method**  
List inside the series

|  |  |
| --- | --- |
| 1  2 | series2 = pd.Series([1,2,3,4,5])  print(series2) |
| 1  2  3  4  5  6  7 | Output >>>            0    1            1    2            2    3            3    4            4    5            dtype: int64 |

in index parameter, default index is start from 0 to n (0,1,2,….n) when index is not identified  
Here we are creating series with the index parameter  
Index length should have equal to the number of data values, otherwise, it shows error

|  |  |  |
| --- | --- | --- |
| 1  2 | series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c'])  print(series2) | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | | Output >>>  ---------------------------------------------------------------------------  ValueError                                Traceback (most recent call last)  <ipython-input-11-c6475a37a2e3> in <module>  ----> 1 series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c'])        2 print(series2)    C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in \_\_init\_\_(self, data, index, dtype, name, copy, fastpath)      247                             'Length of passed values is {val}, '      248                             'index implies {ind}'  --> 249                             .format(val=len(data), ind=len(index)))      250                 except TypeError:      251                     pass    ValueError: Length of passed values is 5, index implies 3 | |

**ValueError: Length of passed value is 5, index implies 3**We got an error because we passed 3 indexes for 5 data values

|  |  |
| --- | --- |
| 1  2 | series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'])  print(series2) |
| 1  2  3  4  5  6  7 | Output >>>            a    1            b    2            c    3            d    4            e    5            dtype: int64 |

We can change index to any numbers, alphabates, names etc.

Above you can see **dtype: int64**, this means our data type has stored in integer 64 bit.  
we can change the data type of series  
**Changing data type of series (Convert int into a float)**

|  |  |  |
| --- | --- | --- |
| 1  2 | series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'], dtype = float)  print(series2) | |
| 1  2  3  4  5  6  7 | Output >>>            a    1.0            b    2.0            c    3.0            d    4.0            e    5.0            dtype: float64 |

**4. Creating series from scalar values**  
scalar values means single value  
e.g. 1, 0.5, ‘indian’

|  |  |
| --- | --- |
| 1  2 | s3\_scalar = pd.Series(2)  print(s3\_scalar) |
| 1  2  3 | Output >>>            0    2            dtype: int64 |

for more data values index should be needed.

|  |  |
| --- | --- |
| 1  2 | s3\_scalar = pd.Series(2, index = [1,2,3,4,5])  print(s3\_scalar) |
| 1  2  3  4  5  6  7 | Output >>>            1    2            2    2            3    2            4    2            5    2            dtype: int64 |

**5. Creating series from python dictionary**

|  |  |
| --- | --- |
| 1  2 | s4\_dict = pd.Series({'a':1, 'b':2, 'c':3})  print(s4\_dict) |
| 1  2  3  4  5 | Output >>>            a    1            b    2            c    3            dtype: int64 |

**Accessing element from series**

Pandas Series supports most Python functions.  
Now, we are accessing element from series2

|  |  |
| --- | --- |
| 1 | print(series2) |
| 1  2  3  4  5  6  7 | Output >>>            a    1.0            b    2.0            c    3.0            d    4.0            e    5.0            dtype: float64 |

We can access any value or data from series by putting index value

|  |  |
| --- | --- |
| 1 | series2[3] |
| 1  2 | Output >>>            4.0 |

|  |  |
| --- | --- |
| 1 | series2[4] |
| 1  2 | Output >>>            5.0 |

**Slicing series**

Here we are slicing series with index value 1 to 4 that means 1 is inclusive(it can be taken) and 4 is exclusive(it can be not taken)

|  |  |
| --- | --- |
| 1 | series2[1:4] |
| 1  2  3  4  5 | Output >>>            b    2.0            c    3.0            d    4.0            dtype: float64 |

series can be done by using mathematical operators

**Adding two serieses**

|  |  |
| --- | --- |
| 1  2  3  4  5 | s5 = pd.Series([1,2,3,4,5])  s6 = pd.Series([1,2,3,4,5])    a = s5 + s6  print(a) |
| 1  2  3  4  5  6  7 | Output >>>            0     2            1     4            2     6            3     8            4    10            dtype: int64 |

**we can also add series using add method**

|  |  |
| --- | --- |
| 1 | s5.add(s6) |
| 1  2  3  4  5  6  7 | Output >>>            0     2            1     4            2     6            3     8            4    10            dtype: int64 |

**min() operator gives minimum value of particular series**

|  |  |
| --- | --- |
| 1 | min(a) |
| 1 | Output >>>   2 |

**max() operator gives maximum value**

|  |  |
| --- | --- |
| 1 | max(a) |
| 1 | Output >>>   10 |

**Conditional operator**

If you want to print less than 8 values

|  |  |
| --- | --- |
| 1 | a[a < 8] |
| 1  2  3  4  5 | Output >>>            0    2            1    4            2    6            dtype: int64 |

Using **drop()** function we can eliminate any index value

|  |  |
| --- | --- |
| 1 | a.drop(4) |
| 1  2  3  4  5  6 | Output >>>            0    2            1    4            2    6            3    8            dtype: int64 |

Now we are printing series6 (s6)

|  |  |
| --- | --- |
| 1 | print(s6) |
| 1  2  3  4  5  6  7 | Output >>>            0    1            1    2            2    3            3    4            4    5            dtype: int64 |

|  |  |
| --- | --- |
| 1  2 | s7 = pd.Series([1,2,3])  print(s7) |
| 1  2  3  4  5 | Output >>>            0    1            1    2            2    3            dtype: int64 |

Pandas have additional functions to fill missing values, it does not show an error when the value is missing. Missing values are shown by NaN.  
See below example:

In pandas, we can add the unequal data values series. Here series s6 have 5 data values and s7 have 3 data values, when we perform addition operation it adds successfully

|  |  |
| --- | --- |
| 1 | s6 + s7 |
| 1  2  3  4  5  6  7 | Output >>>            0    2.0            1    4.0            2    6.0            3    NaN            4    NaN            dtype: float64 |