

Introduction to Data Structures

Pandas deals with the following three data structures:

- Series: 1D labeled homogeneous array, size immutable
- DataFrame: General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
- Panel: General 3D labeled, size-mutable array.

These data structures are built on top of Numpy array, which means they are fast.

For example, with tabular data (DataFrame) it is more semantically helpful to think of the **index** (the rows) and the **columns** rather than axis 0 and axis 1.

Mutability

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

pandas.Series

A pandas Series can be created using the following constructor –

```
pandas.Series( data, index, dtype, copy)
```

The parameters of the constructor are as follows –

Sr.No	Parameter & Description
1	data data takes various forms like ndarray, list, constants
2	index Index values must be unique and hashable, same length as data. Default np.arange(n) if no index is passed.
3	dtype dtype is for data type. If None, data type will be inferred

4

copy

Copy data. Default False

A **series** can be created using various inputs like –

- Array
- Dict
- Scalar value or constant

Create an Empty Series

A basic series, which can be created is an Empty Series.

Example:

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

Its output is as follows –

```
Series([], dtype: float64)
```

Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., **[0,1,2,3.... range(len(array))-1]**.

Example 1

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Its output is as follows –

```
0    a
```

```
1    b
2    c
3    d
dtype: object
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to **len(data)-1**, i.e., 0 to 3.

Example 2

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a', 'b', 'c', 'd'])
s = pd.Series(data, index=[100, 101, 102, 103])
print s
```

Its output is as follows –

```
100    a
101    b
102    c
103    d
dtype: object
```

We passed the index values here. Now we can see the customized indexed values in the output.

Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

Example 1

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

Its output is as follows –

```
a    0.0
```

```
b 1.0
c 2.0
dtype: float64
```

Observe – Dictionary keys are used to construct index.

Example 2

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print s
```

Its output is as follows –

```
b 1.0
c 2.0
d NaN
a 0.0
dtype: float64
```

Observe – Index order is persisted and the missing element is filled with NaN (Not a Number).

Create a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**

[Live Demo](#)

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

Its output is as follows –

```
0    5
1    5
2    5
3    5
dtype: int64
```

Accessing Data from Series with Position

Data in the series can be accessed similar to that in an **ndarray**.

Example 1

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

[Live Demo](#)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element
print s[0]
```

Its **output** is as follows –

1

Example 2

Retrieve the first three elements in the Series. If a **:** is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with **:** between them) is used, items between the two indexes (not including the stop index)

[Live Demo](#)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element
print s[:3]
```

Its **output** is as follows –

```
a    1
b    2
c    3
dtype: int64
```

Example 3

Retrieve the last three elements.

[Live Demo](#)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
```

```
#retrieve the last three element
print s[-3:]
```

Its output is as follows –

```
c    3
d    4
e    5
dtype: int64
```

Retrieve Data Using Label (Index)

A Series is like a fixed-size **dict** in that you can get and set values by index label.

Example 1

Retrieve a single element using index label value.

[Live Demo](#)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element
print s['a']
```

Its output is as follows –

```
1
```

Example 2

Retrieve multiple elements using a list of index label values.

[Live Demo](#)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements
print s[['a','c','d']]
```

Its output is as follows –

```
a    1
c    3
d    4
dtype: int64
```

Example 3

If a label is not contained, an exception is raised.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements
print s['f']
```

Its **output** is as follows –

```
...
KeyError: 'f'
```