What is Pandas?

- Pandas stands for **Panel Data**, a reference to the tabular format.
- Pandas is an open-source library that is used for data analysis.
- It allows efficient data cleaning and manipulation operations.
- Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas.

Why Pandas? (Features of Pandas)

1. Easy import and export of data into a tabular format.
2. Fast and efficient for manipulating and analyzing of data.
3. Easy handling of missing data, generally represented as NaN (Not a Number).
4. Enables dataset sorting, merging, ranking and joining operations.
5. Flexible reshaping and pivoting (re-arranging data based on rows or columns) of data sets.
6. Group-By functionality to perform split-apply-combine operations.
7. Well integrated with other libraries like NumPy and Matplotlib.

How to use Pandas?

To use Pandas it is required to import the Pandas module:

> **import pandas as pd**

Here, pd is an alias (rename) to the pandas.

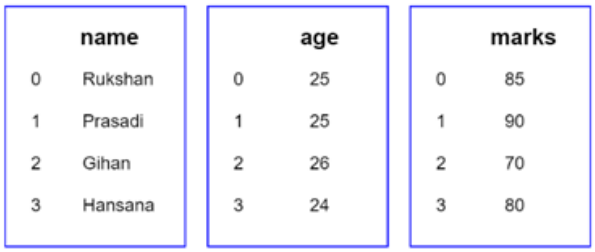
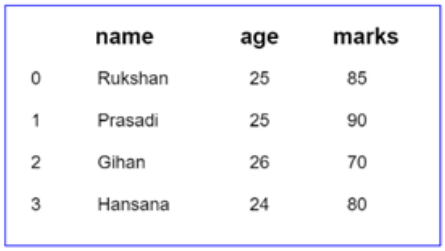
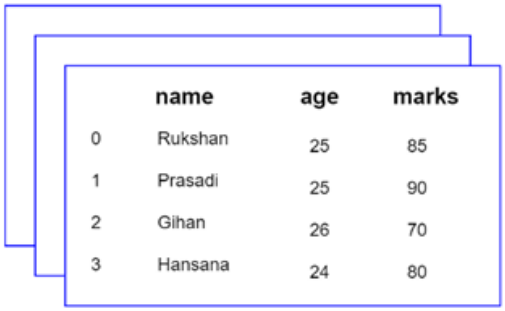
Note:

You can choose any other name too for the alias, but pd is accepted as default alias industry wise and in most of the source codes you will find pd as the pandas alias.

Pandas Data Structures

- A data structure is a collection of data values.
- It defines the relationship between the data, and the operations that can be performed on the data.
- **Pandas mainly provide three data structures for data manipulating and analysis. They are:**

> 1. Series
> 2. Dataframe
> 3. Panel

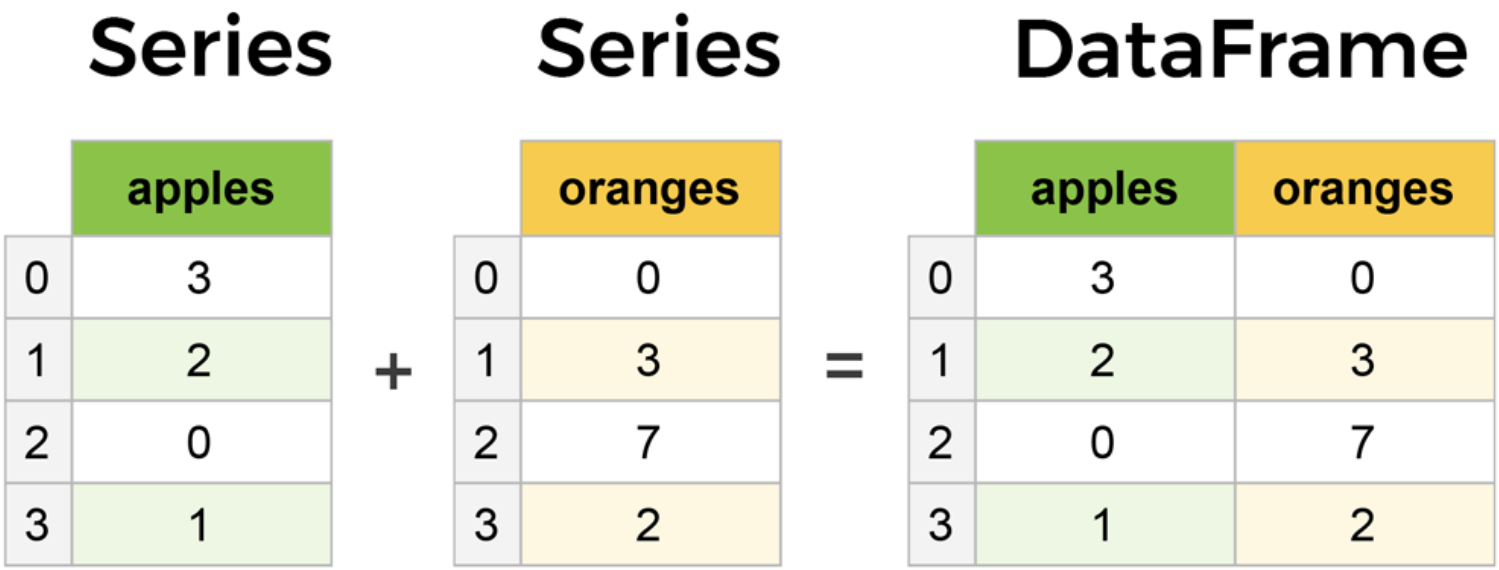
Data Structure	Dimensionality	Format	View
Series	1D	Column	
DataFrame	2D	Single Sheet	
Panel	3D	Multiple Sheets	

Note:

The most widely used pandas data structures are the Series and the DataFrame.

Simply, a Series is similar to a single column of data while a DataFrame is similar to a sheet with rows and

Likewise, a Panel can have many DataFrames.



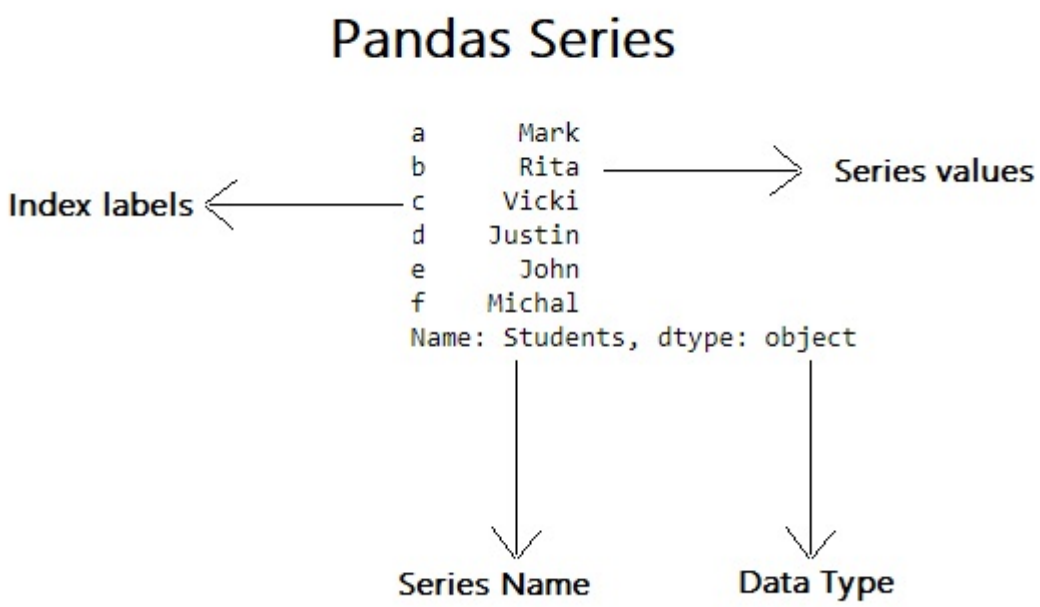
1. Series

- The Pandas module provides a one-dimensional data structure called Series in Python.
- It is like a one-dimensional labeled array that can store elements of different data types (integer, string, float, etc.)
- Each value in the Series has a label/index associated with it.
- A Series is essentially a column.
- A series consists of two components.

> Data/Values

> Labels/Index

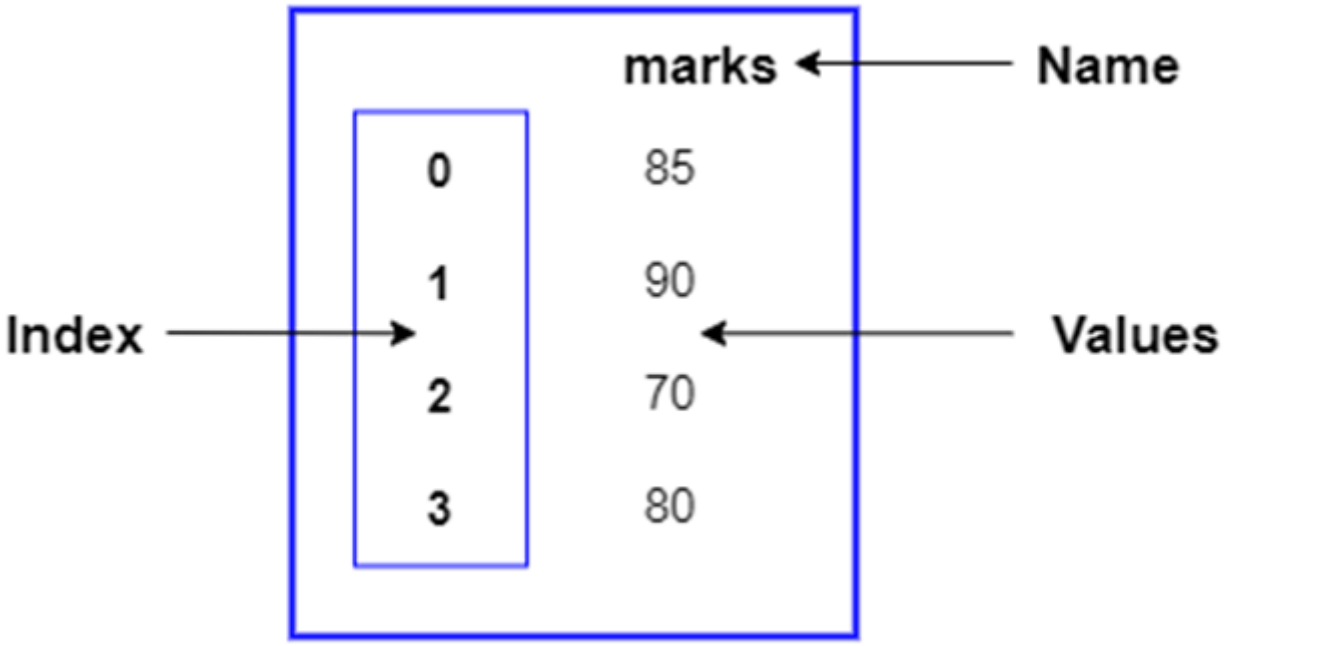
Figure 1:



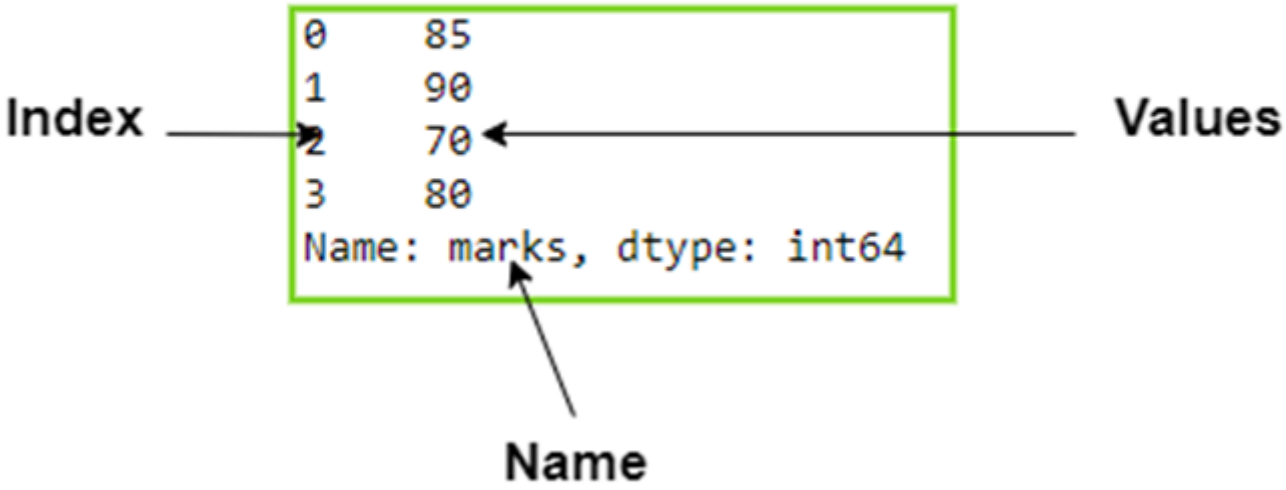
- Elements in the **right-hand side column are the Series actual Values**.
- Elements in the **left-hand side column are the index or labels associated with each value.**
- We can access values from the Series using the label name.

Think of Series like a column in an Excel file. In Excel, each cell box in the column has a row label associated with it, similar to that each value in a Series has a label associated with it.

Figure 2:



pd.Series([85, 90, 70, 80], name='marks')



How to Create Series?

- The Pandas module provides a function Series(), which accepts a data as the argument and returns a Series object containing the given elements.

Syntax:

> **pandas.Series(data=None, index=None, dtype=None, name=None)**

Parameters:

> **data:** values to be stored in Series.

> **index:** labels or ID for each value (optional)

> **dtype:** str, numpy.dtype (optional)

> **name:** str (optional)

- In Series() the data can be:
    1. A Python list, tuple or dictionary.
    2. A NumPy ndarray
    3. A scalar value (any number, like 6)

Note:

- *These are called Methods of creating Series.*

Note:

In the real project, a Pandas Series will be created by loading the datasets from existing storage. These storage can be SQL Database, CSV file, an Excel file.

Method 1: To create a series using list or tuple or dictionary

List is passed as parameter in Series().

The variable declared for Series() is called as Series object.

This creates a Series with items in the list as Values.

Each Values are aasigned with default index labels with it.

By default the index label is numeric and starts from 0.

```
import pandas as pd

# Create a Series object from a LIST
a = pd.Series(['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'])

# Display the Pandas series
print(a)
```

```
0      Mark
1      Rita
2     Vicki
3    Justin
4      John
5    Michal
dtype: object
```

```
import pandas as pd

# Create a Series object from a TUPLE
a = pd.Series(('Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'))

# Display the Pandas series
print(a)
```

```
0      Mark
1      Rita
2     Vicki
```

```
3    Justin
4      John
5    Michal
dtype: object
```

To create a Series from dictionary:

- All the keys from dictionary will be used as index labels for the Series object
- All the value fields from dictionary will be used as values for the Series object.

```
import pandas as pd
D = { 'a': 'Mark', 'b': 'Rita', 'c': 'Vicki', 'd': 'Justin', 'e': 'John', 'f': 'Michal'}

# Create a Series object from a DICTIONARY
a = pd.Series(D)

# Display the Pandas series object
print(a)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object
```

Method 2: To create a series using NumPy ndarray

We can pass a numpy array to the Series() function to get a Series.

```
import pandas as pd
import numpy as np

# Array of numbers
a = np.array([100, 200, 300, 400, 500, 600])

# Create a Series object from a NumPy Array
b = pd.Series(a, index = ['a', 'b', 'c', 'd', 'e', 'f'])

# Display the Pandas series object
print(b)
```

```
a    100
b    200
c    300
d    400
e    500
f    600
dtype: int64
```

```
import pandas as pd
import numpy as np
```

```
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])

print(s)
```

```
a    0.252610
b    1.478915
c    0.176634
d    0.867737
e   -2.074245
dtype: float64
```

Method 3: To create a series using Scalar Values

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

```
a = pd.Series(2)

print(a)
print("\n")

b = pd.Series(5.0, index=["a", "b", "c", "d", "e"])

print(b)
```

```
0    2
dtype: int64


a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

How to set the custom index in Series?

Using index parameter in Series()

> Pass the index parameter with label names in the Series() function for the custom index labels.

> A Series object contains the labeled values and it is like a single column of Excel file.

```
import pandas as pd

# Create a Series object from a list
a = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
              index = ['a', 'b', 'c', 'd', 'e', 'f'])

# Display the Pandas series
print(a)
```

```
a      Mark
b      Rita
```

```
c     Vicki
d    Justin
e      John
f    Michal
dtype: object
```

It returned a Series object, where index labels are custom string values. In this Series object, each value has a custom label i.e.,

Value ‘Mark’ has an index label ‘a’

Value ‘Rita’ has an index label ‘b’

Value ‘Vicki’ has an index label ‘c’

Value ‘Justin’ has an index label ‘d’

Value ‘John’ has an index label ‘e’

Value ‘Michal’ has an index label ‘f’

How to set the custom data type of values?

Using dtype parameter in Series()

```
import pandas as pd
import numpy as np

# Array of numbers
a = np.array([100, 200, 300, 400, 500, 600])

# Create a Series object from a NumPy Array
b = pd.Series(  a, index = ['a', 'b', 'c', 'd', 'e', 'f'], dtype = float)

# Display the Pandas Series object
print(b)
```

```
a    100.0
b    200.0
c    300.0
d    400.0
e    500.0
f    600.0
dtype: float64
```

How to Create Series with mixed data type values?

Here, we created a Series object where values are of integer type and labels are of string type.

```
import pandas as pd

# Create a Series object with mixed data type values
```

```
a = pd.Series(  ['Mark', 100, 'Tokyo', 89.22])
print(a)
```

```
0     Mark
1      100
2    Tokyo
3    89.22
dtype: object
```

```
import pandas as pd

# a simple char list
a = ['g', 'r', 'e', 'a', 't']

# create series form a char list
b = pd.Series(a)
print(b)
```

```
0    g
1    r
2    e
3    a
4    t
dtype: object
```

How to set the Name of the Series?

Using Name parameter in Series()

> Similar to column name in Excel, Series also has a name associated with it.

```
import pandas as pd

# Create a Series object from a list
x = pd.Series(['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
              index = ['a', 'b', 'c', 'd', 'e', 'f'],
              name = "Students")

# Display the Pandas Series
print(x)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
Name: Students, dtype: object
```

We can access the name of the Series object using the name property of the Series.

```
# Display the name attribute of the Series Object
print(x.name)
```

```
Students
```

We can also change the name of the existing Series object using name property.

```
x.name = 'Users'
# Display the Pandas Series
print(x)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
Name: Users, dtype: object
```

To get count of number of elements in Series

The Series object provides a property size, which returns the count of number of elements on the Series.

```
# Get the count of elements in Series
print(x.size)
```

```
6
```

To Check if Series is empty or not

The Series object provides a **property empty**.

It returns True if Series is empty, otherwise returns False.

```
import pandas as pd
# Create a Series object from a list
x = pd.Series(['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
              index = ['a', 'b', 'c', 'd', 'e', 'f'],
              name = "Students")

# check if series is empty or not
print(x.empty)
```

```
False
```

To first N elements of Pandas Series

The Series object provides a function **head()**.

It returns the first n values of the Series object.

```
import pandas as pd
# Create a Series object from a list
```

```
a = pd.Series(['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
              index = ['a', 'b', 'c', 'd', 'e', 'f'],
              name = "Students")

# Get first 3 elements of series
subset = a.head(3)

# Display the Subset of Series
print(subset)
```

```
a     Mark
b     Rita
c    Vicki
Name: Students, dtype: object
```

To get last N elements of Pandas Series

The Series object provides a function **tail()**.

It returns the last n values of the Series object.

```
import pandas as pd
# Create a Series object from a list
users = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'],
                    name = "Students")

# Get last 3 elements of series
subset = users.tail(3)

# Display the Subset of Series
print(subset)
```

```
d    Justin
e      John
f    Michal
Name: Students, dtype: object
```

To get the count of non NaN (Not a Number) values

The Series object provides a function **count()**.

It returns the count of non (Not a Number) NaN values in the Series object.

```
import pandas as pd
import numpy as np

# Create a Series object from a list
y = pd.Series(['Mark', np.NaN, 'Vicki', 'Justin', np.NaN, 'Michal'])
print(y)
print("\n")
```

```
# Get count of non NaN values in Pandas Series
z = y.count()
print(z)
```

```
0      Mark
1       NaN
2     Vicki
3    Justin
4       NaN
5    Michal
dtype: object


4
```

Indexing Series elements / Accessing Series Elements

We can access elements in Series by:

1. Positional indexing
2. Label names

A) Accessing Series elements using Positional Indexing

- Indexing in Python starts from 0.
- It means if Series contains N elements then,

  > 1st element has index position 0
  >
  > 2nd element has index position 1
  >
  > 3rd element has index position 2
  >
  > .....
  >
  > .....
  >
  > Nth element has index position N-1

- To access elements in Series by the index position, pass the index position in the subscript operator with the Series object.
- It will return the value at that index position.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
```

```
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)

# Access first element of the Series object
first_element = names[0]
print('First Element: ', first_element)

# Access 3rd element of the Series object
third_element = names[2]
print('Third Element: ', third_element)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object
First Element:  Mark
Third Element:  Vicki
```

Access multiple elements of Series by specific index positions

We can also pass a list of index positions in the subscript operator of the Series object.

It will return a Series object containing the specified elements only.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print("\n")

# Select elements at index position 2, 3 and 0 only
few_names = names[[2, 3, 0]]

# Display the subset of Series
print(few_names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


c     Vicki
d    Justin
a      Mark
dtype: object
```

It selected the values at index position 2, 3 and 0 only.

B) Access Series elements using Label names

- To access elements in Series by the label name, pass the label name in the subscript operator of the Series object. It will return the value associated with the label.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

# Access element with label 'd'
print( names['d'] )
```

```
Justin
```

Access multiple elements of Series by specific label names

We can also pass a list of label names in the subscript operator of the Series object. It will return a Series object containing the specified elements only.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print("\n")

# Select elements at index labels 'd', 'e' and 'a'
few_names = names[['d', 'e', 'a']]

# Display the subset of Series
print(few_names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


d    Justin
e      John
a      Mark
dtype: object
```

It selected the values with label ‘d’, ‘e’ and ‘a’.

C) Access subset of Series using Index / Label Range (Slicing)

- Using slicing, we can access a range of elements from the series object i.e.

  > seriesObject[start : end]

> It will given an access to Series elements from index position start to end-1.

> OR

> It will given an access to Series elements from label position start to end-1.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print("\n")

# Select elements from index position 1 till 3
few_names = names[1:4]

# Display the subset of Series
print(few_names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


b      Rita
c     Vicki
d    Justin
dtype: object
```

Similarly we can provided the label range instead of index range.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print("\n")

# Select elements from index label 'b' till label 'e'
few_names = names['b' : 'e']
```

```
# Display the subset of Series
print(few_names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


b      Rita
c     Vicki
d    Justin
e      John
dtype: object
```

Changing elements in the Series

When we access Series elements using the subscript operator, we can directly use that to change the content of the Series object.

A) Change single element in Series by index position

Access the element at specified index position using subscript operator and directly assign new value to it.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print('\n')

# Change the 3rd value of Series
names[2] = 'Sanjay'

# Display the Series
print(names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


a      Mark
b      Rita
c    Sanjay
```

```
d    Justin
e      John
f    Michal
dtype: object
```

B) Change single element in Series by label value

Access the element by specifying the label name using subscript operator and directly assign new value to it.

```
import pandas as pd

# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])

print(names)
print('\n')

# Change the value at label 'e'
names['e'] = 'Harsha'

# Display the Series
print(names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object


a      Mark
b      Rita
c     Vicki
d    Justin
e    Harsha
f    Michal
dtype: object
```

C) Change multiple element in Series

Access multiple elements using index range or label range using subscript operator and directly assign new values to it.

```
import pandas as pd

# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])
```

```
print(names)
print()

# Change the first three values to same value
names[0 : 3] = 'John Doe'

# Display the Series
print(names)
print()

# Change the values from label 'a' till 'd' to same value
names['a' : 'd'] = 'Smriti'

# Display the Series
print(names)
```

```
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object

a    John Doe
b    John Doe
c    John Doe
d      Justin
e        John
f      Michal
dtype: object

a    Smriti
b    Smriti
c    Smriti
d    Smriti
e      John
f    Michal
dtype: object
```

Deleting elements from series

Tthe Series provides a function **drop()**, to delete the elements based on index labels.

It accepts a list of index labels and delete the values associated with those labels.

```
import pandas as pd
# Create a Series object from a list
names = pd.Series(  ['Mark', 'Rita', 'Vicki', 'Justin', 'John', 'Michal'],
                    index = ['a', 'b', 'c', 'd', 'e', 'f'])
print('Original Series: ')
print(names)
print()

# Delete elements at given index labels
names = names.drop(['b', 'c', 'e'])
```

```
print('Modified Series: ')
print(names)
```

```
Original Series:
a      Mark
b      Rita
c     Vicki
d    Justin
e      John
f    Michal
dtype: object

Modified Series:
a      Mark
d    Justin
f    Michal
dtype: object
```

Additional Points:

Write a Pandas program to add, subtract, multiple and divide two Pandas Series.

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 9])
ds = ds1 + ds2
print("Add two Series:")
print(ds)
print("Subtract two Series:")
ds = ds1 - ds2
print(ds)
print("Multiply two Series:")
ds = ds1 * ds2
print(ds)
print("Divide Series1 by Series2:")
ds = ds1 / ds2
print(ds)
```

```
Add two Series:
0     3
1     7
2    11
3    15
4    19
dtype: int64
Subtract two Series:
```

```
0    1
1    1
2    1
3    1
4    1
dtype: int64
Multiply two Series:
0     2
1    12
2    30
3    56
4    90
dtype: int64
Divide Series1 by Series2:
0    2.000000
1    1.333333
2    1.200000
3    1.142857
4    1.111111
dtype: float64
```

Write a Pandas program to compare the elements of the two Pandas Series.

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 10])
print("Series1:")
print(ds1)
print("Series2:")
print(ds2)
print("Compare the elements of the said Series:")
print("Equals:")
print(ds1 == ds2)
print("Greater than:")
print(ds1 > ds2)
print("Less than:")
print(ds1 < ds2)
```

```
Series1:
0     2
1     4
2     6
3     8
4    10
dtype: int64
Series2:
0     1
1     3
2     5
3     7
4    10
dtype: int64
Compare the elements of the said Series:
Equals:
0    False
1    False
2    False
```

```
3    False
4     True
dtype: bool
Greater than:
0     True
1     True
2     True
3     True
4    False
dtype: bool
Less than:
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

Adding/Merging Series together

The Series provides a function add() to merge two Series object i.e.

> Series.add(other, fill_value=None)

It accepts another Series as an argument and merges all the elements of that Series to the calling Series object.

As Series values are labeled, therefore while merging, elements with same labels will be added together (binary add) and values with unique labels will be added independently.

It returns a new Series object with the merged content.

```
import pandas as pd
# Create first Series object from a list
first = pd.Series(  [100, 200, 300, 400, 500],
                    index = ['a', 'b', 'e', 'f', 'g'])

# Create second Series object from a list
second = pd.Series( [11, 12, 13, 14],
                    index = ['a', 'b', 'h', 'i'])

# Add two Series objects together
total = first.add(second)

# DIsplay the Series object
print(total)
```

```
a    111.0
b    212.0
e      NaN
f      NaN
g      NaN
h      NaN
i      NaN
dtype: float64
```

As label ‘a’ is in both the Series, so values from both the Series got added together and final value became 111.

As label ‘b’ is in both the Series, so values from both the Series got added together and final value became 212.

As label ‘e’ is in first Series only, therefore it got added in new Series as NaN.

As label ‘f’ is in first Series only, therefore it got added in new Series as NaN.

As label ‘g’ is in first Series only, therefore it got added in new Series as NaN.

As label ‘h’ is in second Series only, therefore it got added in new Series as NaN.

As label ‘i’ is in second Series only, therefore it got added in new Series as NaN.

Note:

Values with similar labels got added together, but values with unique labels got added as NaN. What if we want to keep the original values for them too? How to do that?

For that we need to use the fill_value parameter of the add() function. If provided then while adding it uses the given value for the missing(NaN) entries. So if we provide fill_value=0 in the add() function, it will use value 0 for the missing labels, while adding the Series objects.

```
import pandas as pd
# Create first Series object from a list
first = pd.Series(  [100, 200, 300, 400, 500],
                    index = ['a', 'b', 'e', 'f', 'g'])

# Create second Series object from a list
second = pd.Series( [11, 12, 13, 14],
                    index = ['a', 'b', 'h', 'i'])

# Add two Series objects together
total = first.add(second, fill_value=0)

# DIsplay the Series object
print(total)
```

```
    a    111.0
    b    212.0
    e    300.0
    f    400.0
    g    500.0
    h     13.0
    i     14.0
    dtype: float64
```

As label ‘a’ is in both the Series, so values from both the Series got added together and final value became 111.

As label ‘b’ is in both the Series, so values from both the Series got added together and final value became 212.

As label ‘e’ is in first Series only, so for the second Series it used the default value from fill_value i.e. 0 and final value became 300.

As label ‘f’ is in first Series only, so for the second Series it used the default value from fill_value i.e. 0 and final value became 400.

As label ‘g’ is in first Series only, so for the second Series it used the default value from fill_value i.e. 0 and final value became 500.

As label ‘h’ is in second Series only, so for the first Series it used the default value from fill_value i.e. 0 and final value became 13.

As label ‘i’ is in second Series only, so for the first Series it used the default value from fill_value i.e. 0 and final value became 14.

Similarly, if we have any NaN values in any of the Series object and fill_value is provided, then the default value will be used instead of NaN while adding the Series objects.

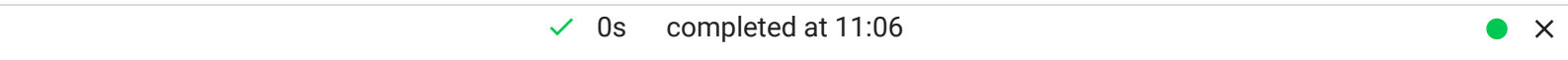
```
import pandas as pd
import numpy as np
# Create first Series object from a list
first = pd.Series(  [100, 200, 300, 400, 500],
                    index = ['a', 'b', 'e', 'f', 'g'])

# Create second Series object from a list
second = pd.Series( [11, np.NaN, 13, 34],
                    index = ['a', 'b', 'h', 'i'])

# Add two Series objects together
total = first.add(second, fill_value=0)

# DIsplay the Series object
print(total)
```

```
a    111.0
b    200.0
e    300.0
f    400.0
g    500.0
h     13.0
i     34.0
dtype: float64
```


0s
completed at 11:06