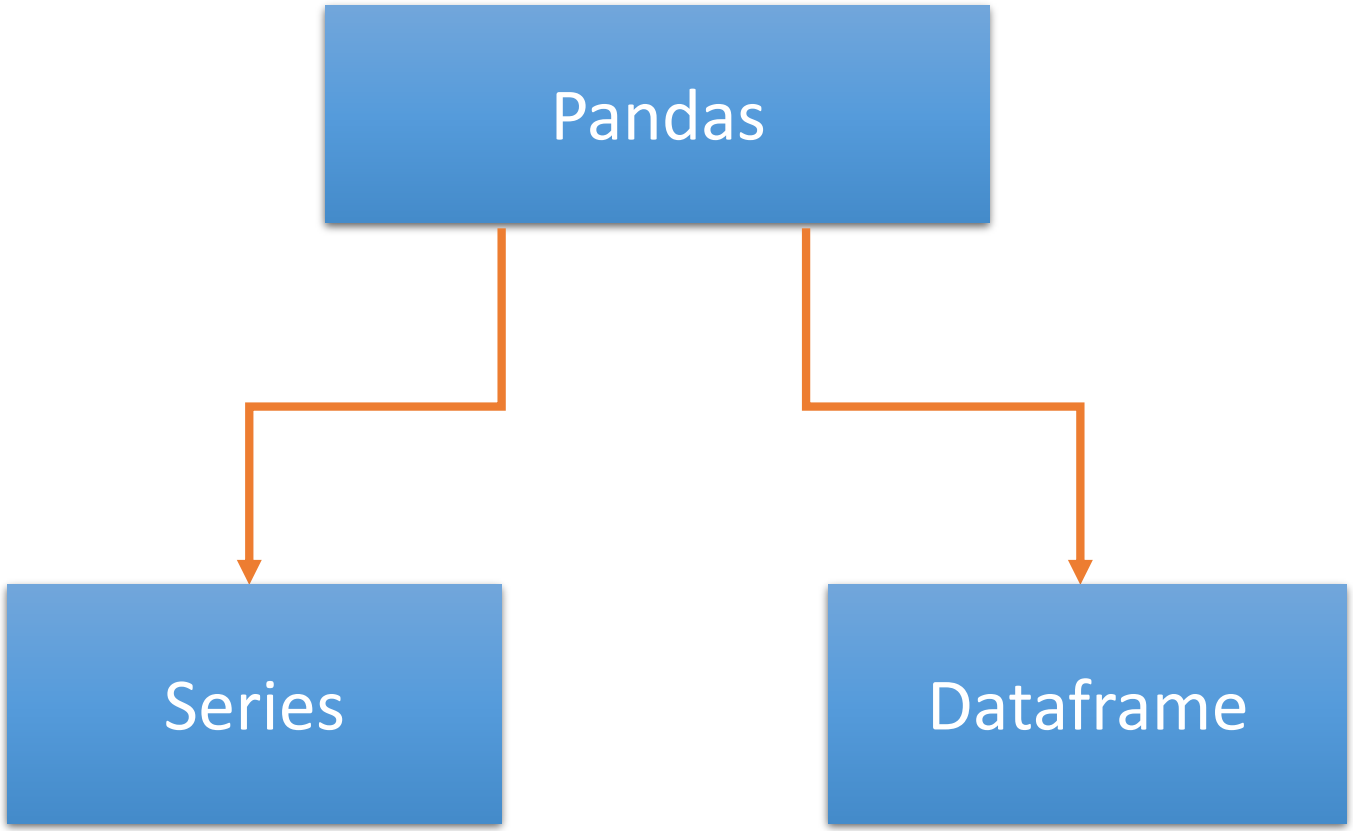
PANDAS

BASIC DATA MANIPULATION

What is Pandas?

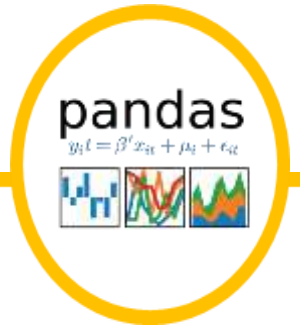
- Python's pandas library is one of the things that makes Python a great programming language for data analysis.
- Pandas makes importing, analyzing, and visualizing data much easier.
- It builds on packages like NumPy and matplotlib to give you a single, convenient, place to do most of your data analysis and visualization work.


Pandas
Series
Dataframe

Series

The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types. We can easily convert the list, tuple, and dictionary into series using "**series**' method.

The row labels of series are called the index. A Series cannot contain multiple columns.



It has the following parameter:

1. **data:** It can be any list, dictionary, or scalar value.

2. **index:** The value of the index should be unique. It must be of the same length as data. If we do not pass any index, default **np.arrange(n)** will be used.

3. **dtype:** It refers to the data type of series.

4. **copy:** It is used for copying the data.

In Python, we are used to working with lists as such:

```
exampleList = [1, 2, 3, 4, 5]
```

The Series data structure in Pandas is the equivalent of a list in python. It is a single dimensional data structure, and is represented as a column. A Python list can be converted into a series in Pandas like so:

```
exampleSeries = pd.Series(exampleList)
```

When a series is printed, the output is represented in two columns. The values in the left column is known as the index. If no index is specified, it is automatically generated and ranges from 0 to n-1, where n is the number of elements in the Series.

However, it is possible to assign a custom index to a series by passing the index argument when converting a list to a series.


```
exampleSeriesTwo = pd.Series(exampleList, index = ['A', 'B', 'C',
'D', 'E'])
```

Take note that the index argument must have the same number of elements as the list that is being converted to a series.

Creating a Series:

We can create a Series in two ways:

1. Create an empty Series
2. Create a Series using inputs.

Create an Empty Series:

We can easily create an empty series in Pandas which means it will not have any value.

The syntax that is used for creating an Empty Series:

```
<series object> = pandas.Series()
```

The below example creates an Empty Series type object that has no values and having default datatype, i.e., **float64**.

```
In [1]: import pandas as pd
        x = pd.Series()
        print (x)

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

Creating a Series using inputs:

We can create Series by using various inputs:

- Array
- Dict
- Scalar value

Creating Series from Array:

- Before creating a Series, firstly, we have to import the **numpy** module and then use array() function in the program. If the data is ndarray, then the passed index must be of the same length.

- If we do not pass an index, then by default index of **range(n)** is being passed where n defines the length of an array, i.e., [0,1,2,....**range(len(array))-1**].

Example

In [2]:
```
import numpy as np
array_list = np.array(['P','a','n','d','a','s'])
series_array = pd.Series(array_list)
print(series_array)
```

```
0    P
1    a
2    n
3    d
4    a
5    s
dtype: object
```

Create a Series from dict

- We can also create a Series from dict. **If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index**.

- If index is passed, then values correspond to a particular label in the index will be extracted from the **dictionary**.

Example

In [3]:
```
array_list2 = {'x' : 0., 'y' : 1., 'z' : 2.}
series_array = pd.Series(array_list2)
print (series_array)
```

```
x    0.0
y    1.0
z    2.0
dtype: float64
```

Create a Series using Scalar:

If we take the scalar values, then the index must be provided. The scalar value will be repeated for matching the length of the index.

```
In [4]: scalar = pd.Series(14, index=[0, 1, 2, 3])
        print (scalar)
```

```
0    14
1    14
2    14
3    14
dtype: int64
```

Accessing data from series with Position:

- Once you create the Series type object, you can access its indexes, data, and even individual elements.

- The data in the Series can be accessed similar to that in the ndarray.

```
In [5]: data = pd.Series([1,2,3],index = ['ajab','kajol','shivaji'])
        #retrieve the first element
        print (data[0])
```

```
1
```

Series object attributes

Attributes	Description
Series.index	Defines the index of the Series.
Series.shape	It returns a tuple of shape of the data.
Series.dtype	It returns the data type of the data.
Series.size	It returns the size of the data.
Series.empty	It returns True if Series object is empty, otherwise returns false.
Series.hasnans	It returns True if there are any NaN values, otherwise returns false.
Series.nbytes	It returns the number of bytes in the data.
Series.ndim	It returns the number of dimensions in the data.
Series.itemsize	It returns the size of the datatype of item.

Retrieving Index array and data array of a series object

We can retrieve the index array and data array of an existing Series object by using the attributes index and values.

```
In [6]: x=pd.Series(data=[1,2,3,4])
        y=pd.Series(data=[1.12,11.7,21.4], index=['ajab','kajol','shivaji'])
        print(x.index)
        print(x.values)
        print(y.index)
        print(y.values)

        RangeIndex(start=0, stop=4, step=1)
        [1 2 3 4]
        Index(['ajab', 'kajol', 'shivaji'], dtype='object')
        [ 1.12 11.7  21.4 ]
```