**Chapter 5 : Getting Started with Pandas**

**Book : Python for Data Analysis**

**5.1 Introduction to Pandas Data Structures**

Pandas is an open source Python package that is most widely used for data science / data analysis and machine learning tasks. Pandas contains data structures and data manipulation tools designed to make data cleaning and analysis fast and easy in Python. Pandas is often used in tandem with numerical computing tools like NumPy and SciPy, analytical libraries like stats models and scikit-learn, and data visualization libraries like matplotlib. While pandas adopt many coding idioms from NumPy, the biggest difference is that *pandas is designed for working with tabular or heterogeneous data*. Whereas *NumPy, is best suited for working with homogeneous numerical array data*.

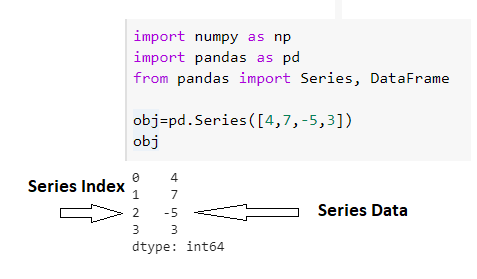
We will use following import convention for pandas: *import pandas as pd*

Therefore, whenever you see pd in code, it’s referring to pandas.

To understand pandas, we need to understand two of its important data structures: *Series* and *DataFrame*. Although these data structures are not a universal solution of every problem, but they provide a solid and easy-to-use basis for most applications. We can import Series and DataFrame into our code as: *from pandas import Series, DataFrame*

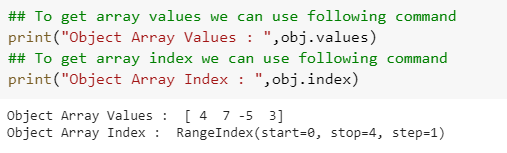
**Series**

A Series is a one-dimensional array-like object containing a sequence of values and an associated array of data labels, called its index. The simplest Series is formed from only an array of data. For Example consider the following code:

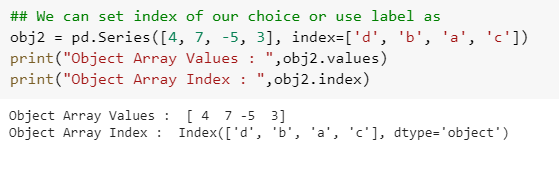


In the above mentioned code we have created a series with four values 4,7,-5,3. Pandas has automatically added index with these values which is 0 for 1st value , 1 for 2nd value and so on.

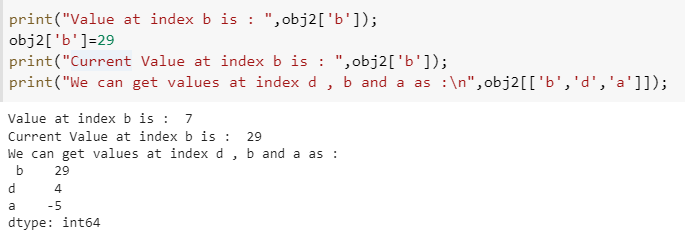
We can get the array values using *objectname.values* and indexes as *objectname.index*. e.g:  
In above example we can get array values as *obj.values* and index values as *obj.index*.



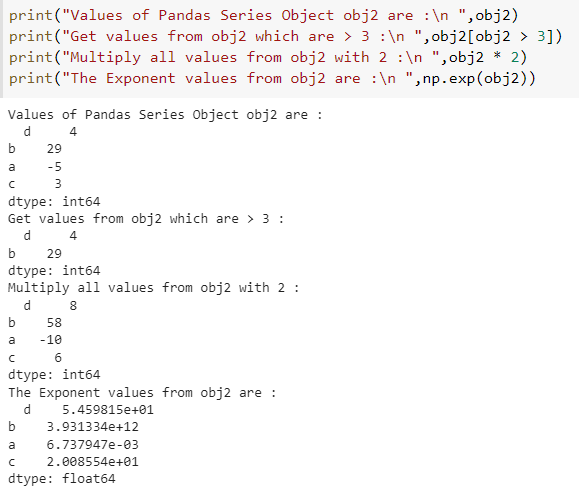
We can also create a Series with an index identifying each data point with a label of our choice. For Example:



We can use labels in the index to get or set single value. We can also specify more than one indexes to get values from pandas series array as:



In Pandas Series we can also perform operations, such as filtering with a Boolean array, scalar multiplication, or applying math functions, will preserve the index-value. So obj2[ obj2 > 5] will return values from series that are greater than 5. Similarly obj2 \* 2 will multiply each value of the series with 2 but the index number remains the same. For Example:

:

Similarly we can also check if a particular label or index in present in a series object as *‘b’ in obj*. It will return True if ‘b’ index is present otherwise will return False.

We can also create a series object if we have data in a Python dictionary as

*sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}  
obj3 = pd.Series(sdata)*

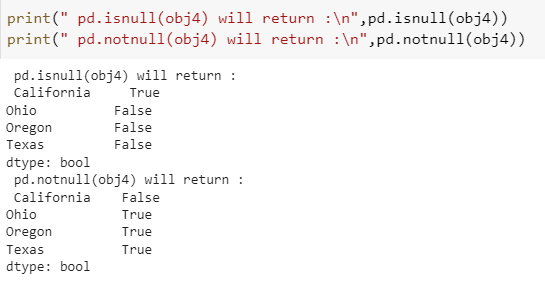
When you are only passing a dictionary, the index in the resulting Series will have the dictionaries keys in sorted order. You can override this by passing the dictionary keys in the order you want them to appear in the resulting Series as:

*states = ['California', 'Ohio', 'Oregon', 'Texas']*

*obj4 = pd.Series(sdata, index=states)*

Note: Three values found in sdata were placed in the appropriate locations, but since no value for 'California' was found, it appears as NaN (not a number), which is considered in pandas to mark missing or NA values. Since 'Utah' was not included in states, it is excluded from the resulting object.

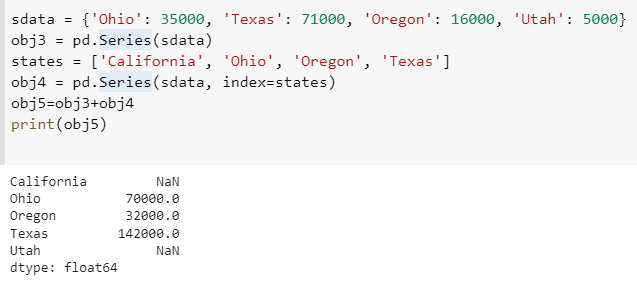
We can also check for the missing values in the series using *pd.isnull()* and *pd.notnull()* methods as:



Series also has these as instance methods which can be called using object instance as obj4.isnull() and obj4.notnull(). These will return the same results as pd methods.

**Arithmetic Operation between Series Objects**

Another feature of Pandas Series is that it automatically aligns by index label in arithmetic operations: For Example: if we have two series object with some common values and some different values then Pandas Series will automatically add the index values which are common in both lists and then add key of those that are present in only series object with NaN to indicate missing value.

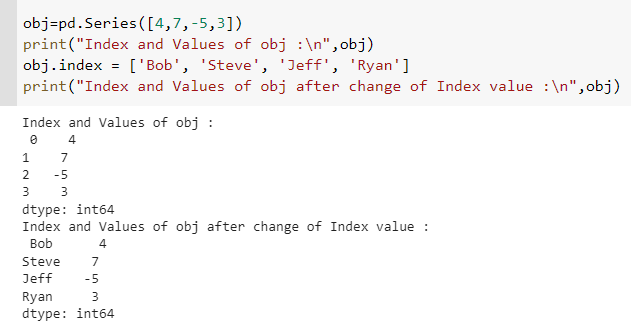


Both Series Object itself and its index have a name attribute, which integrates with other key areas of pandas functionality. For Example:

*obj4.name = 'population'*

*obj4.index.name = 'state'*

A Series’s index can be altered in-place by assignment. For example :



**Dataframe**