# Introduction to Pandas

Pandas is a name from “Panel Data” and is a Python library used for data manipulation and analysis. Pandas provides a convenient way to analyze and clean data.
The Pandas library introduces two new data structures to Python
- **Series** and **DataFrame**, both of which are built on top of NumPy.

## Why Use Pandas?

1. **Handle Large Data Efficiently**
2. **Tabular Data Representation**
3. **Data Cleaning and Preprocessing**
4. **Free and Open-Source**

## Install Pandas

To install pandas, you need Python and PIP installed in your system. If you have Python and PIP installed already, you can install pandas by entering the following command in the terminal:

```
pip install pandas
```

## Import Pandas in Python

We can import Pandas in Python using the import statement.

```
import pandas as pd
```

The code above imports the `pandas` library into our program with the alias `pd`. After this `import` statement, we can use Pandas functions and objects by calling them with `pd`.

For example, you can use Pandas dataframe in your program using `pd.DataFrame()`.

# Pandas Series

A Pandas Series is a one-dimensional labeled array-like object that can hold data of any type.

A Pandas Series can be thought of as a column in a spreadsheet or a single column of a DataFrame. It consists of two main components: the labels and the data.

For example,

```
0    'John'
1    30
2    6.2
3    False
dtype: object
```

Here, the series has two columns, labels (**0**, **1**, **2** and **3**) and data (`'John'`, `30`, `6.2`, `False`).

## Create a Pandas Series

There are multiple ways to create a Pandas Series, but the most common way is by using a Python list. Let's see an example of creating a Series using a list:

```
import pandas as pd

# create a list
data = [10, 20, 30, 40, 50]

# create a series from the list
my_series = pd.Series(data)
```

```
print(my_series)
```
Run Code

**Output**

```
0   10
1   20
2   30
3   40
4   50
dtype: int64
```

## Labels

The labels in the Pandas Series are index numbers by default. Like in dataframe and array, the index number in series starts from **0**.

Such labels can be used to access a specified value. For example,

```
import pandas as pd

# create a list
data = [10, 20, 30, 40, 50]

# create a series from the list
my_series = pd.Series(data)

# display third value in the series
print(my_series[2])
```

**Output**

```
30
```

We can also specify labels while creating the series using the `index` argument in the `Series()` method. For example,

```
import pandas as pd

# create a list
a = [1, 3, 5]

# create a series and specify labels
```

```
my_series = pd.Series(a, index = ["x", "y", "z"])



print(my_series)
```
Run Code

**Output**

```
x  1
y  3
z  5
dtype: int64
```

## Create Series From a Python Dictionary

You can also create a Pandas Series from a Python dictionary. For example,

```
import pandas as pd

# create a dictionary
grades = {"Semester1": 3.25, "Semester2": 3.28, "Semester3": 3.75}

# create a series from the dictionary
my_series = pd.Series(grades)
my_specific_series = pd.Series(grades, index = ["Semester1", "Semester2"])
# display the series
print(my_series)
print("SPECIFIC SERIES -")
print(my_specific_series)
```

**Output**

```
Semester1  3.25
Semester2  3.28
Semester3  3.75
dtype: float64
SPECIFIC SERIES -
Semester1  3.25
Semester2  3.28
dtype: float64
```

## Accessing element of Series

There are two ways through which we can access element of series, they are :

1. **Accessing Element from Series with Position :** In order to access the series element refers to the index number. Use the index operator [ ] to access an element in a series. The index must be an integer. In order to access multiple elements from a series, we use Slice operation.

```
import pandas as pd
import numpy as np
# creating simple array
data = np.array(['g','e','e','k','s','f','o','r','g','e','e','k','s'])
ser = pd.Series(data)
#retrieve the first element
print(ser[:5])
```

**OUTPUT –**

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

**2. Accessing Element Using Label (index) :**
In order to access an element from series, we have to set values by index label. A Series is like a fixed-size dictionary in that you can get and set values by index label.
Accessing a single element using index label

```
import pandas as pd
import numpy as np
# creating simple array
data =
np.array(['g','e','e','k','s','f','o','r','g','e','e','k','s'])
ser =
pd.Series(data,index=[10,11,12,13,14,15,16,17,18,19,20,21,22])
# accessing a element using index element
print(ser[16])
```

**Output :**

```
o
```

## Series object attributes

The Series attribute is defined as any information related to the Series object such as size, datatype. etc. Below are some of the attributes that you can use to get the information about the Series object:

| 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. |

## Example - Retrieving Index array , data array ,Types (dtype) and Size of Type (itemsize) of a series object

```
1. import numpy as np
2. import pandas as pd
3. x=pd.Series(data=[2,4,6,8])
4. y=pd.Series(data=[11.2,18.6,22.5], index=['a','b','c'])
```

```
5. print(x.index)
6. print(x.values)
7. print(y.index)
8. print(y.values)
9. print(x.dtype)
10. print(x.itemsize)
11. print(y.dtype)
12. print(y.itemsize)
```

**Output**

```
RangeIndex(start=0, stop=4, step=1)
[2 4 6 8]
Index(['a', 'b', 'c'], dtype='object')
[11.2 18.6 22.5]
int64
8
float64
8
```

## Retrieving Dimension, Size and Number of bytes:

```
1. import numpy as np
2. import pandas as pd
3. a=pd.Series(data=[1,2,3,4])
4. b=pd.Series(data=[4.9,8.2,5.6],
5. index=['x','y','z'])
6. print(a.ndim, b.ndim)
7. print(a.size, b.size)
8. print(a.nbytes, b.nbytes)
```

**Output**

```
1 1
4 3
32 24
```

## Checking Emptiness and Presence of NaNs

To check the Series object is empty, you can use the **empty attribute**. Similarly, to check if a series object contains some NaN values or not, you can use the **hasans** attribute.

**Example**

```
1. import numpy as np
2. import pandas as pd
3. a=pd.Series(data=[1,2,3,np.NaN])
4. b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
5. c=pd.Series()
6. print(a.empty,b.empty,c.empty)
7. print(a.hasnans,b.hasnans,c.hasnans)
8. print(len(a),len(b))
9. print(a.count( ),b.count( ))
```

**Output**

```
False   False   True
True    False   False
4   3
3   3
```

# CRUD OPERATION ON PANDA SERIES –

```
import pandas as pd

Phymarks = pd.Series([70,80,90,100])

print printing the original Series")

print(Phymarks)

print("Accessing the first element")

print(Phymarks[0])    #Read operation
```

```
print(“Modifying the first element as 57”)

phymarks[0]=57          #Update operation

print(Phymarks)

print(“Deleting an element”)

del Phymarks[3]     # delete operation

print(Phymarks)

print(“print marks greater than 80”)

print(Phymarks[Phymarks>80])
```

Output –

```
Printing the original Series
     0    70
     1    80
     2    90
     3    100
     dtype : int64
Accessing the first element
70
Modifying the first element as 57
     0    57
     1    80
     2    90
     3    100
    dtype:int64
Deleting an element
     0    57
     1    80
     2    90
    dtype:int64
Print marks greater than 80
     2    90
     dtype: int64
```