CHAPTER-1 Data Handling using Pandas –I

Pandas:

- It is a package useful for data analysis and manipulation.
- Pandas provide an easy way to create, manipulate and wrangle the data.
- Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Data scientists use Pandas for its following advantages:

- Easily handles missing data.
- It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure.
- It provides an efficient way to slice the data.
- It provides a flexible way to merge, concatenate or reshape the data.

DATA STRUCTURE IN PANDAS

A data structure is a way to arrange the data in such a way that so it can be accessed quickly and we can perform various operation on this data like- retrieval, deletion, modification etc.

Pandas deals with 3 data structure-

1. Series
2. Data Frame
3. Panel

We are having only series and data frame in our syllabus.

Series

Series-Series is a one-dimensional array like structure with homogeneous data, which can be used to handle and manipulate data. What makes it special is its index attribute, which has incredible functionality and is heavily mutable.

It has two parts-

1. **Data part (An array of actual data)**
2. **Associated index with data (associated array of indexes or data labels)**

e.g.-

Index	Data
0	10
1	15
2	18
3	22

- ✓ We can say that **Series** is a *labeled one-dimensional array which can hold any type of data.*
- ✓ Data of **Series** is *always mutable*, means it can be changed.
- ✓ But the size of Data of **Series** is *always immutable*, means it cannot be changed.
- ✓ **Series** may be considered as a **Data Structure with two arrays** out which **one array** works as *Index (Labels)* and the **second array** works as *original Data.*
- ✓ ***Row Labels*** in Series are called *Index*.

Syntax to create a Series:

<Series Object>=pandas.Series (data, index=idx *(optional)*)

- ✓ Where data may be *python sequence (Lists)*, ndarray, scalar value or a python dictionary.

How to create Series with nd array

Program-

```
import pandas as pd

import numpy as np

arr=np.array([10,15,18,22])

s = pd.Series(arr)

print(s)
```

Output-

```
0    10
1    15
2    18
3    22
```

Default Index

Data

Here we create an array of 4 values.

How to create Series with Mutable index

Program-

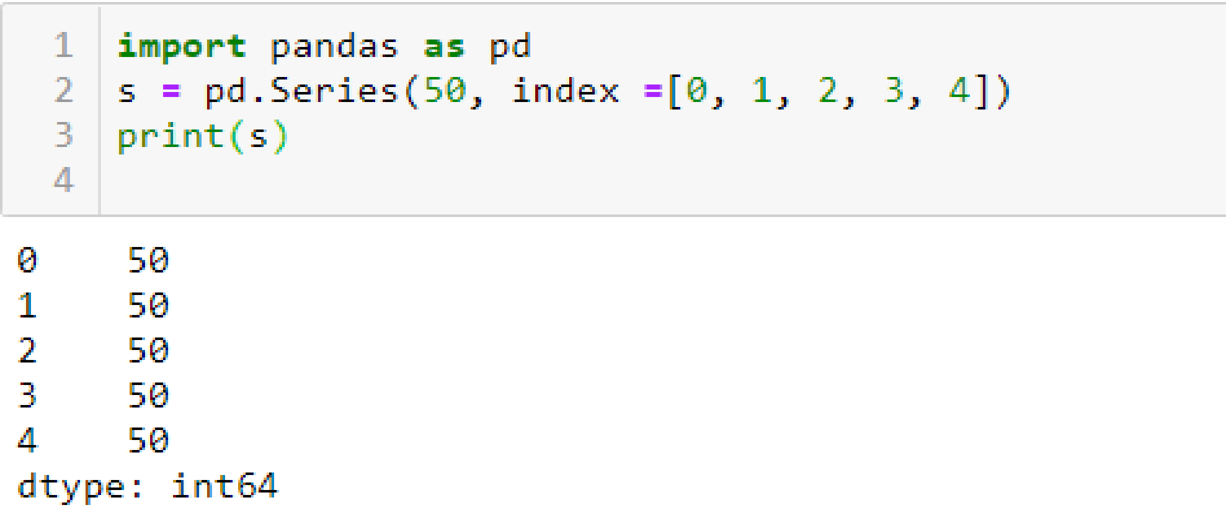
```
import pandas as pd

import numpy as np

arr=np.array(['a','b','c','d'])

s=pd.Series(arr,

    index=['first','second','third','fourth'])

print(s)
```

Output-

```
first      a
second     b
third      c
fourth     d
```

Creating a series from Scalar value

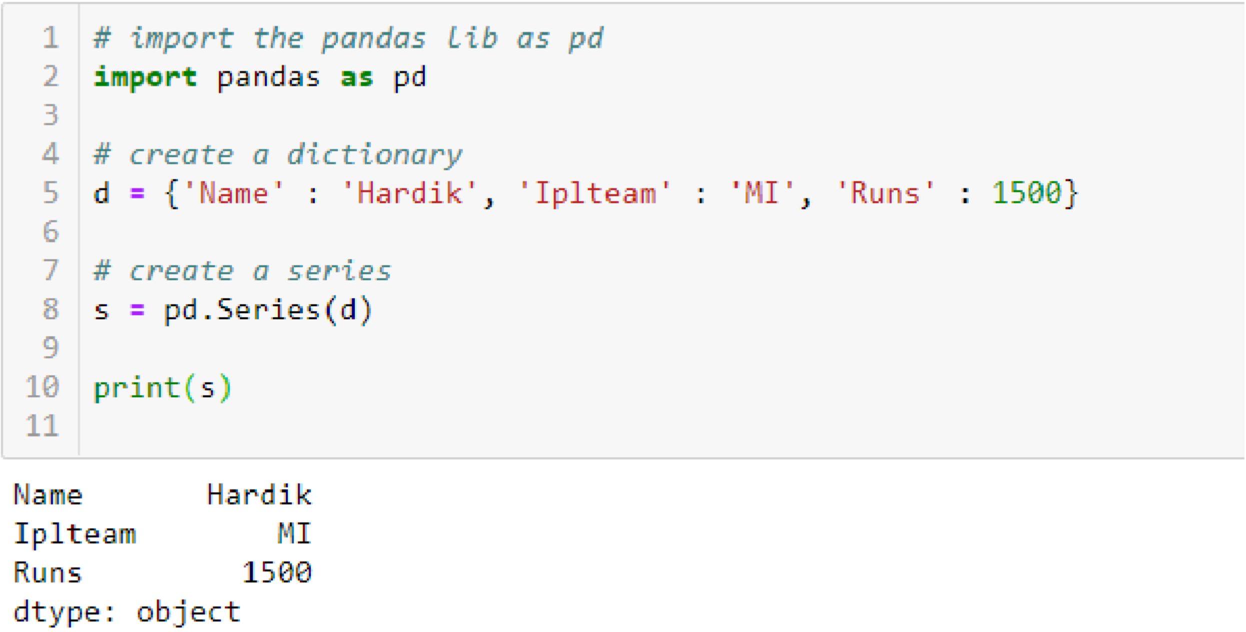
To create a series from scalar value, an index must be provided. The scalar value will be repeated as per the length of index.



```python
import pandas as pd
s = pd.Series(50, index =[0, 1, 2, 3, 4])
print(s)

```

```
0    50
1    50
2    50
3    50
4    50
dtype: int64
```

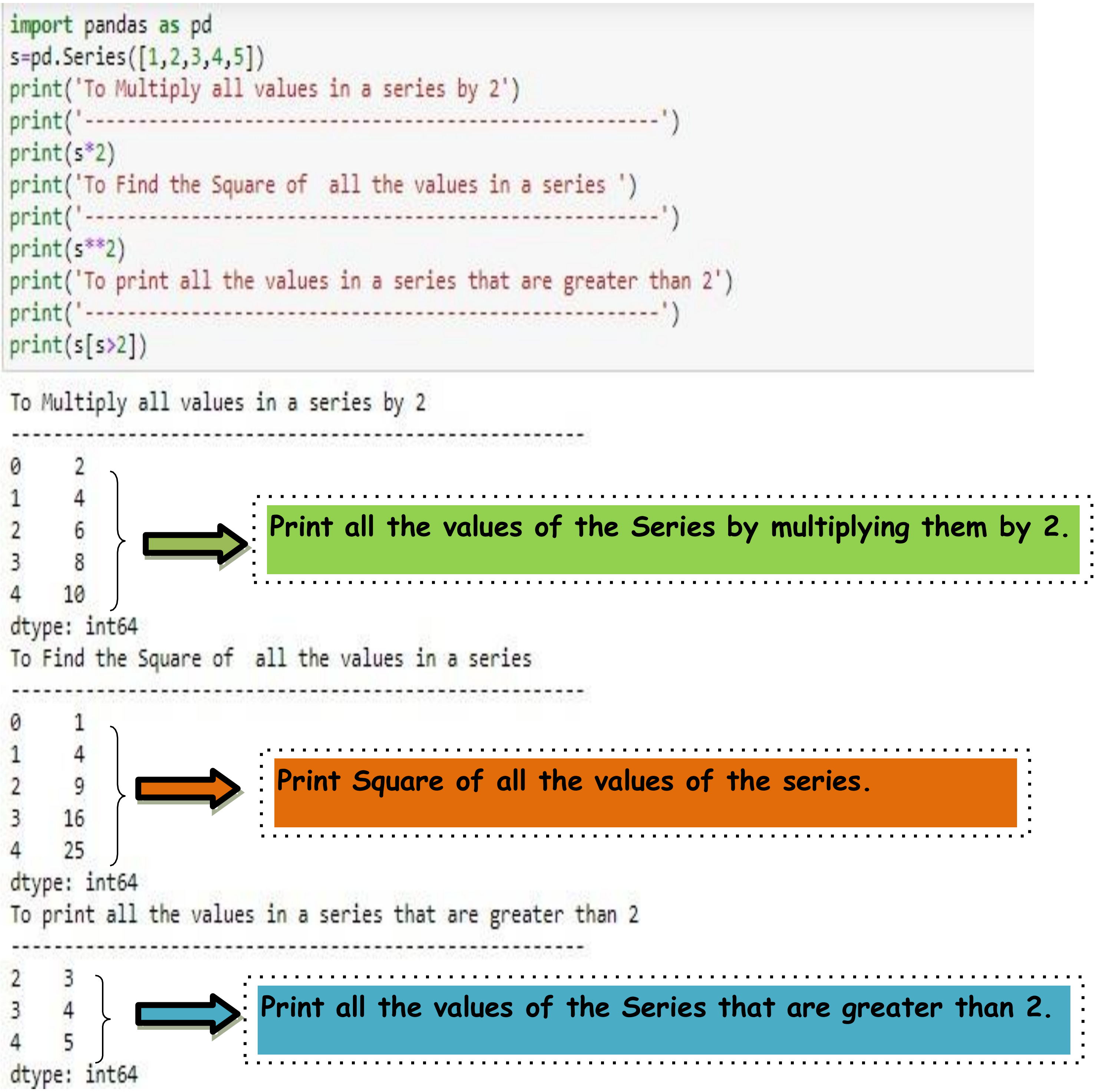
Creating a series from a Dictionary



```python
# import the pandas lib as pd
import pandas as pd

# create a dictionary
d = {'Name' : 'Hardik', 'Iplteam' : 'MI', 'Runs' : 1500}

# create a series
s = pd.Series(d)

print(s)

```

```
Name       Hardik
Iplteam        MI
Runs         1500
dtype: object
```

Mathematical Operations in Series



```
import pandas as pd
s=pd.Series([1,2,3,4,5])
print('To Multiply all values in a series by 2')
print('-----------------------------------------------------')
print(s*2)
print('To Find the Square of  all the values in a series ')
print('-----------------------------------------------------')
print(s**2)
print('To print all the values in a series that are greater than 2')
print('-----------------------------------------------------')
print(s[s>2])
```

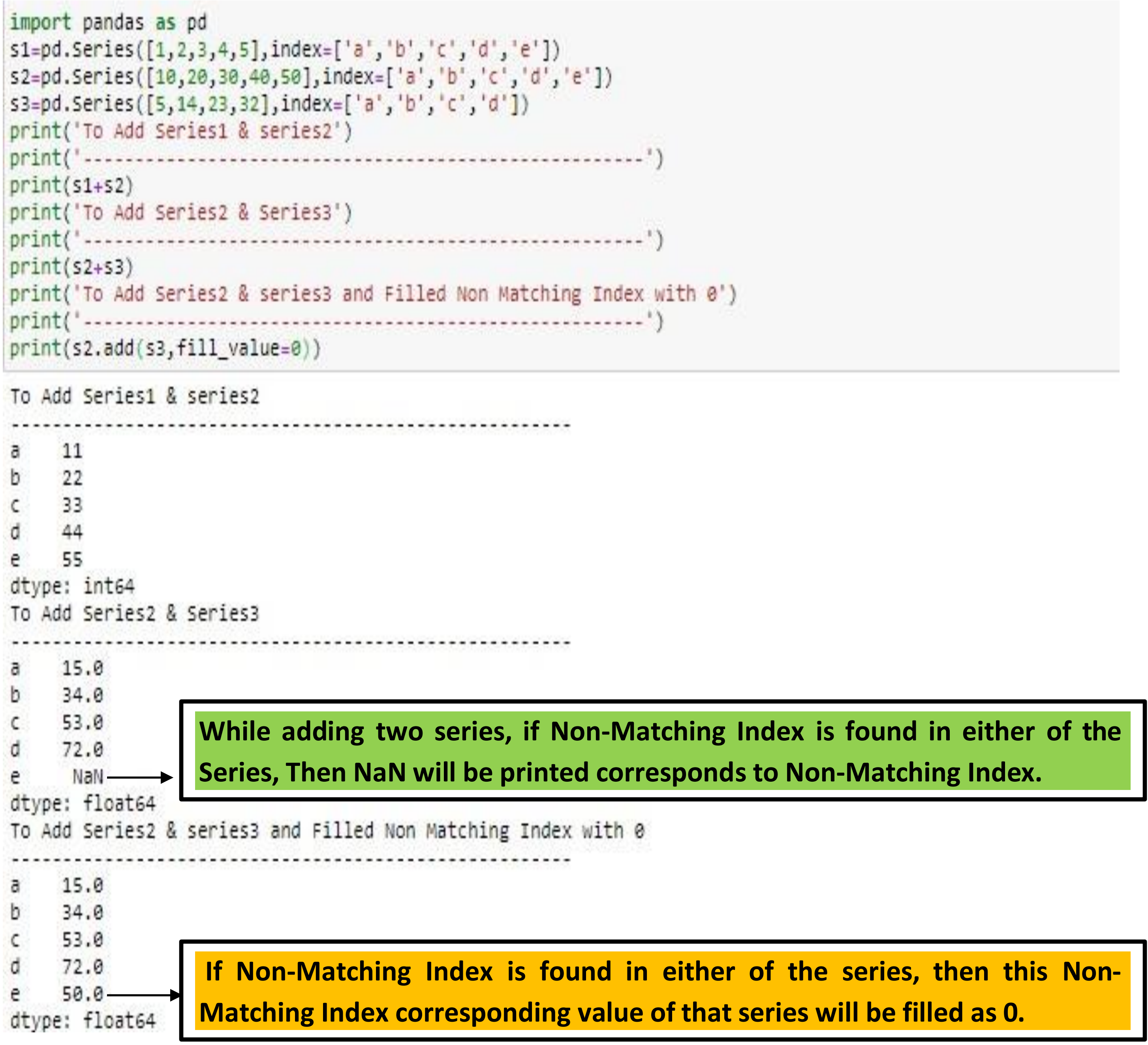
```
To Multiply all values in a series by 2
-----------------------------------------------------
0     2
1     4
2     6
3     8
4    10
dtype: int64
To Find the Square of  all the values in a series
-----------------------------------------------------
0     1
1     4
2     9
3    16
4    25
dtype: int64
To print all the values in a series that are greater than 2
-----------------------------------------------------
2    3
3    4
4    5
dtype: int64
```

Print all the values of the Series by multiplying them by 2.

Print Square of all the values of the series.

Print all the values of the Series that are greater than 2.

Example-2



```python
import pandas as pd
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s2=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
s3=pd.Series([5,14,23,32],index=['a','b','c','d'])
print('To Add Series1 & series2')
print('-----------------------------------------------------')
print(s1+s2)
print('To Add Series2 & Series3')
print('-----------------------------------------------------')
print(s2+s3)
print('To Add Series2 & series3 and Filled Non Matching Index with 0')
print('-----------------------------------------------------')
print(s2.add(s3,fill_value=0))
```

```
To Add Series1 & series2
-----------------------------------------------------
a    11
b    22
c    33
d    44
e    55
dtype: int64
To Add Series2 & Series3
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e     NaN
dtype: float64
To Add Series2 & series3 and Filled Non Matching Index with 0
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e    50.0
dtype: float64
```

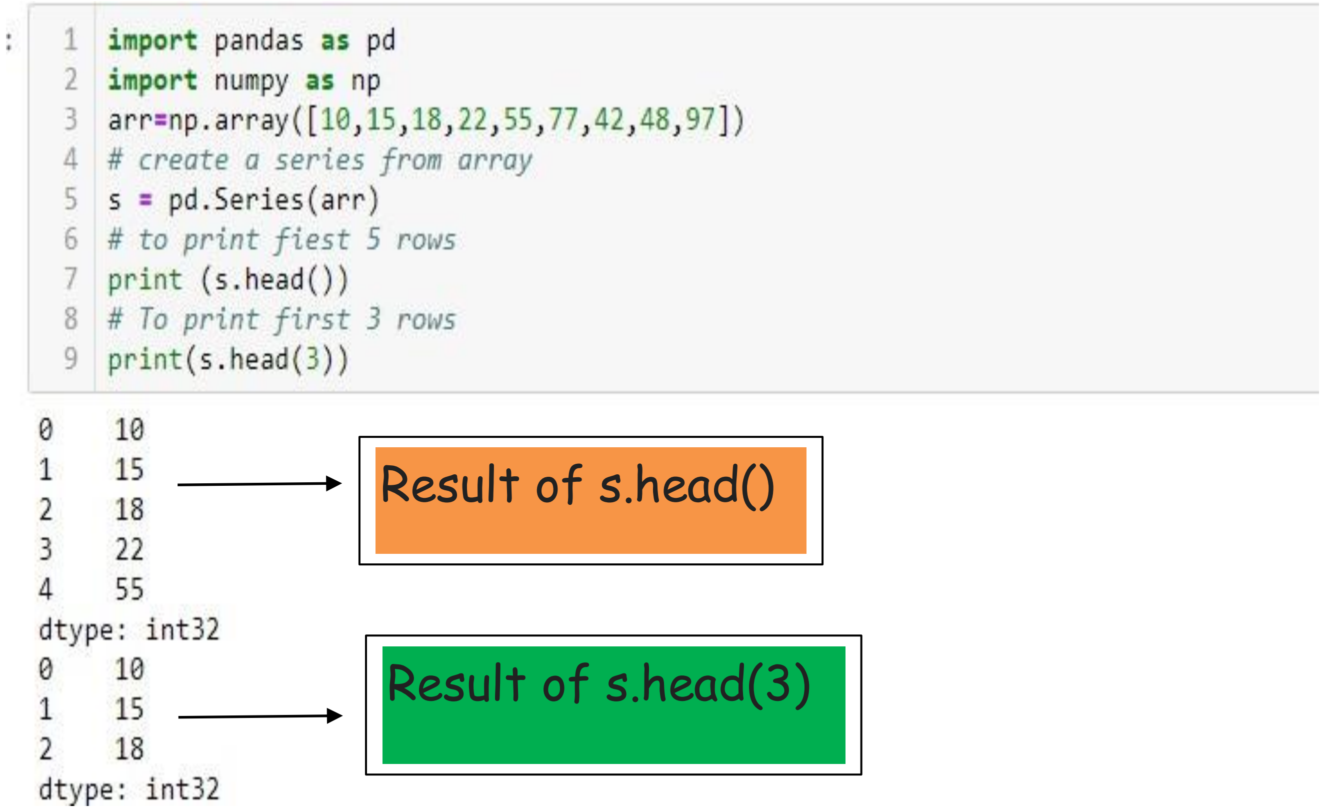
While adding two series, if Non-Matching Index is found in either of the Series, Then NaN will be printed corresponds to Non-Matching Index.

If Non-Matching Index is found in either of the series, then this Non-Matching Index corresponding value of that series will be filled as 0.

Head and Tail Functions in Series

head (): It is used to access the first 5 rows of a series.

Note :To access first 3 rows we can call series_name.head(3)

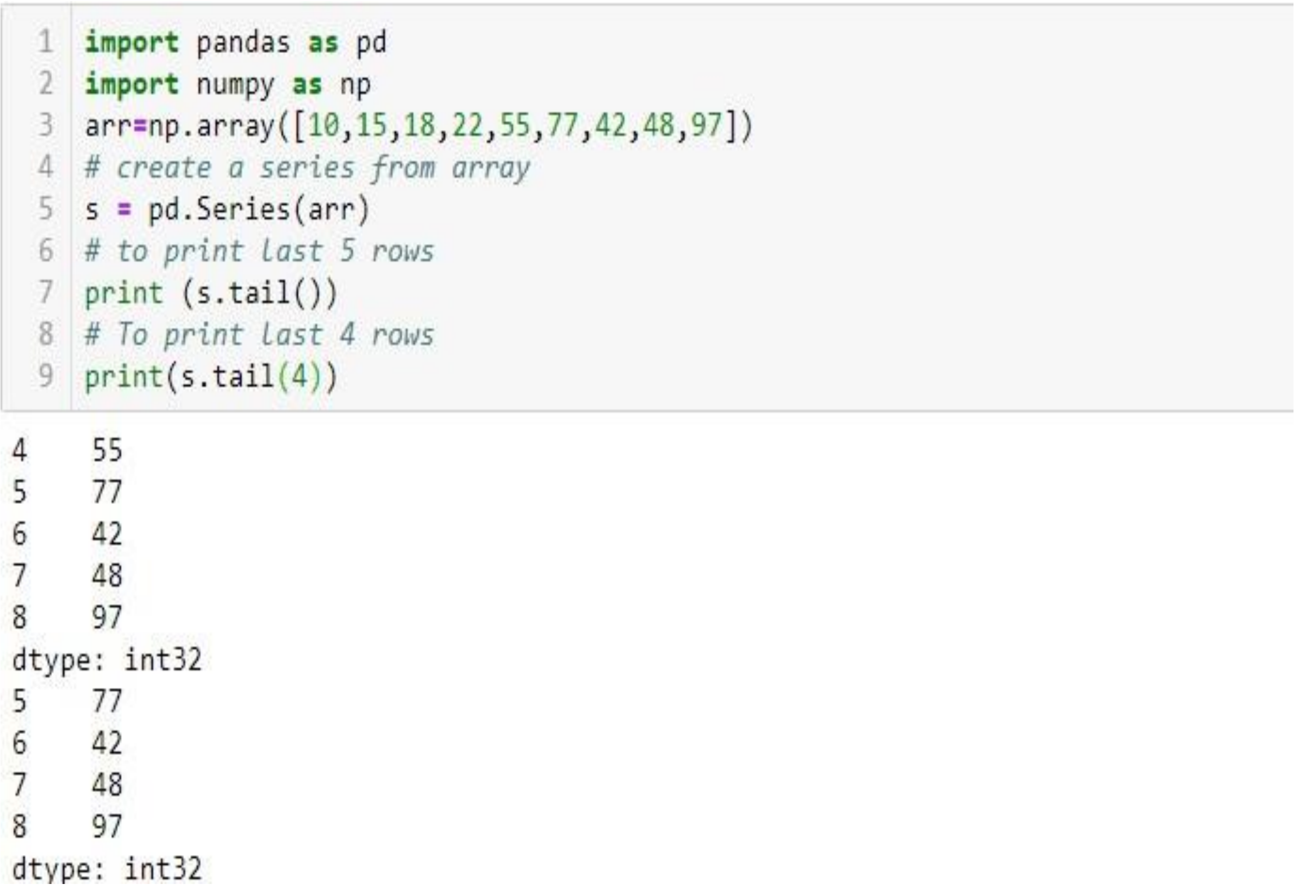


```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77,42,48,97])
# create a series from array
s = pd.Series(arr)
# to print fiest 5 rows
print (s.head())
# To print first 3 rows
print(s.head(3))
```

```
0    10
1    15
2    18
3    22
4    55
dtype: int32
0    10
1    15
2    18
dtype: int32
```

tail(): It is used to access the last 5 rows of a series.

Note :To access last 4 rows we can call series_name.tail (4)



```python
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77,42,48,97])
# create a series from array
s = pd.Series(arr)
# to print last 5 rows
print (s.tail())
# To print last 4 rows
print(s.tail(4))
```

```
4    55
5    77
6    42
7    48
8    97
dtype: int32
5    77
6    42
7    48
8    97
dtype: int32
```

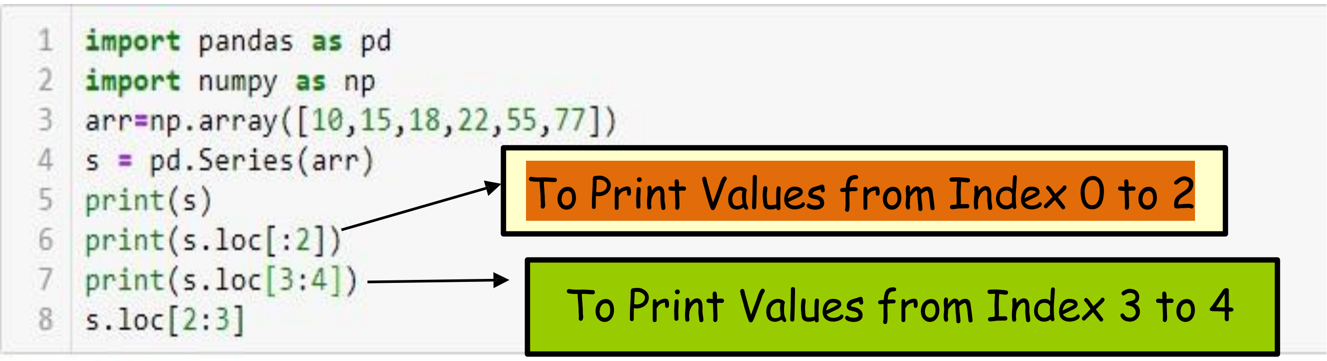
Selection in Series

Series provides index label loc and ilocand [] to access rows and columns.

1. loc index label :-

Syntax:-series_name.loc[StartRange: StopRange]

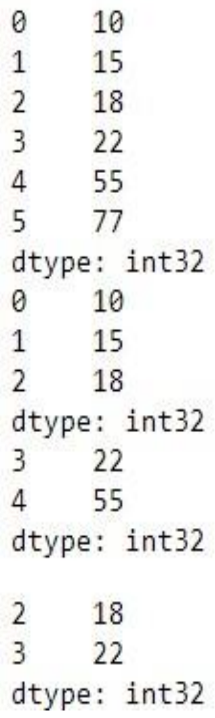
Example-



```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77])
s = pd.Series(arr)
print(s)
print(s.loc[:2])
print(s.loc[3:4])
s.loc[2:3]
```

To Print Values from Index 0 to 2

To Print Values from Index 3 to 4

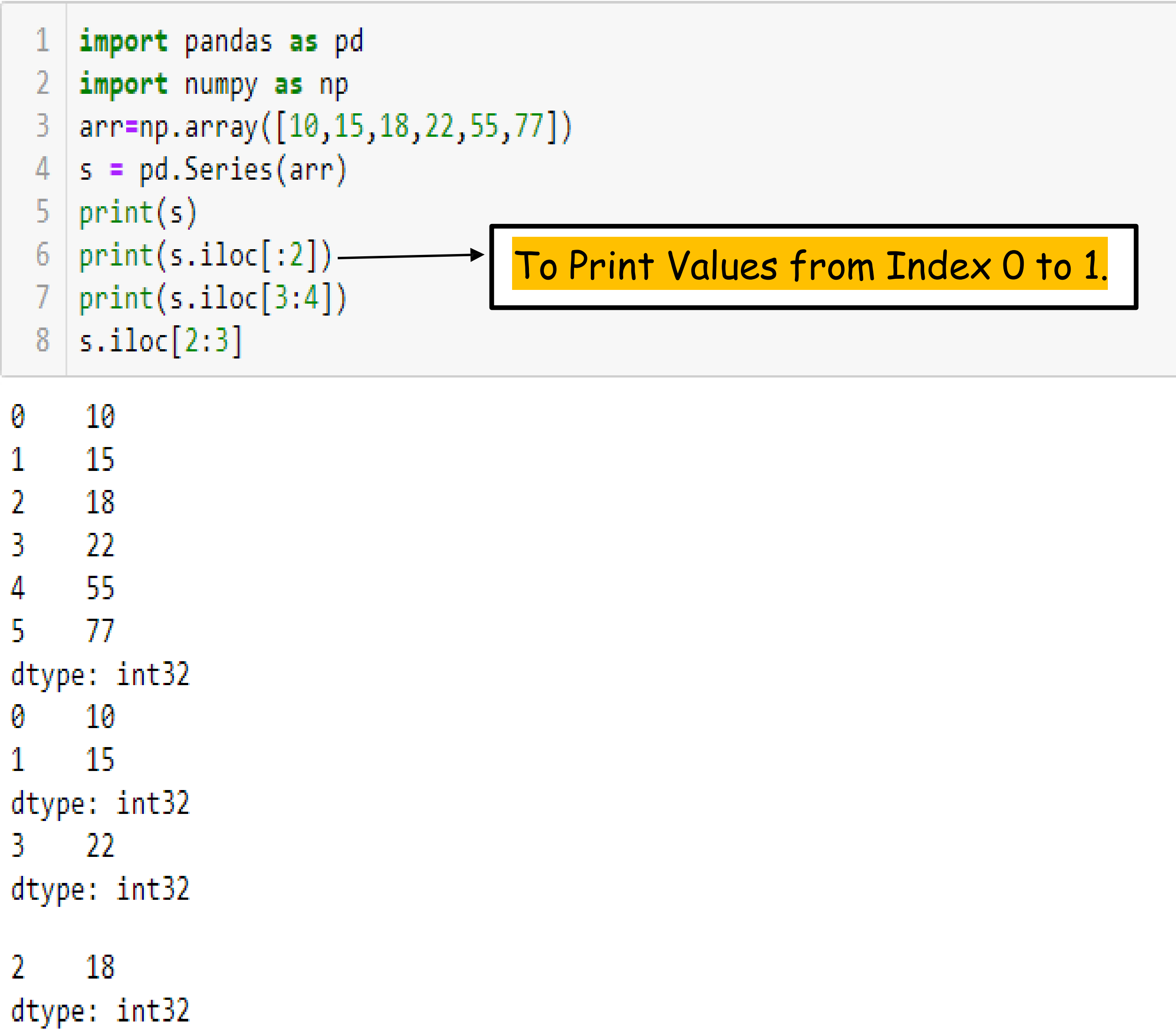


```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
2    18
dtype: int32
3    22
4    55
dtype: int32

2    18
3    22
dtype: int32
```

2. Selection Using iloc index label :-

Syntax:-series_name.iloc[StartRange : StopRange]

Example-



```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77])
s = pd.Series(arr)
print(s)
print(s.iloc[:2])
print(s.iloc[3:4])
s.iloc[2:3]
```

To Print Values from Index 0 to 1.

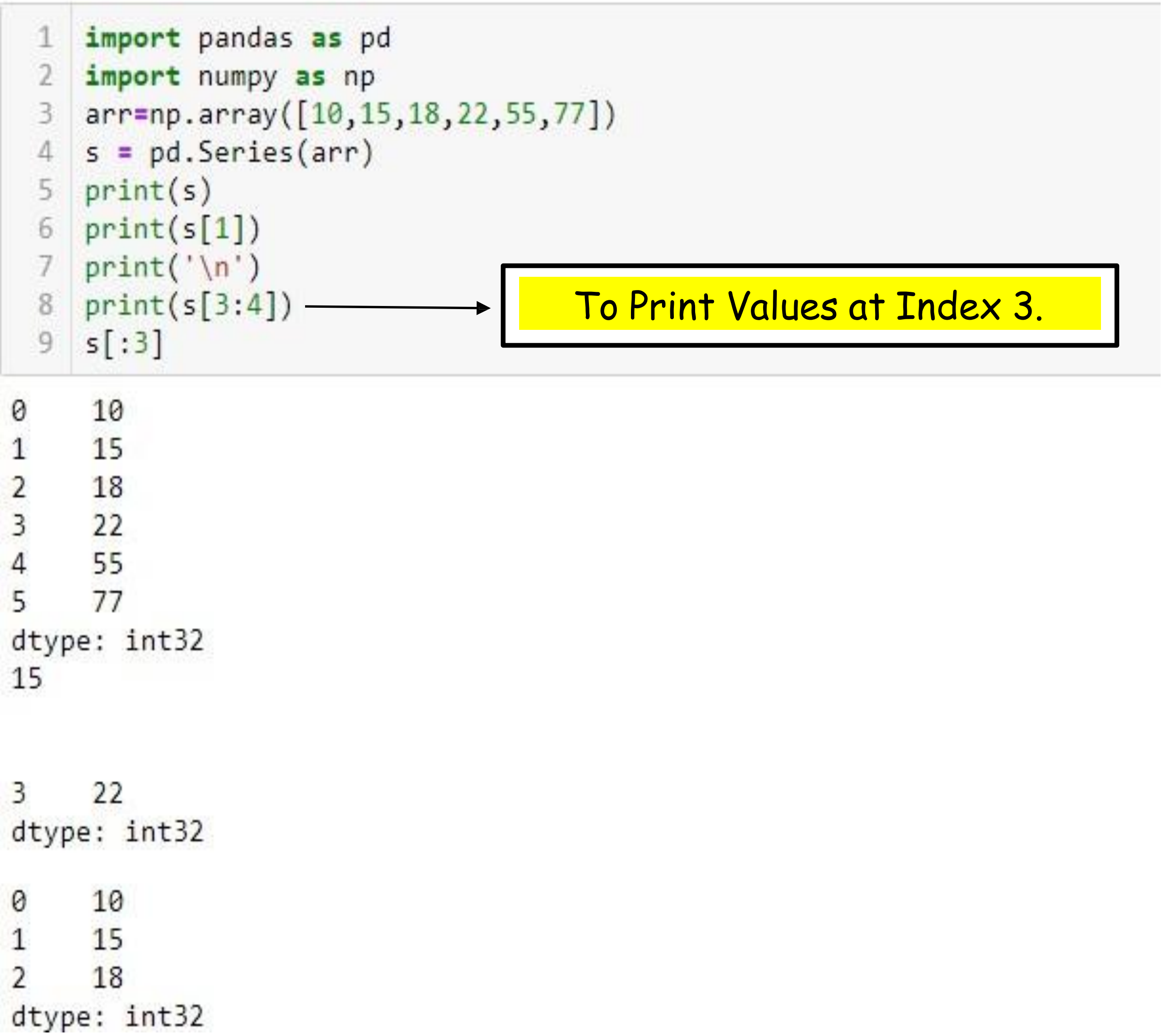
```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
dtype: int32
3    22
dtype: int32

2    18
dtype: int32
```

3. Selection Using [] :

Syntax:-series_name[StartRange> : StopRange] or

series_name[index]

Example-



```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77])
s = pd.Series(arr)
print(s)
print(s[1])
print('\n')
print(s[3:4])
s[:3]
```

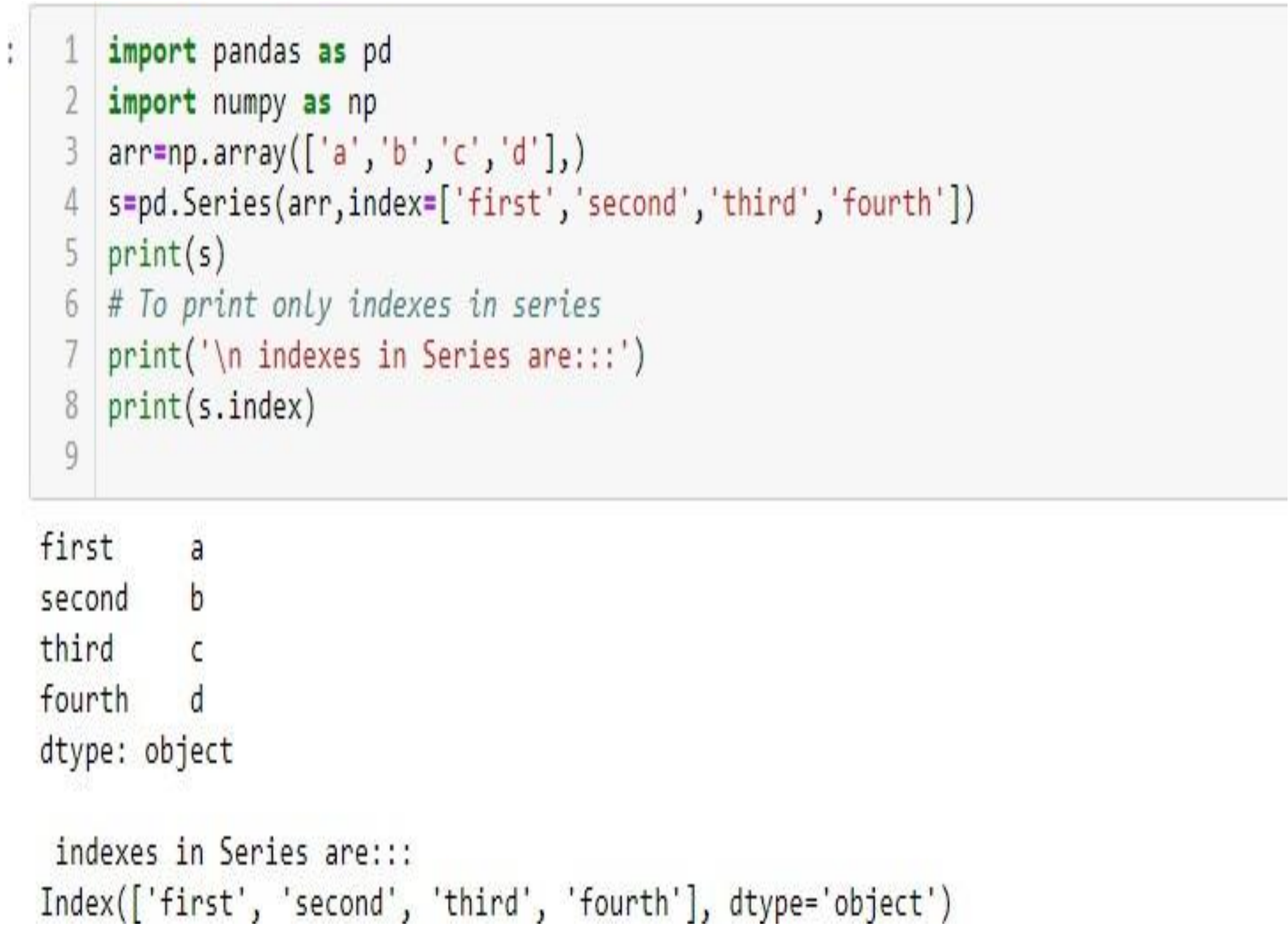
To Print Values at Index 3.

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
15


3    22
dtype: int32

0    10
1    15
2    18
dtype: int32
```

Indexing in Series

Pandas provide index attribute to get or set the index of entries or values in series.

Example-



```
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'],)
s=pd.Series(arr,index=['first','second','third','fourth'])
print(s)
# To print only indexes in series
print('\n indexes in Series are:::')
print(s.index)
```

```
first     a
second    b
third     c
fourth    d
dtype: object

 indexes in Series are:::
Index(['first', 'second', 'third', 'fourth'], dtype='object')
```

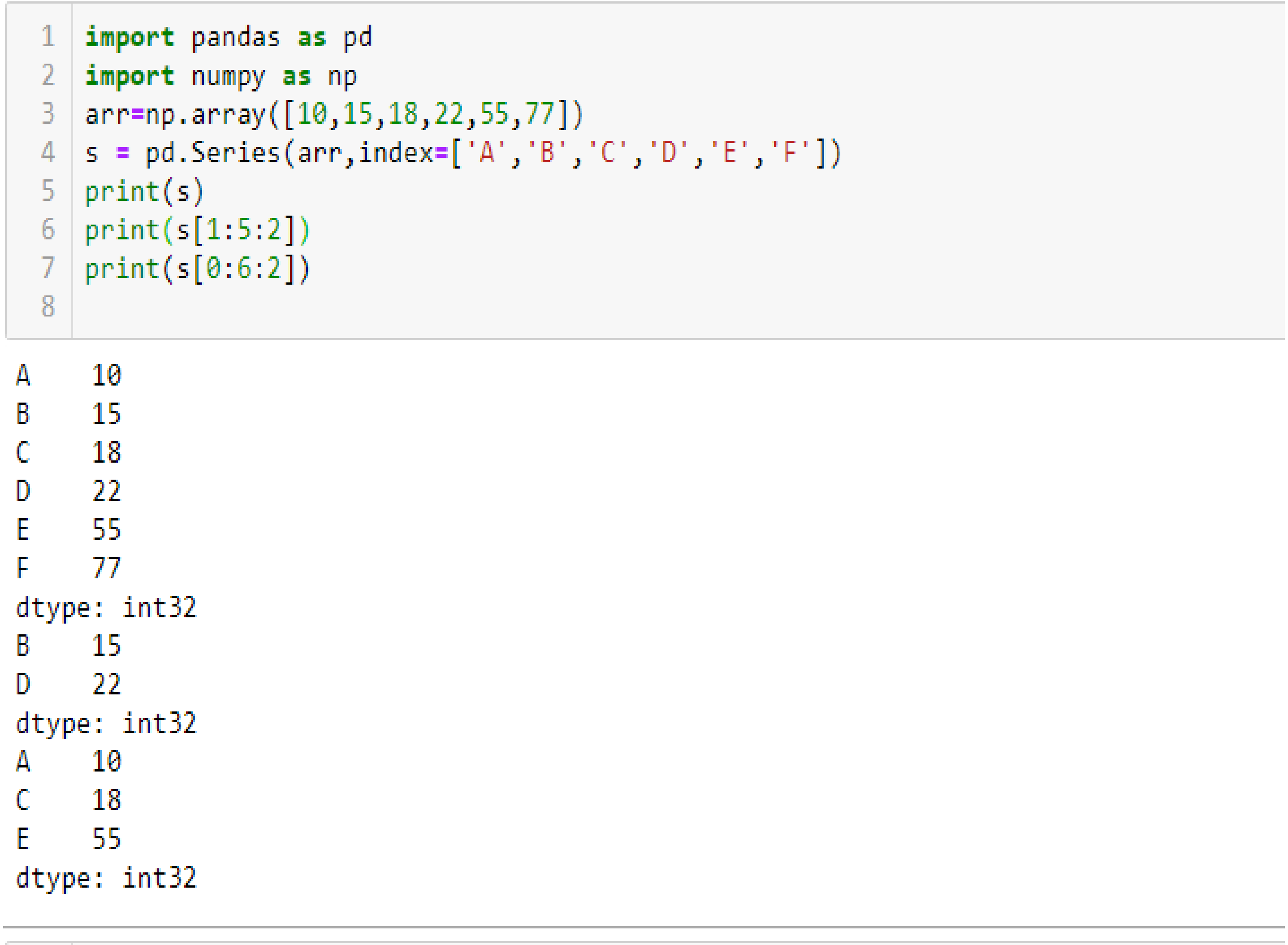
Slicing in Series

Slicing is a way to retrieve subsets of data from a pandas object. A slice object syntax is –

SERIES_NAME [start:end: step]

The segments start representing the first item, end representing the last item, and step representing the increment between each item that you would like.

Example :-



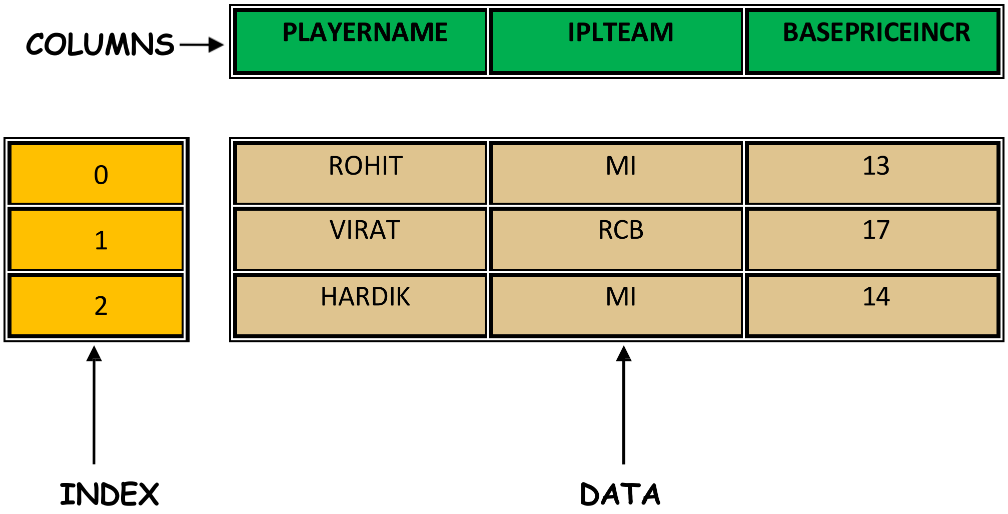
```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77])
s = pd.Series(arr,index=['A','B','C','D','E','F'])
print(s)
print(s[1:5:2])
print(s[0:6:2])

```

```
A    10
B    15
C    18
D    22
E    55
F    77
dtype: int32
B    15
D    22
dtype: int32
A    10
C    18
E    55
dtype: int32
```

DATAFRAME

DATAFRAME-It is a two-dimensional object that is useful in representing data in the form of rows and columns. It is similar to a spreadsheet or an SQL table. This is the most commonly used pandas object. Once we store the data into the Dataframe, we can perform various operations that are useful in analyzing and understanding the data.

DATAFRAME STRUCTURE



	PLAYERNAME	IPLTEAM	BASEPRICEINCR
0	ROHIT	MI	13
1	VIRAT	RCB	17
2	HARDIK	MI	14

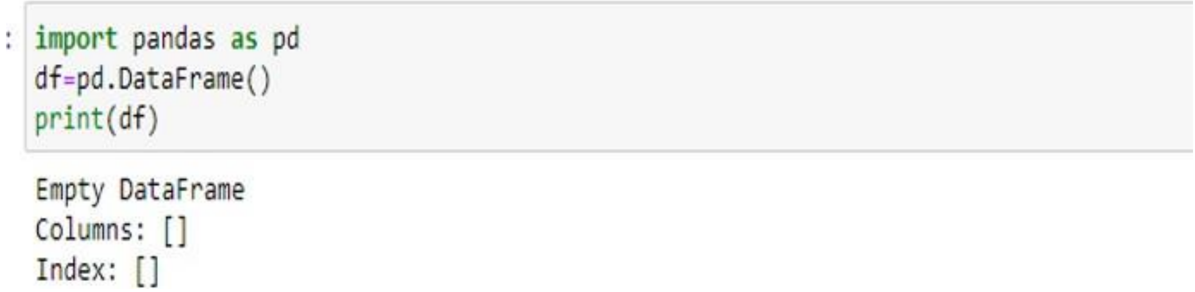
PROPERTIES OF DATAFRAME

1. A Dataframe has axes (indices)-
   - Row index (axis=0)
   - Column index (axes=1)
2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.
3. A Dataframe contains Heterogeneous data.
4. A Dataframe Size is Mutable.
5. A Dataframe Data is Mutable.

A data frame can be created using any of the following-

1. **Series**
2. **Lists**
3. **Dictionary**
4. **A numpy 2D array**

How to create Empty Dataframe



```
: import pandas as pd
  df=pd.DataFrame()
  print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

How to create Dataframe From Series

Program-

```
import pandas as pd

s = pd.Series(['a','b','c','d'])

df=pd.DataFrame(s)

print(df)
```

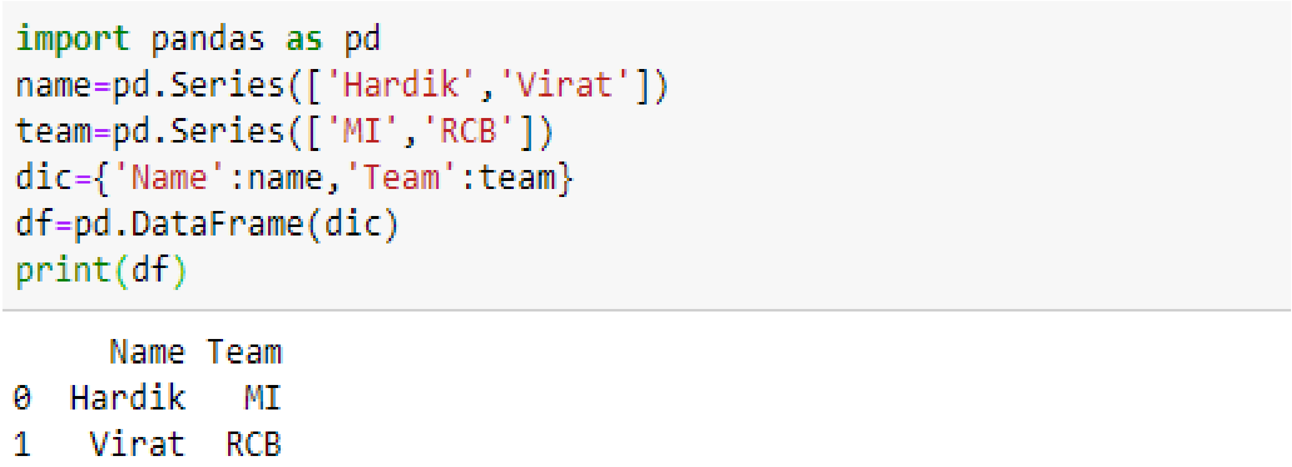
Output-

```
   0
0  a
1  b
2  c
3  d
```

Default Column Name As 0

DataFrame from Dictionary of Series

Example-

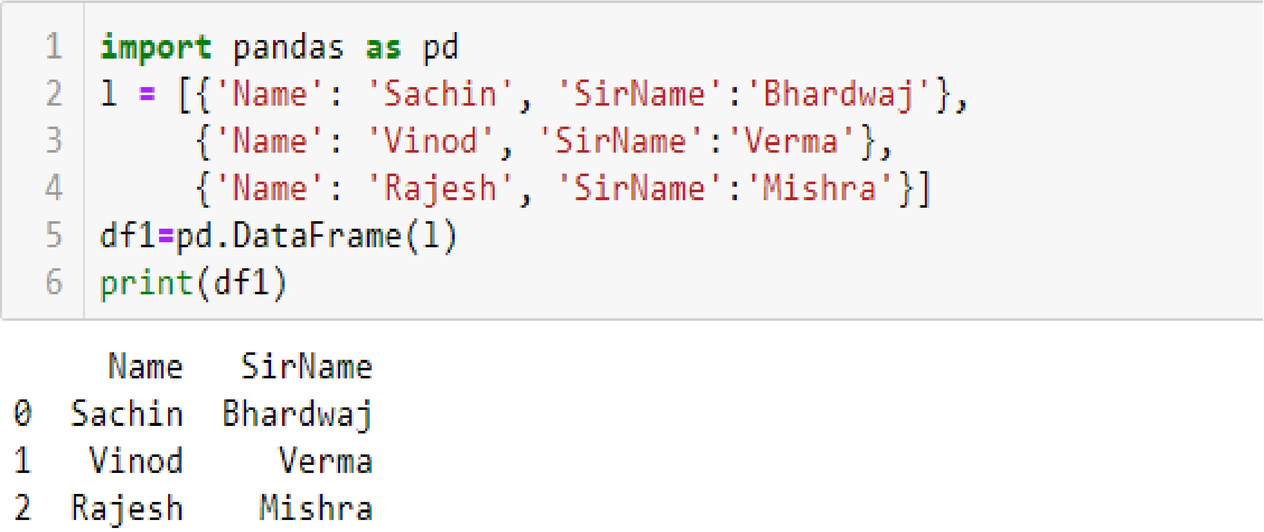


```python
import pandas as pd
name=pd.Series(['Hardik','Virat'])
team=pd.Series(['MI','RCB'])
dic={'Name':name,'Team':team}
df=pd.DataFrame(dic)
print(df)
```

```
     Name Team
0  Hardik   MI
1   Virat  RCB
```

DataFrame from List of Dictionaries

Example-



```python
import pandas as pd
l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
     {'Name': 'Vinod', 'SirName':'Verma'},
     {'Name': 'Rajesh', 'SirName':'Mishra'}]
df1=pd.DataFrame(l)
print(df1)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma
2  Rajesh    Mishra
```

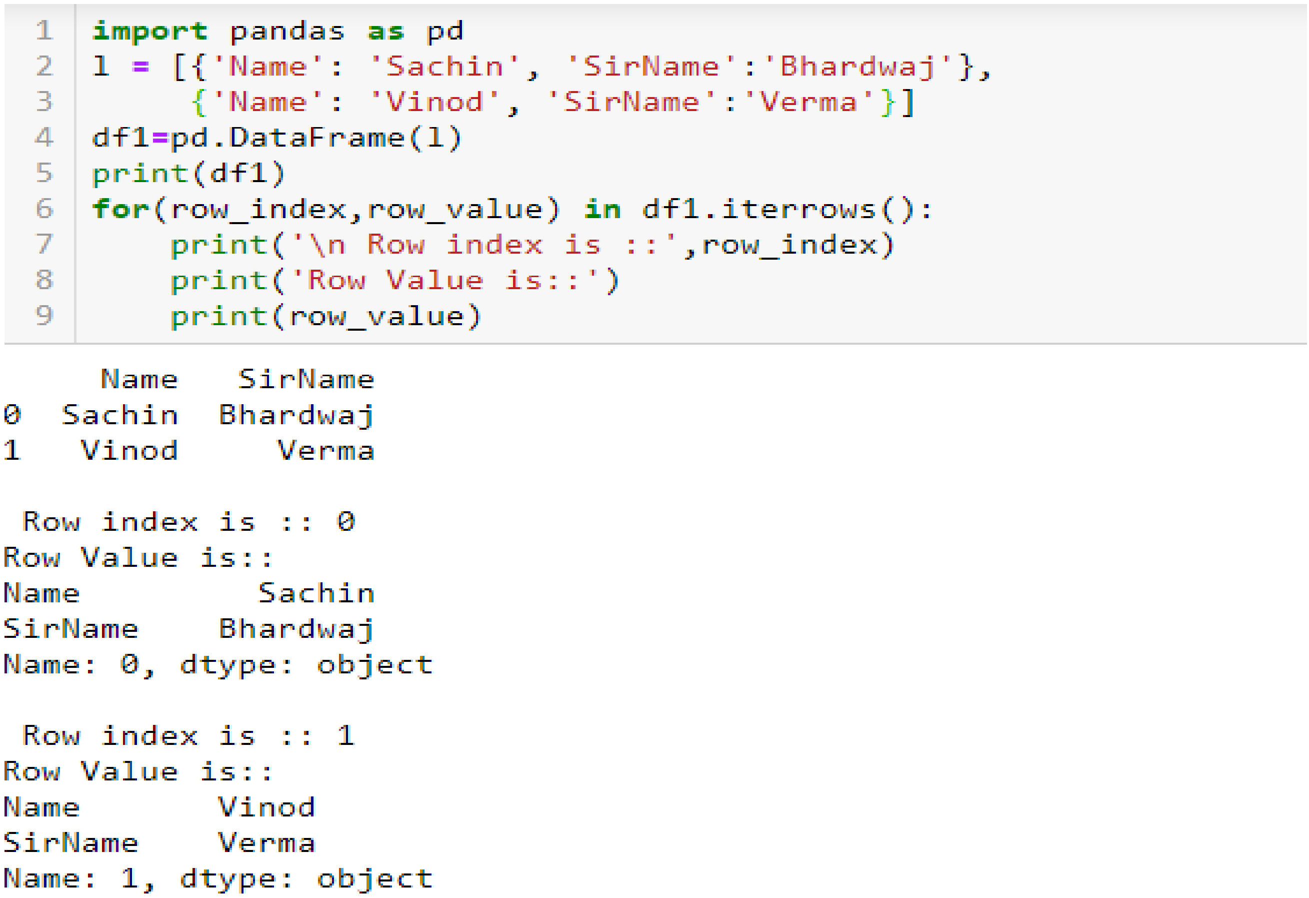
Iteration on Rows and Columns

If we want to access record or data from a data frame row wise or column wise then iteration is used. Pandas provide 2 functions to perform iterations-

1. iterrows ()
2. iteritems ()

iterrows()

It is used to access the data row wise. Example-



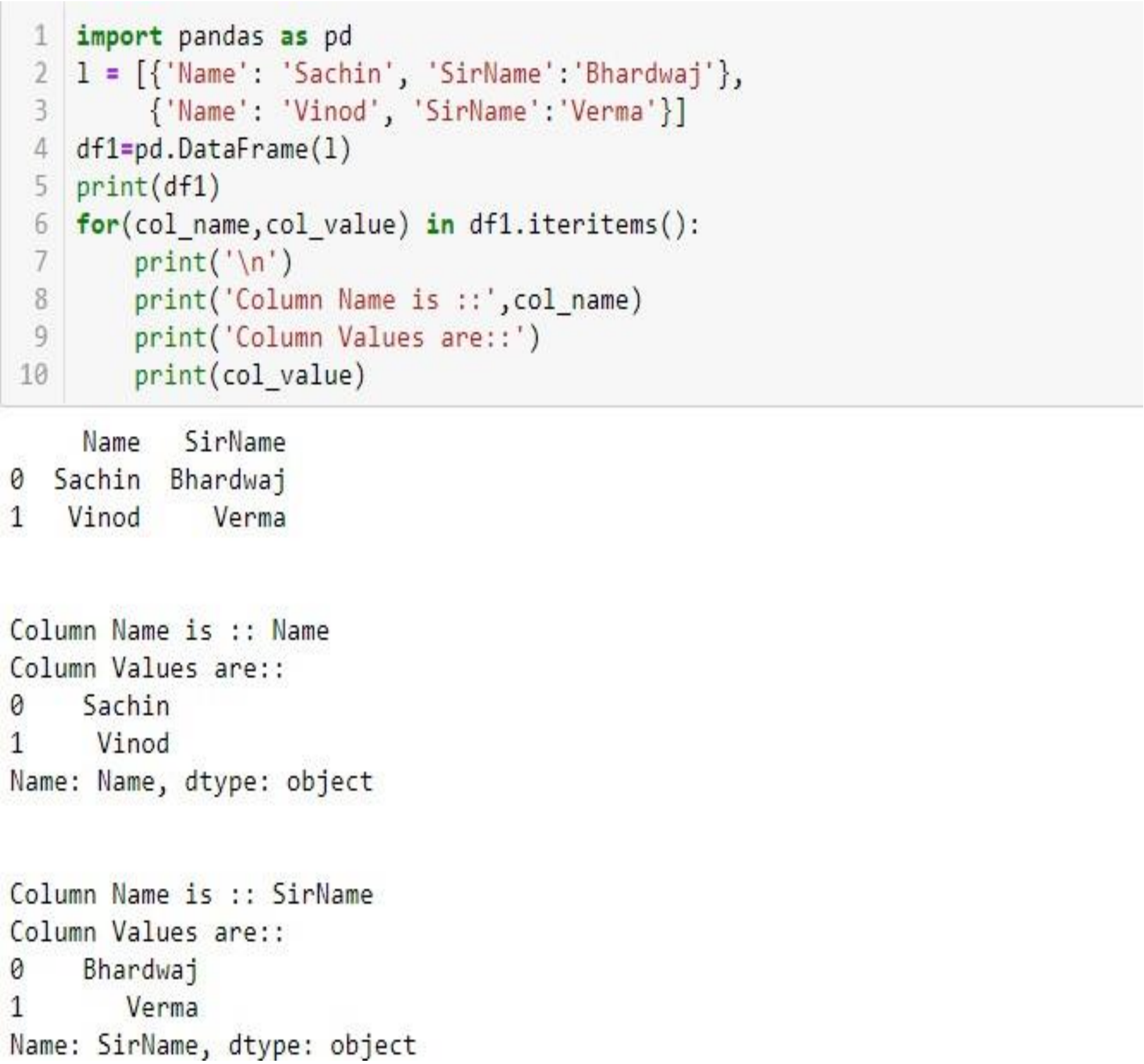
```python
import pandas as pd
l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
     {'Name': 'Vinod', 'SirName':'Verma'}]
df1=pd.DataFrame(l)
print(df1)
for(row_index,row_value) in df1.iterrows():
    print('\n Row index is ::',row_index)
    print('Row Value is::')
    print(row_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma

 Row index is :: 0
Row Value is::
Name         Sachin
SirName    Bhardwaj
Name: 0, dtype: object

 Row index is :: 1
Row Value is::
Name       Vinod
SirName    Verma
Name: 1, dtype: object
```

iteritems()

It is used to access the data column wise.

Example-



```python
import pandas as pd
l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
     {'Name': 'Vinod', 'SirName':'Verma'}]
df1=pd.DataFrame(l)
print(df1)
for(col_name,col_value) in df1.iteritems():
    print('\n')
    print('Column Name is ::',col_name)
    print('Column Values are::')
    print(col_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma


Column Name is :: Name
Column Values are::
0    Sachin
1     Vinod
Name: Name, dtype: object


Column Name is :: SirName
Column Values are::
0    Bhardwaj
1       Verma
Name: SirName, dtype: object
```

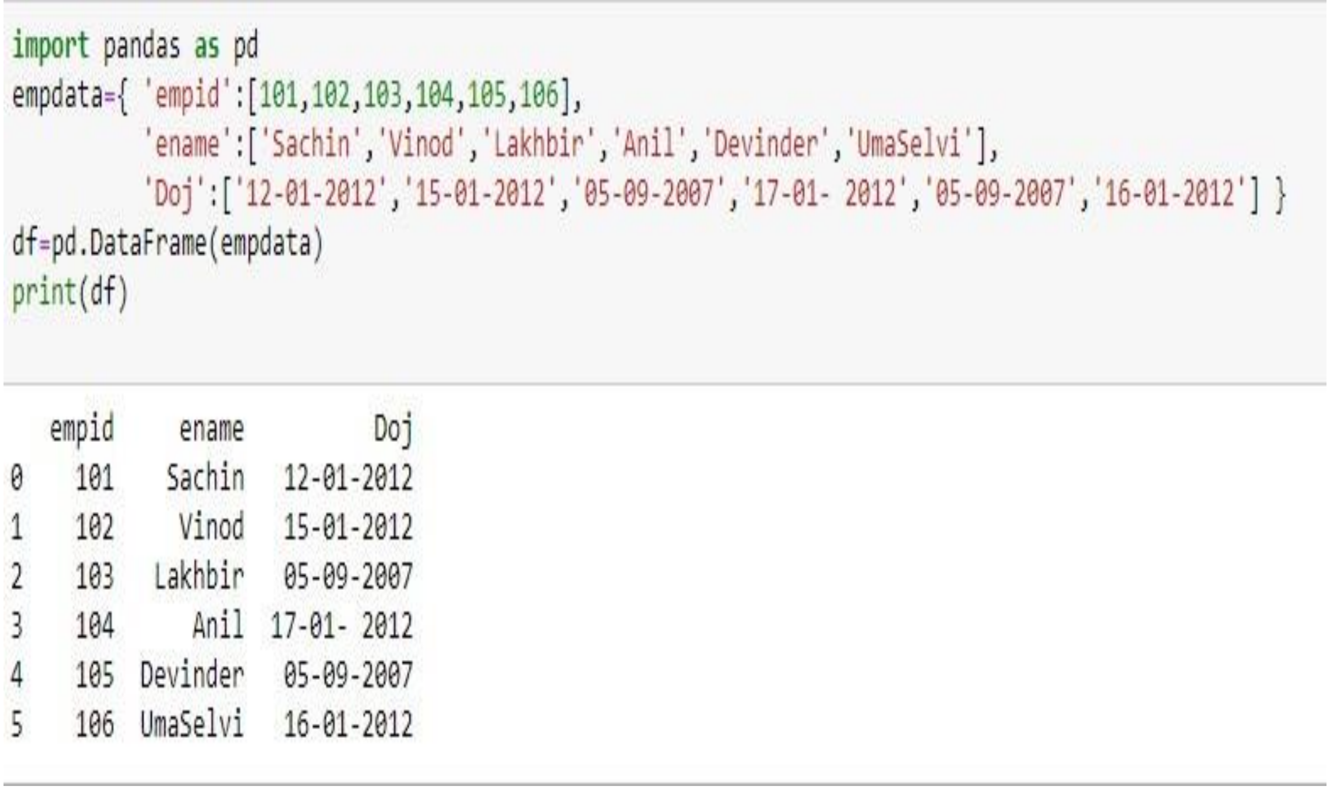
Select operation in data frame

To access the column data ,we can mention the column name as subscript.

e.g. - df[empid] This can also be done by using df.empid.

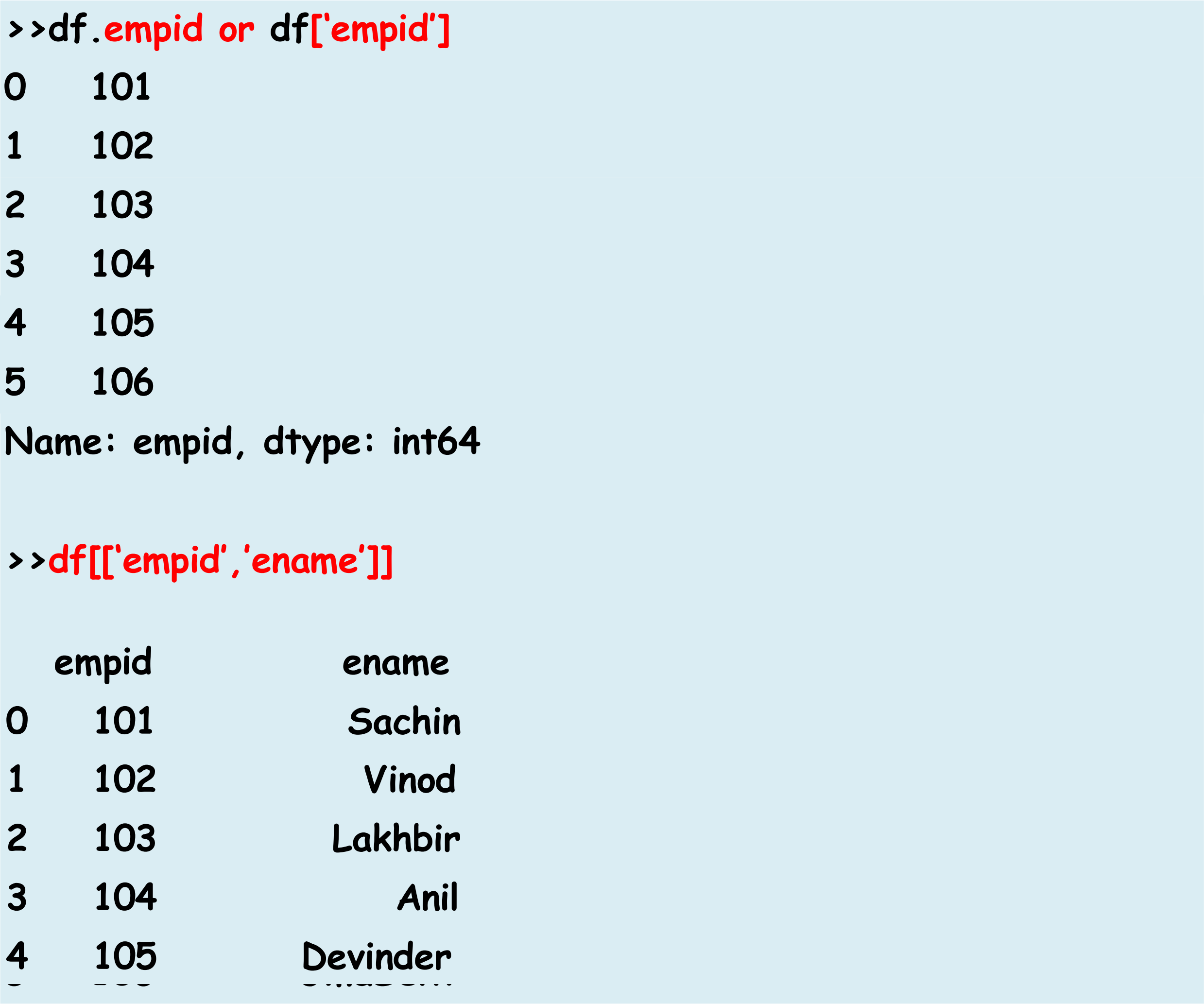
To access multiple columns we can write as df[[col1, col2,---]]

Example -



```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
        'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
```

```
   empid     ename          Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
```



```
>>df.empid or df['empid']
0      101
1      102
2      103
3      104
4      105
5      106
Name: empid, dtype: int64

>>df[['empid','ename']]

   empid            ename
0    101           Sachin
1    102            Vinod
2    103          Lakhbir
3    104             Anil
4    105         Devinder
```

To Add & Rename a column in data frame

```
import pandas as pd

s = pd.Series([10,15,18,22])

df=pd.DataFrame(s)

df.columns=[‘List1‘]
```

→ To Rename the default column of Data Frame as List1

```
df[‘List2’]=20
```

→ To create a new column List2 with all values as 20

```
df[‘List3’]=df[‘List1’]+df[‘List2’]
```

Add Column1 and Column2 and store in New column List3

```
print(df)
```

Output-

```
   List1  List2  List3
0    10     20     30
1    15     20     35
2    18     20     38
3    22     20     42
```

To Delete a Column in data frame

We can delete the column from a data frame by using any of the the following –

1. del
2. pop()
3. drop()

>>del df['List3'] ⟶ We can simply delete a column by passing column name in subscript with df

>>df

Output-

```
  List1  List2
0    10     20
1    15     20
2    18     20
3    22     20
```

>>df.pop('List2') ⟶ we can simply delete a column by passing column name in pop method.

>>df

```
   List1
0    10
1    15
2    18
3    22
```

To Delete a Column Using drop()

```
import pandas as pd
s= pd.Series([10,20,30,40])
df=pd.DataFrame(s)
df.columns=[‘List1‘]
df[‘List2’]=40
df1=df.drop(‘List2’,axis=1)  → (axis=1) means to delete Data
                                column wise
df2=df.drop(index=[2,3],axis=0)→   (axis=0) means to delete
                             data row wise with given index
print(df)
print(“ After deletion::”)
print(df1)
print (“ After row deletion::”)
print(df2)
```

Output-

```
   List1  List2
0    10     40
1    20     40
2    30     40
3    40     40
After deletion::
    List1
0    10
1    20
2    30
3    40
After row deletion::
   List1
0     10
```

1 20

Accessing the data frame through loc() and iloc() method or indexing using Labels

Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

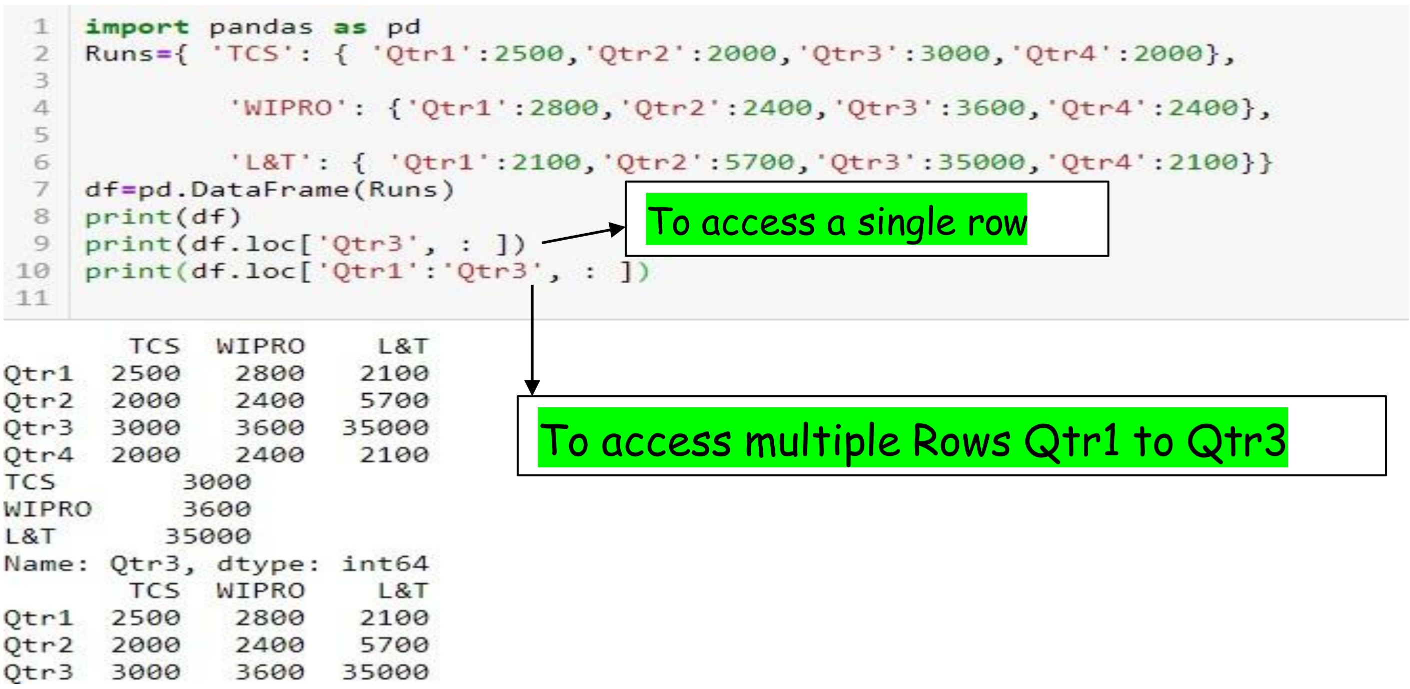
Accessing the data frame through loc()

It is used to access a group of rows and columns.

Syntax-

Df.loc[StartRow : EndRow, StartColumn : EndColumn]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.



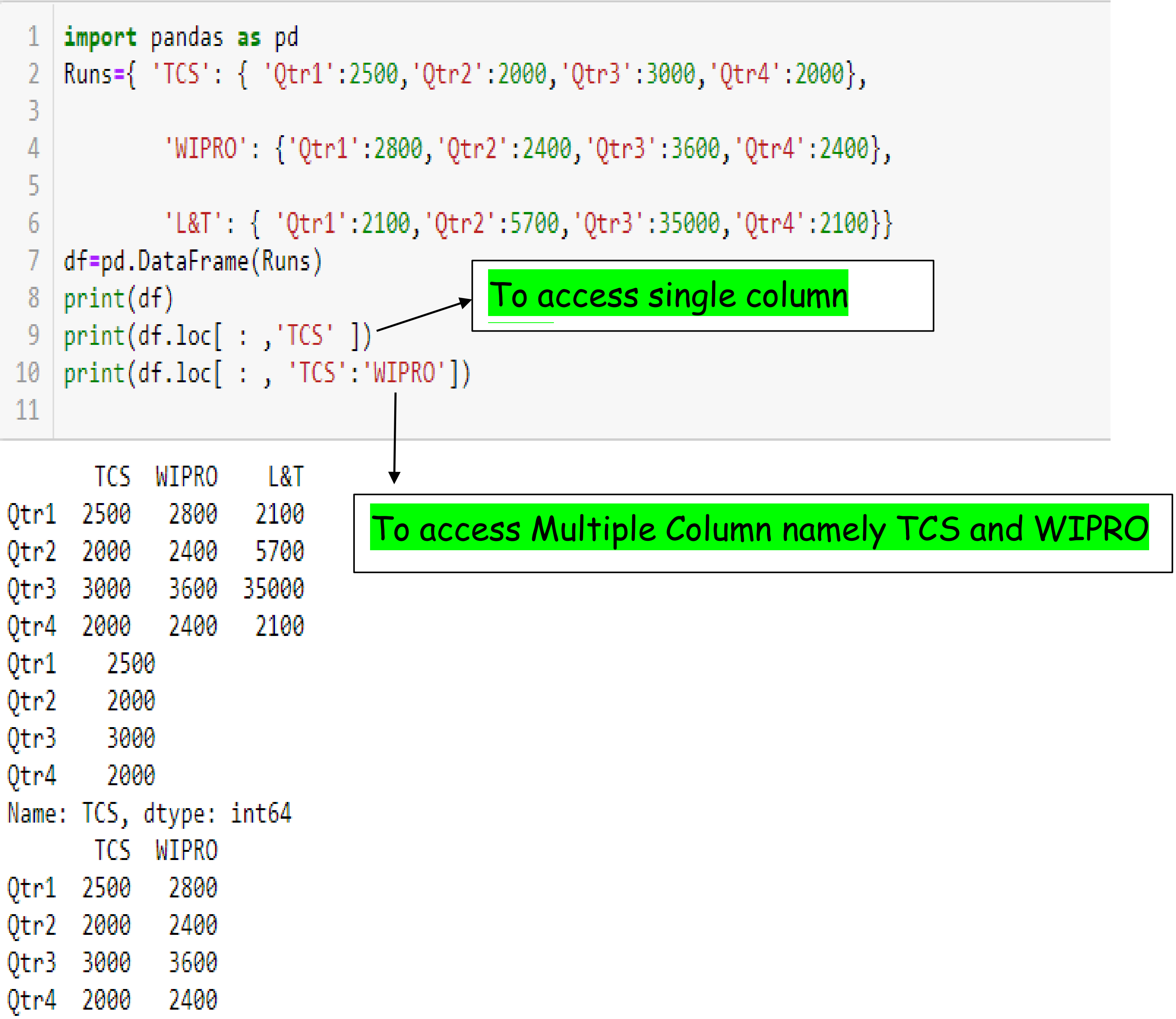
```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},

        'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df.loc['Qtr3', : ])
print(df.loc['Qtr1':'Qtr3', : ])

```

To access a single row

To access multiple Rows Qtr1 to Qtr3

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
TCS        3000
WIPRO      3600
L&T       35000
Name: Qtr3, dtype: int64
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
```

Example 2:-



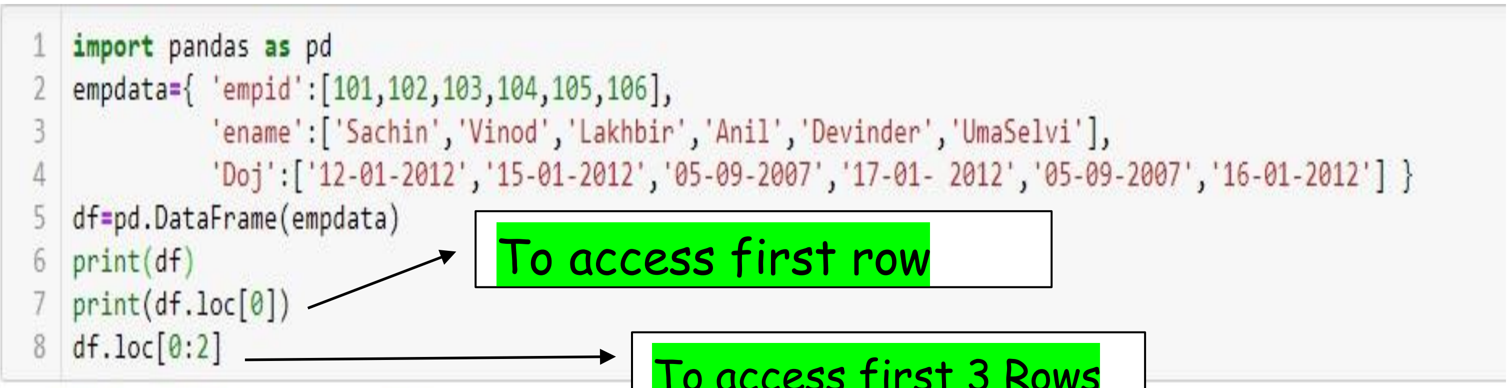
```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},

        'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df.loc[ : ,'TCS' ])
print(df.loc[ : , 'TCS':'WIPRO'])

```

To access single column

To access Multiple Column namely TCS and WIPRO

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
Qtr1    2500
Qtr2    2000
Qtr3    3000
Qtr4    2000
Name: TCS, dtype: int64
       TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

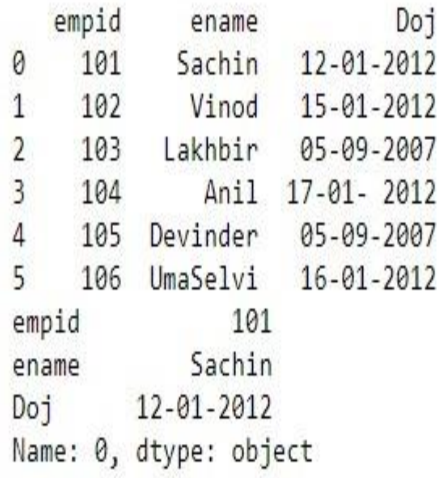
Example-3



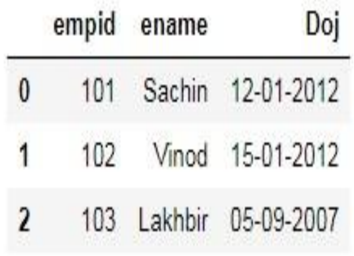
```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
          'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
print(df.loc[0])
df.loc[0:2]
```

To access first row

To access first 3 Rows



```
   empid     ename          Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
empid           101
ename        Sachin
Doj      12-01-2012
Name: 0, dtype: object
```



	empid	ename	Doj
0	101	Sachin	12-01-2012
1	102	Vinod	15-01-2012
2	103	Lakhbir	05-09-2007

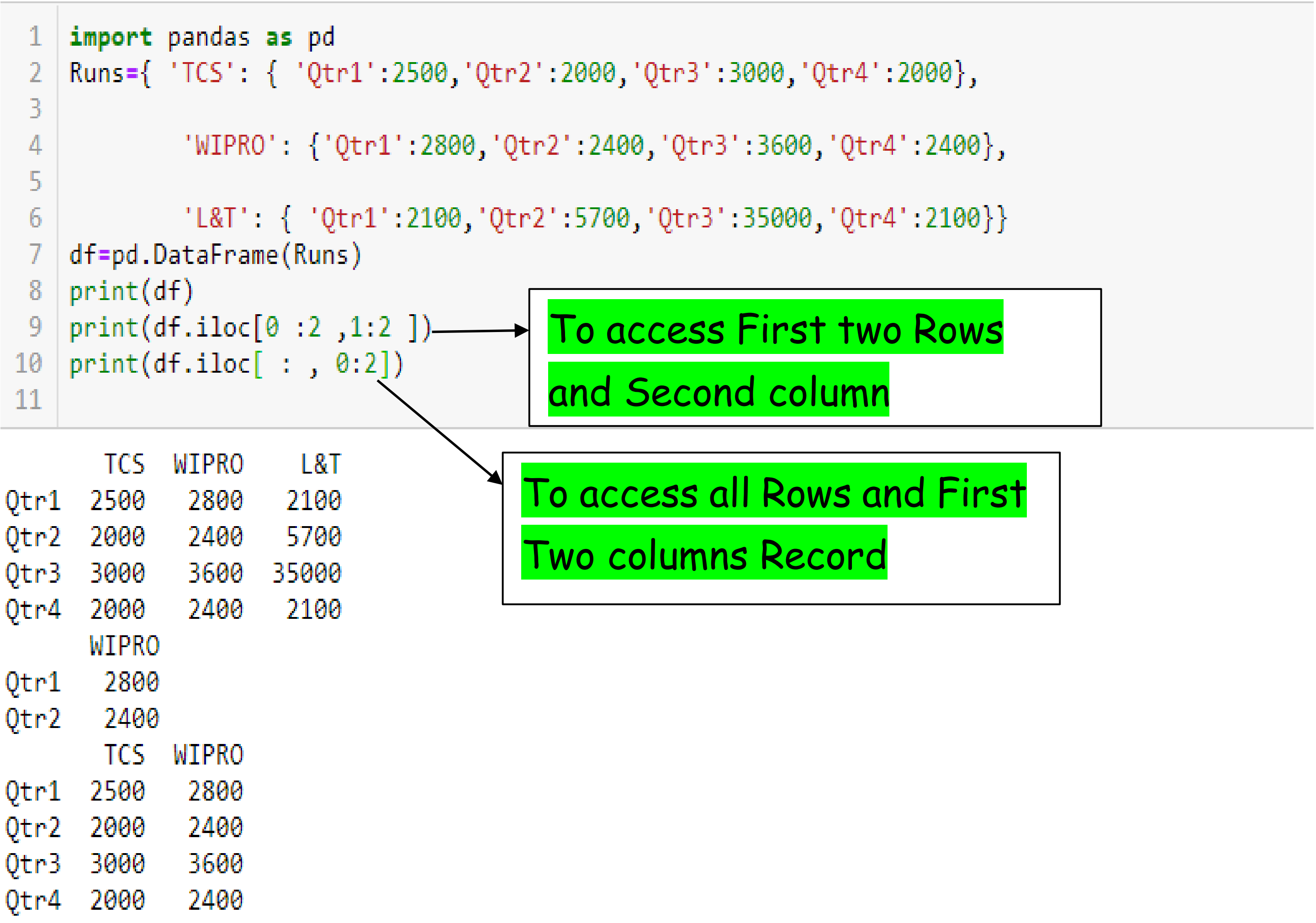
Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax-

Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.



```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},

        'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df.iloc[0 :2 ,1:2 ])
print(df.iloc[ : , 0:2])

```

To access First two Rows and Second column

To access all Rows and First Two columns Record

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
      WIPRO
Qtr1   2800
Qtr2   2400
       TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

head() and tail() Method

The method head() gives the first 5 rows and the method tail() returns the last 5 rows.

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                  '17-01-2012','05-09-2007','16-01-2012'],
             'empid':[101,102,103,104,105,106],
             'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
             }
df=pd.DataFrame(empdata)
print(df)
print(df.head())
print(df.tail())
```

Output-

```
          Doj  empid     ename
0  12-01-2012    101    Sachin
1  15-01-2012    102     Vinod
2  05-09-2007    103   Lakhbir  ──────→  Data Frame
3  17-01-2012    104      Anil
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

```
          Doj  empid     ename
0  12-01-2012    101    Sachin
1  15-01-2012    102     Vinod  ──────→  head() displays first 5 rows
2  05-09-2007    103   Lakhbir
3  17-01-2012    104      Anil
4  05-09-2007    105  Devinder
```

```
          Doj  empid     ename
1  15-01-2012    102     Vinod
2  05-09-2007    103   Lakhbir
3  17-01-2012    104      Anil  ──────→  tail() display last 5 rows
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

To display first 2 rows we can use head(2) and to returns last2 rows we can use tail(2) and to return 3rd to 4th row we can write df[2:5].

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                  '17-01-2012','05-09-2007','16-01-2012'],
             'empid':[101,102,103,104,105,106],
             'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
             }
df=pd.DataFrame(empdata)
print(df)
print(df.head(2))
print(df.tail(2))
print(df[2:5])
```

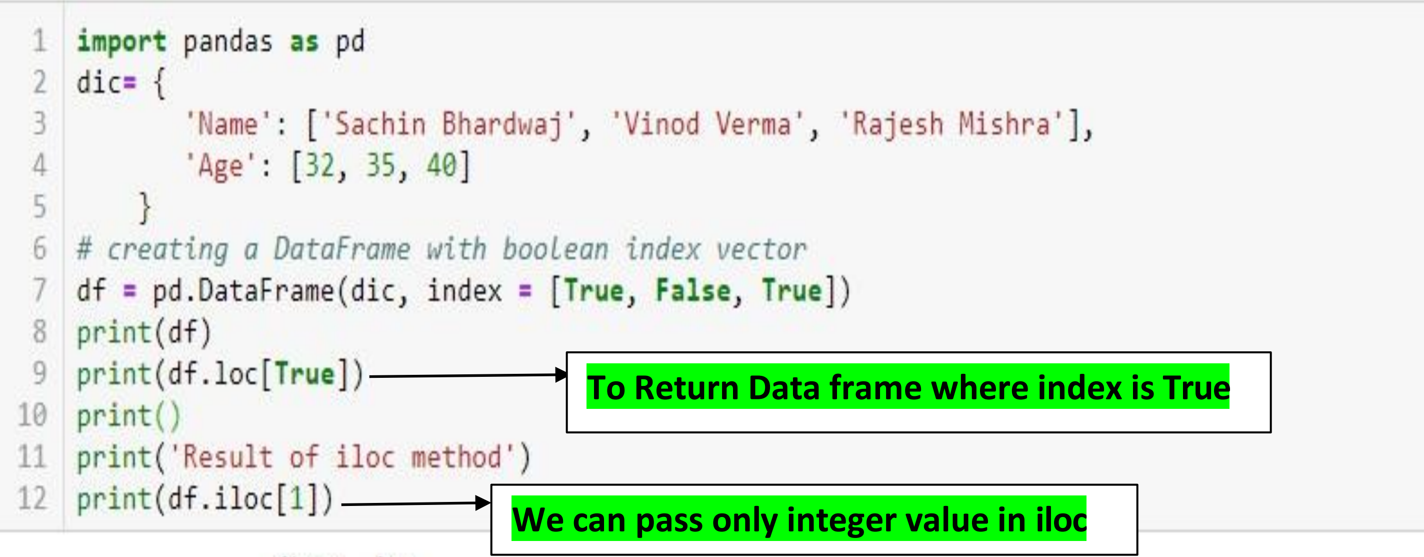
Output-

```
                Doj   empid     ename
0    12-01-2012     101      Sachin
1    15-01-2012     102       Vinod
2    05-09-2007     103     Lakhbir
3   17-01- 2012     104        Anil
4    05-09-2007     105    Devinder
5    16-01-2012     106    UmaSelvi

              Doj    empid   ename
0   12-01-2012      101   Sachin      ──────→   head(2) displays first 2 rows
1   15-01-2012      102    Vinod

               Doj   empid      ename
4   05-09-2007      105   Devinder     ──────→   tail(2) displays last 2 rows
5   16-01-2012      106   UmaSelvi
               Doj   empid     ename
2   05-09-2007      103    Lakhbir
3   17-01- 2012     104       Anil     ──────→   df[2:5] display 2nd to 4th row
```

4	05-09-2007	105	Devinder

Boolean Indexing in Data Frame

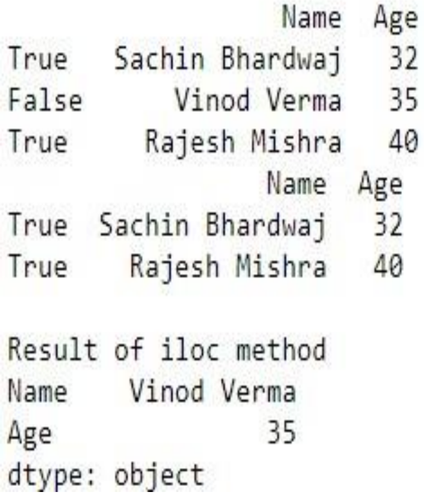
Boolean indexing helps us to select the data from the DataFrames using a boolean vector. We create a DataFrame with a boolean index to use the boolean indexing.



```
import pandas as pd
dic= {
        'Name': ['Sachin Bhardwaj', 'Vinod Verma', 'Rajesh Mishra'],
        'Age': [32, 35, 40]
    }
# creating a DataFrame with boolean index vector
df = pd.DataFrame(dic, index = [True, False, True])
print(df)
print(df.loc[True])
print()
print('Result of iloc method')
print(df.iloc[1])
```

To Return Data frame where index is True

We can pass only integer value in iloc



```
                  Name  Age
True   Sachin Bhardwaj   32
False      Vinod Verma   35
True     Rajesh Mishra   40
                 Name  Age
True  Sachin Bhardwaj   32
True    Rajesh Mishra   40

Result of iloc method
Name    Vinod Verma
Age              35
dtype: object
```

Concat operation in data frame

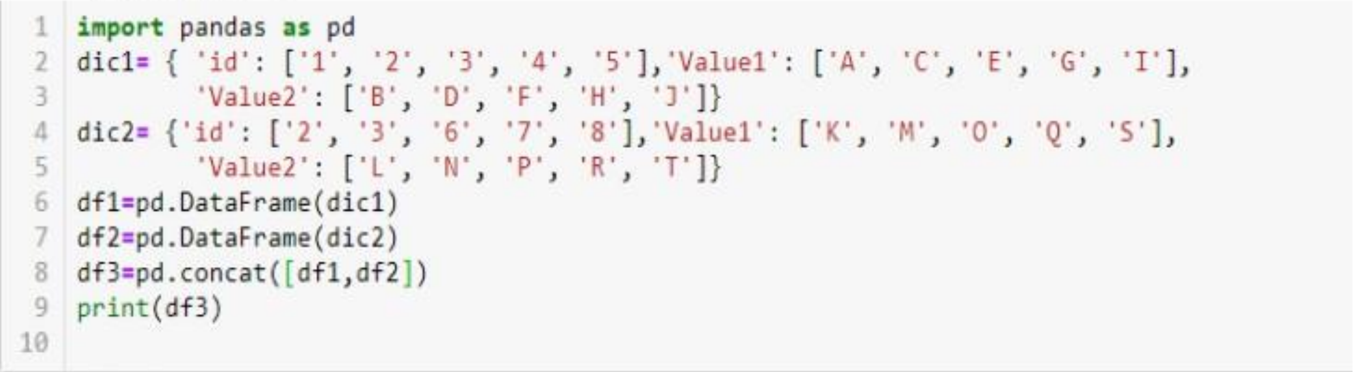
Pandas provides various facilities for easily combining together **Series, DataFrame.**

```
pd.concat(objs, axis=0, join='outer', join_axes=None,ignore_index=False)
```

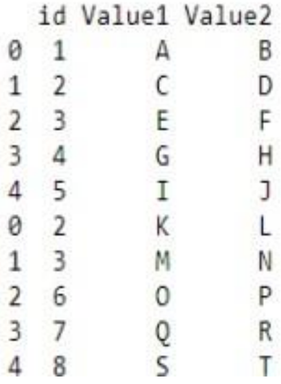
- **objs** – This is a sequence or mapping of Series, DataFrame, or Panel objects.
- **axis** – {0, 1, ...}, default 0. This is the axis to concatenate along.
- **join** – {‘inner’, ‘outer’}, default ‘outer’. How to handle indexes on other axis(es). Outer for union and inner for intersection.
- **ignore_index** – boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
- **join_axes** – This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

The Concat() performs concatenation operations along an axis.

Example-1

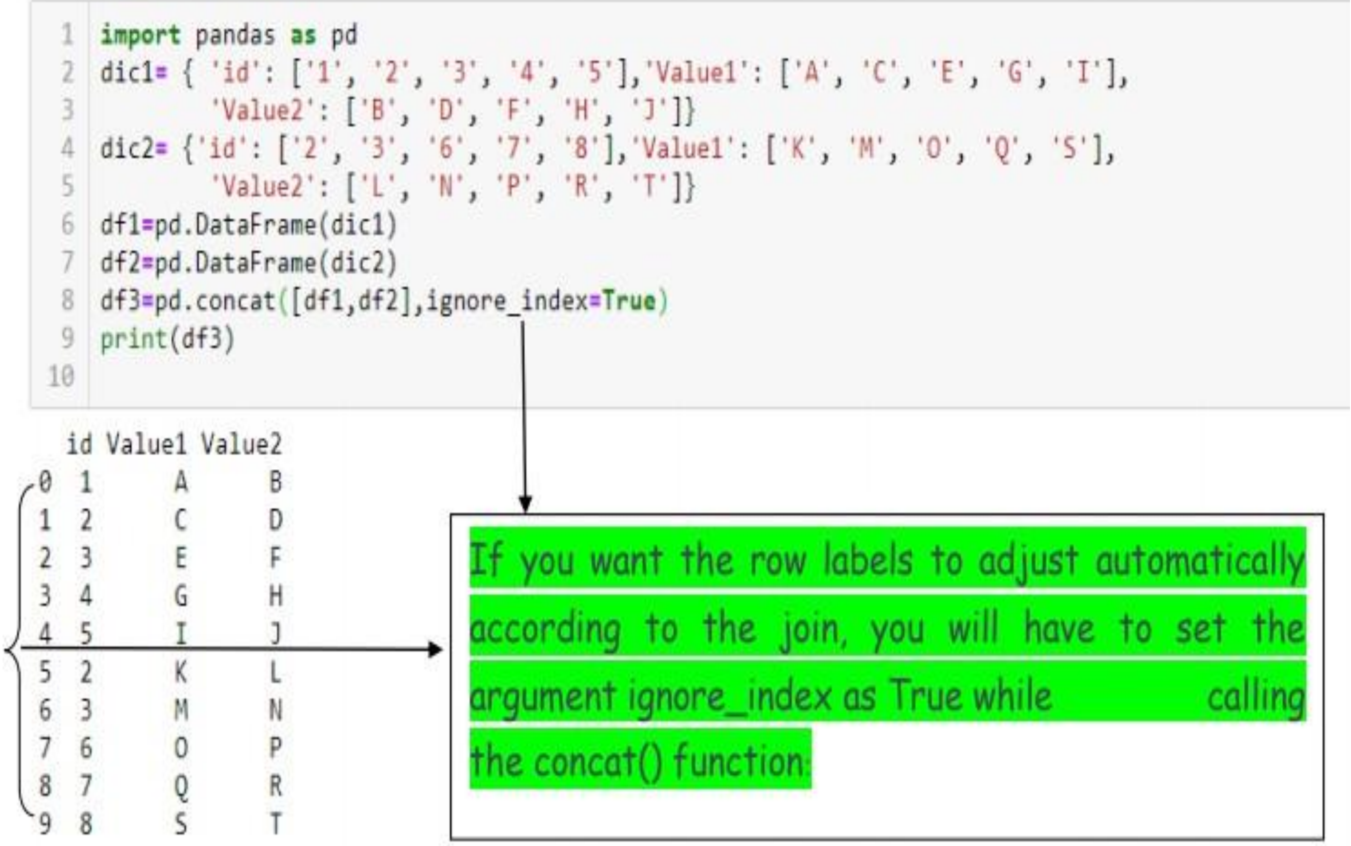


```python
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2])
print(df3)
```



```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
0  2      K      L
1  3      M      N
2  6      O      P
3  7      Q      R
4  8      S      T
```

Example-2

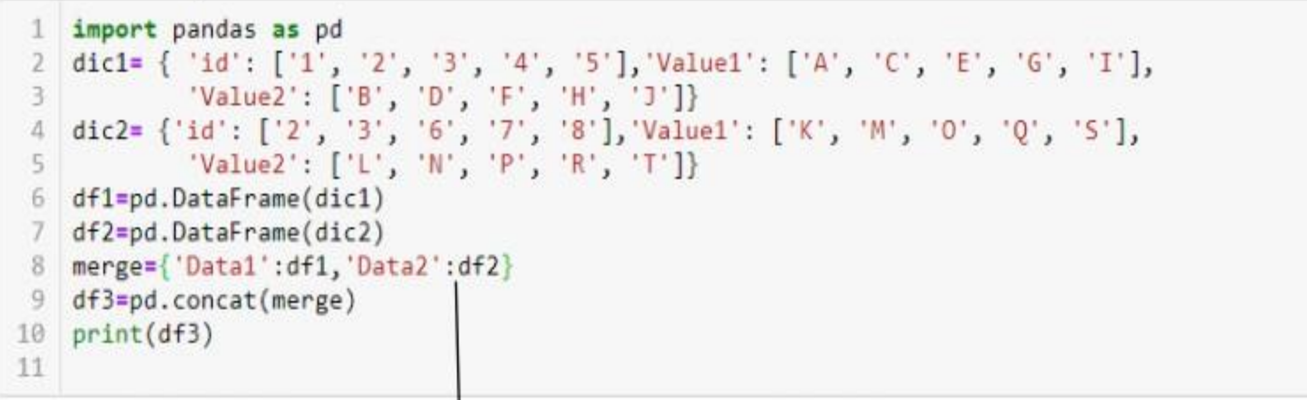


```python
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2],ignore_index=True)
print(df3)
```

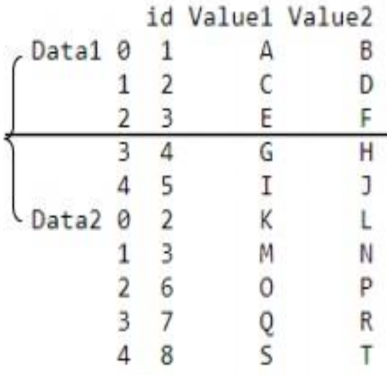
```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
5  2      K      L
6  3      M      N
7  6      O      P
8  7      Q      R
9  8      S      T
```

If you want the row labels to adjust automatically according to the join, you will have to set the argument ignore_index as True while calling the concat() function.

Example-3



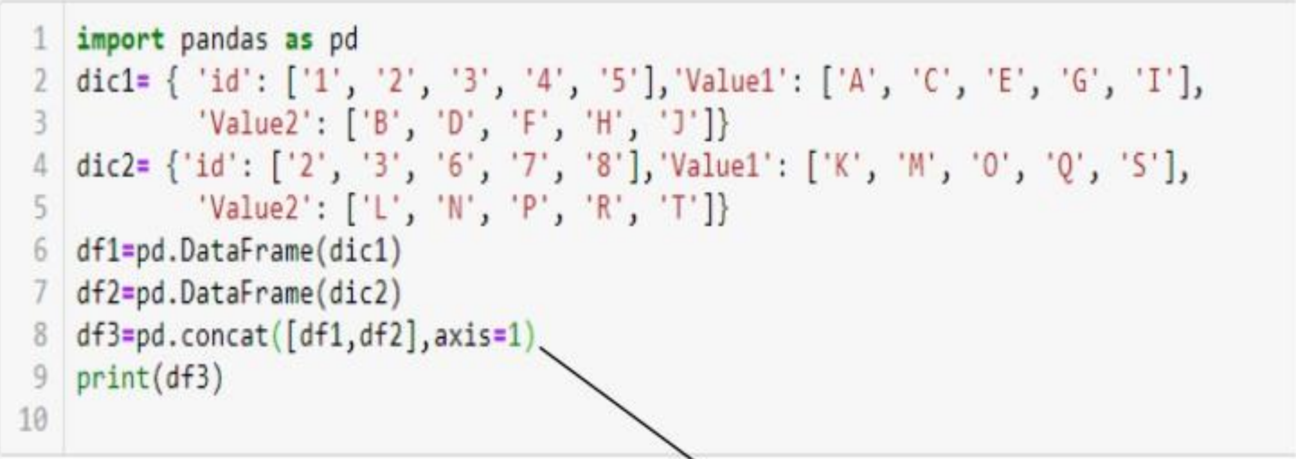
```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
merge={'Data1':df1,'Data2':df2}
df3=pd.concat(merge)
print(df3)
```



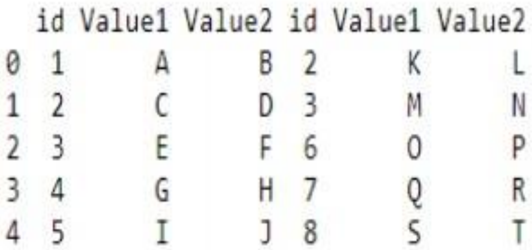
```
        id Value1 Value2
Data1 0  1      A      B
      1  2      C      D
      2  3      E      F
      3  4      G      H
      4  5      I      J
Data2 0  2      K      L
      1  3      M      N
      2  6      O      P
      3  7      Q      R
      4  8      S      T
```

pandas also provides you with an option to label the DataFrames, after the concatenation, with a key so that you may know which data came from which DataFrame.

Example-4



```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2],axis=1)
print(df3)
```



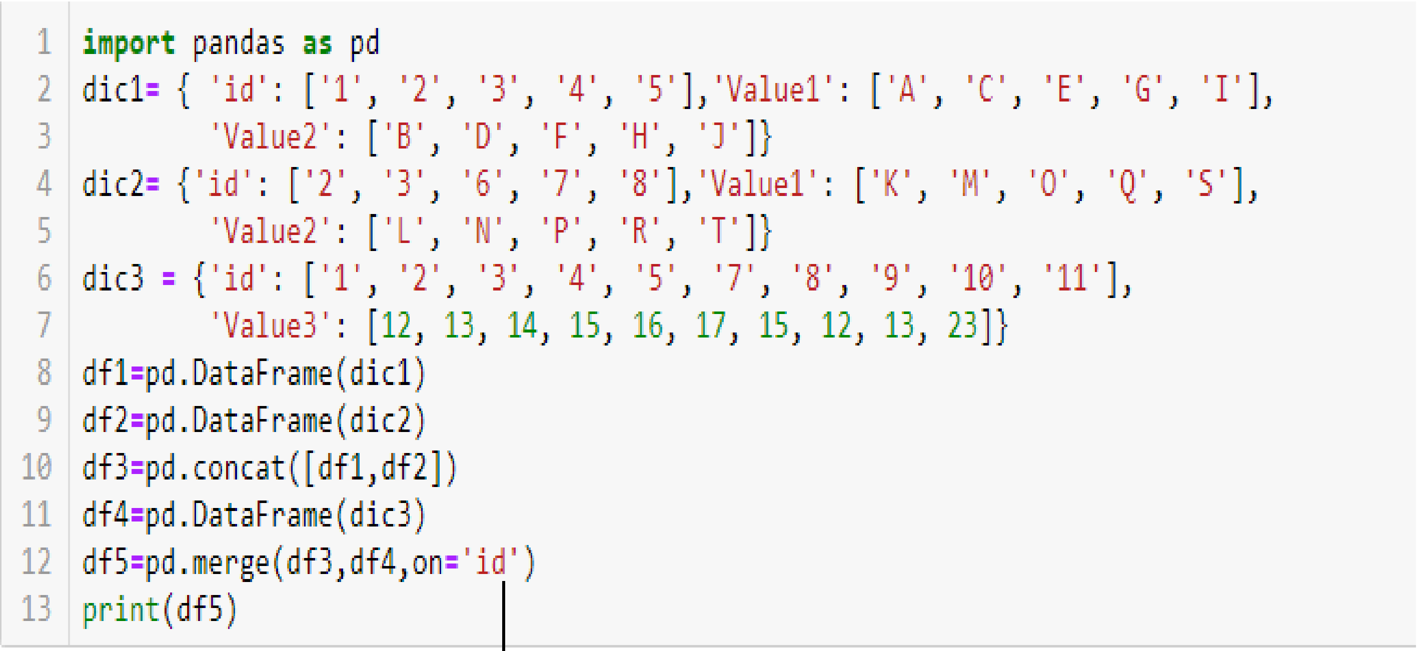
```
  id Value1 Value2 id Value1 Value2
0  1      A      B  2      K      L
1  2      C      D  3      M      N
2  3      E      F  6      O      P
3  4      G      H  7      Q      R
4  5      I      J  8      S      T
```

To concatenate DataFrames along column, you can specify the axis parameter as 1.

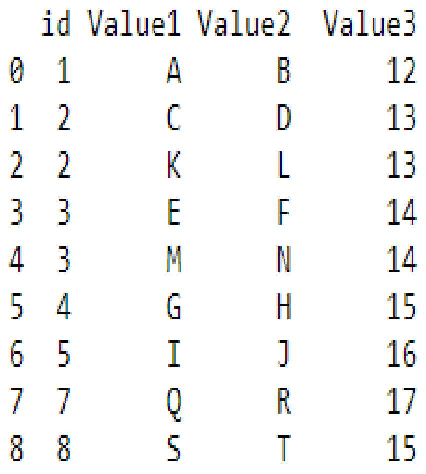
Merge operation in data frame

Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column. To join these DataFrames, pandas provides multiple functions like merge(), join() etc.

Example-1



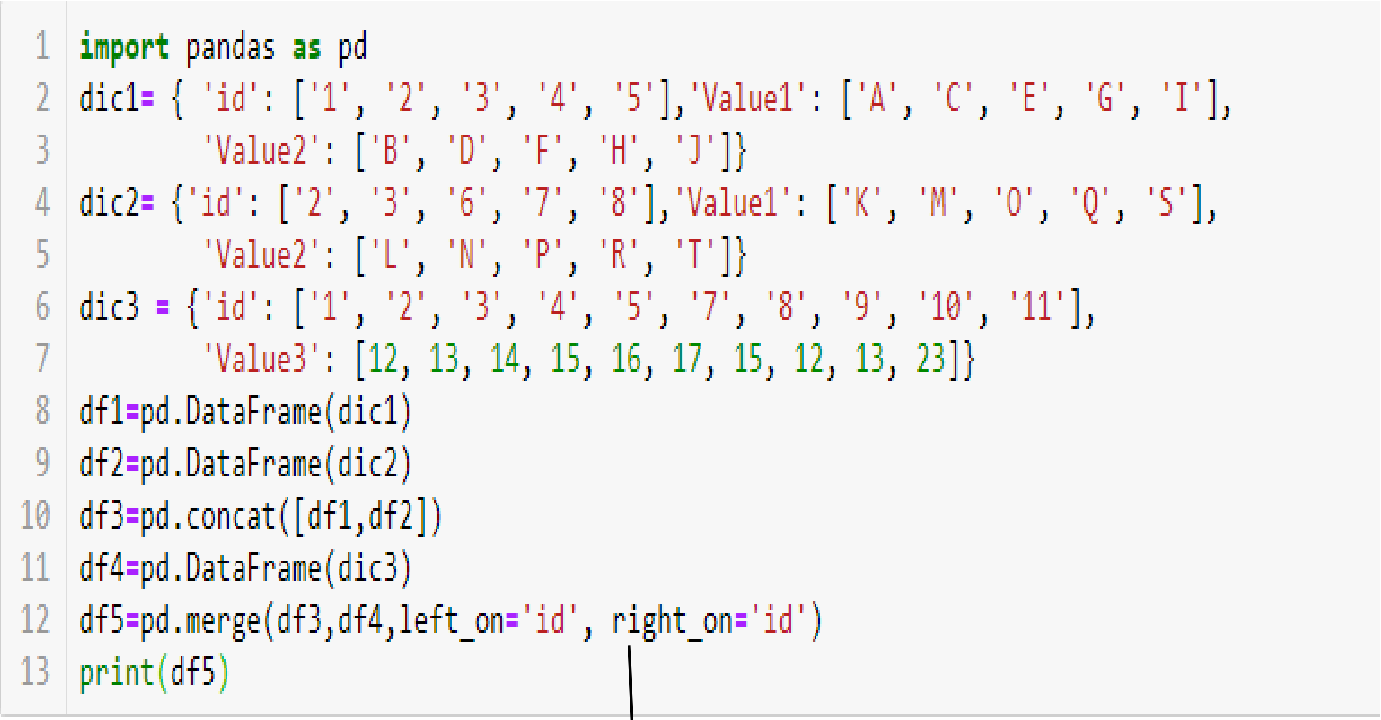
```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2])
df4=pd.DataFrame(dic3)
df5=pd.merge(df3,df4,on='id')
print(df5)
```



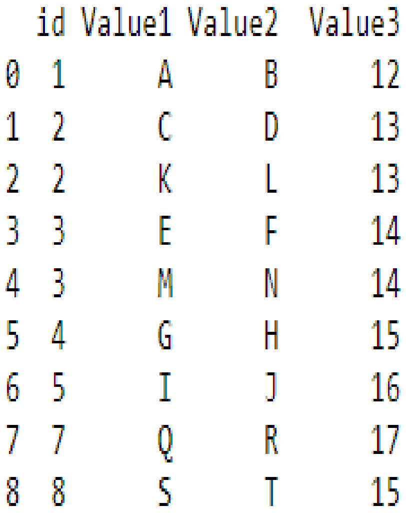
```
  id Value1 Value2  Value3
0  1      A      B      12
1  2      C      D      13
2  2      K      L      13
3  3      E      F      14
4  3      M      N      14
5  4      G      H      15
6  5      I      J      16
7  7      Q      R      17
8  8      S      T      15
```

This will give the common rows between the two data frames for the corresponding column values ('id').

Example-2



```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2])
df4=pd.DataFrame(dic3)
df5=pd.merge(df3,df4,left_on='id', right_on='id')
print(df5)
```



```
  id Value1 Value2  Value3
0  1      A      B      12
1  2      C      D      13
2  2      K      L      13
3  3      E      F      14
4  3      M      N      14
5  4      G      H      15
6  5      I      J      16
7  7      Q      R      17
8  8      S      T      15
```

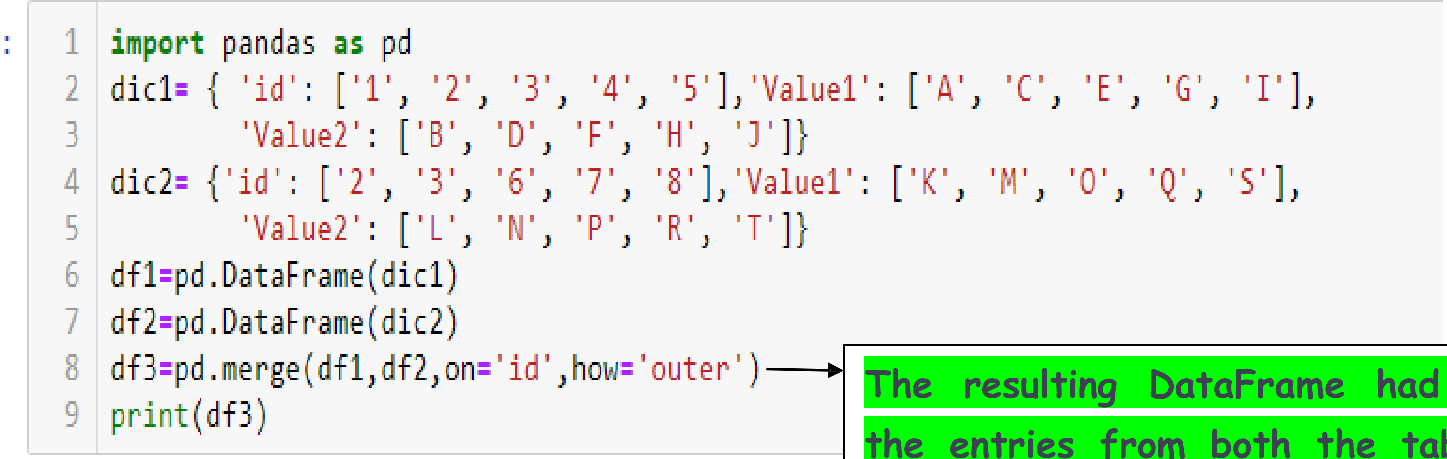
It might happen that the column on which you want to merge the Data Frames have different names (unlike in this case). For such merges, you will have to specify the arguments left_on as the left DataFrame name and right_on as the right DataFrame name.

Join operation in data frame

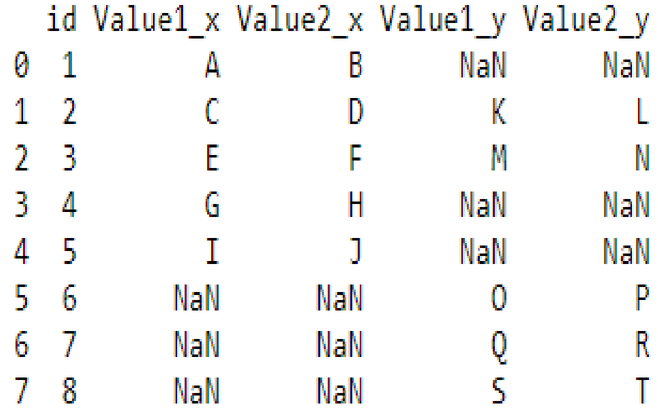
It is used to merge data frames based on some common column/key.

1. Full Outer Join:- The full outer join combines the results of both the left and the right outer joins. The joined data frame will contain all records from both the data frames and fill in NaNs for missing matches on either side. You can perform a full outer join by specifying the how argument as outer in merge() function.

Example-



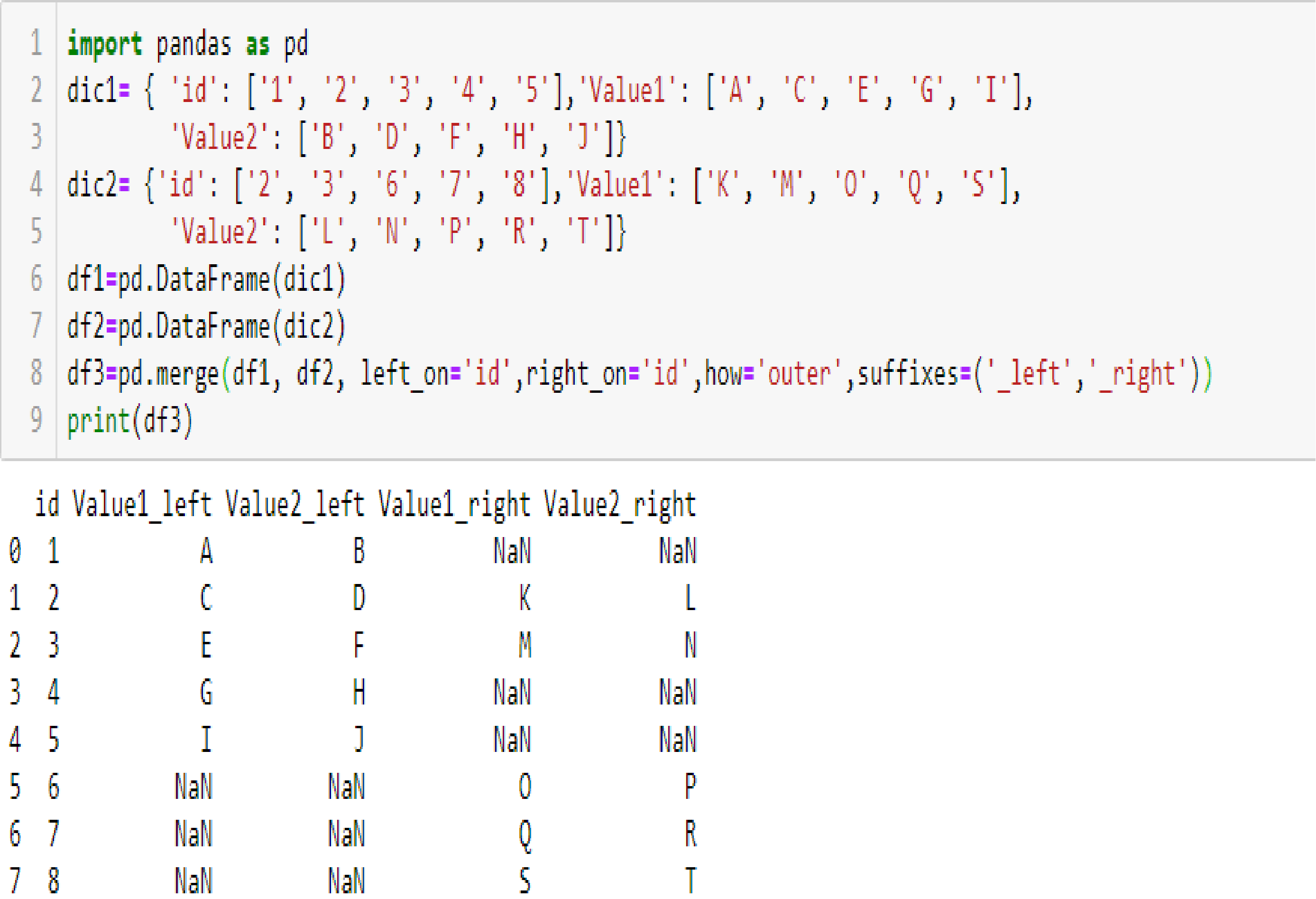
```python
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
         'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
         'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1,df2,on='id',how='outer')
print(df3)
```



```
  id Value1_x Value2_x Value1_y Value2_y
0  1        A        B      NaN      NaN
1  2        C        D        K        L
2  3        E        F        M        N
3  4        G        H      NaN      NaN
4  5        I        J      NaN      NaN
5  6      NaN      NaN        O        P
6  7      NaN      NaN        Q        R
7  8      NaN      NaN        S        T
```

The resulting DataFrame had all the entries from both the tables with NaN values for missing matches on either side. However, one more thing to notice is the suffix which got appended to the column names to show which column came from which DataFrame. The default suffixes are x and y, however, you can modify them by specifying the suffixes argument in the merge() function.

Example-2

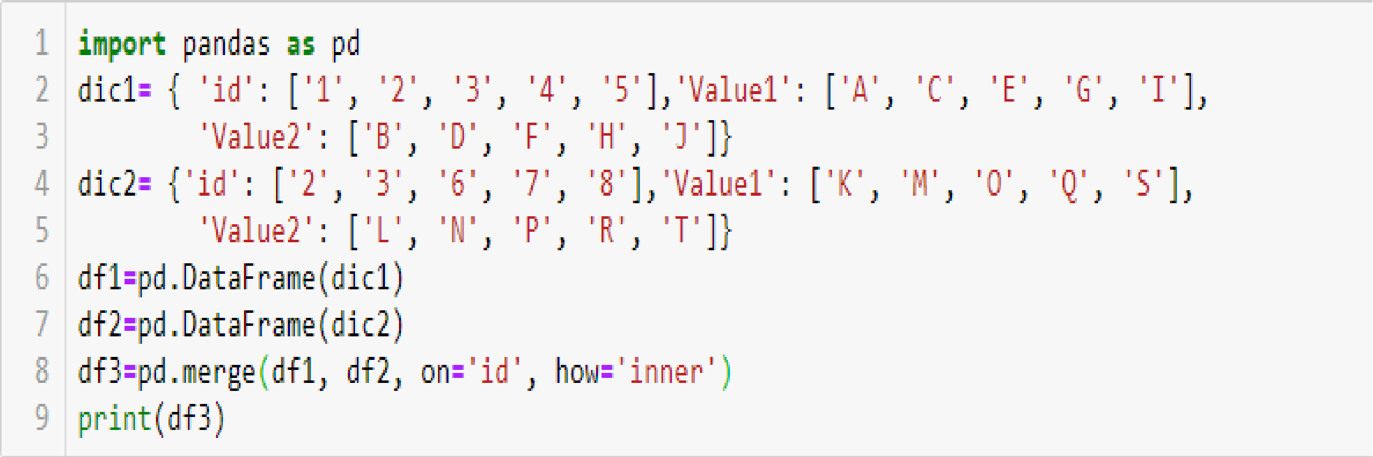


```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, left_on='id',right_on='id',how='outer',suffixes=('_left','_right'))
print(df3)
```

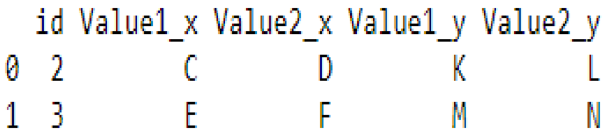
```
  id Value1_left Value2_left Value1_right Value2_right
0  1           A           B          NaN          NaN
1  2           C           D            K            L
2  3           E           F            M            N
3  4           G           H          NaN          NaN
4  5           I           J          NaN          NaN
5  6         NaN         NaN            O            P
6  7         NaN         NaN            Q            R
7  8         NaN         NaN            S            T
```

2. Inner Join :- The inner join produce only those records that match in both the data frame. You have to pass inner in how argument inside merge() function.

Example-



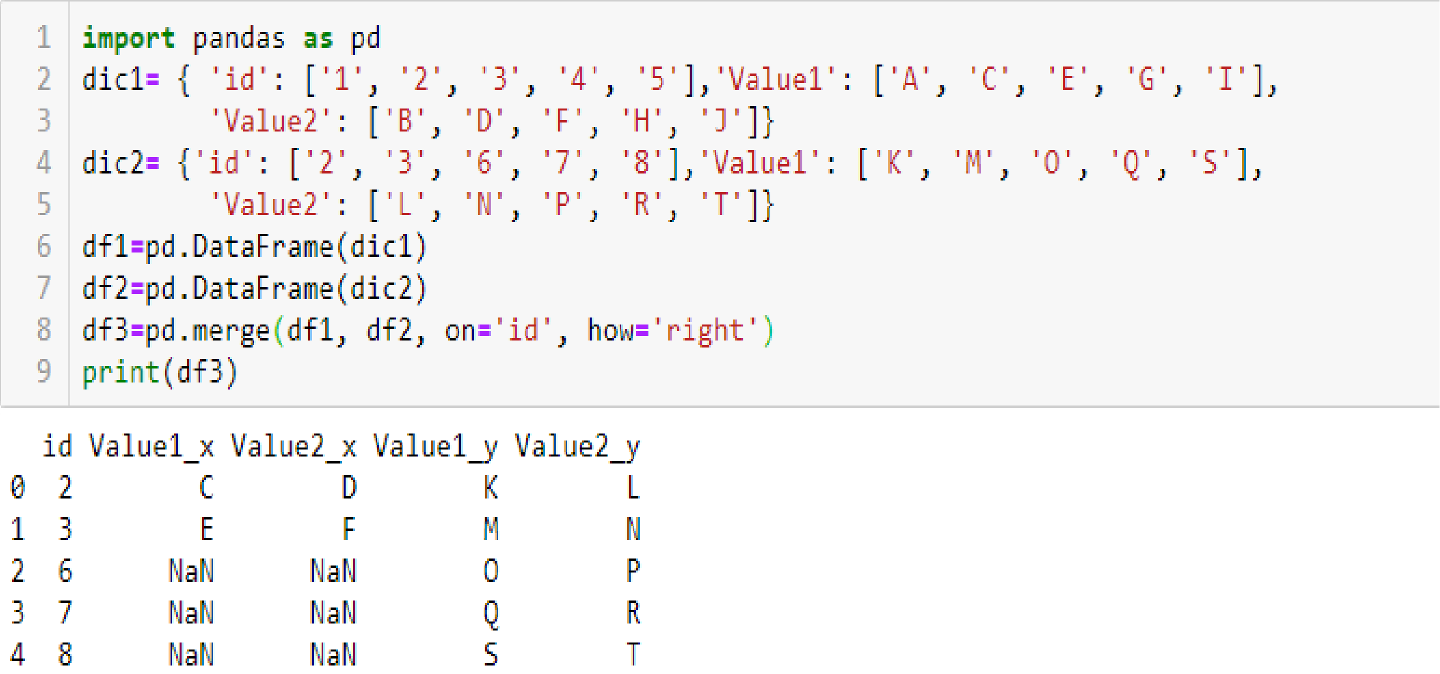
```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='inner')
print(df3)
```



```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
```

3. RightJoin :-The right join produce a complete set of records from data frame B(Right side Data Frame) with the matching records (where available) in data frame A(Left side data frame). If there is no match right side will contain null. You have to pass right in how argument inside merge() function.

Example-

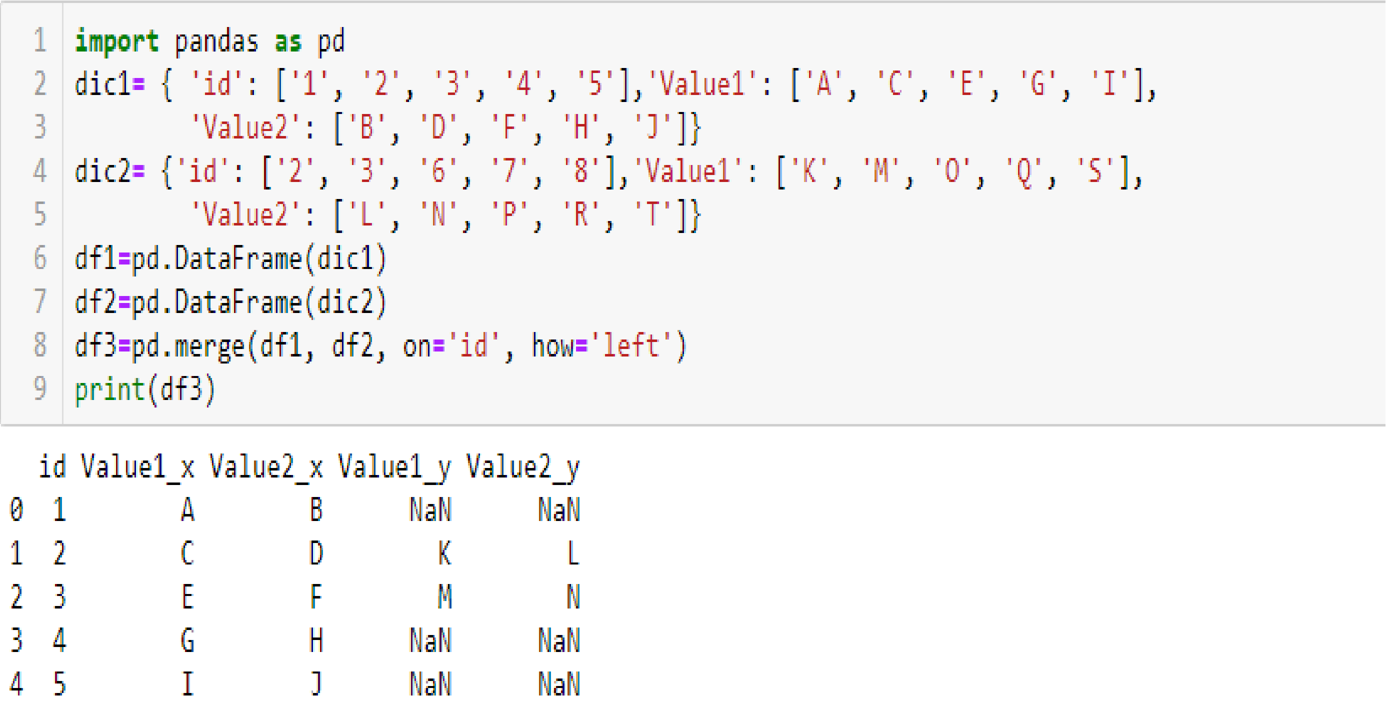


```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='right')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
2  6      NaN      NaN        O        P
3  7      NaN      NaN        Q        R
4  8      NaN      NaN        S        T
```

4. Left Join :- The left join produce a complete set of records from data frame A(Left side Data Frame) with the matching records (where available) in data frame B(Right side data frame). If there is no match left side will contain null. You have to pass left in how argument inside merge() function.

Example-

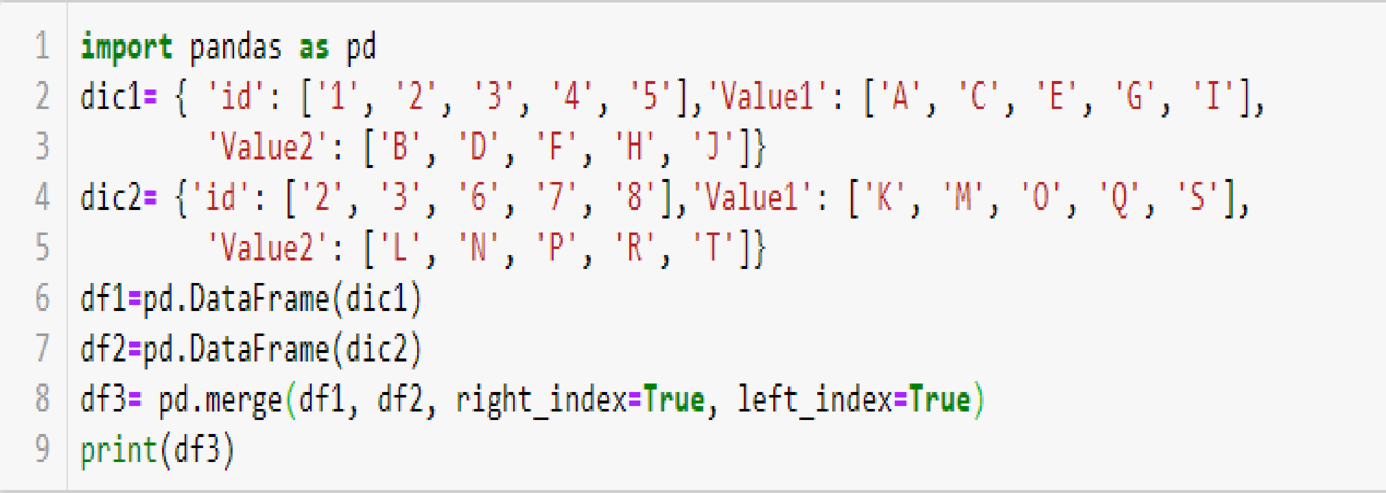


```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='left')
print(df3)
```

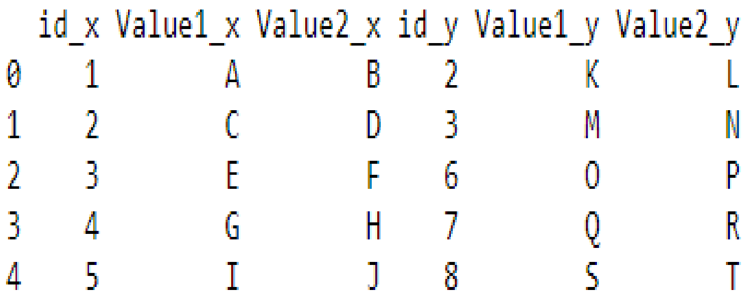
```
  id Value1_x Value2_x Value1_y Value2_y
0  1        A        B      NaN      NaN
1  2        C        D        K        L
2  3        E        F        M        N
3  4        G        H      NaN      NaN
4  5        I        J      NaN      NaN
```

5. Joining on Index :-Sometimes you have to perform the join on the indexes or the row labels. For that you have to specify right_index(for the indexes of the right data frame) and left_index(for the indexes of left data frame) as True.

Example-



```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3= pd.merge(df1, df2, right_index=True, left_index=True)
print(df3)
```



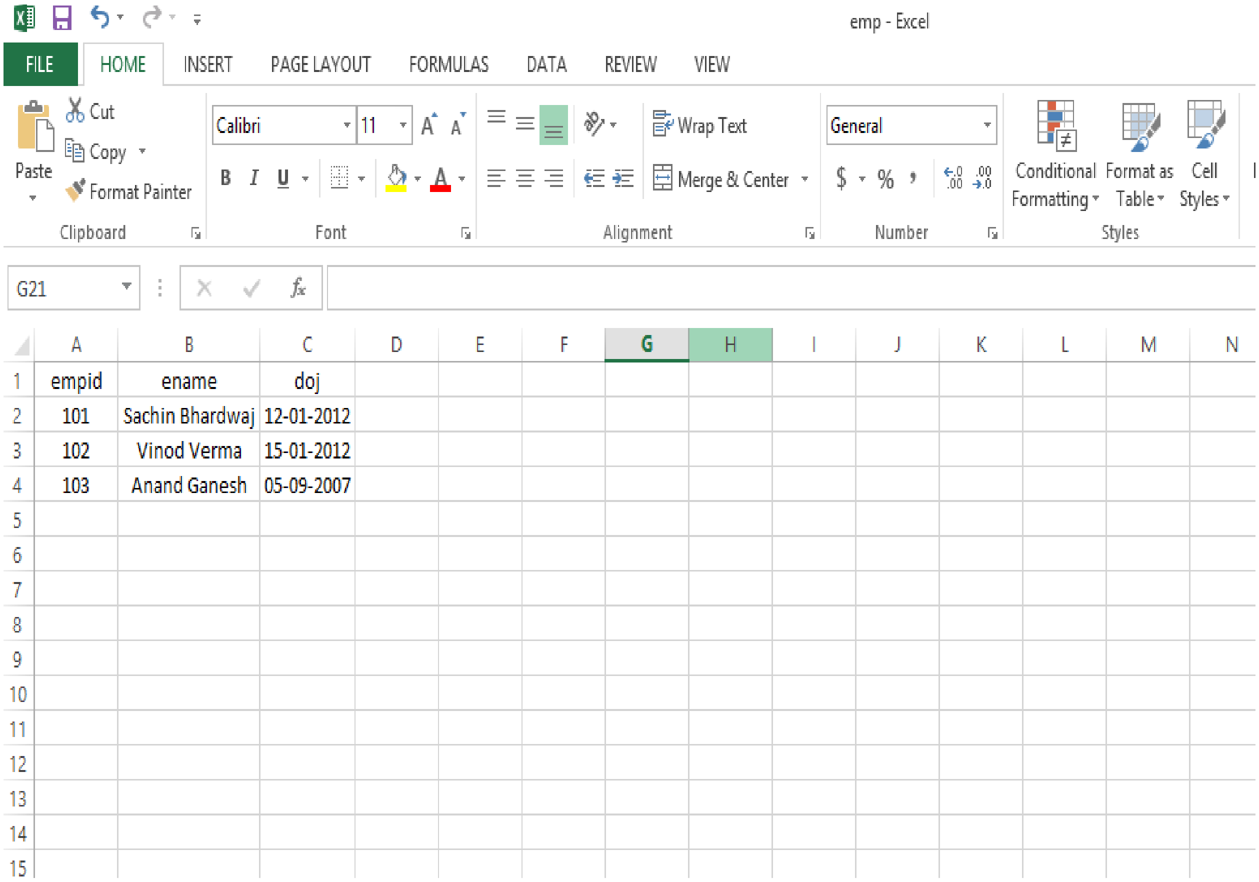
```
  id_x Value1_x Value2_x id_y Value1_y Value2_y
0    1        A        B    2        K        L
1    2        C        D    3        M        N
2    3        E        F    6        O        P
3    4        G        H    7        Q        R
4    5        I        J    8        S        T
```

CSV File

A CSV is a comma separated values file, which allows data to be saved in a tabular format. CSV is a simple file such as a spreadsheet or database. Files in the csv format can be imported and exported from programs that store data in tables, such as Microsoft excel or Open Office.

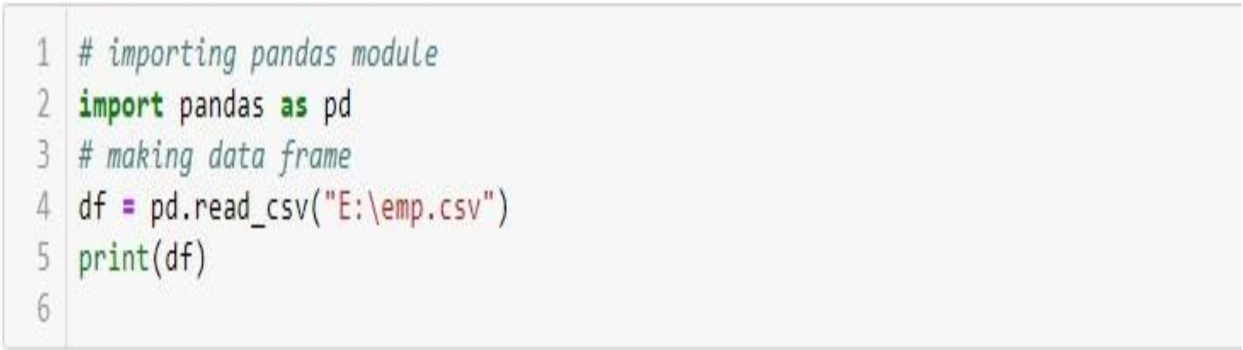
CSV files data fields are most often separated, or delimited by a comma. Here the data in each row are delimited by comma and individual rows are separated by newline.

To create a csv file, first choose your favorite text editor such as- Notepad and open a new file. Then enter the text data you want the file to contain, separating each value with a comma and each row with a new line. Save the file with the extension.csv. You can open the file using MS Excel or another spread sheet program. It will create the table of similar data.

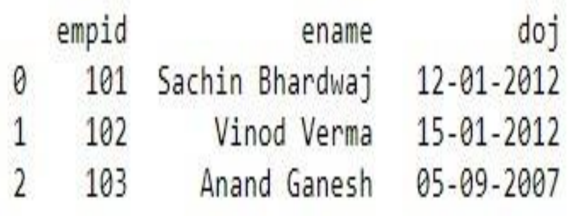


	A	B	C
1	empid	ename	doj
2	101	Sachin Bhardwaj	12-01-2012
3	102	Vinod Verma	15-01-2012
4	103	Anand Ganesh	05-09-2007

pd.read_csv() method is used to read a csv file.



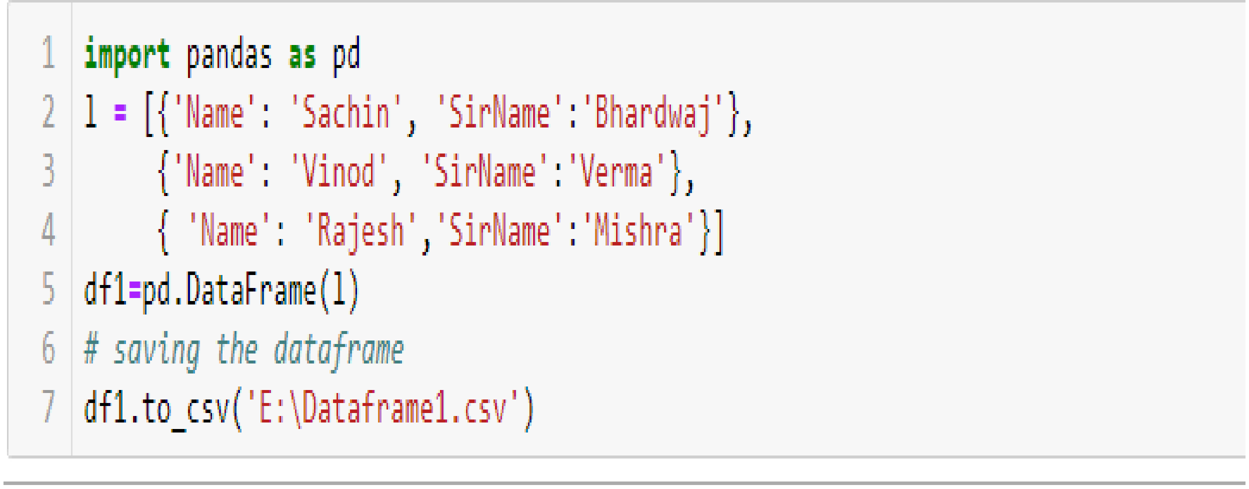
```python
# importing pandas module
import pandas as pd
# making data frame
df = pd.read_csv("E:\emp.csv")
print(df)

```



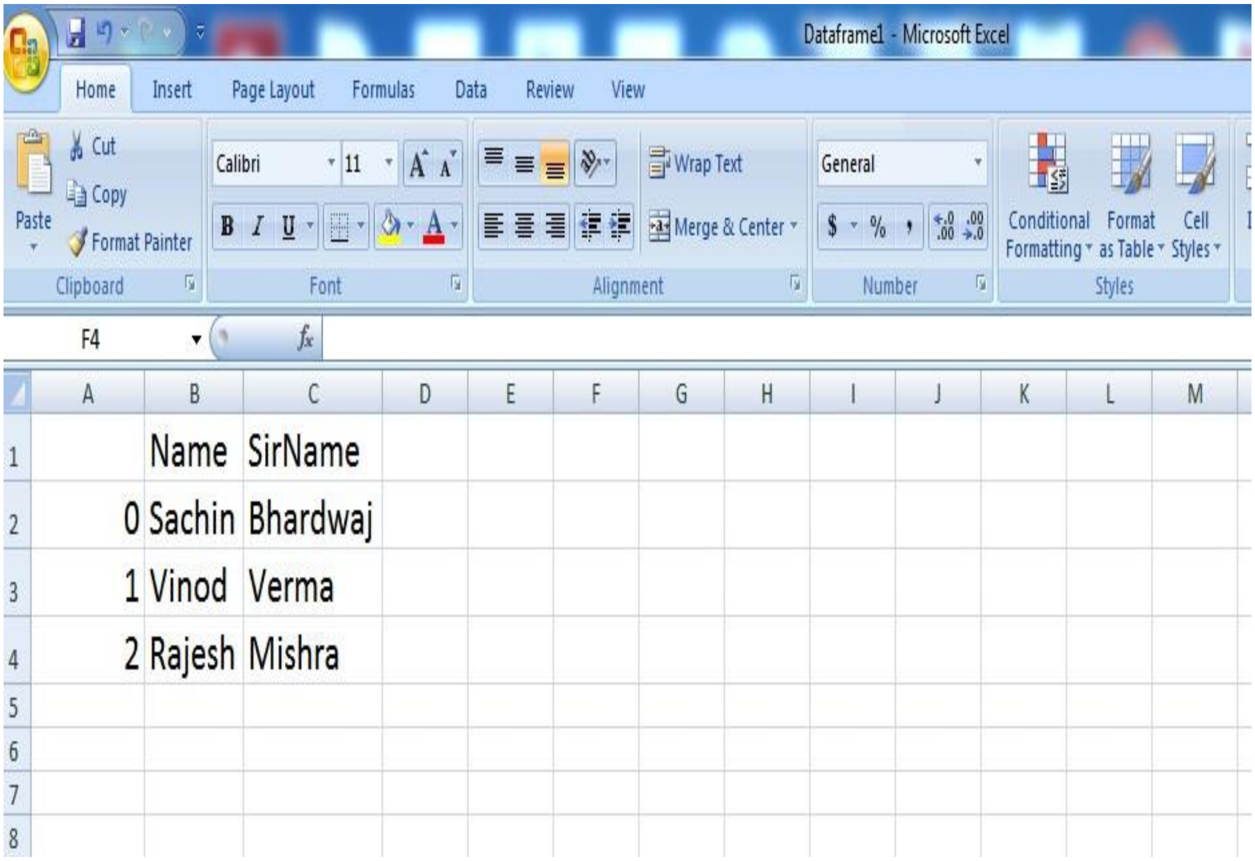
```
   empid            ename         doj
0    101  Sachin Bhardwaj  12-01-2012
1    102      Vinod Verma  15-01-2012
2    103     Anand Ganesh  05-09-2007
```

Exporting data from dataframe to CSV File

To export a data frame into a csv file first of all, we create a data frame say df1 and use dataframe.to_csv(‘ E:\Dataframe1.csv ’) method to export data frame df1 into csv file Dataframe1.csv.



```
import pandas as pd
l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
     {'Name': 'Vinod', 'SirName':'Verma'},
     { 'Name': 'Rajesh','SirName':'Mishra'}]
df1=pd.DataFrame(l)
# saving the dataframe
df1.to_csv('E:\Dataframe1.csv')
```



	A	B	C
1		Name	SirName
2	0	Sachin	Bhardwaj
3	1	Vinod	Verma
4	2	Rajesh	Mishra

And now the content of df1 is exported to csv file Dataframe1.