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

●Series objects

○ 1D array, similar to a column in a spreadsheet

● DataFrame objects

○ 2D table, similar to a spreadsheet

● Panel objects

○ Dictionary of DataFrames

```
#Series Object
#Creating a series
import pandas as pd
s = pd.Series([2,-1,3,5])
print(s)
```

```
0    2
1   -1
2    3
3    5
dtype: int64
```

```
#Pass as parameter to numpy functions
import numpy as np
np.square(s)
```

```
0     4
1     1
2     9
3    25
dtype: int64
```

```
#Arithmetic operations
s + [1000,2000,3000,4000]
```

```
0    1002
1    1999
2    3003
3    4005
dtype: int64
```

```
#broadcasting
s+1000
```

```
0    1002
1     999
2    1003
3    1005
dtype: int64
```

```
#binary and conditional operations
s<0
```

```
0    False
1     True
```

```
2    False
3    False
dtype: bool
```

```
#Index Labels
s1=pd.Series([56,23,200,87])
print(s1)

#setting index manually
s1=pd.Series([56,23,200,87],index=["alice","bob","joy","celen"])
print(s1)
```

```
0     56
1     23
2    200
3     87
dtype: int64
alice     56
bob       23
joy      200
celen     87
dtype: int64
```

```
#Accessing items in a series
#Specifying the index
print(s1[2])

#Specifying the label
print(s1["bob"])
```

```
200
23
```

```
#Accessing using iloc attribute
print(s1.iloc[2])

#Accessing using the loc attribute
print(s1.loc["bob"])
```

```
200
23
```

```
#Creating series from python dictionary
weights = {"alice": 68, "bob": 83, "colin": 86,
"darwin": 68}
s2 = pd.Series(weights)
print(s2)
```

```
alice     68
bob       83
colin     86
darwin    68
dtype: int64
```

```
#Automatic allignment- When an operation involves multiple Series objects Pandas automatically aligns items
print(s1+s2)
```

```
alice     124.0
bob       106.0
celen       NaN
colin       NaN
```

```
darwin      NaN
joy         NaN
dtype: float64
```

```
#Do not forget to set the right index labels, else you may get surprising results
s3=pd.Series([1000,1000,1000,1000])
print(s1+s3)
```

```
0       NaN
1       NaN
2       NaN
3       NaN
alice   NaN
bob     NaN
celen   NaN
joy     NaN
dtype: float64
```

```
s4=pd.Series(42,["alice","bob","celen"])
print(s4)
```

```
alice    42
bob      42
celen    42
dtype: int64
```

```
#Series object can have a name
s5 = pd.Series([83, 68], index=["bob", "alice"],
name="weights")
print(s5)
```

```
bob      83
alice    68
Name: weights, dtype: int64
```

```
#Plotting a series

%matplotlib inline
import matplotlib.pyplot as plt
temperatures =[4.4,5.1,6.1,6.2,6.1,6.1,5.7,5.2,4.7,4.1,3.9,3.5]
s6 = pd.Series(temperatures, name="Temperature")
s6.plot()
plt.show()
```


```
df=pd.DataFrame(data=np.array([[1,2,3],[4,5,6],[7,8,9]],dtype=int),columns=['A','B','C'])
df
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |

```
df.columns
```

```
Index(['A', 'B', 'C'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
df['A']
```

```
0    1
1    4
2    7
Name: A, dtype: int64
```

```
df.A
#df.A is often confused with the dataframe methods.So the convention which is normally used is df['A']
```

```
0    1
1    4
2    7
Name: A, dtype: int64
```

```
type(df['A'])
```

```
pandas.core.series.Series
```

```
df[['A','B']]
```

| | A | B |
|---|---|---|
| **0** | 1 | 2 |
| **1** | 4 | 5 |
| **2** | 7 | 8 |

```
df['new']=df['A']+df['B']
df
```

| | A | B | C | new |
|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 3 |

```
#Removing columns
df.drop('new',axis=1)
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |

```
df
```

| | A | B | C | new |
|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 3 |
| **1** | 4 | 5 | 6 | 9 |
| **2** | 7 | 8 | 9 | 15 |

```
#Removing columns
df.drop(['new'],axis=1,inplace=True)
df
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |

```
#Removing rows
df2=df.drop(df.index[1],axis=0)
df2
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **2** | 7 | 8 | 9 |

```
df
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |

```
people_dict = {
"weight": pd.Series([68, 83, 112],index=["alice","bob", "charles"]),
"birthyear": pd.Series([1984, 1985, 1992],
index=["bob", "alice", "charles"], name="year"),
```

```
"children": pd.Series([0, 3], index=["charles","bob"]),
"hobby": pd.Series(["Biking", "Dancing"],
index=["alice", "bob"]),
}


"""
• Series were automatically aligned based on their index
• Missing values are represented as NaN
• Series names are ignored (the name "year" was dropped)
"""
people = pd.DataFrame(people_dict)
people
```

| | weight | birthyear | children | hobby |
|---|---|---|---|---|
| **alice** | 68 | 1985 | NaN | Biking |
| **bob** | 83 | 1984 | 3.0 | Dancing |
| **charles** | 112 | 1992 | 0.0 | NaN |

```
#DataFrame - Accessing a column
people["birthyear"]
```

```
alice      1985
bob        1984
charles    1992
Name: birthyear, dtype: int64
```

```
#DataFrame - Access the multiple columns
people[["birthyear","hobby"]]
```

| | birthyear | hobby |
|---|---|---|
| **alice** | 1985 | Biking |
| **bob** | 1984 | Dancing |
| **charles** | 1992 | NaN |

```
#Creating DataFrame - Include columns and/or rows and guarantee order
d2 = pd.DataFrame(people_dict,columns=["birthyear", "weight", "height"],index=["bob", "alice", "eugene"])
print(d2)
```

```
         birthyear  weight height
bob         1984.0    83.0    NaN
alice       1985.0    68.0    NaN
eugene         NaN     NaN    NaN
```

```
#DataFrame - Accessing rows
#Using loc
people.loc["charles"]
#Using iloc
people.iloc[2]
```

```
weight        112
birthyear    1992
children      0.0
hobby         NaN
Name: charles, dtype: object
```

```
# DataFrame - Get a slice of rows
people.iloc[1:3]
```

| | weight | birthyear | children | hobby |
|---|---|---|---|---|
| **bob** | 83 | 1984 | 3.0 | Dancing |
| **charles** | 112 | 1992 | 0.0 | NaN |

```
# DataFrame - Pass a boolean array
people[np.array([True, False, True])]
```

| | weight | birthyear | children | hobby |
|---|---|---|---|---|
| **alice** | 68 | 1985 | NaN | Biking |
| **charles** | 112 | 1992 | 0.0 | NaN |

```
people["birthyear"] < 1990
```

```
alice       True
bob         True
charles    False
Name: birthyear, dtype: bool
```

```
#DataFrame - Pass boolean expression
people[people["birthyear"] < 1990]
```

| | weight | birthyear | children | hobby |
|---|---|---|---|---|
| **alice** | 68 | 1985 | NaN | Biking |
| **bob** | 83 | 1984 | 3.0 | Dancing |

```
people[people["birthyear"] < 1990]["weight"]
```

```
alice    68
bob      83
Name: weight, dtype: int64
```

```
#DataFrame- Adding and removing columns
# Adds a new column "age"
people["age"] = 2016 - people["birthyear"]
# Adds another column "over 30"
people["over 30"] = people["age"] > 30
# Removes "birthyear" and "children" columns
birthyears = people.pop("birthyear")
del people["children"]
people
```

| | weight | hobby | age | over 30 |
|---|---|---|---|---|
| **alice** | 68 | Biking | 31 | True |
| **bob** | 83 | Dancing | 32 | True |
| **charles** | 112 | NaN | 24 | False |

```
#DataFrame - A new column must have the same number of rows
# alice is missing, eugene is ignored
people["pets"] = pd.Series({
"bob": 0,
"charles": 5,
"eugene":1
})
people
```

| | weight | hobby | age | over 30 | pets |
|---|---|---|---|---|---|
| **alice** | 68 | Biking | 31 | True | NaN |
| **bob** | 83 | Dancing | 32 | True | 0.0 |
| **charles** | 112 | NaN | 24 | False | 5.0 |

```
#DataFrame - Add a new column using insert method after an existing column
people.insert(1, "height", [172, 181, 185])
people
```

| | weight | height | hobby | age | over 30 | pets |
|---|---|---|---|---|---|---|
| **alice** | 68 | 172 | Biking | 31 | True | NaN |
| **bob** | 83 | 181 | Dancing | 32 | True | 0.0 |
| **charles** | 112 | 185 | NaN | 24 | False | 5.0 |

```
#DataFrame - Add new columns using assign method
(people.assign(body_mass_index = lambda df:df["weight"]/ (df["height"] / 100) ** 2).assign(overweight = lam
```

| | weight | height | hobby | age | over 30 | pets | body_mass_index | overweight |
|---|---|---|---|---|---|---|---|---|
| **alice** | 68 | 172 | Biking | 31 | True | NaN | 22.985398 | False |
| **bob** | 83 | 181 | Dancing | 32 | True | 0.0 | 25.335002 | True |
| **charles** | 112 | 185 | NaN | 24 | False | 5.0 | 32.724617 | True |

```
#DataFrame - Sorting a DataFrame
people.sort_index(ascending=False)
```

| | weight | height | hobby | age | over 30 | pets |
|---|---|---|---|---|---|---|
| **charles** | 112 | 185 | NaN | 24 | False | 5.0 |
| **bob** | 83 | 181 | Dancing | 32 | True | 0.0 |
| **alice** | 68 | 172 | Biking | 31 | True | NaN |

```
#DataFrame - Sorting a DataFrame - inplace argument
people.sort_index(inplace=True)
people
```

| | weight | height | hobby | age | over 30 | pets |
|---|---|---|---|---|---|---|
| **alice** | 68 | 172 | Biking | 31 | True | NaN |
| **bob** | 83 | 181 | Dancing | 32 | True | 0.0 |
| **charles** | 112 | 185 | NaN | 24 | False | 5.0 |

```
#DataFrame - Sorting a DataFrame - Sort By Value
people.sort_values(by="age", inplace=True)
people
```

| | weight | height | hobby | age | over 30 | pets |
|---|---|---|---|---|---|---|
| **charles** | 112 | 185 | NaN | 24 | False | 5.0 |
| **alice** | 68 | 172 | Biking | 31 | True | NaN |
| **bob** | 83 | 181 | Dancing | 32 | True | 0.0 |

```
df=pd.DataFrame({"A":[10,11,12,13],"B":[14,11,12,13],"C":[10,11,12,14]},index=["P","Q","R","S"])
df
```

| | A | B | C |
|---|---|---|---|
| **P** | 10 | 14 | 10 |
| **Q** | 11 | 11 | 11 |
| **R** | 12 | 12 | 12 |
| **S** | 13 | 13 | 14 |

```
df["A"].nunique()
```

```
4
```

```
df["A"].isnull()
```

```
P    False
Q    False
R    False
S    False
Name: A, dtype: bool
```

```
df["A"].value_counts()
```

```
10    1
11    1
12    1
13    1
Name: A, dtype: int64
```

```
df["A"].sum()
```

```
46
```

```
df["A"].apply(lambda x:x*2)
```

```
P    20
Q    22
R    24
S    26
Name: A, dtype: int64
```

```
def times2(x):
    return x*2
df["A"].apply(times2)
```

```
P    20
Q    22
R    24
S    26
Name: A, dtype: int64
```

```
my_df = pd.DataFrame(
[
["Biking", 68.5, 1985, np.nan],
["Dancing", 83.1, 1984, 3]
],
columns=["hobby","weight","birthyear","children"],
index=["alice", "bob"]
)
my_df
```

| | hobby | weight | birthyear | children |
|---|---|---|---|---|
| **alice** | Biking | 68.5 | 1985 | NaN |
| **bob** | Dancing | 83.1 | 1984 | 3.0 |

```
#DataFrames - Saving
#Save to CSV
my_df.to_csv("my_df.csv")
#Save to HTML
my_df.to_html("my_df.html")
#Save to JSON
my_df.to_json("my_df.json")
```

```
for filename in ("my_df.csv", "my_df.html",
"my_df.json"):
    print("#", filename)
    with open(filename, "rt") as f:
        print(f.read())
        print()
```

```
# my_df.csv
,hobby,weight,birthyear,children
alice,Biking,68.5,1985,
bob,Dancing,83.1,1984,3.0
```

```
# my_df.html
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>hobby</th>
      <th>weight</th>
      <th>birthyear</th>
      <th>children</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>alice</th>
      <td>Biking</td>
      <td>68.5</td>
```

```
      <td>1985</td>
      <td>NaN</td>
    </tr>
    <tr>
      <th>bob</th>
      <td>Dancing</td>
      <td>83.1</td>
      <td>1984</td>
      <td>3.0</td>
    </tr>
  </tbody>
</table>

# my_df.json
{"hobby":{"alice":"Biking","bob":"Dancing"},"weight":{"alice":68.5,"bob":83.1},"birthyear":{"alice":19
```