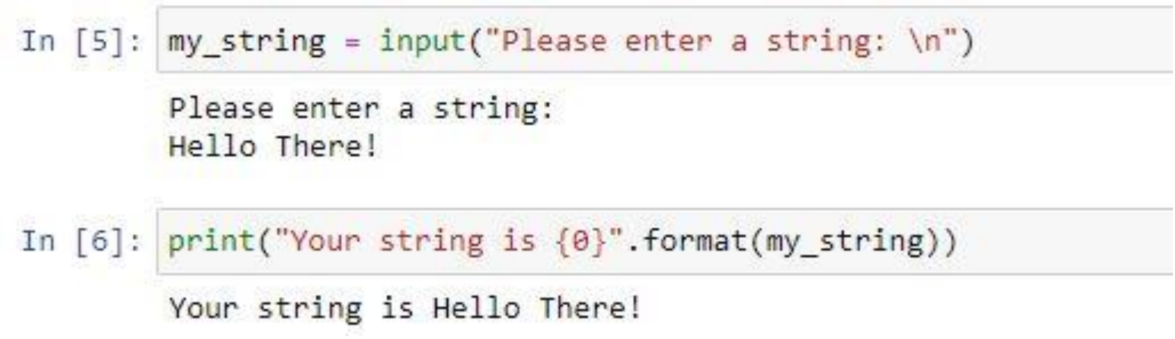
File Handling

User input from the keyboard

The input from the user can be read as a string and can be assigned to a variable.

The syntax **input()** function reads value entered by the user.

Ex:
my_string = **input**(“Please enter a string:\n”)
print(“Your string is **{0}**”.**format**(my_string)

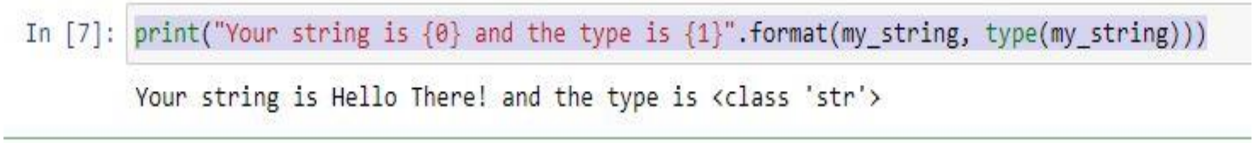


```
In [5]: my_string = input("Please enter a string: \n")

        Please enter a string:
        Hello There!

In [6]: print("Your string is {0}".format(my_string))

        Your string is Hello There!
```

Type of user input entered value

The value entered is always converted to a string and is assigned to a variable.
We can use **type()** function to check the type of the variable.

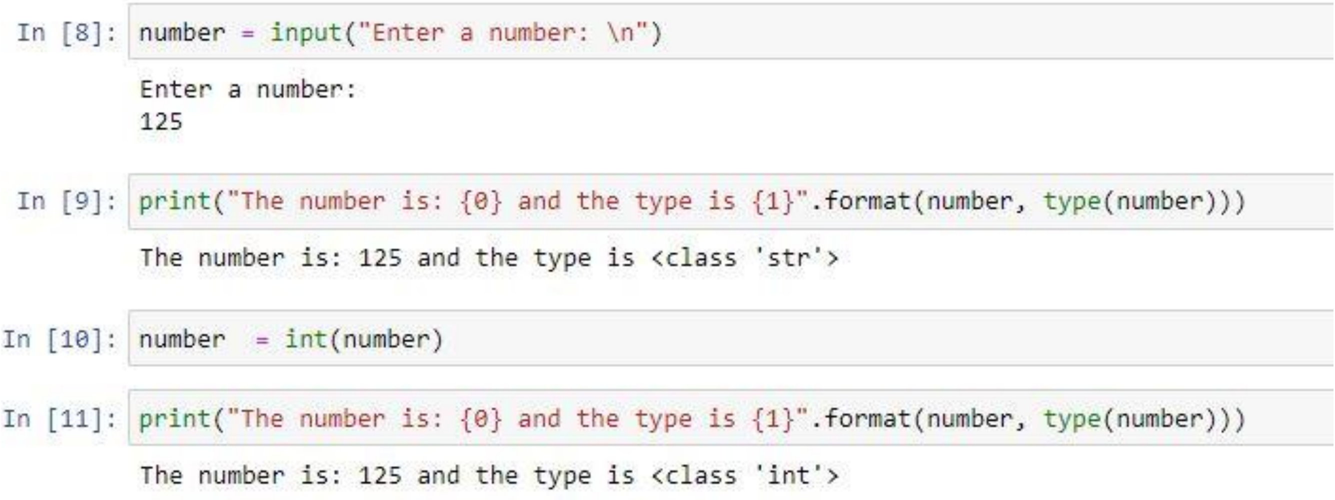
Ex:
print("Your string is **{0}** and the type is **{1}**".**format(my_string, type(my_string)))**)



```
In [7]: print("Your string is {0} and the type is {1}".format(my_string, type(my_string)))

        Your string is Hello There! and the type is <class 'str'>
```

Getting an integer as the user input

The only way to get an integer is by using built-in functions to convert the entered string to the integer or any other type.



```
In [8]:  number = input("Enter a number: \n")

         Enter a number:
         125

In [9]:  print("The number is: {0} and the type is {1}".format(number, type(number)))

         The number is: 125 and the type is <class 'str'>

In [10]: number  = int(number)

In [11]: print("The number is: {0} and the type is {1}".format(number, type(number)))

         The number is: 125 and the type is <class 'int'>
```

Printing output

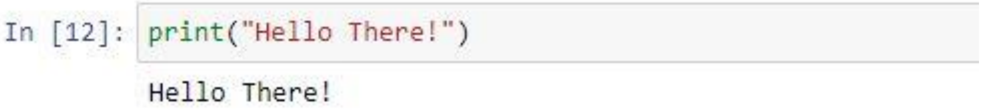
The syntax **print()** function will print the given input to the screen or corresponding file stream.

print(object, sep = ‘ ‘ , end = ‘\n’ , file = sys.stdout, flush = false)

print(required, optional, optional, optional, optional)

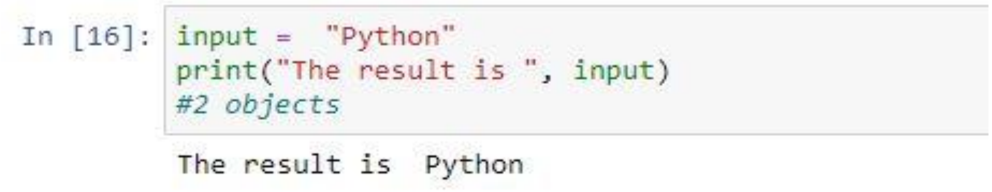
object: objects to be printed. Can be one or more objects.
sep: objects are separated by sep. Default value is ‘ ’.
end: printed at last. Default is ‘\n’.
file: must be object with write string. If not, sys.stdout is used.
flush: if true, the stream is forcibly flushed. By default, it is false.

Basic print statement.



```
In [12]: print("Hello There!")

         Hello There!
```

Printing multiple objects.

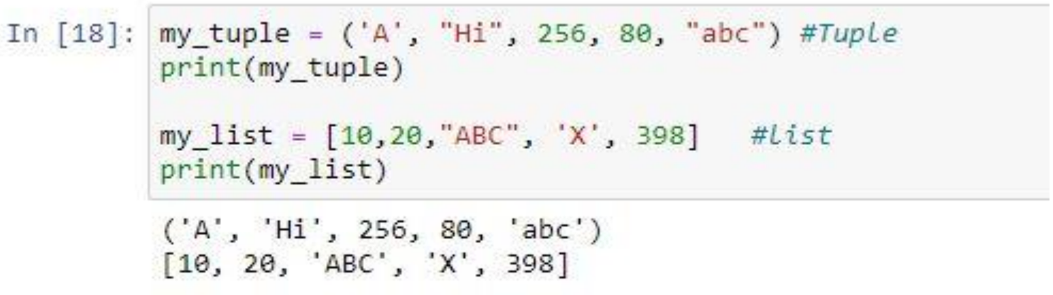


In [16]:

```
input =  "Python"
print("The result is ", input)
#2 objects
```

```
The result is  Python
```

Printing tuples and lists.



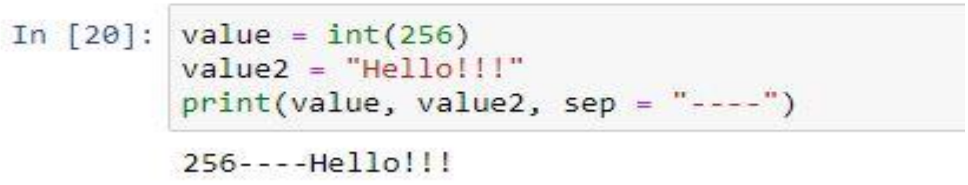
In [18]:

```
my_tuple = ('A', "Hi", 256, 80, "abc") #Tuple
print(my_tuple)

my_list = [10,20,"ABC", 'X', 398]   #list
print(my_list)
```

```
('A', 'Hi', 256, 80, 'abc')
[10, 20, 'ABC', 'X', 398]
```

Printing with “sep” keyword.

By default, values are separated by space. But user can modify it by replacing the space with any other value.



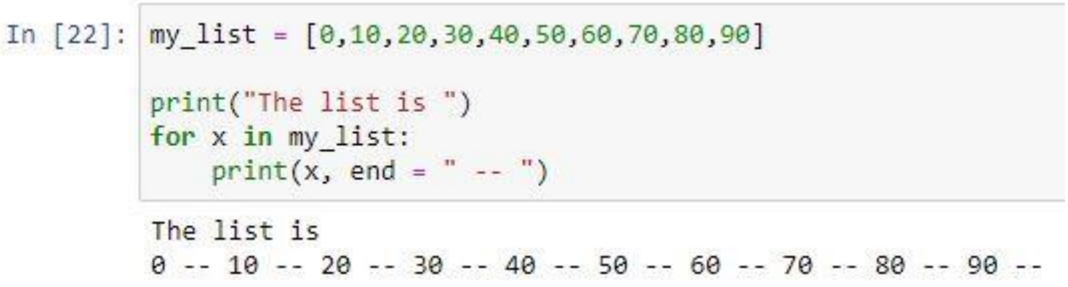
In [20]:

```
value = int(256)
value2 = "Hello!!!"
print(value, value2, sep = "----")
```

```
256----Hello!!!
```

Printing with “end” keyword.

Default value is ‘\n’. User can modify the value with any symbol.



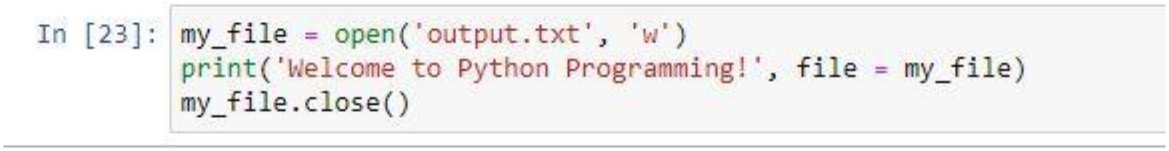
In [22]:

```
my_list = [0,10,20,30,40,50,60,70,80,90]

print("The list is ")
for x in my_list:
    print(x, end = " -- ")
```

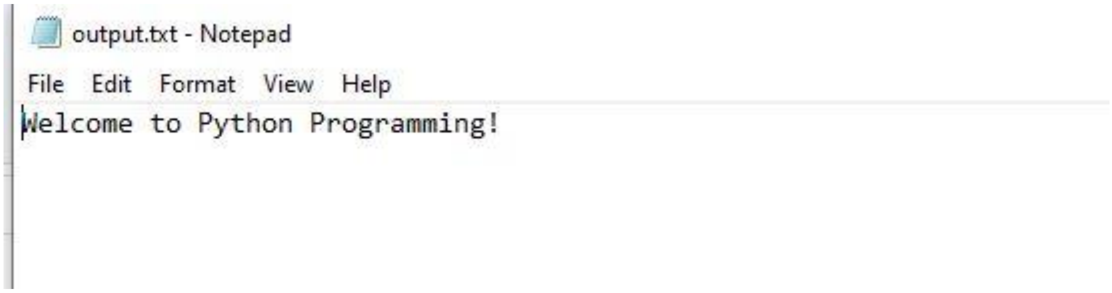
```
The list is 
0 -- 10 -- 20 -- 30 -- 40 -- 50 -- 60 -- 70 -- 80 -- 90 -- 
```

Print using “file” keyword.

Enable user to write to a file. If the file does not exist**,** it creates a new file and writes to the file.



```
In [23]: my_file = open('output.txt', 'w')
         print('Welcome to Python Programming!', file = my_file)
         my_file.close()
```



```
Welcome to Python Programming!
```

File Reading

There are 3 ways to read from a file.

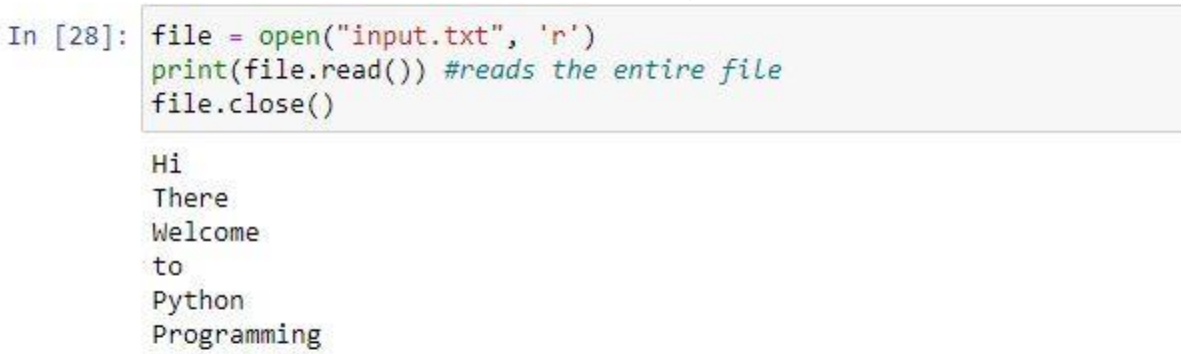
1. **Read()**- Reads the whole content of the file when no arguments are passed. If ‘n’ is passed returns n bytes from the file as a string.
2. **Readline()**: Reads a single line and returns as a string. When n is specified, reads atmost n without exceeding a line.
3. **Readlines()**: Returns a list of string.

Various modes that a file can be opened.

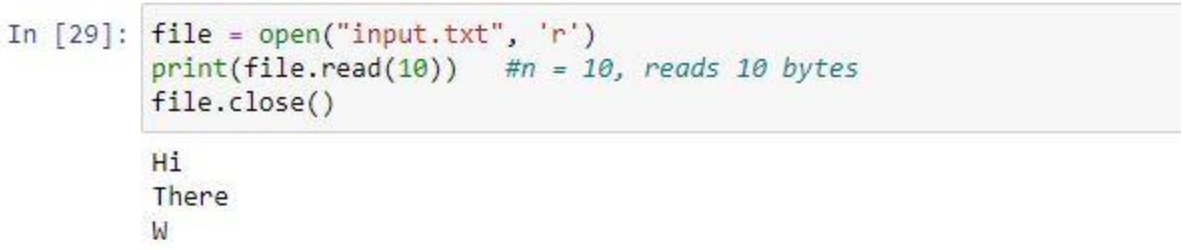
r- read only
w- only write
a- append only
r+ - read as well as write
w+ - write as well as read
a+- append as well as read

Read()

A pre-defined function. Returns the read data as a string.



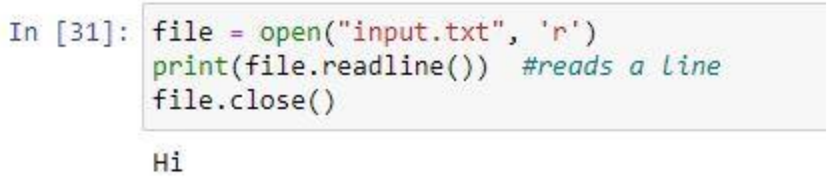
```
In [28]: file = open("input.txt", 'r')
         print(file.read()) #reads the entire file
         file.close()

         Hi
         There
         Welcome
         to
         Python
         Programming
```



```
In [29]: file = open("input.txt", 'r')
         print(file.read(10))   #n = 10, reads 10 bytes
         file.close()

         Hi
         There
         W
```

Readline()

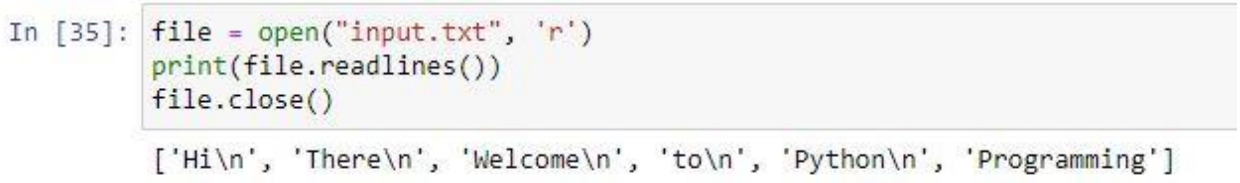
A predefined function.



```
In [31]: file = open("input.txt", 'r')
         print(file.readline())  #reads a line
         file.close()

         Hi
```

Readlines()

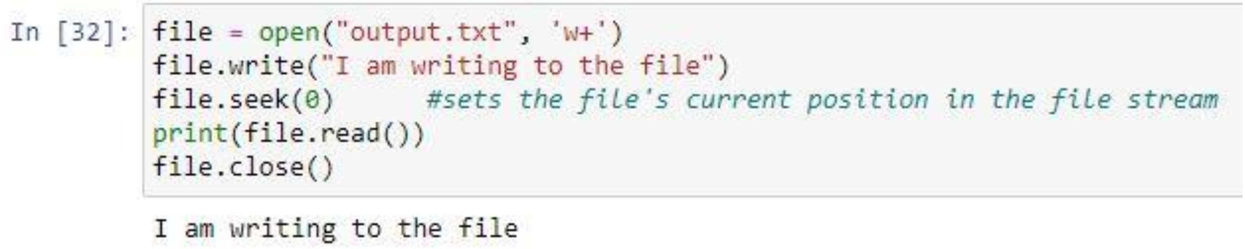
Reads all the lines in the file and returns a list containing the string forms.



```
In [35]: file = open("input.txt", 'r')
         print(file.readlines())
         file.close()

         ['Hi\n', 'There\n', 'Welcome\n', 'to\n', 'Python\n', 'Programming']
```

File Writing

Use **write()** function, we can write a string to a file.

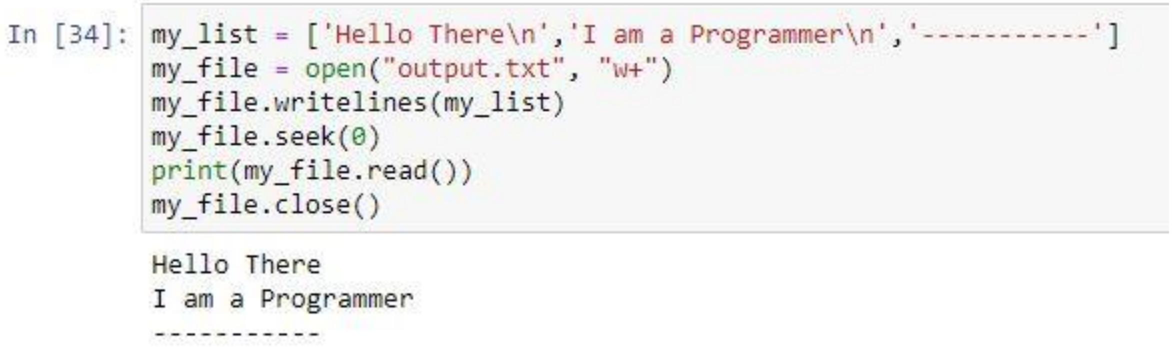


```
In [32]: file = open("output.txt", 'w+')
         file.write("I am writing to the file")
         file.seek(0)      #sets the file's current position in the file stream
         print(file.read())
         file.close()

         I am writing to the file
```

Using writelines()

A pre-defined function that is used to write multiple lines.

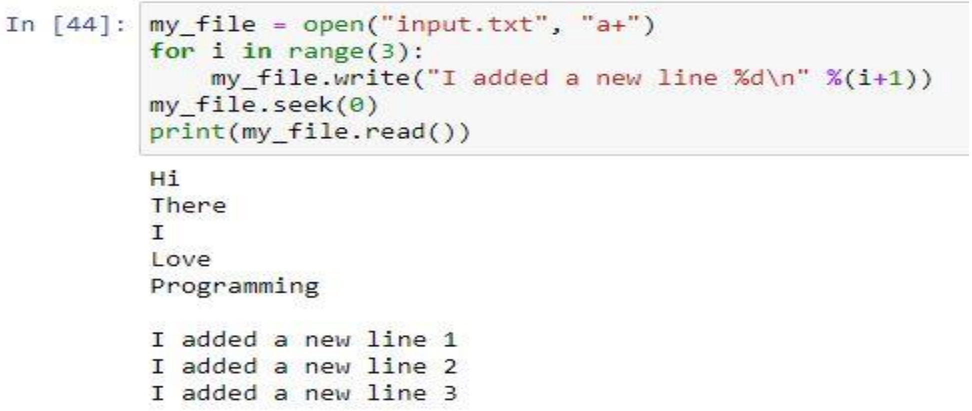
Use a list of string elements to pass as the argument.



```
In [34]: my_list = ['Hello There\n','I am a Programmer\n','-----------']
         my_file = open("output.txt", "w+")
         my_file.writelines(my_list)
         my_file.seek(0)
         print(my_file.read())
         my_file.close()

         Hello There
         I am a Programmer
         -----------
```

Appending to an existing file

Append/Add a new text to the already existing file.



```
In [44]: my_file = open("input.txt", "a+")
         for i in range(3):
             my_file.write("I added a new line %d\n" %(i+1))
         my_file.seek(0)
         print(my_file.read())

         Hi
         There
         I
         Love
         Programming

         I added a new line 1
         I added a new line 2
         I added a new line 3
```

Introduction to Operating System (OS)

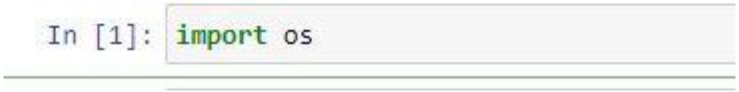
Python OS module allows user to gain access to the OS information.

Contains pre-defined functions which serves as a way to interact with the OS.

1. Importing OS Module.

First thing to do is to import the OS module before using the functionalities.

Syntax: - **import** os

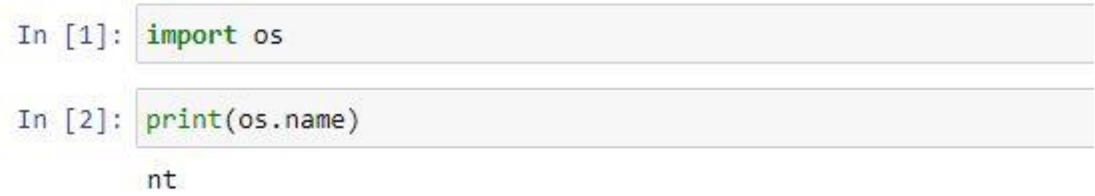


```
In [1]: import os
```

2. OS Name

The name of the OS module that is imported. The name differs by the OS the user is using.

Syntax: os.name



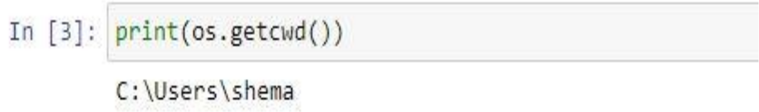
```
In [1]: import os

In [2]: print(os.name)

        nt
```

Windows OS gives **nt** while Mac OS gives **posix**

nt or **Windows NT** is an OS produced by Microsoft. **Posix** or **Portable Operating System Interface** is an OS built for UNIX-like systems.

3. Current Working Directory (cwd)

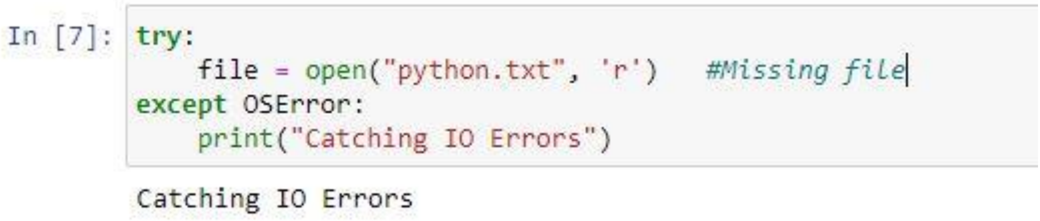
Returns the directory that is used to execute and run the code in python.
Syntax: os.getcwd()



```
In [3]: print(os.getcwd())

        C:\Users\shema
```

4. os.error

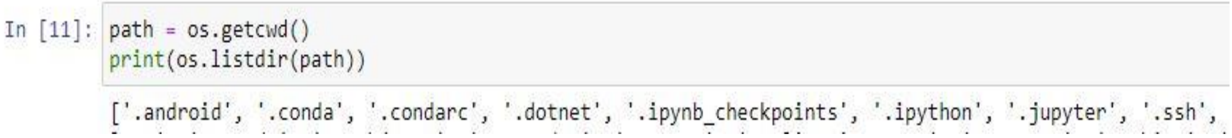
The base class for IO related errors.



```
In [7]: try:
            file = open("python.txt", 'r')    #Missing file
        except OSError:
            print("Catching IO Errors")

        Catching IO Errors
```

5. List of files and Directories.

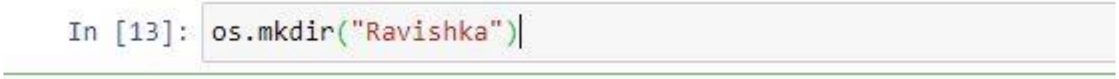
Returns the list of files and directories presented in your Current Working Directory that is passed as parameter.
Syntax: os.listdir(path)



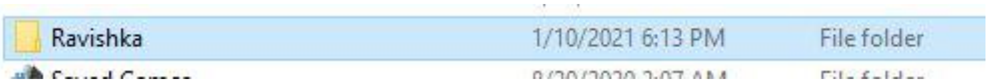
```
In [11]: path = os.getcwd()
         print(os.listdir(path))

         ['.android', '.conda', '.condarc', '.dotnet', '.ipynb_checkpoints', '.ipython', '.jupyter', '.ssh',
```

6. Create a new directory.

Create a new directory.
syntax: os.mkdir(name)



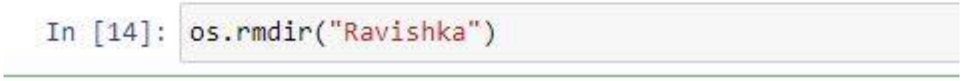
```
In [13]: os.mkdir("Ravishka")
```



```
Ravishka                         1/10/2021 6:13 PM          File folder
```

7. Remove a Directory.

Syntax: os.rmdir(name)



```
In [14]: os.rmdir("Ravishka")
```

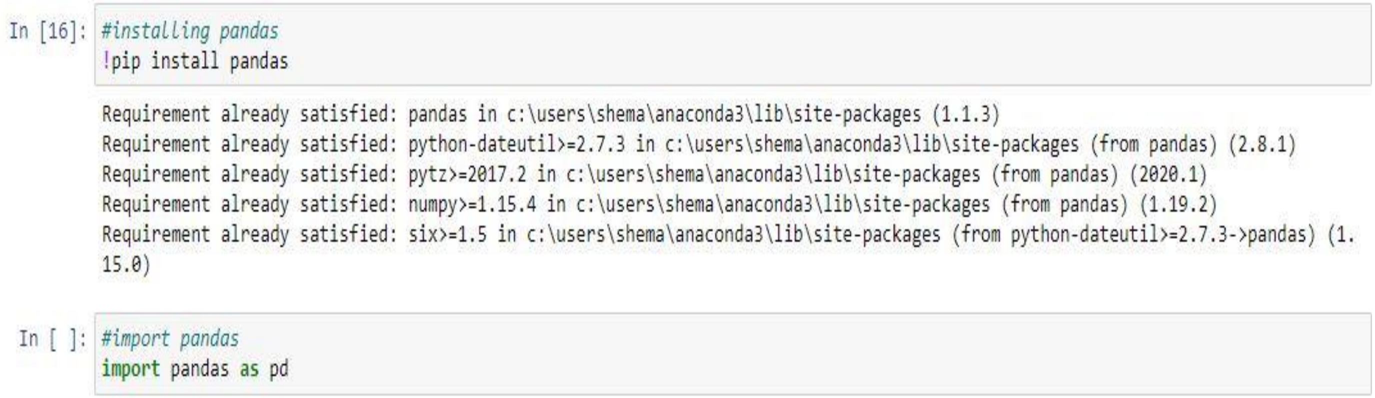
Introduction to Pandas

An open-source library that is built on top of NumPy.
Excels in performance and productivity.
Contains built in visualization features.

Installation:

Install using the command line or terminal. Use either **conda install pandas** for anaconda distribution of python or use **pip install pandas** if you have any other installation method for python.

Install and import pandas.



```
In [16]: #installing pandas
         !pip install pandas

         Requirement already satisfied: pandas in c:\users\shema\anaconda3\lib\site-packages (1.1.3)
         Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\shema\anaconda3\lib\site-packages (from pandas) (2.8.1)
         Requirement already satisfied: pytz>=2017.2 in c:\users\shema\anaconda3\lib\site-packages (from pandas) (2020.1)
         Requirement already satisfied: numpy>=1.15.4 in c:\users\shema\anaconda3\lib\site-packages (from pandas) (1.19.2)
         Requirement already satisfied: six>=1.5 in c:\users\shema\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.
         15.0)

In [ ]:  #import pandas
         import pandas as pd
```

Core components.

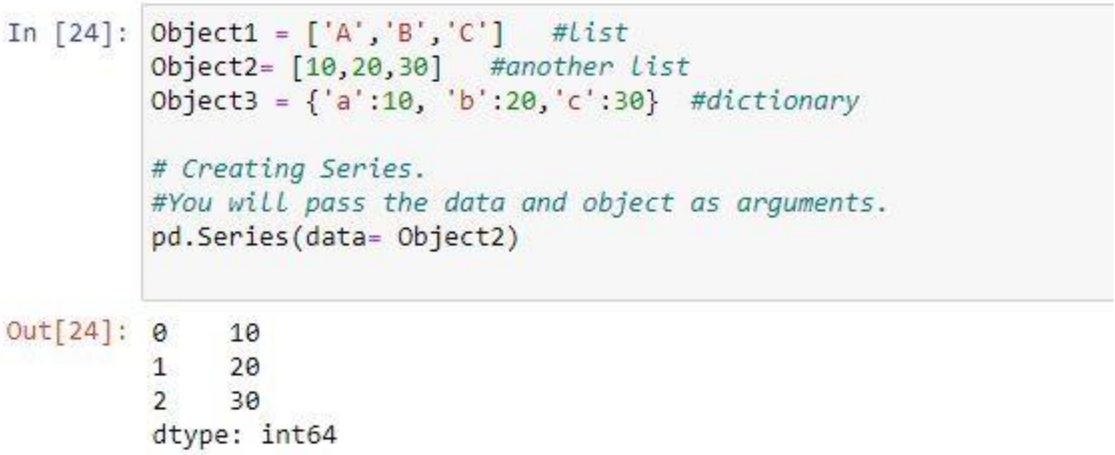
There are two primary components. **Series** and **DataFrame.**

Series is essentially a column.
DataFrame is a multi-dimensional tables or collections of Series.

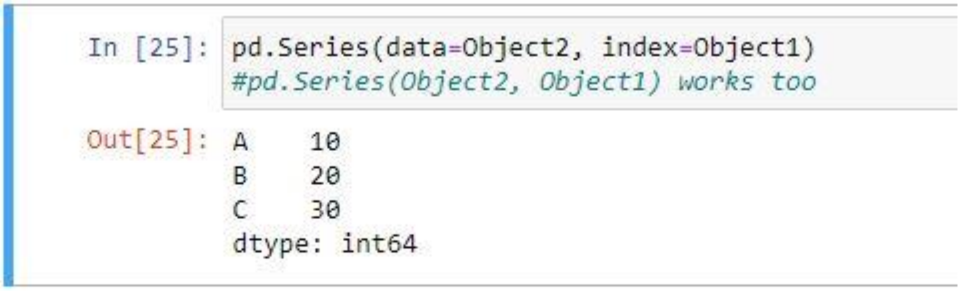
Series

Similar to NumPy arrays but contains indexed labels.

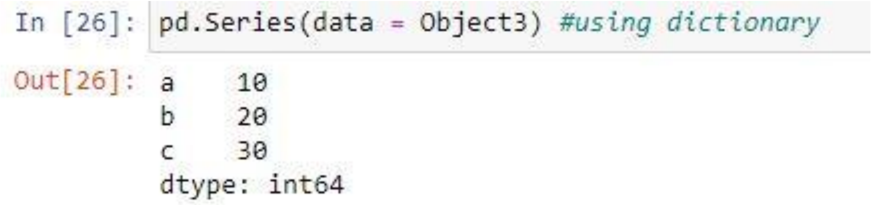
Create Series using different object types.



```
In [24]: Object1 = ['A','B','C']   #list
         Object2= [10,20,30]   #another list
         Object3 = {'a':10, 'b':20,'c':30}  #dictionary

         # Creating Series.
         #You will pass the data and object as arguments.
         pd.Series(data= Object2)

Out[24]: 0    10
         1    20
         2    30
         dtype: int64
```



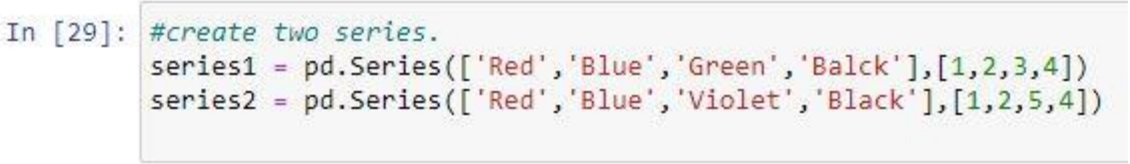
```
In [25]: pd.Series(data=Object2, index=Object1)
         #pd.Series(Object2, Object1) works too

Out[25]: A    10
         B    20
         C    30
         dtype: int64
```



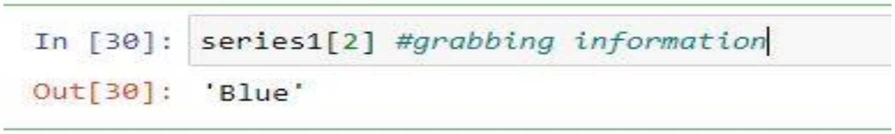
```
In [26]: pd.Series(data = Object3) #using dictionary

Out[26]: a    10
         b    20
         c    30
         dtype: int64
```

Using Index in a Series

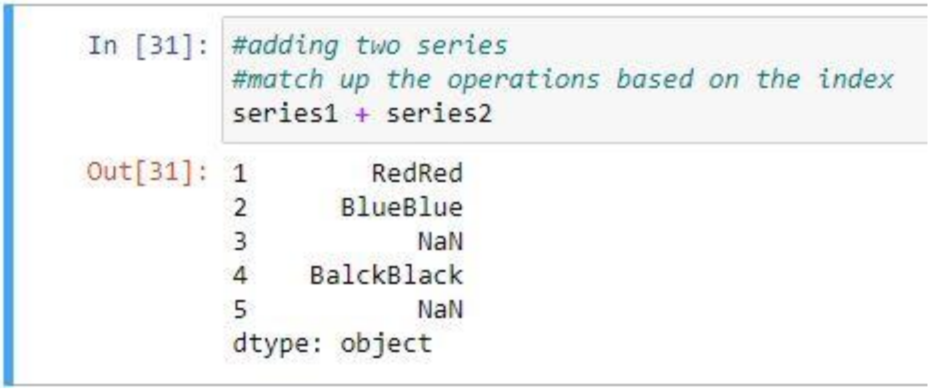
Key of using Series in pandas is to understand its index.
Works like a look up table (hash table or dictionary)



```
In [29]: #create two series.
         series1 = pd.Series(['Red','Blue','Green','Balck'],[1,2,3,4])
         series2 = pd.Series(['Red','Blue','Violet','Black'],[1,2,5,4])
```



```
In [30]: series1[2] #grabbing information

Out[30]: 'Blue'
```

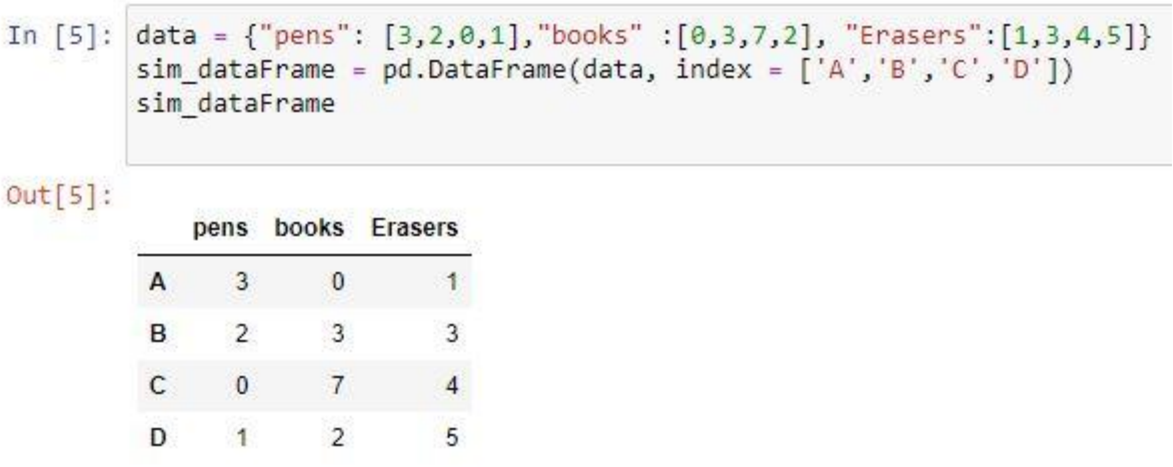


```
In [31]: #adding two series
         #match up the operations based on the index
         series1 + series2

Out[31]: 1          RedRed
         2        BlueBlue
         3             NaN
         4      BalckBlack
         5             NaN
         dtype: object
```

DataFrames

Creating DataFrames from the scratch.

Useful when testing new methods and functions .

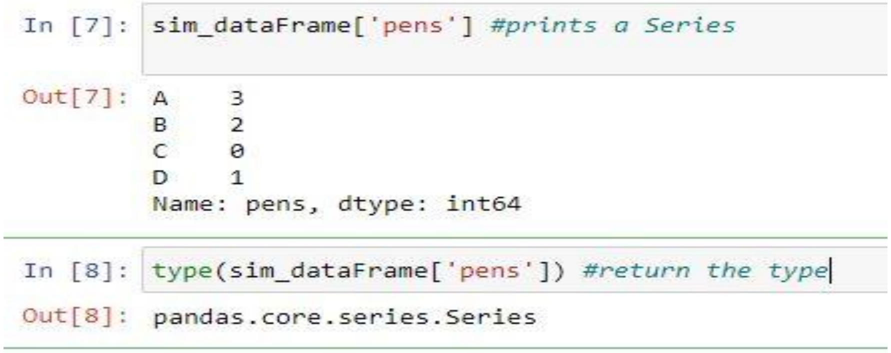


```
In [5]: data = {"pens": [3,2,0,1],"books" :[0,3,7,2], "Erasers":[1,3,4,5]}
        sim_dataFrame = pd.DataFrame(data, index = ['A','B','C','D'])
        sim_dataFrame

Out[5]:
```

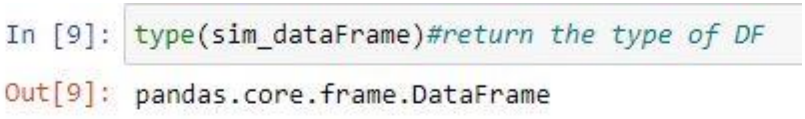
	pens	books	Erasers
A	3	0	1
B	2	3	3
C	0	7	4
D	1	2	5

Each (key, value) item correspondence to a column in a DataFrame.

Using indexing to get a Series object in a DataFrame.

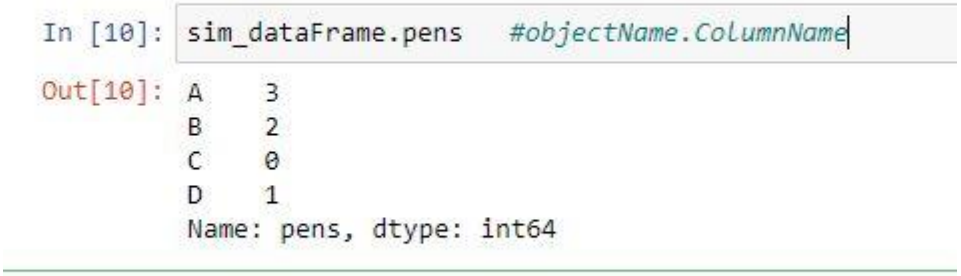


```
In [7]: sim_dataFrame['pens'] #prints a Series

Out[7]: A    3
        B    2
        C    0
        D    1
        Name: pens, dtype: int64

In [8]: type(sim_dataFrame['pens']) #return the type

Out[8]: pandas.core.series.Series
```



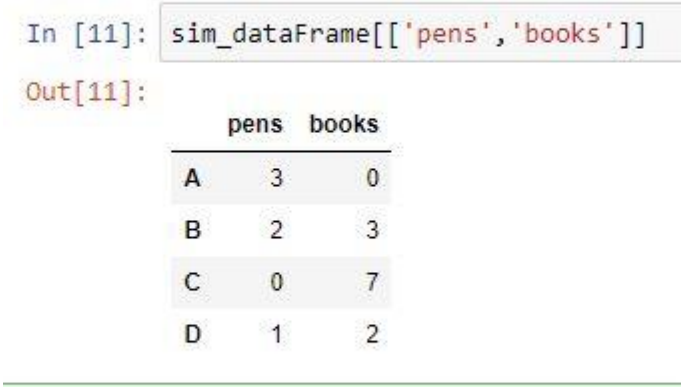
```
In [9]: type(sim_dataFrame)#return the type of DF

Out[9]: pandas.core.frame.DataFrame
```

Alternate way - might confuse the built-in methods with column name



```
In [10]: sim_dataFrame.pens   #objectName.ColumnName

Out[10]: A    3
         B    2
         C    0
         D    1
         Name: pens, dtype: int64
```

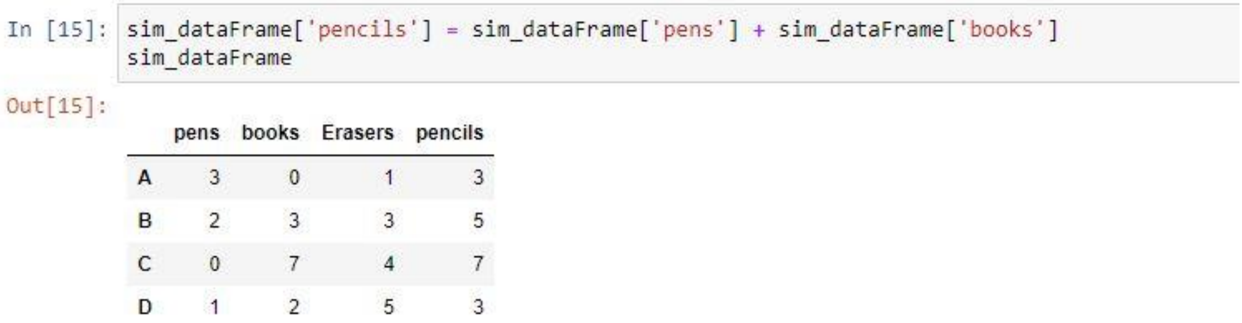
Multiple columns



```
In [11]: sim_dataFrame[['pens','books']]

Out[11]:
```

	pens	books
A	3	0
B	2	3
C	0	7
D	1	2

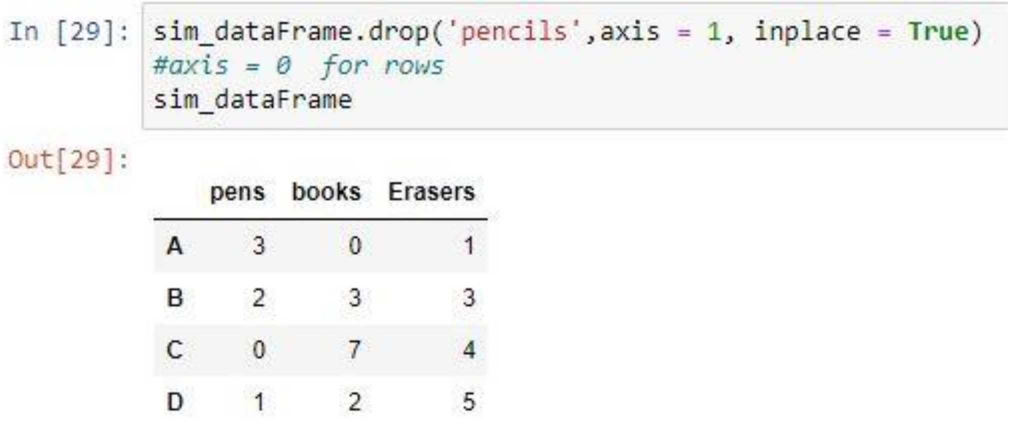
Creating new columns



```
In [15]: sim_dataFrame['pencils'] = sim_dataFrame['pens'] + sim_dataFrame['books']
         sim_dataFrame

Out[15]:
```

	pens	books	Erasers	pencils
A	3	0	1	3
B	2	3	3	5
C	0	7	4	7
D	1	2	5	3

Deleting a column



```
In [29]: sim_dataFrame.drop('pencils',axis = 1, inplace = True)
         #axis = 0  for rows
         sim_dataFrame
```

Out[29]:

	pens	books	Erasers
A	3	0	1
B	2	3	3
C	0	7	4
D	1	2	5

Inplace = True to delete the column permanently.

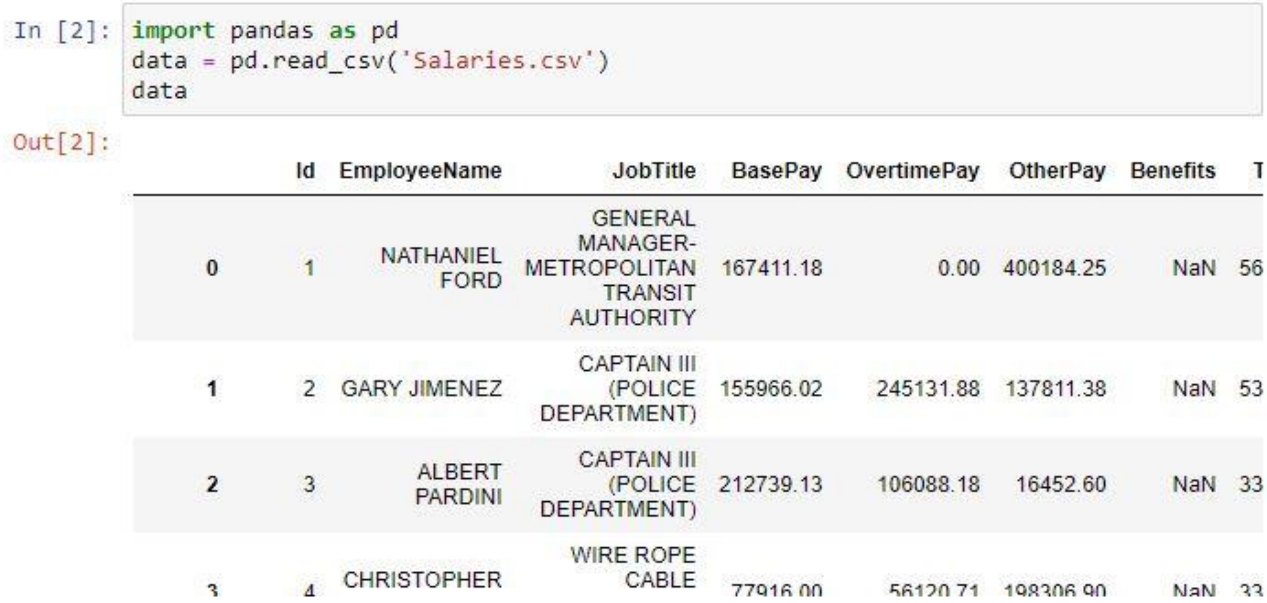
To get the size of the DataFrame use objectName.**shape** that will return (#of rows, #of columns)

Reading Data from CSV files

.data files are used to store data. The data might be stored in a comma separated value format or tab separated value format.

CSV- Comma-Sperated Values files. Allows data to be saved in a tabular format.

Use **read_csv()** function to read csv files.

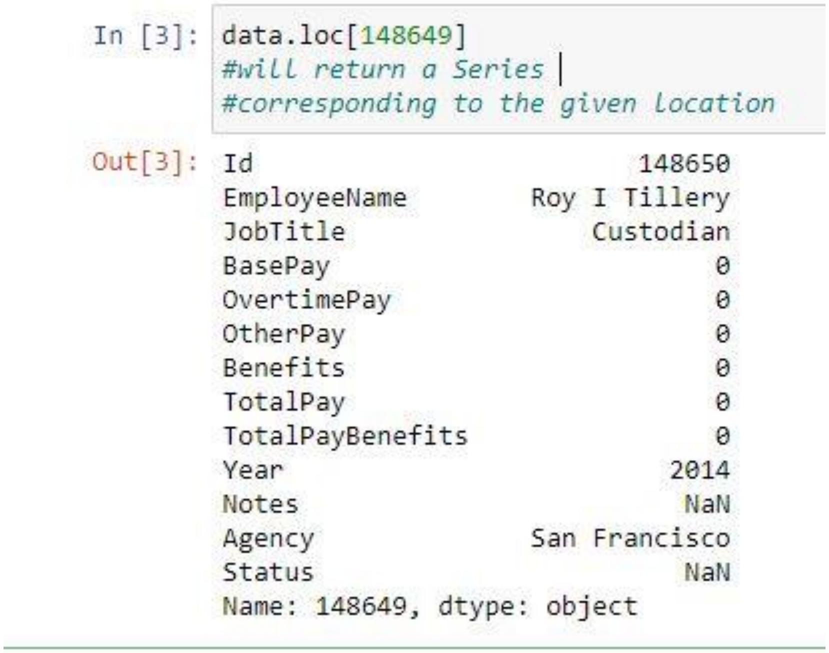


```
In [2]: import pandas as pd
        data = pd.read_csv('Salaries.csv')
        data
```

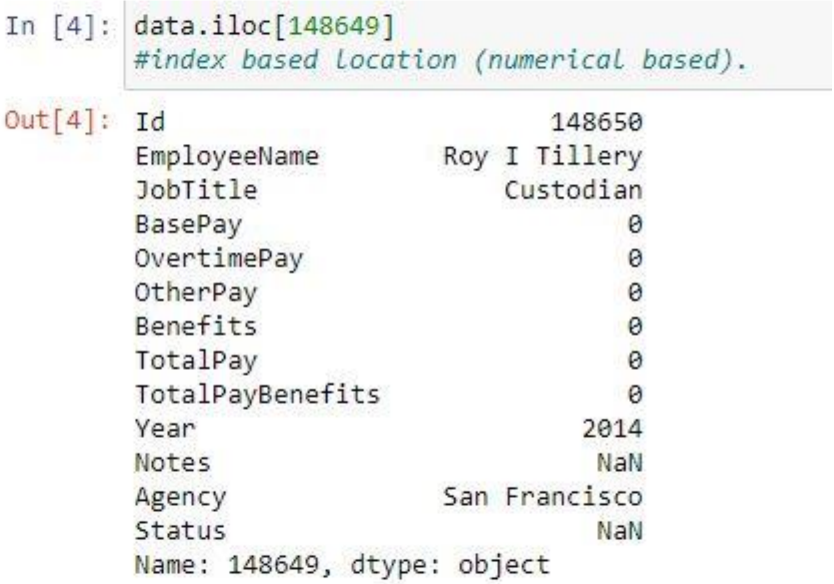
Out[2]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	T
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.00	400184.25	NaN	56
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	53
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.60	NaN	33
3	4	CHRISTOPHER	WIRE ROPE CABLE	77916.00	56120.71	198306.90	NaN	33

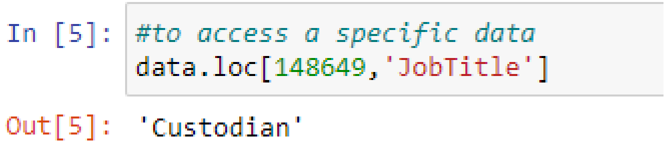
To select rows, we can use two ways to select rows.



```
In [3]: data.loc[148649]
        #will return a Series
        #corresponding to the given location

Out[3]: Id                            148650
        EmployeeName           Roy I Tillery
        JobTitle                   Custodian
        BasePay                            0
        OvertimePay                        0
        OtherPay                           0
        Benefits                           0
        TotalPay                           0
        TotalPayBenefits                   0
        Year                            2014
        Notes                            NaN
        Agency                 San Francisco
        Status                           NaN
        Name: 148649, dtype: object
```



```
In [4]: data.iloc[148649]
        #index based location (numerical based).

Out[4]: Id                            148650
        EmployeeName           Roy I Tillery
        JobTitle                   Custodian
        BasePay                            0
        OvertimePay                        0
        OtherPay                           0
        Benefits                           0
        TotalPay                           0
        TotalPayBenefits                   0
        Year                            2014
        Notes                            NaN
        Agency                 San Francisco
        Status                           NaN
        Name: 148649, dtype: object
```

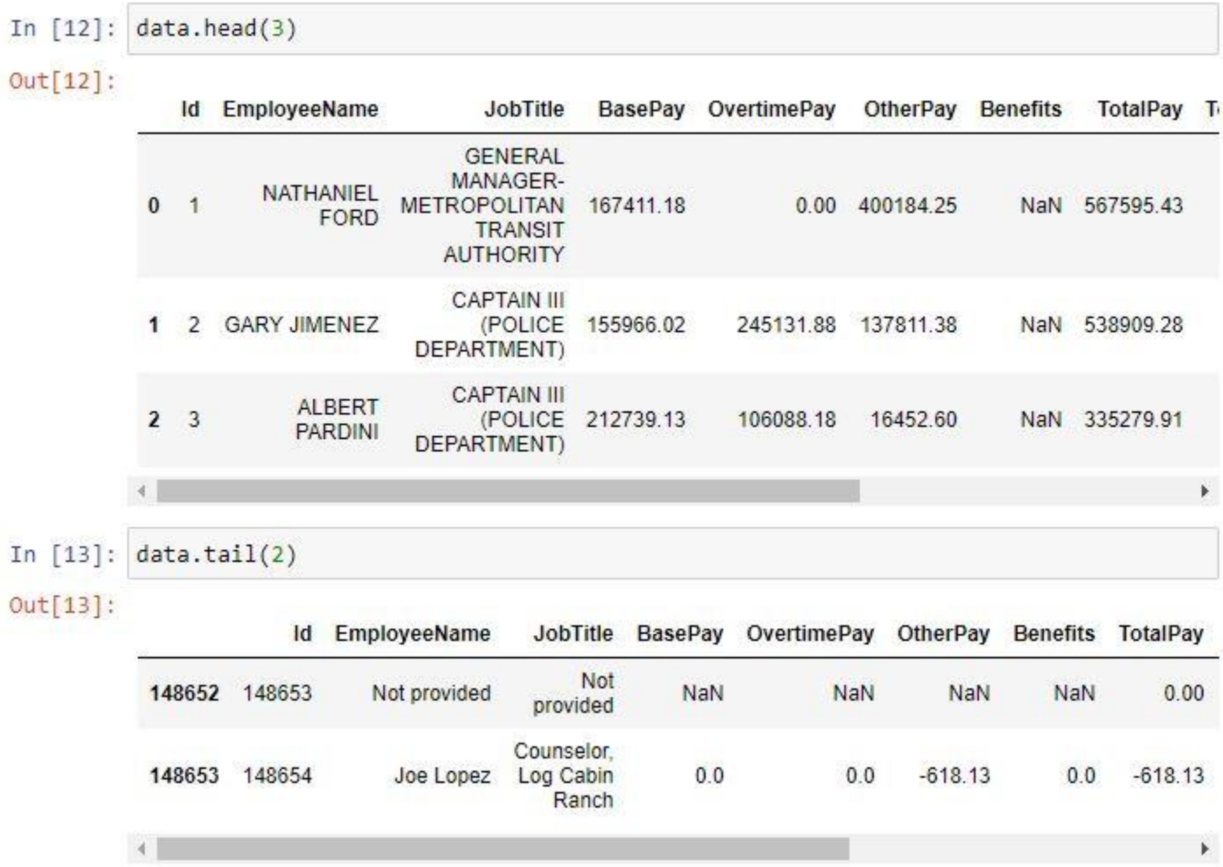


```
In [5]: #to access a specific data
        data.loc[148649,'JobTitle']

Out[5]: 'Custodian'
```

Using .head() and .tail() to view your DataFrame

.head() outputs first five rows by default. But you can use .head(number) to process upto that number of rows to be displayed from the top.

.tail() outputs last five rows by default. But you can use .tail(number) to process upto that number of rows to be displayed from the bottom.



```
In [12]: data.head(3)
```

Out[12]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay	T
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.00	400184.25	NaN	567595.43	
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	538909.28	
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.60	NaN	335279.91	

```
In [13]: data.tail(2)
```

Out[13]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay
148652	148653	Not provided	Not provided	NaN	NaN	NaN	NaN	0.00
148653	148654	Joe Lopez	Counselor, Log Cabin Ranch	0.0	0.0	-618.13	0.0	-618.13

Storing a DataFrame as a CSV file.

Create your DataFrame.

Then use: DataFrameName.**to_csv**('Filename.csv)