Python Beginner

Develop a passion for Learning……

Content

Introduction to Python

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- It was created by Guido van Rossum during 1985- 1990.
- Like Perl, Python source code is also available under the GNU General Public License (GPL).
-

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- Execute Python Syntax
- As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

-
  ```
   print("Hello, World!")
  Hello, World!
  ```
- Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:
- C:\Users\Your Name>python myfile.py
- Python Indentation
- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.
- Example

-
  ```
  if 5 > 2:
    print("Five is greater than two!")
  ```
- Python will give you an error if you skip the indentation:
- Example
- Syntax Error:

-
  ```
  if 5 > 2:
  print("Five is greater than two!")
  ```

- The number of spaces is up to you as a programmer, but it has to be at least one.
- Example

-
  ```
  if 5 > 2:
   print("Five is greater than two!")
  if 5 > 2:
          print("Five is greater than two!")
  ```
- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:
- Example
- Syntax Error:

-
  ```
  if 5 > 2:
   print("Five is greater than two!")
          print("Five is greater than two!")
  ```
- Python Variables
- In Python variables are created the moment you assign a value to it:
- Example
- Variables in Python:

-
  ```
  x = 5
  y = "Hello, World!"
  ```

- Python has no command for declaring a variable.
- You will learn more about variables in the Python Variables chapter.
- Comments
- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment:
- Example
- Comments in Python:
- #This is a comment.
  print("Hello, World!")
- Run example »
-

- Python Operators
- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Java: java is statick typed programming language

```
Eg:public s v m
{
Int a, b
a=2
b=3
Print(a+b)
}
```

Python :is a dynamic typed programming langa

```
Eg: a,b=10,20
z=a+b
print("sum of two no", z)
---------------------------
----------------------------
Type (z)
---------
Int a                                                       a=10
a=10                                                        a=true
```

print a=true ------error in java int to bool conversion not possible in java or other langauge , but in python its dynamical typed

Python Features

- **Easy-to-learn** – python has few keywords, simple structure, and a clearly defined syntax. this allows to pick up the language quickly.
- **Easy-to-read –** python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – python’s source code is fairly easy-to-maintain.
- **A broad standard library** – Python’s bulk of the library is very portable and cross-platform compatible on UNIX, Windows.

- **Interactive Mode –** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platform and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Python interfaces

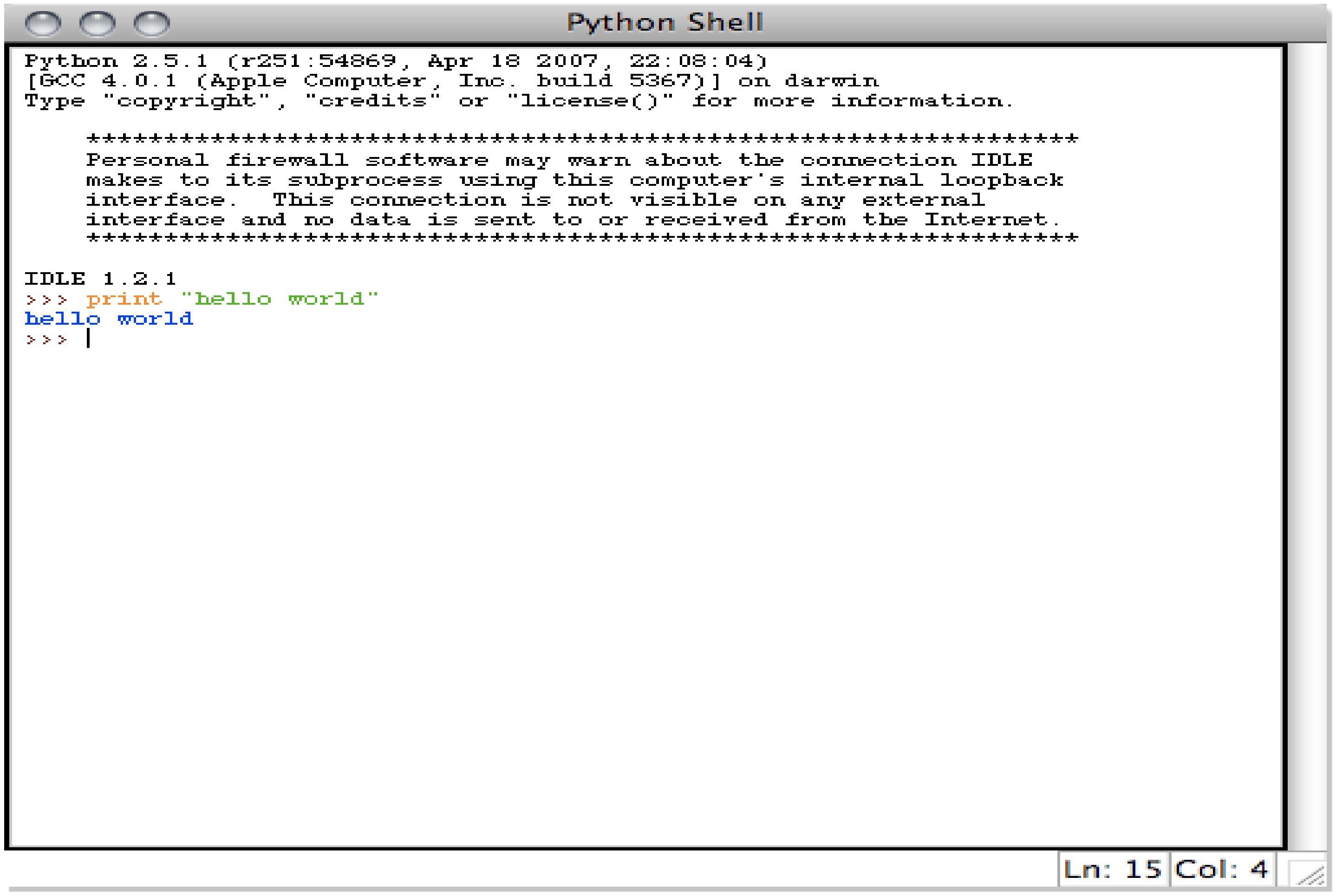
- IDLE – a cross-platform Python development environment
- PythonWin – a Windows only interface to Python
- Python Shell – running 'python' from the Command Line opens this interactive shell
- For the exercises, we'll use IDLE, but you can try them all and pick a favorite

Example python

- print (“Hello, Python!“)
- Hello World

```
print “hello world”
```

- Prints hello world to standard out
- Open IDLE and try it out yourself
- Follow along using IDLE



```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.1
>>> print "hello world"
hello world
>>> 
```

Datatype

1. String.
2. List.
3. Tuple.
4. Set
5. Dictionary.

Numbers -- Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

- For example −
- var1 = 1
- var2 = 10
- You can also delete the reference to a number object by using the **del** statement.
- The syntax of the del statement is −
- **del var1[,var2[,var3[....,varN]]]]**

- Python support four different numerical types:

1. **int(signed integers)**
2. **Long(long integers)**
3. **Float(floating point real values)**
4. **Complex(complex numbers)**

- Examples
- a=5
- Print(a,”is of type”, type(a))

a=2.0

Print(a,”is of type”, type(a))

a=1+2j

Print(a,”is complex number?”, isinstance(1+2j, complex))

- Here are some examples of numbers

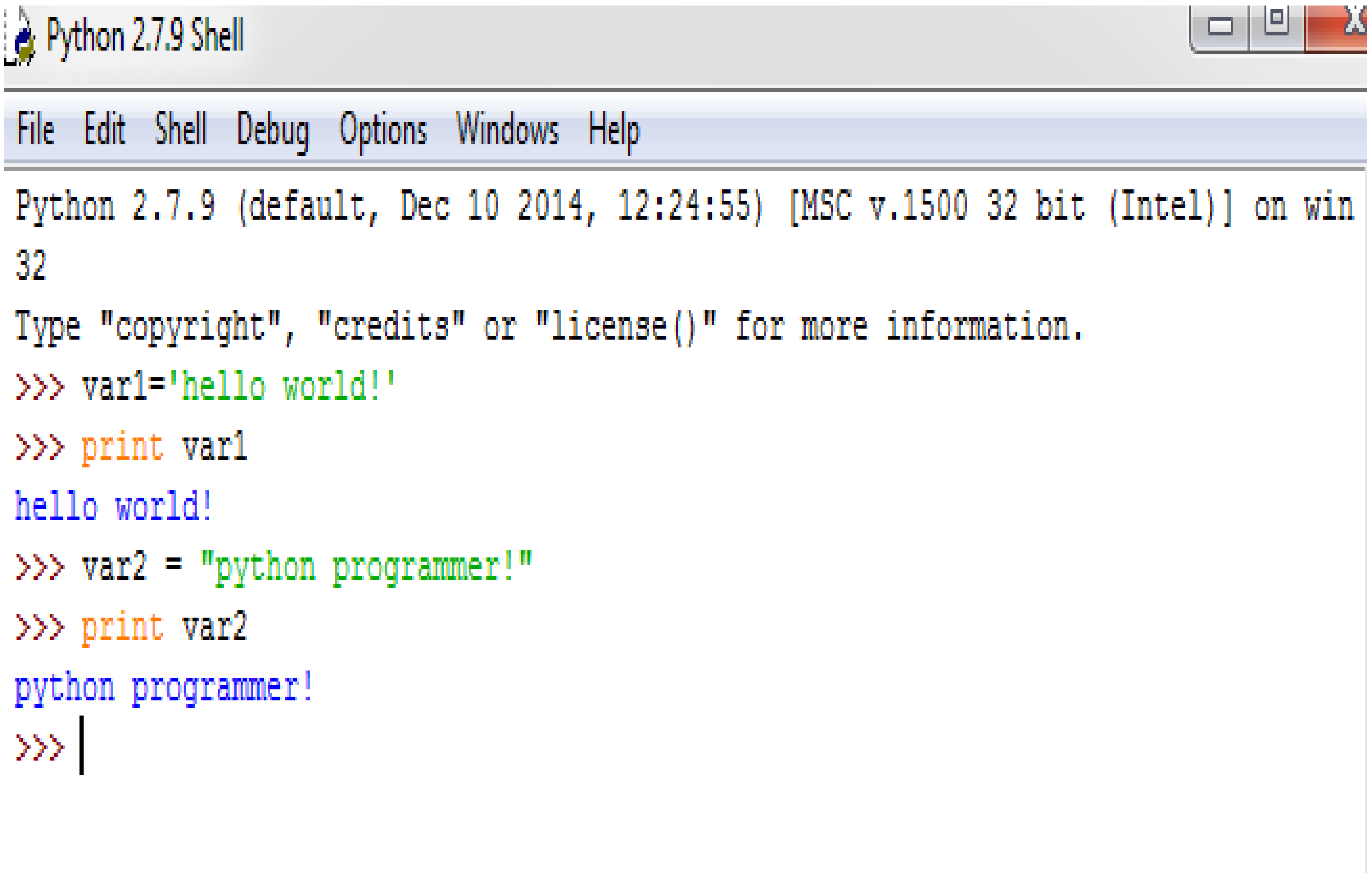
Int	**long**	**float**	**complex**
10	**51924361L**	**0.0**	**3.14j**

Python- String

- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.
- For Examples:

Var1= ‘Hello World!’

Var2 = “python programmer ”



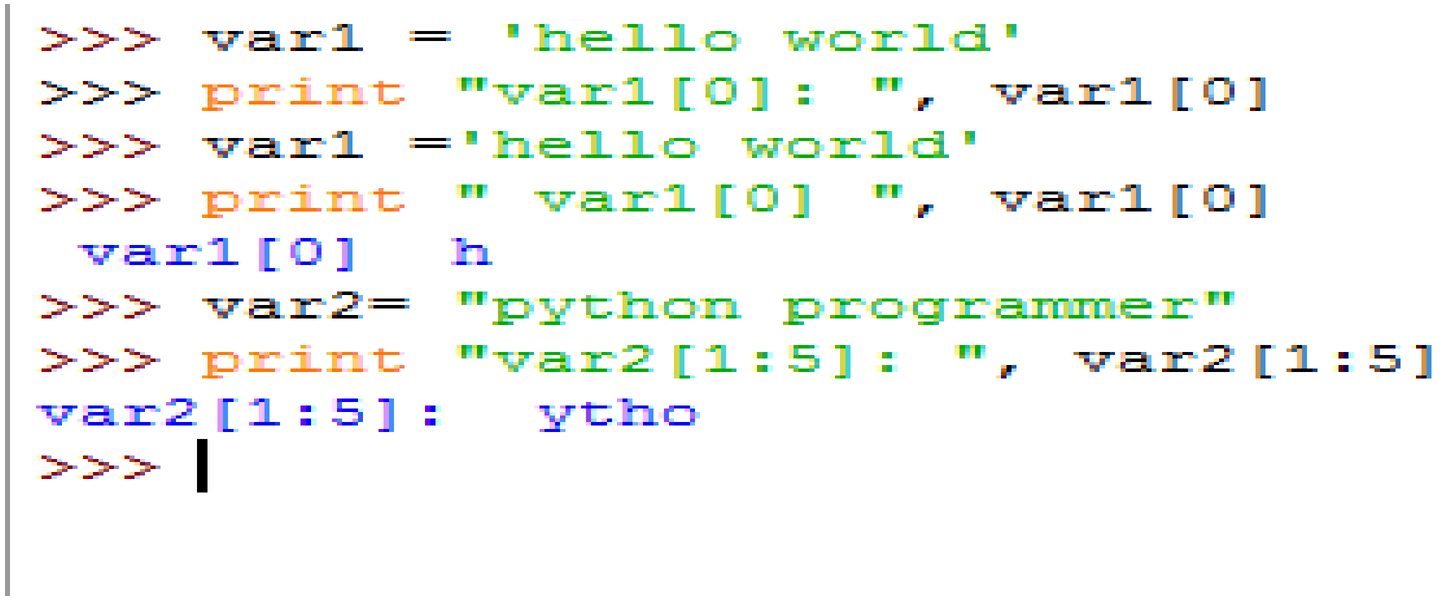
```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> var1='hello world!'
>>> print var1
hello world!
>>> var2 = "python programmer!"
>>> print var2
python programmer!
>>>
```

Accessing Values in Strings

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.

For example :

- var1 = 'Hello World!‘
- var2 = "Python Programming“
- print "var1[0]: ", var1[0]
- print "var2[1:5]: ", var2[1:5]



```
>>> var1 = 'hello world'
>>> print "var1[0]: ", var1[0]
>>> var1 ='hello world'
>>> print " var1[0] ", var1[0]
 var1[0]  h
>>> var2= "python programmer"
>>> print "var2[1:5]: ", var2[1:5]
var2[1:5]:   ytho
>>> 
```

Examples

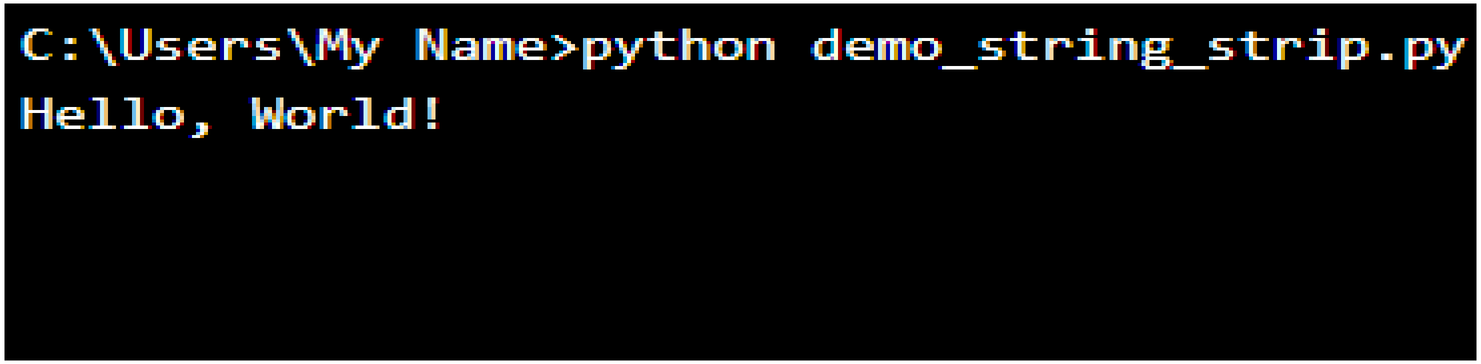
**1.The Strip(**)– to remove any white space from the beginning or the end:

- **a = “ hello, world !”**
- **print(a.strip())**

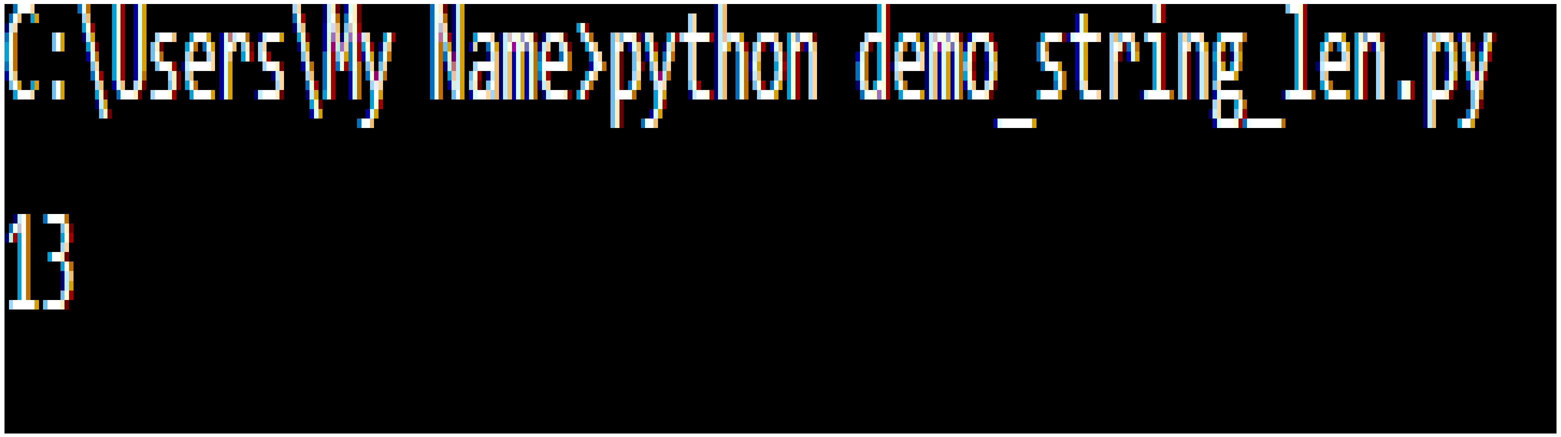
2**. Len-**The len() method returns the length of a string

a = “ Hello, world!”

Print(len(a))



```
C:\Users\My Name>python demo_string_strip.py
Hello, World!
```



```
C:\Users\My Name>python demo_string_len.py

13
```

3.lower() : the lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

4.Upper(): the upper() method returns the string in upper case:

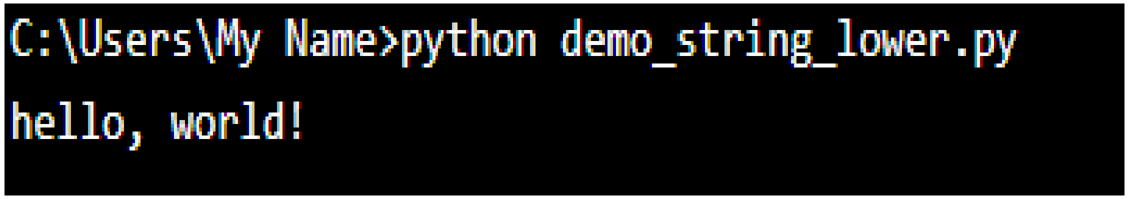
```
a = "Hello, World!"
print(a.upper())
```

5. Replace

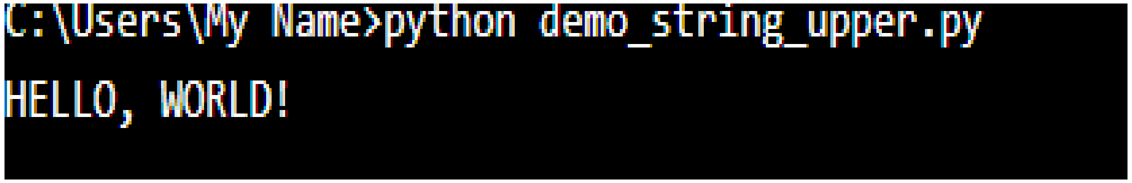
```
a = "Hello, World!"
print(a.replace("H", "J"))
```

6.Splite :

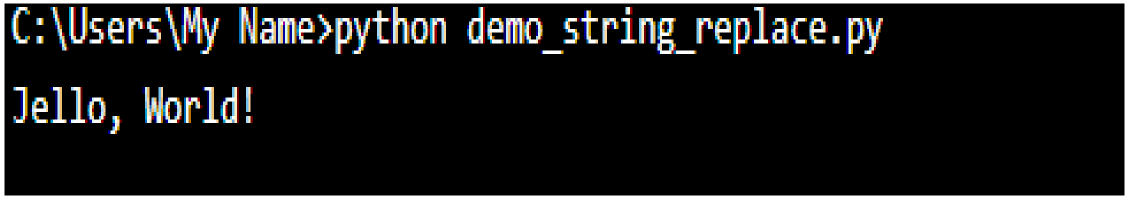
```
a = "Hello, World!"
print(a.splite(", "))
```



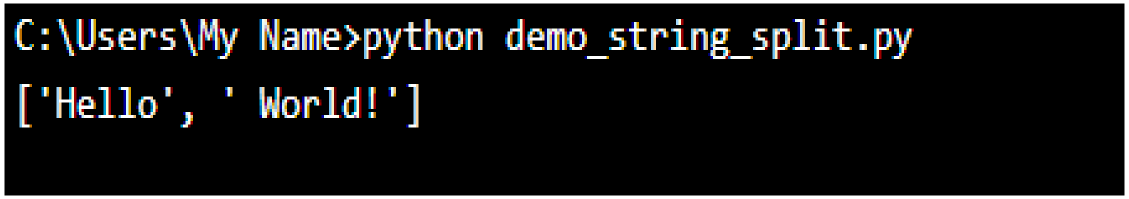
```
C:\Users\My Name>python demo_string_lower.py
hello, world!
```



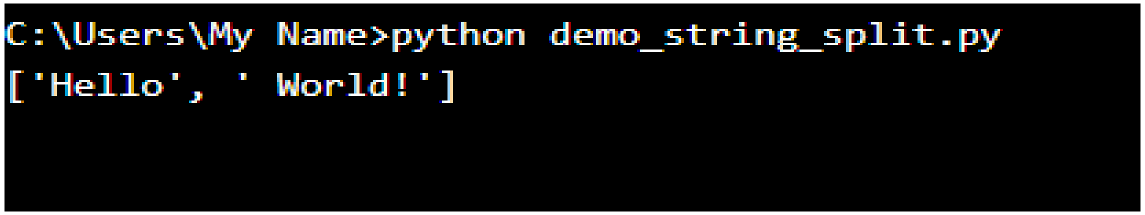
```
C:\Users\My Name>python demo_string_upper.py
HELLO, WORLD!
```



```
C:\Users\My Name>python demo_string_replace.py
Jello, World!
```



```
C:\Users\My Name>python demo_string_split.py
['Hello', ' World!']
```



```
C:\Users\My Name>python demo_string_split.py
['Hello', ' World!']
```

Python- List

- The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.
- is a collection which is ordered and changeable. Allows duplicate members.

Examples

- **thislist = ["apple", "banana", "cherry"]**
  print(thislist)
- Output: **"apple", "banana", "cherry“**

1. Access Items:

- thislist = ["apple", "banana", "cherry"]
  print thislist[1]

2. **Change the items**:

  thislist = ["apple", "banana", "cherry"]
  thislist[1] = "blackcurrant"
  print thislist

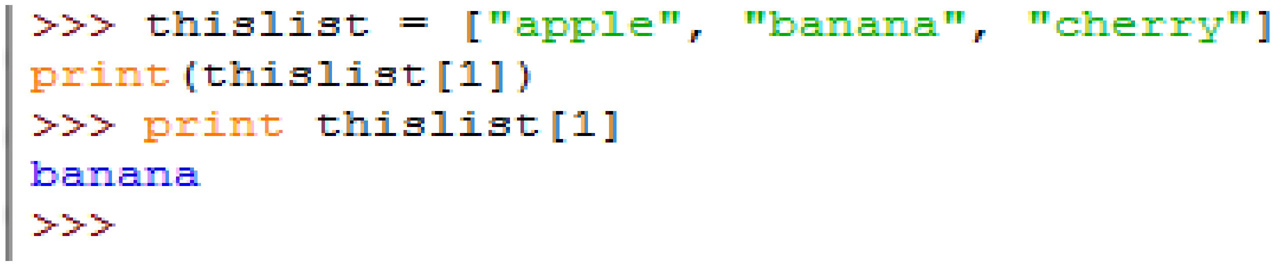
3. Add Items:

thislist = [“apple”, “banana”, “cherry”]

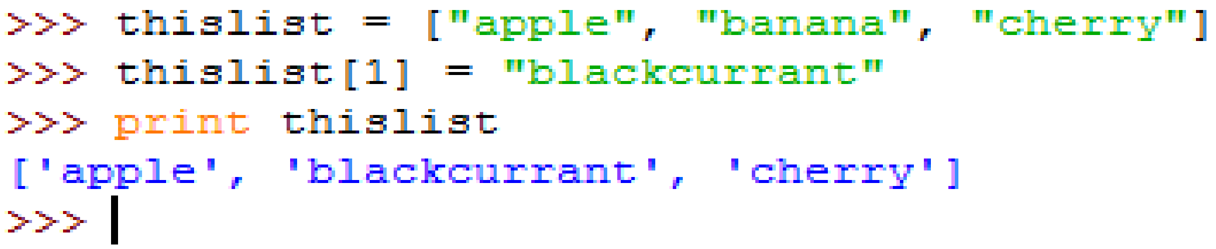
Thislist.append(“orange”)

Print thislist

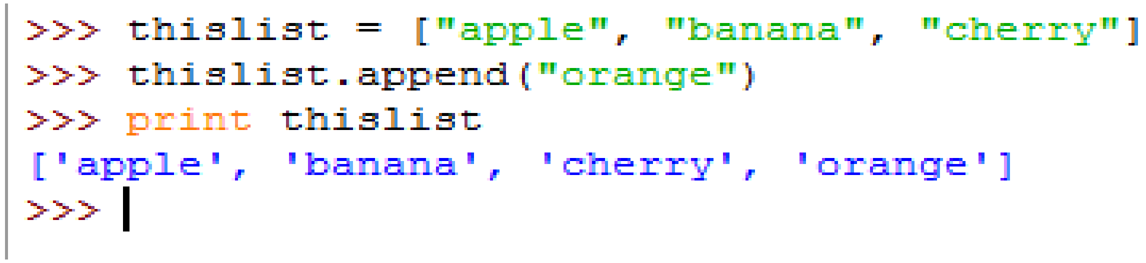
Output(screenshot)



```
>>> thislist = ["apple", "banana", "cherry"]
print(thislist[1])
>>> print thislist[1]
banana
>>>
```



```
>>> thislist = ["apple", "banana", "cherry"]
>>> thislist[1] = "blackcurrant"
>>> print thislist
['apple', 'blackcurrant', 'cherry']
>>>
```



```
>>> thislist = ["apple", "banana", "cherry"]
>>> thislist.append("orange")
>>> print thislist
['apple', 'banana', 'cherry', 'orange']
>>>
```

4. Insert: insert the item at the specifix index

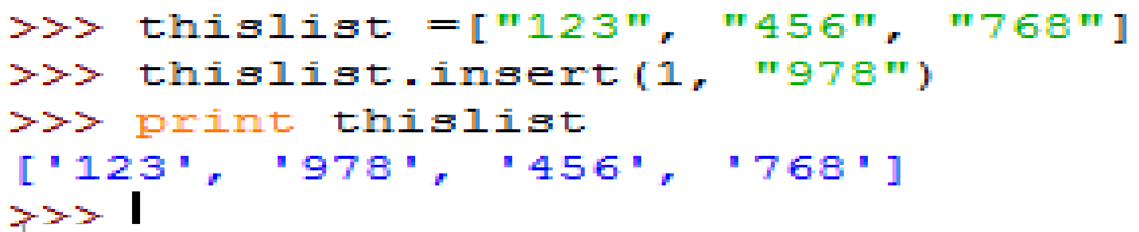
```
thislist = [“123”, “456”, “768”]
thislist.insert(1, “987”)
print thislist
```

5. Remove: Remove() the spceified items.

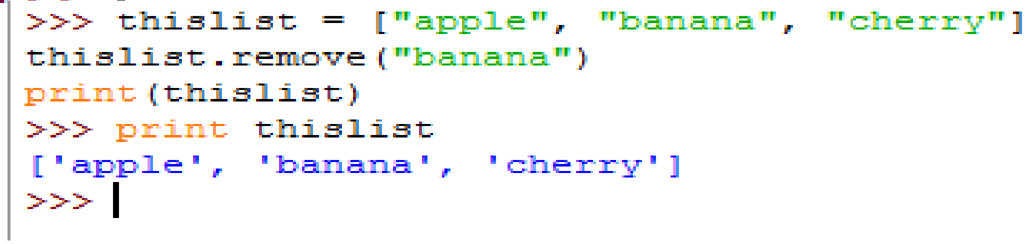
```
 thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print thislist
```

6. Delete: delete the spceified index

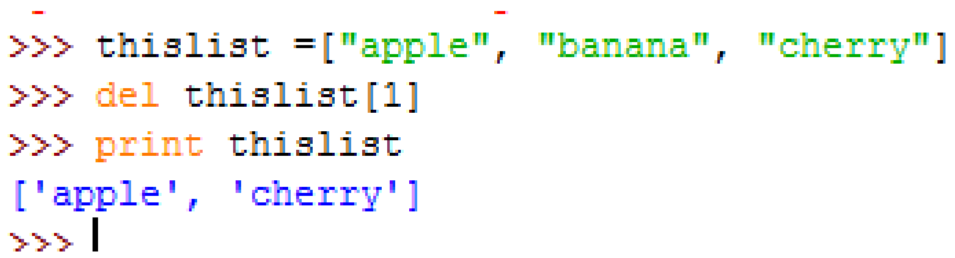
```
thislist = ["apple", "banana", "cherry"]
del thislist[1]
print thislist
```



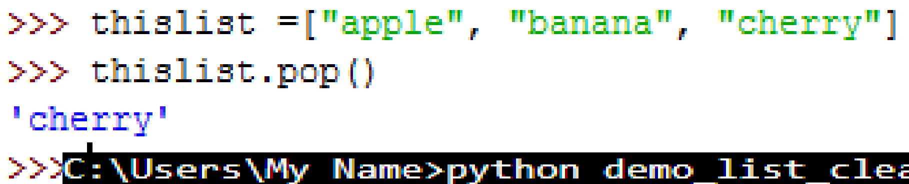
```
>>> thislist =["123", "456", "768"]
>>> thislist.insert(1, "978")
>>> print thislist
['123', '978', '456', '768']
>>>
```



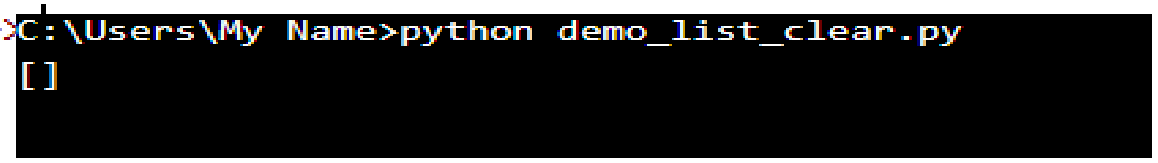
```
>>> thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
>>> print thislist
['apple', 'banana', 'cherry']
>>>
```



```
>>> thislist =["apple", "banana", "cherry"]
>>> del thislist[1]
>>> print thislist
['apple', 'cherry']
>>>
```



```
>>> thislist =["apple", "banana", "cherry"]
>>> thislist.pop()
'cherry'
>>>
```



```
C:\Users\My Name>python demo_list_clear.py
[]
```

7.pop() : this method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print thislist
```

8.Clear():

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print thislist
```

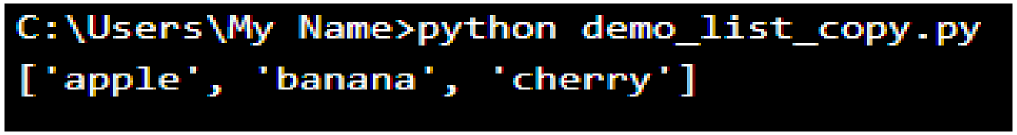
Copy a list

- You cannot copy a list simply by typing list2 = list1, because: list2 will only be a *reference* to list1, and changes made in list1 will automatically also be made in list2.
- There are ways to make a copy, one way is to use the built-in List method copy().

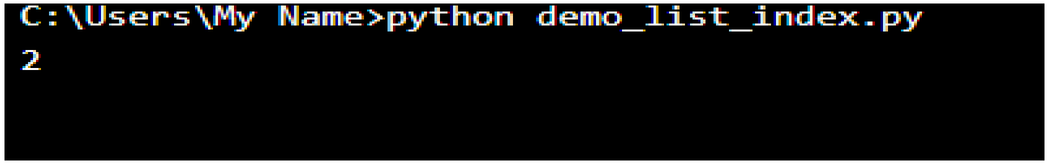
Example:

- Make a copy of a list with the copy() method:

-
  ```
  thislist = ["apple", "banana", "cherry"]
  mylist = thislist.copy()
  print(mylist)
  ```



```
C:\Users\My Name>python demo_list_copy.py
['apple', 'banana', 'cherry']
```



```
C:\Users\My Name>python demo_list_index.py
2
```

9. *Index:(syntax- list*.index(*elmnt*)

What is the position of the value "cherry":

```
 fruits = ['apple', 'banana', 'cherry‘]
 x = fruits.index("cherry")
```

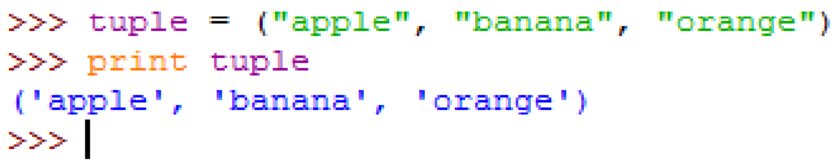
Python - Tuple

- A **tuple** is a sequence of immutable Python objects. Tuples are sequences, just like lists.
- are, the tuples**The differences between tuples and lists** cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

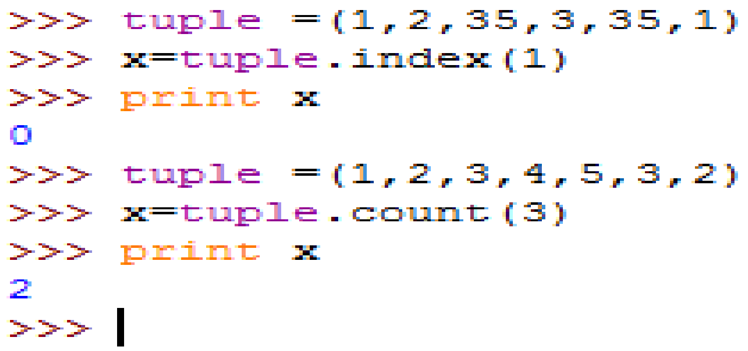
Create a Tuple:

tuple= (“apple”, “banana”, “orange”)

Print tuple



```
>>> tuple = ("apple", "banana", "orange")
>>> print tuple
('apple', 'banana', 'orange')
>>> 
```



```
>>> tuple =(1,2,35,3,35,1)
>>> x=tuple.index(1)
>>> print x
0
>>> tuple =(1,2,3,4,5,3,2)
>>> x=tuple.count(3)
>>> print x
2
>>> 
```

- Access Tuple items :

tuple =(“in”, are“”, “ we”)

Print tuple[1]

Adding, deleting, modification are not possible in tuple.

1.tuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = tuple.count(5)

print(x)

2.tuple=(1,2,3,5,7,8,3,5,3,8)

x=tuple.index(3)

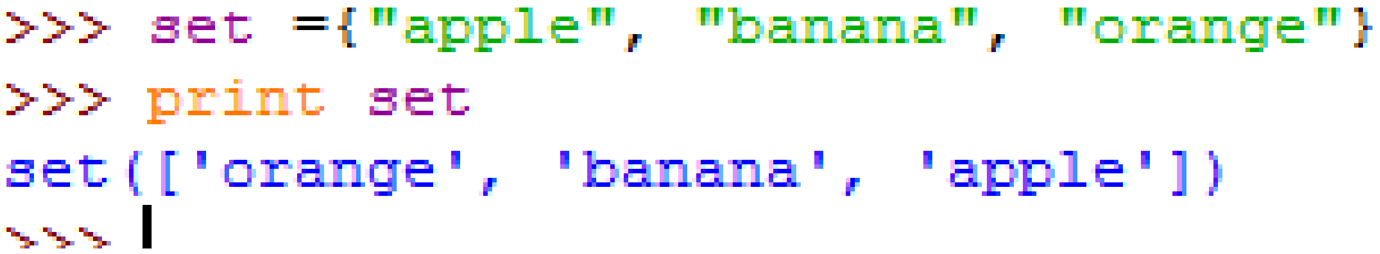
Print x

Python - Set

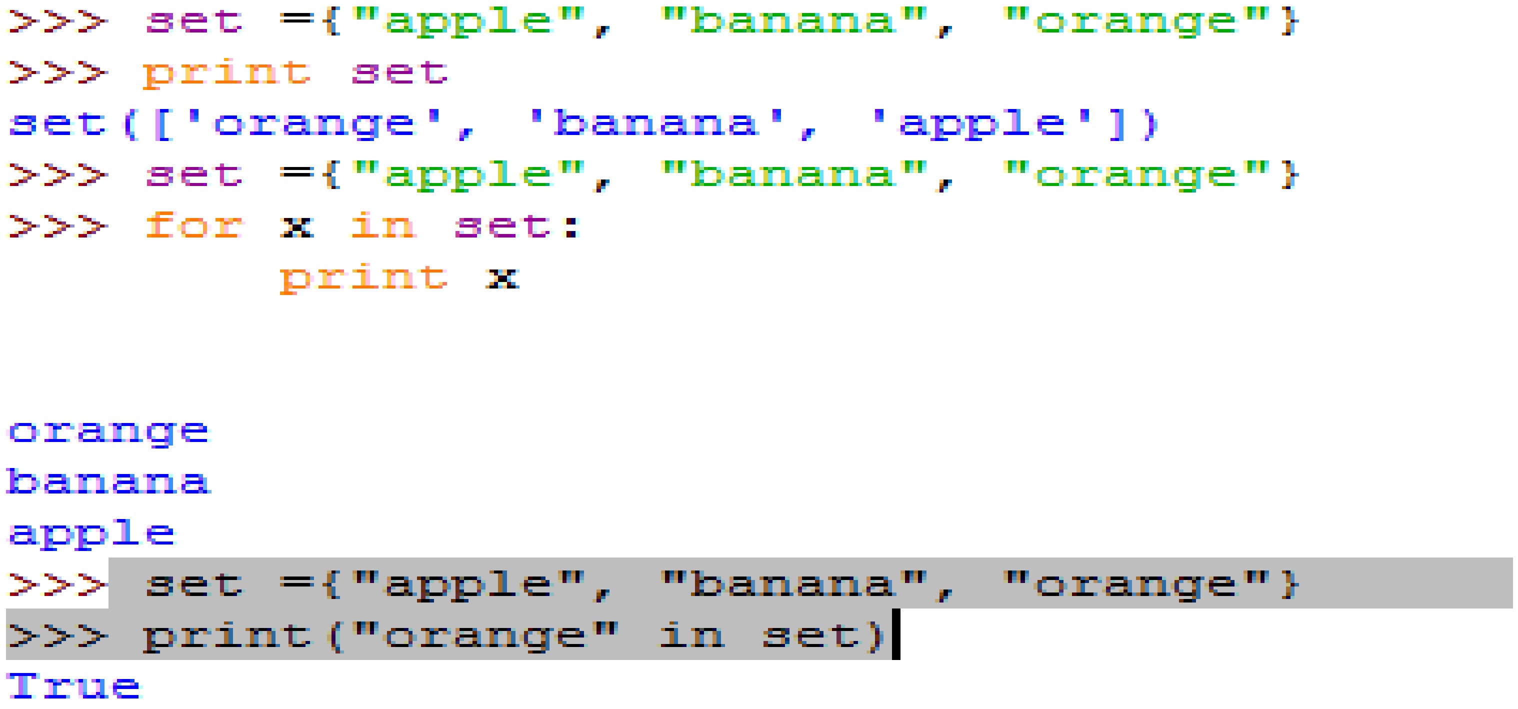
- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.
- Once the set is created you cannot change its items, but you can add new items.
- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using thein keyword.

- **Examples:**

```
Set = {“apple”, “banana”, “cherry”}
print set
2. set = {"apple", "banana", "cherry“}
   for x in set:
    print(x)
3. set ={"apple", "banana", "orange"}
   print("orange" in set)
```



```
>>> set ={"apple", "banana", "orange"}
>>> print set
set(['orange', 'banana', 'apple'])
>>> 
```



```
>>> set ={"apple", "banana", "orange"}
>>> print set
set(['orange', 'banana', 'apple'])
>>> set ={"apple", "banana", "orange"}
>>> for x in set:
        print x


orange
banana
apple
>>> set ={"apple", "banana", "orange"}
>>> print("orange" in set)
True
```

4.Get the number of item in a set using **length**:

set={“apple”, “banana”, “orange”}

Print len(set)

5. Set ={"apple", "banana", "orange"}

set.remove(“orange”)

print set

6.Remove the item using Discard() method.

set = {"apple", "banana", "orange"}

set.discard(“orange”)

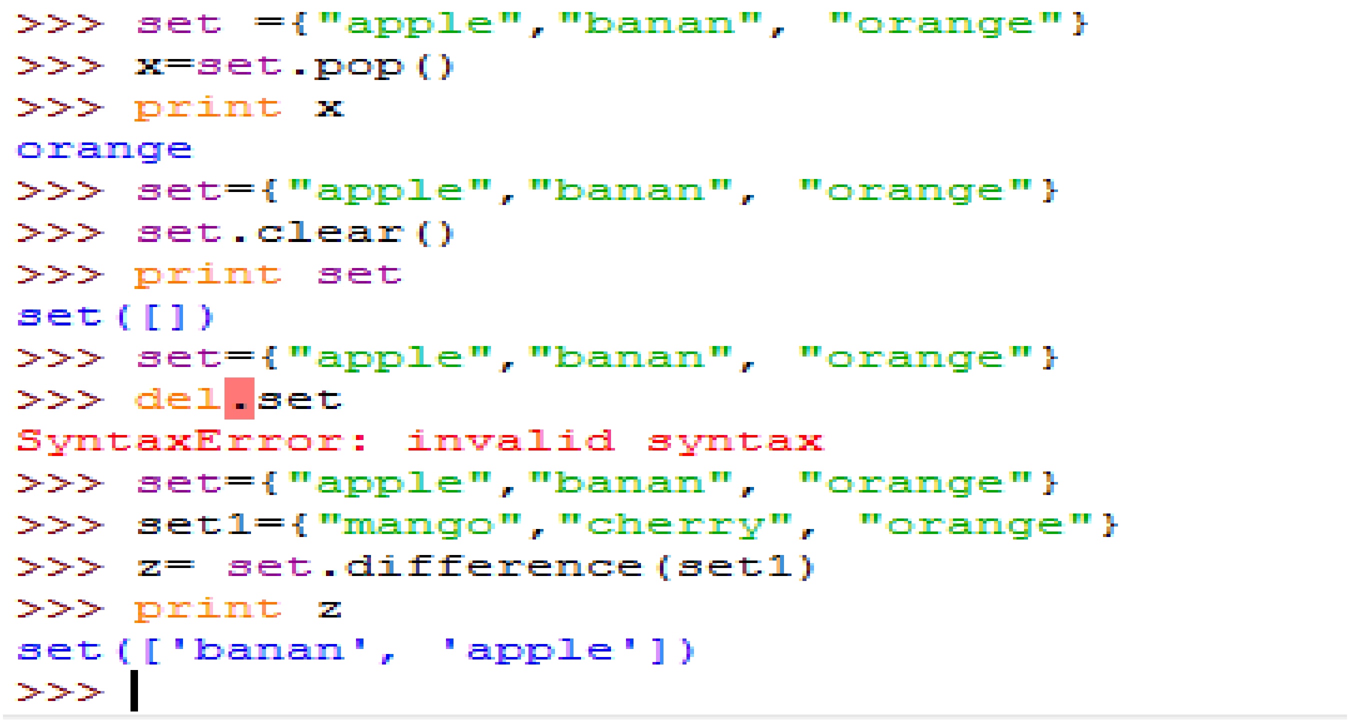
print set

7.set = {"apple", "banana", "cherry"}
```
x = set.pop()
print(x)
print set
```

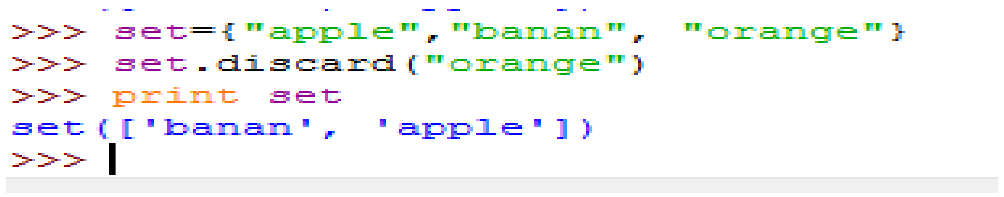
8. set = {"apple", "banana", "cherry"}
```
set.clear()
print set
```

9. set = {"apple", "banana", "cherry"}
```
del.set
print set
```

10. x = {"apple", "banana", "cherry"}
```
y={“mango”, “orange”, “banan”}
z= x.differnce(y)
print z
```



```
>>> set ={"apple","banan", "orange"}
>>> x=set.pop()
>>> print x
orange
>>> set={"apple","banan", "orange"}
>>> set.clear()
>>> print set
set([])
>>> set={"apple","banan", "orange"}
>>> del.set
SyntaxError: invalid syntax
>>> set={"apple","banan", "orange"}
>>> set1={"mango","cherry", "orange"}
>>> z= set.difference(set1)
>>> print z
set(['banan', 'apple'])
>>> 
```



```
>>> set={"apple","banan", "orange"}
>>> set.discard("orange")
>>> print set
set(['banan', 'apple'])
>>> 
```

1.
   ```
   x = {"apple", "banana", "cherry"}
   y = {"google", "microsoft", "apple"}
   x.update(y)
   print(x)
   ```

o/p: {'google', 'cherry', 'microsoft', 'apple', 'banana'}

2.
   ```
   x = {"apple", "banana", "cherry"}
   y = {"google", "microsoft", "apple"}
   z = x.intersection(y)
   print(z)
   ```

o/p :{ ‘apple’}

Python - Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example:

```
dict={“brand” : “ Ford”,
        “model”: “mustang”,
        “year”: 1993}
Print dict
```

1. **Accessing Items**: You can access the items of a dictionary by referring to its key name.

- Example: 1) x = thisdict["model"]

2. There is also a method called get()

x = thisdict.get("model")

2. **Change Values**: You can change the value of a specific item by referring to its key name.

```
thisdict = {
   "brand": "Ford",
   "model": "Mustang",
   "year": 1964
  }
  thisdict["year"] = 2018
```

3. **Dictionary Length**: print(len(thisdict))

4. **Adding Items**: thisdict["color"] = "red"
   print(thisdict)

5. **Removing Items:** thisdict.pop("model")
   print(thisdict)

The popitem() method removes the last inserted item

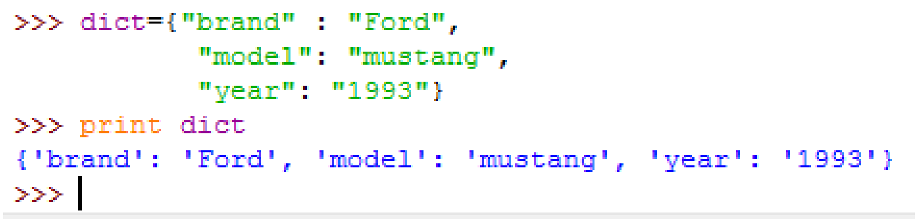
1.thisdict.popitem()

print(thisdict)

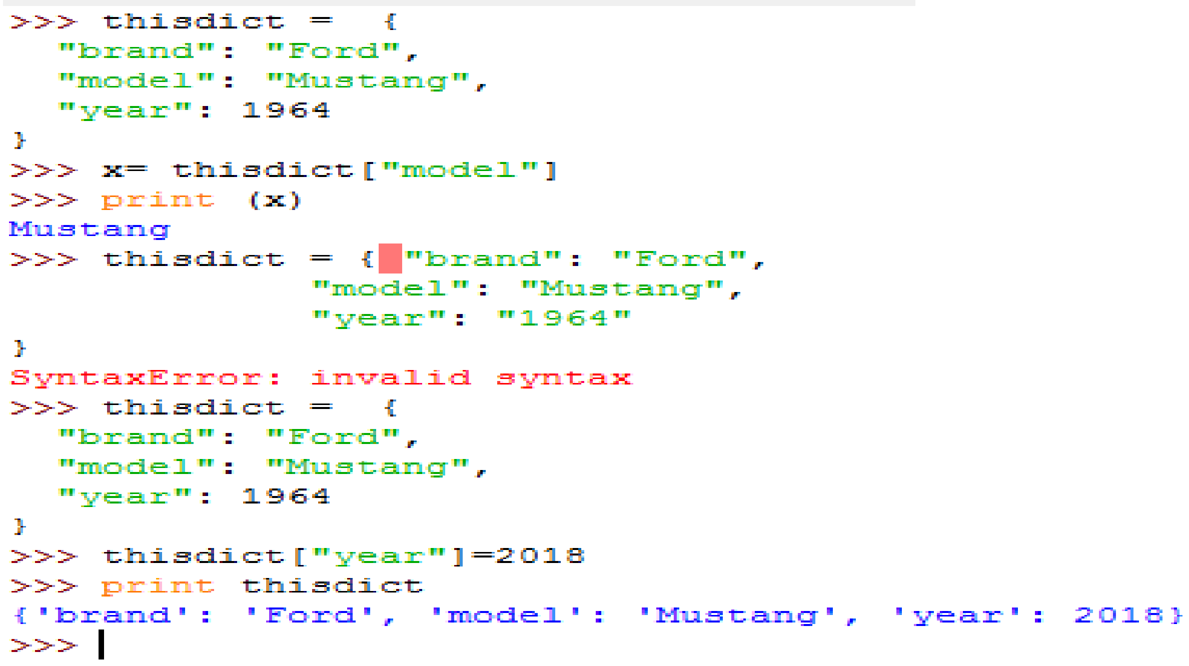
2. del thisdict["model"]
    print(thisdict)

3. thisdict.clear()
    print(thisdict)

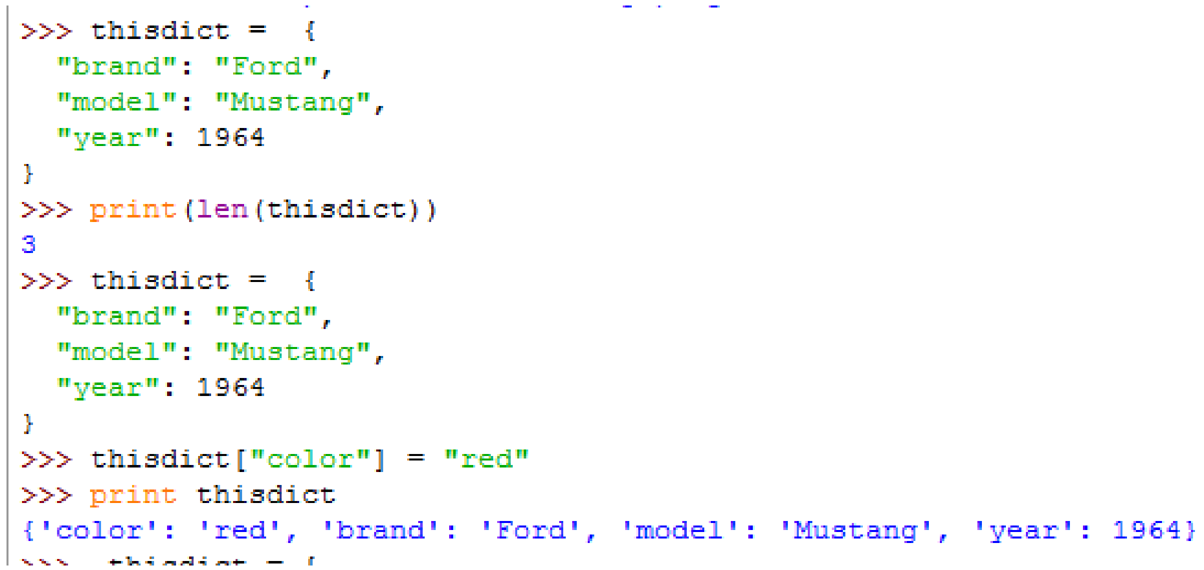
4. mydict = thisdict.copy()
    print(mydict)



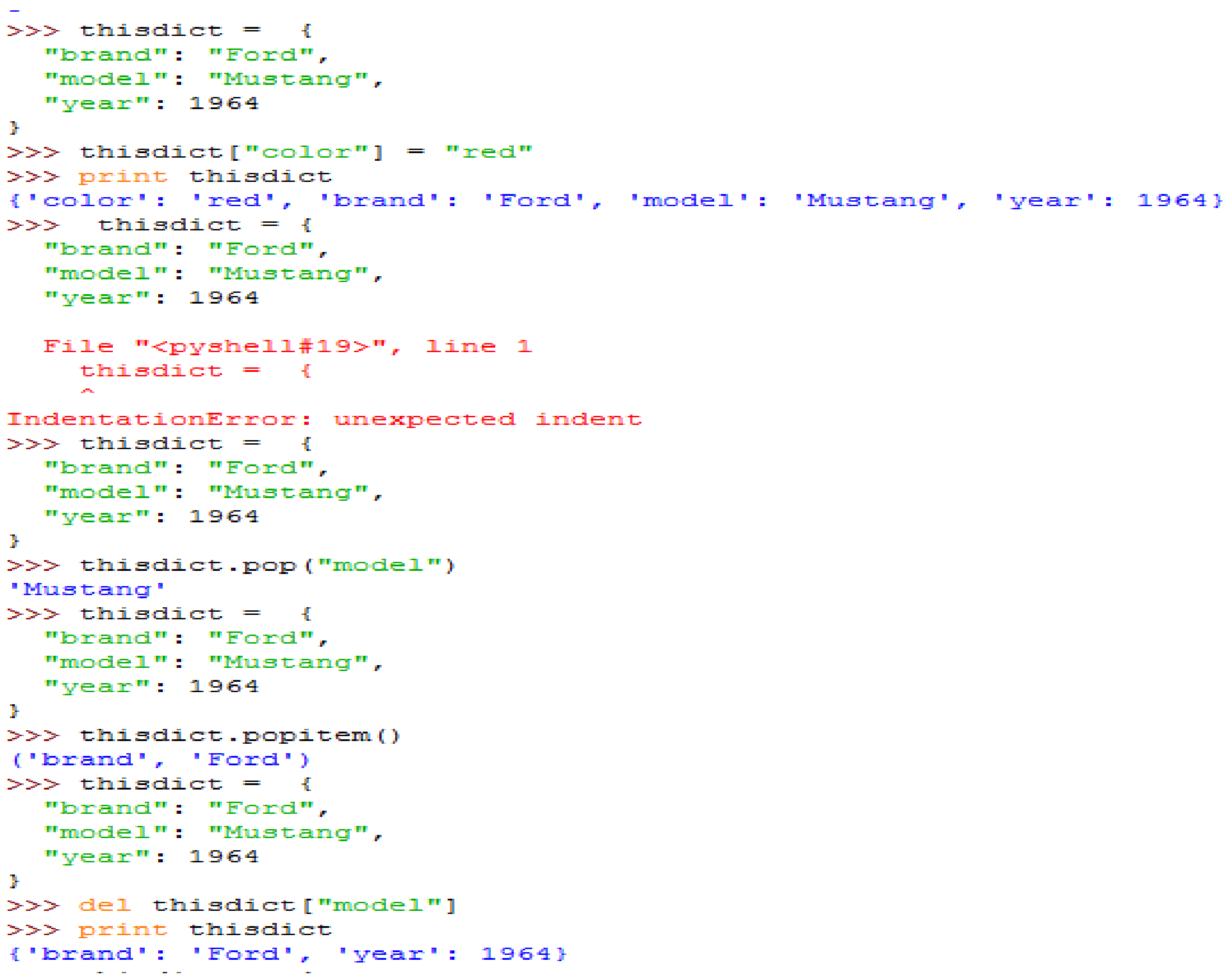
```
>>> dict={"brand" : "Ford",
           "model": "mustang",
           "year": "1993"}
>>> print dict
{'brand': 'Ford', 'model': 'mustang', 'year': '1993'}
>>> 
```



```
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> x= thisdict["model"]
>>> print (x)
Mustang
>>> thisdict = { "brand": "Ford",
                 "model": "Mustang",
                 "year": "1964"
}
SyntaxError: invalid syntax
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict["year"]=2018
>>> print thisdict
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
>>> 
```

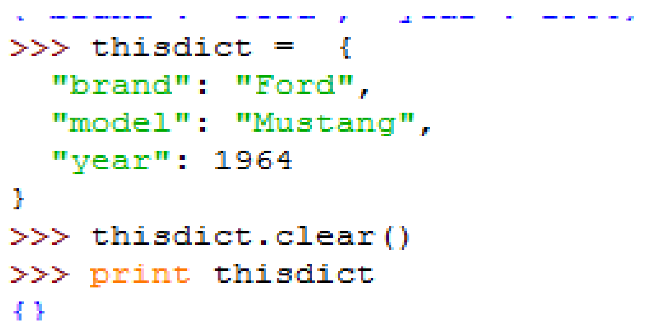


```
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> print(len(thisdict))
3
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict["color"] = "red"
>>> print thisdict
{'color': 'red', 'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

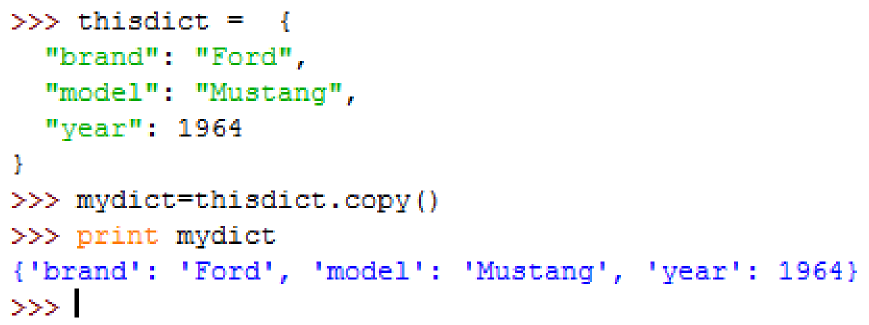


```
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict["color"] = "red"
>>> print thisdict
{'color': 'red', 'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>>  thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964

  File "<pyshell#19>", line 1
    thisdict =  {
    ^
IndentationError: unexpected indent
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict.pop("model")
'Mustang'
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict.popitem()
('brand', 'Ford')
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> del thisdict["model"]
>>> print thisdict
{'brand': 'Ford', 'year': 1964}
```



```
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict.clear()
>>> print thisdict
{}
```



```
>>> thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> mydict=thisdict.copy()
>>> print mydict
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>>
```

Python-Conditional statment

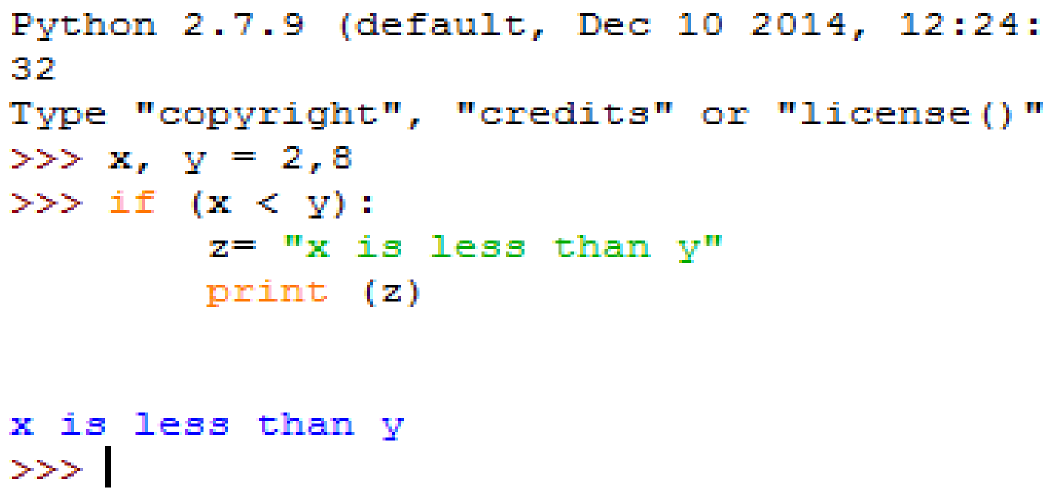
- **What are Conditional Statements?**
- Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

- **What is If Statement? How to Use it?**
- In Python, If Statement is used for decision making. It will run the body of code only when IF statement is true.
- When you want to justify one condition while the other condition is not true, then you use "if statement".
- Syntax:

```
if expression:
  Statement
else:
  Statement
```

Examples:

-
  ```
  x, y = 2,8
  if (x < y):
  z= "x is less than y“
  print (z)
  ```
- How to use "else condition"
- The "else condition" is usually used when you have to judge one statement on the basis of other. If one condition goes wrong, then there should be another condition that should justify the statement or logic.

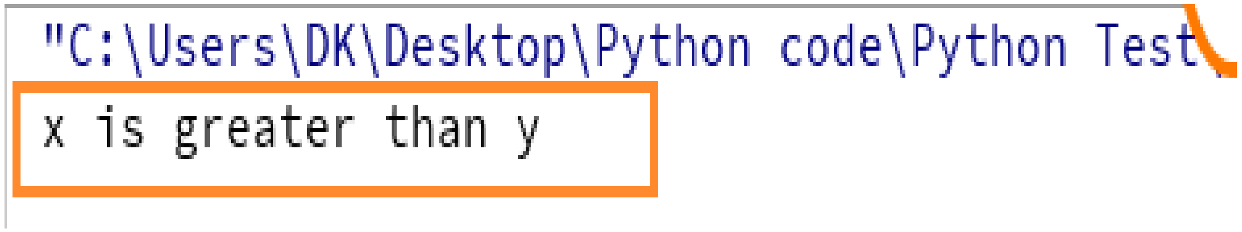


```
Python 2.7.9 (default, Dec 10 2014, 12:24:
32
Type "copyright", "credits" or "license()"
>>> x, y = 2,8
>>> if (x < y):
        z= "x is less than y"
        print (z)


x is less than y
>>> 
```

Examples:

```
def main():
x, y = 2,8
    if (x < y):
    z= "x is less than y“
    else:
            z=“x is greater than y”
     print (z)
If __name__ == “ “”main__”:
      main()
```



```
"C:\Users\DK\Desktop\Python code\Python Test
x is greater than y
```

‘

How to use "elif" condition

- To correct the previous error made by "else condition", we can use **"elif"** statement. By using "**elif**" condition, you are telling the program to print out the third condition or possibility when the other condition goes wrong or incorrect.

Examples:

```
x,y =8,8
if(x < y):
        st= "x is less than y“
 elif (x == y):
        st= "x is same as y“
 else:
      st="x is greater than y“
      print(st)
```

Nested IF Statement:

Examples:

```
total = 100
 #country = "US“
 country = "AU“
 if country == "US“
: if total <= 50:
 print("Shipping Cost is $50")
 elif total <= 100:
print("Shipping Cost is $25")
 elif total <= 150:
 print("Shipping Costs $5")
 else:
 print("FREE")
if country == "AU":
if total <= 50:
print("Shipping Cost is $100")
 else:
 print("FREE")
```

- Switch Statement:

A switch statement is a multi way branch statement that compare the value of a variable the value specified in case statements.

Python language doesn’t have a switch statement.

Python- loop statment

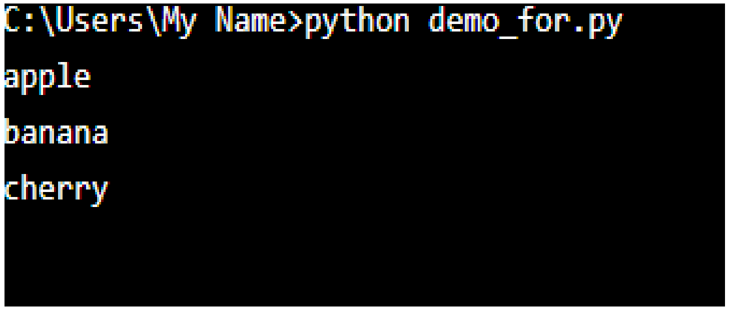
- A loop statement allows us to execute a statement or group of statements multiple times.

For Loop: Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

- While Loop: Repeats a statement or group of statements while a given condition is loop body. TRUE. It tests the condition before executing the
- Examplers:

1. Ex. For loop: fruits = ["apple", "banana", "cherry"]

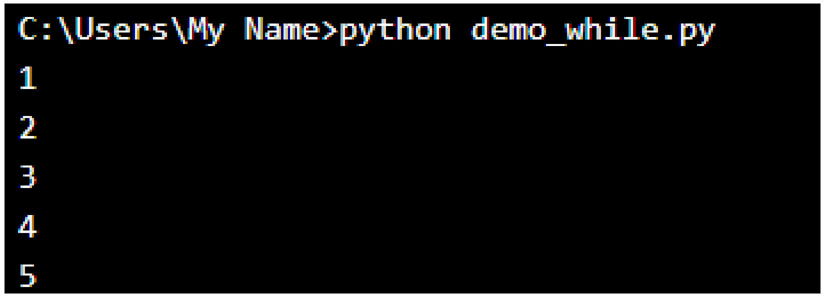
```
for x in fruits:
print(x)
```



```
C:\Users\My Name>python demo_for.py
apple
banana
cherry
```

2. ex. While loop:

```
i = 1
    while i < 6:
     print(i)
     i += 1
```



```
C:\Users\My Name>python demo_while.py
1
2
3
4
5
```

Python-loop control statement

- **Break Statement**: The break statement we can stop the loop before it has looped through all the items:

Examples:

```
X=[“ apple”, “orange”, “ banana”]
for a in x:
Print (a)
If a ==“orange”:
Break
Print (a)
```

- **Continue Statement:** With the continue statement we can stop the current iteration of the loop, and continue with the next:

Examples:

```
fruits = ["apple", "banana", "cherry"]
    for x in fruits:
      if x == "banana":
        continue
      print(x)
```

- **Pass Stat**: The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- Example:
-
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    pass
  print(x)
```

Python- Function

- **Function:** is a block of code which only runs when it is called. And you can pass the data known as parameters, into a function.
- A function can return data as a result.
- In python a function defined using the **def** keyword **.**

Examples:

```
def my_function():
    print("Hello from a function")
```

- **Calling a Function**
- To call a function, use the function name followed by parenthesis:
- **def my_function():**

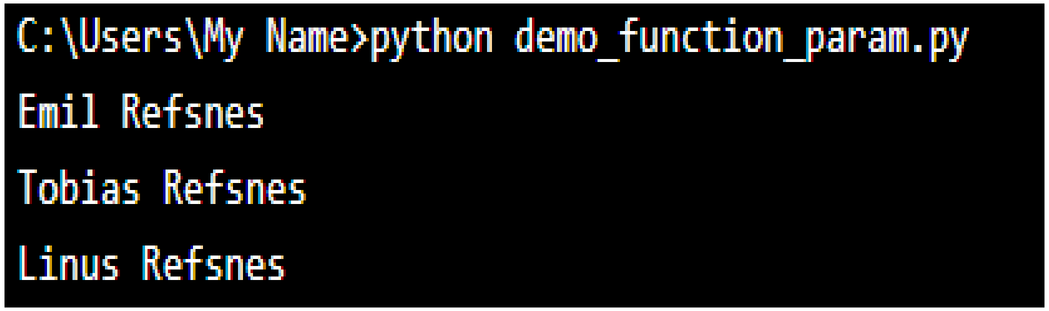
```
def my_function():
  print("Hello from a function")
  my_function()
```

Parameters

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
- **Examples:**
- The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

-
  ```
  def my_function(fname):
    print(fname + " Refsnes")

  my_function("Emil")
  my_function("Twitter")
  my_function("Linus")
  ```
- Output:



```
C:\Users\My Name>python demo_function_param.py
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

The *return* Statement

- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Example:

```
def sum( arg1, arg2 ):
# Add both the parameters and return them."
total = arg1 + arg2
print "Inside the function : ", total
return total;
 # Now you can call sum function
total = sum( 10, 20 );
 print "Outside the function : ", total
```

OutPut: Inside the function : 30

```
            Outside the function : 30
```

Python –OOPs Concepts

Python class and objects:

A blueprint created by a programmer for an object. This defines a set of attributes that will characterize any object that is instantiated from this class.

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class: To create a class, use the keyword **class**:

Example:

```
class MyClass:
 x=5:
```

- **Object** — An instance of a class.
- **Examples:**

```
Class Shark:
        def swim(self):
                print(“thr shark is swimming”)
       def  be_awesome(self):
                 print(“the shark is being awwsome”)
#o object creating
A= Shark()
A.swim()
A.be_awesome()
```

The constructor method

- The constructor method is used to initialize data. It is run as soon as an object of a class is instantiated. Also known as the __init__ method, it will be the first definition of a class and looks like this:
- Examples:

```
class Shark:
 def __init__(self, name):
self.name = name
def swim(self):
print(self.name + " is swimming.")
def be_awesome(self):
 print(self.name + " is being awesome.")
 def main():
 # Set name of Shark object
q = Shark("Sammy")
 q.swim()
q.be_awesome()
 if __name__ == "__main__": main()
```

Output:

```
q is swimming.
q is being awesome.
```

Python- Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.

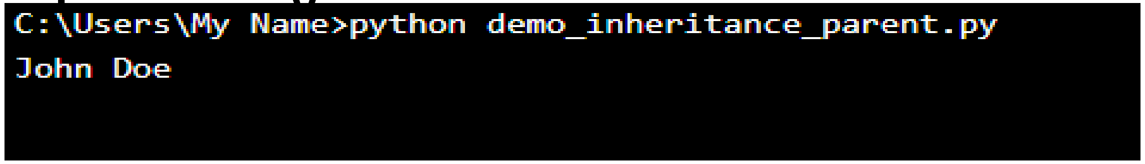
Create a Parent Class:

Any class can be a parent class, so the syntax is the same as creating any other class

Examples:

-
  ```
  class Person:
    def __init__(self, fname, lname):
      self.firstname = fname
      self.lastname = lname

    def printname(self):
      print(self.firstname, self.lastname)
  #Use the Person class to create an object, and then execute the
  printname method:
  x = Person(“john", “Doe")
  x.printname()
  ```



```
C:\Users\My Name>python demo_inheritance_parent.py
John Doe
```

Create a Child class:

Examples:

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.
- Create a class named Student, which will inherit the properties and methods from the Person class.

```
class Student(Person):
    pass
```

Python - Encapsulation

Encapsulation means that the internal representation of on object is generally hidden from view outside of the object’s definition.

Encapsulation:we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation

In Python, we denote private attribute using underscore as prefix i.e single “ _ “ or double “ __“.

- Example:

```
class Computer:
def __init__(self):
self.__maxprice = 900
def sell(self):
print("Selling Price: {}".format(self.__maxprice))
def setMaxPrice(self, price):
self.__maxprice = price
c = Computer()c.sell()
# change the price
c.__maxprice = 1000
c.sell()
# using setter function
c.setMaxPrice(1000)
c.sell()
```

Python- Regular Expr

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

- Python has a built-in package called re, which can be used to work with Regular Expressions.
- Import the re module

```
import re
```

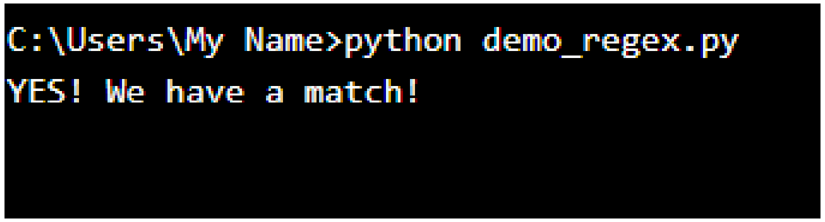
RegEx in Python:

When you have imported the re module, you can start using regular expressions:

- **Example**

Search the string to see if it starts with "The" and ends with "Spain"

```
import re
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```



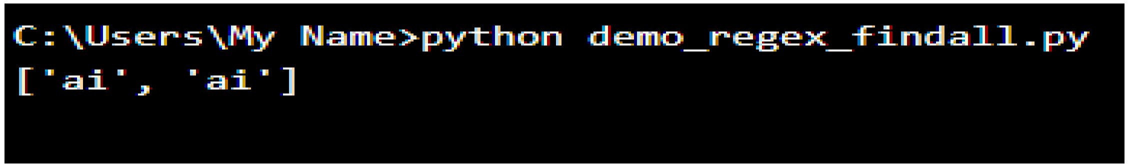
```
C:\Users\My Name>python demo_regex.py
YES! We have a match!
```

- **RegEx Functions**:

1. **Findall**: Returns a list containing all matches
2. **Search**: Returns a match object if there is a match anywhere in the string.
3. **Split**: Returns a list where the string has been split at each match
4. **Sub**: Relplaces one or many matches with a string.

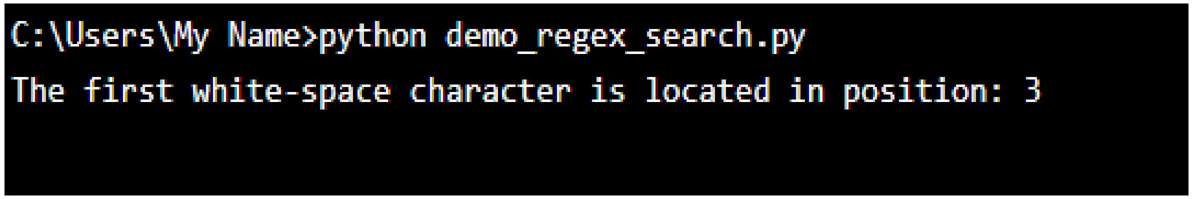
Examples:

1. import re
   str = "The rain in Spain"
   x = re.findall("ai", str)
   print(x)



```
C:\Users\My Name>python demo_regex_findall.py
['ai', 'ai']
```

- import re
  str = "The rain in Spain"
  x = re.search("\s", str)
  print("The first white-space character is located in position:", x.start())

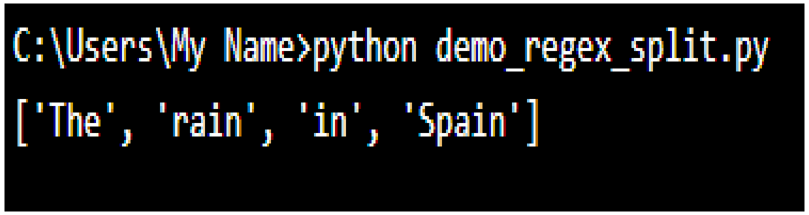


```
C:\Users\My Name>python demo_regex_search.py
The first white-space character is located in position: 3
```

3.import re

```
str = "The rain in Spain"
x = re.split("\s", str)
print(x)
```

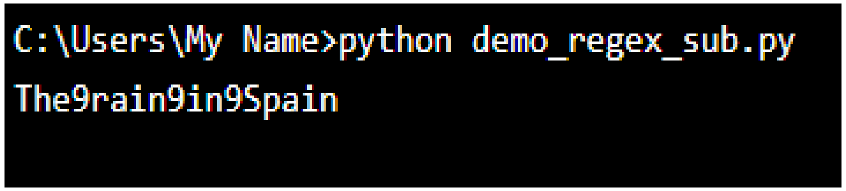
Output:====➔



```
C:\Users\My Name>python demo_regex_split.py
['The', 'rain', 'in', 'Spain']
```

4. import re

```
str = "The rain in Spain"
x = re.sub("\s", "9", str)
print(x)
```



```
C:\Users\My Name>python demo_regex_sub.py
The9rain9in9Spain
```

Python –File Handling

- File handling is an important part of any web application.
- Python has several functions for creating, reading, updating, and deleting files.
- The key function for working with files in Python is the **open()** function.
- The open() function takes two parameters; ***filename*, and *mode*.**
- There **are four different methods (modes**) for opening a file:
- **"r"** - Read - Default value. Opens a file for reading, error if the file does not exist
- **"a"** - Append - Opens a file for appending, creates the file if it does not exist
- **"w"** - Write - Opens a file for writing, creates the file if it does not exist

Syntax: To open a file for reading it is enough to specify the name of the file

f = open("demofile.txt")

- **Open a file on the server:** assume we have the following file, located in the same folder as pythom.

demofile.txt

“ Hello! Welcome to demofile.txt

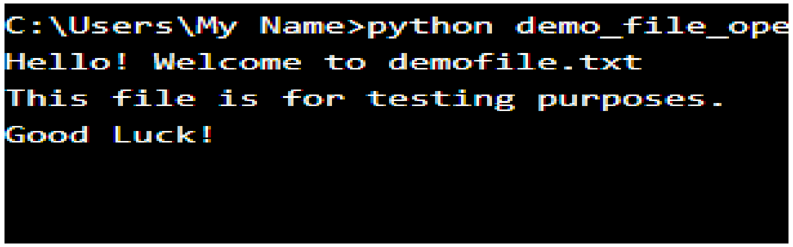
this file is for testing purposes.

Good Luck”

- To open the file, use the built-in open() function.
- The open() function returns a file object, which has a read() method for reading the content of the file

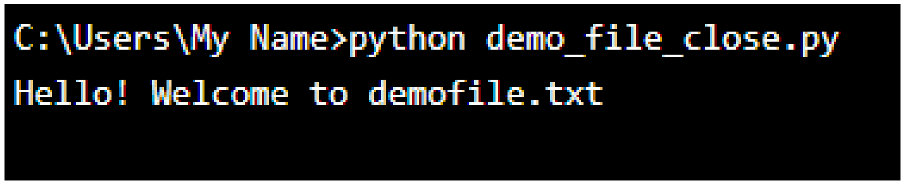
Examples:

1. **f = open("demofile.txt", "r")**
   print(f.read())



```
C:\Users\My Name>python demo_file_ope
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

2. **f = open("demofile.txt", "r")**
   print(f.readline())
   f.close()



```
C:\Users\My Name>python demo_file_close.py
Hello! Welcome to demofile.txt
```

Python- file write

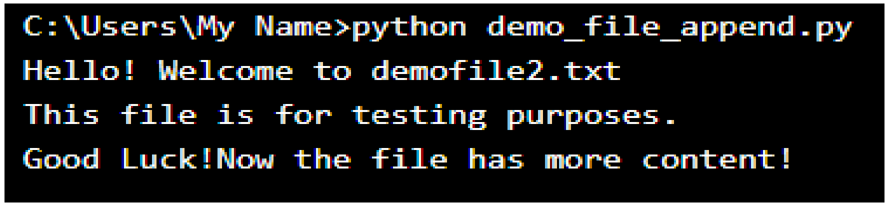
- To write to an existing file, you must add a parameter to the open() function:
- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content

Examples:f =

```
open("demofile2.txt", "a")
 f.write("Now the file has more content!")
 f.close()
```

#open and read the file after the appending:

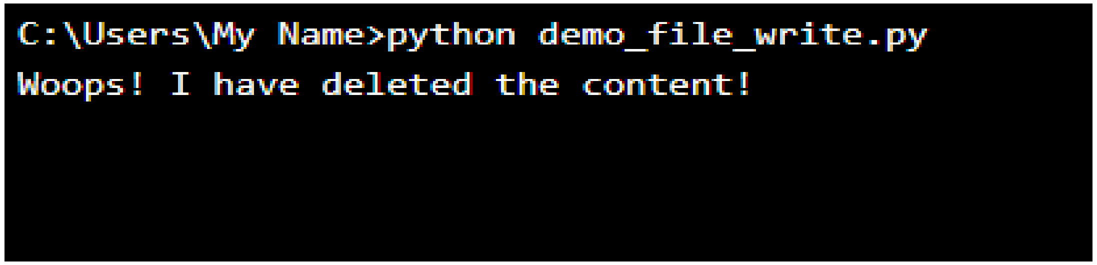
```
f = open("demofile2.txt", "r")
 print(f.read())
```


```
C:\Users\My Name>python demo_file_append.py
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

Example1:open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
    f.write("Woops! I have deleted the content!")
    f.close()

    #open and read the file after the appending:
    f = open("demofile3.txt", "r")
    print(f.read())
```


```
C:\Users\My Name>python demo_file_write.py
Woops! I have deleted the content!
```

- **Python-delete file:**

To delete a file, you must import the OS module, and run it’s os.remove() function.

Example:
```
import os
 os.remove(“demofile.txt”)
```

Check if File Exist:

Check if file exist, then delete it:

```
import os
if os.path.exists(“demofile.txt”):
    os.remove(demofile.txt”)
 else:
    print(“The file does not  exist”)
```

To delete an entire folder, use the **os.rmdir()** method:

Example:

```
import os
os.rmdir("myfolder")
```

Web Scraping

- **Why Web Scraping?**

Web scraping is used to collect large information from websites. But why does someone have to collect such large data from websites? To know about this, let's look at the applications of web scraping:

- **Price Comparison:** Services such as ParseHub use web scraping to collect data from online shopping websites and use it to compare the prices of products.
- **Email address gathering:** Many companies that use email as a medium for marketing, use web scraping to collect email ID and then send bulk emails.
- **Social Media Scraping:** Web scraping is used to collect data from Social Media websites such as Twitter to find out what's trending.
- **Research and Development:** Web scraping is used to collect a large set of data (Statistics, General Information, Temperature, etc.) from websites, which are analyzed and used to carry out Surveys or for R&D.

- What is web scraping?

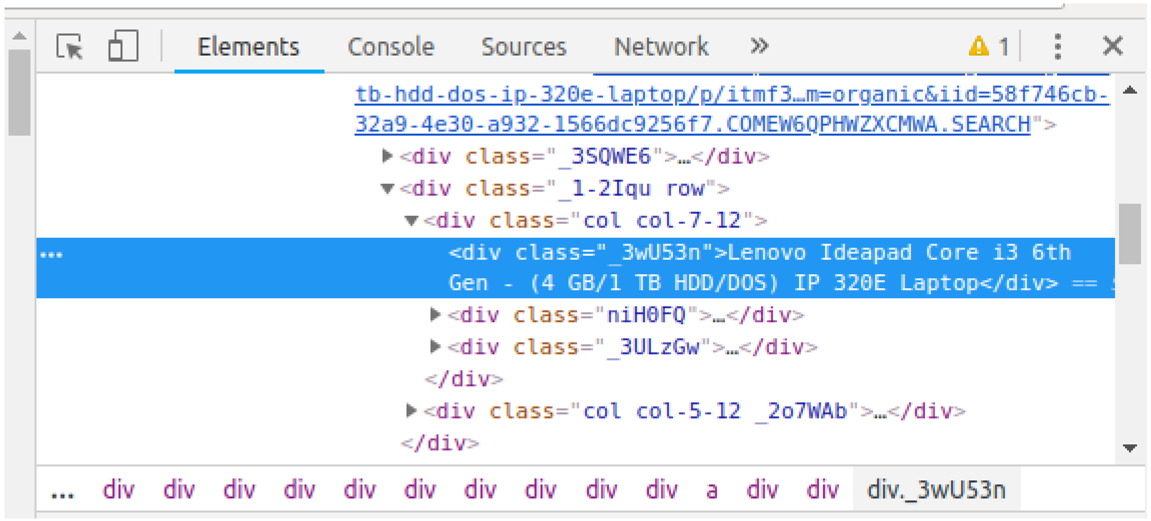
Web scraping is an automated method used to extract large amounts of data from websites. The data on the websites are unstructured. Web scraping helps collect these unstructured data and store it in a structured form.

How does web scraping works?

- To extract data using web scraping with python, you need to follow these basic steps:
- Find the URL that you want to scrape
- Inspecting the Page
- Find the data you want to extract
- Write the code
- Run the code and extract the data
- Store the data in the required format

- **Demo: Scraping Flipkart Website**
- **Step 1: Find the URL that you want to scrape**
- For this example, we are going scrape **Flipkart** website to extract the Price, Name, and Rating of Laptops. The URL for this page is https://www.flipkart.com/laptops/~buyback-guarantee-on-laptops-/pr?sid=6bo%2Cb5g&uniqBStoreParam1=val1&wid=11.productCard.PMU_V2.
- **Step 2: Inspecting the Page**
- The data is usually nested in tags. So, we inspect the page to see, under which tag the data we want to scrape is nested. To inspect the page, just right click on the element and click on “Inspect”.

- When you click on the “Inspect” tab, you will see a “Browser Inspector Box” open.



```
            tb-hdd-dos-ip-320e-laptop/p/itmf3…m=organic&iid=58f746cb-
            32a9-4e30-a932-1566dc9256f7.COMEW6QPHWZXCMWA.SEARCH">
  ▶<div class="_3SQWE6">…</div>
  ▼<div class="_1-2Iqu row">
    ▼<div class="col col-7-12">
        <div class="_3wU53n">Lenovo Ideapad Core i3 6th
        Gen - (4 GB/1 TB HDD/DOS) IP 320E Laptop</div> ==
      ▶<div class="niH0FQ">…</div>
      ▶<div class="_3ULzGw">…</div>
     </div>
    ▶<div class="col col-5-12 _2o7WAb">…</div>
   </div>
… div div div div div div div div div div a div div div._3wU53n
```

- **Step 3: Find the data you want to extract**
- Let’s extract the Price, Name, and Rating which is nested in the “div” tag respectively.
-

- **Step 4: Write the code**
- First, let’s create a Python file. To do this, open the terminal in Ubuntu and type gedit <your file name> with .py extension.
- I am going to name my file “web-s”. Here’s the command:

- **gedit web-s.py**
- Now, let’s write our code in this file.
- First, let us import all the necessary libraries:

```
from selenium import webdriver
 from BeautifulSoup import BeautifulSoup
import pandas as pd
```

- To configure webdriver to use Chrome browser, we have to set the path to chromedriver

```
driver = webdriver.Chrome("/usr/lib/chromium-browser/chromedriver")
```

```
products=[] #List to store name of the product
prices=[] #List to store price of the product
ratings=[] #List to store rating of the product
driver.get("https://www.flipkart.com/laptops/~buyback-guarantee-on-laptops-
    /pr?sid=6bo%2Cb5g&uniq"
content = driver.page_source
soup = BeautifulSoup(content)
for a in soup.findAll('a',href=True, attrs={'class':'_31qSD5'}):
name=a.find('div', attrs={'class':'_3wU53n'})
price=a.find('div', attrs={'class':'_1vC4OE _2rQ-NK'})
rating=a.find('div', attrs={'class':'hGSR34 _2beYZw'})
products.append(name.text)
prices.append(price.text)
ratings.append(rating.text)
```

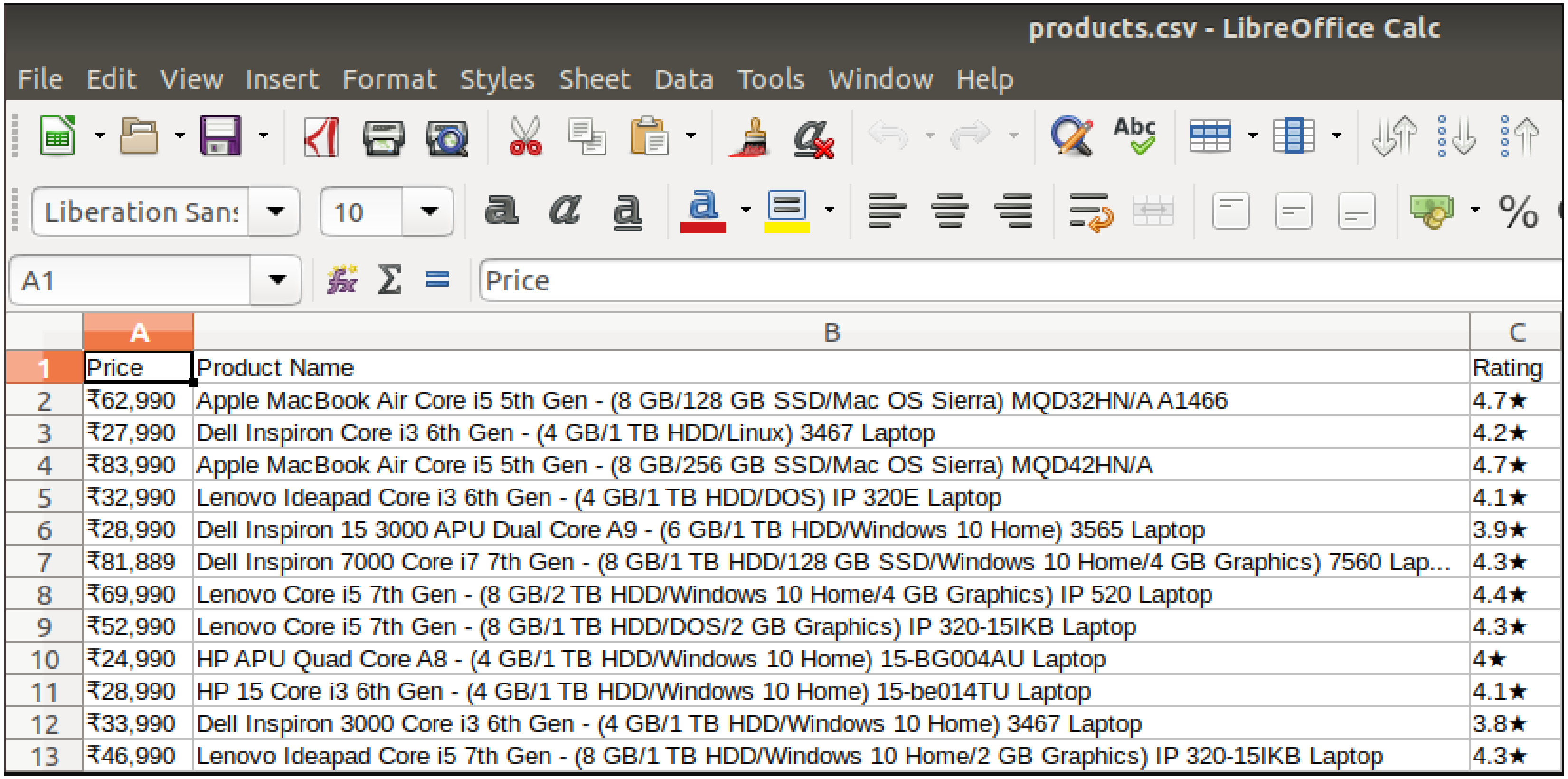
- **Step 5: Run the code and extract the data**
- To run the code, use the below command:

```
python web-s.py
```

- **Step 6: Store the data in a required format**
- After extracting the data, you might want to store it in a format. This format varies depending on your requirement. For this example, we will store the extracted data in a CSV (Comma Separated Value) format. To do this, I will add the following lines to my code:

```
df=pd.DataFrame({'ProductName':products,'Price':prices,'Ra
ting':ratings})
df.to_csv('products.csv', index=False, encoding='utf-8')
```

- Now, I'll run the whole code again.
- A file name "products.csv" is created and this file contains the extracted data.


products.csv - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools Window Help
Liberation Sans
10
A1
Price

	A	B	C
1	Price	Product Name	Rating
2	₹62,990	Apple MacBook Air Core i5 5th Gen - (8 GB/128 GB SSD/Mac OS Sierra) MQD32HN/A A1466	4.7★
3	₹27,990	Dell Inspiron Core i3 6th Gen - (4 GB/1 TB HDD/Linux) 3467 Laptop	4.2★
4	₹83,990	Apple MacBook Air Core i5 5th Gen - (8 GB/256 GB SSD/Mac OS Sierra) MQD42HN/A	4.7★
5	₹32,990	Lenovo Ideapad Core i3 6th Gen - (4 GB/1 TB HDD/DOS) IP 320E Laptop	4.1★
6	₹28,990	Dell Inspiron 15 3000 APU Dual Core A9 - (6 GB/1 TB HDD/Windows 10 Home) 3565 Laptop	3.9★
7	₹81,889	Dell Inspiron 7000 Core i7 7th Gen - (8 GB/1 TB HDD/128 GB SSD/Windows 10 Home/4 GB Graphics) 7560 Lap...	4.3★
8	₹69,990	Lenovo Core i5 7th Gen - (8 GB/2 TB HDD/Windows 10 Home/4 GB Graphics) IP 520 Laptop	4.4★
9	₹52,990	Lenovo Core i5 7th Gen - (8 GB/1 TB HDD/DOS/2 GB Graphics) IP 320-15IKB Laptop	4.3★
10	₹24,990	HP APU Quad Core A8 - (4 GB/1 TB HDD/Windows 10 Home) 15-BG004AU Laptop	4★
11	₹28,990	HP 15 Core i3 6th Gen - (4 GB/1 TB HDD/Windows 10 Home) 15-be014TU Laptop	4.1★
12	₹33,990	Dell Inspiron 3000 Core i3 6th Gen - (4 GB/1 TB HDD/Windows 10 Home) 3467 Laptop	3.8★
13	₹46,990	Lenovo Ideapad Core i5 7th Gen - (8 GB/1 TB HDD/Windows 10 Home/2 GB Graphics) IP 320-15IKB Laptop	4.3★