PYTHON LECTURE NOTES

INTRODUCTION

Python is one of the most popular programming languages. It delivers both the power and general applicability of traditional compiled languages with the ease of use of simpler scripting and interpreted languages. Python is developed by Guido van Rossum.

Applications of Python:

Python is used to develop

- ✓ Artificial Intelligence Applications
- ✓ Automation and Web Applications
- ✓ GUI (Graphical User Interface) Applications
- ✓ Scientific and Business Applications
- ✓ Console Based Applications
- ✓ Audio & Video Based Applications
- ✓ Image Processing & 3D CAD Applications
- ✓ Enterprise Applications

Features of Python:

The following are set of python features that describes about python.

High Level:

Python is a high level language with more powerful concepts like lists, tuples and dictionaries. With help of these concepts one can minimizes the development time as well as code size.

Object Oriented:

Object-oriented programming (OOP) adds another dimension to structured and procedural languages where data and logic are discrete elements of programming. OOP allows for associating specific behaviors, characteristics, and/or capabilities with the data that they execute on or are representative of. Python is an object-oriented (OO) language, all the way down to its core. However, Python is not just an OO language like Java or Ruby. It is actually a pleasant mix of multiple programming paradigms. For instance, it even borrows a few things from functional languages like Lisp and Haskell.

Scalable

Python is scalable where we can grow our code from project to project, add other new or existing Python elements, and reuse code.

Extensible

As the amount of Python code increases in your project, you will still be able to organize it logically by separating your code into multiple files, or modules, and be able to access code from one module and attributes from another.

Portable

Python can be found on a wide variety of systems, contributing to its continued rapid growth in today's computing domain. Because Python is written in C, and because of C's portability, Python is available on practically every type of platform that has an ANSI C compiler.

Easy to Learn

Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows anyone to pick up the language in a relatively short period of time.

Easy to Read

Python making it easier for others to understand your code faster and vice versa. Because of its syntax and styles.

Easy to Maintain

Maintaining source code is part of the software development lifecycle. Much of Python's success is that source code is fairly easy to maintain, dependent, of course, on size and complexity.

Robust

Python even gives you the ability to monitor for errors and take an evasive course of action if such an error does occur during runtime. Python's robustness is beneficial for both the software designer and the user. There is also some accountability when certain errors occur that are not handled properly. The stack trace that is generated as a result of an error reveals not only the type and location of the error, but also in which module the erroneous code resides.

Where can I get the python Installer file ?

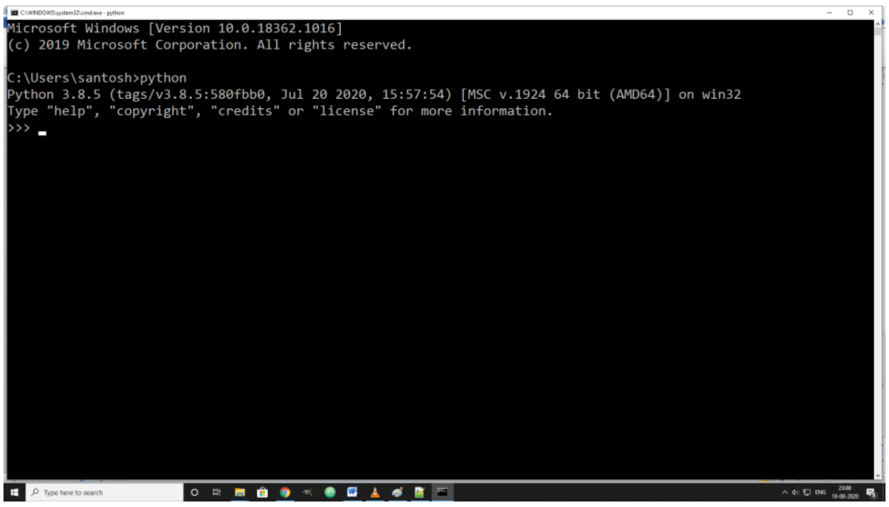
Visit python official website link

https://www.python.org/downloads/windows/ for windows and
https://www.python.org/downloads/mac-osx/ for mac and
https://www.python.org/download/other/ for other platforms.

Now download the recent installer that matches your operating system version.Now install python with the help of the file that you just downloaded.

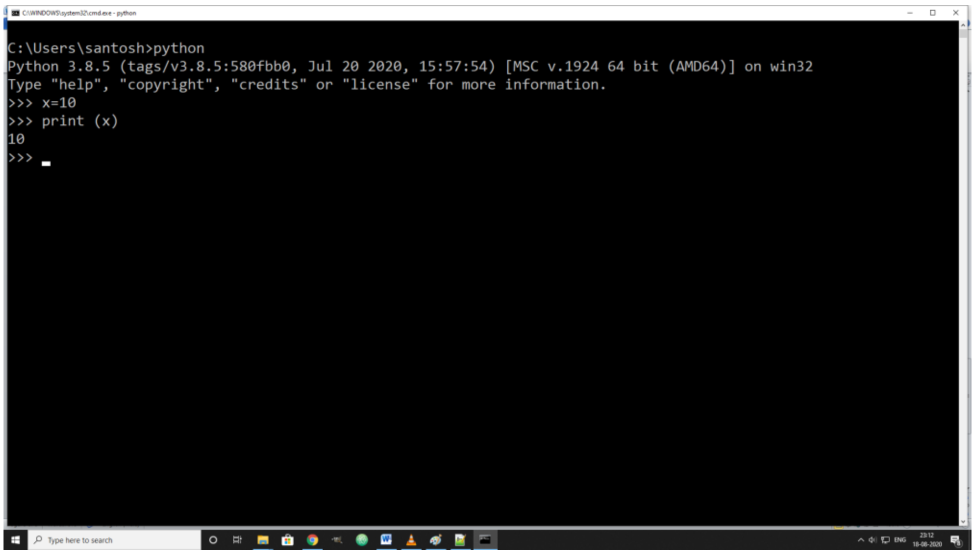
Running Python

We can run python programs in command prompt itself by using python command.



```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\santosh>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

In above screenshot >>> is called as python prompt where python programs will be typed and executed.

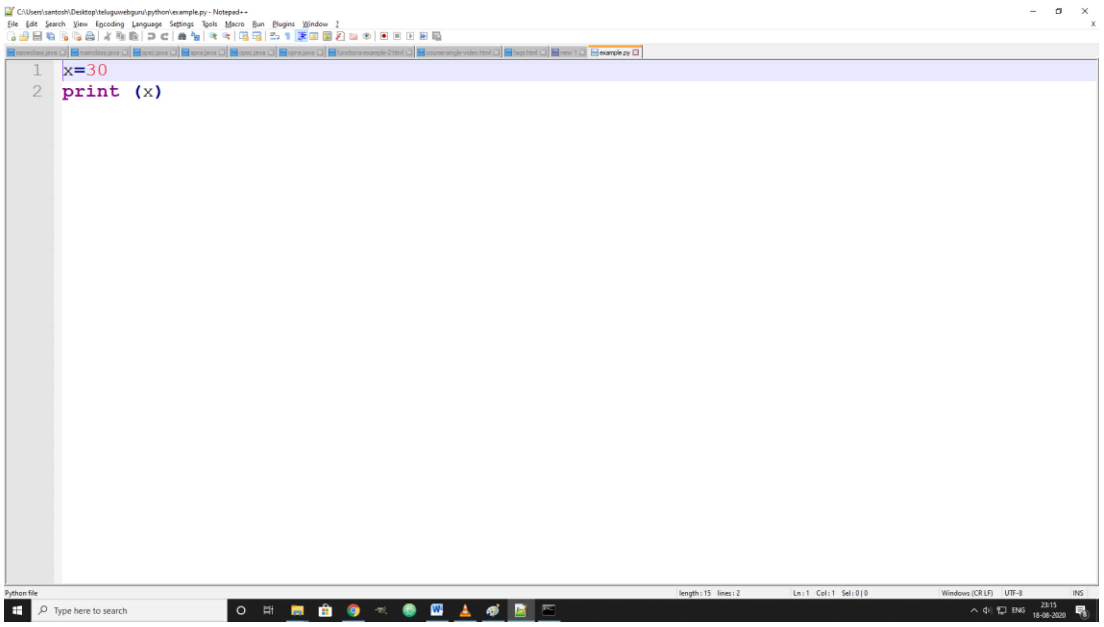


```
C:\Users\santosh>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x=10
>>> print (x)
10
>>>
```

For example in the above pic, a small program is written in which a variable x is initialized with the value 10 and it is printed.

It is possible to type and save our python programs through text editors like notepad and we can execute the program with python command.

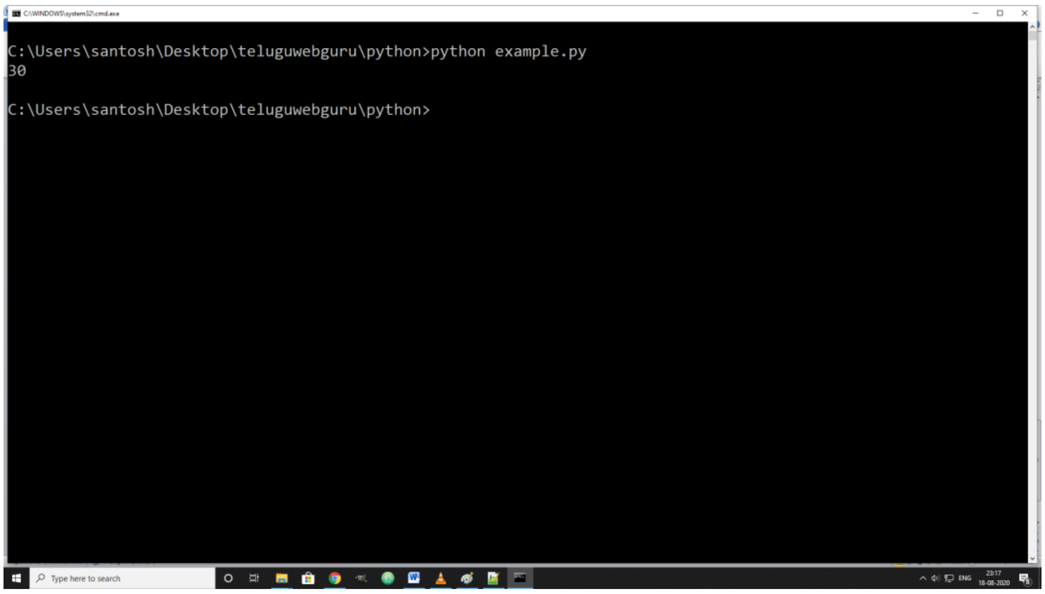
Example:



```
x=30
print (x)
```

For example a small python program is typed using text editor and saved it as example.py

Now we can execute the same program by using python command in command prompt as follows.



```
C:\Users\santosh\Desktop\teluguwebguru\python>python example.py
30

C:\Users\santosh\Desktop\teluguwebguru\python>
```

How to read input from user in python?

In python we can read input from user by using a method “input”. Syntax is as follows.

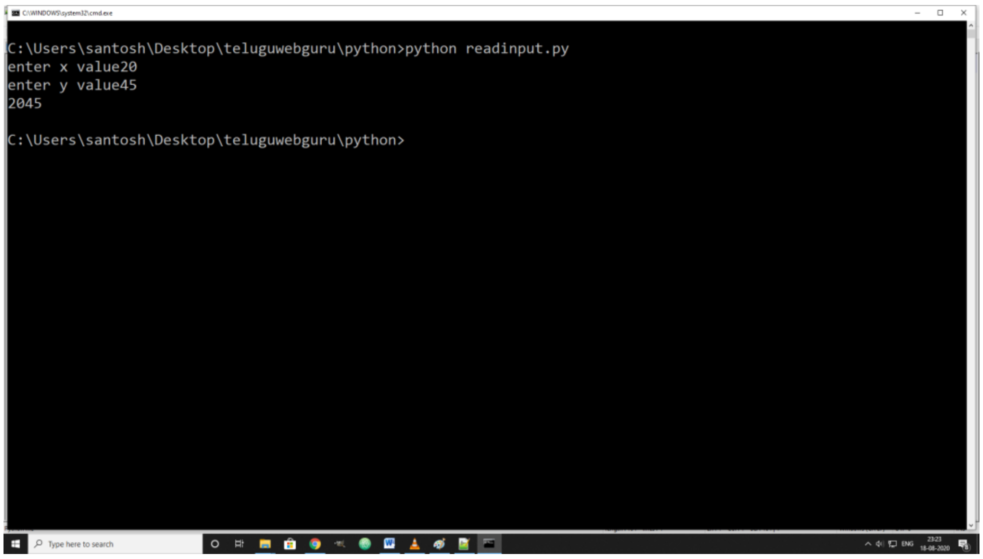
Syntax:

input(“message”)

Example:

```
#program to read input from user
x = input("enter x value");
y = input("enter y value");
print (x+y);
```

Output



```
C:\Users\santosh\Desktop\teluguwebguru\python>python readinput.py
enter x value20
enter y value45
2045

C:\Users\santosh\Desktop\teluguwebguru\python>
```

In above program both the values are concatenated because “input” method always reads the user inputs as strings. We must convert those values into numbers before performing arithmetic operations. These concepts will be explained after datatypes.

PYTHON SYNTAX & STYLE

i) Blocks are represented by indentation instead of curly braces.

For example while writing a block with if condition (or any type of block), please observe that curly braces are not used to indicate starting and ending of the block. Instead of that indentation is used. All the statements in that block are maintained the same indentation (spaces).

Example:

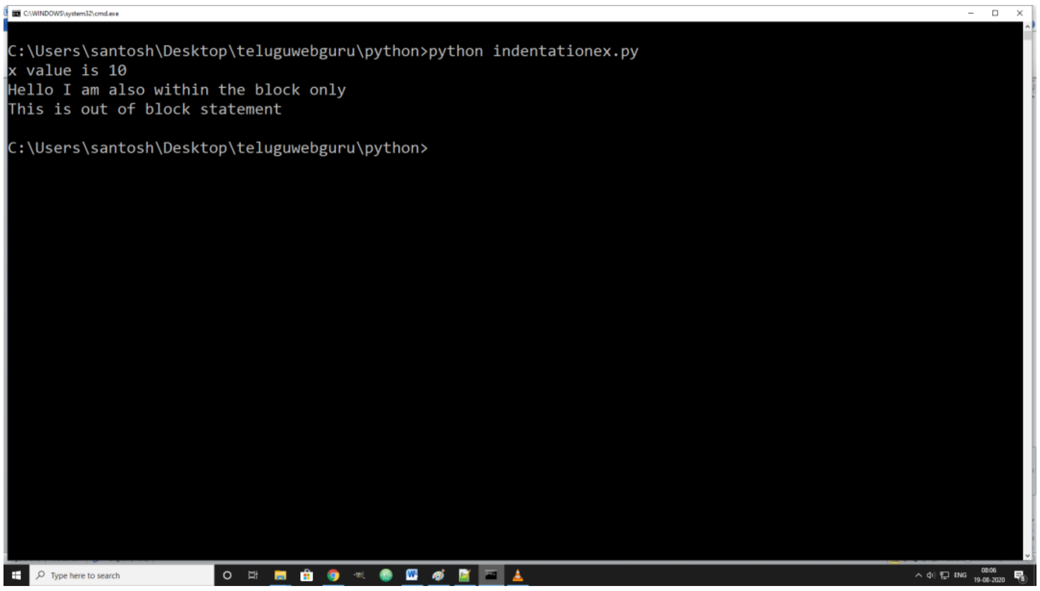
```
x = 10

if(x==10) :

    print ("x value is 10")

    print ("Hello I am also within the block only");

print ("This is out of block statement");
```

In the above example colon (:) indicates the starting of the block. All the statements in the block are maintain the same indentation (at least one space - generally 4 or 8 spaces to improve understanding ability but not mandatory).

First 2 print statements maintained the same indentation (spaces at starting position) after colon that indicates these two statements are belonged to the ‘ if ’ block.

Third print statement is written without indentation means it is not related to that block.

<u>output</u>



```
C:\Users\santosh\Desktop\teluguwebguru\python>python indentationex.py
x value is 10
Hello I am also within the block only
This is out of block statement

C:\Users\santosh\Desktop\teluguwebguru\python>
```

ii) How to write comments in python ?

We can write comments by using the symbol hash (#).

Example

```
#This is a comment

x = 10 #this comment starts from here
```

Generally writing comments is good programming habit. It will improve program understand ability and we can temporarily suspend executing some statements by putting them in comments.

iii) Continuation (\ ) operator

While developing projects, there are some cases where a single statement may occupies more than one line. In that case if we use continuation operator then python interpreter can understand and treat those multiple line statement as single statement.

Example

```
x = “hello my channel name is ” +\

“Telugu Web Guru”
```

In the above example \ is placed at the end of first statement that indicates next line also belonged to this statement only.

iv) Suites or Blocks

As mentioned in above examples, blocks are not represented by curly braces in python. Instead of it colon and indentation are used. Colon indicates the starting of a block and indentation indicates the statements that are belonged to the particular block.

Example

```
If(condition):

    Statement1

    Statement2
```

Consider the above block of statements. Please observe we did not use any open curly brace to indicate starting of the block instead we used colon (:) symbol which indicates some block is started at this point. All the statements related to this block are written with some indentation.

v) Use of Semicolons (;)

In python semicolons are not mandatory to use at the end of each and every statement. If more than one statement is written in the same line then we will use semicolons to separate one from the other.

Example

```
x=10;y=20
```

vi) Variable assignments

In python there are no keywords for datatypes (like int, float etc.,). Variables data types are decided based on the value assigned to that variable. This type of languages are called as dynamic typed languages (data type will be decided at runtime based on values).

Example :

```
a=10       #integer
b=1.25     #float
c=1+6j     #complex

#multiple assignment with same value
a = b = c = 10

#multiple assignment with different values
a,b,c = 1,2,3
```

As shown in above example we can assign multiple variables with same value or multiple variables with multiple values.

Memory Management:

Python supports memory management automatically. Here we no need to allocate memory or deallocate memory in manual way. In python we no need to declare the variables with datatypes. Whenever values are assigned to the variables, corresponding data types will be decided and memory also allocated according to the type. If you want to deallocate memory manually we can do that with 'del' statement.

Python deallocate the variables from memory, which are not in use. This process is called as Garbage Collection.

Python Objects:

As python is object oriented language, every construct (variables etc.,) will be considered as an object.

Every object has three different characteristics.

- Identifier : unique identifier to identify each object.
- Type : To indicate which type of value it stores
- Value : value of the object.

NUMBERS IN PYTHON

Python supports following types of numbers.

- Integers (Plain Integers, Long)
- Floating Point Real Numbers
- Complex Numbers

Python numbers are immutable. That means new memory locations are allocated on changing the values of numbers every time. Once a value is assigned in a memory location we can't change the value.

<u>Plain Integers :</u>

It is universal integer type. We can assign decimal or octal or hexadecimal values to the variables.

Example :

```
X = 10          #decimals

X = 0           #values started with 0 are octal numbers (base 8)

X = 0x or 0X    #values started with 0x or 0X are hexadecimals (base 16)
```

<u>long :</u>

Whenever we want to work with large numbers then long type will be preferred and there will be no ranges for the datatypes. The limit is virtual memory limit only. That means we can store a long value as big as it occupies our full virtual memory space.

<u>Floating point real numbers :</u>

This can store double precision decimal values. In this we can store normal decimal point values or exponent values also.

Example:

```
X = 1.25

Y = 1.45E+12

Z = float(10)
```

<u>Complex Numbers :</u>

We can assign complex numbers to the python variables.

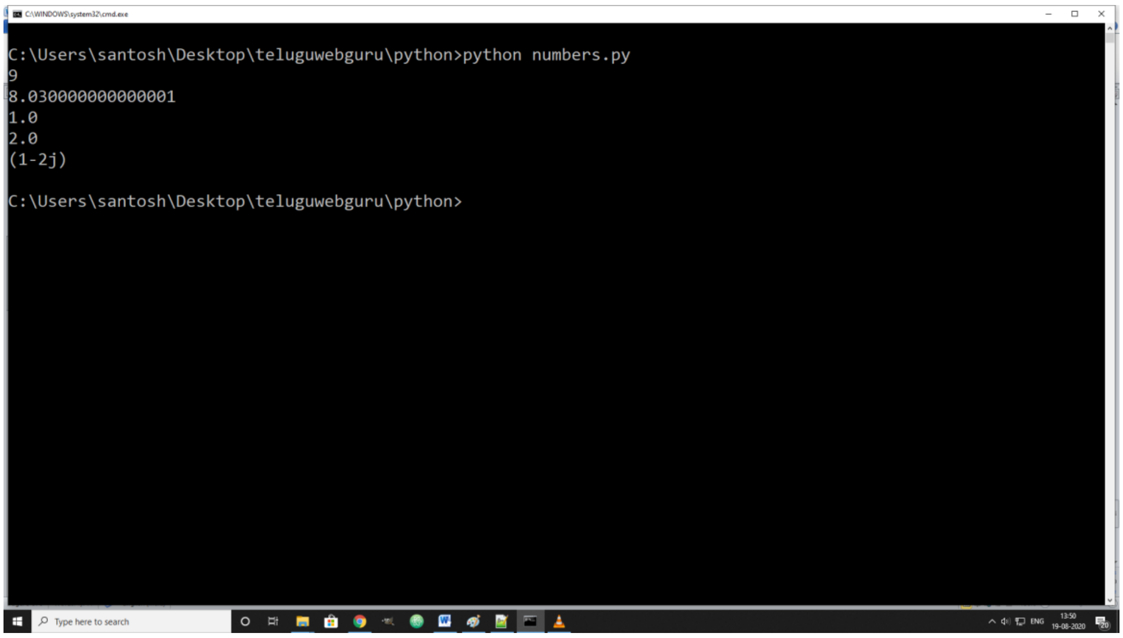
Example:

```
x = 1+2j

print(x.real)       #retrieves the real part

print(x.imag)       #retrieves the imaginary part

print(x.conjugate) #retrieves the complex conjugate of a complex number
```

Full Example Program with all the numbers:

```
#program to demonstrate numbers in python

#integers
x=4
y=5
print (x+y)

#floating point real numbers
a=1.25
b=6.78
print (a+b)

#complex numbers
x=1+2j
print(x.real)
print(x.imag)
print(x.conjugate())
```

Output



```
C:\Users\santosh\Desktop\teluguwebguru\python>python numbers.py
9
8.030000000000001
1.0
2.0
(1-2j)

C:\Users\santosh\Desktop\teluguwebguru\python>
```

CONTROL STATEMENTS IN PYTHON

Control statements in python are quite different in syntax when we compared with other languages.

If condition :

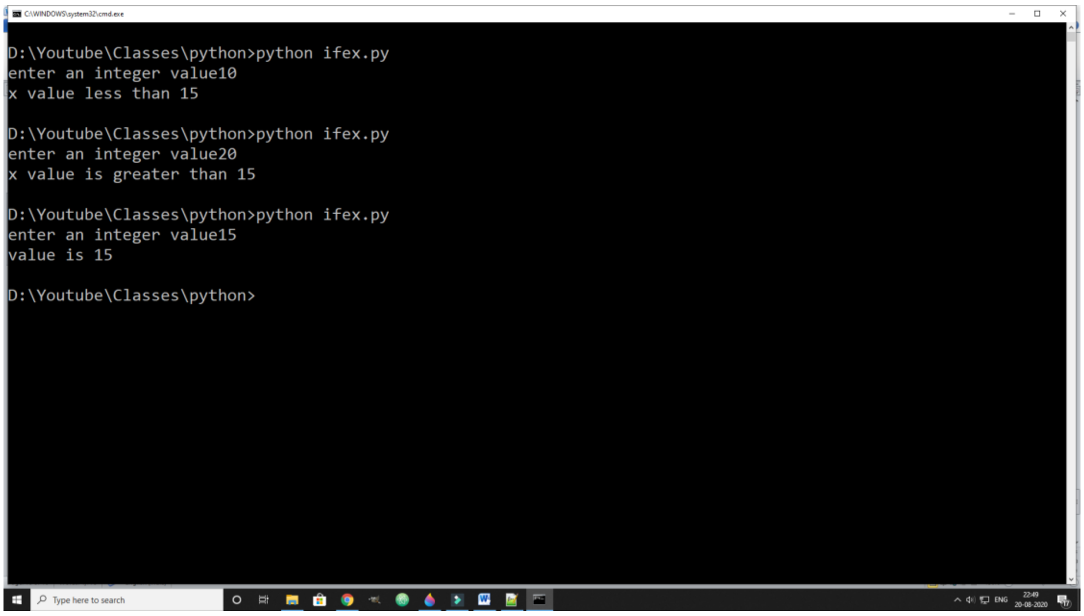
If condition syntax is as follows.

```
If condition:
   Statement1
   Statement2
elif condittion2:
   Statement1
   Statement2
else:
   Statements
```

Example :

```
#program to demonstrate if condition
x=int(input("enter an integer value"))

if x==15:
   print("value is 15")
elif x<15:
   print("x value less than 15");
else:
   print("x value is greater than 15")
```

Output



```
D:\Youtube\Classes\python>python ifex.py
enter an integer value10
x value less than 15

D:\Youtube\Classes\python>python ifex.py
enter an integer value20
x value is greater than 15

D:\Youtube\Classes\python>python ifex.py
enter an integer value15
value is 15

D:\Youtube\Classes\python>
```

for loop :

for loop in python uses range function. range(5) indicates 0 to 4. range(5,10) indicates 5 to 9. range(0,10,3) indicates 0,3,6,9 (it starts from 0 and increments its value by 3 every time until the value reaches 10).

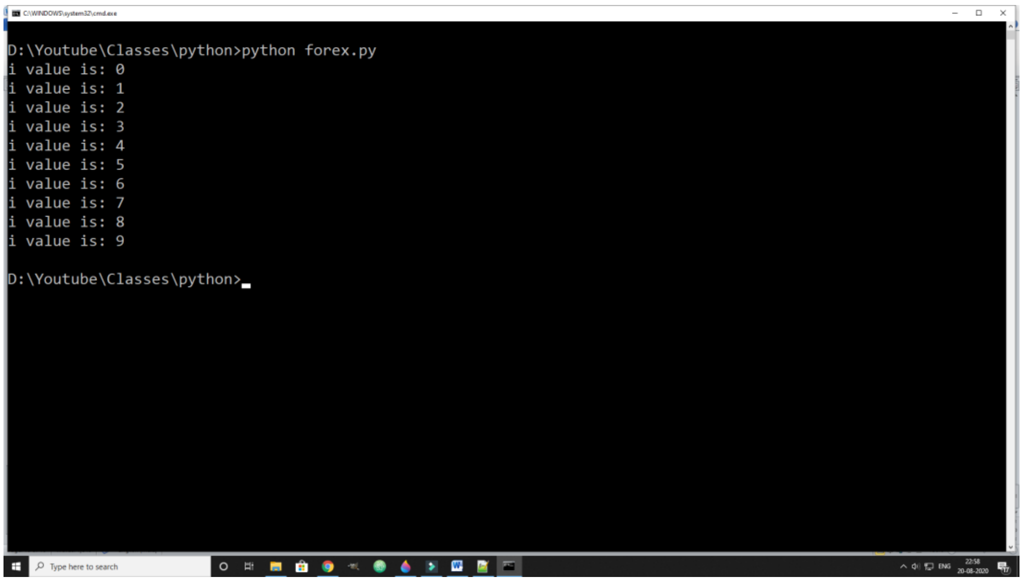
Syntax:

```
        for i in range(n):
            statements
```

Example:

```
#program to demonstrate for loop in python
for i in range(10):
   print("i value is:",i)
```

Output:

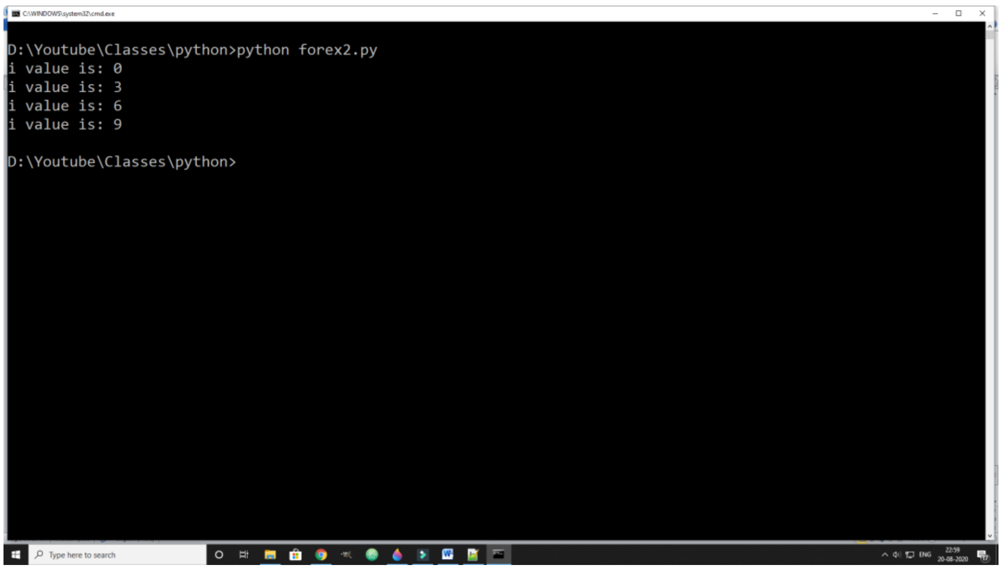


```
D:\Youtube\Classes\python>python forex.py
i value is: 0
i value is: 1
i value is: 2
i value is: 3
i value is: 4
i value is: 5
i value is: 6
i value is: 7
i value is: 8
i value is: 9

D:\Youtube\Classes\python>
```

Example2:

```
#program to demonstrate for loop in python
for i in range(0,10,3):
   print("i value is:",i)
```

Output



```
D:\Youtube\Classes\python>python forex2.py
i value is: 0
i value is: 3
i value is: 6
i value is: 9

D:\Youtube\Classes\python>
```

We can run a for loop on lists also

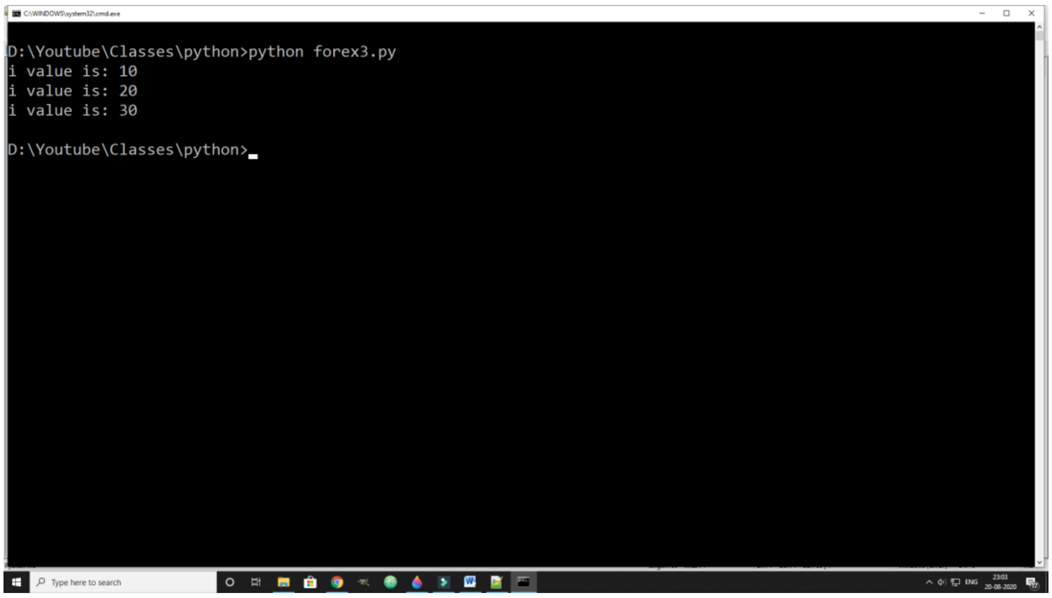
Syntax:

```
        for i in listname:
                statements;
```

Example 3:

```
#program to demonstrate for loop in python
s = [10,20,30]
for i in s:
   print("i value is:",i)
```

Output



```
D:\Youtube\Classes\python>python forex3.py
i value is: 10
i value is: 20
i value is: 30

D:\Youtube\Classes\python>
```

break statement:

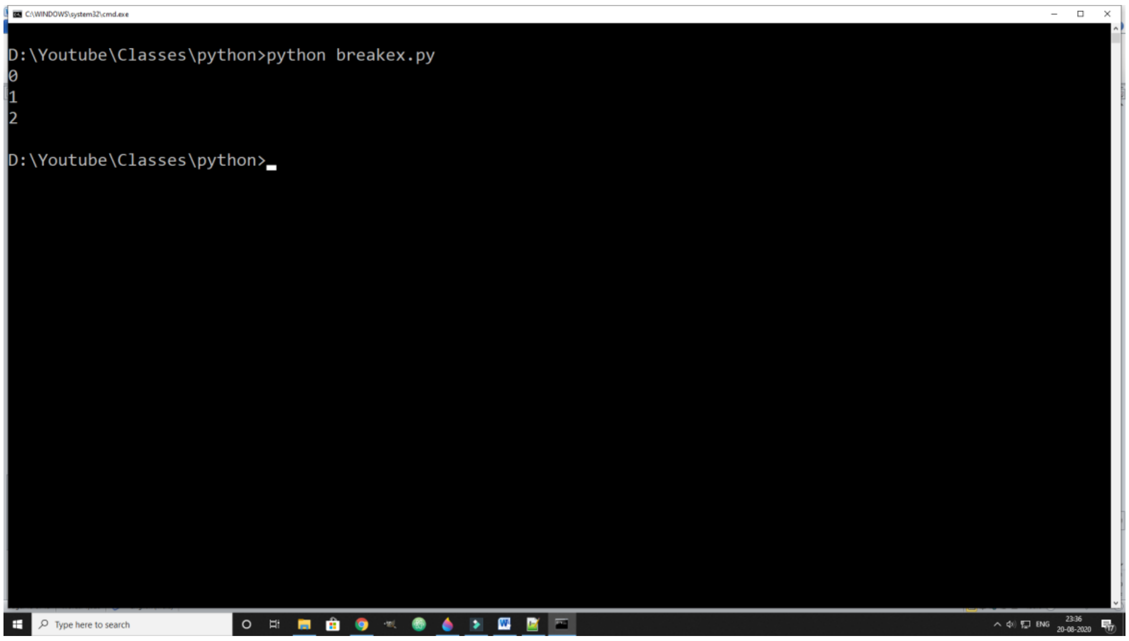
break statement terminates the control from the block.

Example

```
#program to demonstrate break statement
for i in range(10):
  if i==3:
    break
  print (i)
```

As the below output shows whenever condition met control terminated from the loop. That is why we are unable to see the I values after 3.

Output:



```
D:\Youtube\Classes\python>python breakex.py
0
1
2

D:\Youtube\Classes\python>_
```

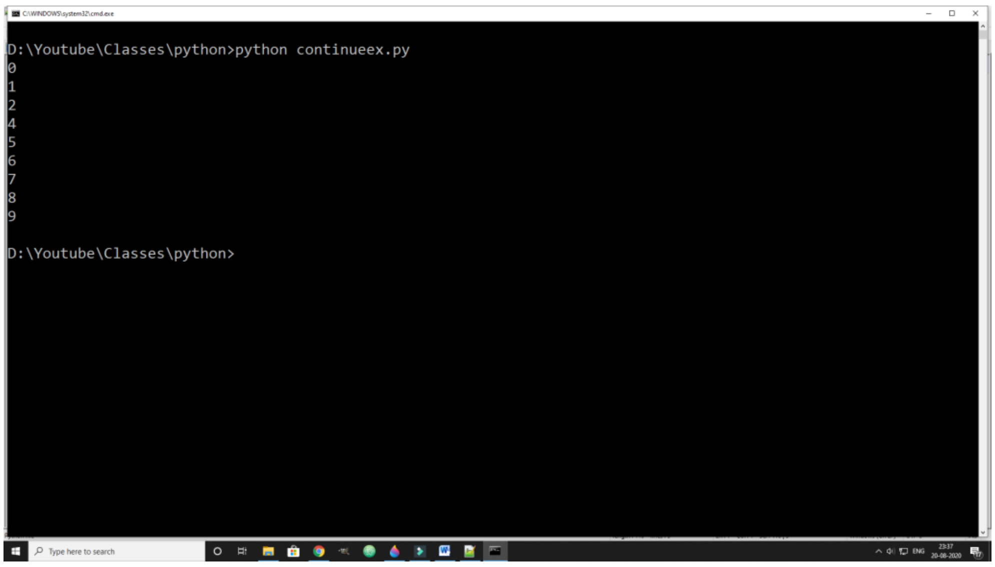
continue statement:

continue statement skips the current iteration of the block.

Example:

```
#program to demonstrate continue statement
for i in range(10):
  if i==3:
    continue
  print (i)
```

Output:



```
D:\Youtube\Classes\python>python continueex.py
0
1
2
4
5
6
7
8
9

D:\Youtube\Classes\python>
```

As output show continue will skip the only iteration that meets the condition. That is why only printing the value 3 is skipped and loop continued from i=4.

pass statement :

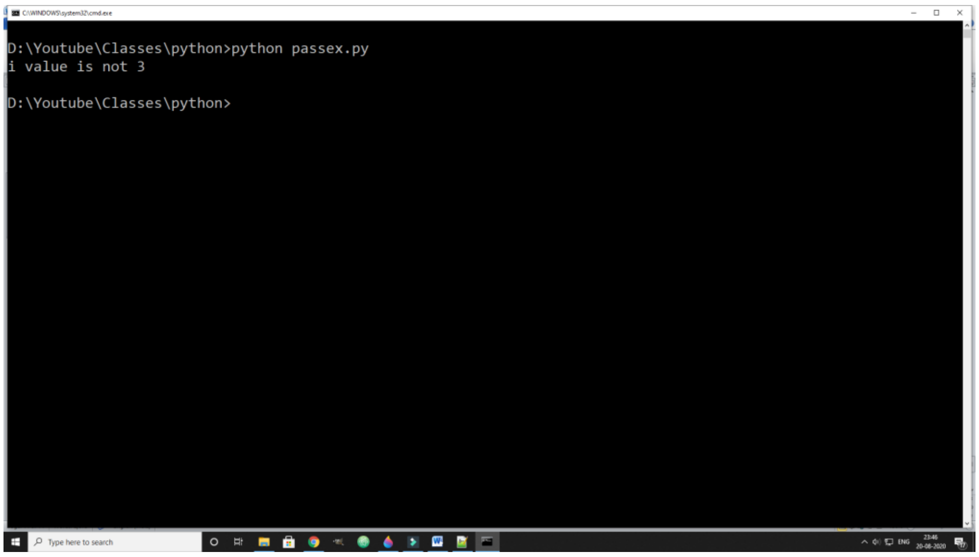
Generally pass statement do nothing. This statement will be useful in some blocks that insists us to write at least one statement but programmatically we don't want to write any statement including print statement.

For example consider the following example where we want to print some value in all cases except I value is 3;

Example:

```
#program to demonstrate pass statement
i=10
if i==3:
   pass
else:
   print("i value is not 3")
```

Output



```
D:\Youtube\Classes\python>python passex.py
i value is not 3

D:\Youtube\Classes\python>
```

Consider above example. I want to print I value in all cases except I value is 3. But whenever we write if condition if expects at least one statement. So we wrote the pass statement here. So whenever i value is 3 then it does nothing. In all other cases in prints the i value.

SEQUENCES IN PYTHON

Sequences allow us to store elements in a particular order and we can access those elements by using their index.

We have three types of sequences in python

1) Strings
2) Lists
3) Tuples

<u>Strings:</u>

Collection of characters is said to be a string. In python strings are case sensitive. So the variable ‘s’ is different from variable ‘S’.

s = “ Telugu Web Guru ”

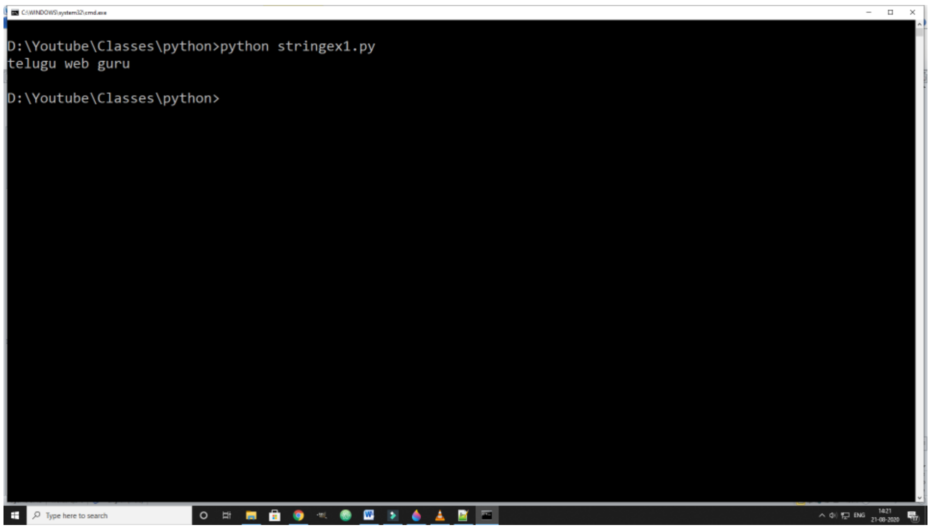
s = ‘ Telugu Web Guru ‘

We can represent strings by using single or double quotes.

Example:

```
#program to demonstrate strings in python
cname="telugu web guru";
myname="santosh raju dabbiru";
print (cname);
```

Output



```
D:\Youtube\Classes\python>python stringex1.py
telugu web guru

D:\Youtube\Classes\python>
```

In python strings are immutable. That means we can’t change the value of a memory location related to an string object. Every time we change the value, then a new memory location is allotted and that new reference is pointed by the string object.

We can clear the contents of the string as follows.

cname=""

We can remove the string from memory by using 'del' as follows.

del cname;

We can access strings by using index as below

cname="My Channel name is Telugu Web Guru"

We can retrieve the character at index 0 by using cname[0].

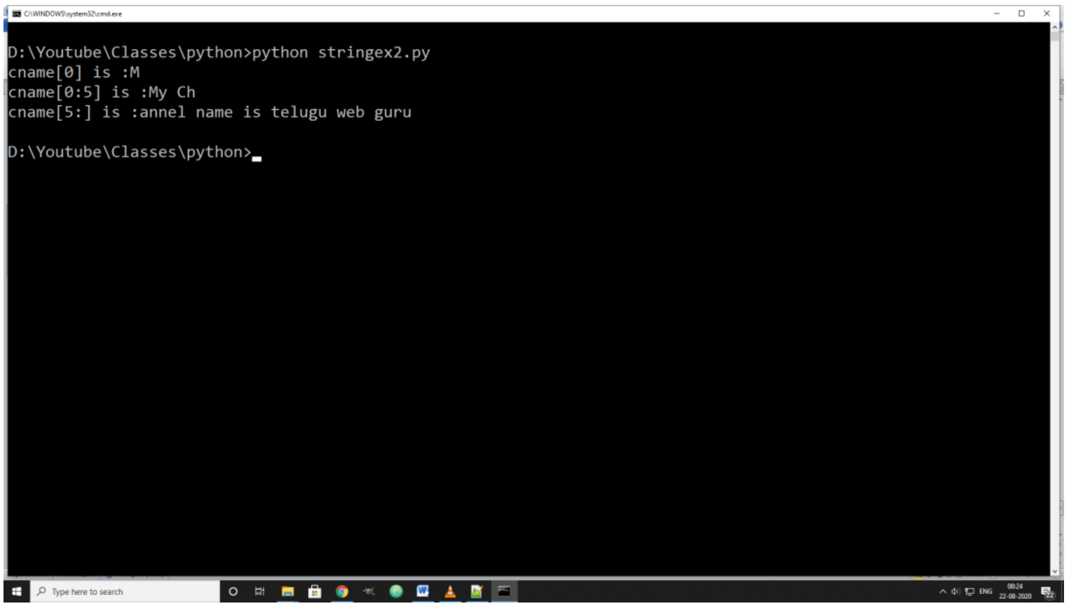
We can retrieve the substring between index 0 to 4 as cname[0:5] where 0 is inclusive and 5 is exclusive

cname[5:] returns substring starts from index 5.

Example:

```
#program to demonstrate strings in python
cname="My Channel name is telugu web guru";

print ("cname[0] is :"+cname[0])
print ("cname[0:5] is :"+cname[0:5])
print ("cname[5:] is :"+cname[5:])
```

Output:



```
D:\Youtube\Classes\python>python stringex2.py
cname[0] is :M
cname[0:5] is :My Ch
cname[5:] is :annel name is telugu web guru

D:\Youtube\Classes\python>
```

As shown in above example, string indexing starts from 0. And in the same way negative indexing (from -1) starts from right to left as shown below.

s = "abcd"

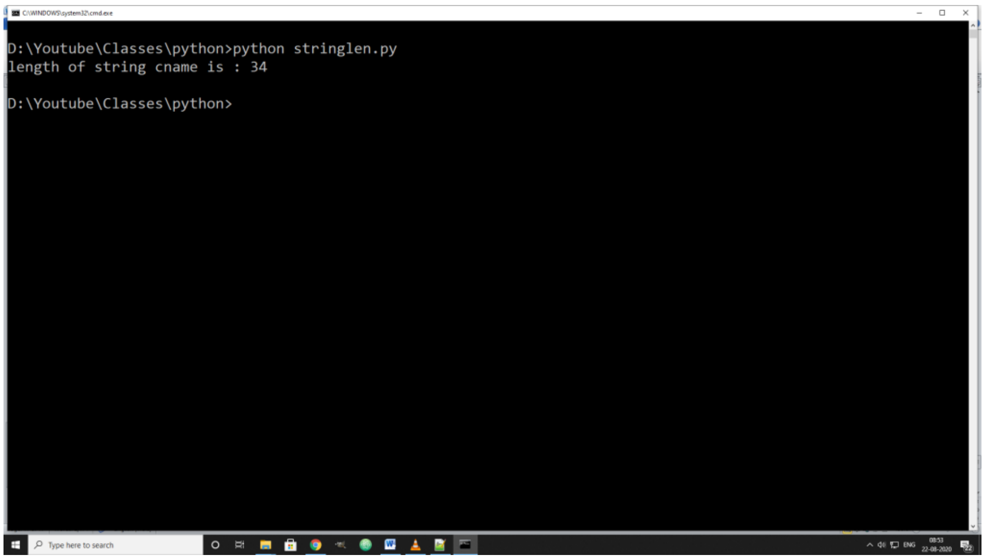
Indexing is as show below.

```
        -4  -3  -2  -1   Negative indexing

s =      a   b   c   d

         0   1   2   3   Positive indexing
```

len method : We can calculate length of a string in python by using len method as follows.

```
#program to demonstrate len method in python
cname="My Channel name is telugu web guru";

print ("length of string cname is :",len(cname))
```

output:



```
D:\Youtube\Classes\python>python stringlen.py
length of string cname is : 34

D:\Youtube\Classes\python>
```

max(string) :

max method returns the character with highest ASCII value.

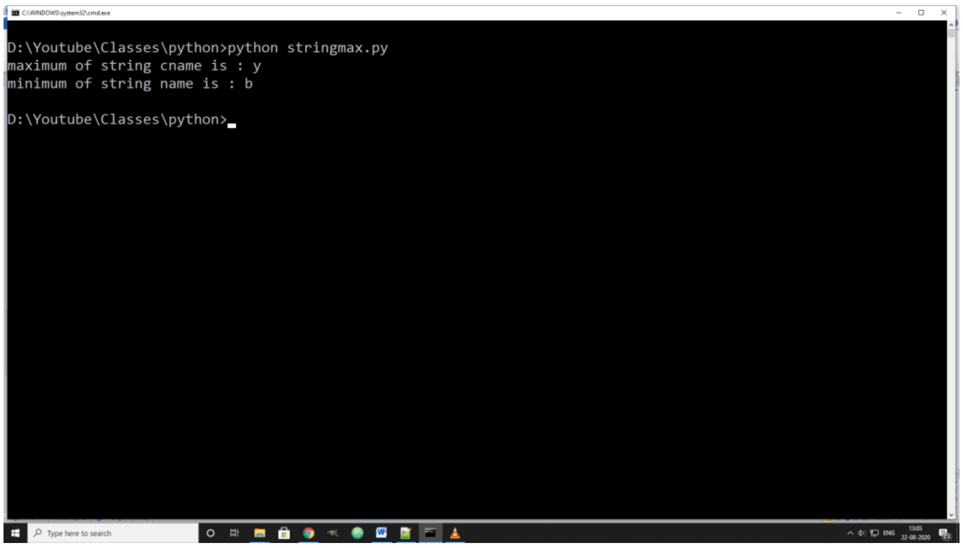
min(string) :

min method returns the character with small ASCII value.

example

```
#program to demonstrate max method in python
cname="My Channel name is telugu web guru";
print ("maximum of string cname is :",max(cname))

name="teluguwebguru";
print ("minimum of string name is :",min(name))
```

Output:



```
D:\Youtube\Classes\python>python stringmax.py
maximum of string cname is : y
minimum of string name is : b

D:\Youtube\Classes\python>_
```

As character y has highest ASCII value among others in cname and b has small ASCII in name, these two were returned.

Some Builtin Functions in strings :

string.capitalize()

Converts the first character to upper case

string.center(width)

Returns a centered string and sets the total length of the string to given width.

endswith(substr,beg=0,end=len(string))

Returns true if the string ends with the specified value

find(substr,beg,end)

Searches the string for a specified value and returns the position of where it was found

isalnum()

Returns True if all characters in the string are alphanumeric

isalpha()

Returns True if all characters in the string are in the alphabet

isdecimal()

Returns True if all characters in the string are decimals

isdigit()

Returns True if all characters in the string are digits

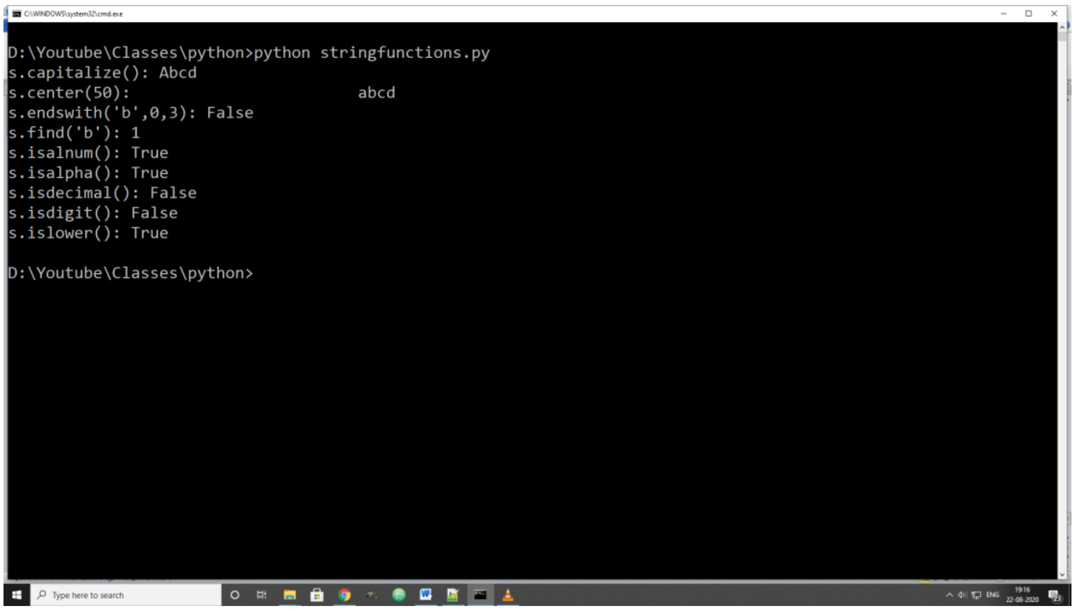
islower()

Returns True if all characters in the string are lower case

Example program :

```
#program to demonstarte string builtin functions
s="abcd"
print ("s.capitalize():",s.capitalize())
print("s.center(50):",s.center(50))
print("s.endswith('b',0,3):",s.endswith('b',0,3))
print("s.find('b'):",s.find('b'))
print("s.isalnum():",s.isalnum())
print("s.isalpha():",s.isalpha())
print("s.isdecimal():",s.isdecimal())
print("s.isdigit():",s.isdigit())
print("s.islower():",s.islower())
```

Output:



```
D:\Youtube\Classes\python>python stringfunctions.py
s.capitalize(): Abcd
s.center(50):                        abcd
s.endswith('b',0,3): False
s.find('b'): 1
s.isalnum(): True
s.isalpha(): True
s.isdecimal(): False
s.isdigit(): False
s.islower(): True

D:\Youtube\Classes\python>
```

Lists:

List is a collection which is ordered and changeable. Lists always allow storing duplicate elements. Lists are also one type of sequences.

One more important point is list can store set of objects which are of different type. That means we can store values that are belonged to different data types in the same list. We can insert, remove elements from the list as we want.

How to create a List?

We can create the list by using square braces [ ] as follows.

s = [1,2,3,4,5,"Telugu Web Guru"]

A list can also contain another list as elements

s= [1,2,[3.5,4.2]]

How to retrieve elements from list?

We can retrieve the elements from a list by using index as follows.

s[0]	-	Returns first element from the list
s[2][0]	-	Returns 3.5 which is first element(0th index) of inner list in the main list item 3(2nd index)
s[0:3]	-	Returns elements at index from 0 to 2

Update elements in a list?

we can update elements by assigning new value in place of existing one with the help of index as follows.

s[1] = 10

Above statement will replaces the element at index 1 in list s with the value 10 so new list becomes

s = [1,10,[3.5,4.2]]

How to remove an element from the list?

We can remove elements from the list using 'del' keyword.

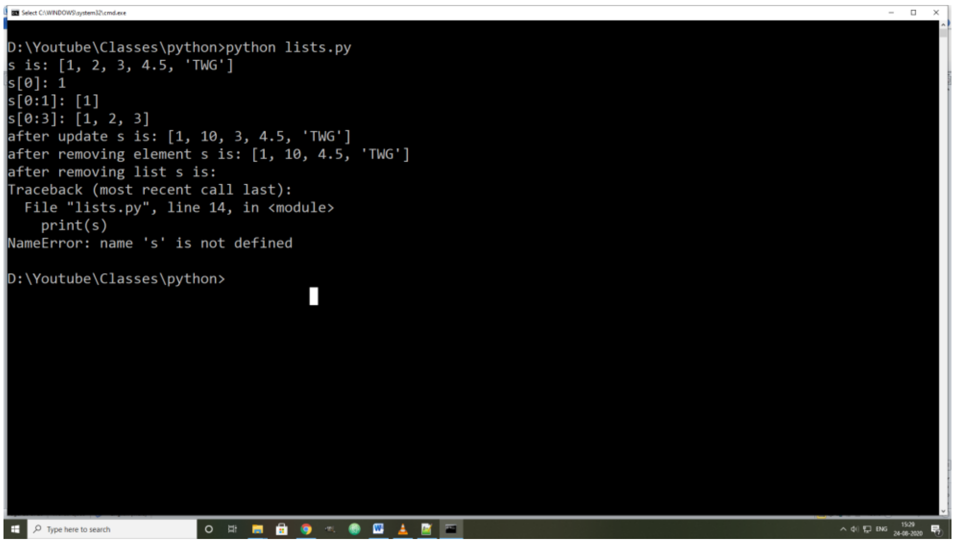
del s[2]

Above statement removes 3rd element (element at index 2) from the list.

Example:

```
#program to demonstrate lists in python
s=[1,2,3,4.5,"TWG"]
print("s is:",s)
print("s[0]:",s[0])
print("s[0:1]:",s[0:1])
print("s[0:3]:",s[0:3])
s[1]=10
print("after update s is:",s)
del s[2]
print("after removing element s is:",s)
del s
print("after removing list s is:")
print(s)
```

Output:



```
D:\Youtube\Classes\python>python lists.py
s is: [1, 2, 3, 4.5, 'TWG']
s[0]: 1
s[0:1]: [1]
s[0:3]: [1, 2, 3]
after update s is: [1, 10, 3, 4.5, 'TWG']
after removing element s is: [1, 10, 4.5, 'TWG']
after removing list s is:
Traceback (most recent call last):
  File "lists.py", line 14, in <module>
    print(s)
NameError: name 's' is not defined

D:\Youtube\Classes\python>
```

In above output last print statement returns error because we removed entire list by using del command and then we are trying to print the list which does not exists. so it returns not defined error.

Operators on List:

We can apply different operators on lists to check where two lists are equal or not, whether one list is greater than or less than the other list or not etc.,

==	operator is used to check whether two lists are equal or not
>	operator is used to check whether the first list is greater than second list or not]
<	operator is used to check whether the first list is less than the second list or not

Membership operators (in, not in): These are used to check whether an item exists in list or not. in checks whether the specified list item exists in the list or not, not in checks whether the list item does not exists in list or not

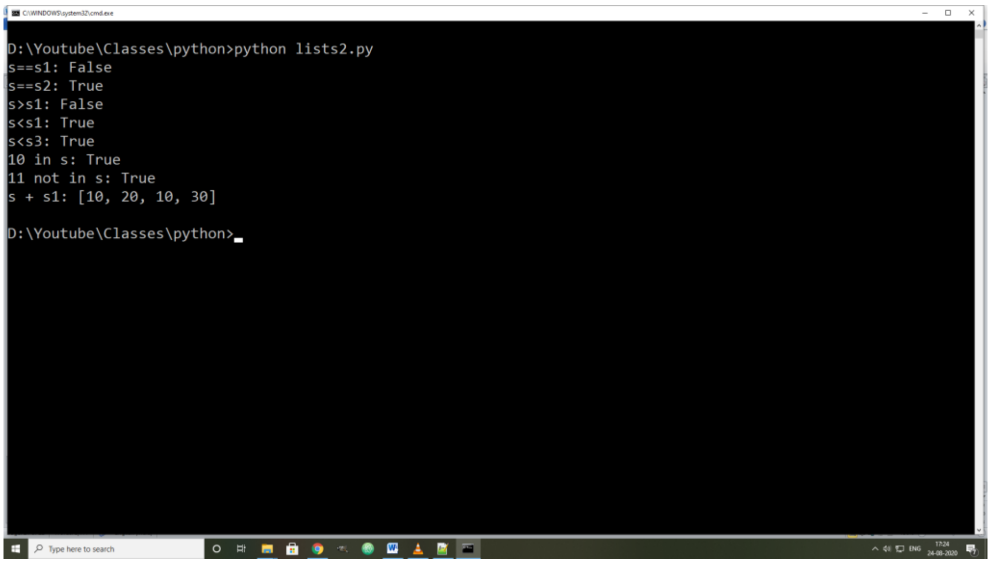
Concatenating Lists (+): We can concatenate two lists by using the concatenate operator +

Repetition Operator (*) : We can use the repetition operator to repeat all the elements of the list for the specified number of times.

Example:

```
#program to demonstrate operators on lists
s=[10,20]
s1=[10,30]
s2=[10,20]
s3=[10,40,50]
print ("s==s1:",(s==s1))
print ("s==s2:",(s==s2))
print ("s>s1:",(s>s1))
print ("s<s1:",(s<s1))
print ("s<s3:",(s<s3))
print ("10 in s:",(10 in s))
print ("11 not in s:",(11 not in s))
print ("s + s1:",(s+s1))
```

Output:



```
D:\Youtube\Classes\python>python lists2.py
s==s1: False
s==s2: True
s>s1: False
s<s1: True
s<s3: True
10 in s: True
11 not in s: True
s + s1: [10, 20, 10, 30]

D:\Youtube\Classes\python>
```

Built in Methods in List:

list.append(obj) : Adds an element at the end of the list

list.count(obj) : Returns the number of elements with the specified value

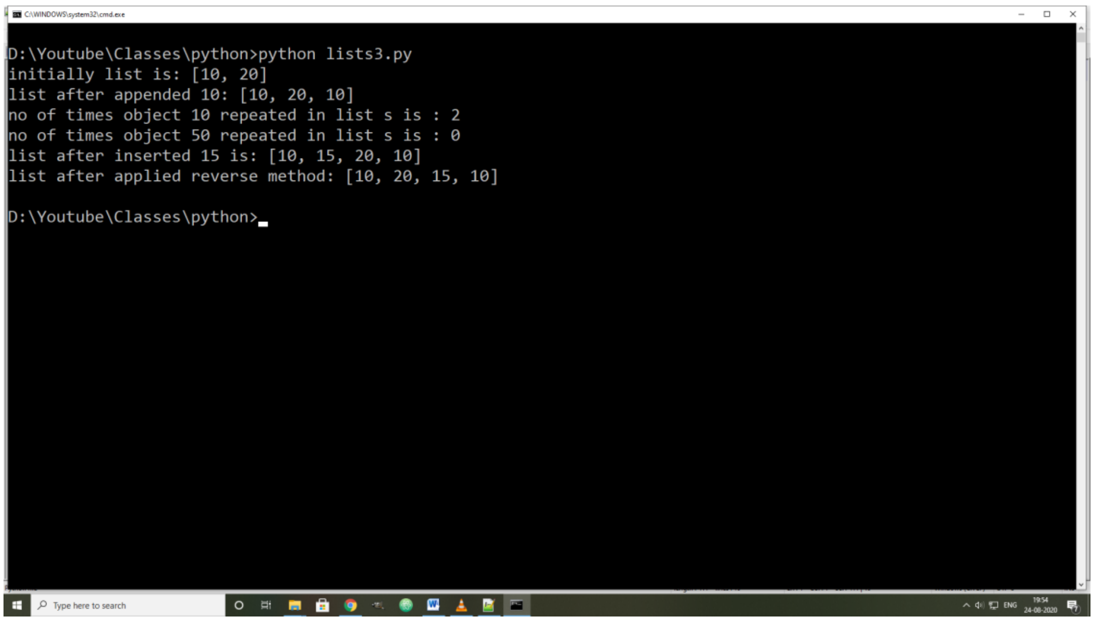
list.index(obj) : Returns the index of the first element with the specified value

list.insert(index,obj) : Adds an element at the specified position

list.reverse() : Reverses the order of the list

```
#program to demonstrate builtin methods on lists
s=[10,20]
print("initially list is:",s);


s.append(10)
print("list after appended 10:",s);

print("no of times object 10 repeated in list s is :",s.count(10))
print("no of times object 50 repeated in list s is :",s.count(50))

s.insert(1,15)
print("list after inserted 15 is:",s);

s.reverse()
print("list after applied reverse method:",s);
```

Output



```
D:\Youtube\Classes\python>python lists3.py
initially list is: [10, 20]
list after appended 10: [10, 20, 10]
no of times object 10 repeated in list s is : 2
no of times object 50 repeated in list s is : 0
list after inserted 15 is: [10, 15, 20, 10]
list after applied reverse method: [10, 20, 15, 10]

D:\Youtube\Classes\python>
```

Tuples: ( )

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

It is also called as read only list which means both list and tuple are same with a small difference that once a tuple is created with elements, we can't update the elements.

Example:

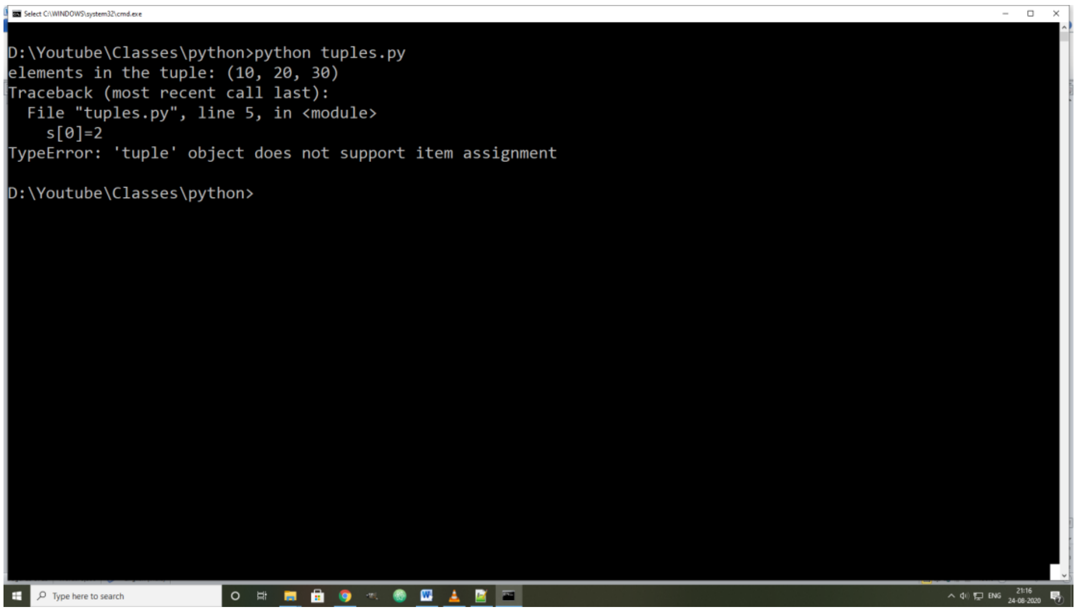
s=(10,20,30)

We can access the tuple elements (same as list) by using index.

Example Program

```
#program to demonstrate tuples
s=(10,20,30)
print("elements in the tuple:",s)

s[0]=2
```

Output



```
D:\Youtube\Classes\python>python tuples.py
elements in the tuple: (10, 20, 30)
Traceback (most recent call last):
  File "tuples.py", line 5, in <module>
    s[0]=2
TypeError: 'tuple' object does not support item assignment

D:\Youtube\Classes\python>
```

Above program returned error because we are trying to update the value of a tuple's element which is not possible.

DICTIONARIES IN PYTHON

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.

Dictionary stores the elements in key-value pairs. We can access the elements by using keys. and Dictionaries are mutable which means we can update the elements any number of times.

We can create dictionaries by using curly braces as follows

```
x = {'country':'India', 'channel':'Telugu Web Guru'}
```

We can access the elements of a dictionary by using keys as follows

```
print (x['country'])          - This statement prints 'India' as output
```

We can insert elements at any time to the existing dictionary by using key value pairs as follows

```
x [' Mentor '] = ' Santosh '
```

We can remove element from the dictionary by using del statement

```
del x['country']
```

We can remove all elements from the dictionary by using clear() method

```
x.clear()
```

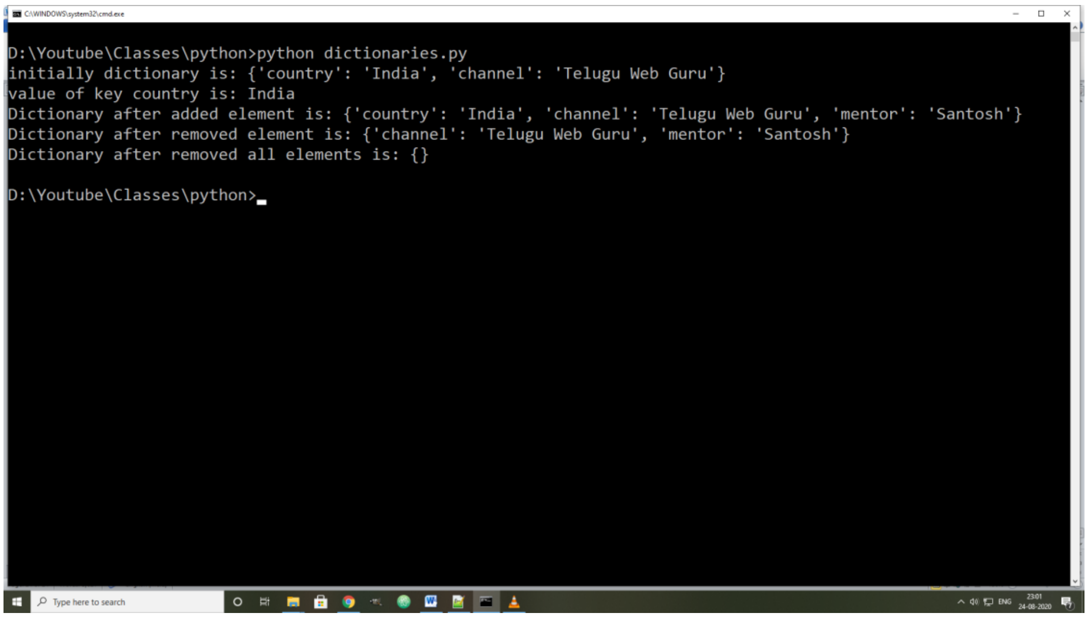
We can delete the entire dictionary also by using del statement as follows

```
del x
```

Example Program

```
# program to demonstrate dictionaries

x={'country':'India', 'channel':'Telugu Web Guru'}
print("initially dictionary is:",x)

print("value of key country is:",x['country'])

#adding element
x['mentor']='Santosh'
print("Dictionary after added element is:",x)

#removing element
del x['country']
print("Dictionary after removed element is:",x)

#remove all elements from dictionary
x.clear()
print("Dictionary after removed all elements is:",x)
```

Output



```
D:\Youtube\Classes\python>python dictionaries.py
initially dictionary is: {'country': 'India', 'channel': 'Telugu Web Guru'}
value of key country is: India
Dictionary after added element is: {'country': 'India', 'channel': 'Telugu Web Guru', 'mentor': 'Santosh'}
Dictionary after removed element is: {'channel': 'Telugu Web Guru', 'mentor': 'Santosh'}
Dictionary after removed all elements is: {}

D:\Youtube\Classes\python>
```

FUNCTIONS IN PYTHON

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

How to create a function?

We can create functions by using def keyword as shown in following syntax.

```
def functionname (Argument List) :
        statement1
        statement2
```

Example :

```
def sum(a,b):
        print (a+b)
```

How to call functions in python?

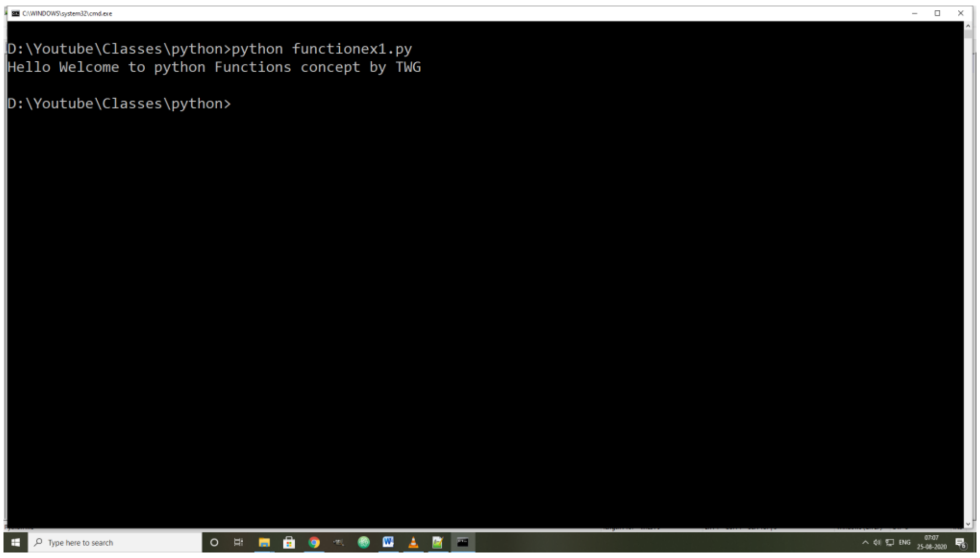
functions will be executed only whenever they are called. we can call the function as

functionname(optional argument values)

Example Program:

```
#program to demonstrate functions in python

#function definition
def sayhello():
   print ("Hello Welcome to python Functions concept by TWG")


#function calling
sayhello()
```

Output:



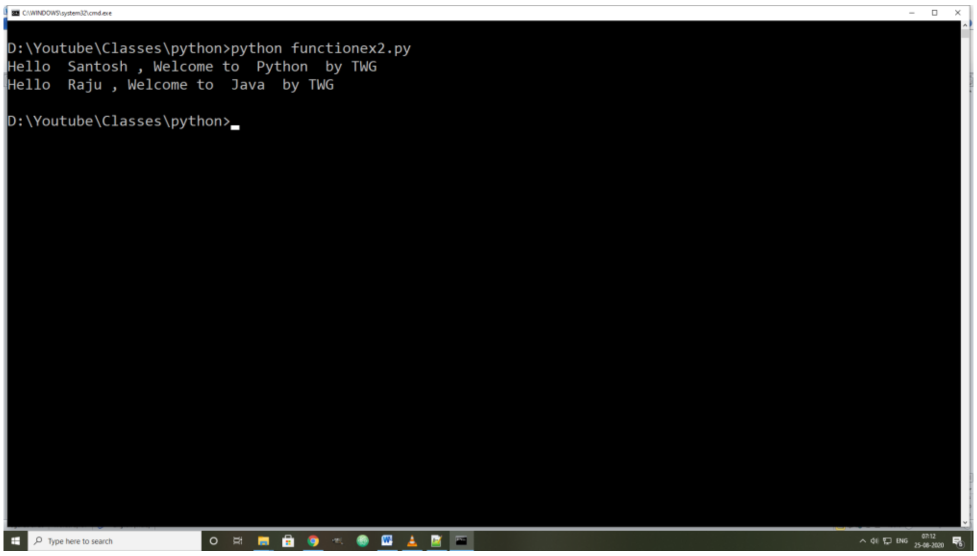
```
D:\Youtube\Classes\python>python functionex1.py
Hello Welcome to python Functions concept by TWG

D:\Youtube\Classes\python>
```

In the above example we can pass the arguments also.

```
#program to demonstrate functions with arguments in python

#function definition
def sayhello(name,course):
   print ("Hello ",name,", Welcome to ",course," by TWG")

#function calling
sayhello("Santosh","Python")

sayhello("Raju","Java")
```

above function only will be called whenever we call with proper arguments that means sayhello will be executed only whenever we call with two argument values.

Output:



```
D:\Youtube\Classes\python>python functionex2.py
Hello  Santosh , Welcome to  Python  by TWG
Hello  Raju , Welcome to  Java  by TWG

D:\Youtube\Classes\python>
```

Setting default values to the arguments:

We can set default values to arguments so that they will be considered when we are not passing argument values in function calling as follows.

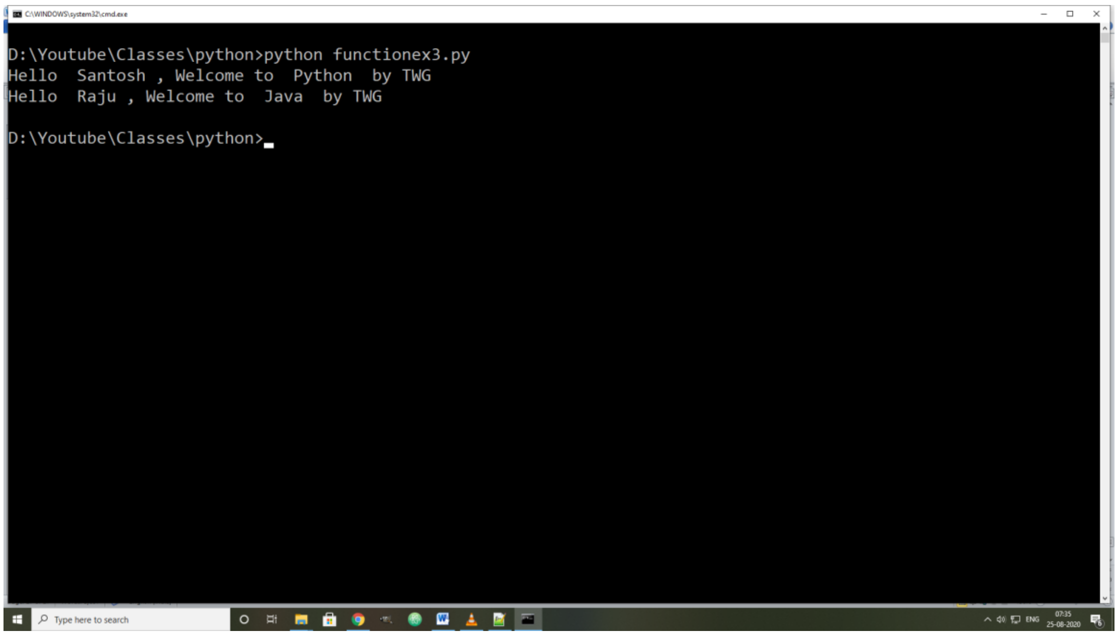
```
functionname(argument1=value1,argument2=value2):

        statements
```

Example:

```
#program to demonstrate functions with default values in python

#function definition
def sayhello(name="Santosh",course="Python"):
   print ("Hello ",name,", Welcome to ",course," by TWG")
```

```
#function calling
sayhello()

sayhello("Raju","Java")
```

Output:



```
D:\Youtube\Classes\python>python functionex3.py
Hello  Santosh , Welcome to  Python  by TWG
Hello  Raju , Welcome to  Java  by TWG

D:\Youtube\Classes\python>
```

In the above example please observe in first function calling, sayhello function is called without arguments. In this type of situations python will consider the default values Santosh & Python which are provided in function definition. Whenever we are passing values then these passed values will be considered.

Lambda Functions :

Lambda functions are anonymous functions (functions without name) in python.

We can define lambda function by using 'lambda' keyword as follows.

```
lambda arg1,arg2:statements
```

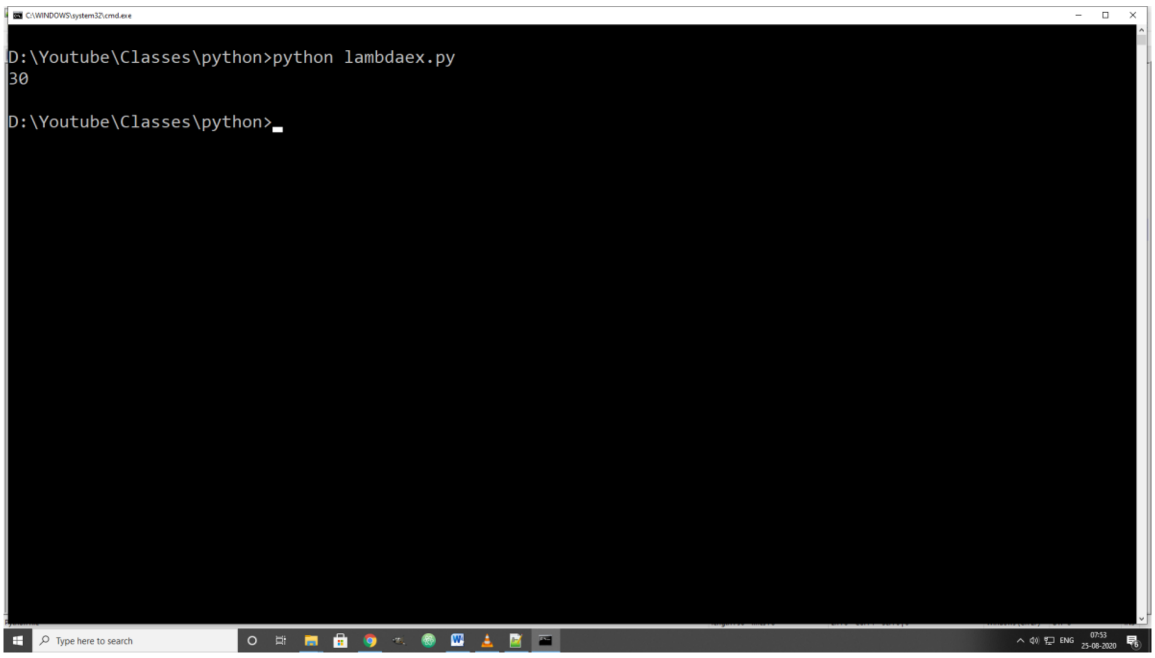
```
def f(arg1,arg2):
        return arg1+arg2
```

Example :

```
#program to demonstrate python lambda functions

f = lambda a,b : a+b

print(f(10,20))
```

Please observe that the lambda function is assigned to variable f as lambda functions does not have any name. Whenever we call this lambda function we can call with associated name ' f '.

Output:



```
D:\Youtube\Classes\python>python lambdaex.py
30

D:\Youtube\Classes\python>
```

MODULES & PACKAGES IN PYTHON

Modules are useful to organize the code logically. Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application. Modules uses files concept to organize it's code.

How to create a module?

To create a module just save the code you want in a file with the file extension .py

Example: file1.py

```
x=10

def show(x):

    print(x)
```

How to use a module in other files?

We can use modules in other files by using import statement. Consider the following statement. in file2.py if we want to use variable x of module file1 (above example) then we will first import it into file2.

Example : file2.py

```
import file1                    #imports entire module

or

from file1 import x             #imports only x from file1 module so that we can use  only x of file1 in file2
```

Example Program:

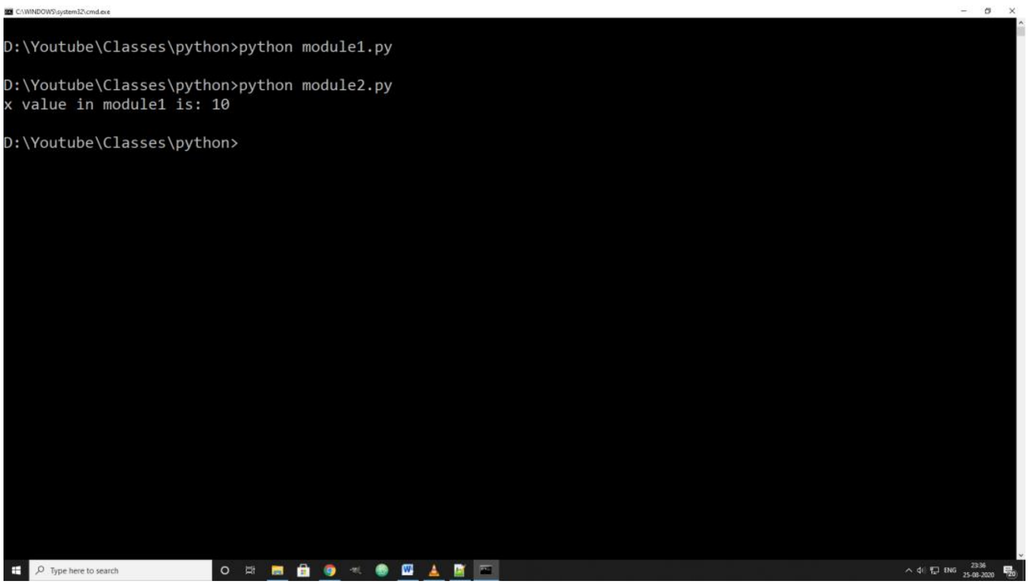
module1.py

```
x=10
def show():
   print("x value in module1 is:",x)
```

module2.py

```
import module1
module1.show()
```

Output



```
D:\Youtube\Classes\python>python module1.py

D:\Youtube\Classes\python>python module2.py
x value in module1 is: 10

D:\Youtube\Classes\python>
```

In above example module2 imports entire module1 so it can access both variable (x) and method (show) of module1.

If we want to access only variable and not method then we can do the same with the help of from – import combination as follows in module1

    from first import x

If we write the above import statement in module2 then we can access only variable x of module1 in module2.

Packages :

Collection of related modules are said to be a package. We will implement this concept by storing all related modules in the same directory/folder. packages concept gathers all related modules at one place so that related modules can use each other in more flexible manner and because of this packages concept, we can avoid name collisions by inserting modules with same name into different packages.

Every package contains an initialization method __init__.py.

```
college/
    __init__.py
   branches.py

  branch /
     __init__.py
     students.py
     staff.py
```

In above example college is a main package that contains __init__ method and a module called branches. Main package college internally contains a sub package with the name branch which internally contains 2 modules studentsand staff, an __init__ method

If we want to import students module we can write the statement as follows

```
import college.branch.students
```

If we want to import only a variable (for ex : name) of students module then we will write as follows

```
from college.branch.students import name
```

FILES IN PYTHON

The key function for working with files in Python is the open() function.

The open() function takes three parameters in which first two are mandatory filename, and mode. third parameter is buffering

access mode tells in which mode file is opened i.e., read / write / append etc.,

buffering means we can use a buffer between program and file so that if either of these two is speed in processing than other then there will be no chances of data losses.

buffering -1 means it takes system default buffer value. 0 means no buffering, 1 means line buffering.

Syntax :

```
open(filename, access mode, buffering)
```

Example:

```
obj = open("sample.txt", 'r', -1)
```

In above example we opened a file and assigned to a variable obj which will act as a file handler so that we can do different read/write operations on it.

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

"x" - Create - Creates the specified file, returns an error if the file exists

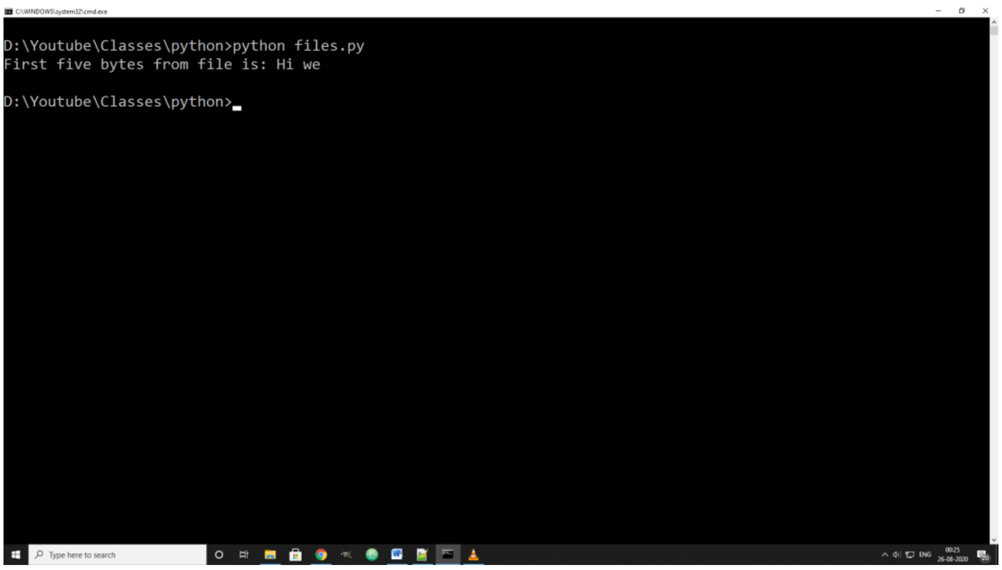
files.py

```
#program to demonstrate files
f = open("sample.txt","r",-1)

a = f.read(5) #reads 5 bytes from the file

f.close() #closing the file

print ("First five bytes from file is:",a)
```

sample.txt

```
Hi we are learning python files concept from telugu web guru santosh
```

Output



```
D:\Youtube\Classes\python>python files.py
First five bytes from file is: Hi we

D:\Youtube\Classes\python>
```

read :

As shown in above example, we can use read method to read specified number of bytes. If we pass -1 then we can read entire content. We can also use readline() to read online from the specified file

We can use readlines() to read all lines (entire content) from the specified file. It creates a list and insert each line of a file as list item and returns the entire list.

write:

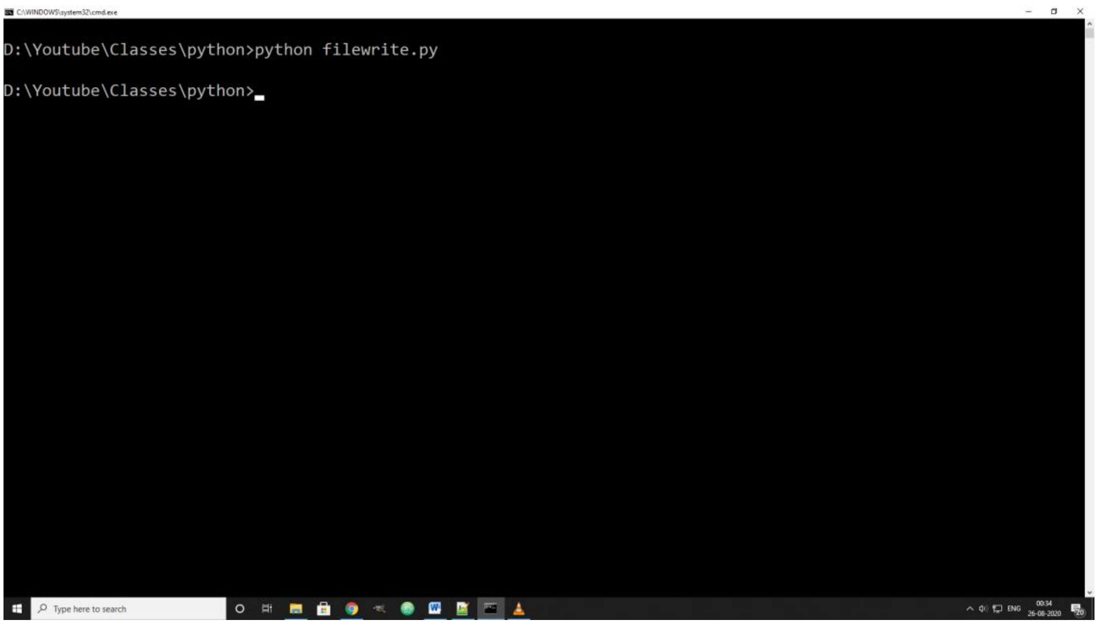
We can write contents into the file by using write method as follows

write(content)	–	will writes the content into the file
write(list)	-	will writes the list content into the file where each list item will be inserted into a separate line.

Example 1 :

```
#program to write contents into a file

f = open("sample1.txt","w")
f.write('hello')
f.close()

mylist = ["Telugu Web Guru","Santosh","Python Classes"]
f1 = open("sample2.txt","w")
f1.writelines(mylist);
f1.close()
```

Output:



```
D:\Youtube\Classes\python>python filewrite.py

D:\Youtube\Classes\python>_
```

After execution new file is created with the name ‘sample1.txt’,’sample2.txt’ with the contents as shown below

sample1.txt

```
hello
```

sample2.txt

```
Telugu Web GuruSantoshPython Classes
```

In above example writelines() method is used to write entire list contents into the file.

append:

We can append the contents into the file where new contents will be appended to the end of the existing content where in write mode old contents were replaced by new contents.

Example

```
f = open("sample.txt" , 'a' )
```

ERRORS & EXCEPTIONS IN PYTHON

There are two types of errors.

Syntax Errors: errors in writing statements with wrong syntax.

Logical Errors: Errors occurred in logic which leads to error.

Because of the exceptions that are occurred in few lines, entire program execution will be stopped until we clear those errors.

To solve this, we need to handle the exceptions. To handle the exceptions, python provides two keywords try, except.

try	-	It will identify the exceptions within it's block, create the objects for those exceptions and throw them to the handling area.
except	-	These are exception handlers that catches the object from the try block and run the exception handler code which is defined in its block.

Example:

```
try:
   statement1
   statement2
except IndexError:
   statements
except KeyError:
   statements
```

Consider the above example in which set of two statements were put in try block. if there are any exceptions occurred try will identify it and create the corresponding object and throw it to the related except block. Now except block catches the object and run the handling code.

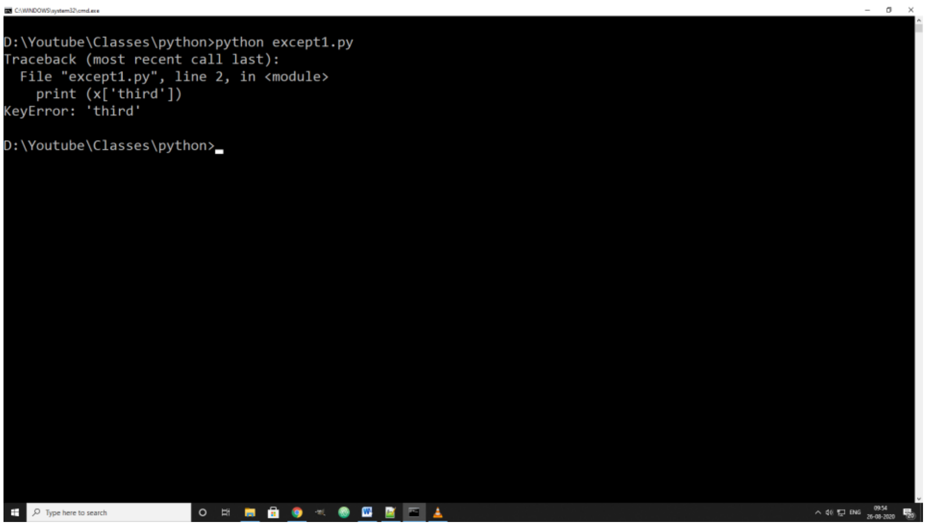
For example consider the below program

```
x={'first':'Python','second':'Java'}
print (x['third'])
print("Hi")
```

In the above example there no element exists with the key 'third' . So second statement contains error. Because of this third statement is also not executed until we handle the exception in statment2.

So exceptions are the abnormal conditions which stop the program's execution.

Output

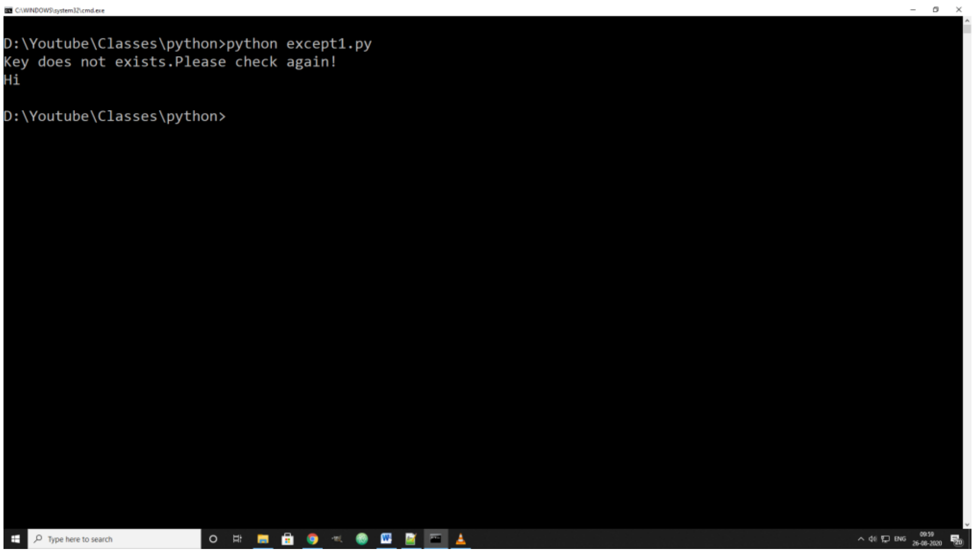


```
D:\Youtube\Classes\python>python except1.py
Traceback (most recent call last):
  File "except1.py", line 2, in <module>
    print (x['third'])
KeyError: 'third'

D:\Youtube\Classes\python>
```

To handle this, we must provide corresponding try-except blocks

```
x={'first':'Python','second':'Java'}

try:
   print (x['third'])
   print ("I am after exception statement in try")
except KeyError:
   print("Key does not exists.Please check again!")
print("Hi")
```

Output



```
D:\Youtube\Classes\python>python except1.py
Key does not exists.Please check again!
Hi

D:\Youtube\Classes\python>
```

Because we handled the exception with try-except our program is executed successfully and hi message is also displayed. But the statements that are existed after exception statement in try block will never be executed because control never goes back after except block execution. So we may need to take care of it by writing separate try-except blocks if necessary for the important statements.

INTRODUCTION TO OOPS

There are different programming paradigms or approaches available. Those are procedure-oriented programming (POP) and object-oriented programming (OOP).

Procedure oriented programming always concentrates on task/function to be performed. It encourages to implement a separate function for every activity which leads to increase in program complexity. Problems in POP approach includes under stability, security as all the variables and functions are accessed by any other module in the program and there is less or no reusability of existing code.

To solve all the above said problems, OOP always concentrates on class and object. Here object is a real time entity. OOP allows to bind classes and variables into a single unit through which it can provide security. It allows code reusability through inheritance.

OOP Features :

The following are the OOP features.

1) **Encapsulation**: Wrapping / Binding of data(variables) and code(methods) into a single unit is said to be encapsulation. It helps us to secure our members (Member variables & member methods).
2) **Class**: Class is the basis where encapsulation is implemented. Class contains collection of variables and Methods. Class provides the definition of an object.
3) **Object**: It is real time entity. It is also defined as instance of a class.
4) **Data Abstraction**: Representing essential features without including background details is said to be data abstraction. For example, we use predefined methods (for example strlen in C language) in any programming language without knowing about its background implementation details.
5) **Inheritance**: creating a new class from already existing class is said to be inheritance. It can also be defined as object of one class acquires the properties of object of another class.
6) **Polymorphism** (Poly + Morphic): polymorphism is from the Greek words poly (many) and morphic (forms). Polymorphism is defined as "Same operation exhibits different behaviors in different instances".

   For example, the operator +, if we place it between two values, it returns addition as result. The same + operator returns concatenation as result if we place it between two strings. This is called as operator overloading which is one of the best examples of operator overloading.

   In the same way, if we call the method draw () with the circle, then it draws the circle and if we call it with rectangle then it draws rectangle. This is called as method overloading. This is another best example for polymorphism.

CLASSES & OBJECTS IN PYTHON

Class provides the definition of an object. Object is instance of a class. Class and object are interrelated with each other.

The following syntax specifies how to declare a class.

```
class classname (baseclass):
        'documentation line'
        class suite or code
```

here class is a keyword that is used to declare a class. baseclass is the super or parent class of the class that is being created. object is the default base class which is super class of all classes in python. Documentation line describes about the class and its members and it's functionalities in single line.

The classes that are defined with base class are said to be New Model Classes and the classes that are declared without base class are said to be classic classes.

Example:

```
class BestMobile(object):
    ' This class defines the best mobile objects '
    print (" Best mobile must have good processor, camera and memory.")
```

Creating Object:

We can create objects by using the following syntax.

Syntax:

```
Objectname = classname()
```

Example

```
samsungX = BestMobile()
appleY = BestMobile()
```

METHODS IN PYTHON

Functions in classes are said to be methods. Methods contains set of statements that performs a task.

Procedure to create a method:

- Define a class with required method definitions
- Creating object of that class
- Invoke the method by using object

Syntax to create a method:

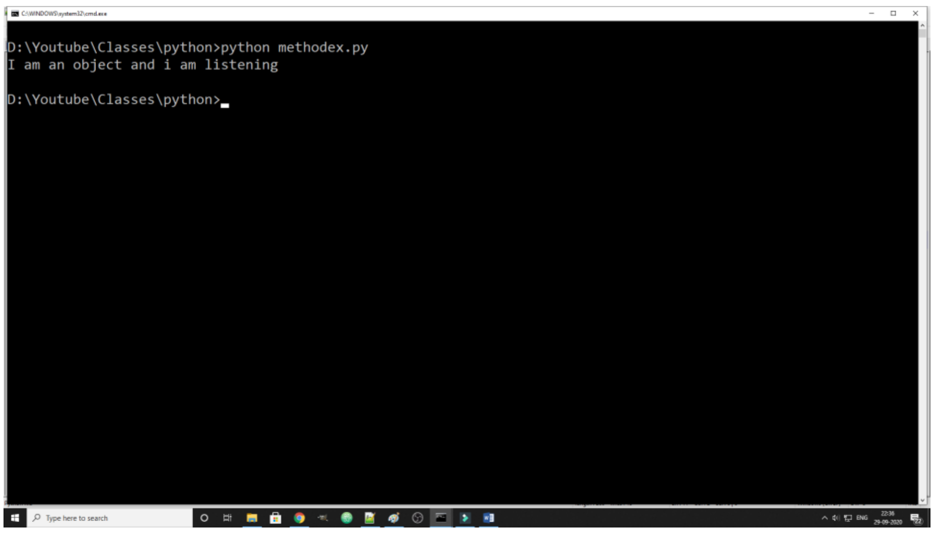
```
class  classname (base class):
    'documentation line'
     def methodname(self):
          method code
```

The fist argument in method must be self. You may or may not send any number of arguments but a single argument self is must. Python system will replaces the current object in place of self.

Example

```
#program to demonstrate class and method
class pythontraining(object):
    def listen(self):
        print('I am an object and i am listening');

#creating an object or instance
x = pythontraining()
#invoke or call the method
x.listen();
```

Output



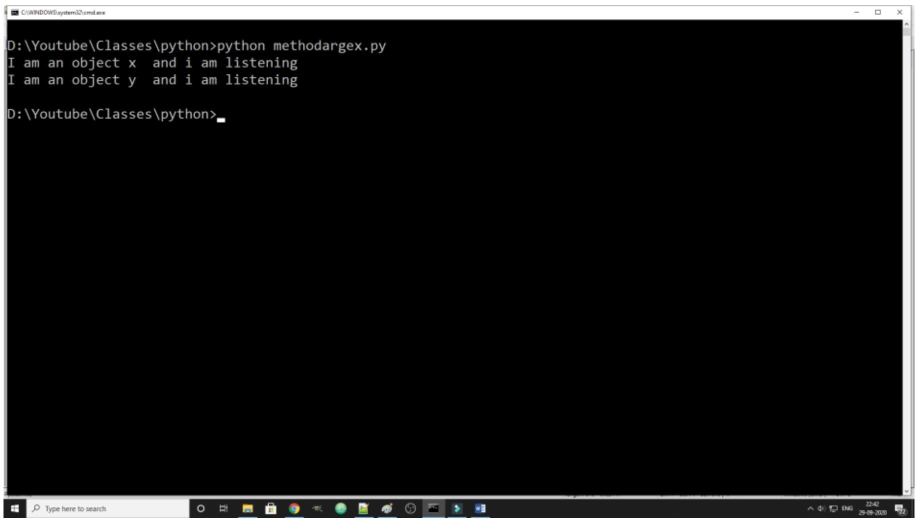
```
D:\Youtube\Classes\python>python methodex.py
I am an object and i am listening

D:\Youtube\Classes\python>
```

In the following example we are adding one more argument 'name' to the method.

Like this you can add any number of arguments, but first argument must be self.

```
#program to demonstrate class and method
class pythontraining(object):
   def listen(self,name):
      print('I am an object',name,' and i am listening');


#creating an object or instance
x = pythontraining()

#invoke or call the method
x.listen("x");


y=pythontraining()
y.listen("y");
```

Output



```
D:\Youtube\Classes\python>python methodargex.py
I am an object x  and i am listening
I am an object y  and i am listening

D:\Youtube\Classes\python>
```

VARIABLES IN PYTHON

Instance variables are the variables in which a separate copy is maintained by each and every object.

Instance variables are declared after creating objects.

Example :

```
x = pythontraining()

x.pen = "Parker"

x.book ="White NoteBook"
```

In the above example pen and book are instance variables which are related to object 'x' .

Class Variables:

These are the variables in which a single copy is maintained for entire class i.e., for all objects only single copy is maintained which is shared by all the objects of that class.

The variables that are directly declared are said to be class variables.

Example:

```
class pythontraining(object):

      board = "White Board"
```

Instance Methods:

These are the methods in which a separate copy is maintained by each and every object

The methods that are declared in a class without any decorator (@staticmethod, @classmethod) are said to be instance methods.

Example :

```
class pythontraining:

    def bookwriting(self):

        print('I am writing notes in my own book');
```

Static Methods:

These methods are shared by all objects of that class but when compared with class methods, statics methods does not access state of the class .

Class Methods :

These methods are shared by all the objects because a single copy is maintained for all the objects.

There are two types of class methods: class methods & instance methods.

We can declare class methods and static methods by using decorators : @staticmethod and @classmethod

Example :

```
class pythontraining:
    @staticmethod
    def boardreading():
            print('everyone is reading the contents on the board')
   @classmethod
    def boardreading(cls):
            print('everyone is reading the contents on the board')
```

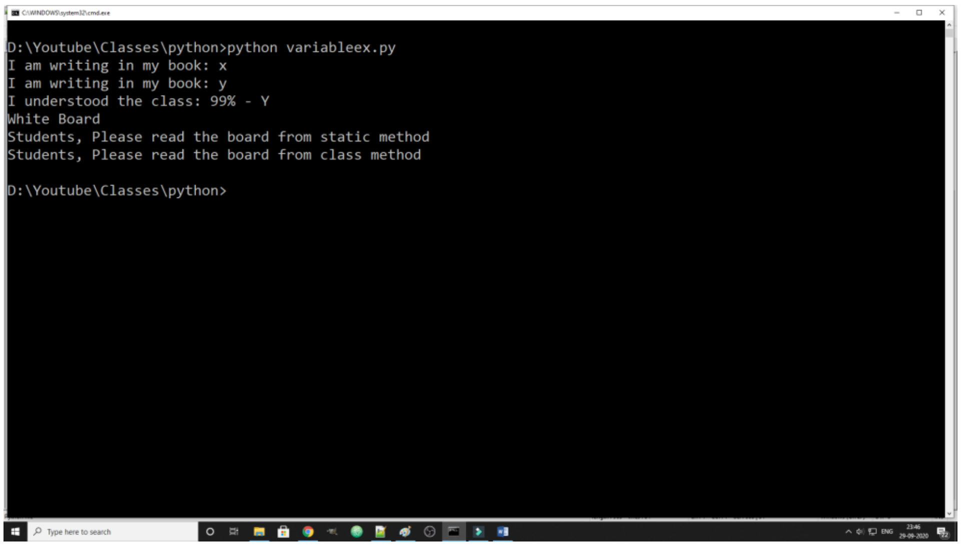
In class methods we need to add an argument cls which will be replaced by class name dynamically so that method can access the state of the class.

Example Program :

```
#program to demonstrate variables and methods
class PythonTraining:
    board = "White Board" #class variable

    def bookwriting(self,name): #instance method
        print('I am writing in my book:',name)

    def listening(self,name): #instance method
        print('I am Listening :',name);

    def understanding(self,percentage): #instance method
        print('I understood the class:',percentage)

    @staticmethod
    def boardreading(): #static method
        print('Students, Please read the board from static method')

    @classmethod
    def boardreading2(cls): #static method
        print('Students, Please read the board from class method')


x = PythonTraining() #object x
x.bookwriting("x")

y = PythonTraining() #object y
y.bookwriting('y')
y.understanding('99% - Y')

print(PythonTraining.board)
```

```
PythonTraining.boardreading()

PythonTraining.boardreading2()
```

Output



```
D:\Youtube\Classes\python>python variableex.py
I am writing in my book: x
I am writing in my book: y
I understood the class: 99% - Y
White Board
Students, Please read the board from static method
Students, Please read the board from class method

D:\Youtube\Classes\python>
```

CONSTRUCTORS IN PYTHON

Constructors are used to initialize values to the instance variables.

If we want to initialize the values to instance variables with methods, then it is a two step process.

Step - 1: Create the object

```
x = PythonTraining()
```

step -2: Initialize values to the instance variables

```
x.pen = "Parker"
```

The same can be achieved in a single step by using constructor as follows.

```
x = PythonTraining("Parker", "White NoteBook")
```

To implement constructors, we need to write __init__(self) method in the class as follows.

```
Class PythonTraining:
    def __init__(self,p,b):
        self.pen=p;
        self.book=b;
```

Another method related to constructors that creates objects is __new__(cls) as follows.

```
def __new__(cls):
    #code
```

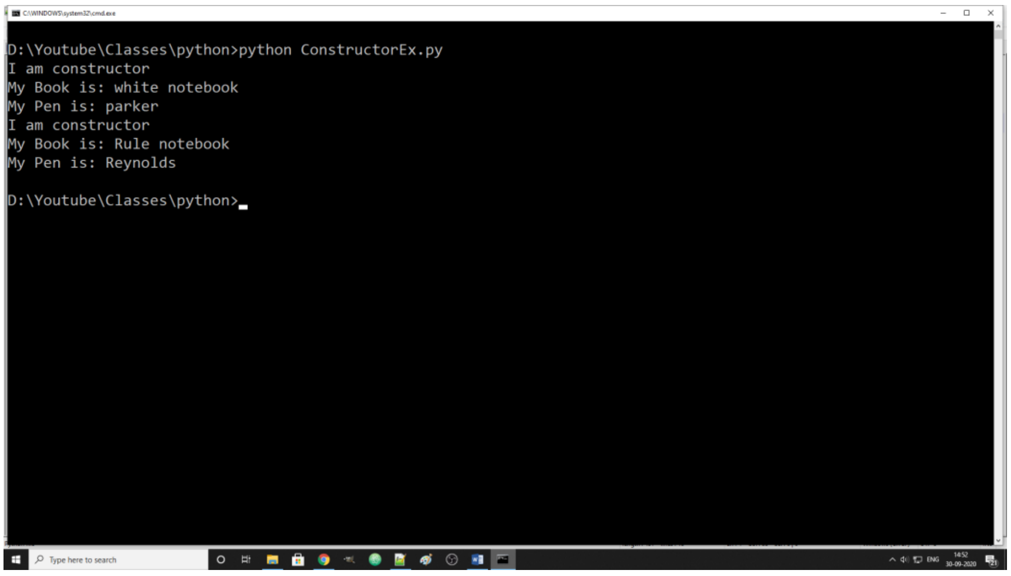
Always __new__ method will creates object and pass it to the __init__ method where __init__ will initialize the values.

Example:

```
#program to demonstrate constructors in python
class PythonTraining:
    def __init__(self,a,b):
        print('I am constructor')
        self.book=b
        self.pen=a

    def display(self):
        print('My Book is:',self.book)
        print('My Pen is:',self.pen)

#object creation
x=PythonTraining('parker','white notebook')
x.display()
```

```
y=PythonTraining('Reynolds','Rule notebook')
y.display()
```

Output:



```
D:\Youtube\Classes\python>python ConstructorEx.py
I am constructor
My Book is: white notebook
My Pen is: parker
I am constructor
My Book is: Rule notebook
My Pen is: Reynolds

D:\Youtube\Classes\python>
```

INHERITANCE IN PYTHON

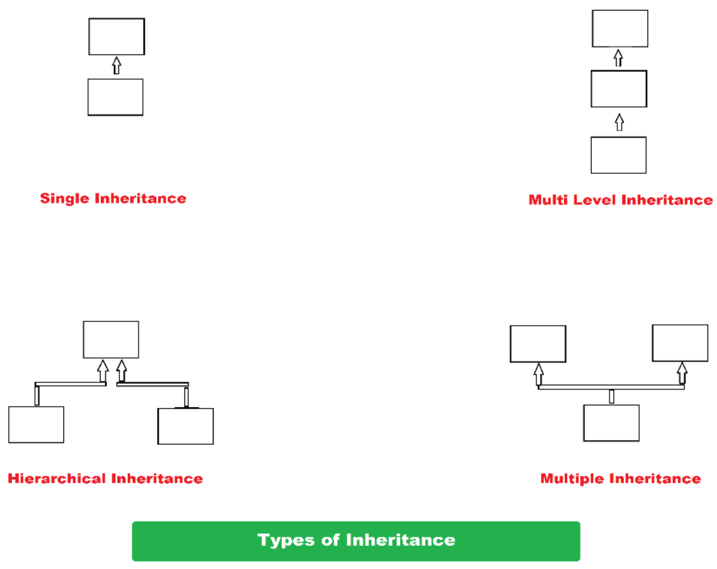
Creating a new class from already existing class is said to be inheritance.

Or

Object of one class acquires the properties of object of another class is said to be inheritance.

In this inheritance process, the class which is already existed is called as base class or super class and the class that is newly derived is said to be derived class or sub class.

Types of Inheritance:



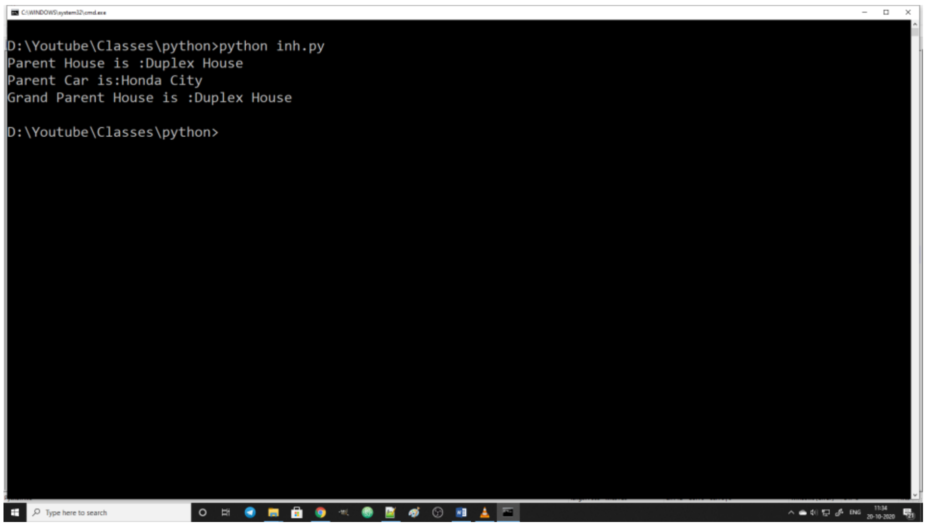
Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

Example:

```
class GrandParent:
   def __init__(self,h):
      self.house=h

   def displayGrandParentProperties(self):
      print('Grand Parent House is :'+self.house)

class Parent(GrandParent):
   def __init__(self,h,c):
      self.car=c
      super().__init__(h);

   def displayParentProperties(self):
      print('Parent House is :'+self.house)
      print('Parent Car is:'+self.car)
```

```
sub1 = Parent('Duplex House','Honda City');
sub1.displayParentProperties();
sub1.displayGrandParentProperties();
```

Output:



```
D:\Youtube\Classes\python>python inh.py
Parent House is :Duplex House
Parent Car is:Honda City
Grand Parent House is :Duplex House

D:\Youtube\Classes\python>
```

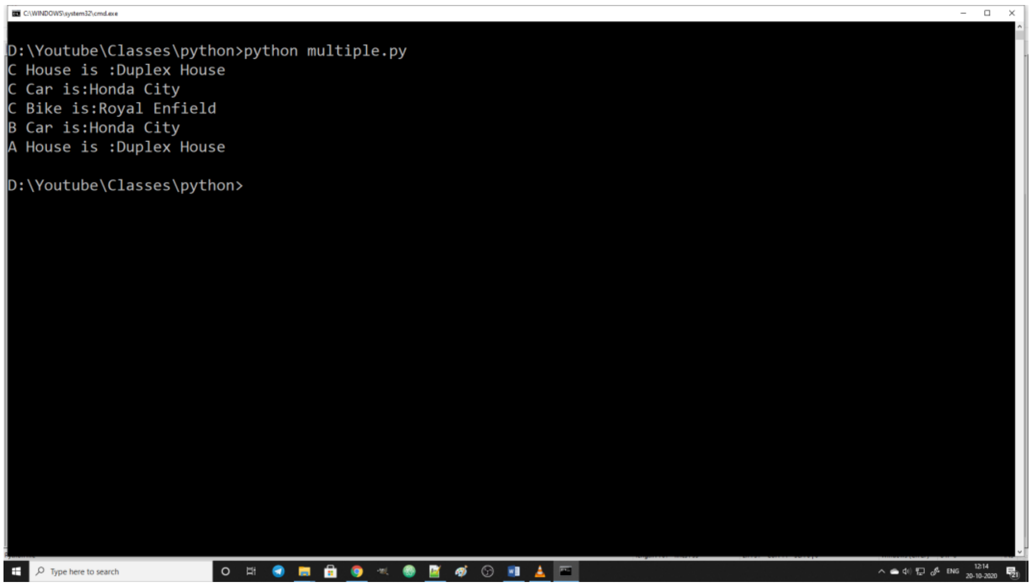
Multiple Inheritance: When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.

Example:

```
class A:
   def __init__(self,h):
      self.house=h

   def displayAProperties(self):
      print('A House is :'+self.house)

class B:
   def __init__(self,c):
      self.car=c


   def displayBProperties(self):
      print('B Car is:'+self.car)
```

```
class C(A,B):
   def __init__(self,h,c,b):
      self.bike=b
      self.house=h;
      self.car=c;

   def displayCProperties(self):
      print('C House is :'+self.house)
      print('C Car is:'+self.car)
      print('C Bike is:'+self.bike)

sub1 = C('Duplex House','Honda City','Royal Enfield');
sub1.displayCProperties();
sub1.displayBProperties();
sub1.displayAProperties();
```

Output:



```
D:\Youtube\Classes\python>python multiple.py
C House is :Duplex House
C Car is:Honda City
C Bike is:Royal Enfield
B Car is:Honda City
A House is :Duplex House

D:\Youtube\Classes\python>
```

Multilevel Inheritance

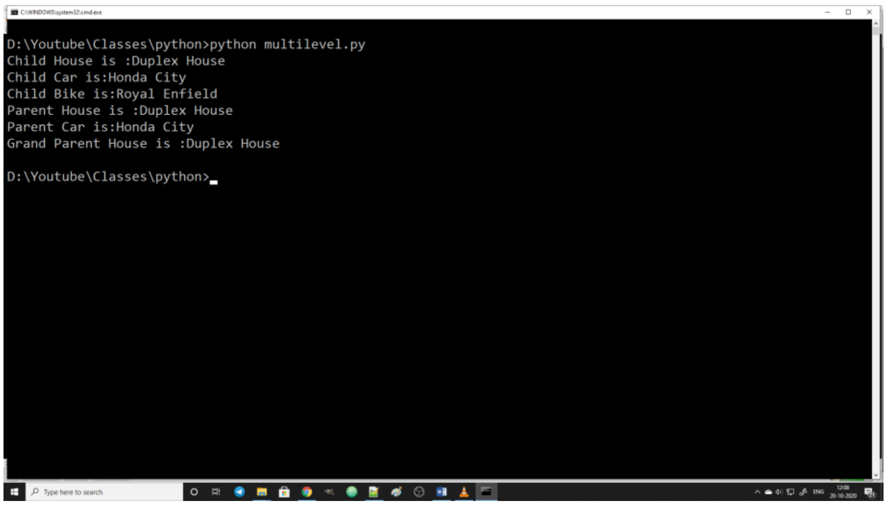
In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.

Example:

```
class GrandParent:
   def __init__(self,h):
      self.house=h

   def displayGrandParentProperties(self):
      print('Grand Parent House is :'+self.house)

class Parent(GrandParent):
   def __init__(self,h,c):
      self.car=c
      super().__init__(h);
```

```
  def displayParentProperties(self):
      print('Parent House is :'+self.house)
      print('Parent Car is:'+self.car)


class Child(Parent):
  def __init__(self,h,c,b):
      self.bike=b
      super().__init__(h,c);

  def displayChildProperties(self):
      print('Child House is :'+self.house)
      print('Child Car is:'+self.car)
      print('Child Bike is:'+self.bike)

sub1 = Child('Duplex House','Honda City','Royal Enfield');
sub1.displayChildProperties();
sub1.displayParentProperties();
sub1.displayGrandParentProperties();
```

Output:



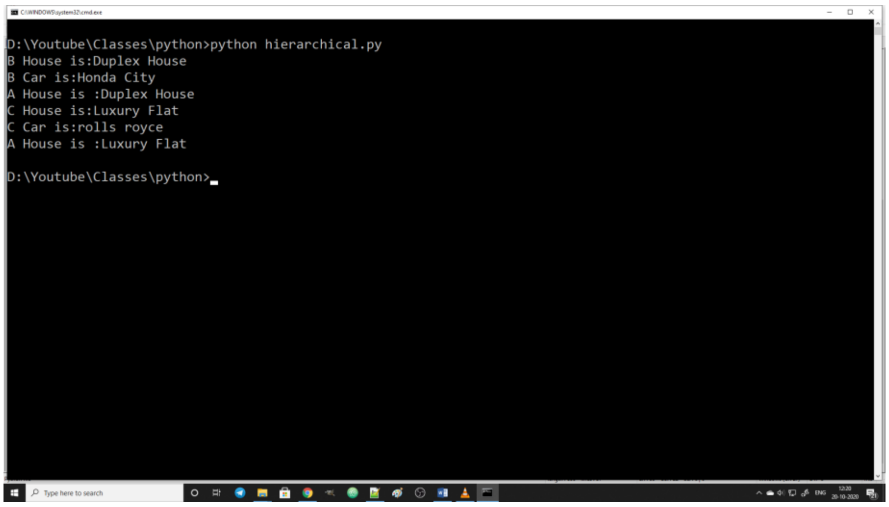
```
D:\Youtube\Classes\python>python multilevel.py
Child House is :Duplex House
Child Car is:Honda City
Child Bike is:Royal Enfield
Parent House is :Duplex House
Parent Car is:Honda City
Grand Parent House is :Duplex House

D:\Youtube\Classes\python>_
```

Hierarchical Inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

Example:

```
class A:
  def __init__(self,h):
      self.house=h

  def displayAProperties(self):
      print('A House is :'+self.house)

class B(A):
  def __init__(self,h,c):
      self.car=c
      super().__init__(h);


  def displayBProperties(self):
      print('B House is:'+self.house)
      print('B Car is:'+self.car)


class C(A):
  def __init__(self,h,c):
      self.car=c
      super().__init__(h);


  def displayCProperties(self):
      print('C House is:'+self.house)
      print('C Car is:'+self.car)

sub1 = B('Duplex House','Honda City');
sub1.displayBProperties();
sub1.displayAProperties();

sub2 = C('Luxury Flat','rolls royce');
sub2.displayCProperties();
sub2.displayAProperties();
```

Output:



```
D:\Youtube\Classes\python>python hierarchical.py
B House is:Duplex House
B Car is:Honda City
A House is :Duplex House
C House is:Luxury Flat
C Car is:rolls royce
A House is :Luxury Flat

D:\Youtube\Classes\python>
```

SETS IN PYTHON

A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets. Sets never store duplicate elements.

How to create a set ?

S1 = {1,a,3.6}   # we can create by assigning values

S2 = set([4,'hello',10.6])   # we can create by using set constructor also

How to add elements to the existing set?

We can add elements to the existing set by using add method as follows.

s1.add(10);

We can add multiple elements to the existing set at a time as follows.

s3 = {7,' x ',6.8}

s1.update(s3)   # entire elements of s3 will be added to s1 at a time.

We can remove elements from a set by using remove method.

s1.remove(10) will removes the element 10 from the set

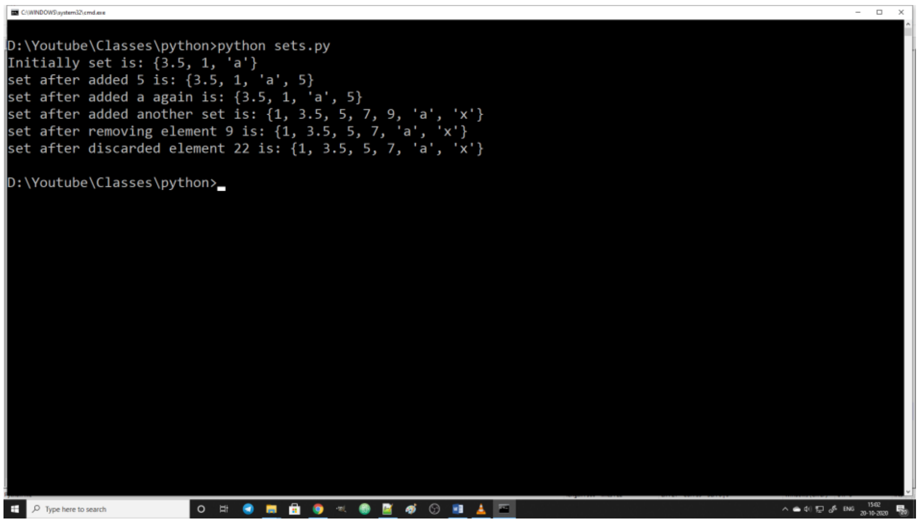
If the element that we are trying to remove does not exists then it returns KeyError.

We can an alternate method discard() to remove elements where it does not returns any error if the element that we are trying to remove does not exist.

Example:

```
#program to demonstrate sets in python

s1 = {1,'a',3.5}
print ("Initially set is:",s1)

s1.add(5)
print ("set after added 5 is:",s1)

s1.add('a') # duplicate element. Won't be added again
print ("set after added a again is:",s1)

s3 = {7,9,'x'}
s1.update(s3)
print ("set after added another set is:",s1)

s1.remove(9);
print ("set after removing element 9 is:",s1)

#s1.remove(22) # it will returns KeyError because element does not exist
s1.discard(22) # it wont returns any error in case element does not exists.
print ("set after discarded element 22 is:",s1)
```

Output



```
D:\Youtube\Classes\python>python sets.py
Initially set is: {3.5, 1, 'a'}
set after added 5 is: {3.5, 1, 'a', 5}
set after added a again is: {3.5, 1, 'a', 5}
set after added another set is: {1, 3.5, 5, 7, 9, 'a', 'x'}
set after removing element 9 is: {1, 3.5, 5, 7, 'a', 'x'}
set after discarded element 22 is: {1, 3.5, 5, 7, 'a', 'x'}

D:\Youtube\Classes\python>
```

Union (|) , Intersection (&) , difference (-) & symmetric difference (^) on sets:

Union (|) will combine two sets as common sets which does not contains duplicate elements.

Intersection (&) will returns the common elements in both the sets.

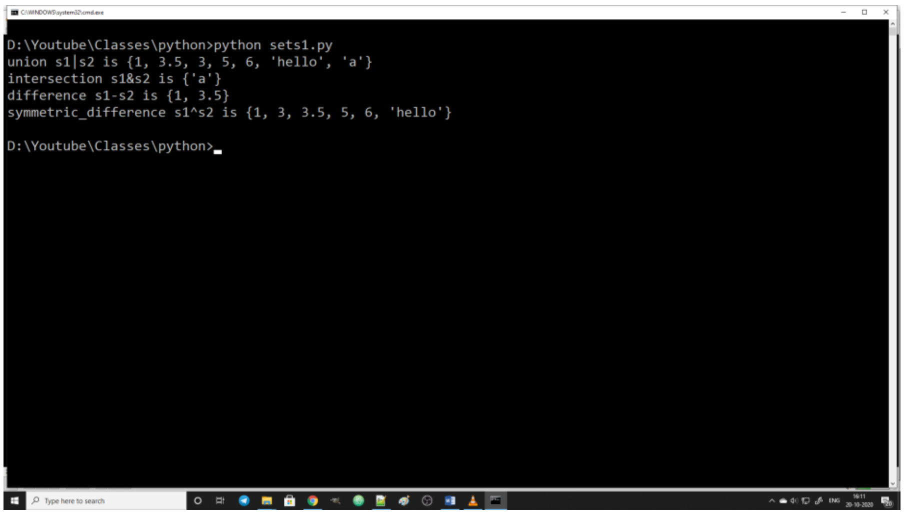
Difference (s1-s2) will returns all the elements that are in s1 but not in s2.

Symmetric Difference (s1^s2) returns all the elements from both the sets by excluding common elements.

Example :

```
#program to demonstrate set operations
s1 = {1,'a',3.5}
s2 = {'a',3,5,6,'hello'}
print ("union s1|s2 is",(s1|s2));
print ("intersection s1&s2 is",(s1&s2));
print ("difference s1-s2 is",(s1-s2));
print ("symmetric_difference s1^s2 is",(s1^s2));
```

Output



```
D:\Youtube\Classes\python>python sets1.py
union s1|s2 is {1, 3.5, 3, 5, 6, 'hello', 'a'}
intersection s1&s2 is {'a'}
difference s1-s2 is {1, 3.5}
symmetric_difference s1^s2 is {1, 3, 3.5, 5, 6, 'hello'}

D:\Youtube\Classes\python>
```

ACCESS MODIFIERS IN PYTHON

Most programming languages has three forms of access modifiers, which are **Public**, **Protected** and **Private** in a class that are used to allow or control access at various levels in a program.

```
class PythonTraining:

        def __init__(self,n,b,p):

                self.name = n      #public variable
                self._book=b       #protected variables are declared with single underscore
                self.__pen=p;      #private variable are declared with double underscore
```

public variables can be accessible from any program

protected variables can be accessible within the class & from sub classes of the class

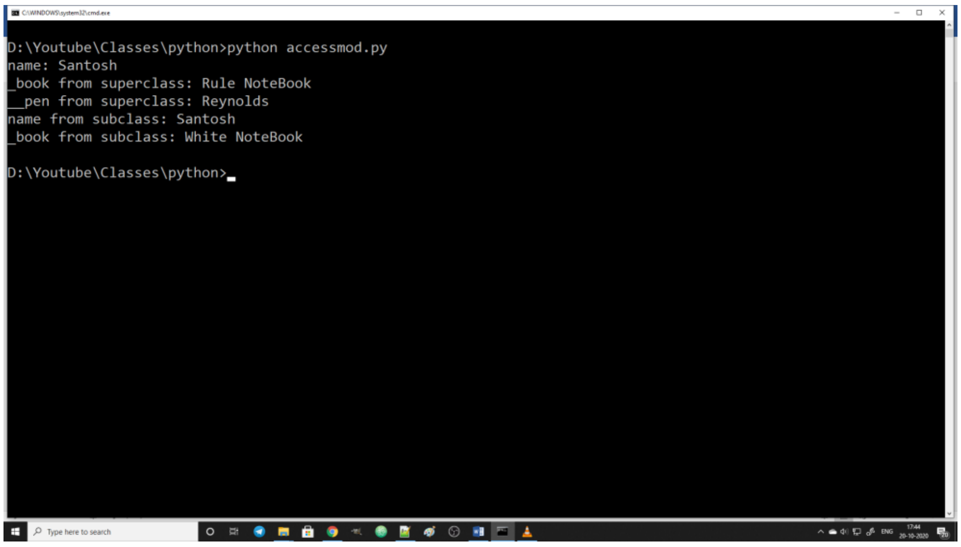
private variables can accessible only within the class where it is declared. Even sub classes of this class also cannot access private members of parent class.

Example :

```
class PythonTraining:
   def __init__(self,n,b,p):
               self.name = n   #public variable
               self._book=b    #protected variables are declared with single underscore
               self.__pen=p;   #private variable are declared with double underscore

   def setPen(self,p):
      self.__pen=p
   def getPen(self):
      return self.__pen

   def setBook(self,b):
      self._book=b
   def getBook(self):
      return self._book

   def setName(self,n):
      self.name=n
   def getName(self):
      return self.name

x = PythonTraining("Santosh","White NoteBook","Parker")
x.setBook("Rule NoteBook");
x.setPen("Reynolds");

print("name:",x.getName())
print("_book from superclass:",x.getBook())
print("__pen from superclass:",x.getPen())

class Child(PythonTraining):
   def __init__(self,n,b,p):
      super().__init__(n,b,p)

   def printAll(self):
      print("name from subclass:",self.name)
      print("_book from subclass:",self._book)
      #print("__pen from subclass:",self.__pen)  error because it is private
```

```
        y = Child("Santosh","White NoteBook","Parker")
        y.printAll();
```

output:



```
D:\Youtube\Classes\python>python accessmod.py
name: Santosh
_book from superclass: Rule NoteBook
__pen from superclass: Reynolds
name from subclass: Santosh
_book from subclass: White NoteBook

D:\Youtube\Classes\python>_
```

MULTI THREADING IN PYTHON

Process: Program in execution is said to be a process.

Modes of execution:

While executing multiple processes at a time operating system follows one of the following ways.

<u>Sequential Execution:</u> One after another - Second process must wait for the completion of first process

<u>Concurrent Execution:</u> Parallel Execution – Second process need not wait for the completion of first process

Thread: Single flow of control is said to be a thread.

While implementing concurrent programming we may develop different functions as different programs (multi programming) or as different threads under a single program (multi-threading).

Concurrent execution of multiple threads is said to be multi-threading whereas concurrent or parallel execution of multiple programs is said to be multi programming.

Both multi-threading and multiprogramming are different ways to implement multi-tasking. Multiprogramming is said to be heavy weight processing because here each function is implemented as a separate program where system allocates separate resources to each program. So, it occupies multiples of resources whenever compared with multithreading.

Multithreading is said to light weight processing because different functions will be under same program and these are implemented as threads under the program. So, system allocates resources to programs and not to the threads. So, with the resources of single program all the threads will finish their execution.

How to create a thread ?

1) Create a class that extends Thread

```
class thread1 (Thread):
```

2) Define run() method

```
def run(self):
    #code
```

3) Create object of the class and call start method

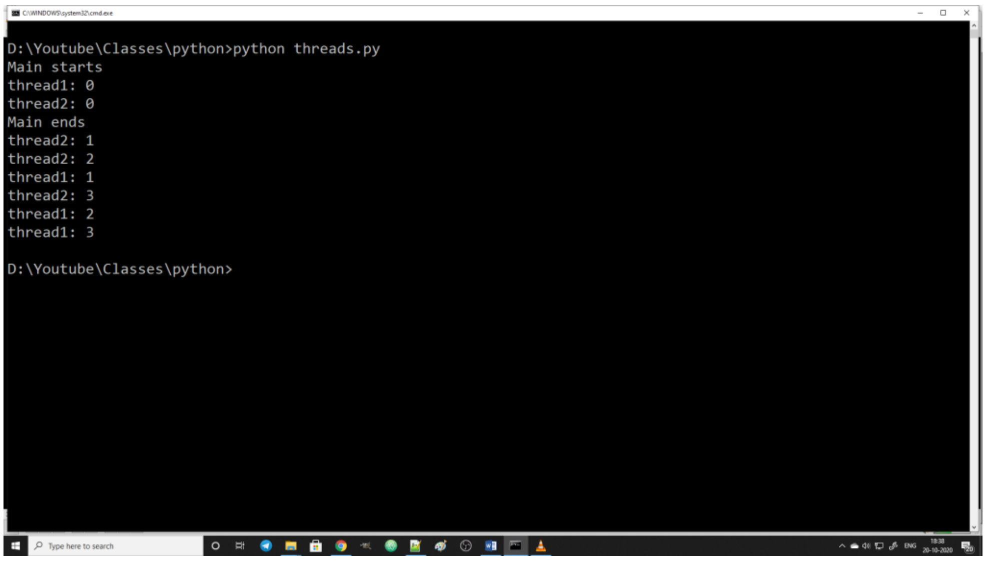
```
obj = thread1()

obj.start()    # calling run method of thread by using start() method
```

Example

```
#program to demonstrate multithreading in python

#step-1 : create a thread --> create subclass of thread
import threading.*
import time

print ("Main starts")
```

```
class thread1(threading.Thread):
   def run(self):#step-2 define run method
      for i in range(4):
         print ("thread1:",str(i))
         time.sleep(6)

class thread2(threading.Thread):
   def run(self):#step-2 define run method
      for i in range(4):
         print ("thread2:",str(i))
         time.sleep(2)

#step - 3 : create object and call the run method by using start method
t1 = thread1()
t1.start()

t2 = thread2()
t2.start()

print ("Main ends")
```

Output :



```
D:\Youtube\Classes\python>python threads.py
Main starts
thread1: 0
thread2: 0
Main ends
thread2: 1
thread2: 2
thread1: 1
thread2: 3
thread1: 2
thread1: 3

D:\Youtube\Classes\python>
```

<u>Join</u> <u>in thead :</u>

In order to stop execution of current program until a thread is complete, we use **join** method

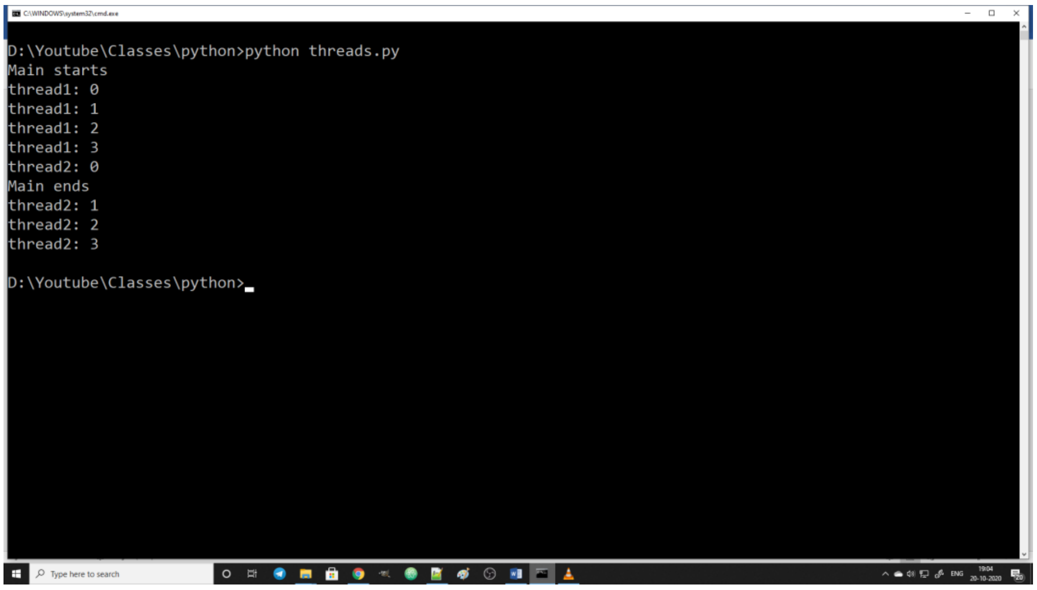
For example if we call t1.join() in above program all the other threads will wait until completion of t1 thread and then only they continue the execution.

Example :

```
#program to demonstrate multithreading in python

#step-1 : create a thread --> create subclass of thread
```

```
import threading
import time

print ("Main starts")

class thread1(threading.Thread):
    def run(self):#step-2 define run method
        for i in range(4):
            print ("thread1:",str(i))
            time.sleep(6)

class thread2(threading.Thread):
    def run(self):#step-2 define run method
        for i in range(4):
            print ("thread2:",str(i))
            time.sleep(2)

#step - 3 : create object and call the run method by using start method
t1 = thread1()
t1.start()
t1.join()

t2 = thread2()
t2.start()

print ("Main ends")
```

Output:



```
D:\Youtube\Classes\python>python threads.py
Main starts
thread1: 0
thread1: 1
thread1: 2
thread1: 3
thread2: 0
Main ends
thread2: 1
thread2: 2
thread2: 3

D:\Youtube\Classes\python>
```

In above program as t1 called join all the remaining threads paused their execution until t1 completes its execution process.

DATABASE CONNECTIVITY IN PYTHON

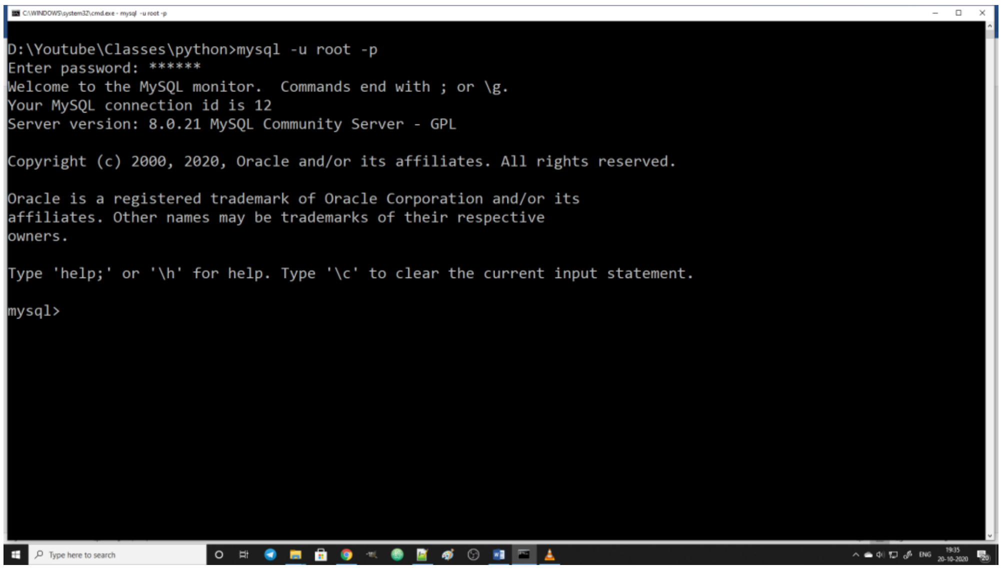
This concept allows us to write database queries in python programs. Then these queries will be sent to database server and execute there and get the results back from the database.

Steps to implement database connectivity:

1) Establish the connection between our python and database
2) Get the cursor object of the connection in order to execute queries in the database system and carry the results back to the python
3) Execute the queries
4) Close the connection

Please remember that we must close the connection in our program otherwise the concerned database may not give access to the other programs when open connections exceed the limit.

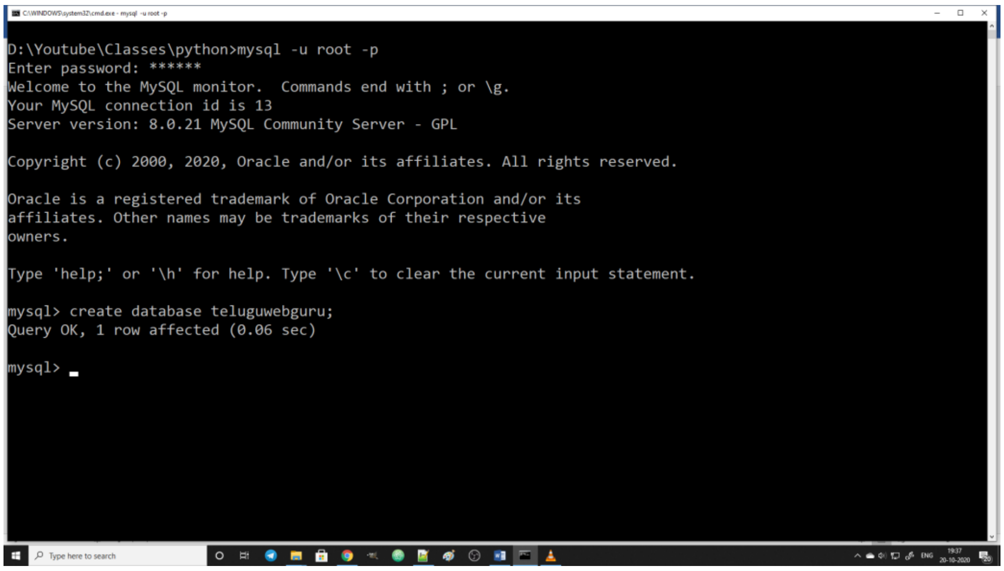
To implement this example first we need to install mysql and open it by using the command "mysql -u root -p"



```
D:\Youtube\Classes\python>mysql -u root -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.21 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Create the database called “teluguwebguru” by using the below command

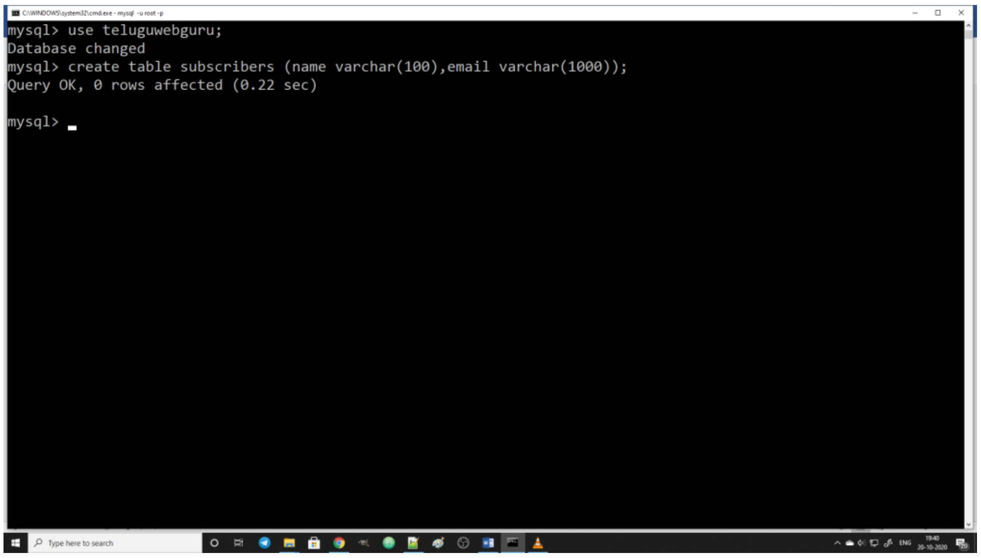
```
create database teluguwebguru;
```



```
D:\Youtube\Classes\python>mysql -u root -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.21 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database teluguwebguru;
Query OK, 1 row affected (0.06 sec)

mysql>
```

Create a table "subscribers"

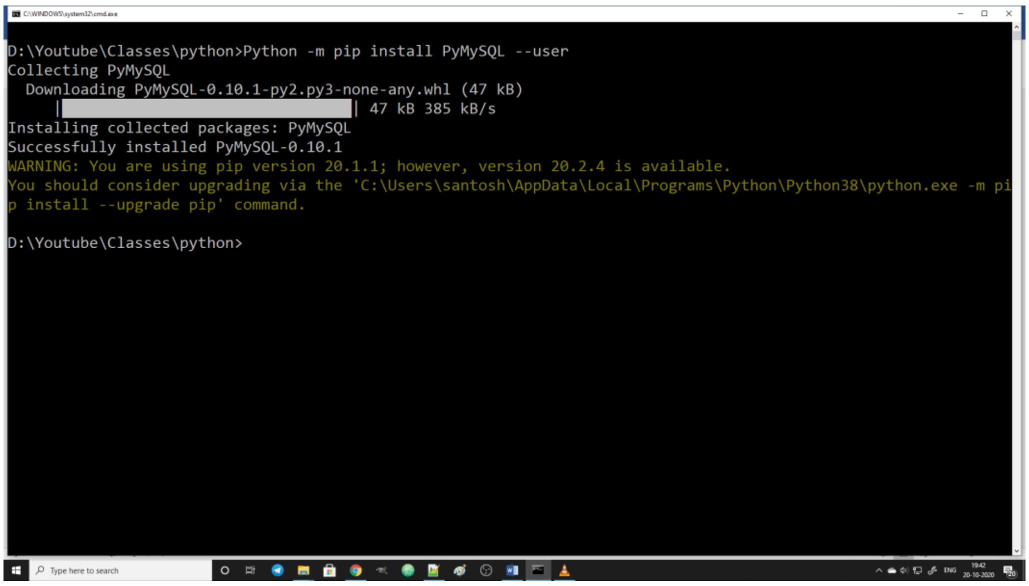
create table subscribers (name varchar(100),email varchar(1000));



```
mysql> use teluguwebguru;
Database changed
mysql> create table subscribers (name varchar(100),email varchar(1000));
Query OK, 0 rows affected (0.22 sec)

mysql>
```

Now we need to install pymysql module by using the following command

Python -m pip install PyMySQL --user

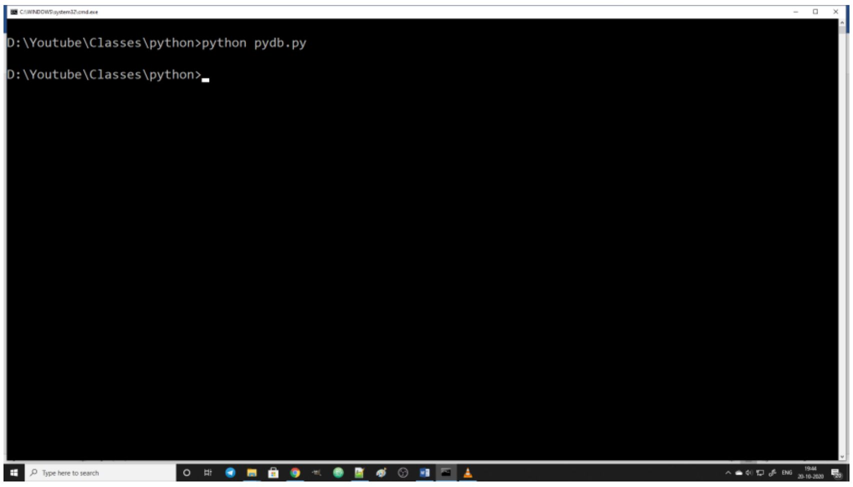


```
D:\Youtube\Classes\python>Python -m pip install PyMySQL --user
Collecting PyMySQL
  Downloading PyMySQL-0.10.1-py2.py3-none-any.whl (47 kB)
     |████████████████████████████████| 47 kB 385 kB/s
Installing collected packages: PyMySQL
Successfully installed PyMySQL-0.10.1
WARNING: You are using pip version 20.1.1; however, version 20.2.4 is available.
You should consider upgrading via the 'C:\Users\santosh\AppData\Local\Programs\Python\Python38\python.exe -m pi
p install --upgrade pip' command.

D:\Youtube\Classes\python>
```

Example:

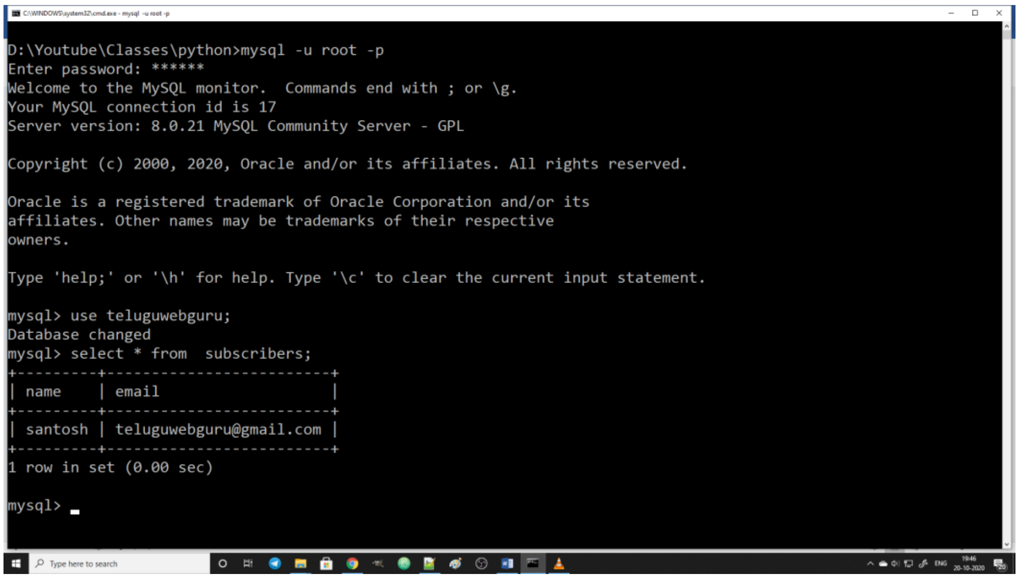
```
#program to demonstrate python database connectivity

import pymysql

#step-1 : Establish a connection
con = pymysql.connect(host="localhost",user="root",password="twg123",db="teluguwebguru")

#step -2 : get the cursor object
cur = con.cursor()

#step - 3 : execute the query
qry = "insert into subscribers values('santosh','teluguwebguru@gmail.com')";
cur.execute(qry)
con.commit()

#step -4 : close the connection
con.close()
```

Output:



```
D:\Youtube\Classes\python>python pydb.py

D:\Youtube\Classes\python>_
```

Now let us check in the database whether this query is executed successfully or not



```
D:\Youtube\Classes\python>mysql -u root -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 8.0.21 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use teluguwebguru;
Database changed
mysql> select * from  subscribers;
+---------+------------------------+
| name    | email                  |
+---------+------------------------+
| santosh | teluguwebguru@gmail.com |
+---------+------------------------+
1 row in set (0.00 sec)

mysql> _
```

Our query executed successfully and the row that we entered through our python program inserted into the database.

<u>Retrieving data from tables and print in python output:</u>

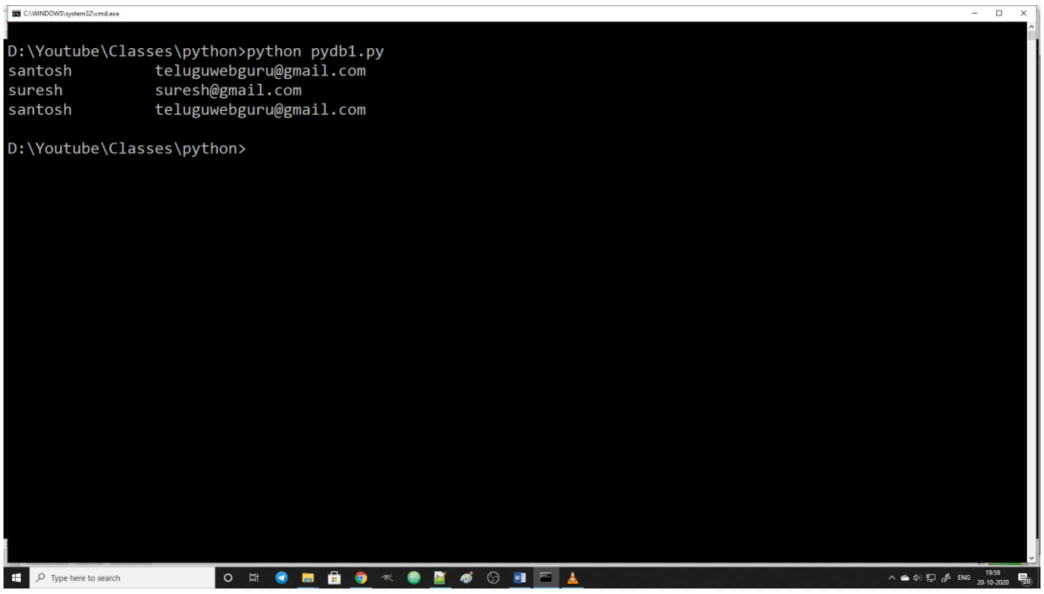
While retrieving data from the tables we use fetchone(), fetchall() methods to retrieve results from the database query execution process.

Fetchall() returns the result as list. You can print the entire list at a time by using print statement or you can run a loop and print them in a format as follows.

Example:

```
#program to demonstrate python database connectivity

import pymysql

#step-1 : Establish a connection
con = pymysql.connect(host="localhost",user="root",password="twg123",db="teluguwebguru")

#step -2 : get the cursor object
cur = con.cursor()

#step - 3 : execute the query
qry = "select * from subscribers";
cur.execute(qry)
subs = cur.fetchall()

for i in subs:
   print (i[0]+"\t\t"+i[1]);
   con.commit()

#step -4 : close the connection
con.close()
```

Output:



```
D:\Youtube\Classes\python>python pydb1.py
santosh         teluguwebguru@gmail.com
suresh          suresh@gmail.com
santosh         teluguwebguru@gmail.com

D:\Youtube\Classes\python>
```

In this way we can implement python database connectivity.