ignou
THE PEOPLE'S UNIVERSITY

Indira Gandhi National Open University
School of Computer and Information Sciences (SOCIS)

MCS-201 Programming in C and Python

Advanced features in python

4

MCS-201

Block

4

ADVANCED FEATURES IN PYTHON

BLOCK 4 INTRODUCTION

Python is the most popular high-level programming language that can be used for real-world programming. It is a dynamic and object-oriented programming language thatcan be used in a vast domain of applications, especially data handling. Python was designed as a very user-friendly general-purpose language with a collection of modules and packages and gained popularity in the recent times. Wheneverwe think of machine learning, big data handling, dataanalysis, statisticalalgorithms, python is the only name which comes first in our mind. It is a dynamically typed language, which makes it highly flexible. Furthermore, creating a database and linking of the database is very fast and secure.It makes the program to compile and run in an extremely flexible environment can support different styles of programming, including structural and object-oriented. Its ability to use modular components that were designed in other programming languages allows a programmer to embed the code written in python for any interface. Due to its enormous features, python has become the first choice of learners in the field of data science and machine learning.

This block is specially designed for beginners to understand the concepts of object oriented programming in python and extend the understanding to create and connect database in python. You will learn to connect data through SQL as well as CSV files with the help of real dataset (PIMA Indian Dataset), used most commonly by researchers and academia in data analysis.

This block consists of 4 units and is organized as follows:

Unit - 13Introduces you about the Concept of Classes in Python

Unit - 14Makes you to understand the concept of exception handling in Python

Unit – 15 provides understanding of decorators, iterators, generators and co-routines in Python.

Unit - 16 introduces you to work with databases and datasets in Python.

Happy Programming!!

MCS-201
PROGRAMMING IN
C AND PYTHON

Block

4

ADVANCED FEATURES IN PYTHON

BLOCK PREPARATION TEAM

Dr.Sudhansh Sharma, Assistant Professor
School of Computers and Information Sciences,
IGNOU, New Delhi*(Writer Unit-15)*

Dr.Parmod Kumar, Assistant Professor (Sr.G) ,
Department of Computer Applications, SRM
Institute of Science and Technology, Delhi
NCR Campus, Modinagar, U.P.*(Writer Unit-13 & 14)*

Ms.Sofia Goel, Research Scholar
School of Computers and Information Sciences,
IGNOU,NewDelhi*(Writer Unit-16)*

Prof S.R.N. Reddy, *(Content Editor)*
HOD, Dept. of CSE,
IGDTUW, Delhi

Prof.Parmod Kumar *(Language Editor)*
School of Humanities, IGNOU,
New Delhi *(Unit-15)*

Dr.DevenderTanwar,*(Language Editor)*Associate Professor,Department of English, AryabhattCollge Delhi University*(Unit-13,14,& 16)*

Course Coordinator: Dr.Sudhansh Sharma

PRINT PRODUCTION

Mr.Tilak Raj
Assistant Registrar (Publication)
MPDD, IGNOU, New Delhi

Mr.Yashpal
Assistant Registrar (Publication)
MPDD, IGNOU, New Delhi

March, 2021

© Indira Gandhi National Open University, 2021

ISBN-

Printed and published on behalf of the Indira Gandhi National Open University by Registrar, MPDD, IGNOU, New Delhi

Laser Composed by Tessa Media & Computers, C-206, ShaheenBagh, Jamia Nagar, New Delhi

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at MaidanGarhi, New Delhi-110 068.

UNIT 13 CLASSES IN PYTHON

Structure

13.0 INTRODUCTION TO OBJECT ORIENTED PARADIGMS

Object-oriented programming is a programming paradigm which define a class(a group of similar types of objects)and objects (containing both data and methods) to achieve the modularity and portability of the various components of the programs.

In Python programming language, we can define new classes. These classes are customized to a particular application which helps the user to perform its task in an easy way and maintain all the things required in the application.

The reason to define a new class helps in solving the structured application program in terms of object-oriented programming.In structured programming, language functions are defined. These functions are used by the users, but the implementations are hidden by the users. Similarly, a class defines methods and these methods are used by the users but how these methods are implemented are hidden from the users. A customized namespace is associated with every class and objects.

13.1 OBJECTIVES

After going through this unit you will be able to :

- Understand concepts of Object Oriented programming in Python
- Create your own Classes and Objects of classes
- Apply the concept of Inheritance
- Understand the concept of Overloading and Overriding

13.2 CLASSES AND INSTANCES

Classes

A class is a group of similar types of objects.For example, a university is a class, and various objects of the class are open_university, government_university, central_university, state_private_university, private_university, deemed_university.

To create a class we use keyword 'class'. Following is the syntax of class:

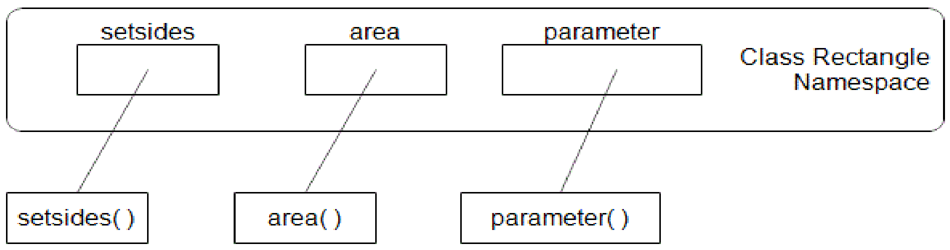
```
Classclass_name:
        class_variable1= value_of_variable
        class_variable2= value_of_variable
        class_variable3= value_of_variable
        …
        defclass_method1(self,arg1,arg2,…):
                method1_code
        def class_method2(self,arg1,arg2,…):
                method2_code
        …
```

The first class we defined is a Rectangle.

```
>>>
class Rectangle:
        defsetsides (self,x,y):
                self.height=x
                self.width=y
                print('The sides of the rectangle are {} and
{}:'.format(self.height,self.width))
        def area(self):
                return(self.height*self.width)
        def parameter(self) :
                return(2*(self.height*self.width))
```

It's a convention to write the name of a class with the first character in a capital letter. But it supportsa small letter also. After writing the name of a class,it must be preceded by a colon ':'. In C++ or Java, we were using pair brackets () but python-support colon ':' only.The statement is written after the colon ':' will be taken by the Python interpreter as part of the class document.

When we create a class,a namespace is created for the class Rectangle. This namespace store all the attributes of the class Rectangle. This namespace will specify the names of class Rectangle method.



Object

In Python, whatever value we are storing everything is taken as an object. For example, the string value 'IGNOU University' or the list ['IGNOU',22] or the integer value 28 all are stored in memory as an object. We can think object as a container which stores value in computer memory (RAM).Objects are the containers used to store the values, and it hides the details of the storage from the programmers and only gives the information which is required during the implementation.
Each object contains two things: *types*and*values*.
The type specifies what type of values can be assigned to object or the type of operations that can be operated by the objects.
>>>object1=Rectangle()

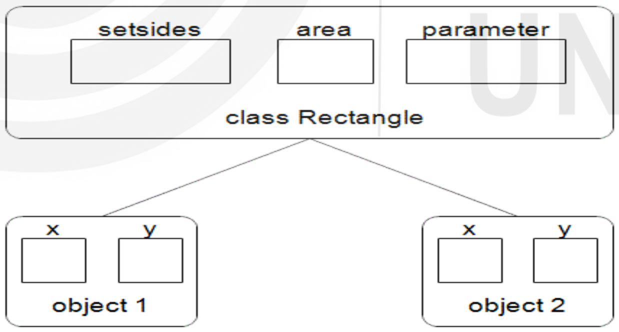
In Python, a namespace is created for each class. Each namespace is specified with a name, and this name is same asclass name—all the attributes of the class use this namespace for the storage.Thus in our example namespace Rectangle must contain names of all the class methods.

Whenever an object object1is created of the class Rectangle(),the separate namespace is created for the object1 also.

Object Object1 namespace

Let us describe the various methods of the class Rectangle:

setsides(x,y) :	method to describe the sides of rectangle (height and width).
area() :	method which return the area of rectangle.
parameter() :	method which return the parameter of rectangle.



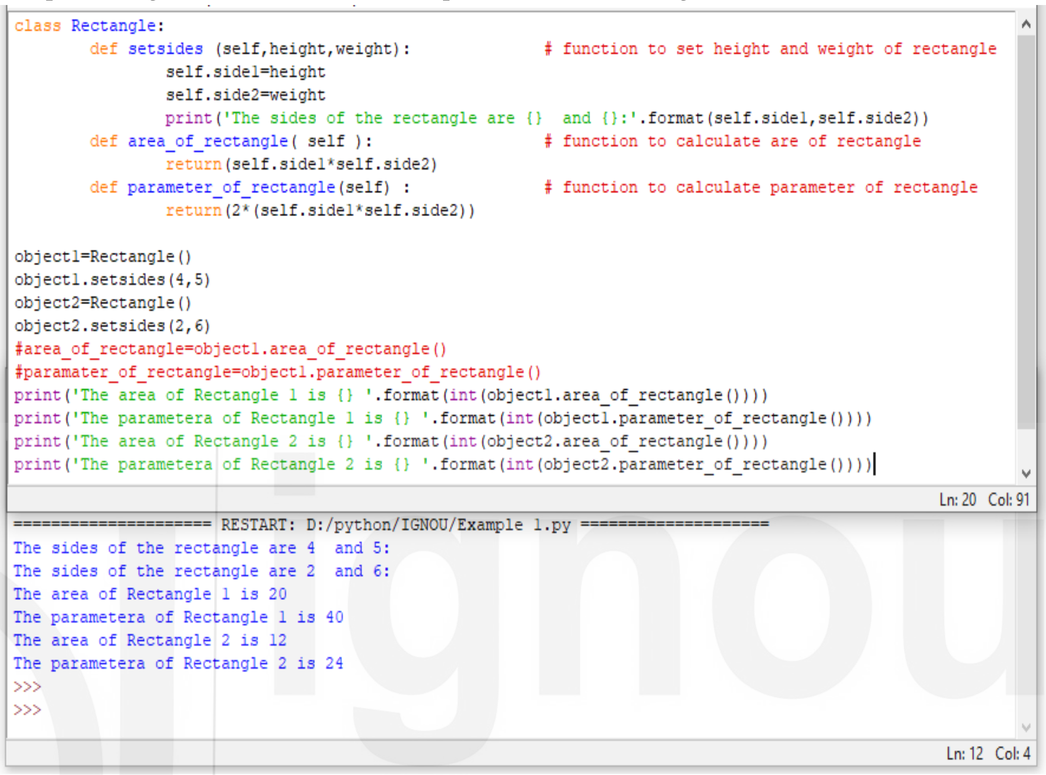
The function setsides(),area() and parameter() are defined in the name space. The syntax of methodwould be:

```
defsetsides(self, x,y):
        # implementation of setsides()
def area():
        # implementation of area()
def parameter():
        # implementation of parameter()
```

Instance

Variables that point to the object namespace is called instance variables. Each instance variable containsa different value for different objects. All the variable define inside the __init__valiable are called as instance variables. And all the variable defined inside the class but outside the __init__valiable is called class variables.A class variable is also called as static variables.

Eample1: Program to calculate area and parameter of a rectangle.

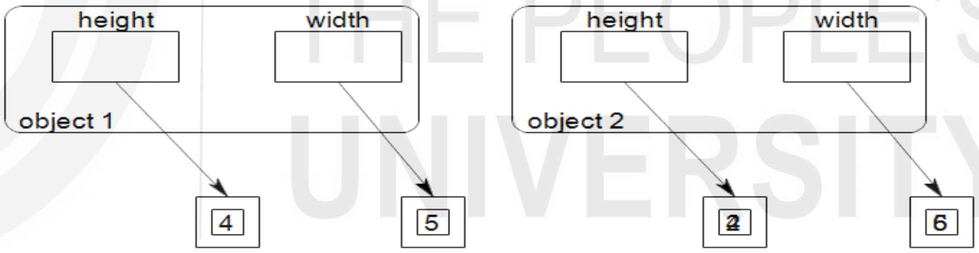


```
class Rectangle:
        def setsides (self,height,weight):          # function to set height and weight of rectangle
                self.side1=height
                self.side2=weight
                print('The sides of the rectangle are {}  and {}:'.format(self.side1,self.side2))
        def area_of_rectangle( self ):              # function to calculate are of rectangle
                return(self.side1*self.side2)
        def parameter_of_rectangle(self) :          # function to calculate parameter of rectangle
                return(2*(self.side1*self.side2))

object1=Rectangle()
object1.setsides(4,5)
object2=Rectangle()
object2.setsides(2,6)
#area_of_rectangle=object1.area_of_rectangle()
#paramater_of_rectangle=object1.parameter_of_rectangle()
print('The area of Rectangle 1 is {} '.format(int(object1.area_of_rectangle())))
print('The parametera of Rectangle 1 is {} '.format(int(object1.parameter_of_rectangle())))
print('The area of Rectangle 2 is {} '.format(int(object2.area_of_rectangle())))
print('The parametera of Rectangle 2 is {} '.format(int(object2.parameter_of_rectangle())))
```

Ln: 20 Col: 91

```
==================== RESTART: D:/python/IGNOU/Example 1.py ====================
The sides of the rectangle are 4  and 5:
The sides of the rectangle are 2  and 6:
The area of Rectangle 1 is 20
The parametera of Rectangle 1 is 40
The area of Rectangle 2 is 12
The parametera of Rectangle 2 is 24
>>>
>>>
```

Ln: 12 Col: 4

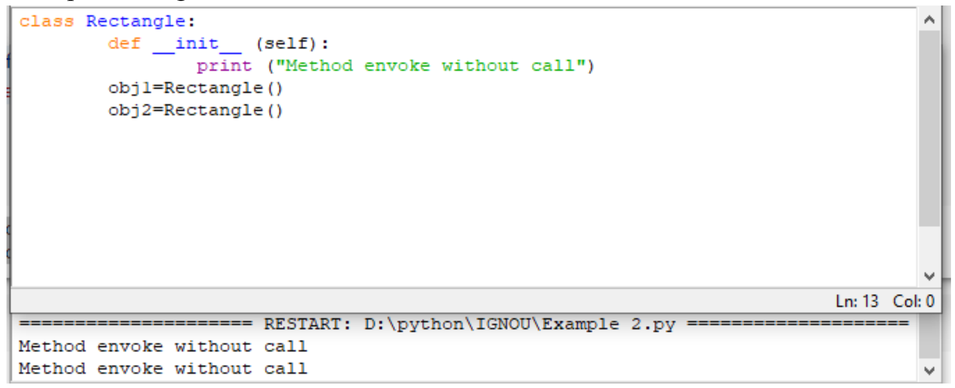


13.3 CLASSES METHOD CALLS

To call a method of a class first, we have to create an object of the same class. After object creation, we will envoke the method defined in a class with the dot '.' operator.

When we create an object of the class, the constructors declared in the classenvoke automatically to which it belongs. In Python, there is a unique method _ _init_ _ to implement constructor of the class in which it is defined. The first argument of the method _ _init _ _ must be self. Self is a reference to a class to which this object belongs.

Example 2: Program to call a constructor



```
class Rectangle:
        def __init__ (self):
                print ("Method envoke without call")
        obj1=Rectangle()
        obj2=Rectangle()
```

```
==================== RESTART: D:\python\IGNOU\Example 2.py ====================
Method envoke without call
Method envoke without call
```

When two objects obj1 and obj2 are created __init__ method is called automatically.

Class	Methods
Predefined word *class*is used to describe a class.	Predefined word *def*is used to describe the method of a class.
Each class statement defines a new type with a given name.	A def statement defines a new function with a given name.
Each class name is preceded by a colon ':' statement.	Each method name is preceded by a colon ':' statement.
Example -- class Rectangle:	Example -- defsetsides(self):

Table.13.1 Comparison between class and method

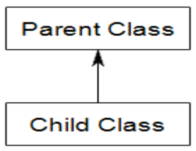
13.4 INHERITANCE AND COMPOSITIONS

Inheritance is the mechanism by which object acquire the property of another object derived in a hierarchy.

The class which is inherited is called a base class or parent class or superclass, and the inheriting class is called the derived class or child class or subclass.For our reference, we will use the term parent class and child class. A child class acquire all the properties of the parent class, but the vice versa is not true; parent class can't access any features of a child class. The main motto behind the inheritance is code reusability. Once a code is defined in a class if it is required by another class then by inheriting the class code can be reused.

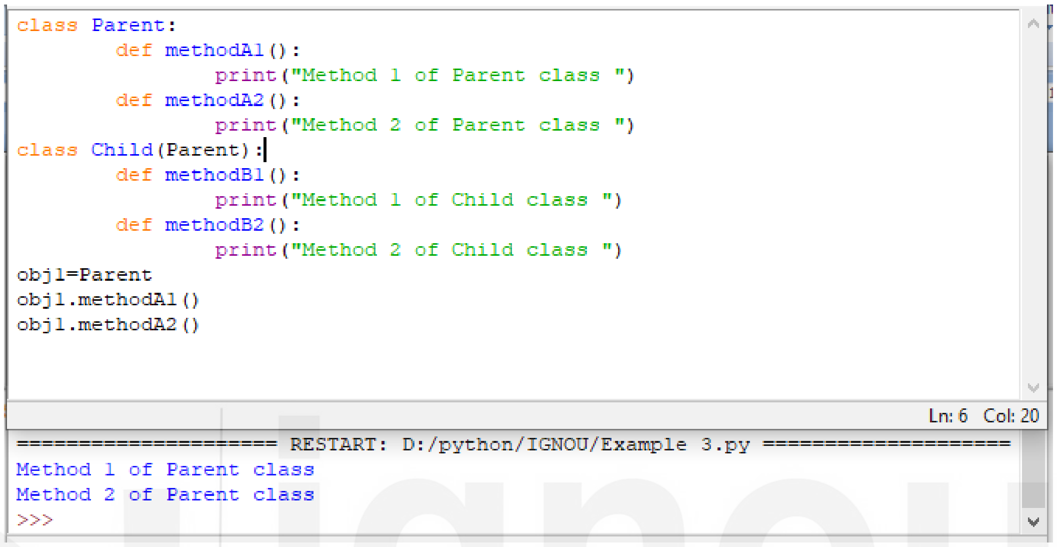
Syntax of inheritance:

class<Child Class>:
If this child class inherits the parent class, then the statement will be changed to
class<Child Class> (<Parent class>):



Consider aclass Parent containing two methods methodA1() and methodA2() and a class Child with method methodB1() and methodB2().Child class is defined to be the subclass of Parent class.Therefore it inherits both the methods methodA1() and methodA2() of Parent class .

Example 3: Inheritance of a parent class by child class

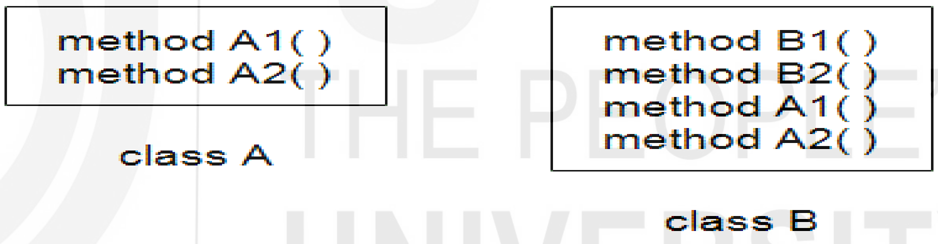


```
class Parent:
        def methodA1():
                print("Method 1 of Parent class ")
        def methodA2():
                print("Method 2 of Parent class ")
class Child(Parent):
        def methodB1():
                print("Method 1 of Child class ")
        def methodB2():
                print("Method 2 of Child class ")
obj1=Parent
obj1.methodA1()
obj1.methodA2()
```

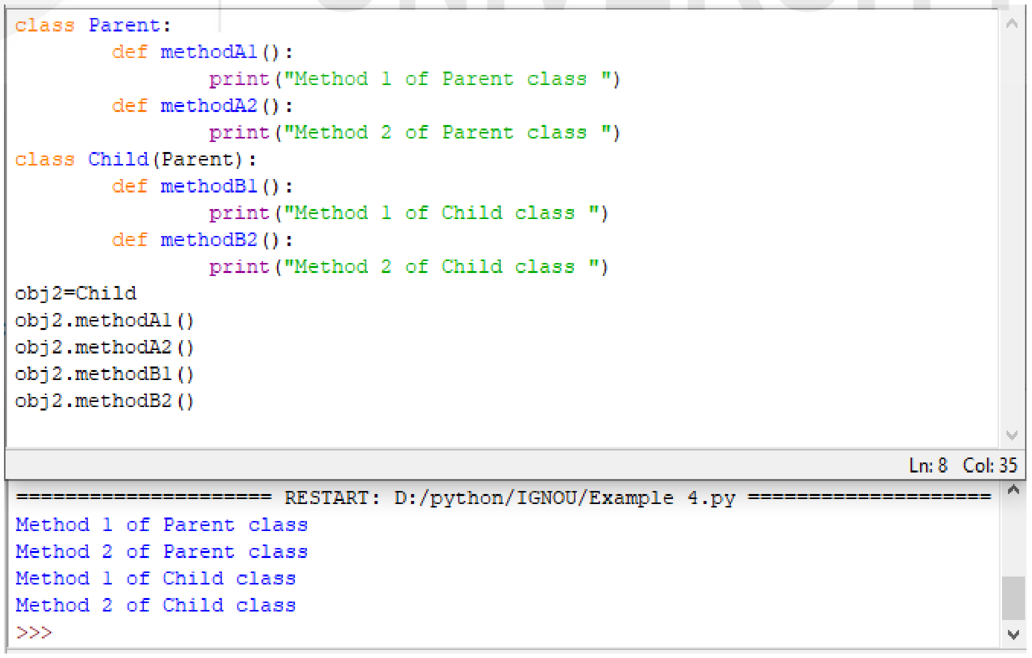
Ln: 6   Col: 20

```
===================== RESTART: D:/python/IGNOU/Example 3.py =====================
Method 1 of Parent class
Method 2 of Parent class
>>>
```

After inheritance class child contains methods methodB1() and methodB2() as well as methods of class Parent , methodA1() and methodA2().



Example 4: Calling method of parent class by object of child class.



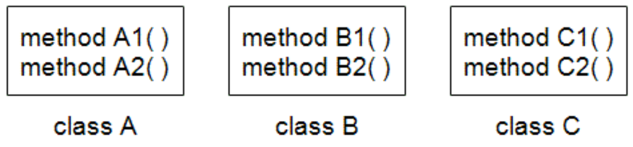
```
class Parent:
        def methodA1():
                print("Method 1 of Parent class ")
        def methodA2():
                print("Method 2 of Parent class ")
class Child(Parent):
        def methodB1():
                print("Method 1 of Child class ")
        def methodB2():
                print("Method 2 of Child class ")
obj2=Child
obj2.methodA1()
obj2.methodA2()
obj2.methodB1()
obj2.methodB2()
```

Ln: 8   Col: 35

```
===================== RESTART: D:/python/IGNOU/Example 4.py =====================
Method 1 of Parent class
Method 2 of Parent class
Method 1 of Child class
Method 2 of Child class
>>>
```

Multilevel inheritance:

Consider three classes class A,class B and class C

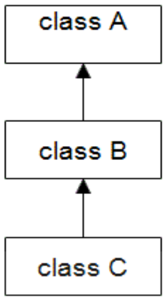


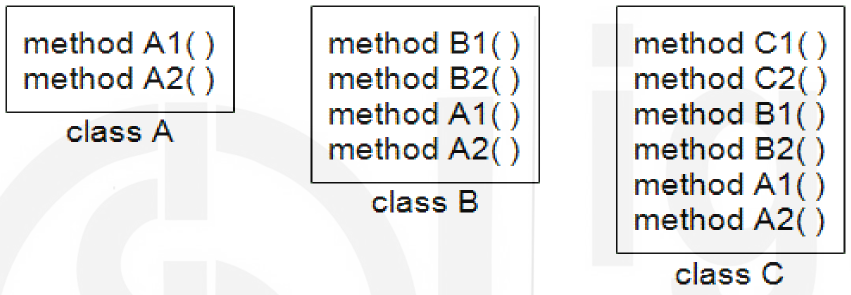
When an inherited class is inherited by another class, this is called multilevel inheritance.

Consider Class B inherits class A, and class C inherits class B.

Class B is called subclass of Class A,and Class C is called subclass of class B.

Class C willcontain all the attributes and methods of class A and class B.

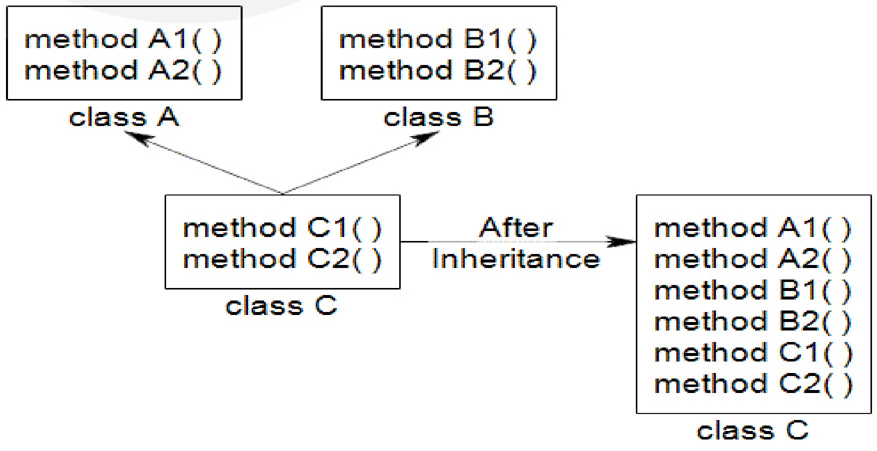




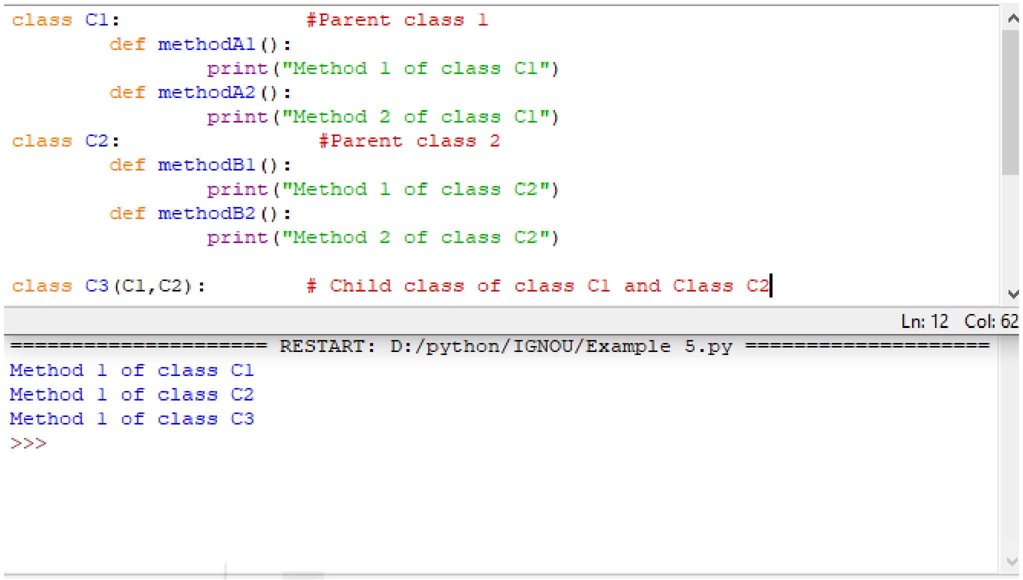
Multiple Inheritance:

When a single class inherit two or more classes,then it is called multiple inheritances.

Consider two classes, class A with two methods methodA1(),methodA2() and class B with methodB1() and methodB2(). If another class C inherits both class A and class B,then class B will inherit the features of class A and B.



Example 5: Implementation of multiple inheritance

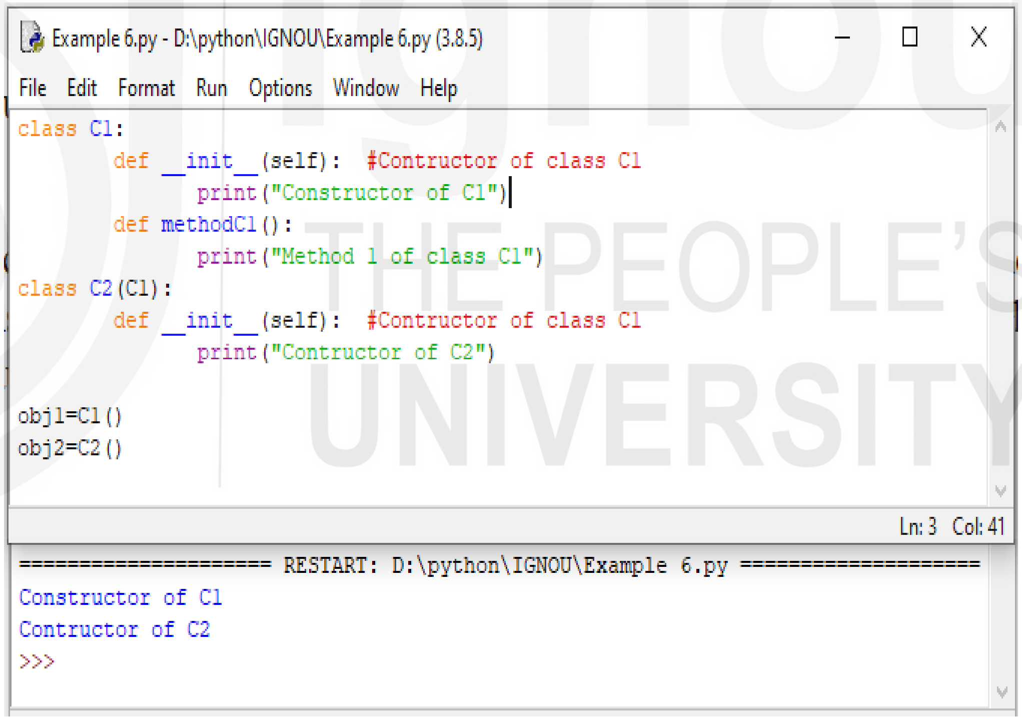


```
class C1:                #Parent class 1
        def methodA1():
                print("Method 1 of class C1")
        def methodA2():
                print("Method 2 of class C1")
class C2:                 #Parent class 2
        def methodB1():
                print("Method 1 of class C2")
        def methodB2():
                print("Method 2 of class C2")

class C3(C1,C2):         # Child class of class C1 and Class C2
```

```
==================== RESTART: D:/python/IGNOU/Example 5.py ====================
Method 1 of class C1
Method 1 of class C2
Method 1 of class C3
>>>
```

The behaviour of constructor in inheritance:

Example 6: Constructor of class



Example 6.py - D:\python\IGNOU\Example 6.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 3 Col: 41

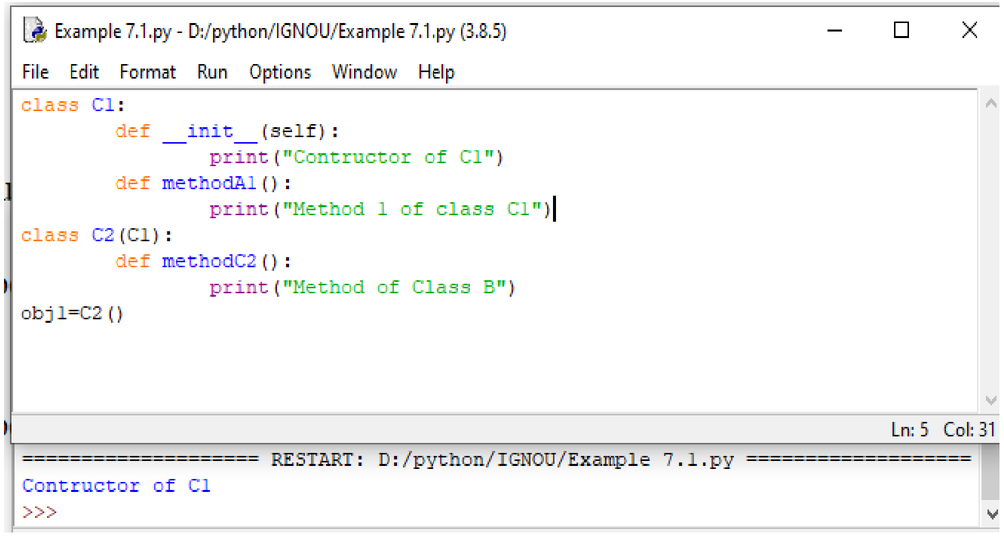
```
class C1:
        def __init__(self):  #Contructor of class C1
                print("Constructor of C1")
        def methodC1():
                print("Method 1 of class C1")
class C2(C1):
        def __init__(self):  #Contructor of class C1
                print("Contructor of C2")

obj1=C1()
obj2=C2()
```

```
==================== RESTART: D:\python\IGNOU\Example 6.py ====================
Constructor of C1
Contructor of C2
>>>
```

Since the creation of an object of a class will automatically invoke the constructor of the class. Here in the example class C1 object will call a constructor of class C1 automatically, and creation of an object of class C2 will call the constructor of class C2 automatically.

Creating an object of child classwill first search the__init__method of the child class. If __init__ method is in the child class then it will be executed first if it is not in the child class then it will go to the __init__ method of the parent class.

Example 7:

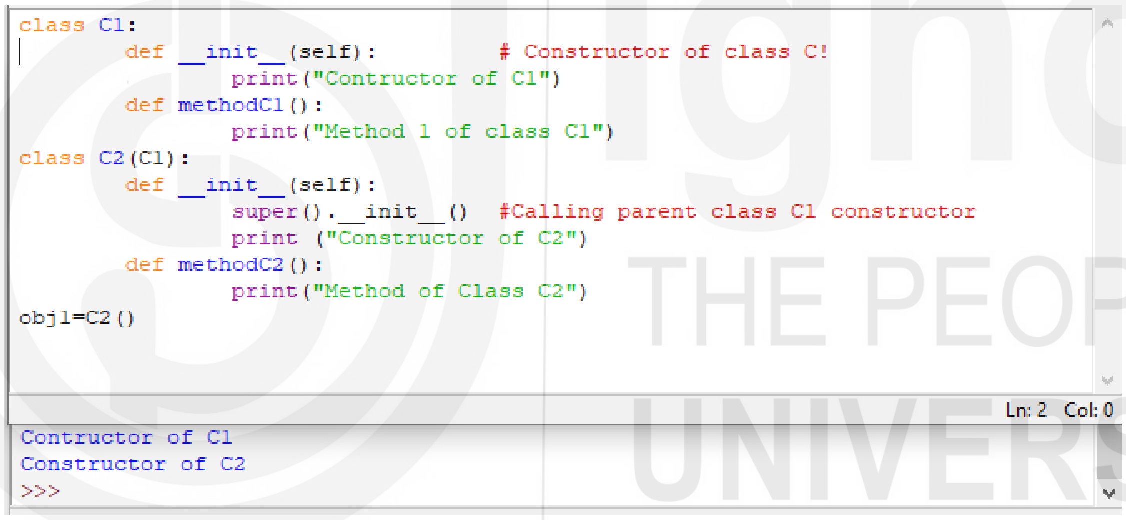

Example 7.1.py - D:/python/IGNOU/Example 7.1.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 5 Col: 31

```
class C1:
        def __init__(self):
                print("Contructor of C1")
        def methodA1():
                print("Method 1 of class C1")
class C2(C1):
        def methodC2():
                print("Method of Class B")
obj1=C2()
```

```
==================== RESTART: D:/python/IGNOU/Example 7.1.py ===================
Contructor of C1
>>>
```

If we want to call the init method of the parent class also then we can call it with the help of keyword super.

Example 8: Use of super keyword to call a method of the parent class.



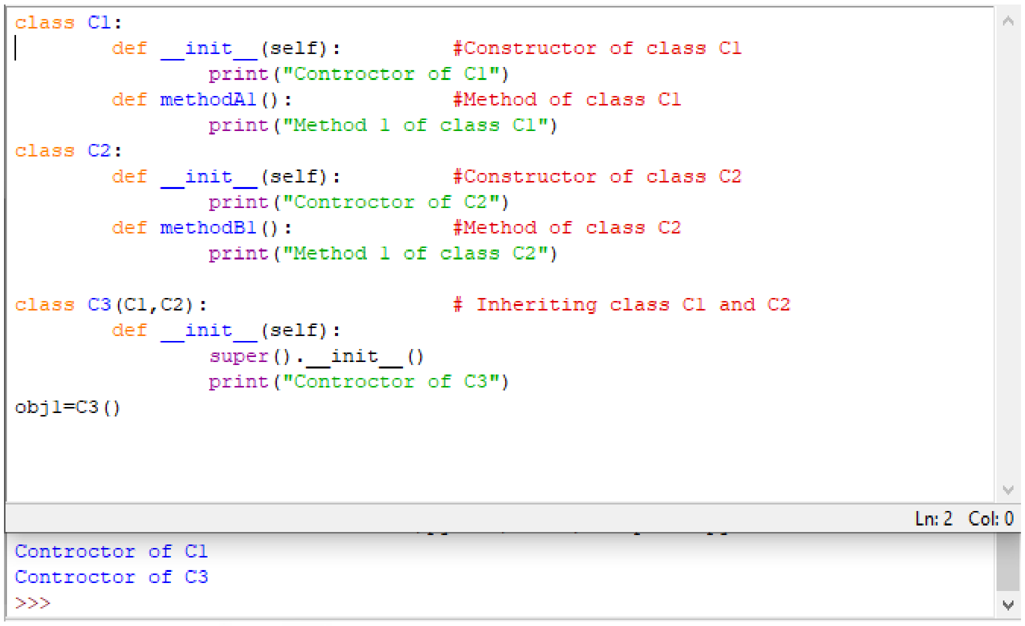
```
class C1:
        def __init__(self):          # Constructor of class C!
                print("Contructor of C1")
        def methodC1():
                print("Method 1 of class C1")
class C2(C1):
        def __init__(self):
                super().__init__()  #Calling parent class C1 constructor
                print ("Constructor of C2")
        def methodC2():
                print("Method of Class C2")
obj1=C2()
```

```
Contructor of C1
Constructor of C2
>>>
```

Thus when we create an object of the child class, it will call the init method of the child class first. If we have called super, then it will first call init of the parent class then it will call int of the child class.

If we have two classes, A and B inherited by class C. If init method is defined in all three classes then what will happen if call inits of the parent class with the help of super() keyword? Then which class init method will be called?

Example 9: Uses of the super keyword in multiple inheritances.



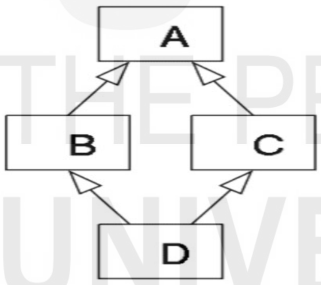
```
class C1:
        def __init__(self):          #Constructor of class C1
                print("Contrctor of C1")
        def methodA1():              #Method of class C1
                print("Method 1 of class C1")
class C2:
        def __init__(self):          #Constructor of class C2
                print("Contrctor of C2")
        def methodB1():              #Method of class C2
                print("Method 1 of class C2")

class C3(C1,C2):                     # Inheriting class C1 and C2
        def __init__(self):
                super().__init__()
                print("Contrctor of C3")
obj1=C3()
```

```
Contrctor of C1
Contrctor of C3
>>>
```

It will first call the init method of the child class then with the help of super() keyword it will call the init method of the left parent class that is A in our example.

In multiple inheritances, when we inherit the classes, then the method is executed based on the order specified, and this is called Method Resolution Order (MRO).

If class A is inherited by class B and Class C and then class D inheriting class B and class C.



$$\begin{tikzcd} & A & \\ B \arrow[ur] & & C \arrow[ul] \\ & D \arrow[ul] \arrow[ur] & \end{tikzcd}$$

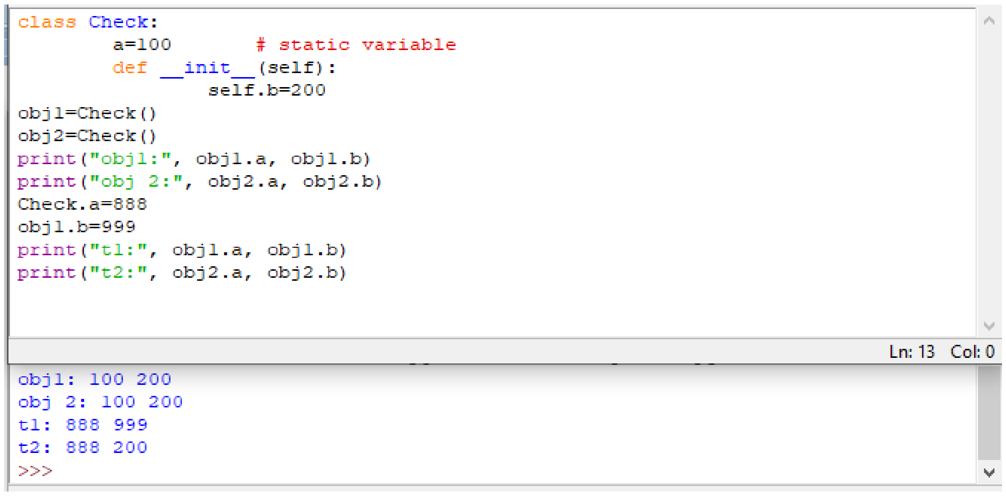
Then according to the Method of Resolution Order (MRO) the orderof execution of the methods and attributes are: class D→class B→class C→class A

13.5 STATIC AND CLASS METHODS

A static variable is that variable whose value remainsthe same for all the objects of the class. It creates one copy of the variable which is shared by all objects of the class. To declare a static variable, it must be declared inside the class. No method is used to declare a static variable it is declared without any method.

To use the static variable, we will use the class name to which this static variable belong, or we will use the reference of the object. But it is always preferred to use class name in place of reference of the object.

Example 10: Implementation of static variable.



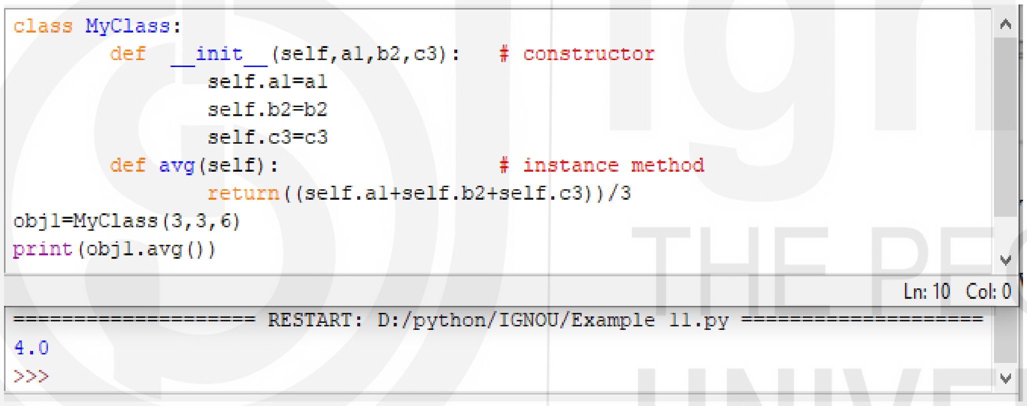
```
class Check:
        a=100        # static variable
        def __init__(self):
                self.b=200
obj1=Check()
obj2=Check()
print("obj1:", obj1.a, obj1.b)
print("obj 2:", obj2.a, obj2.b)
Check.a=888
obj1.b=999
print("t1:", obj1.a, obj1.b)
print("t2:", obj2.a, obj2.b)
```

```
obj1: 100 200
obj 2: 100 200
t1: 888 999
t2: 888 200
>>>
```

In Python, there are three types of methods: Instance methods, class methods and static methods.

Instance methods:

Example 11: Implementation of an instance method.

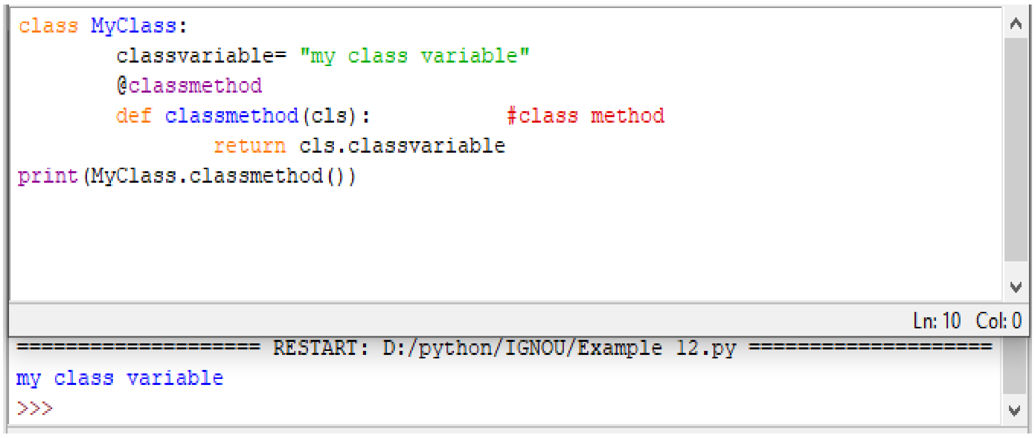


```
class MyClass:
        def  __init__(self,a1,b2,c3):   # constructor
                self.a1=a1
                self.b2=b2
                self.c3=c3
        def avg(self):                  # instance method
                return((self.a1+self.b2+self.c3))/3
obj1=MyClass(3,3,6)
print(obj1.avg())
```

```
==================== RESTART: D:/python/IGNOU/Example 11.py ====================
4.0
>>>
```

In the above example first method __init__ is a constructor of the class. The second method avg(self)is the instance method of the class.This method contains one parameter self, which points to the instance of the
classMyClass, 3.0 will be printed as an output.
When the method is called Python, replace the self argument with the instance of the object.Thus instance method always works on the object.

Class method



```
class MyClass:
        classvariable= "my class variable"
        @classmethod
        def classmethod(cls):           #class method
                return cls.classvariable
print(MyClass.classmethod())
```

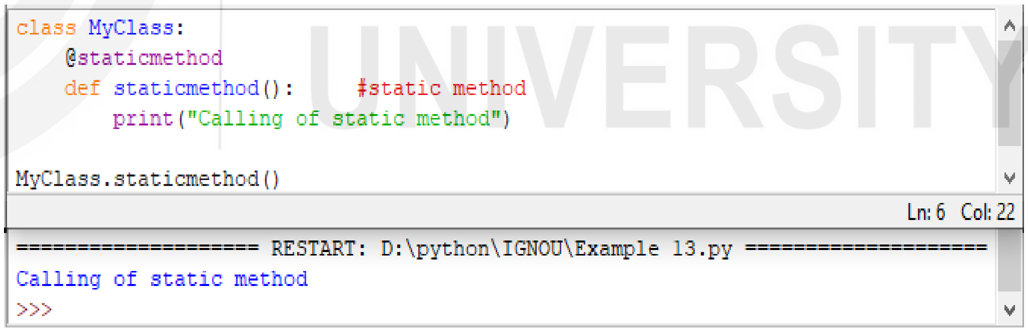
```
==================== RESTART: D:/python/IGNOU/Example 12.py ====================
my class variable
>>>
```

A class method is a method which is bound to the class and not the object of the class. A special symbol called a decorator ' @' followed by the keyword classmethod is used to define the class method.

A class method can be called by the class or by the object of the class to which this method belongs. The first parameter of the class method is class itself.Thus as an instance method is used to call the instance variable similarly class method is used to call class variables.

Static method

Suppose we are looking to a method which is not a concern to the instance variable neither to the class variable. In that case, we will use a static method.Static method in Python isused when such methods are called another class or methods are used to perform some mathematic calculations based on the values received as an argument. A special symbol called as a decorator '@' is used followed by the keywordstaticmethod is used to define a static method. Thus a static method can be invoked without the use of the object of the class.



```
class MyClass:
    @staticmethod
    def staticmethod():     #static method
        print("Calling of static method")

MyClass.staticmethod()
```

```
==================== RESTART: D:\python\IGNOU\Example 13.py ====================
Calling of static method
>>>
```

Thus if we want to work with the variables other then class variable and instance variable, we will use static.

13.6 OPERATOR OVERLOADING

Consider an operator '+.'

>>>2+5

7

Here it performs arithmetic addition of two numbers.

```
>>>[2,3,4]+[5,6]

[2,3,4,5,6]
```

Here it performs concatenation of two lists.

```
>>> 'IGNO'+ 'U University'

IGNOU University
```

Here it performs concatenation of two strings.

The operator '+' is said to be an overloaded operator. If an operator is defined for more than one classes, then it is called the overloaded operator. For each class, the implementation of the overloaded operator is different. In our example, the overloaded operator ' +' is defined for the in-class,list class and string class.
The Python interpreter will take the operator '+' as x._ _ add_ _(y), and this is called method invocation.
_ _add_ _(..)

x._ _ add_ _(y) equivalent to x+y
Thus when we are performing 2+5, it is

```
>>int(2)._ _ add(5)
7
>>>[2,3,4]._ _add_ _[5,6]
[2,3,4,5,6]

>>> 'IGNO'. _ _add_ _ 'U University.'
'IGNOU University'
```

Operator	Method	Number	List &String
n1 + n2	n1.__add__(n2)	Adding n1&n2	Concatenation
n1–n2	n1.__sub__(n2)	Subtracting n2 from n1	—
n1 * n2	n1.__mul__(n2)	Multiplyingn1 & n2	Self concatenation
n1 / n2	n1.__truediv__(n2)	Dividing n1 by n2	—
n1 // n2	n1.__floordiv__(n2)	Integer division of n1 by n2	—
n1 % n2	n1.__mod__(n2)	Remainder after division of n1 by n2	—
n1 == n2	n1.__eq__(n2)	n1 & n2 both are same	
n1 != n2	n1.__ne__(n2)	n1 & n2 both are different	
n1>n2	n1.__gt__(n2)	n1 is larger than n2	
n1 >= n2	n1.__ge__(n2)	n1 is large than n2or equal to n2	
n1 < n2	n1.__lt__(n2)	n1 is small than n2	
n1<= n2	n1.__le__(n2)	n1 is small than or equal to n2	
repr(n1)	n1.__repr__()	Canonical representation of string n1	
str(n1)	n1.__str__()	Informal representation of string n1	
len(n1)	n1.__len__() —	Size of n1	
<type>(n1)	<type>.__init__(n1)	Constructor	

Table 13.2: Overloaded operators

If we want to add a and b where a=2 and b= 'RAM.'

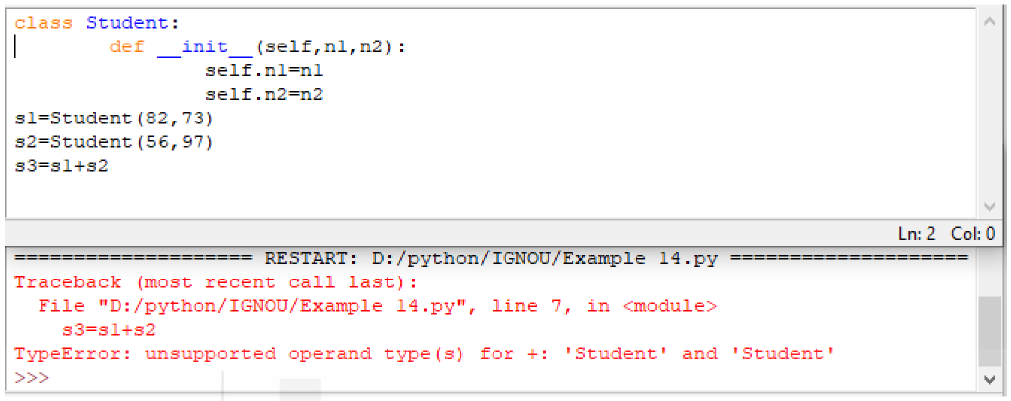
>>>a+b

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Because different types are not defined with the operator addition.

Consider a Student class

Example 14: Addition of two objects.

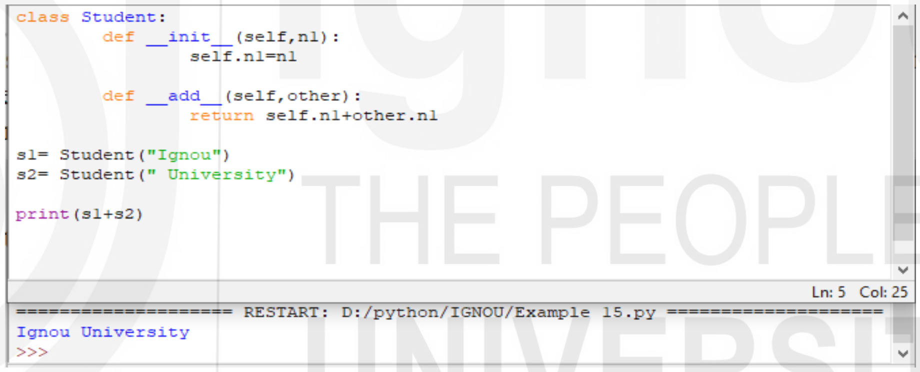


```
class Student:
        def __init__(self,n1,n2):
                self.n1=n1
                self.n2=n2
s1=Student(82,73)
s2=Student(56,97)
s3=s1+s2
```

```
==================== RESTART: D:/python/IGNOU/Example 14.py ====================
Traceback (most recent call last):
  File "D:/python/IGNOU/Example 14.py", line 7, in <module>
    s3=s1+s2
TypeError: unsupported operand type(s) for +: 'Student' and 'Student'
>>>
```

When we add two objects s1 & s2, it will show an error. Here it is not defined to the Python interpreter to add two objects. Hence we have to overload operator add.

Example 15: Operator overloading



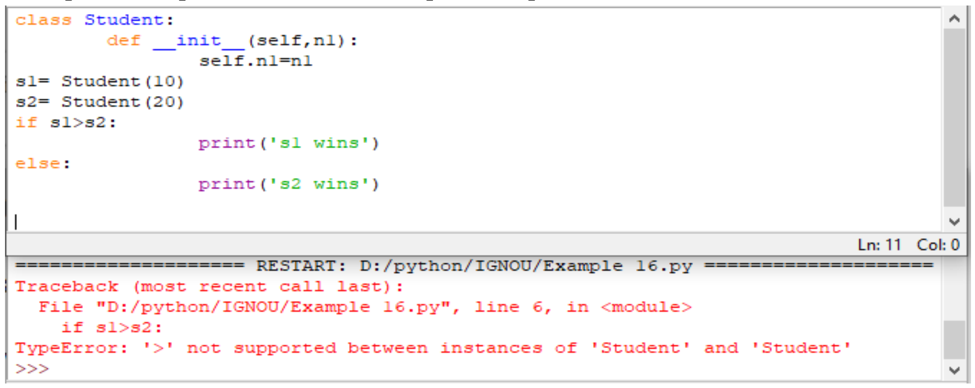
```
class Student:
        def __init__(self,n1):
                self.n1=n1

        def __add__(self,other):
                return self.n1+other.n1

s1= Student("Ignou")
s2= Student(" University")

print(s1+s2)
```

```
==================== RESTART: D:/python/IGNOU/Example 15.py ====================
Ignou University
>>>
```

Here we overloaded operator add.

Suppose we want to compare s1and s2

Example 16: Implementation of a comparison operator

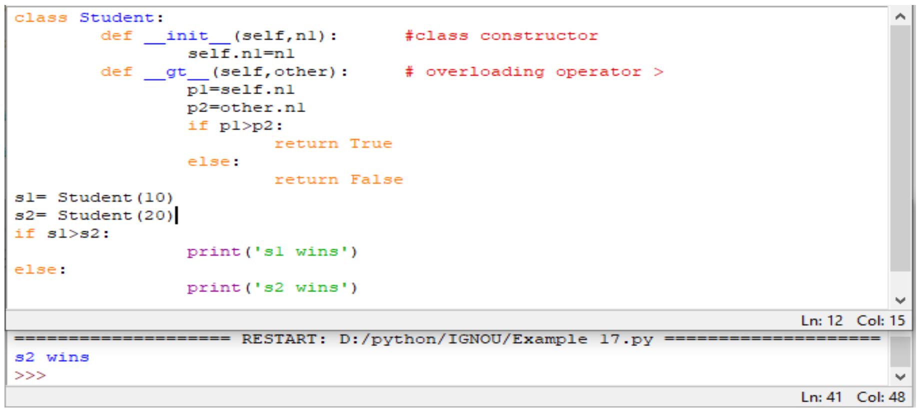


```
class Student:
        def __init__(self,n1):
                self.n1=n1
s1= Student(10)
s2= Student(20)
if s1>s2:
                print('s1 wins')
else:
                print('s2 wins')
```

```
==================== RESTART: D:/python/IGNOU/Example 16.py ====================
Traceback (most recent call last):
  File "D:/python/IGNOU/Example 16.py", line 6, in <module>
    if s1>s2:
TypeError: '>' not supported between instances of 'Student' and 'Student'
>>>
```

The comparison operator is not defined to the Python interpreter for the objects. Hence we have to redefine it or overload it. The function which corresponds to the symbol greater than '>' is __gt__ (refer to table 13.2). So we have to overload this operator by defining the function.

Example 17: Overloading operator '>.'



```
class Student:
        def __init__(self,n1):       #class constructor
                self.n1=n1
        def __gt__(self,other):      # overloading operator >
                p1=self.n1
                p2=other.n1
                if p1>p2:
                        return True
                else:
                        return False
s1= Student(10)
s2= Student(20)
if s1>s2:
                print('s1 wins')
else:
                print('s2 wins')
```

```
Ln: 12   Col: 15
==================== RESTART: D:/python/IGNOU/Example 17.py ====================
s2 wins
>>>
Ln: 41   Col: 48
```

because the value of s2 is higher than s1 hence s2 wins.

Now, what will happen if we want to print object s1

```
>>>print(s1)
…
<__main__.Student object at 0x00000192B57EE390>
```

It will print the address of the object. Now we want to print the value of the object then we have to redefined the function __str__().

```
>>>def __str__(self):
                return (' {} , {}'.format(self.n1,self.n2))
```

The value of two objects s1 and s2 printed.

13.7 POLYMORPHISM

Poly means 'multiple' and Morph means 'forms'. Thus polymorphism is multiple forms. For example, human being behaves differently in a different environment. The behaviour of a human in the office is different from his behaviour at home, which is different from his behaviour with friends at a party.

Thus in terms of object orientation due to polymorphic characteristic object behave differently in a different situation.

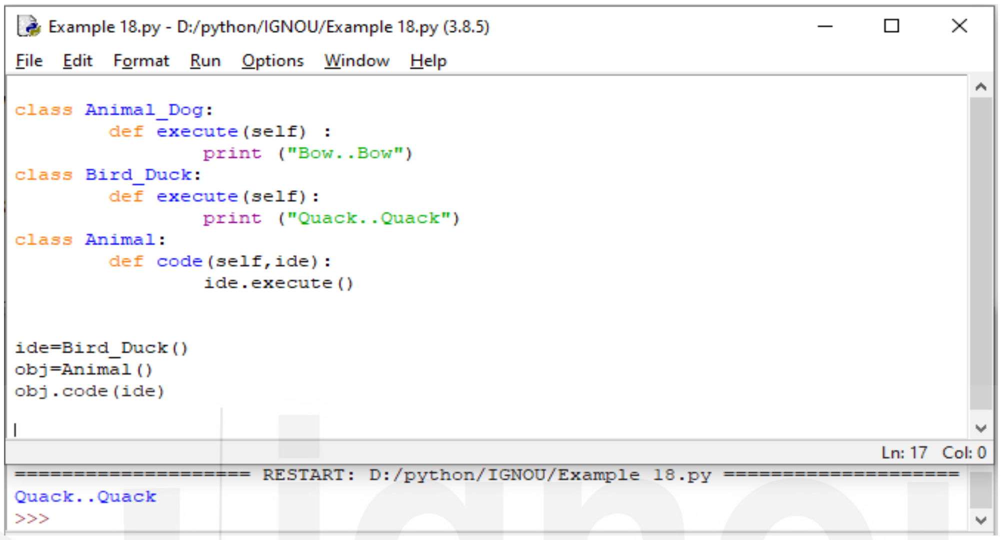
There are four ways to implement polymorphism in Python:
Duck Typing
Operator loading
Method Overloading
Method Overriding

Duck typing

There is a sentence in the English language "if this is a bird which is walking like a duck,quacking like a duck and swimming like a duck then that bird is a duck". It means if the behaviour of the bird is like a duck, then we can say it a Duck.
X=4
and
X= 'Mohit'
In the first statement, X is a variable storing integer value. The type of the X is int. However, in the second statement X= 'Mohit' the storage memory taken by the

variable is a string. In Python, we can't specify the type explicitly. During the runtime, whatever value we are storing in a variable the type is considered automatically. And this is called Duck Typing principle.

Example 18: Duck Typing principle

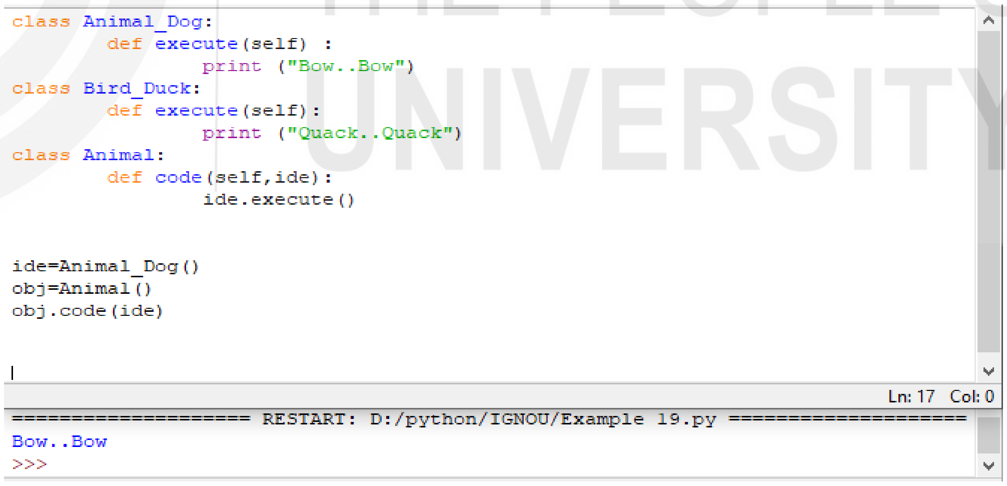

Example 18.py - D:/python/IGNOU/Example 18.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 17 Col: 0

```
class Animal_Dog:
        def execute(self) :
                print ("Bow..Bow")
class Bird_Duck:
        def execute(self):
                print ("Quack..Quack")
class Animal:
        def code(self,ide):
                ide.execute()


ide=Bird_Duck()
obj=Animal()
obj.code(ide)
```

```
==================== RESTART: D:/python/IGNOU/Example 18.py ====================
Quack..Quack
>>>
```

In the above example, the obj is an object of class Animal. When we call the code of Animal class, we have to pass an object ide. So before passing ide, we have to define it at the object of Duck class,one more ide for the class dog is defined. At the moment when we assign ide as an object of Dog class, then it will execute the dog method.

Example 19:



```
class Animal_Dog:
        def execute(self) :
                print ("Bow..Bow")
class Bird_Duck:
        def execute(self):
                print ("Quack..Quack")
class Animal:
        def code(self,ide):
                ide.execute()


ide=Animal_Dog()
obj=Animal()
obj.code(ide)
```

```
==================== RESTART: D:/python/IGNOU/Example 19.py ====================
Bow..Bow
>>>
```

So it doesn't matter which class object we are passing the matter is that object must have executed method. And this is called duck typing.

Operator Loading

Consider A=2 and B=5 are two variables. In a programming language, when we are performing a+b, it will perform the addition of two integer numbers. If X= 'Hello' and Y= 'Hi' then x+y will perform addition of two strings.

A=2

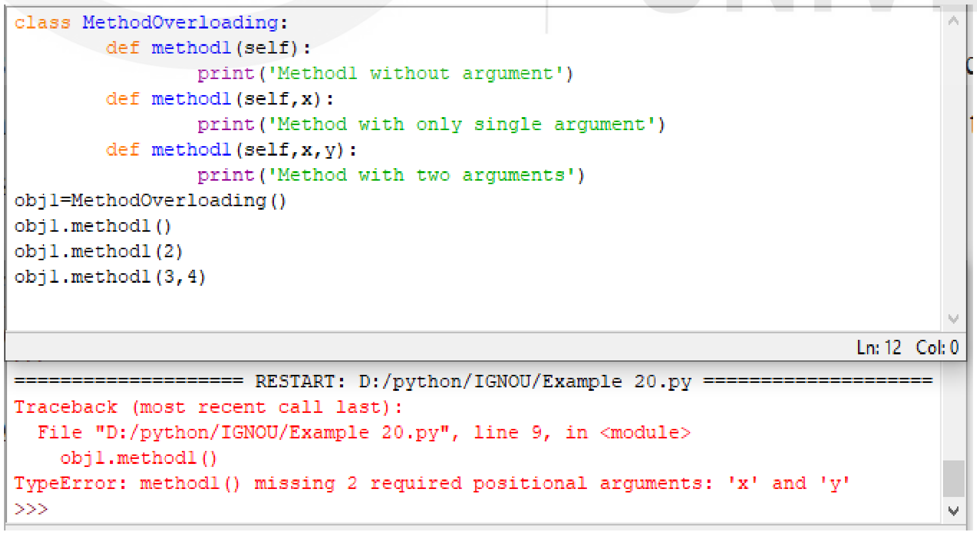
B=5

>>>print(A+B)
…
7
Python interpretor will take it as print(init.__add__(a,b)), where add() is a method of the init class.
>>>print(init.__add__(a,b))
7

X= 'Hello'
Y= 'Hi'
>>>print (X+Y)
HelloHi
Python interpretor will take print(X+Y) as print(str.__add__(X,Y)), where add()is a method osstr class.
>>>print(str.__add__(X,Y))
HelloHi

Method Overloading
Consider two classes having methods with the same name, but with a different number of parameters or different types of parameters, then the methods are called overloaded. In overloaded methods, the number of arguments is different, or types of arguments are different.
class student:
        marks(a,b)
        marks(a,b,c)
here two methods marks are defined one with two parameters and another having three parameters. These two methods are called method overloading. Python does not support method overloading.
If there are multiple methods in a class having the same name then Python will consider only the method which is described at the end of the class.

Example 20: Implementation of method overloading.



```
class MethodOverloading:
        def method1(self):
                print('Method1 without argument')
        def method1(self,x):
                print('Method with only single argument')
        def method1(self,x,y):
                print('Method with two arguments')
obj1=MethodOverloading()
obj1.method1()
obj1.method1(2)
obj1.method1(3,4)
```

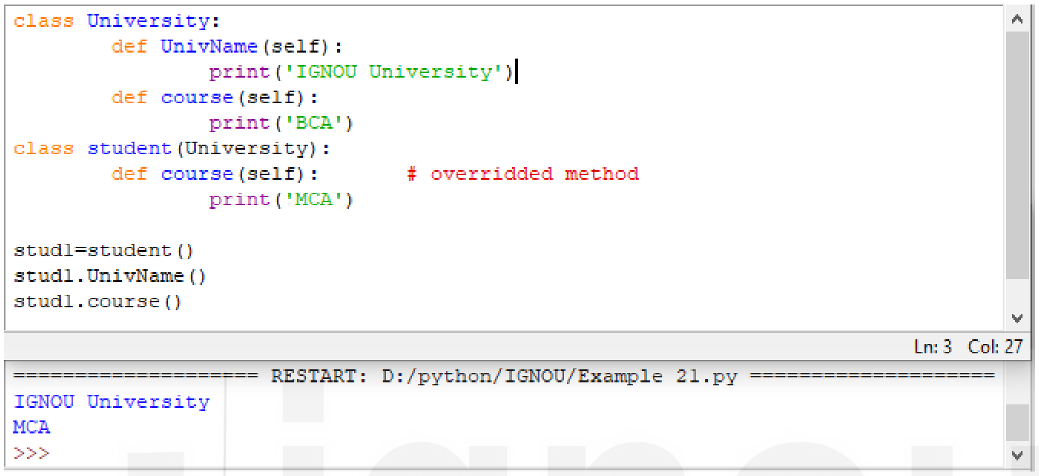
Ln: 12   Col: 0

```
==================== RESTART: D:/python/IGNOU/Example 20.py ====================
Traceback (most recent call last):
  File "D:/python/IGNOU/Example 20.py", line 9, in <module>
    obj1.method1()
TypeError: method1() missing 2 required positional arguments: 'x' and 'y'
>>>
```

Method Overriding
In Python, we can't create the same methods with the same name and the same number of parameters. But we can create methods of the same method in the classes

derived in a hierarchy. Thus the child class redefined the method of the parent class, and this is called method overriding.

Example 21: Implementation of method overriding.

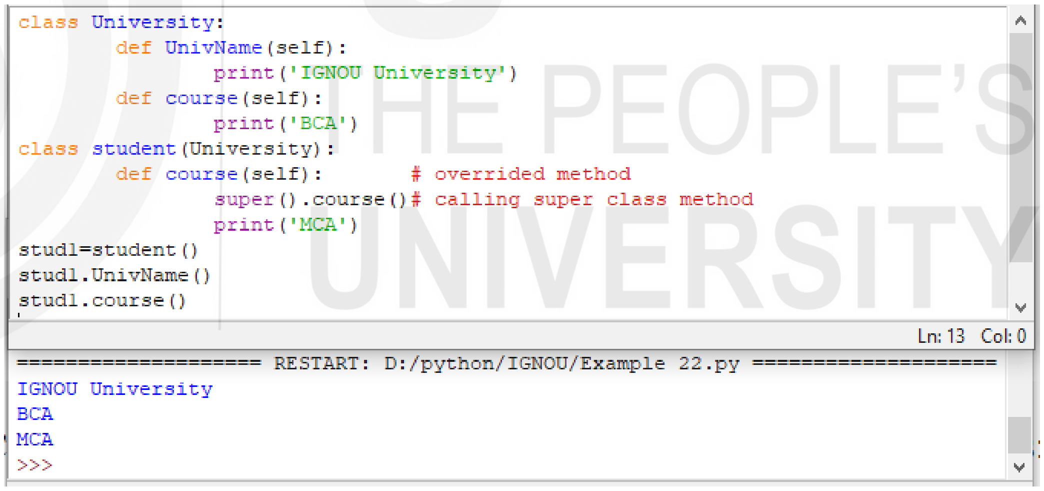


```
class University:
        def UnivName(self):
                print('IGNOU University')
        def course(self):
                print('BCA')
class student(University):
        def course(self):       # overridded method
                print('MCA')

stud1=student()
stud1.UnivName()
stud1.course()
```

```
Ln: 3   Col: 27
==================== RESTART: D:/python/IGNOU/Example 21.py ====================
IGNOU University
MCA
>>>
```

A method which is overridden in a child class can also access the method of its parent class with the help of keyword super().

Example 22: Method overriding with the super keyword



```
class University:
        def UnivName(self):
                print('IGNOU University')
        def course(self):
                print('BCA')
class student(University):
        def course(self):       # overridded method
                super().course()# calling super class method
                print('MCA')
stud1=student()
stud1.UnivName()
stud1.course()
```

```
Ln: 13   Col: 0
==================== RESTART: D:/python/IGNOU/Example 22.py ====================
IGNOU University
BCA
MCA
>>>
```

Check Your Progress - 1

1. Implement the class Circle that represents a circle. The class must contain the following methods:

   Circle_setradius():Takes one number of values as input and sets the radius of the circle.

   circle_perimeter(): Returns the perimeter of the circle.

   Circle_area(): Returns the area of the circle.

   ----

2. Define method overloading and constructor overloading in Python.

3.Select the correct output generated from the following program code:

```
class code1:
        def __init__(self,st="Welcome to Python World"):
                self.st=st
        def output(self):
                print(self.st)
        obj=code1()
        obj.output()
```

a) The code result an error because constructor are defined with default arguments

b) Output in not displayed

c) "Welcome to Python World" is printed

d) The code result an error because parameters are not defined in a function

----

4.What type of inheritance is illustrated in the following Python code?

```
class Class1():
        pass
class Class2(Class1):
        pass
class Class3(Class2):
        pass
```

a) Multilevel inheritance

b) Multiple inheritance

c) Hierarchical inheritance

d) Single-level inheritance

----

5. What is polymorphism and what is the main reason to use it?

13.8 SUMMARY

In this unit, we discussed the concepts of classes and objects. Concept of a namespace to store object and class in memory is also defined in this unit. Different types of class methods namely static methods ,instance methods and class methodsare discussed in this unit.

This unit also focused on inheritance and various types of inheritance: single level inheritance,multilevel inheritance and multiple inheritances.

In this unit, it is described that the same operator can be used for multiple purposes, and this is called operator overloading. I list of operators and corresponding methods, also called as magic methods are also given in the unit.

Polymorphism means it is the ability to behave differently in a different situation. The concept of polymorphism and the implementation of polymorphism with Duck Typing, Operator loading, Method Overloading and Method Overriding are also described in the unit.

SOLUTIONS TO CHECK YOUR PROGRESS

Check Your Progress -1

1.
```
    class Circle:
            defcircle_setradius(self, radius):
            self.r = radius
        defcicle_perimeter(self):
            return 2 * 3.142 * self.r
        defcicle_area(self):
            returnself.x * self.x*3.142
```

2. Iftwo or methods are defined with the same then but they differ in return types or having a different number of arguments or having different types of arguments then these methods are called as method overloading.

   Two constructors are overloaded when both are having different no or different types of arguments.

   In the python method overloading and constructor, overloading is not possible.

3. C
4. A
5. Polymorphism is one of the main features of object-oriented programming languages. With this characteristic, we can implement elegant software.

UNIT 14 EXCEPTION HANDLING IN PYTHON PROGRAMMING

Structure

14.0 INTRODUCTION

Programers always try to write an error-free program, but sometimes a program written by the programmer generates an error or not execute due to the error generated in the program. Some times the program code is correct, but still, it generates an error due to the malicious data is given as an input to the program. This malicious data may be given by the user or from a file that causes an error generation during the execution. This type of error generally occurs when we user server-side programs such as web programming and gaming servers.

Every developer is writing a code that must not generate an error during execution. We will study various types of errors that generate during the program execution or before the execution of the program code.

14.1 OBJECTIVES

After going through this unit, you will be able to :

- Understand the requirement of exception handling in programming
- Raise and catch the exceptions
- Perform exception handling in python programming

14.2 DEFAULT EXCEPTION HANDLER

Consider the following examples in which syntax is correct (as per the Python statement), but it causes an error.

Example 1.

```
>>>3/0
```

Name of Error will be ZeroDivisionError

And the reason for the error is division by zero

Example 2

```
>>> list1=[2, 3, 4, 5, 6]
>>>list1[5]
```

Name of Error is :IndexError
Reson of Error is : list index out of range

Example 3.

```
>>>x+3
```

Name of Error is : Name Error
Reason of Error is : name 'x' is not defined"

Example 4.

```
>>>int('22.5')
```

Name of Error is :ValueError
Reason of error is : invalid literal for int() with base 10: '22.5'

Example 5.

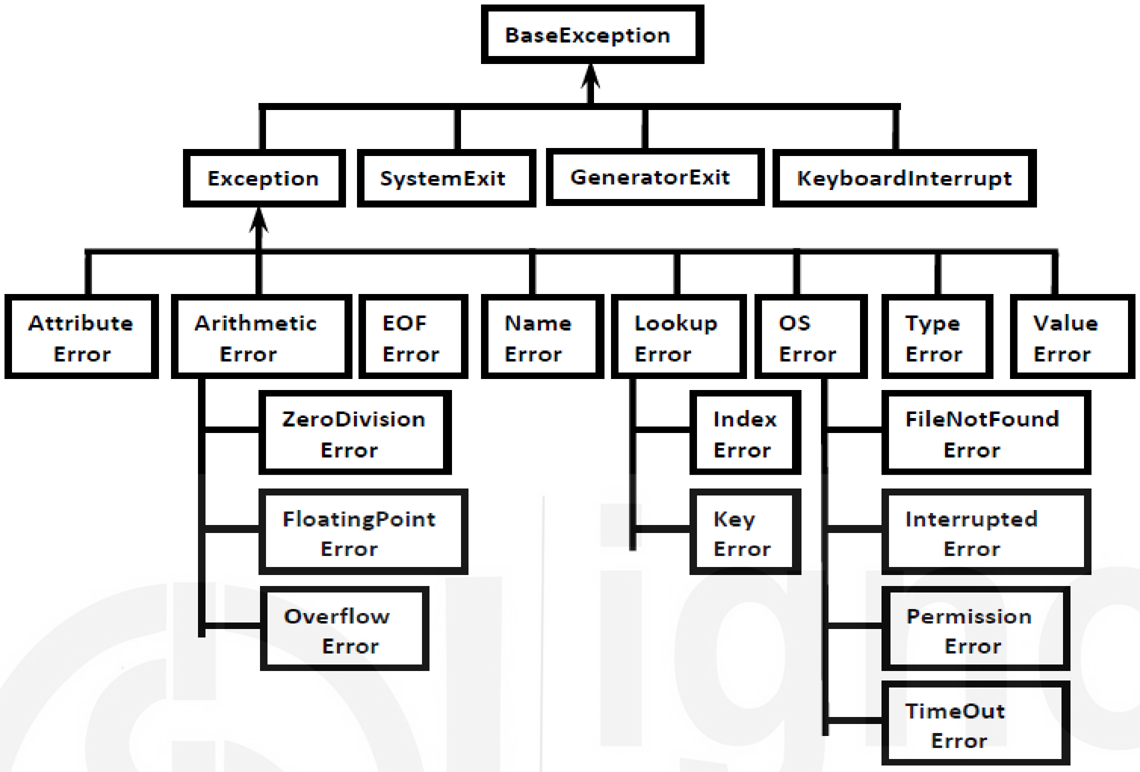
```
>>> '2'*'3'
```

Error name is :TypeError
Reason of Error is : can't multiply sequence by non-int of type 'str'

In each example syntax is correct, but it goes to an invalid state. These are runtime errors. In such a situation, Python interpreter generates an exception for run time errors. This exception generates an object which contains all information related to the error. In Example 5 the error message displayed: what is happening and at which line number it is happening (here line number is one because there is only one statement in the code) and the type of error is also printed.

It is the responsibility of python virtual machine to create the object corresponding to the exception generated to handle the program code. If the code responsible for handling the exception is not there in the program code, then python interpreter will suspend the execution of the program code since an exception is generated the python print the information about the generated exception.

14.3 CATCHING EXCEPTIONS



In Python, every exception is class, and these classes are the derived class of the BaseException class. ThusBaseException class is the topmost class in the hierarchy of exception classes.

Exception handling using try-except:

```
try:
        risky code(generating exception)
except:
        corresponding to risky code(handler)
```

The code which generates an exception must be written in a block with the name specify by 'try'and the code which will handle the exception will be given in 'except' block.

If the three lines code are written in Python

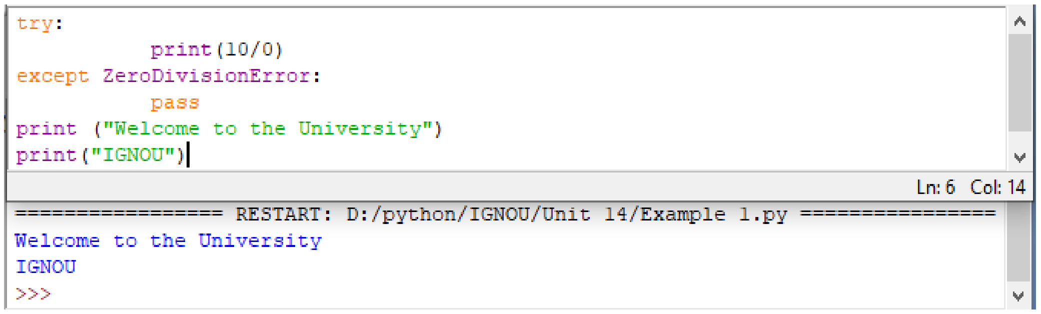
```
print("IGNOU")
print(87/0)
print("Welcome to the University")
>>>
…
```

Here the first line will be executed, and IGNOU will be printed when Python interpreter will execute 2nd lines of the program then it generates an exception, and the default exception handler will print the error code "division by Zero"then after the program is terminated. Thus the third line of the program will not execute. And this is also called as abnormal termination.

If we rewrite the code with the try and except then the code will become

```
print("IGNOU")
```

Example 1: Implementation of try and except block.

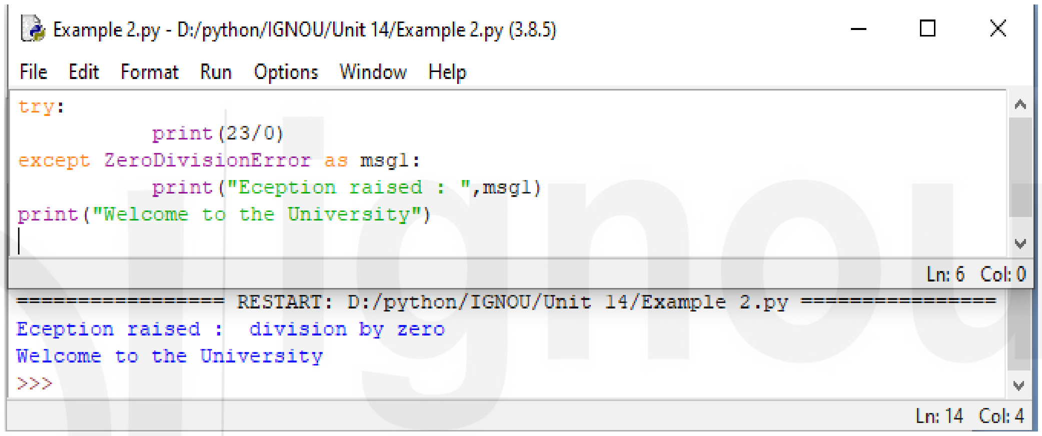


```
try:
        print(10/0)
except ZeroDivisionError:
        pass
print ("Welcome to the University")
print("IGNOU")
```

```
================= RESTART: D:/python/IGNOU/Unit 14/Example 1.py ================
Welcome to the University
IGNOU
>>>
```

Here program terminated successfully. We can also print the error message by modifying the code as:

Example 2:



Example 2.py - D:/python/IGNOU/Unit 14/Example 2.py (3.8.5)
File Edit Format Run Options Window Help
Ln: 6 Col: 0
Ln: 14 Col: 4

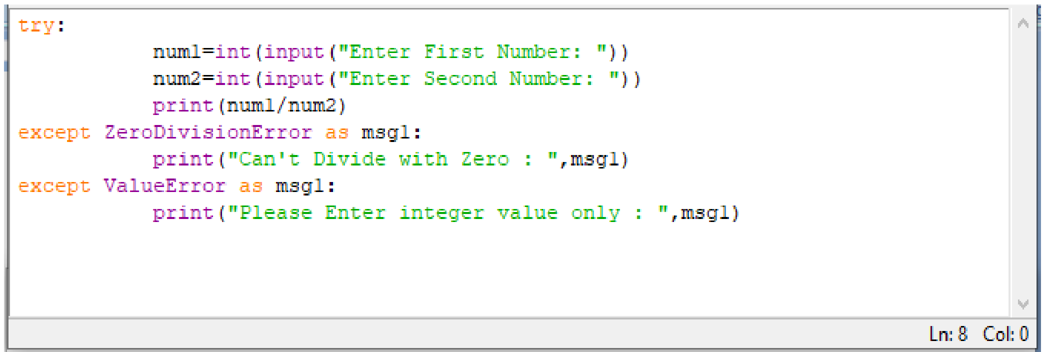
```
try:
        print(23/0)
except ZeroDivisionError as msgl:
        print("Eception raised : ",msgl)
print("Welcome to the University")
```

```
================= RESTART: D:/python/IGNOU/Unit 14/Example 2.py ================
Eception raised :  division by zero
Welcome to the University
>>>
```

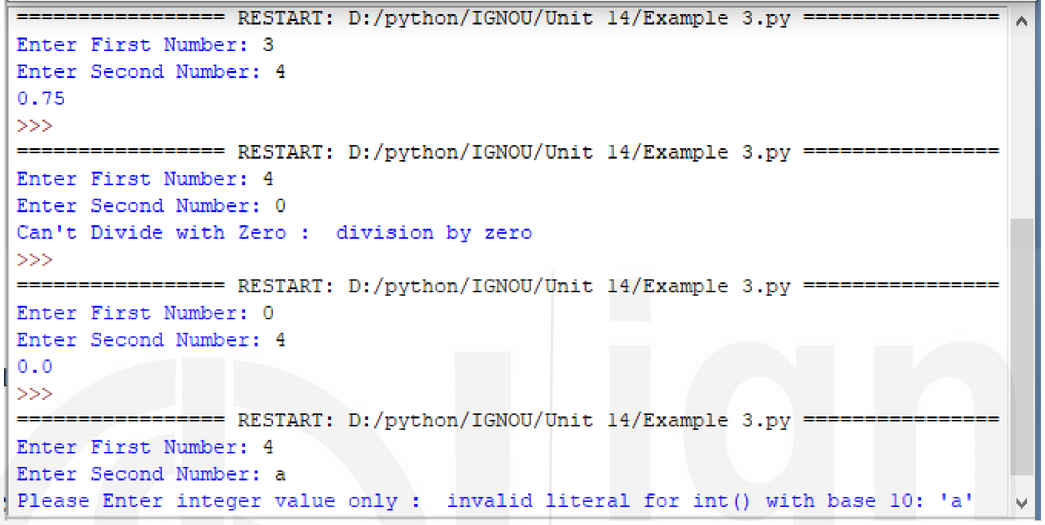
In the above code, the first line executed successfully and "IGNOU" will be printed. When the python interpreter executes the print statement "(23/0)" an exception is raised, and a message given in the print statement will be printed with the name of the exception "division by zero". After that,the interpreter executes the last statement, and it will print "Welcome to the University."

If there are more than one except block written in the try block,then the try block will execute the except block which is responsible for the code.

Example 3: Try block with more than one except block.

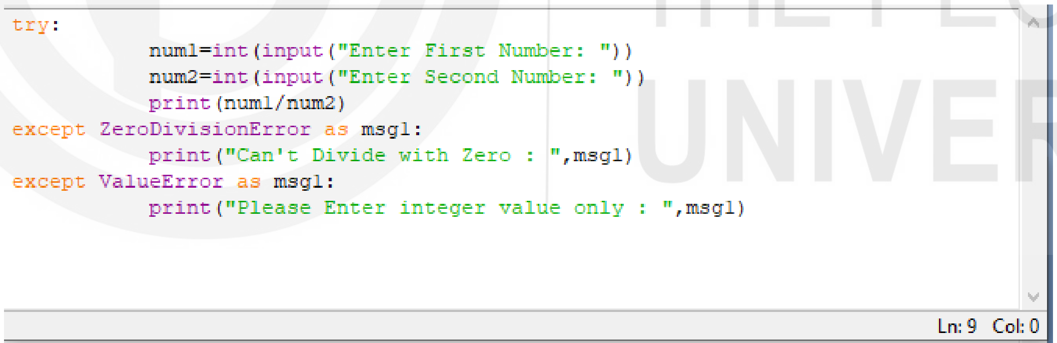


```
try:
            num1=int(input("Enter First Number: "))
            num2=int(input("Enter Second Number: "))
            print(num1/num2)
except ZeroDivisionError as msg1:
            print("Can't Divide with Zero : ",msg1)
except ValueError as msg1:
            print("Please Enter integer value only : ",msg1)
```

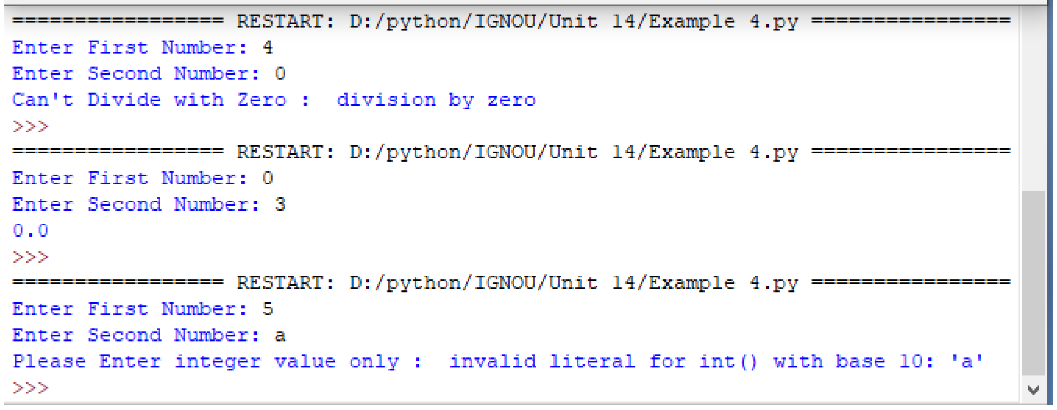


```
================= RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 3
Enter Second Number: 4
0.75
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 4
Enter Second Number: 0
Can't Divide with Zero :  division by zero
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 0
Enter Second Number: 4
0.0
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 4
Enter Second Number: a
Please Enter integer value only :  invalid literal for int() with base 10: 'a'
```

With the help of single except block, we can handle multiple exceptions:
Example 4: Multiple exception handling with a single try block and multiple except block.



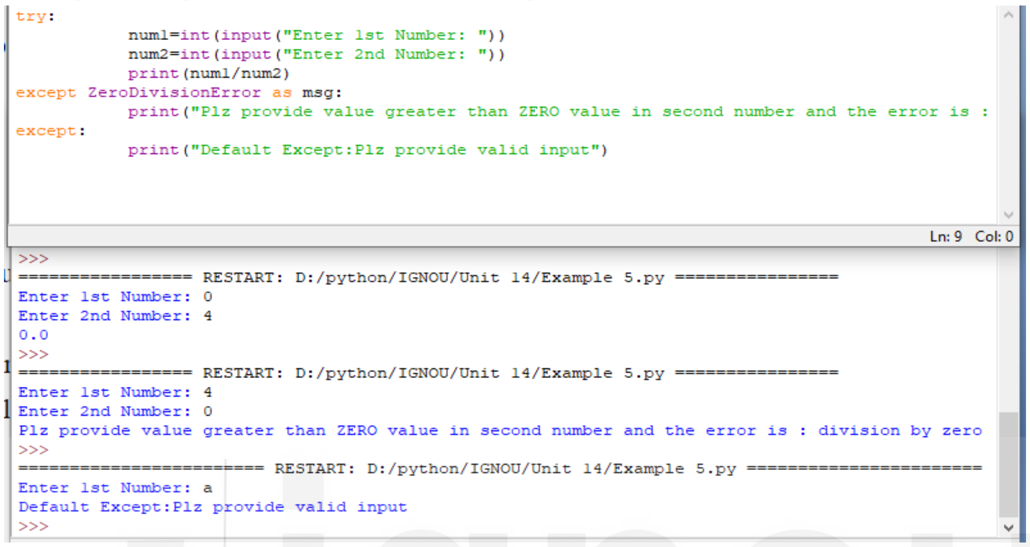
```
try:
            num1=int(input("Enter First Number: "))
            num2=int(input("Enter Second Number: "))
            print(num1/num2)
except ZeroDivisionError as msg1:
            print("Can't Divide with Zero : ",msg1)
except ValueError as msg1:
            print("Please Enter integer value only : ",msg1)
```



```
================= RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 4
Enter Second Number: 0
Can't Divide with Zero :  division by zero
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 0
Enter Second Number: 3
0.0
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 5
Enter Second Number: a
Please Enter integer value only :  invalid literal for int() with base 10: 'a'
>>>
```

Corresponding to each error there will be an except block if corresponding except block is not there then a except block written at the end of all except block is used for the error. And it must be the last except block in all.

Example 5: Implementation of default except for block



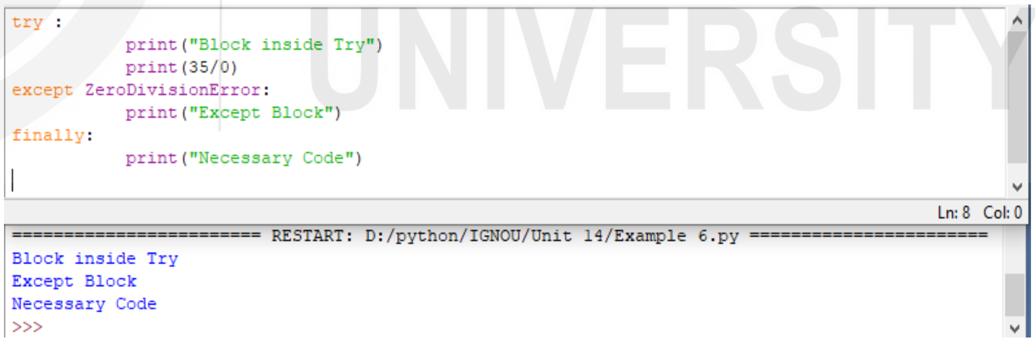
```
try:
        num1=int(input("Enter 1st Number: "))
        num2=int(input("Enter 2nd Number: "))
        print(num1/num2)
except ZeroDivisionError as msg:
        print("Plz provide value greater than ZERO value in second number and the error is :
except:
        print("Default Except:Plz provide valid input")
```

```
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 5.py ================
Enter 1st Number: 0
Enter 2nd Number: 4
0.0
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 5.py ================
Enter 1st Number: 4
Enter 2nd Number: 0
Plz provide value greater than ZERO value in second number and the error is : division by zero
>>>
======================== RESTART: D:/python/IGNOU/Unit 14/Example 5.py =======================
Enter 1st Number: a
Default Except:Plz provide valid input
>>>
```

Some times exceptions may or may not be raised, and sometimes exceptions may or may not be handled. Irrespective of both of these we still want to execute some code. And such codes are written in finally block.

```
try:
        Code with Error
except:
        Code to handle error
finally:
        Necessary code
```

Example 6: Implementation of finally block.



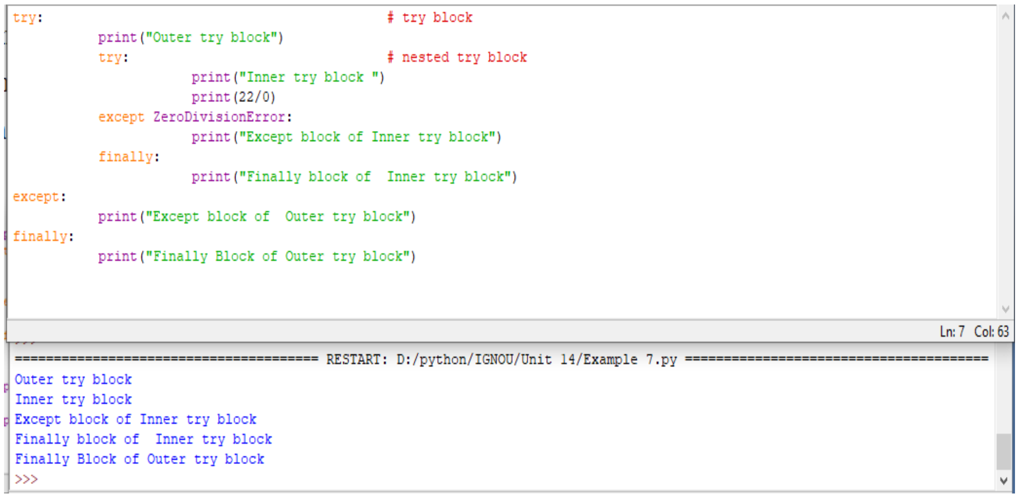
```
try :
        print("Block inside Try")
        print(35/0)
except ZeroDivisionError:
        print("Except Block")
finally:
        print("Necessary Code")
```

```
======================== RESTART: D:/python/IGNOU/Unit 14/Example 6.py =======================
Block inside Try
Except Block
Necessary Code
>>>
```

Nested try-except block:

If a try block is written inside another try block, then it is called as nested. The code having higher risk must be defined in the innermost try block. If there is any exception raised by the innermost try block, then it will be handled by the innermost except block. If the innermost except block is not able to handle the errorthenexceptblock defined outside the inner try block will handle the exception.

Example 7: Implementation of the nested try block

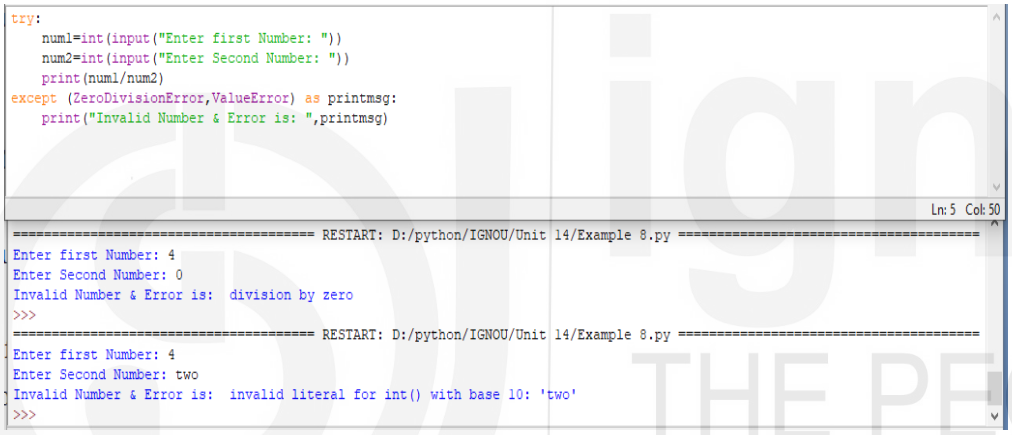


```
try:                                        # try block
          print("Outer try block")
          try:                              # nested try block
                    print("Inner try block ")
                    print(22/0)
          except ZeroDivisionError:
                    print("Except block of Inner try block")
          finally:
                    print("Finally block of  Inner try block")
except:
          print("Except block of  Outer try block")
finally:
          print("Finally Block of Outer try block")
```

```
======================================= RESTART: D:/python/IGNOU/Unit 14/Example 7.py =======================================
Outer try block
Inner try block
Except block of Inner try block
Finally block of  Inner try block
Finally Block of Outer try block
>>>
```

A single except block can be used to handle multiple exceptions. Consider the given an example:

Example 8:Single except block handling multiple exceptions.

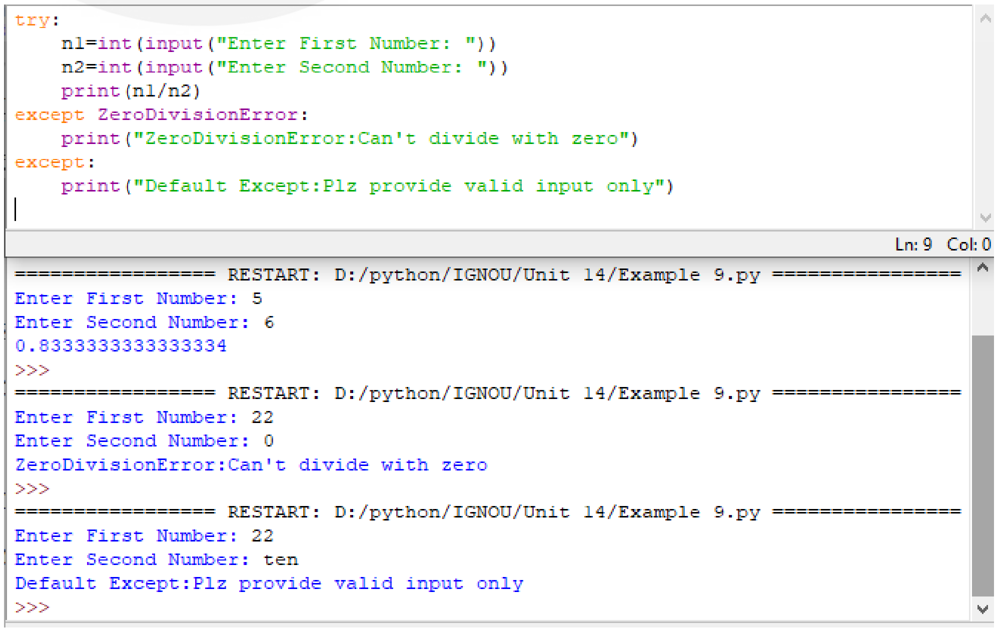


```
try:
    num1=int(input("Enter first Number: "))
    num2=int(input("Enter Second Number: "))
    print(num1/num2)
except (ZeroDivisionError,ValueError) as printmsg:
    print("Invalid Number & Error is: ",printmsg)
```

```
======================================= RESTART: D:/python/IGNOU/Unit 14/Example 8.py =======================================
Enter first Number: 4
Enter Second Number: 0
Invalid Number & Error is:  division by zero
>>>
======================================= RESTART: D:/python/IGNOU/Unit 14/Example 8.py =======================================
Enter first Number: 4
Enter Second Number: two
Invalid Number & Error is:  invalid literal for int() with base 10: 'two'
>>>
```

In case of different types of error, we can use a default except for block. This block is generally used to display normal error messages. The default except block must be the last block of all except blocks.

Example 9: Implementation of default except for block.

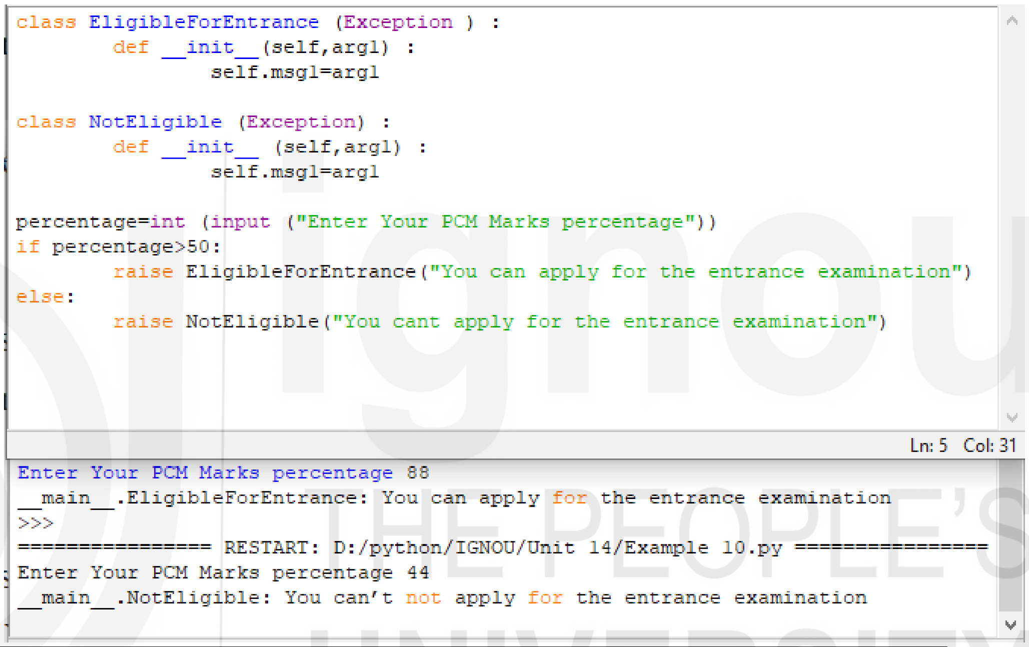


```
try:
    n1=int(input("Enter First Number: "))
    n2=int(input("Enter Second Number: "))
    print(n1/n2)
except ZeroDivisionError:
    print("ZeroDivisionError:Can't divide with zero")
except:
    print("Default Except:Plz provide valid input only")
```

```
================= RESTART: D:/python/IGNOU/Unit 14/Example 9.py ================
Enter First Number: 5
Enter Second Number: 6
0.8333333333333334
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 9.py ================
Enter First Number: 22
Enter Second Number: 0
ZeroDivisionError:Can't divide with zero
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 9.py ================
Enter First Number: 22
Enter Second Number: ten
Default Except:Plz provide valid input only
>>>
```

14.4 RAISE AN EXCEPTION

Some times user want to generate an exception explicitly to inform that this is the exception in this code. Such type of exceptions is called as user-defined exceptions or customized exceptions.
This user-defined exception is a class defined by the user, and this class is derived from the Exception class. Pythoninterpreter does not have any information related to the user-defined exception class. Thus, it must be raised explicitly by the user.
The keyword*raise* is used to raise the class when it is required.
Example 10: Implementation of user-defined exceptions.



```
class EligibleForEntrance (Exception ) :
        def __init__(self,arg1) :
                self.msg1=arg1

class NotEligible (Exception) :
        def __init__ (self,arg1) :
                self.msg1=arg1

percentage=int (input ("Enter Your PCM Marks percentage"))
if percentage>50:
        raise EligibleForEntrance("You can apply for the entrance examination")
else:
        raise NotEligible("You cant apply for the entrance examination")
```

Ln: 5 Col: 31

```
Enter Your PCM Marks percentage 88
__main__.EligibleForEntrance: You can apply for the entrance examination
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 10.py ================
Enter Your PCM Marks percentage 44
__main__.NotEligible: You can't not apply for the entrance examination
```

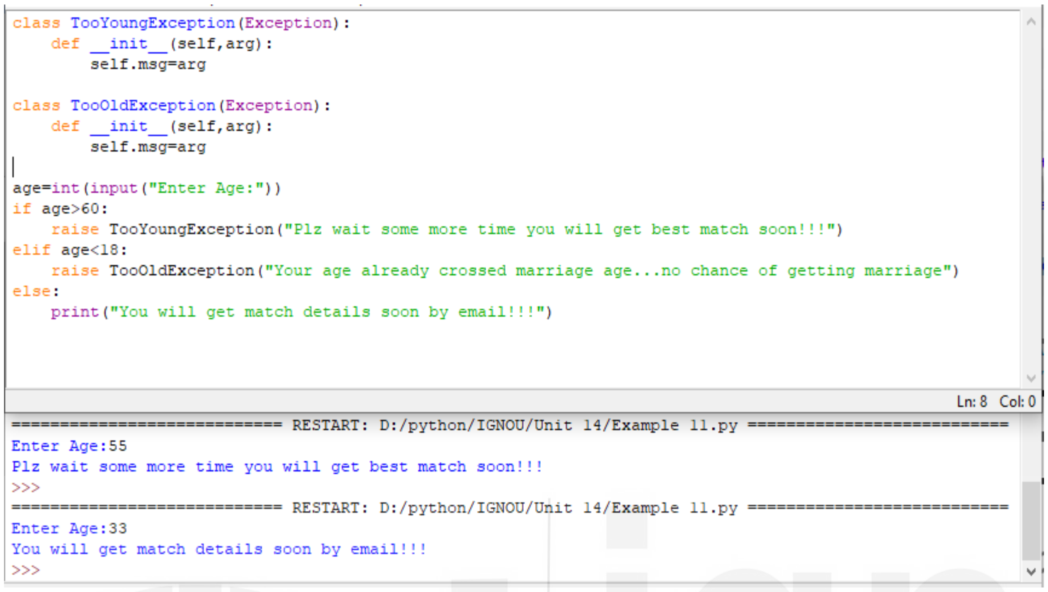
14.5 USER-DEFINED EXCEPTIONS

A programmer can create his exception, called a user-defined exception or custom exception. A user-defined exception is also a class derived from exception class.
Thus, class, is the user define a class which is a subclass of the Exception class.
There are two steps in using user-defined exception:
Step 1: By inheriting Exception class create a user-defined exception class.
Step 2: Raise this exception in a program where we want to handle the exception.
Syntax :

```
classMyException(Exception):
        pass
classMyException(Exception)
        def __init__(self,argumnet):
                self.msg=argumnet
```

A *raise* statement is used to raise the statement.

Example 11: Program to raise an exception.



```
class TooYoungException(Exception):
    def __init__(self,arg):
        self.msg=arg

class TooOldException(Exception):
    def __init__(self,arg):
        self.msg=arg

age=int(input("Enter Age:"))
if age>60:
    raise TooYoungException("Plz wait some more time you will get best match soon!!!")
elif age<18:
    raise TooOldException("Your age already crossed marriage age...no chance of getting marriage")
else:
    print("You will get match details soon by email!!!")
```

Ln: 8 Col: 0

```
============================ RESTART: D:/python/IGNOU/Unit 14/Example 11.py ============================
Enter Age:55
Plz wait some more time you will get best match soon!!!
>>>
============================ RESTART: D:/python/IGNOU/Unit 14/Example 11.py ============================
Enter Age:33
You will get match details soon by email!!!
>>>
```

Check Your Progress

1. Give the name of the error generated by the following python codes.
    a) 4 / 0
    b) (3+4]
    c) lst = [4;5;6]
    d) lst = [14, 15, 16]
       lst[3]
    e) x + 5
    f) '2' * '3'
    g) int('4.5')
    h) 2.0**10000
    i) for i in range(2**100):
       pass

2. Define the minimum number of except statements that can be possible in the try-except block.

----

3. Describe the conditions in which finally block executed.

----

4. Select the keyword from the following, which is not an exception handling in Python.
    a) try
    b) except
    c) accept
    d) finally

----

5. Give the output generated from the following Python code?
    lst = [1, 2, 3, 4]
    lst[4]

----

6. What will be the output generated with the following code?

```
def function1():
        try:
        function2(var1, 22)
        finally:
        print('after function2')
        print('after function2?')
function1()
```

a)Output will not be generated
b) after f?
c) error
d) after f

14.6 SUMMARY

In this unit, it is defined that a program may produce an error even though the programmer is writing error-free code. These are the mistakes that change the behaviour of our program code.
In this unit, it is defined that the default exception handlers are there in Python. How these exception handler works are described in this unit.
By default, there are exception handlers in Python. But a user can also raise the exception which is a customized exception, not a language defined exception.

SOLUTION TO CHECK YOUR PROGRESS

Check Your Progress

1.
   a) ZeroDivisionError
   b) SyntaxError
   c) SyntaxError
   d) IndexError
   e) NameError
   f) TypeError
   g) ValueError
   h) OverflowError
   i) KeyboardInterrupt Error
2. At least one except statement.
3. Finally block is always executed.
4. C
5. IndexErrorbecause the maximum index of the list given is 3.
6. C, since function finction1 is not defined.

UNIT 15 PYTHON-ADVANCE CONCEPTS

Structure

15.0 INTRODUCTION

In unit number 11 of this course we learned about the functions, we learned that we have inbuilt functions and we also learned to develop our own functions just for the sake of revision, some of the examples are listed below

Say the following lines of code are saved in functions.py

```
# In-Built Function

Name = “IGNOU-SOCIS”

print(len(Name))
```

executing the functions.py by running the command $ python function.py we get the output 11, which is the length of the variable Name. here, len() is the in-built function of python, which can be used to calculate length of any string, we need not to write code agin and again to determne the length of any string simply we need to use this built-in function and get the job done.

We also learned to define our own function just to recapitulate lets write our function to add two numbers

```
# User-Defined Functions

defadd_two(a,b) :

returna+b

total = add_two(5,4)

print(total)
```

here add_two(a,b) is the user defined function that takes two numbers as input and return their sum, which is stored in the varable total, the result of total is printed subsequently.

We also learned the concept of anonymous functions i.e lambda functions or Lambda expressions, the concept was introduced in python and later it was adopted by many other languages be it C++ or Java. The advantage of using the lambda function is that, they can we defined with in the code by writing few lines and they doesn't need any name, that's amazing. One example of Lambda expressions is sited below, here we will write the lambda expression equivalent for the User-Defined Function add_two, which is given above

```
# Lambda Expressions (Anonymous Functions)

def add(a,b)

returna+b

add2 = lambda a,b : a+b

print(add2(2,3))
```

here, add2 collects the output of the lambda function, we can see the output by calling lambda function, by writing print(add2(2,3))

Generally lambda functions are not meant for the user defined functions but are used to facilitate the working of various built in functions like, map, reduce, filter, etc.

Modern programming language emphasizes on code reusability, where one need to work with the already written codes, but these codes needs to be customized, i.e. their functionality is required to be enhanced, and to enhance the functionality of already written functions, python has the concept of decorators. We will learn about decorators in our subsequent section 15.2

15.1 OBJECTIVES

After going through this unit, you will be able to:

- Enhance the functionality of functions by using decorators
- Understand the concept of iterators and iterables
- Appreciate the concept of generators
- Understand the concept cooperative multitasking using co-routines

15.2 DECORATORS

Modern programming paradigm recommends the technique of code reusability, where we need to customize the code as per our requirements, without disturbing the actual functionality of the code which is already written. To achieve this, python introduced the concept of decorators, this concept is used to enhance the functionality of functions, which are already written, for example say we have two functions func1() and func2() as given in the python code given below

```
#  Decorators – to enhance functionality of other functions
deffuncA():
print (‘ this is functionA’)
deffuncB():
print (‘ this is functionB’)
funcA()
funcB()
```

on executing the above mentioned code by running deco.py we will get output

```
this is function1
this is function2
```

but without disturbing the existing code if we want that along with, “this is function1” the output should contain the line “this is a wonderful function”, i.e. to enhance the already existing functionality, we need to exercise the concept of decorators.

To understand this concept lets re-write the code given above

```
#  Decorators – to enhance functionality of other functions
defdecorator_funcn (any_funcn):
defwrapper_funcn():
print(‘this is a wonderful function’)
any_funcn()
returnwrapper_funcn
deffuncA():
print (‘ this is functionA’)
deffuncB():
print (‘ this is functionB’)
variable = decorator_funcn (funcA)
variable( )
```

Lets understand the concept of Decorators through the above code. Here, decorator_funcn is defined to enhance the functionality of any function (may be funcA or funcnB) from printing “this is function A” to “this is a wonderful function this is function A” or “this is function B” to “this is a wonderful function this is function B”.

Without altering any line of code of already written functions. To achieve this a decorator function decorator_funcn is defined, this function takes any_funcn as input argument, it can be any function may it be funcnA or funcnB. Inside the decorator_funcn, a wrapper function named wrapper_funcn() is defined to wrapup the new features over the existing features of the function taken as argument by the decorator_funcn(). The body of the wrapper_funcn() includes the additional feature, in our case it is print('this is a wonderful function') an then the original function passed as an argument to the decorator_funcn() i.e. any_funcn() is called, and finally the output of wrapper_funcn is returned. To execute the decorator_funcn, the decorator_funcn with argument funcA is called and its return value is collected in variable, then variable() is executed and the output received is "this is a wonderful function this is function A" or "this is a wonderful function this is function B".

Lets run some shortcuts also to work with the concept of decorators, which involves one more concept of syntactic sugar, where w use @ symbol to use the decorators before executing any function, i.e. to enhance its functionality.

```
#  Decorators – to enhance functionality of other functions
# @ used for decorators
defdecorator_funcn (any_funcn):
defwrapper_funcn():
print('this is a wonderful function')
any_funcn()
returnwrapper_funcn
@decorator_funcn
deffuncA():
print (' this is functionA')
funcA()
@decorator_funcn
deffuncB():
print (' this is functionB')
funcB()
```

whenever we use @decorator_funcn before any function the output of that function preceeds with the output "this is a wonderful function", later the output of actual function turnsup. As in this code, calling funcA() leads to output "this is a wonderful function this is function A" or calling funcB() leads to output "this is a wonderful function this is function B".

In this section we learned, the concept of decorators, as functionality enhancers. Now we will study the concept of Iterators and Iterables.

☞ **Check Your Progress 1**

1) Compare built-in functions with user defined and Lambda functions
2) What are Decorators? Briefly discuss the utility of decorators.

15.3 ITERATORS

In Python Iterator is an object that can be iterated i.e. an object which will return data or one element at a time. Iterators exist every where, but they are implemented well in generators, comprehensions and even in for loops; but are generally hidden in plain sights. In this section we will try to understand this concept by exploring the functioning of for loops. In our earlier units we learned about the concepts of Lists, Tuples, Dictionaries, Sets , Strings etc., infact most of these built in containers are collectively called, iterables. An object is called iterable if we can get an iterator from it..In Python any iterator object must follow the iterator protocol i.e. the implementation of iter() and next() functions, the iter function returns an iterator, and an object is called iterable if we can get an iterator from it (for this we use iter()function). To understand what are iterables lets understand the functioning of for loop.

We learned that numbers = [1,2,3,4] is a list and if we want to print the elements of this list, we may use for loop for this purpose, and we will write

```
# Iterables

numbers = [1,2,3,4]

fori in numbers :

print (i)
```

now lets understand how for loop works behind the scenes, firstly the for loop calls a function called iter() function, this iter function changes the iterable to iterator i.e. it takes list as argument, so in out case we have iter(numbers), and this is now an iterator. Subsequently the next function is called whose argument is this iterator i.e. next(iter(numbers)), as the loop progresses the next function provides the values from the iterable i.e. the list numbers in our case, first run provides 1, second run provides 2 and so on. i.e to see how the for loop works we write the logic of for loop as follows

```
# Iterables

numbers = [1,2,3,4]

number_iter = iter(numbers)

print(next(number_iter))        # Output will be 1
```

print(next(number_iter))	# Output will be 2

print(next(number_iter))	# Output will be 3

print(next(number_iter))	# Output will be 4

print(next(number_iter))	# Output will be Error as the iterable numbers is upto 4 only

In this way only the for loop works on any of the iterables i.e. List or Tuple or String. So, iterables are the python data types which uses the iter and next functions, but iterator can directly use the next function. The iterables uses the iter() function to generate iterator and then uses next function but iteartors don't use iter function they directly use the next function to get the job done. In Python any iterator object must follow the iterator protocol i.e. the implementation of iter() and next() functions, the iter function returns an iterator, and an object is called iterable if we can get an iterator from it (for this we use iter()function)

Now, we will learn about the iterators.

# Iterators

numbers = [1,2,3,4]   # Iterables

Squares =map(lambda a : a**2, numbers)   #Iterator

print(next(squares))   # output will be square of 1 i.e. 1

print(next(squares))   # output will be square of 2 i.e. 4

print(next(squares))   # output will be square of 3 i.e. 9

print(next(squares))   # output will be square of 4 i.e. 16

In this section we learned about the concept of Iterators and Iterables, now we will extend our discussion to Generators, in the next section.

☞ **Check Your Progress 2**

1) What are Iterators? How iterators differ from iterables?
2) Discuss the execution of for loop construct, from iterators point of view

15.4 GENERATORS

Generators are also a kind of iterators, but they are quite memory efficient i.e. needs very less memory hence helps in improving the performance of any programme. We learned in last section that iterators involves production of any sequence, the generators are also generating the sequence but their modus operandi is quite different. Like List say L = [1,2,3,4] is a sequence but it is an iterable, the generator is also a sequence but it is an iterator not a iterable. You might be thinking that we already have a mechanism to refer a sequence i.e. say List, then why do we need a generator. To understand this we need to understand the memory utilization by list and generator. Say, we are having a list with many numbers, when we create a list then it will take some time , secondly these numbers will get stored in to the memory i.e memory usage will also be on higher side. But, I the case of Generators, at one time only one number is generated and the same is used for further processing, i.e. both time and memory space are saved, they are comparatively quite less in case of generators. So, while processing the list entire list is loaded and processed, but in generators one by one elements are generated and are processed accordingly.

Now you might be thinking that, when to use lists then ? the answer is that when you need to use your sequence again and again (may be to perform some functionality) then list is the best option, but when you simply need to use the sequence for one time only, then its better to go for generators, you will understand this, later in this section only.

Lets learn how to write a generator, for this you may use two techniques, i.e. you may use generator function or generator comprehension, as technique to develop your own generator. Generator comprehension is the technique which is quite similar to list comprehension, which is used to generate list.

Firstly we will discuss about generator function, say we need to define a function which is suppose to take a number as an argument and that function is suppose to print number from 1 to that number say 10

We can write the following code to achieve the task:

```
defnums(n)

fori in range (1, n+1) :          # n+1 because for loop goes upto n-1 th term

        print(i)

nums(10)        # calling the function nums with n = 10
```

this will print the numbers from 1 to 10, here function is not returning any thing, but simply printing the numbers. This was quite simple, as you leraned in your earlier units, but if we need to develop our generator to do the same task then you need to replace the print command with yield keyword, and the code will be

```
defnums(n)
```

```
fori in range (1, n+1) :            # n+1 because for loop goes upto n-1 th term

        yield(i)

nums(10)
```

This yield keyword will create a generator, on executing this code nothing will be printed, but if in place of nums(10) i.e. the last line of the above code, we write print(nums(10)) then on execution you will come to know a generator object nums is produced.

Note : a) In normal function we either print or return but in generator function we use yield keyword

b) yield is a keyword, and not a function, so we can write ‘yield i’ in place of ‘yield(i)’

Now, lets understand how a generator function works, for this lets again explore the functioning with for loop, refer to the code given below

```
defnums(n)

fori in range (1, n+1) :            # n+1 because for loop goes upto n-1 th term

        yield(i)

numbers = nums(10)

fornum in numbers :

print(num)
```

Executing the above code the sequence of numbers from 1 to 10 will be generated, but if you again execute the for loop then nothing will be printed i.e.

```
defnums(n)

fori in range (1, n+1)  # n+1 because for loop goes upto n-1 th term

        yield(i)

numbers = nums(10) # here nums(10) is a generator, we can transform nums(10) into list by writing list(nums(10))

fornum in numbers :

print(num)
```

```
fornum in numbers :

print(num)
```

execution of the second for loop will not produce any result because the generaors generates the numbers one by one and they are not retained in to the memory as in the case of lists. So the execution of first for loop will produce a sequence from 1 to 10, i.e. the numbers are placed in to the memory one by one, which is there after refreshed, so past instance is lost. Thus the execution of second for loop has no databecausenums(n) function only exists before first for loop not after it. If the nums(n) also exists after first for loop then data for second for loop is also available.

```
defnums(n)

fori in range (1, n+1)  # n+1 because for loop goes upto n-1 th term

        yield(i)

numbers = list(nums(10))  #here nums(10) is a generator, we can transform nums(10) into list by writing list(nums(10))

fornum in numbers :

print(num)

fornum in numbers :

print(num)
```

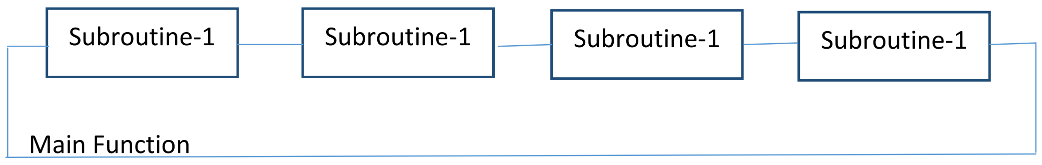
If we re-execute the above code with list and not generator i.e. with list(nums(10)) and not nums(10), then the sequence from 1 to 10 will be printed twice because the content of List persists in the memory, thus the execution of both for loops will produce a separate sequence from 1 to 10.

☞ **Check Your Progress 3**

1) What are generators in Python? Briefly discuss the utility of generators in python
2) Compare Generators and Lists

15.5 CO-ROUTINES

We learned about functions in our earlier units, and we knew that they are also referred as procedures, subroutines, sub-processes etc. In fact a function is packed unit of instructions, required to perform certain task. In a complex function the logic is to divide its working into several self-contained steps, which themselves are functions, such functions are called subroutines or helper functions, these subroutines have single entry point. The coordination of these subroutines is performed by the main function.



The generalized co-routines are referred as subroutines, the working of co-routines relates to cooperative multitasking i.e. when a process voluntarily passes on (yield) the control to another process, periodically or when idle. Co-routines are cooperative that means they link together to form a pipeline. One co-routine may consume input data and send it to other which process it. Finally there may be a co-routine to display result.

This feature of co-routine helps for simultaneous processing of multiple applications. The Co-routines are referred as generalized subroutines, but there is a difference between subroutine and co-routine, the same are given below:

Subroutines	**Co-Routines**
1. Co-routines have many entry points for suspending and resuming execution.	1. Subroutines have single entry point for suspending and resuming execution
2. Co-routine can suspend its execution and transfer control to other co-routine and can resume again execution from the point it left off.	2. Subroutines can't suspend its execution and transfer control to other subroutine and can resume again execution from the point it left off.
3. In Co-routines there is no main function to call co-routines in particular order and coordinate the results.	3. In Subroutines there is main function to call subroutines in particular order and coordinate the results.

From the above discussion it appears that co-routines are quite similar to threads, both seems to do the same job. But, there is a difference in between the thread and the Co-routine, in case of threads, it is the operating system i.e. the run time environment that performs switching in accordance with scheduler. But, in the case Co-routines the decision making for switching is performed by the programmer and programming language. In co-routines the cooperative multitasking, by suspending and resuming at set points is under the control of programmer.

In Python, co-routines are similar to generators but with few extra methods and slight change in how we use yield statement. Generators produce data for iteration while co-routines can also consume data. A generator is essentially a cut down (asymmetric) coroutine. The difference between a coroutine and generator is that a coroutine can accept arguments after it's been initially called, whereas a generator can't.In Python the Co-routines are declared with the async or await syntax, it is the preferred way of writing asyncio applications.

asyncio is a library to write concurrent code using the async/await syntax. asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc. To understand co-routine refer to following example code

Example, the following snippet of code (requires Python 3.7+) prints “hello”, waits 1 second, and then prints “world”:

```
importasyncio

asyncdef main() :

        print (‘hello’)

        awaitasyncio.sleep(1)

        print (‘world’)

asyncio.run(main())
```

It is to be noted that by simply calling a co-routine, it will not be scheduled for the execution. To actually run a co-routine, asyncio provides three main mechanisms, listed below:

1. The asyncio.run() function to run the top-level entry point “main()” function (see the above example.)
2. Awaiting on a co-routine. (see the example given below)

Example - The following snippet of code will print “hello” after waiting for 1 second, and then print “world” after waiting for another 2 seconds:

```
importasyncio

import time

asyncdeftell_after(delay_time, what_to_tell):

        awaitasyncio.sleep(delay_time)

        print(what_to_tell)
```

```
asyncdef main():
        print(f"started at {time.strftime('%X')}")
        awaittell_after(1,'hello')
        awaittell_after(2,'world')
        print(f"finished at {time.strftime('%X')}")
asyncio.run(main())
```

Expected output:

```
Started at 17:11:52
hello
world
finished at 17:11:55
```

3. The asyncio.create_task() function to run coroutines concurrently as asyncio Tasks.

Let's modify the above example and run two tell_after coroutines concurrently:

```
asyncdef main() :
        task1 = asyncio.create_task(tell_after(1,'hello'))
        task2 = asyncio.create_task(tell_after(2,'world'))
        print(f"started at {time.strftime('%X')}")
        # wait until both tasks are completed (should take around 2 seconds)
        awaittell_after(1,'hello')
        awaittell_after(2,'world')
        print(f"finished at {time.strftime('%X')}")
```

Expected Output:

```
Started at 17:24:32
hello
world
```

```
finished at 17:24:34
```

Note that expected output now shows that the snippet runs 1 second faster than before

☞ **Check Your Progress 4**

1) What are Co-routines? How they support cooperative multi-tasking in python
2) Compare Subroutines and Co-routines
3) How Co-routines differ from threads

15.6 SUMMARY

In this you learned about decorators, a way to enhance the functionality of already written functions, which ia useful concept for code reusability. Further, the discussion was enhanced to the concepts of iterables and iterators, through the understanding of the execution of For loop. There after the concept of generators was discussed where the concept of yield keyword and print command were mentioned, the comparative analysis between the generator and a function also clars the concept of performance improvement in python programming. Finally, the understanding of co-routines cleared the learners understanding towards the cooperative multitasking.

UNIT 16 DATA ACCESS USING PYTHON

Structure

16.1 INTRODUCTION

Python is the most popular high-level programming language that can be used for real-world programming. It is a dynamic and object-oriented programming language thatcan be used in a vast domain of applications, especially data handling. Python was designed as a very user-friendly general-purpose language with a collection of modules and packages and gained popularity in the recent times.Wheneverwe think of machine learning, big data handling, dataanalysis, statisticalalgorithms, python is the only name which comes first in our mind. It is a dynamically typed language, which makes it highly flexible. Furthermore, creating a database and linking of the database is very fast and secure.It makes the program to compile and run in an extremely flexible environment can support different styles of programming, including structural and object-oriented. Its ability to use modular components that were designed in other programming languages allows a programmer to embed the code written in python for any interface. Due to its enormous features, python has become the first choice of learners in the field of data science and machine learning.This unit is specially designed for beginners to create and connect database in python. You will learn to connect data through SQL as well as CSV files with the help of real dataset (PIMA Indian Dataset), used most commonly by researchers and academia in data analysis.

16.2 DATABASE CONCEPTS

The database is a collection of an interrelated, organizedform of persistent data. It has properties of integrity and redundancy to serve numerous applications. It is a storehouse of all important information, not only for large business enterprises rather for anyone who is handling data even at the micro-level. Traditionally, we used to store data using file systems, which eventually taken by advanced Database Management Systems software,whichstores, manipulates and secure the data in the most promising manner. To access the data stored in the database, one must know query language like SQL, MySQLetc., which can retrieve the information from the database server and passes it to the client. Thus, client can easily approach the server for day to day transactions by hitting the database without worrying about the load on the processor as it is very fast and consumes power to run the desired query only.

The nature of serendipity in the Python environment is an example of data analysis in machine learning and datascience.Python is a bucket of packages and libraries which enables us to perform data classification,prediction and analysis in a very efficient way.

Now the question is, the database is stored in different forms in different software's like Oracle, MS-Access etc.,so how any programming language like python can accessit.For that, we need to learn about database connectivity in python. In this unit you will learn how to create and connect database in python using MySQL along with basic knowledge of database and its features.

Database Management system DBMS

Database management systems are software that serves the special purpose of storing,managing, extracting and modifying data from a database. It contains a set of programs that allows access to data contained in a database. DBMS also interfaces with application programs so that data contained in the database can be used by multiple application and users. It handles all access to the database and responsible for applying the authorization checks and validation procedures. It acts as a bridge between users and data, and the bridge is crossed using special query language like SQL, MYSQL wrote in high-level languages.

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows −

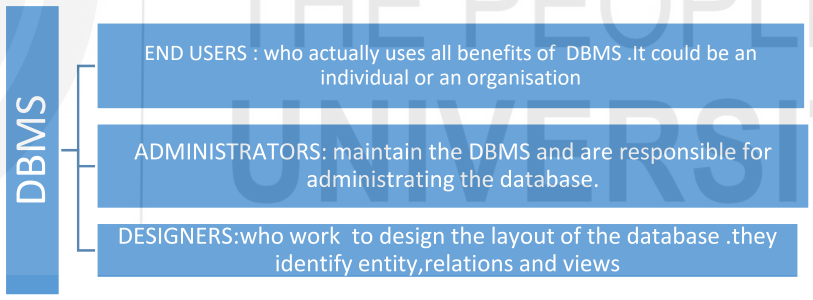

DBMS
END USERS : who actually uses all benefits of DBMS .It could be an individual or an organisation
ADMINISTRATORS: maintain the DBMS and are responsible for administrating the database.
DESIGNERS:who work to design the layout of the database .they identify entity,relations and views

Fig 16.1 (a)Basic structure of users of DBMS

As we discussed no information processing is possible without a database. It is the hub of data in the field of information technology.Some of the major advantages of the database are listed below:

- **Real-world entity** − A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behaviour and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Reduced data redundancy**: Centralized control of database of avoids unnecessary duplication of data and effectively reduces the total amount of data storage required, unlike the non-database system in which each department is maintaining its own copy of data. Avoiding duplication of data also results in

the elimination of data inconsistencies that tend to be present in duplicate data files.

- **Data consistency:** Centralized control over data, ensures data consistency as there is no duplication of data. In the non-database system, there are multiple copies of the same data with each department, whenever the data is updated, the changes are not reflected in each department which leads to data inconsistency. For example, change in address of the student is reflected in the teacher's record but not in account department, which results in inconsistent data.
- **Shared data**: A database allows the sharing of data by multiple application, users and programs like a database of students can be shared by the teachers for making results as well as admin department to keep track of the fees account.
- **Greater data integrity and independence from applications programs**: Integrity of data means that data stored in the database is both accurate and consistent. Centralized control ensures that adequate checks are incorporated in a database while entering data so that DBMS provide data integrity. DBMS provide sufficient validations to make sure that data fall within a specified range and are of the correct format.
- **Improved data control**: In many organizations, where typically each application has its own private files. This makes the operational data widely dispersed and difficult to control. Central database provide the easy maintenance of the database Access to users through the use of host and query languages
- **Improved data security:**The data is protected in a central system. Data is more secured against unauthorized access control with authentication and encoding mechanism. Different checks and rights can be applied for the access of information from the database
- Reduced data entry, storage, and retrieval costs
- Facilitated development of new applications program

Limitations of Database:

- **Substantial hardware and software start-up costs**:Extensive conversion costs is involved in moving from a file-based system to a database system
- **Database systems are complex**, difficult, and time-consuming to design.
- Damage to database affects virtually all applications programs
- Initial training required for all programmers and users.
- Cost: The cost of required hardware, DB development, and DB maintenance is high.
- Complexity: Due to its complexity, the user should understand it well enough to use it efficiently and effectively.

DBMS Architecture

DBMS is designed on the basis of architecture. This design can be of various forms centralized, client-server systems, parallel systems, distributed and hierarchical. The architecture of a DBMS can be seen as either a single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced. Generally, three-tier architecture is followed, which consist of three layers:

1. Presentation layer (laptop,PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server

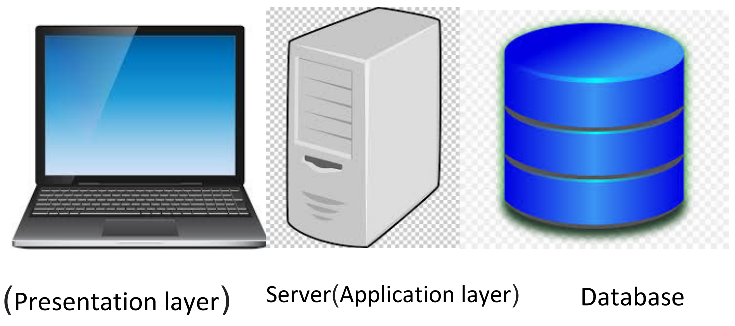


Fig 16.1(b)Three tier architecture of DBMS

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Data models

Once the architecture of DBMS is designed, then the next phase is modelling. The data model represents the logical structure of the database. It decides how the data will be stored, connected to each other, processed and stored. Many data models were proposed to store the data like network model, hierarchical model and relational model where each new model was invented with improved features. The first model was based on **Flat File,** where data is stored in a single table in a file, which could be a plain text file or binary file. It is suitable only for a small amount of data. Records follow a uniform format, and there are no structures for indexing or recognizing relationships between records. Later on, the concept of relation is introduced where data is stored in the form of multiple tables and tables are linked using a common field.It is suitable for handling medium to a large amount of data. This is the most promising model ever implemented for DBMS solutions. So,let us see in detail about the structure and features of the relational model.

Entity-Relationship Model

An **Entity-relationship model (ER model)** describes the logical structure of a database with the help of a diagram, which is known as an **Entity Relationship Diagram (ER Diagram).**It is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates an entity set, relationship set, general attributes and constraints. ER Model is best used for the conceptual design of a database. ER Model is based on − Entities and their attributes. Relationships among entities. These concepts are explained below.

- ***Entity***: An entity is a real-world object which can be easily identifiable like in an employee database, employee, department, products can be considered as entities.

- ***Attributes***: Entities are represented by means of having properties called **attributes**. Every attribute is defined by its set of values called domain. For example, in an employee database, an employee is considered as an entity. An employee has various attributes like empname, age, department, salary etc.
- ***Relationship:*** describes the logical association among entities. These relationships are mapped with entities in various ways, and the number of association between two entities defines mapping cardinalities. Mapping cardinalities are one to one, one to many,many to many and many to one.For example, a doctor can have many patients in a database which shows one to many relationships.

These E-R diagrams are represented by special symbols which are listed below:

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

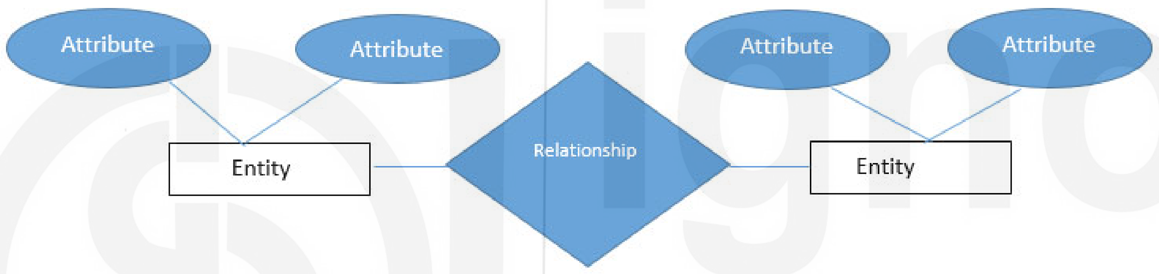


Fig 16.1(c) E-R diagram

Now let us take an example of an organization where they want to design an entity relationship between various departments and their employees. So, In the following E-Rdiagram we have taken two entities Employee and Department and showing their relationship. The relationship between Employee and Department is many to one as a department can have many employees however aemployeecannotwork in multiple departments at the same time. Employee entity has attributes such as Emp_Id, Emp_Name&Emp_Addr and Department entity has attributes such as Dept_ID&Dept_Name.

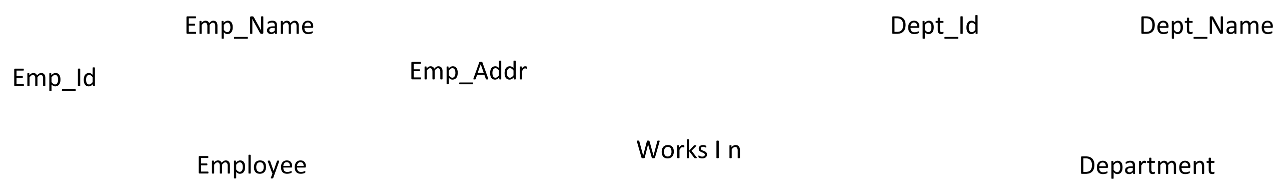


Fig 16.1(d)Sample E-R Diagram showing employee and department relationship.

Structure of Database

A database system is a computer-based system to record and maintain information. The information concerned can be anything of significance to the organization for whose use it is intended.The contents of a database can hold a variety of different things. To make database design more straight-forward, databases contents are divided up into two concepts:

- Schema
- Data

The Schema is the structure of data, whereas the Data are the "facts". Schema can be complex to understand to begin with, but really indicates the rules which the Data must obey. Let us consider a real-world scenario where we want to store facts about employees for an organization. Such facts may include Emp_ name, Emp_address, Date of Birth, and salary. In a database, all the information on all employees would be held in a single storage "container", called a *table*. This table is a tabular object like a spreadsheet page, with different employees as the rows, and the facts (e.g. their names) as columns..Let's call this table EMP, and it could look something like:

Table EMP

Emp_name	Emp_address	Date of Birth	Salary
SmritiKumari	B-15 ,ParkAvenue Delhi	1/3/1986	71000
Samishtha Dutta	D- 45 ,Sector 37,Delhi	7/9/1989	93000
Rahul Sachdeva	C- 6 ,sector 45 Noida	3/2/1982	78000

Fig 16.1(e) Schema of Employee Database

From this information, the *Schema* would define that EMP has four components, "Emp_name","Emp_address","Date of Birth","Salary". As database designers, we can call the columns what we like, a meaningful name helps. In addition to the name, we want to try and make sure that people don't accidentally store a name in the DOB column, or some other silly error. We can say things like:

- Emp_name is a string, holding a minimum of 12 characters.
- Emp_address is a string, holding a minimum of 12 characters
- Date of Birth is a date. The company can put validation for age must be greater than 18 and less than 60 years.
- A salary is a number. It must be greater than zero.

Such rules can be enforced by a database.The three schema architecture separates the user application and physical database. **The internal level schema or Physical Schema or internal Schema**: determines the physical storage structure of the database.**The conceptual Schema** is the high-level description of the whole database. It hides the detail of the physical storage and focuses on the data types, relationships user operations and constraints.**The external view or is the user view** that describes the view of different user groups.

RELATIONAL MODEL

An eminent scientist E.F.Codd proposed a database model,based on the relational structure of data.In this model data is organized in the form of tables,which is acollection of records and each record in a table consist of fields. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. It is proved to be the most effective way of storing data.

Database Design in Relational Model

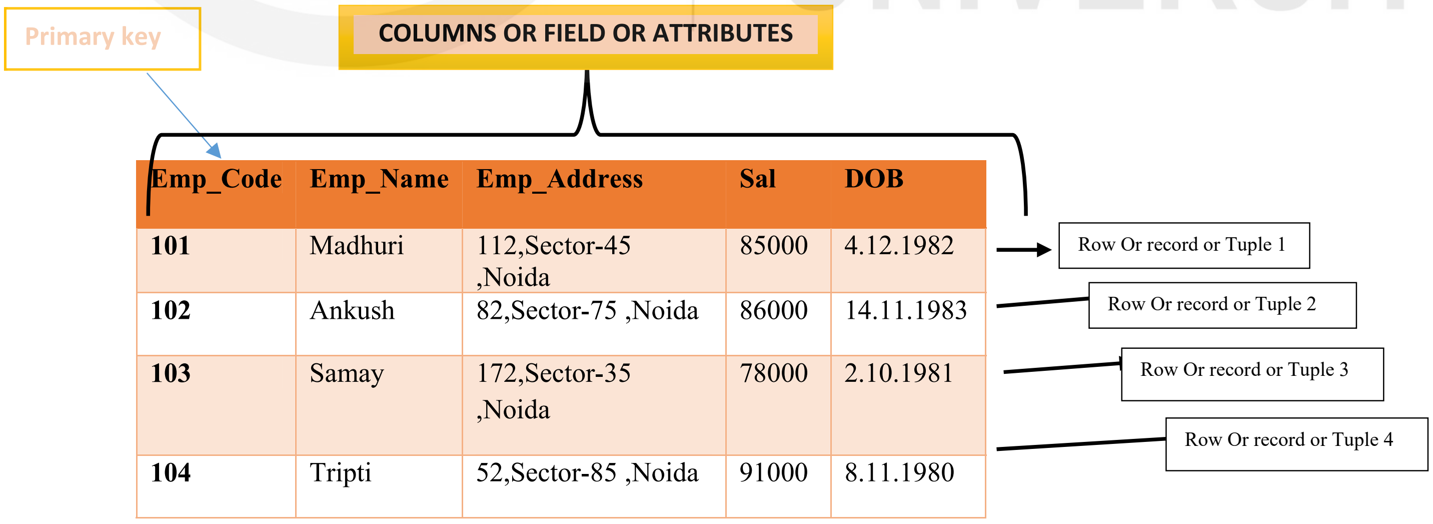
- ***Table***: A table is a set of data elements that is organized using a model of vertical columns and horizontal rows. Each row is identified by a unique key index or the key field.
- ***Columns/Field/Attributes***: A column is a set of data values of a particularly simple type, one for each row of the table. For eg. Emp_Code,Emp_Name,Emp_Address etc.
- ***ROWS/ RECORDS /TUPLES*****:** A row represents a single, data items in a table. Each row in a table represents a set of related data,and every row in the table has the same structure.
- ***DATA TYPES***: -Data types are used to identify the type of data we are going to store in the database.

In the relational model, every tuple must have a unique identification or key based on the data. In this figure, an employee code(Emp_code) is the key that uniquely identifies each tuple in the relation and declares as a primary key. Often, keys are used to join data from two or more relations based on matching identification.

Primary Key: A primary key is a unique value that identifies a row in a table. These keys are also indexed in the database, making itfaster for the database to search a record. All values in the primary key must be unique and NOT NULL, and there can be only one primary key for a table.

Foreign Key: The foreign key identifies a column or set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.

TABLE EMPLOYEE



Emp_Code	Emp_Name	Emp_Address	Sal	DOB
101	Madhuri	112,Sector-45 ,Noida	85000	4.12.1982
102	Ankush	82,Sector-75 ,Noida	86000	14.11.1983
103	Samay	172,Sector-35 ,Noida	78000	2.10.1981
104	Tripti	52,Sector-85 ,Noida	91000	8.11.1980

Fig 16.1(f)Componentsof Relational Database

The relational model indicates that each row in a table is unique. If you allow duplicate rows in a table, then there's no way to uniquely address a given row via programming. This creates all sorts of ambiguities and problems that are best avoided. You guarantee uniqueness for a table by designating a primary key—a column that contains unique values for a table. Each table can have only one primary key, even though several columns or combination of columns may contain unique values.

All columns (or combination of columns) in a table with unique values are referred to as candidate keys, from which the primary key must be drawn. All other candidate key columns are referred to as alternate keys. Keys can be simple or composite. A simple key is a key made up of one column, whereas a composite key is made up of two or more columns.

The relational model also includes concepts of foreign keys, which are primary keys in one relation,that are kept as non-primary key in another relation ,to allow for the joining of data.

Now let us see how to choose the primary key in the table.Consider the Product table given below:

Relating to the employee Table presented in the last section, Now define a second table, order as shown in the figure provided below:

PRODUCT TABLE

Foreign Key in Product Table

Product_ID	EmpCode	Order_Date
P101	101	5/1/2020
P201	102	10/4/2020
P301	103	15/5/2020
P401	104	20/6/2020

Fig 16.1(g)   Demo table of Foreign Key

Product_ID is the PrimaryKeyin above table Product.Emp_code is considered a FOREIGN KEY in ProductTable, since it can be used to refer to given Product in the Product table.

COMMANDS TO DEFINE STRUCTURE AND MANIPULATE THE DATA

In a database, we can define the structure of the data and manipulate the data using some commands.The most common data manipulation language is SQL. SQL commands are instructions used to communicate with the database to perform a specific task that works with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- **Data Definition Language (DDL) -** These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- **Data Manipulation Language (DML) -** These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.
- **Transaction Control Language (TCL) -** These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- **Data Control Language (DCL) -** These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.
- Some of the commonly used datatyes in SQL AND MYSQL are listed below:

Data type	Description
CHAR(n)	Character string, fixed length n ,with a maximum size of 2000 characters . Values of this data type must be enclosed in single quotes '' .the storage size of the char value is equal to the maximum size for this column i.e. nColumns , that will not be used for arithmetic operations usually are assigned data types of char.
CHARACTER ,VARYING(n) or VARCHAR(n) ,VARCHAR2(n)	Variable length character string , can store upto 4000 characters . varchar is a variable-length data type, the storage size of the varchar value is the actual length of the data entered, not the maximum size for this column i.e .n
INT	A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647.
DATE	This data type allows to store valid date type data from January 1, 4712 BC to December 31, 4712 AD with standard oracle data format DD-MM-YY.
TIME	Stores the time in a HH:MM:SS format.
TIMESTAMP	A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).

SQL COMMANDS

DDL COMMANDS[CREATE, ATLTER,DROP]

- **CREATE**

It is the mostly used, Data definition command in SQL . It is used to create table index or view. This command describes the layout of the table. The create statement specifies the name of the table , names and types of each column of the table . Each table must have at least one column. The general syntax for creating table is shown below:

Syntax for CREATE TABLE is:

```
CREATE TABLE <table name>
    (<attribute name><data type>[<size>]<coloumn constraint>,
     (<attribute name><data type>[<size>]<coloumn constraint>,
        ………………………………………………………………..)
```

where

<table name > **:** It is the name, of the table to be created

<attribute name > : It is the name of the column heading or the field in a table.

<data Type> : It defines the type of values that can be stored in the field or the column of the table .

Example:

```
CREATE TABLE Student
(       roll_no  number (5),
        name     char(20),
        birth_data date ) ;
```

- **ALTER**

Alter is a DDL command in SQL which is used to perform the following operations in table.

1. Add column to the existing table
4. To modify the column of the existing table.

- **Adding column(s) to a table**

To add a column to an existing table, the ALTER TABLE syntax is:

ALTER TABLE table_name (ADD column1_name column-definition);

Example :

ALTER TABLE PRODUCT ADD (SALESvarchar2 (50));

This will add a column called *sales* to the *product* table.

- **Modifying column(s) in a table**

To modify a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
    MODIFY (column1_name column_type ,
            [column2_name column_type,]
            ………………………………);
```

Example:

```
ALTER TABLE PRODUCT
 MODIFY Prod_name   varchar2(100)    not null;
```

This will modify the column called *prod_name* to be a data type of varchar2(100) and force the column to not allow null values.

- **Drop column(s) in a table**

To drop a column in an existing table, the ALTER TABLE syntax is:

**ALTER TABLE table_name
DROP COLUMN column_name;**

Example:

ALTER TABLE supplier DROP COLUMN Prod_name;

This will drop the column called *prod_name* from the table called *Product.*

DML COMMANDS –[INSERT,UPDATE,DELETE]

- INSERT Statement is used to add new rows of data to a tablet.The value of each field of the record is inserted in the table.

Syntax :

**INSERT INTO <table _name> [(col1, col2, col3,...colN)]
VALUES (value1, value2, value3,...valueN);
where*col1, col2,...colN -- the names of the columns in the table into which you want to insert data .***

Example:If you want to insert a row to the employee table, the query would be like,

INSERT INTO employee (emp_id, emp_name, emp_dept, age, salary)

VALUES (105, 'Vaishnavi', 'Medical', 27, 33000);

NOTE: *When adding a row, only the characters or date values should be enclosed with single quotes.*

- **SQL SELECT Statement**

The most commonly used SQL command is SELECT statement. The SQL SELECT statement, the only data retrieval in SQL , used to query or retrieve data from a table in the database.
Syntax of SQL SELECT Statement:

**SELECT [distinct] *column_list* FROM *table-name*
[WHERE Clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];**

- ***table-name* is the name of the table from which the information is retrieved.**
- ***column_list* includes one or more columns from which data is retrieved.**
- **The code within the brackets are optional.**
  Example : Write the query in SQL to perform the following in the given Table Student.

R.No	Name	Course	Age	Stream	Sports	Marks	Grade
11	Rajat	BCA	15	Science	Cricket	78	B
12	Anuja	MCA	16	Science	Football	80	B
13	Munil	BCA	15	Science	Cricket	78	B
14	Sonia	MCA	16	Humanities	Badminton	95	A
15	Adya	MCA	15	Commerce	Chess	96	A

a. Display the name of all students from the table Student .
b. Display the name of students who are in course 'MCA'
c. Display the name of students who play 'Chess'
d. Display the name of students according to the marks obtained in descending order.

Solution

```
a. SELECT * FROM student;

   SELECT ALL FROM student;

b. SELECT * FROM student where course='MCA';
c. SELECT * FROM student where sports='Chess';
d. SELECT * FROM student ORDERBY Marks Desc;
```

Note: SQL is case insensitive

- **UPDATE**

Update statement is used to modify the existing data of the tables .

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE <condition>
```

Remember : *The WHERE clause in the UPDATE syntax specifies which record or records that should be updated. If you remove the WHERE clause, all records will be updated by default.*

Example :Modify the marks of Anuja from 80 to 87 which was entered wrongly.

```
UPDATE student

SET marks=87

WHERE name='Anuja' and marks='80';
```

- **DELETE Statement**

The DELETE statement allows you to delete a single record or multiple records or all records from the table.

Syntax:

```
DELETE FROM table
WHERE <condition>;
```

Example :

DELETE FROM Student WHERE course = 'MCA';

This would delete all records from the Student table where the course opted by students is MCA

- **DCL (Data Control Language)**

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
Examples of DCL commands:

  - GRANT-gives user's access privileges to database.
  - REVOKE-withdraw user's access privileges given by using the GRANT command.

- **TCL (transaction Control Language)**

TCL commands deals with the transaction within the database.
Examples of TCL commands:

  - COMMIT– commits a Transaction.
  - ROLLBACK– rollbacks a transaction in case of any error occurs.
  - SAVEPOINT–sets a savepoint within a transaction.
  - SET TRANSACTION–specify characteristics for the transaction.

CHECK YOUR PROGRESS:

1. Define the term database. List its four important features.
2. What do you understand by DDL and DML?
3. How will you explain the term entity and attributes in the table?
4. What is Primary Key?

16.3 CREATING DATABASE

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces follow this standard. Python Database API supports a wide range of database servers such as **MySQL,** PostgreSQL, Microsoft SQL Server 2000,Informix,Interbase,Oracle,Sybase. As a data scientist, you have the freedom to choose the appropriate server for our project.To do so,we need to download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −

- Importing the API module.
- Acquiring a connection with the database.

- Issuing SQL statements and stored procedures.
- Closing the connection

ANACONDA
JUPYTER /SPYDER
NOTEBOOK

MYSQL SERVER
5.1.33

Fig: shows all applications required to install for setting up database connectivity in python with mysql.

Steps for setting python In Anaconda for Database connectivity

1. Install **Anaconda** — https://www.anaconda.com/distribution/
2. Select **python version 3.7 (preferably**)
3. MySQl Server (any version: we have used 5.1.33)
4. Mysql.connector or MySQLdb

NOTE: Anaconda is a python and R distribution that aims to provide everything you need:

- core python language,
- python packages,
- IDE/editor — Jupyter and Spyder
- Package manager — Conda, (for managing your packages)

Once it is done, we are ready to have database connectivity in python.

Steps to install MySql Setup

1. Download and install **MySql Community server** —

   https://downloads.mysql.com/archives/community/

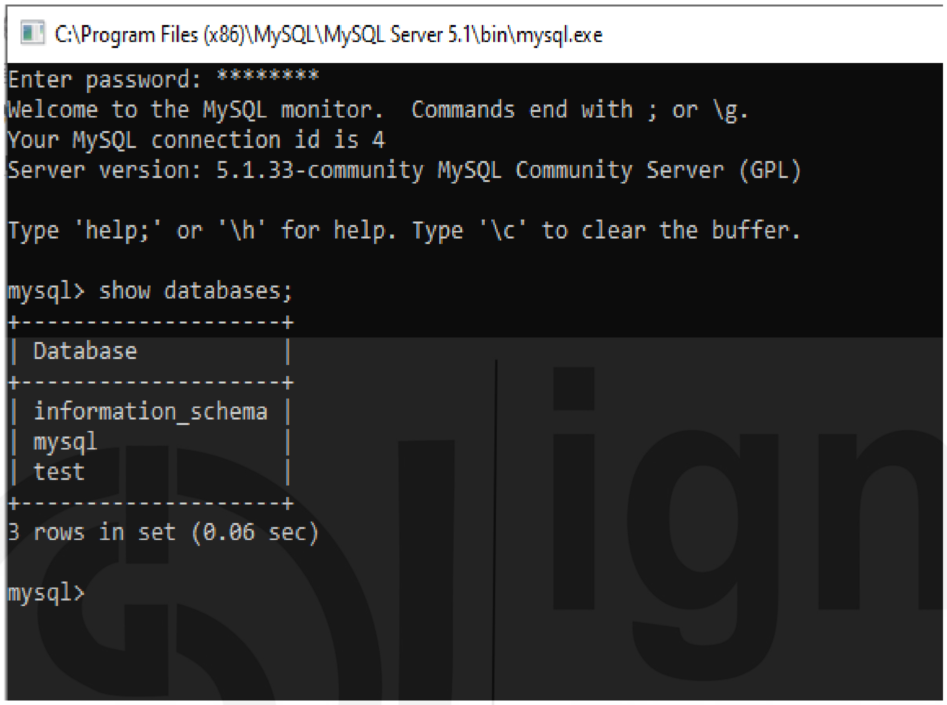
2. During the installation setup, you will be prompted for a "root" password in the server configuration step.

3. Launch workbench, at the home page, setup a new connection profile with the configuration (Connection method: Standard (TCP/IP), Hostname:

127.0.0.1,Port:3306,Username: root, Password: *yourpassword*) and test your connection.

4. Double click on your local instance and it should bring you the schemas view where you can see all your databases and tables.

   The following screenshot gives a glimpse of the screen you may get:



C:\Program Files (x86)\MySQL\MySQL Server 5.1\bin\mysql.exe

```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.06 sec)

mysql>
```

Mysql.connector

It is an interface for connecting to a MySQL database server from python. It implements the Python Database API v2.0 and is built on top of the MySQL C API. It consist of four methods which are used to establish connection. The following methods are listed below:

- **Steps to install mysql.connector in Anaconda :**

  To install mysql.connector navigate to the anaconda prompt and type the following command

1. Using terminal and **conda** to download
   conda install -c anaconda mysql-connector-python

Another way to install mysql.connector is on windows command prompt for using Python d

1. Download Mysql API, exe file and install it.(click here to download)

2. Install mysql-Python Connector (Open command prompt and execute command)

```
>pip install mysql-connector
```

3. Now connect Mysql server using python

4. Write python statement in python shell import mysql.connector

If no error message is shown means mysql connector is properly installed. Mysqlconnector has four important methods which is used to establish and retrieve records from the database.

1. **connect() :** This method is used for creating a connection to our database it have four arguments:
   a. Host Name
   b. Database User Name
   c. Password
   d. Database Name
2. **cursor() :** This method creates a cursor object that is capable for executing sql query on
   database.
3. **execute ():** This method is used for executing SQL query on database. It takes a sql query
   (as string) as an argument.
4. **fetchone() :** This method retrieves the next row of a query result set and returns a single
   sequence, or none if no more rows are available.
5. **close ():** This method close the database connection.

- **In this chapter, we will be using Anaconda Jupyter notebook for demonstration of all programs**

Connect to MySql

1. Launch Anaconda-Navigator to bring you a list of applications available to install. The application we are interested in is **Jupyter Notebook** which is a web-based python IDE. Install and launch it.

2. In the notebook, create a new python file. In the first cell, write the following code to test the mysql connection.

```
importmysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
passwd="password",charset='utf8'
)

print(mydb)
```

```
Optional required only
when this error comes:
ProgrammingError:
1115   (42000):   Unknown
character set: 'utf8mb4'
```

3. If successful, you should get an object returned with its memory address

```
<mysql.connector.connection_cext.CMySQLConnection object at 0x10b4e1320>
```

Cursor object: The MySQLCursor class instantiates objects that can execute operations such as SQL statements. Cursor objects interact with the MySQL server using a MySQLConnection object.

How to create a cursor object and use it import mysql.connector

Example

Following is the example of connecting with MySQL and display the version of the database.

```
importmysql.connector

# Open database connection


mydb = mysql.connector.connect (
  host="localhost",
  user="root",
passwd="admin123",charset='utf8'
)

# prepare a cursor object using cursor() method

mycursor=mydb.cursor()

# execute SQL query using execute() method.

mycursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.

data=mycursor.fetchone()

print ("Database version :",data)

# disconnect from server

mydb.close()
```

creating a connection to our database

creates a cursor object that is capable for executing sql query on database.

retrieves the next row of a query result set

close the database connection.

While running this script, it is producing the following result in my machine. (It could change in your system)

```
Database version : ('5.1.33-community',)
```

In the above code we are creating a cursor to execute the sql query to print database version next line executes the sql query show databases and store result in mycursor as collection ,whose values are being fetched in data. On execution of above program cursor will execute the query and print version of database shown.

Creating Database

1. Let's create a database called *TEST_DB IN Python Anaaconda* .

```
importmysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
passwd="admin123",charset='utf8'
)
mycursor = mydb.cursor()
mycursor.execute("CREATEDATABASETEST_DB")
```

Next, we will try to connect to this new database.

```
importmysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
passwd="admin123",charset='utf8',database= TEST_DB
)
```

CHECK YOUR PROGRESS

1. How will you create a database in mysql .Write the command to create a database named as Start_db.
2. Name the four arguments required to set connection with mysql.connector.connect.

16.4 QUERYING DATABASE

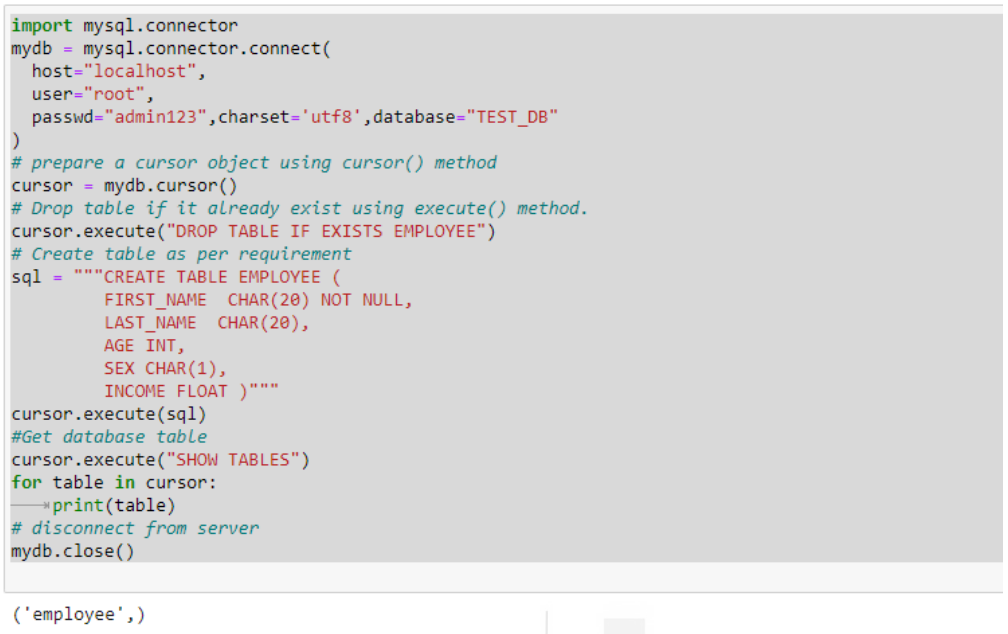
In this section, you will learn to execute SQL queries at run time of the Anaconda environment. To perform SQL queries you need to create a table in the database and then operations like insert, select, update and delete are shown with examples.

- **Creating Table**

Once a database connection is established, we are ready to create tables or records into the database tables using the **execute** method of the created cursor.

Example

To create table EMPLOYEE in database TEST_DB with following data FIRST_NAME,LAST_NAME,AGE,SEX,INCOME.



```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="admin123",charset='utf8',database="TEST_DB"
)
# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""
cursor.execute(sql)
#Get database table
cursor.execute("SHOW TABLES")
for table in cursor:
    print(table)
# disconnect from server
mydb.close()
```

```
('employee',)
```

- **How to change table structure/(add,edit,remove column of a table) at run time**

To modify the structure of the table, we just have to use alter table query.

Below program will add a column address in the EMPLOYEE table.

```
import
mysql.connectormydb=mysql.connector.connect(host="localhost",user="root",
passwd ="root",database="school")
mycursor=mydb.cursor()
mycursor.execute("alter table student add (address varchar(2))")
mycursor.execute("desc employee")
for x in mycursor:
print(x)
```

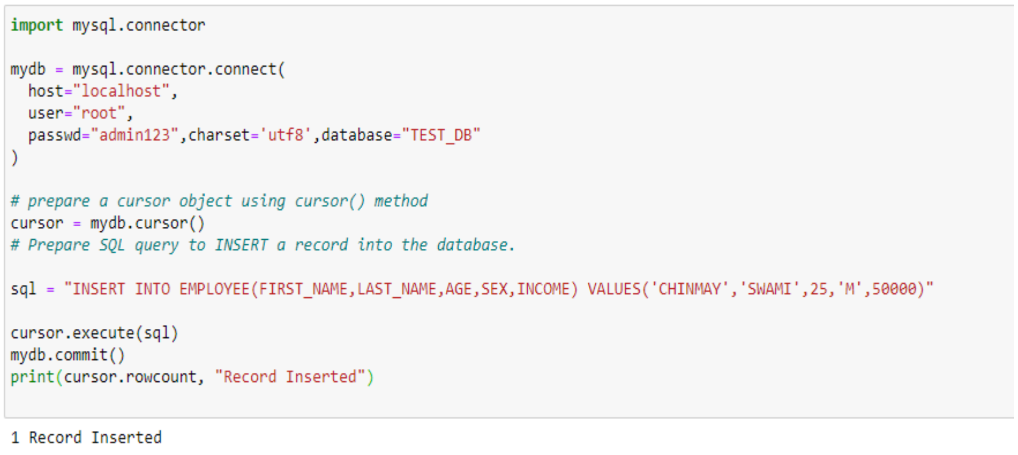
Above program will add a column marks in the table student and will display the structure of the table

- **INSERT Operation**

Insert is used to feed the data in the table created. It is required when you want to create your records into a database table.

Example

The following example, executes SQL *INSERT* statement to create a record into EMPLOYEE table –



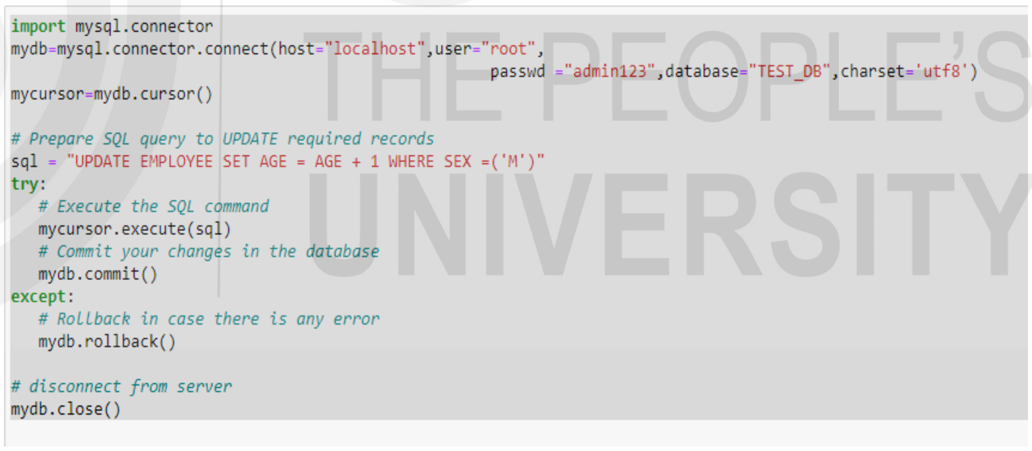
```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="admin123",charset='utf8',database="TEST_DB"
)

# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Prepare SQL query to INSERT a record into the database.

sql = "INSERT INTO EMPLOYEE(FIRST_NAME,LAST_NAME,AGE,SEX,INCOME) VALUES('CHINMAY','SWAMI',25,'M',50000)"

cursor.execute(sql)
mydb.commit()
print(cursor.rowcount, "Record Inserted")
```

```
1 Record Inserted
```

- **UPDATE OPERATION**

UPDATE Operation on any database means to update one or more records, which are already available in the database. Here is an example to show where we run the query to updates all the records having SEX as **'M'**. Here, we increase the AGE of all the males by one year.

Example

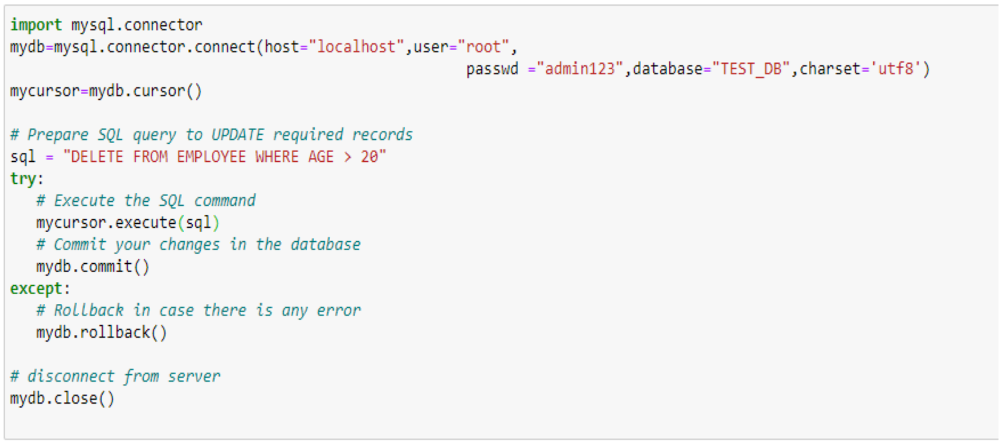


```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",
                                                        passwd ="admin123",database="TEST_DB",charset='utf8')

mycursor=mydb.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX =('M')"
try:
   # Execute the SQL command
   mycursor.execute(sql)
   # Commit your changes in the database
   mydb.commit()
except:
   # Rollback in case there is any error
   mydb.rollback()

# disconnect from server
mydb.close()
```

The above code will update the age of all the male employees in the table EMPLOYEE.

- **DELETE Operation**

DELETE operation is required when you want to delete some records from your database. Following example show the procedure to delete all the records from EMPLOYEE where AGE is more than 20.



```python
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",
                                                    passwd ="admin123",database="TEST_DB",charset='utf8')
mycursor=mydb.cursor()

# Prepare SQL query to UPDATE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > 20"
try:
   # Execute the SQL command
   mycursor.execute(sql)
   # Commit your changes in the database
   mydb.commit()
except:
   # Rollback in case there is any error
   mydb.rollback()

# disconnect from server
mydb.close()
```

The above code will delete all the records from the table EMPLOYEE where age is greater than 20.

Steps to perform SQL queries in the database using Python

Step1. Import mysql.connector

Step 2. Create a variable and assign it to mysql.connector.connect().

Step3. Provide arguments host,user,passwd,database, charset to mysql.connector.connect().

Step4. Call cursor() and assign it to variable.

Step5. Prepare SQL query (Insert or Update or delete)

Step 5. Disconnect from the server using close() function.

Please refer to the above examples with their screenshots which will help in better understanding.

CHECK YOUR PROGRESS:

1. Write the syntax of the following commands in SQL : insert, update, delete

16.5 USING SQL TO GET MORE OUT OF DATABASE

READ Operation

READ operation on any database means to fetch some useful information from the database.

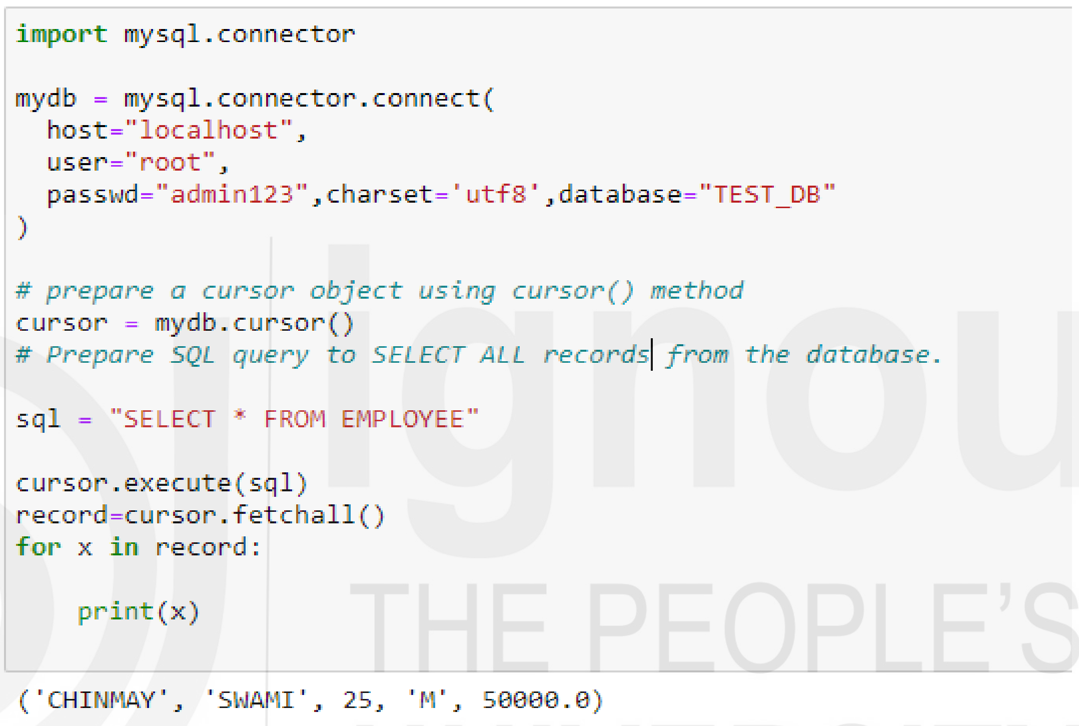
Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch a single record or **fetchall()** method to fetch multiple values from a database table.

- **fetchall()** – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.

- **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

- **fetchall()**

The method fetches all (or all remaining) rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list. The following procedure queries all the records from the EMPLOYEE table



```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="admin123",charset='utf8',database="TEST_DB"
)

# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Prepare SQL query to SELECT ALL records from the database.

sql = "SELECT * FROM EMPLOYEE"

cursor.execute(sql)
record=cursor.fetchall()
for x in record:

    print(x)
```

```
('CHINMAY', 'SWAMI', 25, 'M', 50000.0)
```

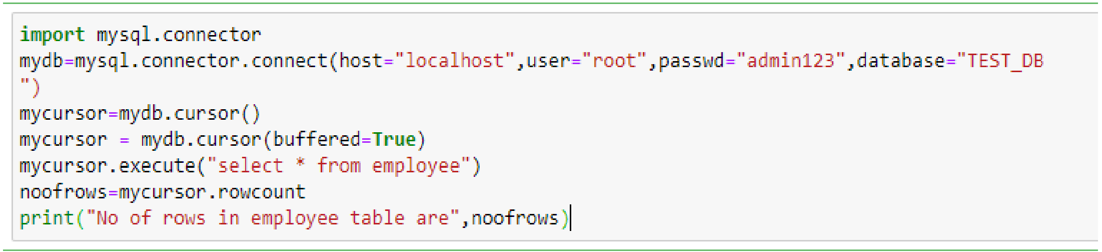
- **fetchone()**

To fetch one record from the table fetchone is used in the same manner as fetchall() is shown in above code. This method retrieves the next row of a query result set and returns a single sequence, or None if no more rows are available. By default, the returned tuple consists of data returned by the MySQL server, converted to Python objects.

```
# Fetch a single row using fetchone() method.

record = cursor.fetchone()
```

- **rowcount()**

Rows affected by the query. We can get a number of rows affected by the query by using rowcount. We will use one SELECT query here.



```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="admin123",database="TEST_DB
")
mycursor=mydb.cursor()
mycursor = mydb.cursor(buffered=True)
mycursor.execute("select * from employee")
noofrows=mycursor.rowcount
print("No of rows in employee table are",noofrows)
```

In the above code buffered=True. We have used mycursor as buffered cursor which fetches rows and buffers them after getting output from MySQL database. It is used as an iterator, but there is no point in using buffered cursor for the single record as in such case if we don't use a buffered cursor, then we will get -1 as output from rowcount.

- **Manage Database Transaction**

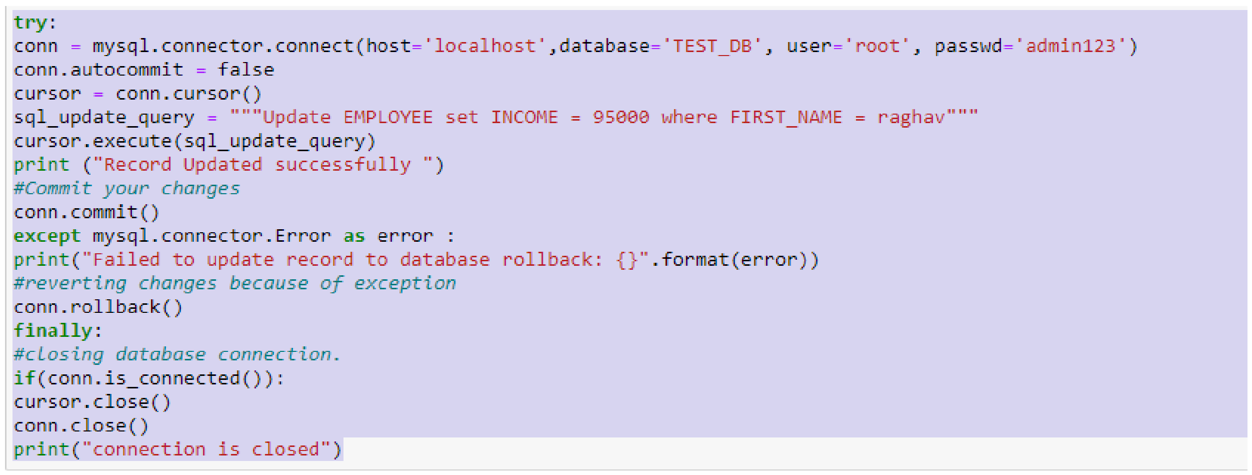
Database transaction represents a single unit of work. Any operation which modifies the state of the MySQL database is a transaction. Python MySQL Connector provides the following method to manage database transactions.

  - **Commit** – MySQLConnection.commit() method sends a COMMIT statement to the MySQL server, committing the current transaction.
  - **Rollback** – MySQLConnection.rollback revert the changes made by the current transaction.
  - **autoCommit** – MySQLConnection.auto-commit value can be assigned as True or False to enable or disable the auto-commit feature of MySQL. By default, its value is False.

Steps to perform commit,rollback and auto-commit.

Step 1. Import the MySQL connector python module
Step 2. After a successful MySQL connection, set auto-commit to false, i.e., we need to commit
the transaction only when both the transactions complete successfully.
Step 3. Call the cursor() function .
Step4. Execute the queries using a cursor. Execute method.
Step 5 .After successful execution of the queries, commit your changes to the database using
a conn.commit() .In case of an exception or failure of one of the queries, you can revert your changes using a conn.rollback()
We placed all our code in the "try-except" block to catch the database exceptions that may occur during the process.

The following code shows the role of commit, rollback and auto-commit while performing SQL queries in python.



```
try:
conn = mysql.connector.connect(host='localhost',database='TEST_DB', user='root', passwd='admin123')
conn.autocommit = false
cursor = conn.cursor()
sql_update_query = """Update EMPLOYEE set INCOME = 95000 where FIRST_NAME = raghav"""
cursor.execute(sql_update_query)
print ("Record Updated successfully ")
#Commit your changes
conn.commit()
except mysql.connector.Error as error :
print("Failed to update record to database rollback: {}".format(error))
#reverting changes because of exception
conn.rollback()
finally:
#closing database connection.
if(conn.is_connected()):
cursor.close()
conn.close()
print("connection is closed")
```

In the above code if update query is successfully executed then commit() method will be executed otherwise,anexception error part will be executed.Rollback issued to revert of update query if happened due to error.Finally, we are closing cursor as well as connection. Here the purpose of conn. Auto-commit = false is to activate the role of rollback else rollback will not work.

CHECK YOUR PROGRESS:

1. What is the purpose of rollback and commit in SQL.
2. Fill in the blank provided :
   importmysql.connector

       mydb = mysql.________________.connect(
       host="localhost",
       _____="myusername",
       passwd="mypassword"
)
       mycursor = mydb.__________()
       mycursor.___________("CREATE DATABASE mydatabase")

16.6 CSV FILES IN PYTHON

As we have studied various operations on databases, the next thing which comes in our way to implement real dataset for machine learning purpose using python. For the same, we should be aware about all the basic types of database connectivity among which the most common type for connecting real dataset is CSV (Comma **Separated Values).**

CSV files are ordinarily made by programs that handle a lot of information. Itis just like a text file in a human-readable format which is used to store tabular data in a spreadsheet or database.They are a helpful method to send out information from spreadsheets and data sets just as import or use it in different projects. For instance, you may trade the aftereffects of an information mining project to a CSV document and afterwards import that into a spreadsheet to dissect the information, create charts for an introduction, or set up a report for distribution. The separator character of CSV files is called a delimiter.Default delimiter is comma (,) others are tab (\t), (:), (;) etc.

CSV documents are exceptionally simple to work for database connectivity. Any language that underpins text record info and string control (like python) can work with CSV documents straightforwardly.

Some of the features of CSV are highlighted below, which makes it exceptionally useful in the analysis of the large dataset.

- Simple and easy to use.
- Can store a large amount of data.
- He was preferred to import and export format for data handling.
- Each line of the file is a record.
- Each record consist of fields separated by commas(delimiter)
- They are used for storing tabular data in a spreadsheet or database.

Let us see how to import CSV file in python.In this unit, we will be using pandas module in python to import the CSVfile.*Pandas* is a powerful Python package that can be used to perform statistical analysis. In this chapter, you'll also see how to use Pandas to calculate stats from an imported CSV file.

CASE STUDY OF PIMA INDIAN DATASET

The dataset used here isPima Indian diabetes data for learning purpose.Pima Indian diabetes dataset describes the medical records for Pima Indiansand whether or not each patient will have an onset of diabetes within given years.

Fields description follow:

preg = number of times pregnant

plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

pres = Diastolic blood pressure (mm Hg)

skin = Triceps skin fold thickness (mm)

test = 2-Hour serum insulin (mu U/ml)

mass = Body mass index (weight in kg/(height in m)^2)

pedi = Diabetes pedigree function

age = Age (years)

class = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

(This dataset can be downloaded from
https://www.kaggle.com/kumargh/pimaindiansdiabetescsv?select=pima-indians-diabetes.csv)

It consists of above mentioned eight features, and one class variableandis very commonly used in the research of diabetes.Given below are the steps and the code to import CSV file in python.

Steps to import a CSV file into Python Using Pandas.

Step1. Import python module pandas.

Step2. Capture the file path where CSV file is stored (don't forget to include filename and file extension).

Step3. Read the CSV file using read_csv

Step4. Print the desired file.

Step5 .Run the code to generate the output.

```
import pandas as pd

df = pd.read_csv ('F:\pimaindiansdiabetescsv\pima-indians-diabetes.csv')

print (df)
```

Note: #read the csv file (put 'r' before the path string to address any special characters in the path, such as '\'). Don't forget to put the file name at the end of the path + ".csv".

This path is taken for reference; it will change as per the location of the file.

Output

```
6  148  72  35    0  33.6  0.627  50  1
0    1   85  66  29    0  26.6  0.351  31  0
1    8  183  64   0    0  23.3  0.672  32  1
2    1   89  66  23   94  28.1  0.167  21  0
3    0  137  40  35  168  43.1  2.288  33  1
4    5  116  74   0    0  25.6  0.201  30  0
..   ..  ...  ..  ..  ...   ...    ... .. ..
762  10  101  76  48  180  32.9  0.171  63  0
763   2  122  70  27    0  36.8  0.340  27  0
764   5  121  72  23  112  26.2  0.245  30  0
765   1  126  60   0    0  30.1  0.349  47  1
766   1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]
```

The above output consists of 769 rows and ninecolumns.TheCSV file is just like excel file which consists of data in tabular form arranged as rows and columns.But the columns are without heading.In the next example, column headings are provided using

```
Colnames=[ ‘col1  ’,‘ col2’,‘col3’,‘col4 ’]
```

Any userdefined name

These are the field names or column names

- **Calculate stats using Pandas from an Imported CSV File.**

Finally you will learn to calculate the following statistics using the Pandas package:

- Mean
- Total sum

- Maximum
- Minimum
- Count
- Median
- Standard deviation
- Variance

Simply functions for each of the above mentioned stats is available in pandas package which can be easily applied in the following manner.

```
import pandas as pd
colnames=['pre','pgc','dbp','tsf','2hs','bmi','dpf','age','class']
df=pd.read_csv('F:\pimaindiansdiabetescsv\pima-
indiansdiabetes.csv',names=colnames)
mean1 = df['age'].mean()
sum1 = df['age'].sum()
max1 = df['age'].max()
min1 = df['age'].min()
count1 = df['age'].count()
median1 = df['age'].median()
std1 = df['age'].std()
var1 = df['age'].var()

# print block 1
print ('Mean Age: ' + str(mean1))
print ('Sum of Age: ' + str(sum1))
print ('Max Age: ' + str(max1))
print ('Min Age: ' + str(min1))
print ('Count of Age: ' + str(count1))
print ('Median Age: ' + str(median1))
print ('Std of Age: ' + str(std1))
print ('Var of Age: ' + str(var1))
```

Output:

```
Mean Age: 33.240885416666664
Sum of Age: 25529
Max Age: 81
Min Age: 21
Count of Age: 768
Median Age: 29.0
Std of Age: 11.76023154067868
Var of Age: 138.30304589037365
```

Here is a table that summarizes the operations performed in the code:

Variable	Syntax	Description
mean1	df['age'].mean()	Average of all values under the age column.
sum1	df['age'].sum()	Sum of all values under the age column.
max1	df['age'].max	Maximum of all values under the age column.
min1	df['age'].min()	Minimum of all values under the age column.
count1	df['age'].count()	Count of all values under the age column.
median1	df['age'].median()	Median of all values under the age column.

std1	df['age'].std()	Standard deviation all values under the age column.
var1	df['age'].var()	Variance of all values under the age column.

We have learnt how to calculate simple stats using *Pandas and import any dataset for machine learning purpose.*

16.7 SUMMARY

- A database is an organized,logical collection of data.It is designed to facilitate the access by one or more applications programs for easy access and analysis and to minimize data redundancy.
- DBMS is a software system that enables you to store, modify, and extract information from a database. It has been designed systematically so that it can be used by multiple applications and users
- A relational database is a collection of related tables
- An entity is a person place or things or event.
- Tables in the database are entities.
- An attribute is a property of entity. Attributes are columns in the table.
- A relationship is the association between tables in the database.
- Each columns of the table correspond to an attribute of the relation and is named
- Fields : Columns in the table are called fields or attributes.
- Rows of the relation is referred as tuples to the relation . A tuple or row contains all the data of a single instance of the table such as a employee number 201.
- Records : Rows in a table often are called records . Rows are also known as tuples.
- The values for an attribute or a column are drawn from a set of values known as domain.
- Primary key : An attribute or a set of attributes which can uniquely identify each record (tuple) of a relation (table). All values in the primary key must be unique and not Null, and there can be only one primary key for a table.
- Foreign Key : An attribute which is a regular attribute in one table but a primary key in another table.
- Mysql.connector is an interface for connecting to a MySQL database server from python
- **connect**() is a method used for creating a connection to our database.
- Connect has four arguments:hostname,username,password,databasename .
- **fetchall()** − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.

- **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.
- Commit,rollback and autocommit are database transactions commands.

16.8 SOLUTIONS TO CHECK YOUR PROGRESS

Section 16.2Solutions

1. Database is a collection of interrelated, organized form of persistent data. It has properties of integrity and redundancy to serve numerous applications. It is storehouse of all important information, not only for large business enterprises rather for anyone who is handling data even at micro level. Its features are
   - Real world entity.
   - Reduced data redundancy.
   - Data consistency.
   - Sharing of Data.
2. DDL and DML (Refer to section 16.2 SQL Commands)
3. Entity and attributes (Refer to section 16.2 **Entity-Relationship Model)**
4. Primary key: A primary key is a unique value that identifies a row in a table. These keys are also indexed in the database, making it faster for the database to search a record. All values in the primary key must be unique and NOT NULL, and there can be only one primary key for a table.

Section 16.3 Solutions

1. Create a database called *Start_db in Python Anaaconda.*

```
importmysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
passwd="admin123",charset='utf8'
)
mycursor = mydb.cursor()
mycursor.execute("CREATEDATABASEStart_db")
```

Next, we will try to connect to this new database.

```
importmysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
passwd="admin123",charset='utf8',database= Start_db)
```

2. Refer to question 1 second part (host,user,passwd,databse)

Section 16.4 Solutions

1. Syntax of Insert :

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Syntax of Update:
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];

Syntax of Delete:
DELETE FROM table_nameWHERE [condition];

Section 16.5 Solutions

1. Rollback and commit(Refer to section 16.5 Manage Database transaction)

2. importmysql.connector

```
mydb = mysql.connector.connect(
host="localhost",
user="myusername",
passwd="mypassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```

MULTIPLE CHOICE QUESTIONS

1. Which SQL keyword is used to retrieve a maximum value?
   a. MAX
   b. MIN
   c. LARGE
   d. GREATER
2. Which SQL keyword is used to specify conditional search?
   a. SEARCH
   b. WHERE
   c. FIND
   d. FROM
3. A relation /table is a
   a. Collection of fields
   b. collection of records
   c. collection of data
   d. collections of tables
4. ________________is a collection of interrelated data and a set of program to access those data .
   a. table
   b. record
   c. Database
   d. Field
5. A row in a table is called as ___________
   a. table
   b. tuple

c. Database
d. Field

6. Each table in a relational Database must have _______________.
   a. primary key
   b. candidate key
   c. Foreign key
   d. composite key
7. Attribute combinations that can serve as a primary key in a table are
   a. primary key
   b. candidate key
   c. Alternate key
   d. composite key
8. Duplication of data is known as _____________
   a. Data redundancy
   b. Data consistency
   c. data management
   d. data schema
9. _____________ is the total number of rows/tuples in the table .
   a. degree
   b. cardinality
   c. records
   d. domain
10. ___________ is the total number of columns/fields in the table
   a. degree
   b. cardinality
   c. records
   d. domain
11. A collection of fields that contains data about a single entity is called
   a. table
   b. record
   c. database
   d. Field
12. A set of related characters is called as
   a. table
   b. record
   c. database
   d. Field
13 _____________is the independence of application programs from the details of the data representation and data storage.
   a. Data redundancy
   b. Data consistency
   c. data independence
   d. data schema
14. The ___________ is used for creating a connection to the database in python.
   a. connect()
   b. cursor()

c. execute()

d. close()

15. Which of the following module is provided by python to do several operations on the CSV files?

a. py

b. xls

c. csv

d. os

Ans

Ans	1-	a	2-	b	3-	b	4-	c	5-	b
	6-	a	7-	b	8-	a	9-	b	10-	a
	11-	b	12-	d.	13-	c	14-	a.	15-	c