UNIT-5

PROCEDURAL AND OBJECT-ORIENTED PROGRAMMING

Procedural programming is a method of writing software. It is a programming practice centered on the procedures or actions that take place in a program.

Object-oriented programming is centered on objects. Objects are created from abstract data types that encapsulate data and functions together.

There are primarily two methods of programming in use today:

1. procedural and
2. object oriented.

The earliest programming languages were procedural, meaning a program was made of one or more procedures.

procedure simply as a function that performs a specific task such as gathering input from the user, performing calculations, reading or writing files, displaying output, and so on.

procedures operate on data items that are separate from the procedures. In a procedural program, the data items are commonly passed from one procedure to another.

procedural programming is centered on creating procedures (functions), *object oriented programming* (*OOP*) is centered on creating objects.

An *object* is a software entity that contains both **data and procedures**. The data contained in an object is known as the object's *data attributes*. An object's data attributes are simply variables that reference data.

The procedures that an object performs are known as *methods*. An object's methods are functions that perform operations on the object's data attributes.

The object is, conceptually, a self-contained unit that consists of data attributes and methods that operate on the data attributes.

DIAGRAM

OOP addresses the problem of code and data separation through encapsulation and data hiding. *Encapsulation* refers to the combining of data and code into a single object.

Data hiding refers to an object's ability to hide its data attributes from code that is outside the object. Only the object's methods may directly access and make changes to the object's data attributes.

a)Object Reusability

In addition to solving the problems of code and data separation, the use of OOP has also been encouraged by the trend of *object reusability*.

An object is not a stand-alone program, but is used by programs that need its services. For example, Sharon is a programmer who has developed a set of objects for rendering 3D images.

Example of an Object

Data attributes:

- current_second (a value in the range of 0–59)
- current_minute (a value in the range of 0–59)
- current_hour (a value in the range of 1–12)

• alarm_time (a valid hour and minute)
• alarm_is_set (True or False)

Object methods:
• set_time
• set_alarm_time
• set_alarm_on
• set_alarm_off

Methods that can be accessed by entities outside the object are known as *public methods*.

private methods, which are part of the object's private, internal workings.

POLYMORPHISM

Polymorphism allows subclasses to have methods with the same names as methods in their super classes. It gives the ability for a program to call the correct method depending on the type of object that is used to call it.

The term *polymorphism* refers to an object's ability to take different forms. It is a powerful feature of object-oriented programming. In this section, we will look at two essential ingredients of polymorphic behavior:

1. The ability to define a method in a superclass, and then define a method with the same name in a subclass. When a subclass method has the same name as a superclass method, it is often said that the subclass method *overrides* the superclass method.

2. The ability to call the correct version of an overridden method, depending on the type of object that is used to call it. If a subclass object is used to call an overridden method, then the subclass's version of the method is the one that will execute. If a superclass object is used to call an overridden method, then the superclass's version of the method is the one that will execute.

EXAMPLE:

```
class Mammal:

 # The __init__ method accepts an argument for
 # the mammal's species.

 def __init__(self, species):
 self.__species = species

 # The show_species method displays a message
 # indicating the mammal's species.
```

```
 def show_species(self):
 print('I am a', self.__species)

 # The make_sound method is the mammal's
 # way of making a generic sound.
def make_sound(self):
21 print('Grrrrr')
```

a) The isinstance Function

Polymorphism gives us a great deal of flexibility when designing programs.
For example,look at the following function:

```
def show_mammal_info(creature):
creature.show_species()
creature.make_sound()
```

We can pass any object as an argument to this function, and as long as it has a show_species method and a make_sound method, the function will call those methods.

What is MySQL?

PyMySQL is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and contains a pure-Python MySQL client library. The goal of PyMySQL is to be a drop-in replacement for MySQLdb.

Connecting with a Database

Before connecting to a MySQL database, make sure of the following points-

- You have created a database TESTDB.
- You have created a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Python module PyMySQL is installed properly on your machine.

Example

Following is an example of connecting with MySQL database "TESTDB"-

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
cursor.execute("SELECT VERSION()")
data = cursor.fetchone()
print ("Database version : %s " % data)
db.close()
```

a) Creating Database Table

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

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
sql = """CREATE TABLE EMPLOYEE (
FIRST_NAME CHAR(20) NOT NULL,
31
LAST_NAME CHAR(20),
AGE INT,
SEX CHAR(1),
INCOME FLOAT )"""
cursor.execute(sql)
db.close()
```

1. INSERT Operation

The INSERT Operation 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 in the EMPLOYEE table-

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
cursor.execute(sql)
db.commit()
except:
db.rollback()
db.close()
```

2. READ Operation

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

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.

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.
rowcount: This is a read-only attribute and returns the number of rows that were affected by an execute() method.
32
The following procedure queries all the records from EMPLOYEE table having salary more than 1000-

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = "SELECT * FROM EMPLOYEE \ WHERE INCOME > '%d'" % (1000)
try:
cursor.execute(sql)
results = cursor.fetchall()
for row in results:
fname = row[0]
lname = row[1]
age = row[2]
sex = row[3]
income = row[4]
print ("fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \(fname, lname, age, sex,
income))
except:
print ("Error: unable to fecth data")
db.close()
```

This will produce the following result

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

3. Update Operation

UPDATE Operation on any database means to update one or more records, which are already available in the database.
The following procedure updates all the records having SEX as 'M'. Here, we increase the AGE of all the males by one year.

Example

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')

try:
cursor.execute(sql)
db.commit()
except:
db.rollback()
db.close()
```

4. DELETE Operation

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

```
import PyMySQL
db = PyMySQL.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
cursor.execute(sql)
db.commit()
except:
db.rollback()
db.close()
```