**LAB # 03**



**Fall 2022**

**CSE208L Object Oriented Programming Lab**

Submitted by:

Shahzad Bangash(21PWCSE1980)

Suleman Shah(21PWCSE1983)

Ali Asghar(21PWCSE2059)

Class Section: **C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Engr. Sumayyea Salahuddin**

22th December , 2022

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Objectives of the Lab:**

Objectives of the lab are to:

# Clearly understand the purpose and advantages of OOP

# Understand the concept of a Class and Objects

# Develop a basic class containing Data Members and Member Functions

# Use access specifiers to access Class Members

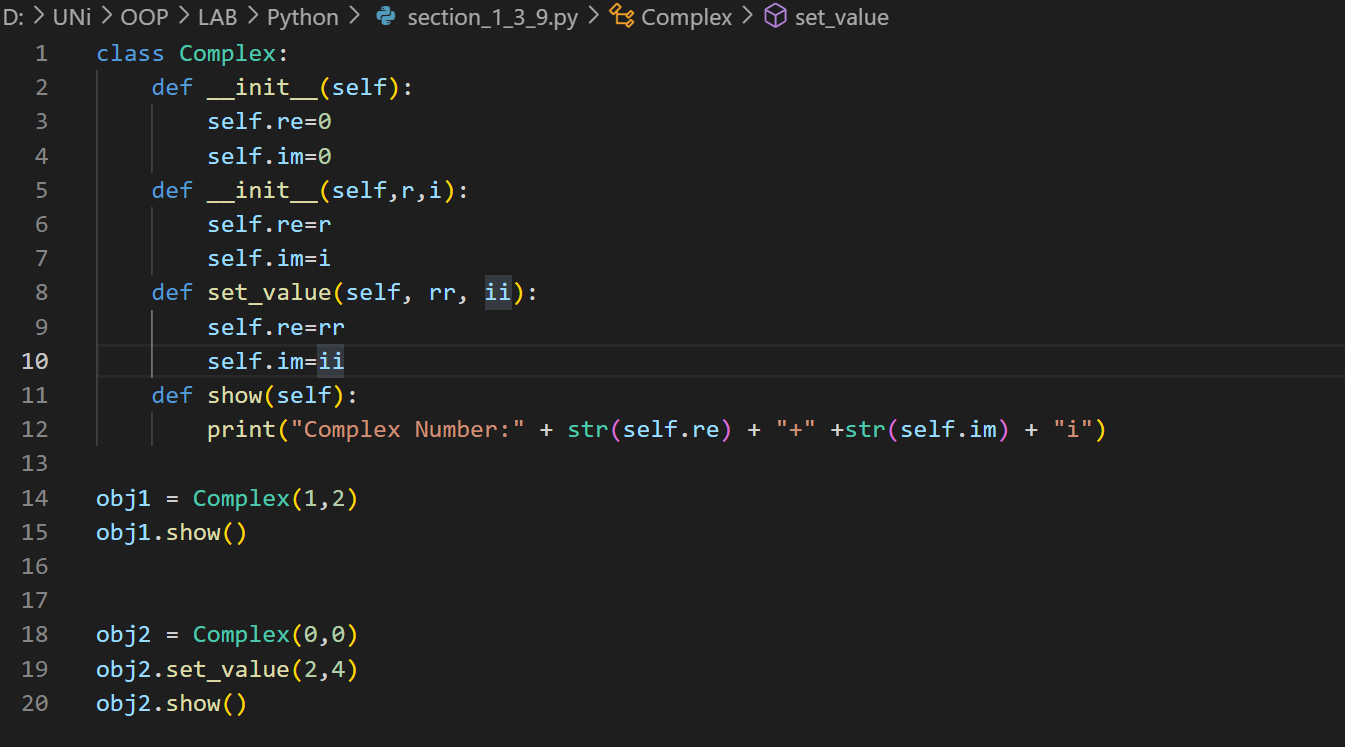
# Make Simple and Overloaded Constructor

# Use the Class Objects and Member Functions to provide and extract data from Object

# Practice with Classes and Objects

**Example # 01(Section 1.3.9)**

**Code:**



**Output:**



**OBSERVATIONS:**

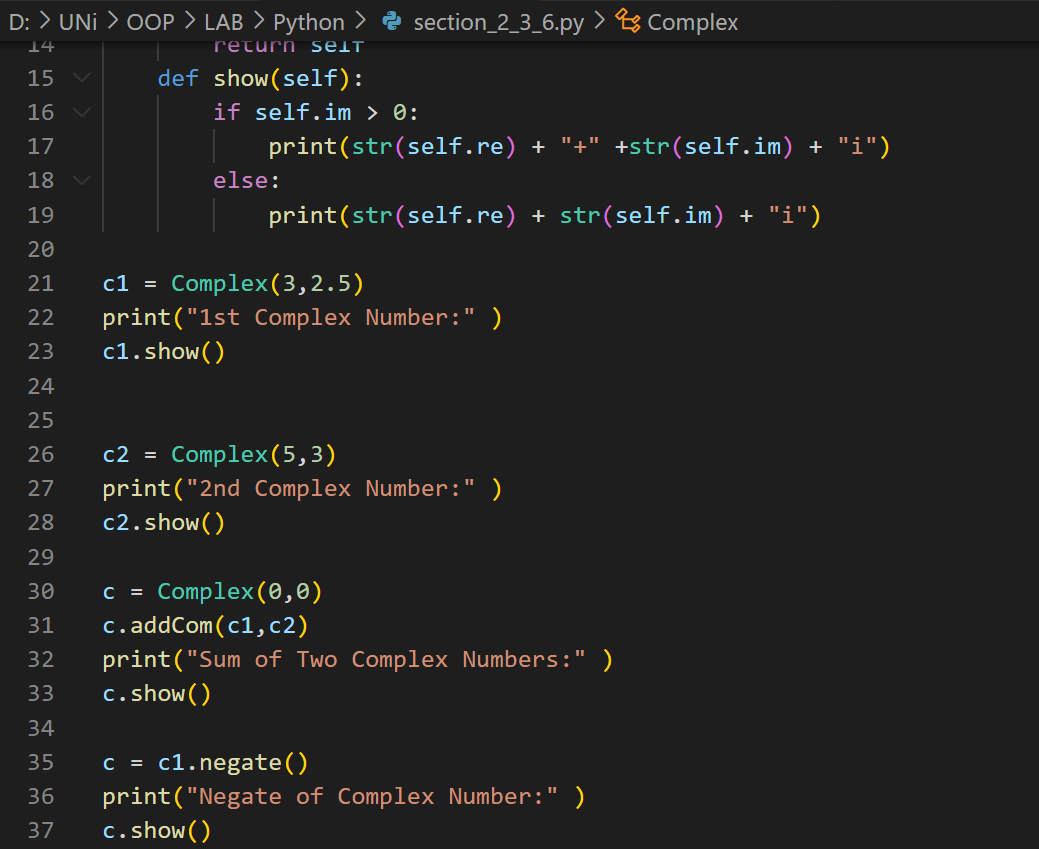
In this example code, we have created a class called Complex

* It has two constructors. The first one has no parameter and it initialize the data members to re and im to zero.
* The second constructor takes two arguments and set im and re to corresponding argument values.
* Then we have a set value function which set the value of re and im to the corresponding values of arguments.
* Lastly, we have show function which shows the current values of data members re and im.

**Example # 02(Section 2.3.6)**

**Code:**





**Output:**



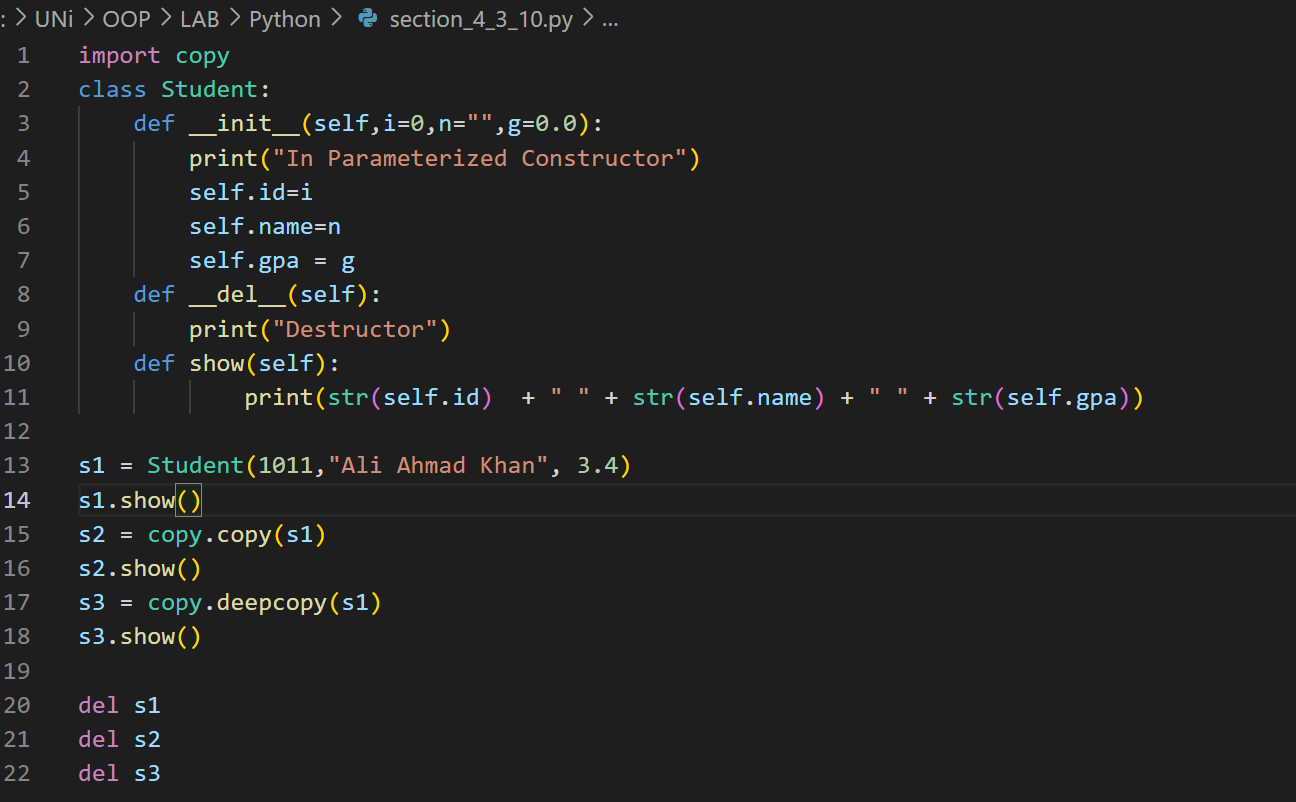
**OBSERVATIONS:**

In this example code, We defined a **Complex** class, having **2 data members** **re** and **im**, 2 constructor (**parameterless** and **parameterized** **constructors**) and **three** **functions** i.e addCom() , negate() and show().

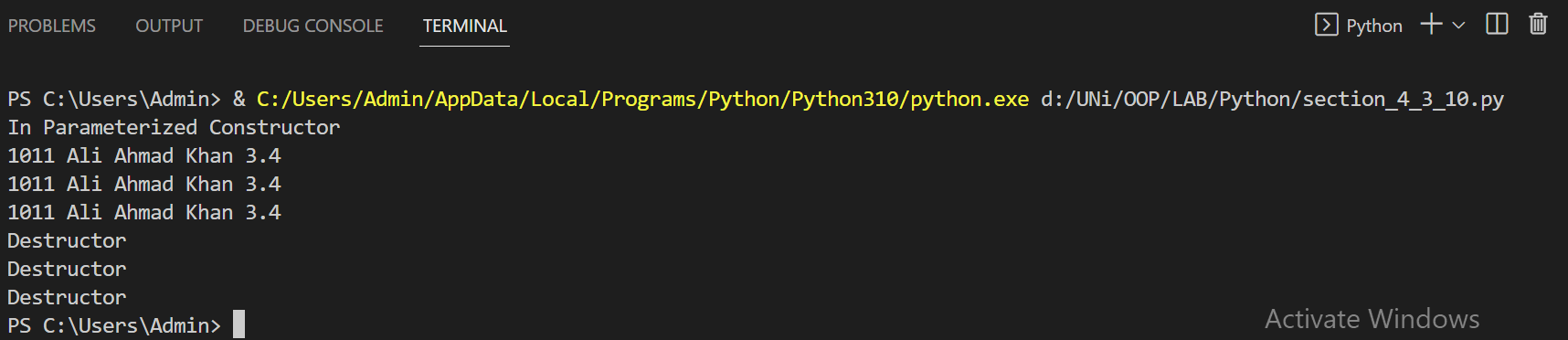
* The paratermeterless constructor **initialize** the data members **re** and **im** to **0**.
* The parameterized constructor **set** the **re** and **im** to the **values** **passed** to it.
* The addComp() function takes **two** parameters and add to the **complex** **re** and **im** data members.
* The negate() function has **no parameters** and **return** the **negate** of the **complex number** by **multplying** the **re** and **im** members with the **negative** **sign**.
* Then we have a **show** **function** which **display** the **real** and **imaginary** data members of Complex class. It **checks** whether Imaginary part is non-negative or not. If it is non-negative then we **concatenate** + sign with imaginary part. Otherwise we don’t concatenate + with imaginary part.
* Lastly we used the class and **declared** some **objects** of the Complex number and called **member functions.**

**Example # 03(Section 4.3.10)**

**Code:**

****

**Output:**



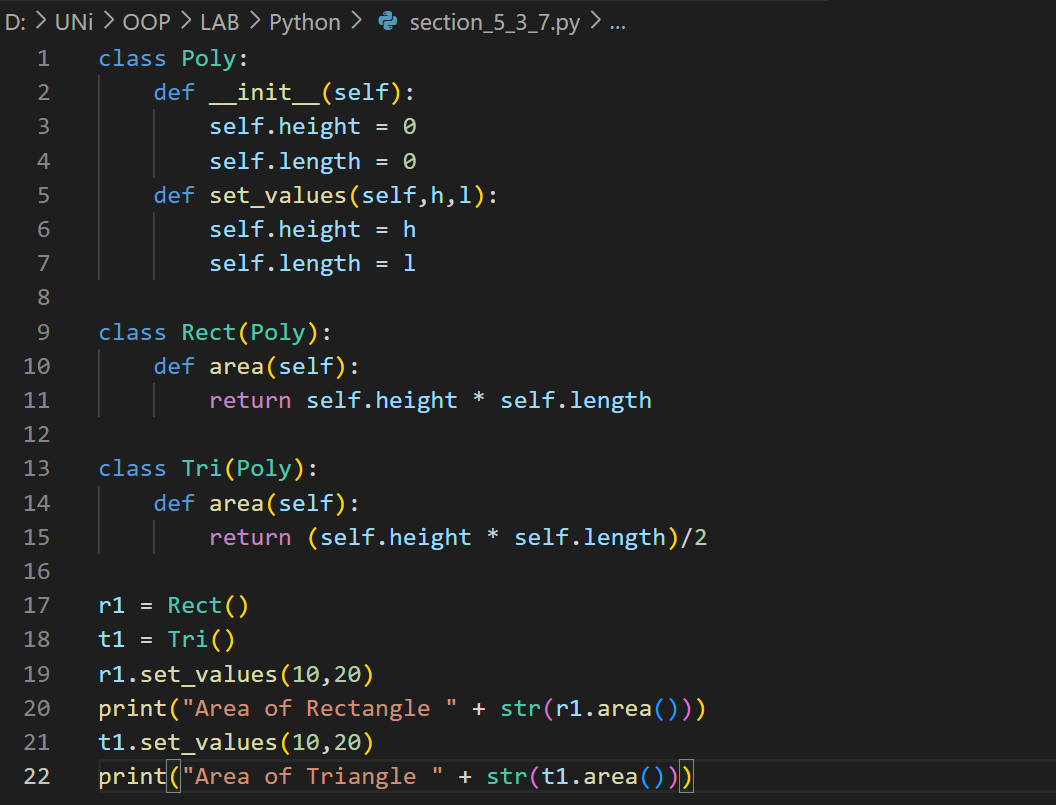
**OBSERVATIONS:**

In this example, we have created a class called **Student**.

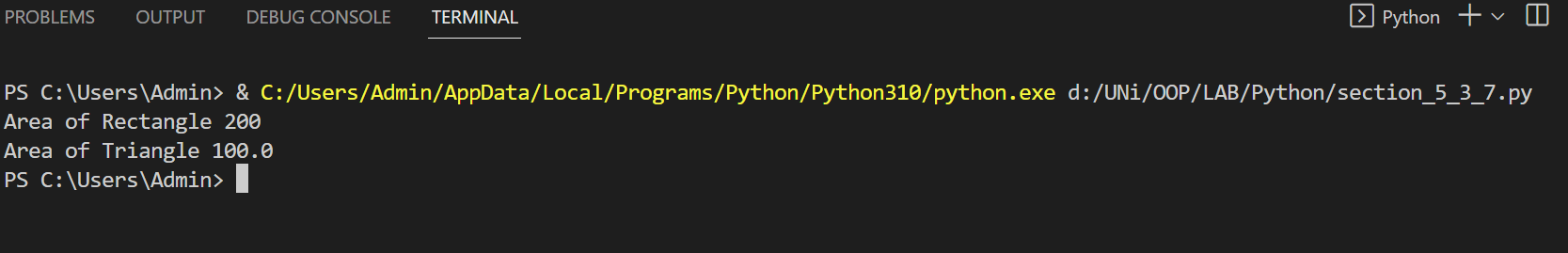
* It consists of a **parameterized** **constructor** which **initiliaze** the **data members** to values given as **arguments** when **declaring** the **object** of this class. There are three data members in this class,
  1. Id(type int)
  2. Name(type string)
  3. Gpa(float)
* Next we have a **destructor** which just prints out **Destructor**.
* Next we have a **show function** which **prints** **out** the **data members** of this class.
* We have **copy module** in this program to perform **deep** and **shallow** **copy**.
* To **demonstrate** **deep** and **shallow** **copy**, **three** **objects** **s1**, **s2**, and **s3** are created. **s1** is created using **parameterized** **constructor**, **s2** is created using **s1** by **shallow** **copy**, and **s3** is created using **s1** by **deep** **copy**. Output of each object is shown. In the end, **destructor** is used to remove the **objects**. In this example, **built-in copy.copy()** and **copy.deepcopy()** functions are just used. These can also be **tailor** **made** and adjusted according to one’s need. A **custom** **shallow** and **deep copy** in Python can also be **implemented**.

**Example # 04(Section 5.3.7)**

**Code:**



**Output:**



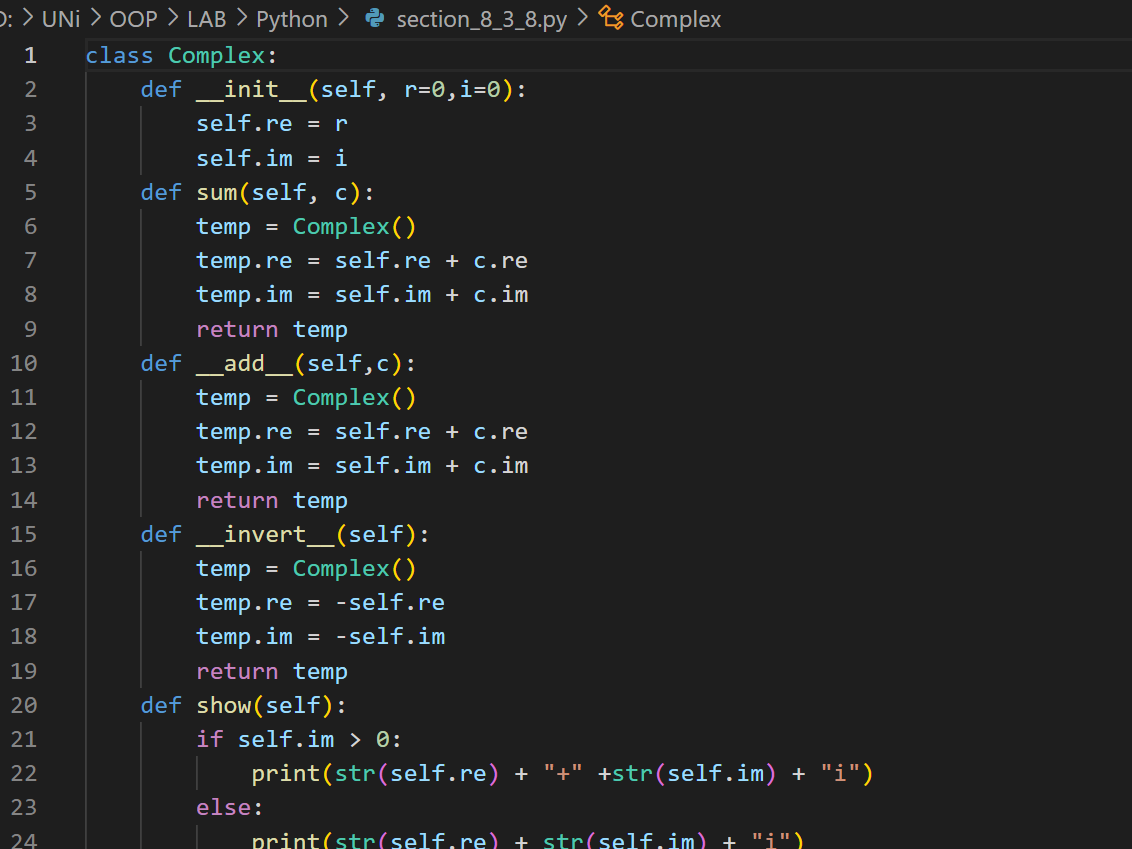
**OBSERVATIONS:**

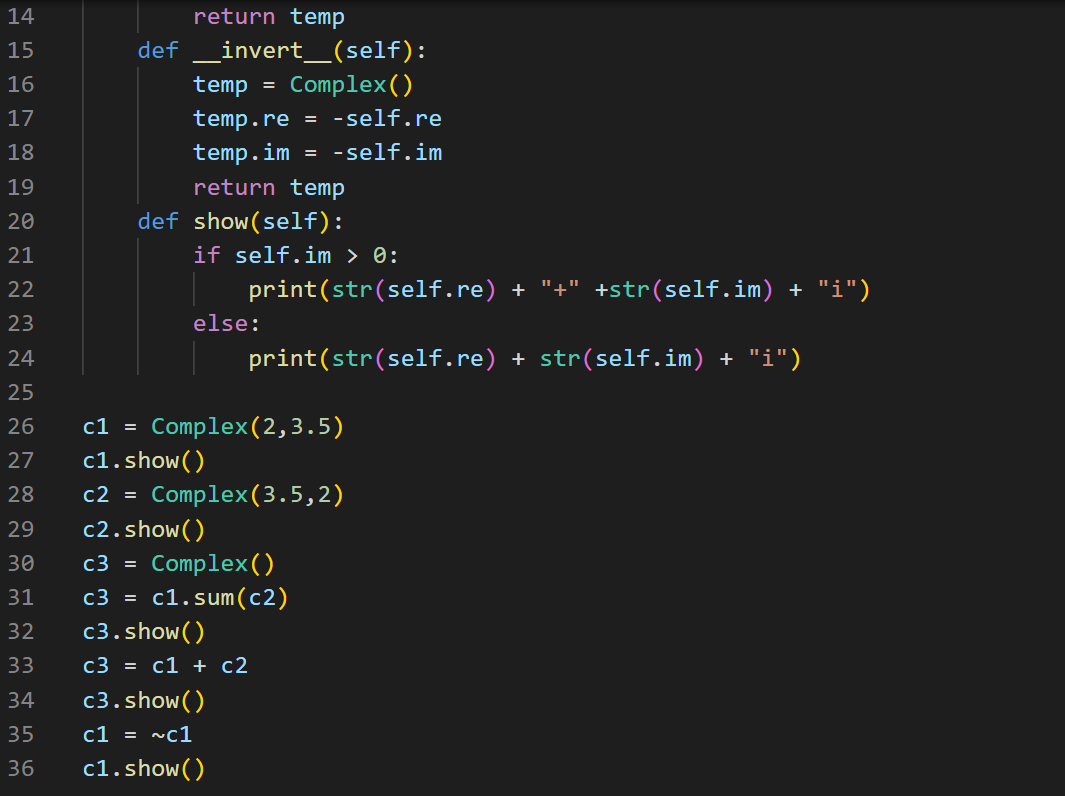
In this example, we have created a base class called **Poly** and then we derived two classes **Rect** and **Tri** from **Poly**(Hierarchical Inheritance).

* In **base** class **Poly**, we have a **constructor** which **initiliaze** the datamembers **height** and **length** to zero. Next we have **set\_values** function which **set** the **values** of **height** and **length** given by values to the **parameters** **h** and **l** respectively.
* In **Rect** class, we have only one area function for **calculating the area of Rect**. It is **return** type function and it **returns** the **value** given by **multiplying** length with height.
* In Tri class, we have the area function with the **same name** and **return type** but the definition of this **function** is **different** from that of the **Rect**. This function returns the **half** of **multiplication** of **length** and **height** which is the main definition **area of triangle**.

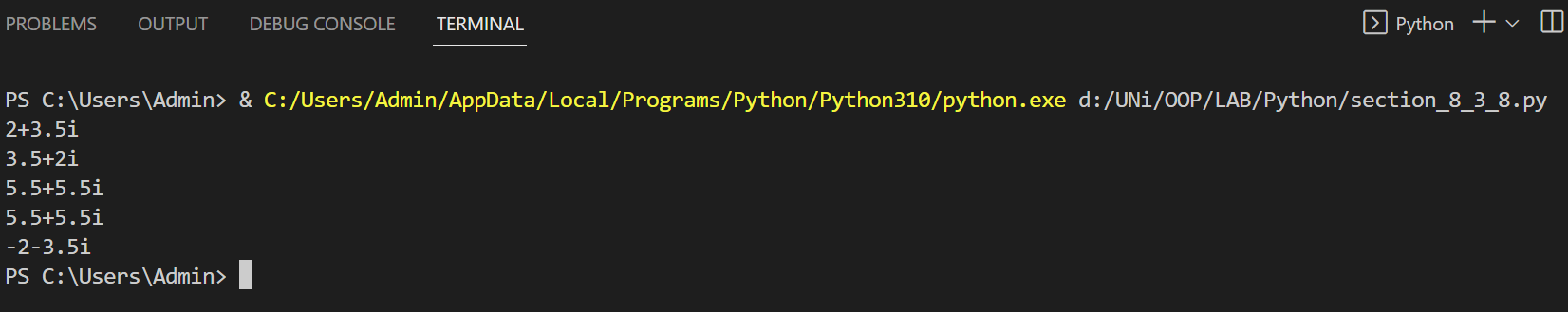
**Example # 05(Section 8.3.8)**

**Code:**





**Output:**



**OBSERVATIONS:**

In this example, we have created a class called **Complex**. This class has two data members **re** and **im**.

* In this class Complex, we have a **parameterized constructor** which initilize the two data members to values given by declaring its object. By **default 0** is passed to this constructor.
* Next we have a function called **sum.** Thisfunction takes an object of type **Complex** as anargument. Then it declares a variable called temp for storing the result of sum. After declaring temp, this function adds the corresponding data members of given argument and the data members of this class and then store them in temp. Finally, it returns the temp object.
* Next we have **overloaded** the **operator +** in the form of **\_\_add\_\_** function. This function overload the **binary operator +** for **adding** two Complex numbers.
* Next we have the **\_\_invert\_\_** function which **overload** the **unary – operator**. This function overload the **unary – operator** for **negating** the **real** and **imaginary part** of the Complex number.
* Lastly, we have a **show** function which display the **real** and **imaginary** **data** **members** of Complex class. It check whether **Imaginary part** is **non-negative** or not. If it is non-negative then we **concatenate** **+** **sign** with **imaginary part**. Otherwise we don’t **concatenate** **+** with **imaginary part**.