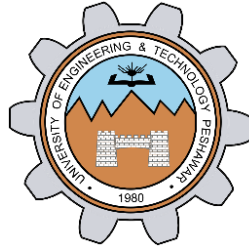


LAB # 03



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:

```
D: > UNi > OOP > LAB > Python > section_1_3_9.py > Complex > set_value
1  class Complex:
2      def __init__(self):
3          self.re=0
4          self.im=0
5      def __init__(self,r,i):
6          self.re=r
7          self.im=i
8      def set_value(self, rr, ii):
9          self.re=rr
10         self.im=ii
11     def show(self):
12         print("Complex Number:" + str(self.re) + "+" +str(self.im) + "i")
13
14 obj1 = Complex(1,2)
15 obj1.show()
16
17
18 obj2 = Complex(0,0)
19 obj2.set_value(2,4)
20 obj2.show()
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Python + - □
PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Programs/Python/Python310/python.exe d:/Uni/OOP/LAB/Python/section_1_3_9.py
Complex Number:1+2i
Complex Number:2+4i
PS C:\Users\Admin>
```

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:

```
D: > UNi > OOP > LAB > Python > section_2_3_6.py > Complex
1  class Complex:
2      def __init__(self):
3          self.re=0.0
4          self.im=0.0
5      def __init__(self,r,i):
6          self.re=r
7          self.im=i
8      def addCom(self,num1,num2):
9          self.re=num1.re + num2.re
10         self.im=num2.im + num2.im
11     def negate(self):
12         self.re=-self.re
13         self.im=-self.im
14         return self
15     def show(self):
16         if self.im > 0:
17             print(str(self.re) + "+" +str(self.im) + "i")
18         else:
19             print(str(self.re) + str(self.im) + "i")
20
21     c1 = Complex(3,2.5)
22     print("1st Complex Number:" )
23     c1.show()
24
```

```

D: > UNi > OOP > LAB > Python > section_2_3_6.py > Complex
14         return self
15     def show(self):
16         if self.im > 0:
17             print(str(self.re) + "+" +str(self.im) + "i")
18         else:
19             print(str(self.re) + str(self.im) + "i")
20
21     c1 = Complex(3,2.5)
22     print("1st Complex Number:" )
23     c1.show()
24
25
26     c2 = Complex(5,3)
27     print("2nd Complex Number:" )
28     c2.show()
29
30     c = Complex(0,0)
31     c.addCom(c1,c2)
32     print("Sum of Two Complex Numbers:" )
33     c.show()
34
35     c = c1.negate()
36     print("Negate of Complex Number:" )
37     c.show()

```

OUTPUT:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Python  +  -  [ ]  [ ]  ^

PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Programs/Python/Python310/python.exe d:/UNi/OOP/LAB/Python/section_2_3_6.py
1st Complex Number:
3+2.5i
2nd Complex Number:
5+3i
Sum of Two Complex Numbers:
8+6i
Negate of Complex Number:
-3-2.5i
PS C:\Users\Admin>

```

Activate Windows
Go to Settings to activate Windows.

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 parameterless 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 **multiplying** 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:

```
: > UNi > OOP > LAB > Python > section_4_3_10.py > ...
1  import copy
2  class Student:
3      def __init__(self,i=0,n="",g=0.0):
4          print("In Parameterized Constructor")
5          self.id=i
6          self.name=n
7          self.gpa = g
8      def __del__(self):
9          print("Destructor")
10     def show(self):
11         print(str(self.id) + " " + str(self.name) + " " + str(self.gpa))
12
13     s1 = Student(1011,"Ali Ahmad Khan", 3.4)
14     s1.show()
15     s2 = copy.copy(s1)
16     s2.show()
17     s3 = copy.deepcopy(s1)
18     s3.show()
19
20     del s1
21     del s2
22     del s3
```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Programs/Python/Python310/python.exe d:/UNI/OOP/LAB/Python/section_4_3_10.py
In Parameterized Constructor
1011 Ali Ahmad Khan 3.4
1011 Ali Ahmad Khan 3.4
1011 Ali Ahmad Khan 3.4
Destructor
Destructor
Destructor
PS C:\Users\Admin>
```

OBSERVATIONS:

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

- It consists of a **parameterized constructor** which **initilize** 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:

```
D: > UNi > OOP > LAB > Python > section_5_3_7.py > ...
1  class Poly:
2      def __init__(self):
3          self.height = 0
4          self.length = 0
5      def set_values(self,h,l):
6          self.height = h
7          self.length = l
8
9  class Rect(Poly):
10     def area(self):
11         return self.height * self.length
12
13     class Tri(Poly):
14         def area(self):
15             return (self.height * self.length)/2
16
17     r1 = Rect()
18     t1 = Tri()
19     r1.set_values(10,20)
20     print("Area of Rectangle " + str(r1.area()))
21     t1.set_values(10,20)
22     print("Area of Triangle " + str(t1.area()))
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Python + - []

PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Programs/Python/Python310/python.exe d:/Uni/OOP/LAB/Python/section_5_3_7.py
Area of Rectangle 200
Area of Triangle 100.0
PS C:\Users\Admin> |
```

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 data members **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:

```

D: > UNi > OOP > LAB > Python > section_8_3_8.py > Complex
1  class Complex:
2      def __init__(self, r=0,i=0):
3          self.re = r
4          self.im = i
5      def sum(self, c):
6          temp = Complex()
7          temp.re = self.re + c.re
8          temp.im = self.im + c.im
9          return temp
10     def __add__(self,c):
11         temp = Complex()
12         temp.re = self.re + c.re
13         temp.im = self.im + c.im
14         return temp
15     def __invert__(self):
16         temp = Complex()
17         temp.re = -self.re
18         temp.im = -self.im
19         return temp
20     def show(self):
21         if self.im > 0:
22             print(str(self.re) + "+" +str(self.im) + "i")
23         else:
24             print(str(self.re) + str(self.im) + "i")

```

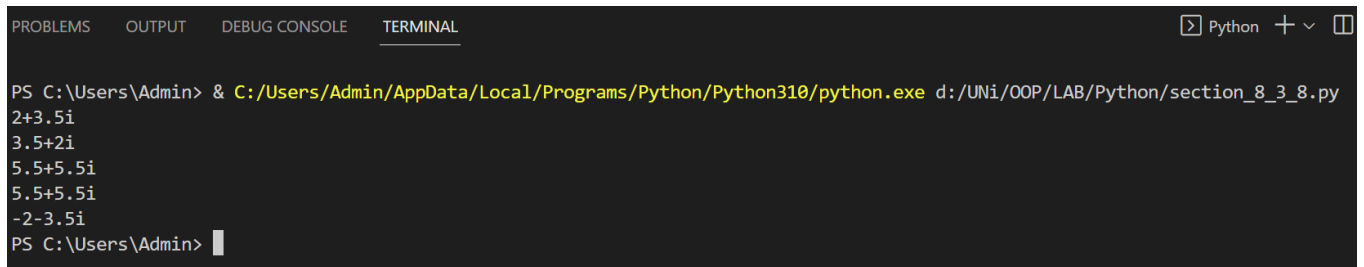


```

14         return temp
15     def __invert__(self):
16         temp = Complex()
17         temp.re = -self.re
18         temp.im = -self.im
19         return temp
20     def show(self):
21         if self.im > 0:
22             print(str(self.re) + "+" +str(self.im) + "i")
23         else:
24             print(str(self.re) + str(self.im) + "i")
25
26     c1 = Complex(2,3.5)
27     c1.show()
28     c2 = Complex(3.5,2)
29     c2.show()
30     c3 = Complex()
31     c3 = c1.sum(c2)
32     c3.show()
33     c3 = c1 + c2
34     c3.show()
35     c1 = ~c1
36     c1.show()

```

OUTPUT:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python + v
PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Programs/Python/Python310/python.exe d:/UNI/OOP/LAB/Python/section_8_3_8.py
2+3.5i
3.5+2i
5.5+5.5i
5.5+5.5i
-2-3.5i
PS C:\Users\Admin>

```

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 initialize 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**. This function takes an object of type **Complex** as an argument. 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**.