Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

Aim: Practical based on OOP concept using Python

IDE:

Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles classes, objects, inheritance, encapsulation, polymorphism, and abstraction programmers can leverage the full potential of Python's OOP capabilities to design elegant and efficient solutions to complex problems.

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

OOPs Concepts in Python

- Class in Python
- Objects in Python
- Polymorphism in Python
- Encapsulation in Python
- Inheritance in Python
- Data Abstraction in Python

Python Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

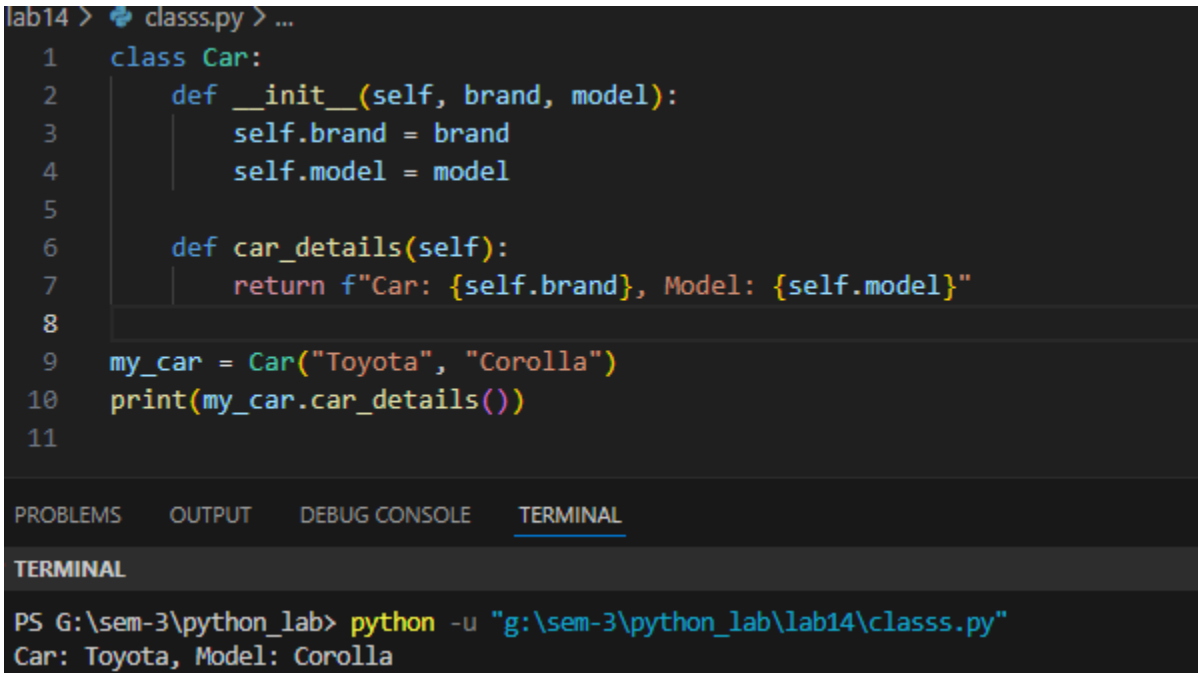
Defining a Class

Example 1:

```
class Car:
    # Constructor to initialize the object
    def __init__(self, brand, model):
        self.brand = brand  # Attribute
        self.model = model  # Attribute

    # Method to describe the car
    def car_details(self):
        return f"Car: {self.brand}, Model: {self.model}"

# Creating an object of the Car class
my_car = Car("Toyota", "Corolla")
print(my_car.car_details())
```

Output:

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**



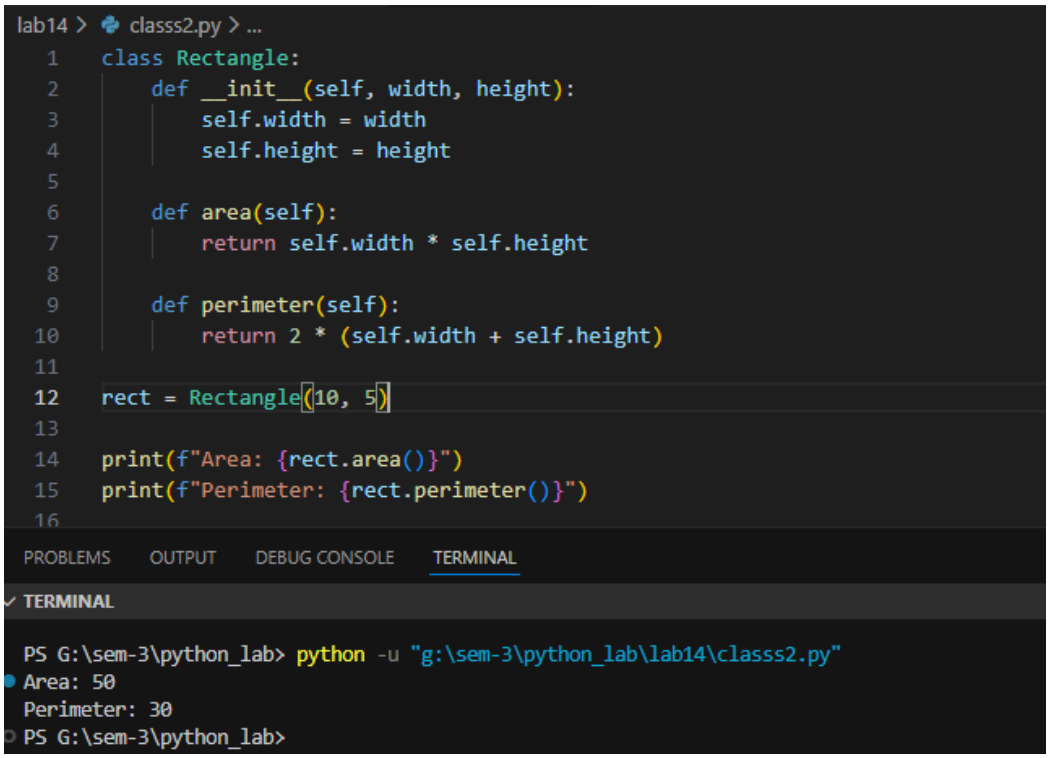
Example 2:

Class with Methods and Attributes

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    # Method to calculate area
    def area(self):
        return self.width * self.height

    # Method to calculate perimeter
    def perimeter(self):
        return 2 * (self.width + self.height)

# Create an object
rect = Rectangle(10, 5)
```

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

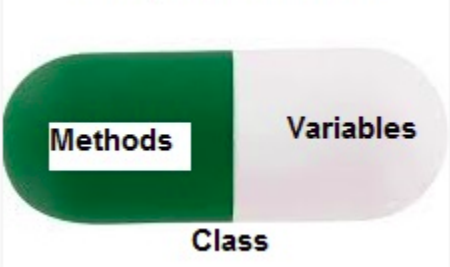
Accessing methods
print(f"Area: {rect.area()}") # Output: Area: 50
print(f"Perimeter: {rect.perimeter()}") # Output: Perimeter: 30
Output:



Encapsulation

In Python object-oriented programming, Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

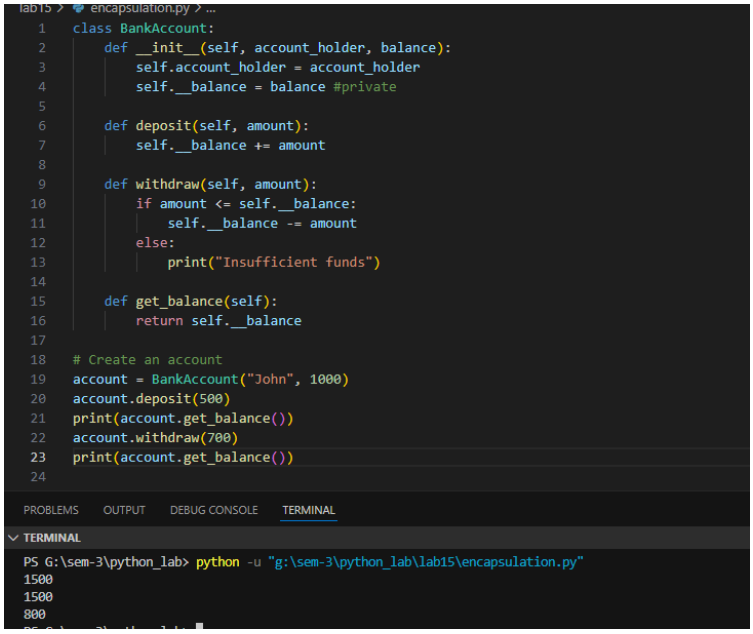


Example 3:

```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.__balance

# Create an account
account = BankAccount("John", 1000)
account.deposit(500)
print(account.get_balance())  #
account.withdraw(700)
print(account.get_balance())  #
```

Output

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**



```python
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.__balance = balance #private

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.__balance

# Create an account
account = BankAccount("John", 1000)
account.deposit(500)
print(account.get_balance())
account.withdraw(700)
print(account.get_balance())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
TERMINAL
PS G:\sem-3\python_lab> python -u "g:\sem-3\python_lab\lab15\encapsulation.py"
1500
1500
800
```

Inheritance

Inheritance allows a new class (child class) to inherit attributes and methods from an existing class (parent class). It promotes code reusability.

Example 4

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return "I am an animal."

# Dog class inherits from Animal class
class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

# Cat class inherits from Animal class
```

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

```
class Cat(Animal):
   def speak(self):
      return f"{self.name} says Meow!"

dog = Dog("Buddy")
cat = Cat("Whiskers")
print(dog.speak())  #
print(cat.speak())  #
```

Output

```
lab15 > inheritance.py > ...
1   class Animal:
2       def __init__(self, name):
3           self.name = name
4
5       def speak(self):
6           return "I am an animal."
7
8   # Dog class inherits from Animal class
9   class Dog(Animal):
10      def speak(self):
11          return f"{self.name} says Woof!"
12
13  # Cat class inherits from Animal class
14  class Cat(Animal):
15      def speak(self):
16          return f"{self.name} says Meow!"
17
18  dog = Dog("Buddy")
19  cat = Cat("Whiskers")
20  print(dog.speak())
21  print(cat.speak())
22

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
TERMINAL
PS G:\sem-3\python_lab> python -u "g:\sem-3\python_lab\lab15\inheritance.py"
Buddy says Woof!
Whiskers says Meow!
```

Polymorphism

Polymorphism is another important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

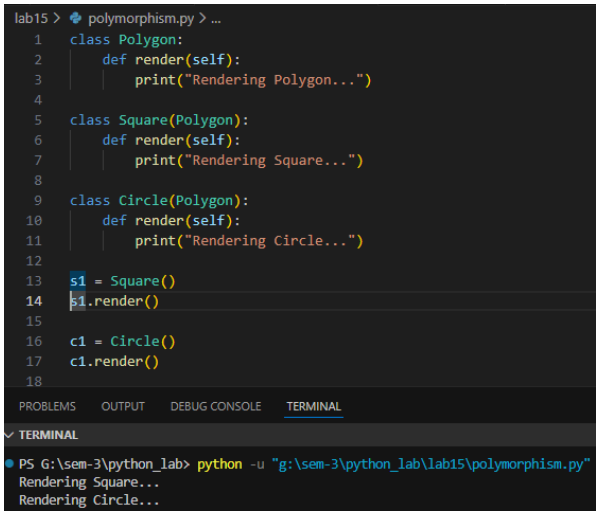
Example 5:

```
class Polygon:
```

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

```
    # method to render a shape
    def render(self):
        print("Rendering Polygon...")

class Square(Polygon):
    # renders Square
    def render(self):
        print("Rendering Square...")

class Circle(Polygon):
    # renders circle
    def render(self):
        print("Rendering Circle...")

# create an object of Square
s1 = Square()
s1.render()

# create an object of Circle
c1 = Circle()
c1.render()
```

Output:



Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

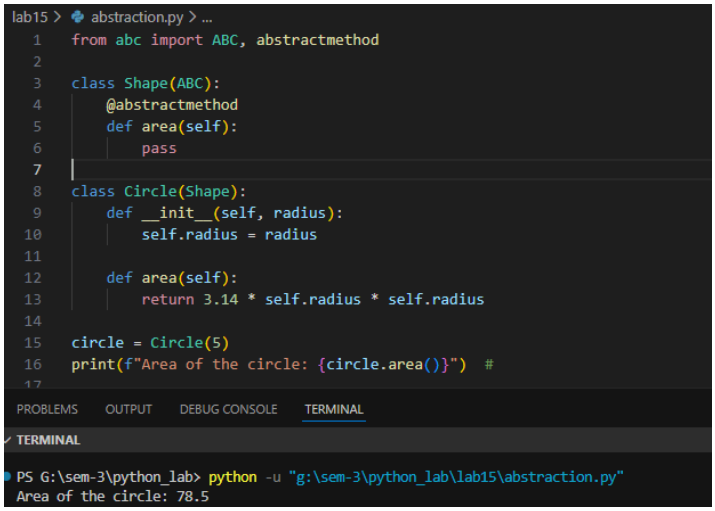
Abstraction

Abstraction focuses on hiding the internal implementation details of a class and exposing only the essential features.

Example 6:

```
from abc import ABC, abstractmethod

# Abstract class
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

circle = Circle(5)
print(f"Area of the circle: {circle.area()}")  #
```

Output:



Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

Post Lab Exercise:

- Write a Python program to create a class representing a Circle. Include methods to calculate its area and perimeter.
- Create a class `Book` that stores details like the title, author, and price of a book. Add methods to display the details of the book and apply a discount to the price. (a) Create two objects for different books and display their details. (b) Apply a 10% discount to one of the books and display the updated price.

```
lab15 > postLab.py > ...
1   import math
2
3   class Circle:
4       def __init__(self, radius):
5           self.radius = radius
6       def area(self):
7           return math.pi * self.radius * self.radius
8       def perimeter(self):
9           return 2 * math.pi * self.radius
10  c = Circle(5)
11  print("Radius:", c.radius)
12  print("Area:", c.area())
13  print("Perimeter:", c.perimeter())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
TERMINAL
PS G:\sem-3\python_lab> python -u "g:\sem-3\python_lab\lab15\PostLab.py"
Radius: 5
Area: 78.53981633974483
Perimeter: 31.41592653589793
```

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on OOP concept using Python	
Experiment No: 14	**Date:**	**Enrollment No: 92400133037**

lab15 > postLab.py > ...

```python
#b
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def display(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Price: {self.price}")

    def apply_discount(self, percent):
        discount_amount = (percent / 100) * self.price
        self.price -= discount_amount

book1 = Book("Python Programming", "Nand Davda", 500)
book2 = Book("Data Structures", "Meet Mehta", 650)

print("Before Discount:")
book1.display()
book2.display()
print("After 10% Discount on book1:")
book1.apply_discount(10)
book1.display()
```

```
Before Discount:
Title: Python Programming
Author: Nand Davda
Price: 500
Title: Data Structures
Title: Python Programming
Author: Nand Davda
Price: 500
Title: Data Structures
Title: Data Structures
Author: Meet Mehta
Price: 650
After 10% Discount on book1:
Title: Python Programming
Title: Python Programming
Author: Nand Davda
Price: 450.0
```

GITHUB LINK:
https://github.com/Heer972005/Python_Lab