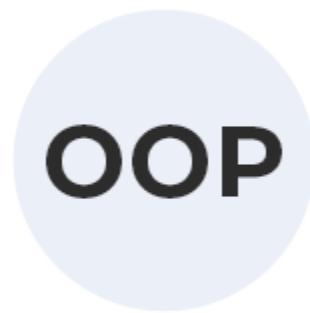


 Marwadi University Marwadi Chandarana Group	NAAC  A+	
Subject: Programming With Python (01CT1309)	Aim: Practical based on OOP concept using Python	
Experiment No: 14	Date: 20/10/2025	Enrollment No: 92510133049

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.





Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

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:



Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  class Car:
2      # Constructor to initialize the object
3      def __init__(self, brand, model):
4          self.brand = brand # Attribute
5          self.model = model # Attribute
6
7      # Method to describe the car
8      def car_details(self):
9          return f"Car: {self.brand}, Model: {self.model}"
10
11 # Creating an object of the Car class
12 my_car = Car("Mahindra", "Thar")
13 print(my_car.car_details())
...
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
● Car: Mahindra, Model: Thar
PS C:\Users\PRAVEEN KUMAR>
```

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)
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

Accessing methods

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

Output:

```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  class Car:
2      # Constructor to initialize the object
3      def __init__(self, brand, model):
4          self.brand = brand # Attribute
5          self.model = model # Attribute
6
7      # Method to describe the car
8      def car_details(self):
9          return f"Car: {self.brand}, Model: {self.model}"
10 # Creating an object of the Car class
11 my_car = Car("Mahindra", "Thar")
12 print(my_car.car_details())
13 class Rectangle:
14     def __init__(self, width, height):
15         self.width = width
16         self.height = height
17     # Method to calculate area
18     def area(self):
19         return self.width * self.height
20     # Method to calculate perimeter
21     def perimeter(self):
22         return 2 * (self.width + self.height)
23 # Create an object
24 rect = Rectangle(10, 10)
25 # Accessing methods
26 print(f"Area: {rect.area()}") # Output: Area: 100
27 print(f"Perimeter: {rect.perimeter()}") # Output: Perimeter: 40
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
Car: Mahindra, Model: Thar
Area: 100
Perimeter: 40
PS C:\Users\PRAVEEN KUMAR>

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.



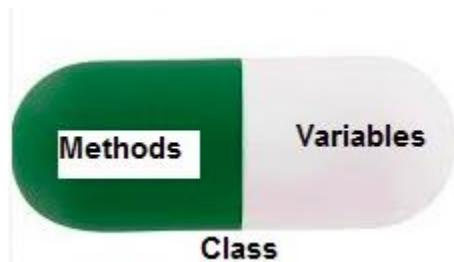
Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049



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 <small>Marwadi Chandarana Group</small>	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: 20/10/2025	Enrollment No: 92510133049

```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  class BankAccount:
2      def __init__(self, account_holder, balance):
3          self.account_holder = account_holder
4          self.__balance = balance # Private attribute
5
6      def deposit(self, amount):
7          self.__balance += amount
8
9      def withdraw(self, amount):
10         if amount <= self.__balance:
11             self.__balance -= amount
12         else:
13             print("Insufficient funds")
14
15     def get_balance(self):
16         return self.__balance
17
18 # Create an account
19 account = BankAccount("Ramani", 2000)
20 account.deposit(1000)
21 print(account.get_balance())
22 account.withdraw(500)
23 print(account.get_balance())
...
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    Python + ▾    ⌂    ⚡    ...    [ ]    X
PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
3000
2500
PS C:\Users\PRAVEEN KUMAR>
```

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."
```



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: 20/10/2025

Enrollment No: 92510133049

```
# Dog class inherits from Animal class
```

```
class Dog(Animal):
```

```
    def speak(self):
```

```
        return f"{self.name} says Woof!"
```

```
# Cat class inherits from Animal class
```

```
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 :
```



Marwadi
University
Marwadi Chandarana Group



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: 20/10/2025	Enrollment No: 92510133049

```
C: > Users > PRAVEEN KUMAR > exp - 14.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 Boow!"
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("rakesh")
19 cat = Cat("dileep")
20 print(dog.speak())
21 print(cat.speak())
22

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
rakesh says Boow!
dileep says Meow!
PS C:\Users\PRAVEEN KUMAR>
```

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:
```

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



Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

```
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



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: 20/10/2025	Enrollment No: 92510133049

```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  class Polygon:
2      # method to render a shape
3      def render(self):
4          print("Rendering Polygon...")
5
6  class Square(Polygon):
7      # renders Square
8      def render(self):
9          print("Rendering Square...")
10
11 class Circle(Polygon):
12     # renders circle
13     def render(self):
14         print("Rendering Circle... ")
15
16 # create an object of Square
17 s1 = Square()
18 s1.render()
19
20 # create an object of Circle
21 c1 = Circle()
22 c1.render()
23
24 # create an object of polygon
25 d1 = Polygon()
26 d1.render()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + ⌂ ⏺ ... | ☰ ×

```
PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN  
KUMAR/exp - 14.py"  
● Rendering Square...  
Rendering Circle...  
Rendering Polygon...  
○ PS C:\Users\PRAVEEN KUMAR>
```

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
```



Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

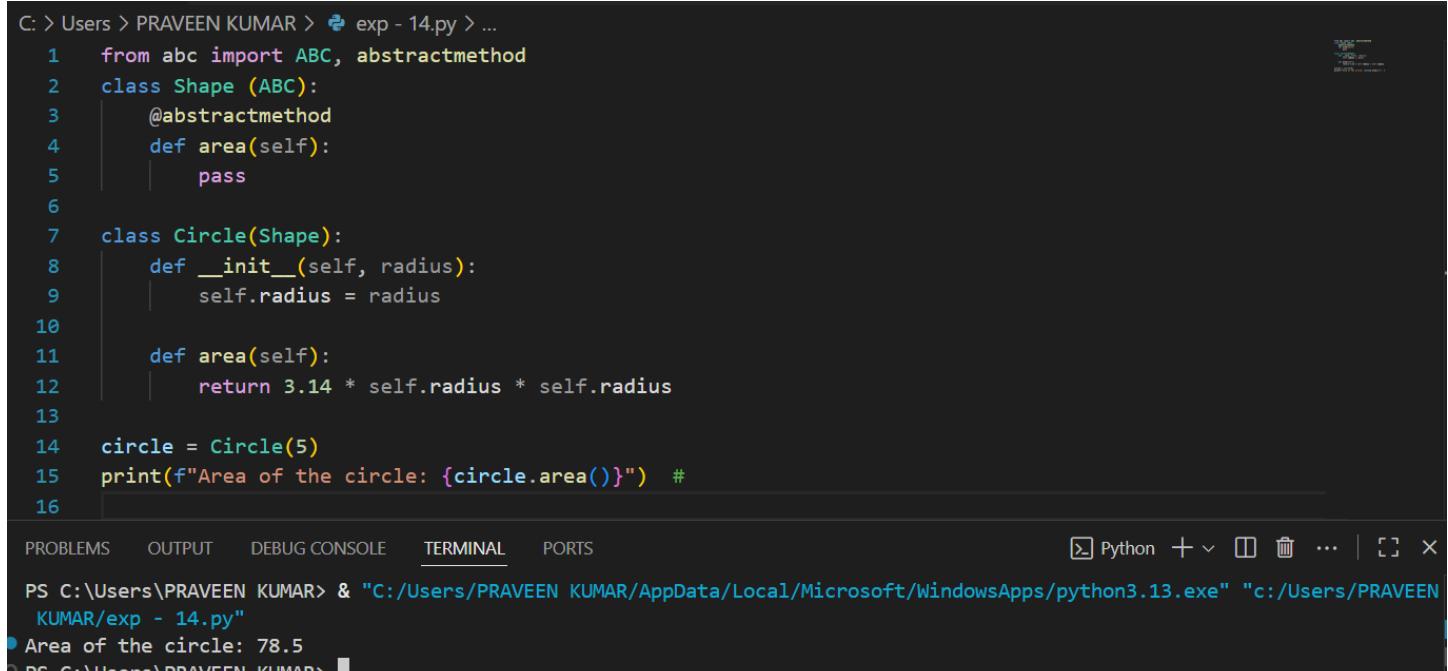
Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

```
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:



```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  from abc import ABC, abstractmethod
2  class Shape (ABC):
3      @abstractmethod
4          def area(self):
5              pass
6
7  class Circle(Shape):
8      def __init__(self, radius):
9          self.radius = radius
10
11     def area(self):
12         return 3.14 * self.radius * self.radius
13
14 circle = Circle(5)
15 print(f"Area of the circle: {circle.area()}") #
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
• Area of the circle: 78.5
D PS C:\Users\PRAVEEN KUMAR>

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.

 Marwadi University <small>Marwadi Chandarana Group</small>	 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: 20/10/2025	Enrollment No: 92510133049

ANSWERS:

Write a Python program to create a class representing a Circle. Include methods to calculate its area and perimeter.



Subject: Programming With Python (01CT1309)

Aim: Practical based on OOP concept using Python

Experiment No: 14

Date: 20/10/2025

Enrollment No: 92510133049

- 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.

```
C: > Users > PRAVEEN KUMAR > exp - 14.py > ...
1  class Book:
2      def __init__(self, title, author, price):
3          self.title = title
4          self.author = author
5          self.price = price
6
7      def display_details(self):
8          """Display the book's details."""
9          print(f"Title: {self.title}")
10         print(f"Author: {self.author}")
11         print(f"Price: ₹{self.price:.2f}")
12
13     def apply_discount(self, discount_percent):
14         """Apply a discount to the book's price."""
15         discount_amount = self.price * (discount_percent / 100)
16         self.price -= discount_amount
17         print(f"Discount of {discount_percent}% applied. New price: ₹{self.price:.2f}")
18
19 # (a) Create two book objects and display their details
20 book1 = Book("The Alchemist", "Paulo Coelho", 499.00)
21 book2 = Book("Sapiens", "Yuval Noah Harari", 799.00)
22
23 print("Book 1 Details:")
24 book1.display_details()
25 print("\nBook 2 Details:")
26 book2.display_details()
27
28 # (b) Apply a 10% discount to book2 and display updated price
29 print("\nApplying 10% discount to Book 2:")
30 book2.apply_discount(10)
```

```
PS C:\Users\PRAVEEN KUMAR> & "C:/Users/PRAVEEN KUMAR/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/PRAVEEN KUMAR/exp - 14.py"
Title: The Alchemist
Author: Paulo Coelho
Price: ₹499.00

Book 2 Details:
Title: Sapiens
Author: Yuval Noah Harari
Price: ₹799.00

Applying 10% discount to Book 2:
Discount of 10% applied. New price: ₹719.10
PS C:\Users\PRAVEEN KUMAR>
```