	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:92510133011**

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.

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

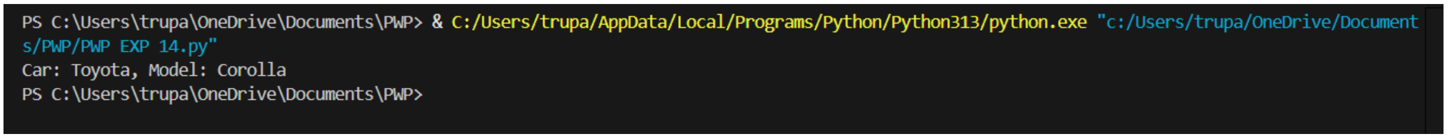
A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

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:



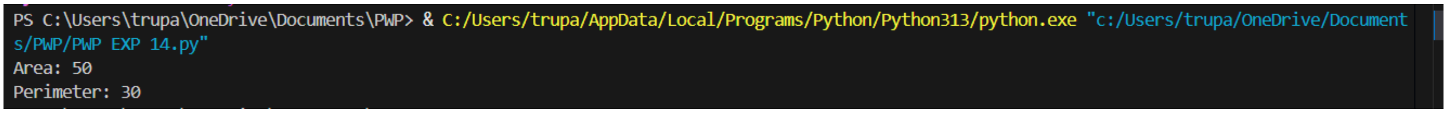
```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Car: Toyota, Model: Corolla
PS C:\Users\trupa\OneDrive\Documents\PWP>
```

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)

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

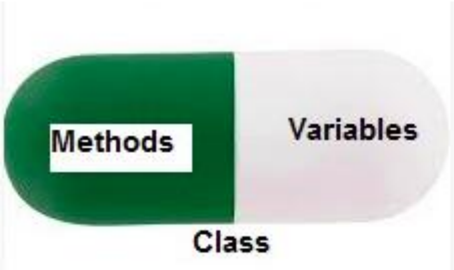
Output:



```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Area: 50
Perimeter: 30
```

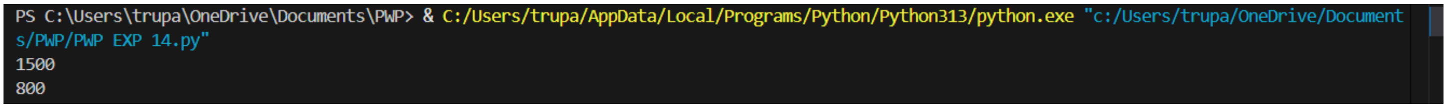
Encapsulation

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.



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 :
```



```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Documents/PWP/PWP EXP 14.py"
1500
800
```

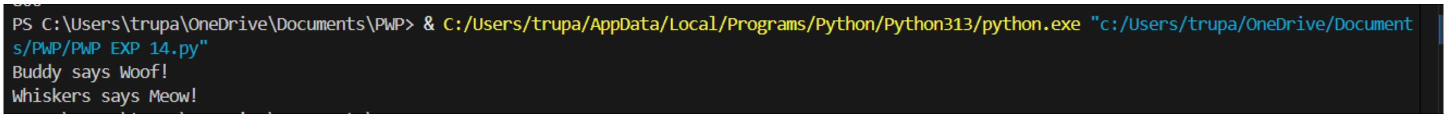
Inheritance

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
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:



```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Buddy says Woof!
Whiskers says Meow!
```

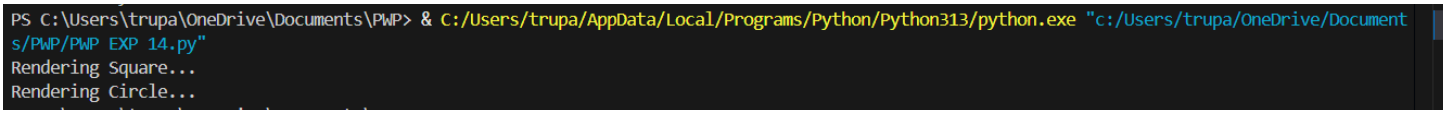
Polymorphism

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:
   # 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:



```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Rendering Square...
Rendering Circle...
```

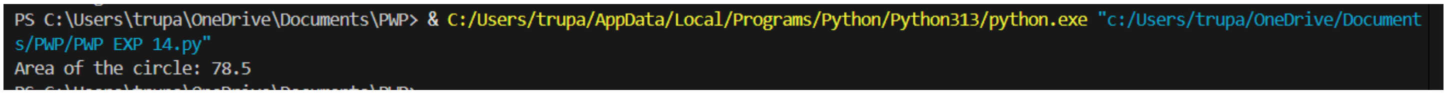
Abstraction

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:



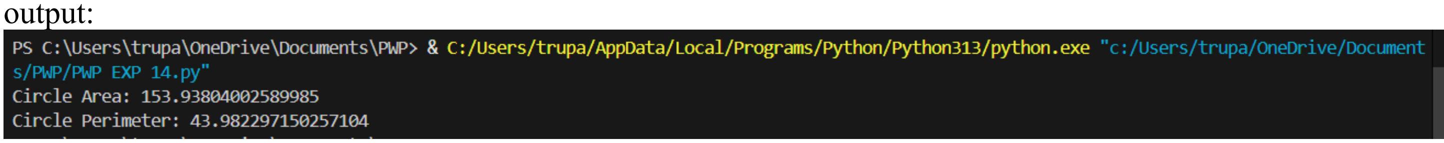
```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Area of the circle: 78.5
PS C:\Users\trupa\OneDrive\Documents\PWP>
```

Post Lab Exercise:

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

```
import math

class Circle:
  def __init__(self, radius):
    self.radius = radius
    # Method to calculate area
    def area(self):
        return math.pi * (self.radius ** 2)

    # Method to calculate perimeter (circumference)
    def perimeter(self):
        return 2 * math.pi * self.radius


# Create object
c = Circle(7)
print("Circle Area:", c.area())
print("Circle Perimeter:", c.perimeter())
```

output:



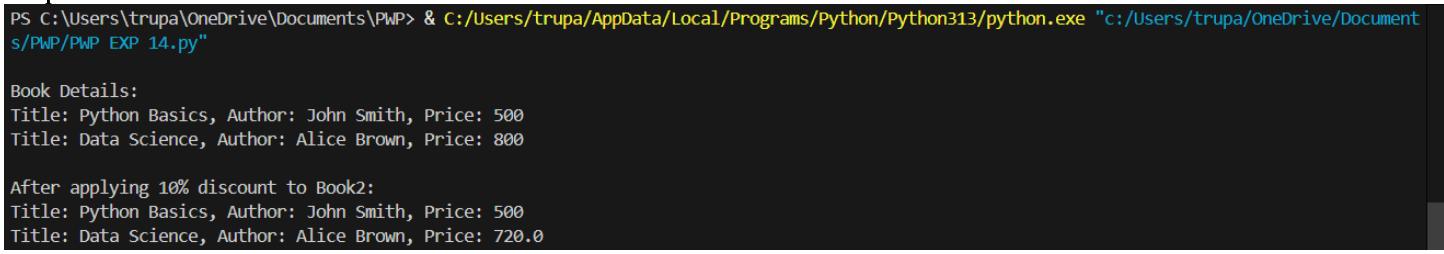
```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"
Circle Area: 153.93804002589985
Circle Perimeter: 43.982297150257104
```

- 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.

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

    # Method to display details
    def display(self):
        print(f"Title: {self.title}, Author: {self.author}, Price: {self.price}")

    # Method to apply discount
    def apply_discount(self, discount_percent):
        self.price = self.price - (self.price * discount_percent / 100)


# Create two book objects
book1 = Book("Python Basics", "John Smith", 500)
book2 = Book("Data Science", "Alice Brown", 800)

# Display details of both books
```

```
print("\nBook Details:")
book1.display()
book2.display()

# Apply 10% discount to book2
book2.apply_discount(10)

print("\nAfter applying 10% discount to Book2:")
book1.display()
book2.display()
```

output:



```
PS C:\Users\trupa\OneDrive\Documents\PWP> & C:/Users/trupa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/trupa/OneDrive/Document
s/PWP/PWP EXP 14.py"

Book Details:
Title: Python Basics, Author: John Smith, Price: 500
Title: Data Science, Author: Alice Brown, Price: 800

After applying 10% discount to Book2:
Title: Python Basics, Author: John Smith, Price: 500
Title: Data Science, Author: Alice Brown, Price: 720.0
```