

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

**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.



 <b>Marwadi University</b> Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

## 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())
```

Code:

```

1  # Define the Car class
2  class Car:
3      # Constructor to initialize the object
4      def __init__(self, brand, model):
5          self.brand = brand # Attribute
6          self.model = model # Attribute
7
8      # Method to describe the car
9      def car_details(self):
10         return f"Car: {self.brand}, Model: {self.model}"
11
12 # Creating an object of the Car class
13 my_car = Car("Toyota", "Corolla")
14
15 # Printing car details
16 print(my_car.car_details())

```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

Output:

```
In [1]: runfile('C:/Users/student/.spyder-py3/temp.py', wdir='C:/Users/
student/.spyder-py3')
Car: Toyota, Model: Corolla

In [2]:
```

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
```

Code:

```
1 class Rectangle:
2     def __init__(self, width, height):
3         self.width = width
4         self.height = height
5
6     # Method to calculate area
7     def area(self):
8         return self.width * self.height
9
10    # Method to calculate perimeter
11    def perimeter(self):
12        return 2 * (self.width + self.height)
13
14    # Create an object
15    rect = Rectangle(10, 5)
16
17    # Accessing methods
18    print(f"Area: {rect.area()}") # Output: Area: 50
19    print(f"Perimeter: {rect.perimeter()}") # Output: Perimeter: 30
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

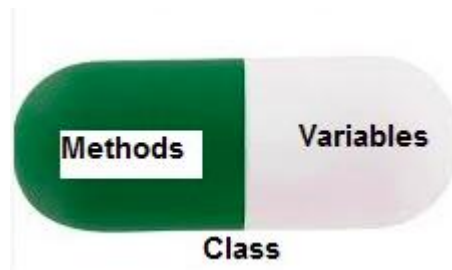
Output:

```
In [2]: runfile('C:/Users/student/.spyder-py3/temp.py', wdir='C:/Users/
student/.spyder-py3')
Area: 50
Perimeter: 30

In [3]:
```

### 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")
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

```
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()) #
```

Code:

```
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()) # Should print 1500
account.withdraw(700)
print(account.get_balance()) # Should print 800
```

Output

```
In [5]: runfile('C:/Users/student/.spyder-py3/untitled1.py', wdir='C:/
Users/student/.spyder-py3')
1500
800

In [6]:
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

## 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()) #
```

Code:



```
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()) #
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

Output

```
In [6]: runfile('C:/Users/student/.spyder-py3/untitled2.py', wdir='C:/Users/student/.spyder-py3')
Buddy says Woof!
Whiskers says Meow!

In [7]:
```

## 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()
```



 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

Code:

```
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:

```
In [7]: runfile('C:/Users/student/.spyder-py3/untitled3.py', wdir='C:/Users/student/.spyder-py3')
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
```



**Subject: Programming With Python (01CT1309)**

**Aim:** Practical based on OOP concept using Python

**Experiment No: 14**

**Date:**

**Enrollment No:92400133065**

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



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

Code:

```
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:

```
In [8]: runfile('C:/Users/student/.spyder-py3/untitled4.py', wdir='C:/  
Users/student/.spyder-py3')  
Area of the circle: 78.5  
  
In [9]:
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

### Post Lab Exercise:

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

Code:

```
import math

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



    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

# Example usage
circle = Circle(5)
print("Area:", circle.area())
print("Perimeter:", circle.perimeter())
```

Output:

```
In [9]: runfile('C:/Users/student/.spyder-py3/untitled5.py', wdir='C:/
Users/student/.spyder-py3')
Area: 78.53981633974483
Perimeter: 31.41592653589793
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Practical based on OOP concept using Python	
<b>Experiment No: 14</b>	<b>Date:</b>	<b>Enrollment No:92400133065</b>

- 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.

Code:

```

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(self):
8          return f"Title: {self.title}, Author: {self.author}, Price: ${self.price:.2f}"
9
10     def apply_discount(self, percent):
11         self.price *= (1 - percent / 100)
12
13     # Create two book objects and display details
14     book1 = Book("Book One", "Author A", 250)
15     book2 = Book("Book Two", "Author B", 300)
16
17     print(book1.display())
18     print(book2.display())
19
20     # Apply 10% discount to book1 and display updated price
21     book1.apply_discount(10)
22     print(book1.display())
23
24

```

Output:

```

In [10]: runfile('C:/Users/student/.spyder-py3/untitled6.py', wdir='C:/Users/student/.spyder-py3')
Title: Book One, Author: Author A, Price: $250.00
Title: Book Two, Author: Author B, Price: $300.00
Title: Book One, Author: Author A, Price: $225.00

In [11]:

```

Github:

<https://github.com/vahchalya-bodas/pwp.git>