Vaishnavi Ashok Patil
C07
SPP-II

Experiment No :10

Title: Implement encapsulation, inheritance (single and multiple), and polymorphism (method overloading and overriding).

Encapsulation

1. Create a class BankAccount where account_holder is public, _account_type is protected, and __balance is private. Write methods to deposit and withdraw money with validation.

```
class BankAccount:
    def __init__(self, account_holder, account_type, initial_balance=0.0):
        self.account_holder = account_holder
        self._account_type = account_type
        self.__balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            return f"Deposit successful. New balance: {self.__balance}"
        else:
            return "Deposit amount must be positive."
    def withdraw(self, amount):
        if amount <= 0:
            return "Withdrawal amount must be positive."
        elif amount > self.__balance:
            return "Insufficient funds for withdrawal."
        else:
            self.__balance -= amount
            return f"Withdrawal successful. New balance: {self.__balance}"
    def get_balance(self):
        return self.__balance
if __name__ == '__main__':
    account = BankAccount("Armaan", "Savings", 500.00)
    print(f"Account Holder: {account.account_holder}")
    print(f"Account Type (Protected): {account._account_type}")
    print(f"Initial Balance: {account.get_balance()}")
    print(account.deposit(200.00))
    print(account.deposit(-50.00))
    print(account.withdraw(100.00))
    print(account.withdraw(800.00))
    try:
        print(f"Attempting to access private balance directly: {account.__balance}")
    except AttributeError as e:
        print(f"Accessing private attribute failed: {e}")
```

```
Account Holder: Armaan
Account Type (Protected): Savings
Initial Balance: 500.0
Deposit successful. New balance: 700.0
Deposit amount must be positive.
Withdrawal successful. New balance: 600.0
Insufficient funds for withdrawal.
Accessing private attribute failed: 'BankAccount' object has no attribute '__balance'
```

2. Design a class Student where name is public, _roll_no is protected, and __marks is private. Ensure marks can only be set between 0 and 100.

```
class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self._roll_no = roll_no
```

```
        self.__marks = None
        self.set_marks(marks)

    def set_marks(self, marks):
        if 0 <= marks <= 100:
            self.__marks = marks
        else:
            print("Error: Marks must be between 0 and 100.")

    def get_marks(self):
        return self.__marks

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self._roll_no}")
        print(f"Marks: {self.__marks}")


s1 = Student("Bhumika", 101, 95)
s1.display_info()
print("\nPublic Access:", s1.name)
print("Protected Access:", s1._roll_no)
print("Private Access via getter:", s1.get_marks())
s1.set_marks(150)
```

```
ERROR!
Name: Aryaman
Roll No: 101
Marks: 95

Public Access: Aryaman
Protected Access: 101
Private Access via getter: 95
Error: Marks must be between 0 and 100.
```

3. Write a class Employee with name as public, _department as protected, and __salary as private. Provide methods to set and get salary safely

```
class Employee:
    def __init__(self, name, department, salary):
        self.name = name
        self._department = department
        self.__salary = None
        self.set_salary(salary)

    def set_salary(self, salary):
        if salary > 0:
            self.__salary = salary
        else:
            print("Error: Salary must be positive.")

    def get_salary(self):
        return self.__salary

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Department: {self._department}")
        print(f"Salary: {self.__salary}")

e1 = Employee("aryaman", "HR", 50000)
e1.display_info()
print("\nPublic Access:", e1.name)
print("Protected Access:", e1._department)
print("Private Access via getter:", e1.get_salary())
e1.set_salary(-10000)
```

Vaishnavi Ashok Patil
C07
SPP-II

```
Name: aryaman
Department: HR
Salary: 50000
ERROR!

Public Access: aryaman
Protected Access: HR
Private Access via getter: 50000
Error: Salary must be positive.
```

4. Create a class Book where title is public, _author is protected, and __copies is private. Make sure copies cannot go below zero.

```
class Book:
    def __init__(self, title, author, copies):
        self.title = title
        self._author = author
        self.__copies = None
        self.set_copies(copies)

    def set_copies(self, copies):
        if copies >= 0:
            self.__copies = copies
        else:
            print("Error: Number of copies cannot be negative.")

    def get_copies(self):
        return self.__copies

    def display_info(self):
        print(f"Title: {self.title}")
        print(f"Author: {self._author}")
        print(f"Copies: {self.__copies}")


b1 = Book("The Alchemist", "Paulo Coelho", 10)
b1.display_info()
print("\nPublic Access:", b1.title)
print("Protected Access:", b1._author)
print("Private Access via getter:", b1.get_copies())
b1.set_copies(-5)
```

```
ERROR!
Title: The Alchemist
Author: Paulo Coelho
Copies: 10

Public Access: The Alchemist
Protected Access: Paulo Coelho
Private Access via getter: 10
Error: Number of copies cannot be negative.
```

5. Develop a class Car with brand as public, _model as protected, and __speed as private. Speed should only change using accelerate and brake methods.

```
class Car:
    def __init__(self, brand, model, speed=0):
        self.brand = brand
        self._model = model
        self.__speed = speed

    def accelerate(self, amount):
        if amount > 0:
            self.__speed += amount
        else:
            print("Error: Acceleration amount must be positive.")
```

```
    def brake(self, amount):
        if 0 < amount <= self.__speed:
            self.__speed -= amount
        elif amount > self.__speed:
            self.__speed = 0
        else:
            print("Error: Brake amount must be positive.")

    def get_speed(self):
        return self.__speed

    def display_info(self):
        print(f"Brand: {self.brand}")
        print(f"Model: {self._model}")
        print(f"Speed: {self.__speed} km/h")


c1 = Car("Tesla", "Model S", 50)
c1.display_info()
c1.accelerate(30)
print("After acceleration:", c1.get_speed(), "km/h")
c1.brake(60)
print("After braking:", c1.get_speed(), "km/h")
print("\nPublic Access:", c1.brand)
print("Protected Access:", c1._model)
print("Private Access via getter:", c1.get_speed())
```

```
Brand: Tesla
Model: Model S
Speed: 50 km/h
After acceleration: 80 km/h
After braking: 20 km/h

Public Access: Tesla
Protected Access: Model S
Private Access via getter: 20
```

6. Make a class ATM with bank_name as public, _location as protected, and __pin and __balance as private. Allow deposit/withdraw only if pin matches.

```
class ATM:
    def __init__(self, bank_name, location, pin, balance=0):
        self.bank_name = bank_name
        self._location = location
        self.__pin = pin
        self.__balance = balance

    def deposit(self, amount, pin):
        if pin == self.__pin:
            if amount > 0:
                self.__balance += amount
                print(f"Deposited ₹{amount}. New balance: ₹{self.__balance}")
            else:
                print("Error: Deposit amount must be positive.")
        else:
            print("Error: Invalid PIN.")

    def withdraw(self, amount, pin):
        if pin == self.__pin:
            if 0 < amount <= self.__balance:
                self.__balance -= amount
                print(f"Withdrew ₹{amount}. Remaining balance: ₹{self.__balance}")
            else:
```

```
            print("Error: Invalid withdrawal amount.")
        else:
            print("Error: Invalid PIN.")

    def get_balance(self, pin):
        if pin == self.__pin:
            return self.__balance
        else:
            print("Error: Invalid PIN.")
            return None

    def display_info(self):
        print(f"Bank Name: {self.bank_name}")
        print(f"Location: {self._location}")


atm1 = ATM("SBI", "Mumbai", 1234, 5000)
atm1.display_info()
atm1.deposit(2000, 1234)
atm1.withdraw(1000, 1234)
print("Balance:", atm1.get_balance(1234))
atm1.withdraw(500, 1111)
```

```
Bank Name: SBI
ERROR!
Location: Mumbai
Deposited ₹2000. New balance: ₹7000
Withdrew ₹1000. Remaining balance: ₹6000
Balance: 6000
Error: Invalid PIN.
```

7. Design a class ShoppingCart with customer_name as public, _cart_id as protected, and __items as private. Items should be added/removed only through methods

```
class ShoppingCart:
    def __init__(self, customer_name, cart_id):
        self.customer_name = customer_name
        self._cart_id = cart_id
        self.__items = []

    def add_item(self, item):
        if item:
            self.__items.append(item)
            print(f"Added '{item}' to cart.")
        else:
            print("Error: Item name cannot be empty.")

    def remove_item(self, item):
        if item in self.__items:
            self.__items.remove(item)
            print(f"Removed '{item}' from cart.")
        else:
            print(f"Error: '{item}' not found in cart.")

    def view_cart(self):
        if self.__items:
            print("Items in cart:", ", ".join(self.__items))
        else:
            print("Cart is empty.")

    def display_info(self):
        print(f"Customer Name: {self.customer_name}")
        print(f"Cart ID: {self._cart_id}")
```

```
cart1 = ShoppingCart("Bhumika", "CART123")
cart1.display_info()
cart1.add_item("Shoes")
cart1.add_item("Bag")
cart1.view_cart()
cart1.remove_item("Shoes")
cart1.view_cart()
print("\nPublic Access:", cart1.customer_name)
print("Protected Access:", cart1._cart_id)
```

```
Customer Name: armaan
Cart ID: CART123
Added 'Shoes' to cart.
Added 'Bag' to cart.
Items in cart: Shoes, Bag
Removed 'Shoes' from cart.
Items in cart: Bag

Public Access: armaan
Protected Access: CART123
```

8. Write a class Patient with patient_name as public, _age as protected, and __disease as private. Only allow doctors (via method) to update disease.

```
class Patient:
    def __init__(self, patient_name, age, disease):
        self.patient_name = patient_name
        self._age = age
        self.__disease = disease

    def update_disease(self, new_disease, is_doctor):
        if is_doctor:
            self.__disease = new_disease
            print(f"Disease updated to '{new_disease}'.")
        else:
            print("Error: Only doctors can update the disease.")

    def get_disease(self, is_doctor):
        if is_doctor:
            return self.__disease
        else:
            print("Error: Access denied. Only doctors can view disease details.")
            return None

    def display_info(self):
        print(f"Patient Name: {self.patient_name}")
        print(f"Age: {self._age}")

p1 = Patient("Aryaman", 25, "Flu")
p1.display_info()
print("Disease (Doctor Access):", p1.get_disease(True))
p1.update_disease("Cold", True)
print("Disease (Patient Access):", p1.get_disease(False))
print("\nPublic Access:", p1.patient_name)
print("Protected Access:", p1._age)
```

```
ERROR!
Patient Name: Aryaman
Age: 25
Disease (Doctor Access): Flu
Disease updated to 'Cold'.
Error: Access denied. Only doctors can view disease details.
Disease (Patient Access): None

Public Access: Aryaman
Protected Access: 25
```

Vaishnavi Ashok Patil
C07
SPP-II

9. Create a class Course where course_name is public, _course_code is protected, and __students is private. Students should be added/removed through methods only.

```
class Course:
    def __init__(self, course_name, course_code):
        self.course_name = course_name
        self._course_code = course_code
        self.__students = []

    def add_student(self, student_name):
        if student_name:
            self.__students.append(student_name)
            print(f"Added student: {student_name}")
        else:
            print("Error: Student name cannot be empty.")

    def remove_student(self, student_name):
        if student_name in self.__students:
            self.__students.remove(student_name)
            print(f"Removed student: {student_name}")
        else:
            print(f"Error: {student_name} not found in the course.")

    def view_students(self):
        if self.__students:
            print("Enrolled Students:", ", ".join(self.__students))
        else:
            print("No students enrolled yet.")

    def display_info(self):
        print(f"Course Name: {self.course_name}")
        print(f"Course Code: {self._course_code}")

c1 = Course("Python Programming", "CS101")
c1.display_info()
c1.add_student("Bhumika")
c1.add_student("Aryaman")
c1.view_students()
c1.remove_student("Bhumika")
c1.view_students()
print("\nPublic Access:", c1.course_name)
print("Protected Access:", c1._course_code)
```

```
Course Name: Python Programming
Course Code: CS101
Added student: Surya
Added student: Aryaman
Enrolled Students: Surya, Aryaman
Removed student: Surya
Enrolled Students: Aryaman

Public Access: Python Programming
Protected Access: CS101
```

10. Make a class Loan where borrower_name is public, _loan_type is protected, and __loan_amount is private. Loan amount must not exceed 10,00,000.

```
class Loan:
    def __init__(self, borrower_name, loan_type, loan_amount):
        self.borrower_name = borrower_name
        self._loan_type = loan_type
        self.__loan_amount = None
        self.set_loan_amount(loan_amount)

    def set_loan_amount(self, amount):
        if 0 <= amount <= 1000000:
```

```
            self.__loan_amount = amount
        else:
            print("Error: Loan amount must not exceed ₹10,00,000.")

    def get_loan_amount(self):
        return self.__loan_amount

    def display_info(self):
        print(f"Borrower Name: {self.borrower_name}")
        print(f"Loan Type: {self._loan_type}")
        print(f"Loan Amount: ₹{self.__loan_amount}")

l1 = Loan("Bhumika", "Home Loan", 800000)
l1.display_info()
print("\nPublic Access:", l1.borrower_name)
print("Protected Access:", l1._loan_type)
print("Private Access via getter:", l1.get_loan_amount())
l1.set_loan_amount(1500000)
```

```
ERROR!
Borrower Name: vaivhavi
Loan Type: Home Loan
Loan Amount: ₹800000

Public Access: vaivhavi
Protected Access: Home Loan
Private Access via getter: 800000
Error: Loan amount must not exceed ₹10,00,000.
```

11. Write a class Mobile with brand as public, _model as protected, and __price as private. Allow price access only with getter and setter methods.

```
class Mobile:
    def __init__(self, brand, model, price):
        self.brand = brand
        self._model = model
        self.__price = None
        self.set_price(price)

    def set_price(self, price):
        if price > 0:
            self.__price = price
        else:
            print("Error: Price must be positive.")

    def get_price(self):
        return self.__price

    def display_info(self):
        print(f"Brand: {self.brand}")
        print(f"Model: {self._model}")
        print(f"Price: ₹{self.__price}")

m1 = Mobile("Apple", "iPhone 15", 120000)
m1.display_info()
print("\nPublic Access:", m1.brand)
print("Protected Access:", m1._model)
print("Private Access via getter:", m1.get_price())
m1.set_price(-5000)
```

```
ERROR!
Brand: Apple
Model: iPhone 15
Price: ₹120000

Public Access: Apple
Protected Access: iPhone 15
Private Access via getter: 120000
Error: Price must be positive.
```

12. Create a class Wallet where owner is public, _wallet_id is protected, and __balance is private. Balance can only be modified using add_funds and spend_funds methods.

```
class Wallet:
    def __init__(self, owner, wallet_id, balance=0):
        self.owner = owner
        self._wallet_id = wallet_id
        self.__balance = balance

    def add_funds(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Added ₹{amount}. New balance: ₹{self.__balance}")
        else:
            print("Error: Amount must be positive.")

    def spend_funds(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Spent ₹{amount}. Remaining balance: ₹{self.__balance}")
        else:
            print("Error: Insufficient balance or invalid amount.")

    def get_balance(self):
        return self.__balance

    def display_info(self):
        print(f"Owner: {self.owner}")
        print(f"Wallet ID: {self._wallet_id}")

w1 = Wallet("Bhumika", "W123", 5000)
w1.display_info()
w1.add_funds(2000)
w1.spend_funds(1500)
print("Current Balance:", w1.get_balance())
print("\nPublic Access:", w1.owner)
print("Protected Access:", w1._wallet_id)
```

```
Owner: Bhumika
Wallet ID: W123
Added ₹2000. New balance: ₹7000
Spent ₹1500. Remaining balance: ₹5500
Current Balance: 5500

Public Access: Bhumika
Protected Access: W123
```

13. Design a class Ticket with passenger_name as public, _flight_number as protected, and __seat_number as private. Seat number should be updated only by staff method.

```
class Ticket:
    def __init__(self, passenger_name, flight_number, seat_number):
        self.passenger_name = passenger_name
        self._flight_number = flight_number
        self.__seat_number = seat_number

    def update_seat(self, new_seat, is_staff):
        if is_staff:
            self.__seat_number = new_seat
```

```
            print(f"Seat number updated to {new_seat}.")
        else:
            print("Error: Only staff can update seat numbers.")

    def get_seat_number(self, is_staff):
        if is_staff:
            return self.__seat_number
        else:
            print("Error: Only staff can access seat number.")
            return None

    def display_info(self):
        print(f"Passenger Name: {self.passenger_name}")
        print(f"Flight Number: {self._flight_number}")


t1 = Ticket("Bhumika", "AI202", "12A")
t1.display_info()
print("Seat Number (Staff Access):", t1.get_seat_number(True))
t1.update_seat("15C", True)
print("Seat Number (Passenger Access):", t1.get_seat_number(False))
print("\nPublic Access:", t1.passenger_name)
print("Protected Access:", t1._flight_number)
```

```
ERROR!
Passenger Name: Bhumika
Flight Number: AI202
Seat Number (Staff Access): 12A
Seat number updated to 15C.
Error: Only staff can access seat number.
Seat Number (Passenger Access): None

Public Access: Bhumika
Protected Access: AI202
```

14. Make a class Customer with name as public, _customer_id as protected, and __credit_score as private. Credit score should be used internally for loan approval.

```
class Customer:
    def __init__(self, name, customer_id, credit_score):
        self.name = name
        self._customer_id = customer_id
        self.__credit_score = credit_score

    def __is_eligible_for_loan(self):
        return self.__credit_score >= 700

    def apply_for_loan(self, amount):
        if self.__is_eligible_for_loan():
            print(f"Loan of ₹{amount} approved for {self.name}.")
        else:
            print(f"Loan of ₹{amount} denied for {self.name} due to low credit score.")

    def display_info(self):
        print(f"Customer Name: {self.name}")
        print(f"Customer ID: {self._customer_id}")


c1 = Customer("huda", "C123", 750)
c1.display_info()
c1.apply_for_loan(500000)

c2 = Customer("Aryaman", "C124", 620)
c2.display_info()
```

```
c2.apply_for_loan(500000)

print("\nPublic Access:", c1.name)
print("Protected Access:", c1._customer_id)
```

```
Customer Name: huda
Customer ID: C123
Loan of ₹500000 approved for huda.
Customer Name: Aryaman
Customer ID: C124
Loan of ₹500000 denied for Aryaman due to low credit score.

Public Access: huda
Protected Access: C123
```

15. Write a class Exam with subject as public, _exam_code as protected, and __result as private. Only a teacher method should update the result.

```
class Exam:
    def __init__(self, subject, exam_code, result=None):
        self.subject = subject
        self._exam_code = exam_code
        self.__result = result

    def update_result(self, new_result, is_teacher):
        if is_teacher:
            self.__result = new_result
            print(f"Result updated to '{new_result}'.")
        else:
            print("Error: Only teachers can update the result.")

    def get_result(self, is_teacher):
        if is_teacher:
            return self.__result
        else:
            print("Error: Only teachers can access the result.")
            return None

    def display_info(self):
        print(f"Subject: {self.subject}")
        print(f"Exam Code: {self._exam_code}")


e1 = Exam("Mathematics", "EX101")
e1.display_info()
e1.update_result("Passed", True)
print("Result (Teacher Access):", e1.get_result(True))
print("Result (Student Access):", e1.get_result(False))
print("\nPublic Access:", e1.subject)
print("Protected Access:", e1._exam_code)
```

```
ERROR!
Subject: Mathematics
Exam Code: EX101
Result updated to 'Passed'.
Result (Teacher Access): Passed
Error: Only teachers can access the result.
Result (Student Access): None

Public Access: Mathematics
Protected Access: EX101
```

Vaishnavi Ashok Patil
C07
SPP-II

Inheritance

1. Write a program to create a class BankAccount with methods to deposit and withdraw money. Create a subclass SavingsAccount that adds a method to calculate interest.

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ₹{amount}. New balance: ₹{self.balance}")
        else:
            print("Error: Deposit amount must be positive.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ₹{amount}. Remaining balance: ₹{self.balance}")
        else:
            print("Error: Invalid or insufficient funds.")

    def display_info(self):
        print(f"Account Holder: {self.account_holder}")
        print(f"Balance: ₹{self.balance}")

class SavingsAccount(BankAccount):
    def calculate_interest(self, rate):
        interest = self.balance * (rate / 100)
        print(f"Interest at {rate}%: ₹{interest}")
        return interest

s1 = SavingsAccount("HUDA", 10000)
s1.display_info()
s1.deposit(5000)
s1.withdraw(2000)
s1.calculate_interest(5)
```

```
Account Holder: HUDA
Balance: ₹10000
Deposited ₹5000. New balance: ₹15000
Withdrew ₹2000. Remaining balance: ₹13000
Interest at 5%: ₹650.0
```

2. Define a class Person with attributes name and age. Create a subclass Student that adds roll number and marks. Write a program to create an object of Student and display all details.

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


class Student(Person):
    def __init__(self, name, age, roll_no, marks):
```

```
        super().__init__(name, age)
        self.roll_no = roll_no
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Roll No: {self.roll_no}")
        print(f"Marks: {self.marks}")


s1 = Student("Saransh", 20, 101, 95)
s1.display_details()
```

```
Name: Saransh
Age: 20
Roll No: 101
Marks: 95
```

3. Create a class Book with attributes title and author. Derive a class Magazine that adds attributes issue number and month. Demonstrate inheritance by creating objects of both.

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

    def display_info(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")


class Magazine(Book):
    def __init__(self, title, author, issue_number, month):
        super().__init__(title, author)
        self.issue_number = issue_number
        self.month = month

    def display_info(self):
        super().display_info()
        print(f"Issue Number: {self.issue_number}")
        print(f"Month: {self.month}")


b1 = Book("The Great Gatsby", "F. Scott Fitzgerald")
m1 = Magazine("National Geographic", "Various", 202, "October")

print("Book Details:")
b1.display_info()

print("\nMagazine Details:")
m1.display_info()
```

Vaishnavi Ashok Patil
C07
SPP-II

```
Book Details:
Title: The Great Gatsby
Author: F. Scott Fitzgerald

Magazine Details:
Title: National Geographic
Author: Various
Issue Number: 202
Month: October
```

4. Write a program with three classes: Grandparent, Parent, and Child. Each should have a method to introduce itself. Show how the child can access all methods.

```
class Grandparent:
    def introduce(self):
        print("I am the Grandparent.")

class Parent(Grandparent):
    def introduce(self):
        print("I am the Parent.")

class Child(Parent):
    def introduce(self):
        print("I am the Child.")

c1 = Child()

c1.introduce()              # Child's method
super(Child, c1).introduce()  # Parent's method using super() won't work directly, so:
print("\nAccessing all introductions:")
c1.introduce()              # Child method
Parent.introduce(c1)        # Parent method
Grandparent.introduce(c1)   # Grandparent method
```

```
I am the Child.
I am the Parent.

Accessing all introductions:
I am the Child.
I am the Parent.
I am the Grandparent.
```

5. Define a class Vehicle with attributes brand and speed. Derive a class Car that adds fuel type, and then derive ElectricCar from Car that adds battery capacity. Demonstrate multilevel inheritance.

```
class Vehicle:
    def __init__(self, brand, speed):
        self.brand = brand
        self.speed = speed

    def display_info(self):
        print(f"Brand: {self.brand}")
        print(f"Speed: {self.speed} km/h")

class Car(Vehicle):
    def __init__(self, brand, speed, fuel_type):
        super().__init__(brand, speed)
        self.fuel_type = fuel_type

    def display_info(self):
        super().display_info()
        print(f"Fuel Type: {self.fuel_type}")
```

```
class ElectricCar(Car):
    def __init__(self, brand, speed, fuel_type, battery_capacity):
        super().__init__(brand, speed, fuel_type)
        self.battery_capacity = battery_capacity

    def display_info(self):
        super().display_info()
        print(f"Battery Capacity: {self.battery_capacity} kWh")

e1 = ElectricCar("Tesla", 200, "Electric", 85)
print("Electric Car Details:")
e1.display_info()
```

```
Electric Car Details:
Brand: Tesla
Speed: 200 km/h
Fuel Type: Electric
Battery Capacity: 85 kWh
```

6. Create a class Employee with a salary attribute. Derive a class Manager that adds department, and then a class Director that adds decision-making methods. Show how inheritance works in the chain.

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

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Salary: ₹{self.salary}")

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display_info(self):
        super().display_info()
        print(f"Department: {self.department}")

class Director(Manager):
    def __init__(self, name, salary, department):
        super().__init__(name, salary, department)

    def make_decision(self, decision):
        print(f"Director Decision: {decision}")

    def display_info(self):
        super().display_info()
        print("Position: Director")

d1 = Director("Rahul", 150000, "Operations")
print("Director Details:")
d1.display_info()
d1.make_decision("Approve new project budget")
```

```
Director Details:
Name: Rahul
Salary: ₹150000
Department: Operations
Position: Director
Director Decision: Approve new project budget
```

Vaishnavi Ashok Patil
C07
SPP-II

7. Write a program with a class Sports that has a method play(), and a class Music that has a method sing(). Create a class Student that inherits from both and demonstrate that a student can do both.

```
class Sports:
    def play(self):
        print("The student is playing a sport.")

class Music:
    def sing(self):
        print("The student is singing a song.")

class Student(Sports, Music):
    def study(self):
        print("The student is studying.")


s1 = Student()
s1.play()
s1.sing()
s1.study()
```

```
The student is playing a sport.
The student is singing a song.
The student is studying.
```

8. Define two classes MathTeacher and ScienceTeacher, each with a teach() method. Create a Teacher class that inherits from both and demonstrate multiple inheritance.

```
class MathTeacher:
    def teach(self):
        print("Teaching Mathematics...")

class ScienceTeacher:
    def teach(self):
        print("Teaching Science...")

class Teacher(MathTeacher, ScienceTeacher):
    def introduce(self):
        print("I am a teacher who can teach multiple subjects.")


t1 = Teacher()
t1.introduce()
t1.teach()
print("\nCalling ScienceTeacher method explicitly:")
ScienceTeacher.teach(t1)
```

```
I am a teacher who can teach multiple subjects.
Teaching Mathematics...

Calling ScienceTeacher method explicitly:
Teaching Science...
```

9. Write a program with a class Phone (with a call method) and a class Camera (with a take_photo method). Create a SmartPhone class that inherits from both and can make calls and take photos.

```
class Phone:
    def call(self):
        print("Making a phone call...")

class Camera:
    def take_photo(self):
        print("Taking a photo...")
```

```
class SmartPhone(Phone, Camera):
    def browse_internet(self):
        print("Browsing the internet...")


sp1 = SmartPhone()
sp1.call()
sp1.take_photo()
sp1.browse_internet()
```

```
Making a phone call...
Taking a photo...
Browsing the internet...
```

10. Create a base class Animal with a method sound(). Derive classes Dog and Cat that override sound(). Demonstrate hierarchical inheritance by creating objects of both.

```
class Animal:
    def sound(self):
        print("This animal makes a sound.")

class Dog(Animal):
    def sound(self):
        print("The dog barks.")

class Cat(Animal):
    def sound(self):
        print("The cat meows.")

dog1 = Dog()
cat1 = Cat()

print("Dog:")
dog1.sound()

print("\nCat:")
cat1.sound()
```

```
Dog:
The dog barks.

Cat:
The cat meows.
```

11. Define a base class Shape with a method area(). Derive classes Circle and Rectangle that implement their own area() methods. Demonstrate inheritance by calculating areas.

```
import math

class Shape:
    def area(self):
        print("Area not defined for generic shape.")

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

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

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

```
    def area(self):
        return self.length * self.width

circle1 = Circle(5)
rectangle1 = Rectangle(4, 6)

print("Area of Circle:", circle1.area())
print("Area of Rectangle:", rectangle1.area())
```

```
Area of Circle: 78.53981633974483
Area of Rectangle: 24
```

12. Create a base class Employee with attributes name and ID. Derive two classes FullTimeEmployee and PartTimeEmployee, each with its own salary calculation. Demonstrate hierarchical inheritance.

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

    def display_info(self):
        print(f"Employee Name: {self.name}")
        print(f"Employee ID: {self.emp_id}")


class FullTimeEmployee(Employee):
    def __init__(self, name, emp_id, monthly_salary):
        super().__init__(name, emp_id)
        self.monthly_salary = monthly_salary

    def calculate_salary(self):
        return self.monthly_salary

class PartTimeEmployee(Employee):
    def __init__(self, name, emp_id, hours_worked, hourly_rate):
        super().__init__(name, emp_id)
        self.hours_worked = hours_worked
        self.hourly_rate = hourly_rate

    def calculate_salary(self):
        return self.hours_worked * self.hourly_rate

full_time = FullTimeEmployee("Aarav", 101, 50000)
part_time = PartTimeEmployee("Neha", 102, 80, 300)

print("Full-Time Employee Details:")
full_time.display_info()
print("Monthly Salary:", full_time.calculate_salary())
print("\nPart-Time Employee Details:")
part_time.display_info()
print("Monthly Salary:", part_time.calculate_salary())
```

```
Full-Time Employee Details:
Employee Name: Aarav
Employee ID: 101
Monthly Salary: 50000

Part-Time Employee Details:
Employee Name: Neha
Employee ID: 102
Monthly Salary: 24000
```

13. Define a base class Person. Create subclasses Teacher and Student. Then create a TeachingAssistant class that inherits from both Teacher and Student. Demonstrate hybrid inheritance.

```
class Person:
    def __init__(self, name):
```

```
        self.name = name

    def show(self):
        print(f"Name: {self.name}")


class Teacher(Person):
    def __init__(self, name, subject):
        super().__init__(name)
        self.subject = subject

    def teach(self):
        print(f"{self.name} teaches {self.subject}.")

class Student(Person):
    def __init__(self, name, course):
        super().__init__(name)
        self.course = course

    def study(self):
        print(f"{self.name} is studying {self.course}.")

class TeachingAssistant(Teacher, Student):
    def __init__(self, name, subject, course):
        Teacher.__init__(self, name, subject)
        Student.__init__(self, name, course)

    def assist(self):
        print(f"{self.name} assists in {self.subject} while studying {self.course}.")

ta = TeachingAssistant("Rahul", "Physics", "Computer Science")
ta.show()
ta.teach()
ta.study()
ta.assist()
```

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 37, in <module>
  File "<main.py>", line 29, in __init__
  File "<main.py>", line 11, in __init__
TypeError: Student.__init__() missing 1 required positional argument: 'course'
```

14. Create a base class Vehicle. Derive TwoWheeler and FourWheeler classes. Then derive a HybridVehicle class that inherits from both. Show how hybrid inheritance works.

```
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def show(self):
        print(f"Vehicle Brand: {self.brand}")

class TwoWheeler(Vehicle):
    def __init__(self, brand, type_2w):
        Vehicle.__init__(self, brand)
        self.type_2w = type_2w

    def display_two(self):
        print(f"Two Wheeler Type: {self.type_2w}")

class FourWheeler(Vehicle):
```

```
    def __init__(self, brand, type_4w):
        Vehicle.__init__(self, brand)
        self.type_4w = type_4w

    def display_four(self):
        print(f"Four Wheeler Type: {self.type_4w}")

class HybridVehicle(TwoWheeler, FourWheeler):
    def __init__(self, brand, type_2w, type_4w):
        TwoWheeler.__init__(self, brand, type_2w)
        FourWheeler.__init__(self, brand, type_4w)

    def show_hybrid(self):
        print(f"{self.brand} is a hybrid of {self.type_2w} and {self.type_4w}.")

hv = HybridVehicle("Toyota", "Bike", "Car")
hv.show()
hv.display_two()
hv.display_four()
hv.show_hybrid()
```

```
Vehicle Brand: Toyota
Two Wheeler Type: Bike
Four Wheeler Type: Car
Toyota is a hybrid of Bike and Car.
```

15. Write a program with a base class Person. Derive Employee and Freelancer from it. Create a HybridWorker class that inherits from both and can earn from multiple sources. Demonstrate hybrid inheritance.

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

    def show(self):
        print(f"Name: {self.name}")

class Employee(Person):
    def __init__(self, name, salary):
        Person.__init__(self, name)
        self.salary = salary

    def show_employee(self):
        print(f"{self.name} earns a salary of ₹{self.salary} as an employee.")

class Freelancer(Person):
    def __init__(self, name, project_income):
        Person.__init__(self, name)
        self.project_income = project_income

    def show_freelancer(self):
        print(f"{self.name} earns ₹{self.project_income} from freelance projects.")

class HybridWorker(Employee, Freelancer):
    def __init__(self, name, salary, project_income):
        Employee.__init__(self, name, salary)
        Freelancer.__init__(self, name, project_income)

    def total_income(self):
        total = self.salary + self.project_income
```

```
        print(f"{self.name}'s total income from both sources is ₹{total}.")

hw = HybridWorker("Karan", 50000, 20000)
hw.show()
hw.show_employee()
hw.show_freelancer()
hw.total_income()
```

```
Name: Karan
Karan earns a salary of ₹50000 as an employee.
Karan earns ₹20000 from freelance projects.
Karan's total income from both sources is ₹70000.
```

Polymorphism

- # Compile-Time Polymorphism

1. Create a class Calculator with a method add() that can add two or three numbers.

```
class Calculator:
    def add(self, a, b, c=None):
        if c is not None:
            return a + b + c
        else:
            return a + b

calc = Calculator()
print("Sum of 2 numbers:", calc.add(5, 10))
print("Sum of 3 numbers:", calc.add(5, 10, 15))
```

```
Sum of 2 numbers: 15
Sum of 3 numbers: 30
```

2. Write a function greet() that prints "Hello, ". If an optional message is provided, it should print "Hello, , ".

```
def greet(name, message=None):
    if message:
        print(f"Hello, {name}, {message}")
    else:
        print(f"Hello, {name}")

greet("Riya")
greet("Riya", "good morning!")
```

```
Hello, Riya
Hello, Riya, good morning!
```

3. Write a method calculate_interest(amount, rate=5) that calculates simple interest. If no rate is given, use the default rate of 5%.

```
 class Bank:
    def calculate_interest(self, amount, rate=5):
        interest = (amount * rate) / 100
        return interest

b = Bank()
print("Interest with default rate:", b.calculate_interest(10000))
print("Interest with custom rate:", b.calculate_interest(10000, 8))
```

```
Interest with default rate: 500.0
Interest with custom rate: 800.0
```

4. Create a class Student with a method show_info() that prints the student's name and roll number. If age is provided, print that too.

Vaishnavi Ashok Patil
C07
SPP-II

```
class Student:
    def show_info(self, name, roll_no, age=None):
        if age:
            print(f"Name: {name}, Roll No: {roll_no}, Age: {age}")
        else:
            print(f"Name: {name}, Roll No: {roll_no}")

s1 = Student()
s1.show_info("Aarav", 101)
s1.show_info("Mira", 102, 20)
```

```
Name: Aarav, Roll No: 101
Name: Mira, Roll No: 102, Age: 20
```

5. Write a method area() that calculates the area of a square when only one parameter is given, and the area of a rectangle when two parameters are given. # Using *args

```
class Shape:
    def area(self, *args):
        if len(args) == 1:
            return args[0] ** 2  # Square
        elif len(args) == 2:
            return args[0] * args[1]  # Rectangle
        else:
            return "Invalid number of arguments"
s = Shape()
print("Area of Square:", s.area(6))
print("Area of Rectangle:", s.area(6, 8))
```

```
Area of Square: 36
Area of Rectangle: 48
```

6. Write a method sum_numbers(*args) that returns the sum of any number of integers passed to it.

```
class Calculator:
    def sum_numbers(self, *args):
        return sum(args)

c = Calculator()
print("Sum:", c.sum_numbers(5, 10, 15, 20))
```

```
Sum: 50
```

7. Write a function multiply(*args) that calculates and returns the product of all given numbers.

```
def multiply(*args):
    product = 1
    for num in args:
        product *= num
    return product

print("Product:", multiply(2, 3, 4))
```

```
Product: 24
```

]

8. Write a function concat_strings(*args) that takes multiple strings and returns them joined together as one string.

```
def concat_strings(*args):
    return "".join(args)

print("Concatenated String:", concat_strings("Hello", " ", "World", "!"))
```

```
Concatenated String: Hello World!
```

9. Write a function find_max(*args) that returns the largest number among the given inputs.

```
def find_max(*args):
```

```
    return max(args)
print("Maximum number:", find_max(10, 25, 7, 42, 19))
```
```
Maximum number: 42
```

10.Write a function calculate_bill(*items) where each item represents a price. The function should return the total bill amount

```
def calculate_bill(*items):
    return sum(items)

print("Total Bill Amount:", calculate_bill(120.50, 89.99, 45.00, 30.25))
```
```
Total Bill Amount: 285.74
```

- # Runtime Polymorphism

Method Overriding

1. Create a Shape base class with a draw() method. Override it in Circle and Square.

```
class Shape:
    def draw(self):
        print("Drawing a generic shape.")

class Circle(Shape):
    def draw(self):
        print("Drawing a Circle.")

class Square(Shape):
    def draw(self):
        print("Drawing a Square.")

s1 = Circle()
s2 = Square()
s1.draw()
s2.draw()
```
```
Drawing a Circle.
Drawing a Square.
```

2. Create a Vehicle class with speed(). Override it in Car and Bike.

```
class Vehicle:
    def speed(self):
        print("The vehicle speed varies.")

class Car(Vehicle):
    def speed(self):
        print("The car runs at 120 km/h.")

class Bike(Vehicle):
    def speed(self):
        print("The bike runs at 80 km/h.")

v1 = Car()
v2 = Bike()
v1.speed()
v2.speed()
```
```
The car runs at 120 km/h.
The bike runs at 80 km/h.
```

3. Create an Employee class with salary(). Override it in Manager and Developer.

```
class Employee:
    def salary(self):
        print("Base salary for an employee.")

class Manager(Employee):
    def salary(self):
```

```
        print("Manager salary: ₹80,000 per month.")

    class Developer(Employee):
        def salary(self):
            print("Developer salary: ₹60,000 per month.")

    m = Manager()
    d = Developer()
    m.salary()
    d.salary()
```

```
Manager salary: ₹80,000 per month.
Developer salary: ₹60,000 per month.
```

4. Create a BankAccount class with interest_rate(). Override in SavingsAccount and CurrentAccount.

```
class BankAccount:
    def interest_rate(self):
        print("Base interest rate applies.")

class SavingsAccount(BankAccount):
    def interest_rate(self):
        print("Savings Account interest rate: 6%.")

class CurrentAccount(BankAccount):
    def interest_rate(self):
        print("Current Account interest rate: 3%.")

s = SavingsAccount()
c = CurrentAccount()
s.interest_rate()
c.interest_rate()
```

```
Savings Account interest rate: 6%.
Current Account interest rate: 3%.
```

5. Create a Bird class with fly(). Override it in Eagle and Penguin.

```
    class Bird:
        def fly(self):
            print("Birds can usually fly.")

    class Eagle(Bird):
        def fly(self):
            print("Eagle soars high in the sky.")

    class Penguin(Bird):
        def fly(self):
            print("Penguins cannot fly but can swim.")

    e = Eagle()
    p = Penguin()
    e.fly()
    p.fly()
```

```
Eagle soars high in the sky.
Penguins cannot fly but can swim.
```

6. Create a Payment class with pay(). Override it in CreditCard and UPI.

```
    class Payment:
        def pay(self):
            print("Processing generic payment.")

    class CreditCard(Payment):
        def pay(self):
```

```
        print("Payment made through Credit Card.")

class UPI(Payment):
    def pay(self):
        print("Payment made through UPI.")

cc = CreditCard()
u = UPI()
cc.pay()
u.pay()
```

```
Payment made through Credit Card.
Payment made through UPI.
```

7. Create a Device class with start(). Override it in Laptop and Mobile.

```
class Device:
    def start(self):
        print("Starting a generic device.")
class Laptop(Device):
    def start(self):
        print("Laptop is booting up.")
class Mobile(Device):
    def start(self):
        print("Mobile is powering on.")
l = Laptop()
m = Mobile()
l.start()
m.start()
```

```
Laptop is booting up.
Mobile is powering on.
```

8. Create a Teacher class with teach(). Override it in MathTeacher and ScienceTeacher.

```
class Teacher:
    def teach(self):
        print("Teaching a general subject.")

class MathTeacher(Teacher):
    def teach(self):
        print("Teaching Mathematics.")

class ScienceTeacher(Teacher):
    def teach(self):
        print("Teaching Science.")

t1 = MathTeacher()
t2 = ScienceTeacher()
t1.teach()
t2.teach()
```

```
Teaching Mathematics.
Teaching Science.
```

9. Create a Sports class with play(). Override it in Football and Cricket.

```
class Teacher:
    def teach(self):
        print("Teaching a general subject.")

class MathTeacher(Teacher):
    def teach(self):
        print("Teaching Mathematics.")

class ScienceTeacher(Teacher):
    def teach(self):
        print("Teaching Science.")
t1 = MathTeacher()
```

```
t2 = ScienceTeacher()
t1.teach()
t2.teach()
```
```
Playing Football.
Playing Cricket.
```

10. Create a Notification class with send(). Override it in EmailNotification and SMSNotification.

```
class Sports:
    def play(self):
        print("Playing a general sport.")

class Football(Sports):
    def play(self):
        print("Playing Football.")

class Cricket(Sports):
    def play(self):
        print("Playing Cricket.")


f = Football()
c = Cricket()
f.play()
c.play()
```
```
Sending Email Notification.
Sending SMS Notification.
```

Duck Typing

1. Write a function make_sound(animal) that works for Dog and Duck classes (both have sound()).

```
class Dog:
    def sound(self):
        print("Dog barks.")

class Duck:
    def sound(self):
        print("Duck quacks.")

def make_sound(animal):
    animal.sound()

d = Dog()
du = Duck()
make_sound(d)
make_sound(du)
```
```
Dog barks.
Duck quacks.
```

2. Create a function start_engine(vehicle) that works for Car and Bike classes (both have start() method).

```
class Car:
    def start(self):
        print("Car engine started.")

class Bike:
    def start(self):
        print("Bike engine started.")

def start_engine(vehicle):
    vehicle.start()

c = Car()
b = Bike()
```

```
start_engine(c)
start_engine(b)
```

```
Car engine started.
Bike engine started.
```

3. Write a function operate(machine) that works for Printer and Scanner classes (both have run() method).

```
class Printer:
    def run(self):
        print("Printer is printing documents.")

class Scanner:
    def run(self):
        print("Scanner is scanning documents.")

def operate(machine):
    machine.run()

p = Printer()
s = Scanner()
operate(p)
operate(s)
```

```
Printer is printing documents.
Scanner is scanning documents.
```

4. Write a function play_game(player) that works for Footballer and Cricketer (both have play() method).

```
class Footballer:
    def play(self):
        print("Footballer is playing football.")

class Cricketer:
    def play(self):
        print("Cricketer is playing cricket.")

def play_game(player):
    player.play()


f = Footballer()
c = Cricketer()
play_game(f)
play_game(c)
```

```
Footballer is playing football.
Cricketer is playing cricket.
```

5. Write a function book_ticket(transport) that works for Bus and Train (both have book() method).

```
class Bus:
    def book(self):
        print("Bus ticket booked.")

class Train:
    def book(self):
        print("Train ticket booked.")

def book_ticket(transport):
    transport.book()


b = Bus()
t = Train()
book_ticket(b)
book_ticket(t)
```

```
Bus ticket booked.
Train ticket booked.
```

6. Write a function make_payment(method) that works for CreditCard and PayPal (both have pay() method).

```
class CreditCard:
    def pay(self):
        print("Payment made using Credit Card.")

class PayPal:
    def pay(self):
        print("Payment made using PayPal.")

def make_payment(method):
    method.pay()

c = CreditCard()
p = PayPal()
make_payment(c)
make_payment(p)
```

```
Payment made using Credit Card.
Payment made using PayPal.
```

7. Write a function charge(device) that works for Phone and Laptop (both have charge() method).

```
class Phone:
    def charge(self):
        print("Phone is charging.")

class Laptop:
    def charge(self):
        print("Laptop is charging.")

def charge(device):
    device.charge()

ph = Phone()
lp = Laptop()
charge(ph)
charge(lp)
```

```
Phone is charging.
Laptop is charging.
```

8. Write a function study(student) that works for SchoolStudent and CollegeStudent (both have study() method).

```
class SchoolStudent:
    def study(self):
        print("School student is studying basic subjects.")

class CollegeStudent:
    def study(self):
        print("College student is studying specialized courses.")

def study(student):
    student.study()

s = SchoolStudent()
c = CollegeStudent()
study(s)
study(c)
```

```
School student is studying basic subjects.
College student is studying specialized courses.
```

9. Write a function draw(shape) that works for Circle and Rectangle (both have draw() method).

```
class Circle:
    def draw(self):
        print("Drawing a Circle.")

class Rectangle:
    def draw(self):
        print("Drawing a Rectangle.")

def draw(shape):
    shape.draw()

c = Circle()
r = Rectangle()
draw(c)
draw(r)
```

```
Drawing a Circle.
Drawing a Rectangle.
```

10. Write a function open_file(filetype) that works for PDF and WordDoc (both have open() method).

```
class PDF:
    def open(self):
        print("Opening a PDF file.")

class WordDoc:
    def open(self):
        print("Opening a Word document.")

def open_file(filetype):
    filetype.open()

p = PDF()
w = WordDoc()
open_file(p)
open_file(w)
```

```
Opening a PDF file.
Opening a Word document.
```

Operator Overloading

1. Create a Point class and overload + to add two points.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"

p1 = Point(2, 3)
p2 = Point(4, 5)
print(p1 + p2)
```

```
(6, 8)
```

2. Create a Vector class and overload - to subtract two vectors.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)
    def __str__(self):
        return f"({self.x}, {self.y})"
v1 = Vector(8, 4)
v2 = Vector(3, 2)
print(v1 - v2)
```

```
(5, 2)
```

3. Create a Distance class and overload > to compare two distances.

```
class Distance:
    def __init__(self, km):
        self.km = km

    def __gt__(self, other):
        return self.km > other.km

d1 = Distance(12)
d2 = Distance(8)
print(d1 > d2)
```

```
True
```

4. Create a Student class and overload == to compare marks of two students.

```
class Student:
    def __init__(self, marks):
        self.marks = marks

    def __eq__(self, other):
        return self.marks == other.marks

s1 = Student(85)
s2 = Student(85)
print(s1 == s2)
```

```
True
```

5. Create a Time class and overload < to compare two times.

```
class Time:
    def __init__(self, hour, minute):
        self.hour = hour
        self.minute = minute

    def __lt__(self, other):
        return (self.hour, self.minute) < (other.hour, other.minute)

t1 = Time(5, 30)
t2 = Time(6, 15)
print(t1 < t2)
```

```
True
```

6. Create a ComplexNumber class and overload * to multiply two complex numbers.

```
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __mul__(self, other):
```

```
        r = self.real * other.real - self.imag * other.imag
        i = self.real * other.imag + self.imag * other.real
        return ComplexNumber(r, i)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = ComplexNumber(3, 2)
c2 = ComplexNumber(1, 7)
print(c1 * c2)
```

```
-11 + 23i
```

7. Create a Book class and overload + to add pages of two books.

```
class Book:
    def __init__(self, pages):
        self.pages = pages

    def __add__(self, other):
        return Book(self.pages + other.pages)

    def __str__(self):
        return f"Total pages: {self.pages}"

b1 = Book(120)
b2 = Book(180)
print(b1 + b2)
```

```
Total pages: 300
```

8. Create a BankAccount class and overload + to merge balances of two accounts.

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def __add__(self, other):
        return BankAccount(self.balance + other.balance)

    def __str__(self):
        return f"Total balance: ₹{self.balance}"

a1 = BankAccount(5000)
a2 = BankAccount(3500)
print(a1 + a2)
```

```
Total balance: ₹8500
```

9. Create a Fraction class and overload / to divide two fractions.

```
class Fraction:
    def __init__(self, num, den):
        self.num = num
        self.den = den

    def __truediv__(self, other):
        return Fraction(self.num * other.den, self.den * other.num)

    def __str__(self):
        return f"{self.num}/{self.den}"

f1 = Fraction(3, 4)
f2 = Fraction(5, 6)
print(f1 / f2)
```

Vaishnavi Ashok Patil
C07
SPP-II

```
18/20
```

10. Create a Rectangle class and overload == to check if two rectangles have the same area.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def __eq__(self, other):
        return self.length * self.width == other.length * other.width
r1 = Rectangle(4, 5)
r2 = Rectangle(10, 2)
print(r1 == r2)
```

```
True
```