Lab#2

Introduction to Object-Oriented Programming (OOP) with Python Lab

This lab introduces Object-Oriented Programming (OOP) in Python, covering classes, objects, encapsulation, inheritance, and polymorphism. Through code examples and exercises, you'll learn how to apply these concepts to structure code effectively.

Objectives

- Understand and implement core OOP principles.
- Create and manipulate classes and objects.
- Apply encapsulation, inheritance, and polymorphism to solve problems.

Part 1: Classes and Objects

In OOP, a **class** is like a template or blueprint that defines the structure and behavior of objects. It specifies what data (attributes) an object can hold and what actions (methods) it can perform. An **object** is a specific instance of a class, created with actual data. Think of a class as a cookie cutter and an object as a cookie made from it. The __init__ method initializes an object's attributes when it's created, and self refers to the object itself, allowing access to its data and methods.

Example:

```
class Car:
    def __init__(self, brand):
        self.brand = brand
        self.running = False

    def start(self):
        self.running = True
        return f"{self.brand} started."

car = Car("Toyota")
print(car.start())  # Output: Toyota started.
```

Part 2: Encapsulation

Encapsulation is about protecting an object's data by making attributes private and controlling access through public methods. In Python, private attributes are denoted with a double underscore (__), which prevents direct access from outside the class. Public methods (like getters and setters) allow controlled interaction with private data, ensuring the object's state remains valid. For example, in a bank account, you'd want to prevent direct changes to the balance but allow deposits through a method that checks for valid amounts.

Example:

```
class BankAccount:
    def __init__(self, holder, balance=0):
        self.holder = holder
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            return f"New balance: ${self.__balance}"
        return "Invalid amount."

    def get_balance(self):
        return self.__balance

account = BankAccount("Alice", 100)
print(account.deposit(50))  # Output: New balance: $150
print(account.get_balance())  # Output: 150
```

Part 3: Inheritance

Inheritance allows a class (child) to inherit attributes and methods from another class (parent), promoting code reuse. The child class can use or modify the parent's features. The super() function calls the parent's methods, such as __init__, to initialize inherited attributes. For example, a Car and Bike might both inherit from a Vehicle class, sharing common traits like brand but adding their own unique features.

Example:

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

    def info(self):
        return f"{self.brand} vehicle"

class Bike(Vehicle):
```

```
    def info(self):  # Override parent method
        return f"{self.brand} bike"

bike = Bike("Yamaha")
print(bike.info())  # Output: Yamaha bike
```

Part 4: Polymorphism

Polymorphism means “many forms” and allows different classes to be treated as instances of a common parent class through shared method names. By overriding a parent class’s method, each child class can provide its own implementation. For example, a Shape class might define an area method, and child classes like Circle and Square implement it differently. This lets you call area on any shape without knowing its specific type.

Example:

```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2

class Square(Shape):
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side ** 2

shapes = [Circle(3), Square(4)]
for shape in shapes:
    print(shape.area())  # Output: 28.26, 16
```

Lab Task

Exercise 1

1. Create a Student class with attributes name, id, and grade_level.
2. Add a method display_info that returns a string with the student's details.
3. Create two Student objects and call display_info for each.

Exercise 2

1. Create a LibraryBook class with private attributes __title, __author, and __is_checked_out.
2. Add methods check_out and return_book to update __is_checked_out.
3. Add a getter method get_book_info to return the book's details and status.
4. Test the class by creating a book, checking it out, and returning it.

Exercise 3

1. Create a parent class Animal with attributes name and species, and a method make_sound.
2. Create a child class Dog that inherits from Animal and overrides make_sound to return "Woof!".
3. Create a Dog object and test the make_sound method.

Exercise 4

1. Create a parent class Employee with a method calculate_salary.
2. Create two child classes FullTimeEmployee and PartTimeEmployee that override calculate_salary.
3. For FullTimeEmployee, return a fixed salary (e.g., 50000).
4. For PartTimeEmployee, calculate salary based on hours worked and hourly rate.
5. Create a list of employees and print their salaries using a loop.

Exercise 5

1. Complete the Library class by adding methods to check out and return books by title.
2. Test the system by adding multiple books, checking some out, and listing available books.
3. Add error handling for cases like checking out an unavailable book.

```
#Task 1
class LibraryBook:
    def __init__(self, title, author):
        self.__title = title
        self.__author = author
        self.__is_checked_out = False
    def check_out(self):
        if not self.__is_checked_out:
            self.__is_checked_out = True
            print(f"'{self.__title}' has been checked out.")
        else:
            print(f"'{self.__title}' is already checked out.")
    def return_book(self):
        if self.__is_checked_out:
            self.__is_checked_out = False
            print(f"'{self.__title}' has been returned.")
        else:
            print(f"'{self.__title}' was not checked out.")
    def get_book_info(self):
        """Returns the book's details and availability status."""
        status = "Checked Out" if self.__is_checked_out else "Available"
        return f"Title: {self.__title}, Author: {self.__author}, Status: {status}"
def main():
    title=input("Enter Book's Title:")
    author=input("Enter Book's Author Name:")
    book = LibraryBook(title,author)
    print(book.get_book_info())
    book.check_out()
    print(book.get_book_info())
    book.check_out()
    book.return_book()
    print(book.get_book_info())
    book.return_book()

#Task 2
class Animal:
    def __init__(self,name,species):
        self.name=name
        self.species=species
    def make_sound(self):
        print("Animal Sound.")
class Dog(Animal):
    def make_sound(self):
        print("Woof!")
def main():
    name=input("Enter Name of Dog:")
    species=input("Enter Specie of Dog:")
    d=Dog(name,species)
    d.make_sound()
main()
```

```
  #Task 3
  import string
  class Student():
     def __init__(self,name,id,grade_level):
        self.name=name
        self.id=id
        self.grade_level=grade_level
     def display_info(self):
        try:
           print("Your Name is:",self.name,"\nyour Id is:",self.id,"\nyour grade level is:",self.grade_level)
        except:
           print("An Error Occurred!")
  def main():
     name=input("Enter Your Name:")
     id=int(input("Enter Your Id:"))
     grade_level=input("Enter Your Grade Level:")
     b=Student(name,id,grade_level)
     b.display_info()
  main()
#Task 4
employee_type=input("Enter Employee Type(Full Time Emplpoyee/Part Time Employee):").lower()
class Employee:
   def calculate_salary(self):
      print("Calculating Salary...")
class FullTimeEmployee(Employee):
   def __init__(self,salary):
      self.salary=salary
      salary=50000
   def calculate_salary(self):
      print("Salary of Full Time Employee is:",self.salary,".Rs")
class PartTimeEmployee(Employee):
   def __init__(self,hourly_rate,hours_worked):
      self.hourly_rate=hourly_rate
      self.hours_worked=hours_worked
   def calculate_salary(self):
      print("Salary of Part Time Employee is:",self.hourly_rate*self.hours_worked,".Rs")
def main():
   if employee_type=="full time employee":
      f=FullTimeEmployee(salary=50000)
      f.calculate_salary()
   elif employee_type=="part time employee":
      hourly_rate=int(input("Enter Your Hourly Rate:"))
      hours_worked=int(input("Enter Hours You Worked:"))
      p=PartTimeEmployee(hourly_rate,hours_worked)
      p.calculate_salary()
   else:
      print("Invalid Employee Type!")
main()
```

```
#Task 5
class LibraryBook:
    def __init__(self, title, author):
        self.__title = title
        self.__author = author
        self.__is_checked_out = False
    def check_out(self):
        if not self.__is_checked_out:
            self.__is_checked_out = True
            print(self.__title,"has been checked out.")
        else:
            print("Error:",self.__title,"is already checked out.")
    def return_book(self):
        if self.__is_checked_out:
            self.__is_checked_out = False
            print(self.__title,"has been returned.")
        else:
            print("Error:",self.__title,"was not checked out.")
    def get_book_info(self):
        status = "Checked Out" if self.__is_checked_out else "Available"
        return ("Title:",self.__title,"Author:",self.__author," Status:",status)
    def get_title(self):
        return self.__title
class Library:
    def __init__(self):
        self.__books = []
    def add_book(self, book):
        self.__books.append(book)
        print("Added",book.get_title(),"to the library.")
    def list_books(self):
        if not self.__books:
            print("No books in the library.")
            return
        print("\nLibrary Catalog:")
        for book in self.__books:
            print("- ",book.get_book_info())
    def check_out_book(self, title):
        for book in self.__books:
            if book.get_title().lower() == title.lower():
                book.check_out()
                return
        print("Error: Book titled,",title,"not found in the library.")
    def return_book(self, title):
        for book in self.__books:
            if book.get_title().lower() == title.lower():
                book.return_book()
                return
        print("Error: Book titled",title,"not found in the library.")
def main():
    library = Library()
    library.add_book(LibraryBook("1984", "George Orwell"))
    library.add_book(LibraryBook("To Kill a Mockingbird", "Harper Lee"))
    library.add_book(LibraryBook("The Great Gatsby", "F. Scott Fitzgerald"))
    library.list_books()
    library.check_out_book("1984")
    library.check_out_book("The Great Gatsby")
    library.check_out_book("1984")
    library.return_book("1984")
    library.return_book("To Kill a Mockingbird")
    library.check_out_book("Moby Dick")
    library.list_books()
main()
```