PYTHON

Assignment

1. **Task: Implement Duck Typing**

- Create two different classes (e.g., `Dog` and `Cat`). Implement a method `make_sound()` in both. Then, create a function that accepts any object and calls the `make_sound()` method, demonstrating duck typing.

2. **Task: Operator Overloading**

- Define a class `Vector` that represents a 2D vector. Overload the `+` and `*` operators to perform vector addition and scalar multiplication, respectively.

3. **Task: Method Overloading**

- Create a class `Calculator` with multiple `add()` methods that accept different numbers of arguments and types (integers, floats). Use default arguments or variable-length arguments to simulate method overloading in Python.

4. **Task: Method Overriding**

- Define a base class `Shape` with a method `area()`. Create two subclasses `Circle` and `Rectangle`, overriding the `area()` method to calculate the respective areas.

5. **Task: Constructor Overloading**

- In the class `Person`, simulate constructor overloading using `__init__()`. Handle multiple ways to instantiate the class (e.g., passing name and age, or only name).

6. **Task: Polymorphism with Inheritance**

- Create an abstract class `Animal` with a method `speak()`. Create subclasses `Cow`, `Dog`, and `Bird`, overriding the `speak()` method to make the appropriate sound. Demonstrate polymorphism by iterating over a list of `Animal` objects.

7. **Task: Mixins with Multiple Inheritance**

- Implement a class `LoggingMixin` that adds logging functionality. Create another class `DatabaseHandler` to manage database operations. Then, create a class `UserHandler` that inherits from both `LoggingMixin` and `DatabaseHandler`, adding user operations along with logging.

8. **Task: Overloading Comparison Operators**

- Create a class `Book` with attributes `title`, `author`, and `pages`. Overload comparison operators (>, <, ==) to compare `Book` objects based on the number of pages.

9. **Task: Method Overriding in Mixins**

- Create a base class `Vehicle` with a method `fuel_efficiency()`. Implement a mixin `HybridMixin` that overrides `fuel_efficiency()` to provide efficiency for hybrid vehicles. Create a `HybridCar` class that uses the mixin and the `Vehicle` class.

10. **Task: Dynamic Dispatch in Polymorphism**

- Implement a base class `Notification` with a method `send()`. Create subclasses `EmailNotification` and `SMSNotification` that override the `send()` method. Write a function that takes any notification object and calls `send()`, demonstrating dynamic dispatch.