OOP assessment Questions.

Class and Object Creation: Write a Python class named `Car` with attributes for `make`, `model`, and `year`. Add a method called `display_info()` that prints out the car's details. Create an instance of the `Car` class and call the `display_info()` method.

Encapsulation: Create a class named `BankAccount` with a private attribute `balance`. Add methods `deposit()` and `withdraw()` to adjust the balance, ensuring `balance` can't be directly accessed or modified outside the class. Write code to demonstrate depositing and withdrawing money while maintaining encapsulation.

Inheritance: Create a base class `Animal` with methods like `eat()` and `sleep()`. Then create a subclass `Dog` that inherits from `Animal` and adds a method `bark()`. Show how you can use both the inherited methods and the new method in an instance of `Dog`.

Polymorphism: Define two classes, `Cat` and `Dog`, each with a `make_sound()` method that prints a different sound specific to each animal. Write a function that takes an object and calls `make_sound()`, then demonstrate polymorphism by passing instances of both `Cat` and `Dog` to this function.

Constructor Overloading: Python does not directly support constructor overloading, but you can simulate it using default arguments. Write a class `Rectangle` with a constructor that can initialize the rectangle with either one (for a square) or two parameters (for a general rectangle). Add a method to calculate the area, and test it by creating squares and rectangles.

Method Overriding: Create a base class `Employee` with a method `calculate_salary()` that prints a generic message. Then create a subclass `Manager` that overrides `calculate_salary()` to provide a specific calculation for a manager's salary. Demonstrate the overridden behavior.

Composition: Define two classes, `Engine` and `Car`. The `Car` class should have an attribute `engine` that is an instance of the `Engine` class. Write methods for `Engine` (e.g., `start()` and `stop()`) and show how the `Car` class can use these methods to control the engine.

Static Methods and Attributes: Write a `Calculator` class with a static method `add()` that takes two numbers and returns their sum. Also, add a static attribute `count` to track the number of times the `add()` method has been called. Show how to use this static method and update `count`.

Operator Overloading: Define a class `Vector` that represents a vector in 2D space with `x` and `y` components. Overload the `+` operator to allow vector addition. Test this functionality by creating two `Vector` objects and adding them.

Abstract Classes: Use the `abc` module to create an abstract base class `Shape` with an abstract method `area()`. Then, implement subclasses `Circle` and `Square` that provide their own implementations of `area()`. Write code to create instances of `Circle` and `Square`, and call `area()` on each.

From this repo class oop3.py = https://github.com/josephbill/PythonWebDev

```
How can we utilize the objects created from the
class blueprint
employee , to get the company's total payroll in
class Payroll ?
```

Submission :

1. Submit Github repository to form on LMS
2. Repo's should have well documented commit messages