**Practice Questions: Object-Oriented Programming (OOP)**

1. **Basic Class and Object Creation (Classes & Objects):**
   * **Task:** Define a class named Dog. Give it a class attribute species set to "Canis familiaris". Define an \_\_init\_\_ method that takes name and age and assigns them to instance attributes.
   * **Action:** Create an instance of Dog named my\_dog with a name and age of your choice. Print both its name and its species.
2. **Instance Methods (Encapsulation):**
   * **Task:** Add a method bark(self) to the Dog class that prints the string "Woof! My name is [name]".
   * **Action:** Call the bark() method using the my\_dog object you created.
3. **Class Method and State:**
   * **Task:** Define a class named Car with a class attribute total\_cars = 0.
   * **Logic:** In the \_\_init\_\_ method, increment total\_cars every time a new Car object is created.
   * **Action:** Create three Car objects and then print the final value of the class attribute Car.total\_cars.
4. **Inheritance (The super() function):**
   * **Task:** Create a subclass named GoldenRetriever that inherits from Dog. Override the bark(self) method to print "Happy Woof!" instead of "Woof!".
   * **Action:** Create an instance of GoldenRetriever and call its bark() method.
5. **Encapsulation (Name Mangling):**
   * **Task:** Modify the Dog class. Change the age attribute to a **private attribute** by naming it \_\_age.
   * **Logic:** Create a public method get\_age(self) that returns the value of the private attribute.
   * **Action:** Try to access my\_dog.\_\_age directly (it should cause an error) and then access it correctly using my\_dog.get\_age().
6. **Polymorphism (Method Overriding):**
   * **Task:** Create a base class Shape with a method area(self) that prints "Area not calculated." Create a subclass Circle that overrides area(self) to print "Area of a Circle is πr2".
   * **Action:** Create an instance of both Shape and Circle and call the area() method on both.
7. **Special Method (\_\_str\_\_):**
   * **Task:** Implement the special method \_\_str\_\_(self) in the Car class (from Question 3) to return a user-friendly string description (e.g., "A Car object").
   * **Action:** Create a Car object and use the built-in print() function on the object directly.
8. **Property Decorator (Getter/Setter):**
   * **Task:** Define a class Temperature. Make the temperature value stored in a private attribute \_temp. Use the **@property decorator** to create a getter method temp.
   * **Logic:** Add a **setter method** (@temp.setter) that ensures the temperature is never set below −273.15 (absolute zero). If it is, raise a ValueError.
   * **Action:** Test the setter with a valid value and an invalid value.
9. **Multiple Inheritance:**
   * **Task:** Define two simple parent classes: Flyable (with method fly()) and Swimmable (with method swim()). Create a child class Duck that inherits from both.
   * **Action:** Create a Duck object and call both the fly() and swim() methods.
10. **Composition:**
    * **Task:** Define two classes: Engine (with a method start()) and Vehicle.
    * **Logic:** The Vehicle class should have an attribute that holds an **instance** of the Engine class (Composition). Give Vehicle a method drive() that calls the start() method of its internal Engine object.
    * **Action:** Create a Vehicle object and call its drive() method.