Assignment: Digital Pet Simulation Using OOP Concepts

Objective: ¶

This assignment is designed to test your understanding of classes, objects, and inheritance in Python. You will create a Digital Pet Simulation where users can interact with different types of pets. This simulation will involve creating a base class for a generic pet and derived classes for specific types of pets like dogs and cats.

Instructions:

1. **Create a Base Class: `Pet`**

   - Attributes:
     - `name` (string): The name of the pet.
     - `age` (integer): The age of the pet.
     - `hunger` (integer): A measure of how hungry the pet is (0-100 scale).
     - `energy` (integer): A measure of the pet's energy level (0-100 scale).
   - Methods:
     - `__init__(self, name, age)` : Initializes the `Pet` object with the given name and age. Set `hunger` to 50 and `energy` to 50.
     - `eat(self)` : Reduces the pet's hunger level by 10 and increases energy by 5. Print a message indicating the pet is eating.
     - `sleep(self)` : Increases the pet's energy level by 20. Print a message indicating the pet is sleeping.
     - `play(self)` : Increases the pet's hunger by 10 and decreases energy by 15. Print a message indicating the pet is playing.
     - `status(self)` : Print the current status of the pet (name, age, hunger, energy).
   - *Example Usage:**

     ```
     pet = Pet("Fluffy", 2)
     pet.status()  # Output: Name: Fluffy, Age: 2, Hunger: 50, E
     nergy: 50
     pet.eat()
     pet.play()
     ```

2. **Create Derived Classes: `Dog` and `Cat`**

   - **Dog Class**:
     - Inherit from `Pet` .
     - Additional Method:
       - `bark(self)` : Print a message indicating that the dog is barking.
   - *Example Usage:**

     ```
     dog = Dog("Buddy", 3)
     dog.status()
     dog.bark()
     dog.eat()
     ```

   - **Cat Class**:

    - Inherit from `Pet`.
    - Additional Method:
        - `meow(self)`: Print a message indicating that the cat is meowing.
- *Example Usage:**

```
cat = Cat("Whiskers", 4)
cat.status()
cat.meow()
cat.sleep()
```

3. **Create a Derived Class: `RobotDog`**

- **RobotDog Class**:
    - Inherit from `Dog`.
    - Override Method:
        - Override the `play` method to decrease the energy by only 5 (since it's a robot).
    - Additional Method:
        - `recharge(self)`: Set energy to 100 and print a message indicating that the robot dog is recharging.
- *Example Usage:**

```
robot_dog = RobotDog("RoboBuddy", 1)
robot_dog.status()
robot_dog.bark()
robot_dog.play()
robot_dog.recharge()
```

4. **Testing Your Classes**

- Create at least one instance of each class (`Pet`, `Dog`, `Cat`, `RobotDog`).
- Call the `status` method on each instance to check their initial state.
- Interact with each pet by calling their methods (`eat`, `sleep`, `play`, `bark`, `meow`, `recharge`).
- Ensure that the overridden and unique methods behave as expected.
- *Example Test Code:**

```
pet = Pet("Fluffy", 2)
pet.status()
pet.eat()
pet.play()

dog = Dog("Buddy", 3)
dog.status()
dog.bark()
dog.eat()

cat = Cat("Whiskers", 4)
cat.status()
cat.meow()
cat.sleep()

robot_dog = RobotDog("RoboBuddy", 1)
robot_dog.status()
robot_dog.bark()
robot_dog.play()
robot_dog.recharge()
```

Expected Output:

```
Name: Fluffy, Age: 2, Hunger: 50, Energy: 50
Fluffy is eating. Hunger: 40, Energy: 55
Fluffy is playing. Hunger: 50, Energy: 40

Name: Buddy, Age: 3, Hunger: 50, Energy: 50
Buddy is barking!
Buddy is eating. Hunger: 40, Energy: 55

Name: Whiskers, Age: 4, Hunger: 50, Energy: 50
Whiskers is meowing!
Whiskers is sleeping. Energy: 70

Name: RoboBuddy, Age: 1, Hunger: 50, Energy: 50
RoboBuddy is barking!
RoboBuddy is playing (robot mode). Hunger: 55, Energy: 45
RoboBuddy is recharging. Energy: 100
```

```

5. **Submission:**

   - Submit the Python script containing your implementation of the `Pet`, `Dog`, `Cat`, and `RobotDog` classes.
   - Ensure that your script includes the test code at the end, demonstrating the functionality of each class.

**Bonus Challenge (Optional):**

- Implement a class called `Bird` that inherits from `Pet`. Add a unique method `sing(self)` that prints a message indicating the bird is singing. Override the `play`
```