Exercise

1

Create a `GoldenRetriever` class that inherits from the `Dog` class. Give the sound argument of `GoldenRetriever.speak()` a default value of "Bark".

Use the following code for your parent `Dog` class:

In [1]:
```python
class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

In [2]:
```python
# Your code here:
```

2

Consider the class `Shape` with a `__init__` method that takes in a `color` attribute and sets it as an instance variable.

In [2]:
```python
class Shape:
    def __init__(self, color):
        self.color = color

    def area(self):
        print("Area not implemented")
```

Define two subclasses `Rectangle` and `Circle` that inherit from `Shape`.

- The `Rectangle` class should have a `__init__` method that calls the **superclass's `__init__` method** and sets additional attributes `width` and `height`. It should also override the area method to return the area of the rectangle.

- The Circle class should have a `__init__` method that calls the **superclass's `__init__` method** and sets an additional attribute `radius`. It should also override the area method to return the area of the circle.

In [3]:
```
# Your code here:
```

3

Define a base class `Person` with a `__init__` method that takes in `name`, `age`, and `gender` attributes and sets them as instance variables. Also define a method `introduce` that prints a message introducing the person.

In [4]:
```
class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def introduce(self):
        print(f"Hi, my name is {self.name}")
```

Then, define two subclasses `Student` and `Instructor` that inherit from ``Person.

The `Student` class should have a `__init__` method that calls the superclass's `__init__` method and sets an additional attribute major. It should also override the `introduce` method to print a message introducing the student and mentioning their major.

The `Instructor` class should have a `__init__` method that calls the superclass's `__init__` method and sets an additional attribute courses_taught. It should also override the `introduce` method to print a message introducing the instructor and mentioning the courses they teach.

In [5]:
```
# Your code here
```

You can test your solution by creating instances of the `Student` and `Instructor` classes and calling the `introduce` method on them. For example:

```
student = Student("John", 20, "Male", "Mathematics")
student.introduce() # Output: "Hi, my name is John. I am majoring
in Mathematics."

instructor = Instructor("Jane", 30, "Female", ["MA101", "MA102"])
instructor.introduce() # Output: "Hi, my name is Jane. I teach the
following courses: MA101, MA102."
```

In [ ]:

4

Define a base class `Character` with a `__init__` method that takes in `name`, `health`, and `power` attributes and sets them as instance variables. Also define a method `attack` that takes in another character and reduces their health by the attacking character's power.

In [6]:
```python
class Character:
    def __init__(self, name, health, power):
        self.name = name
        self.health = health
        self.power = power

    def attack(self, other_character):
        other_character.health -= self.power
```

Then, define two subclasses `Hero` and `Villain` that inherit from `Character`.

The `Hero` class should have a `__init__` method that calls the superclass's `__init__` method and sets an additional attribute `armor`. It should also override the `attack` method to take into account the hero's `armor`. If the hero has armor, reduce the damage taken by the value of the armor. If the hero does not have armor, the attack should work as it does in the base class.

The `Villain` class should have a `__init__` method that calls the superclass's `__init__` method and sets an additional attribute `evil_plan`. It should also override the `attack` method to print a message saying that the villain is executing their evil plan.

In [ ]:

You can test your solution by creating instances of the `Hero` and `Villain` classes and calling the `attack` method on them. For example:

```python
hero1 = Hero("Superman", 100, 30, 10)
villain1 = Villain("Lex Luthor", 100, 30, "Take over the world")

hero1.attack(villain1)
print(villain1.health) # Output: Health 70

villain1.attack(hero1)
print(hero1.health) # Output: Health 80
```

In [ ]:
