Homework 2

Due: Sunday, September 15 @ 11:59 pm

Deliverables:

1. Canvas assignment (GitHub account link)
2. Canvas assignment (Homework 2: Inheritance Practice)

Part 0: Set up VS Code

I recommend using the Visual Studio (VS) Code integrated development environment (IDE) in this class, though you are welcome to use others if you'd like (e.g., pycharm, vim).

Note: If you choose to use something other than VS Code, I may not be able to provide as much technical support.

1. Check whether Visual Studio (VS) Code is installed on the computer you plan to develop on (it *should* already be installed on the lab computers).

    a. If it is not already installed, you can install it from the VS Code website: https://code.visualstudio.com/download

Part 1: Set up GitHub

We will be using GitHub for version control in this class.

1. Create a GitHub account (https://github.com) if you do not have one already.

2. Submit the URL **to your user profile** to Canvas, e.g., https://github.com/akubota95

Part 2: Inheritance Practice

In this assignment, we will gain familiarity with some major concepts of object-oriented programming (OOP) including inheritance and overloading.

1. Get the starter code.
    a. Download the inheritance_practice.py file from Canvas and open it in VS Code.
2. Familiarize yourself with the code in this file (do not run it yet) and answer the following questions in Canvas.
    a. **What is the name of the class defined in this file?**
    b. **What attributes does this class have? If any, what do they represent?**
    c. **What methods does this class have? If any, what do they do?**
    d. **Is `run()` a method of the class? How can you tell?**
    e. **What does the `run()` function do? What, if anything, do you expect to be printed to the terminal when the script is run?**
3. Run the script and confirm it executes without error.
    a. **What was printed to the terminal when you ran the script?**
    b. **Was this what you expected to be printed?**
4. Update the program header with the names of the people you are working with.

Complete the Animal class:

5. Update the Animal class to include the following properties:
   (Helpful coding tip! Rather than making all the changes below AND THEN testing everything at once, it is better practice to test as you write. For example, add `name` and write some code to make sure it works as you expect. This way, debugging becomes easier since you know that any issues were a result of the code you added for `name`.)
    a. Attributes (don't forget `self`!):
        i. `name`: A string storing the name of the given animal (passed in as an argument to the constructor)
        ii. `hungry`: A Boolean representing whether the given animal is hungry (default initial value = False)

iii. Another attribute of your choice. Be sure to add comments explaining what this attribute represents and its initial value.

b. Methods:

i. `sleep`: Make the animal sleep.

1. Arguments: None

2. Output:

a. Print "{name} slept" to the terminal, with "{name}" replaced with the name of the animal.

b. Change the value of `hungry` to True.

c. Do not return any value.

ii. `eat`: Make the animal eat.

1. Argument: `food`: a string stating the food eaten. For example, `an_animal.eat("carrot")` would mean an_animal ate a carrot.

2. Output:

a. If the animal is hungry (i.e., `hungry` is True):

i. Print "{name} ate {food}!" to the terminal, with "{name}" replaced with the name of the animal, and "{food}" replaced with the name of the food eaten.

ii. Set `hungry` to False.

iii. Return True.

b. If the animal is not hungry (i.e., `hungry` is False):

i. Print "{name} did not eat {food}…" to the terminal, with the same replacements as above.

ii. Return False.

iii. Another method of your choice that uses the attribute you created (e.g., prints, checks, or changes its value). Be sure to add comments describing your method's behavior, any arguments, and outputs.

6. Test your code to make sure everything works as you expect. In addition to any other tests you might write, be sure to also do the following:

    a. Add the following lines to the end of `run()` in this order:

        i.
```
an_animal.sleep()
an_animal.eat("carrot")
```

    b. Execute your script and **submit a screenshot of the output to Canvas.**

    c. Flip the order of the lines you added so an_animal eats before moving.

    d. Execute your script and **submit a screenshot of the output to Canvas.**

    e. **What changed between the two outputs? Why?**

Create a Bird child class (subclass) that inherits from the Animal class.

7. Add the following code to your script between the Animal class and the `run()` function (watch out for indentation):

```
class Bird(Animal):
    def fly(self):
        print("I can fly!")
```

8. Add the following lines to the end of run() in this order:

    a.
```
a_bird = Bird("Big Bird")
a_bird.fly()
```

9. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the above? Why?**

10. Execute your script and **submit a screenshot of the output to Canvas.**

Inheriting parent attributes and methods.

11. Now add the following line to the end of `run()`:
    a. `a_bird.sleep()`
12. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the above? Why?**
13. Execute your script and **submit a screenshot of the output to Canvas.**
14. The Bird class does not have a `sleep()` method defined. **Which OOP concept is the reason why the above line ran without error?**

Inheritance is uni-directional

15. Now add the following line to the end of `run()`:
    a. `an_animal.fly()`
16. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the above? Why?**
17. Execute your script and **submit a screenshot of the output to Canvas.**
18. **Why did this line throw an error, while `a_bird.sleep()` ran without issue?**

Polymorphism to overload methods

19. Add the following method to the Bird class:

```
def speak(self):
    print("chirp chirp chirp”)
```

20. Add the following line to the end of `run()`:
    a. `a_bird.speak()`
21. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the above? Why?**

22. Execute your script and **submit a screenshot of the output to Canvas.**

23. **Which definition of `speak()` did this line execute? Which OOP concept is the reason for this?**

Adding attributes to subclasses

24. As you may have noticed, Bird doesn't have its own constructor! **Why doesn't it need one? I.e., what is run when a new Bird is initialized? Why?**

25. Let's say we want to add the new attribute nest_location to the Bird class so our birds can keep track of their eggs!

    Now we will need to write a constructor for Bird to define this new attribute. Add the to your Bird class:

    ```
    def __init__(self, name):
        self.nest_location = None
    ```

26. Now let's test it out. Add the following to the end of run():

    ```
    print(a_bird.nest_location)
    ```

27. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the above? Why?**

28. Execute your script and **submit a screenshot of the output to Canvas.**

29. **Which line threw and error? Why do you think this happened?**

`super()`

30. To fix the previous error, we need to run the super()function in the Bird constructor. **What does the super() function do?**

31. Add the following line to your Bird constructor:

    ```
    super().__init__(name)
    ```

32. The above line runs the constructor for Bird's parent class. **Given this information, why do we pass `name` as an argument here? What would happen if we instead replaced `name` with `"Hello"`?** (Hint: If you are unsure, try it out and see what changes!)

33. Execute your script and **submit a screenshot of the output to Canvas.**

34. **Explain in your own words why the addition of the super() line was necessary.**

Protected Members

35. Sometimes we might have attributes or methods that we don't want to be accessible outside of the class or subclasses. In general, it is good practice to restrict access to attributes and methods. **In your own words, explain why restricting access (i.e., making attributes or methods private or protected) is useful**.

36. For instance, perhaps we have sensitive health data that we only want to be accessible from within the class or any subclasses. Add the following line to the Animal constructor (note there is only one underscore, indicating that the attribute is *protected*):

```
self._healthy = True
```

37. Update `run()` function as follows:

```
def run():
    a_bird = Bird("Big Bird")
    print(a_bird._healthy)
```

38. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by the run() function now? Why?**

39. Execute your script and **submit a screenshot of the output to Canvas.**

40. Surprise! In other languages (C++, Java), the protected status means the attribute would not be accessible outside of the class or its subclasses (e.g., in the `run()` function). In Python, programmers are meant to just follow the *convention* of not accessing these attributes outside of the classes. Python does nothing to enforce this though. Why? I haven't a clue.

Accessing private members outside of a class

41. So, let's make the attribute *private* instead. Add a second underscore before the attribute name in both the Animal constructor and in `run()`.

42. Before executing your script, answer in Canvas: **What, if anything, do you expect to be output by executing the script this time? Why?**

43. Execute your script and **submit a screenshot of the output to Canvas.**

44. **How and why was the result different this time?**

Accessing private members inside of a class?

45. Okay, so where *can* we access the __*healthy* attribute now that it is private? Here is a testing plan we can follow:

    a. Try accessing it within the Animal class

    b. Try accessing it within the Bird class

    c. Write a public "getter" method in the Animal class for the __healthy attribute and try calling it from outside the Animal class

    d. Write a public "getter" method in the Bird class for the __healthy attribute and try calling it from outside the Bird class

46. For each of the above:

    a. **Explain the changes you would make to your code to accomplish the test.**

    b. **Make those changes. Submit a screenshot of any methods you updated, making sure it is clear what class it is from.**

    c. Execute your script and **submit a screenshot of the output to Canvas**.

    d. **Explain why you got the result that you did**.

47. In Canvas, answer the following:

    a. **Based on your tests, where are private data members directly accessible?**

b. **How might programmers strategically use (or prevent the use of) setters to limit who can modify the value of private data members?**

Hierarchical inheritances

48. Write a new class Fish which inherits from the Animal class. Add the following:
    a. Attribute:
        i. `name`: A string representing the fish's name. Turns out fish follow the naming convention "<name>fish", or their given name with "fish" appended to the end. Overload Animal's name attribute to set Fish's name to this value.
    b. Method:
        i. `swim`: Make the fish swim.
            1. Arguments: None
            2. Output:
                a. Print "Just keep swimming" to the terminal.
                b. Do not return any value.
49. Let's test the following:
    a. Print the name attribute of a Fish object.
    b. Make a Fish swim.
    c. Make a Fish fly.
    d. Make a Fish speak.
50. For each of the above:
    a. **Explain the changes you would make to your code to accomplish the test.**
    b. **Make those changes. Submit a screenshot of any methods you updated, making sure it is clear what class they are from.**

c. Execute your script and **submit a screenshot of the output to Canvas**.

d. **Explain why you got the result that you did**.

51. In Canvas, answer the following:

a. **Summarize your findings regarding the relationship between Fish and Animal.**

b. **Summarize your findings regarding the relationship between Fish and Bird.**

c. **What are three benefits of using inheritance?**

Wrapping up (Last section!)

52. Make sure all your code is well commented and your program header is up to date.

53. **Submit your final inheritance_practice.py file to Canvas**.