Introduction to Computer Science I
MCS 177 - Spring 2020
Project 10: Tamagotchi
Due 5/20

Overview

For this project, you will write the implementation for a variation of Tamagotchi pets in Python. A Tamagotchi is a virtual pet, which has various needs to be attended to. Tamagotchis were very popular when Prof. Lynn was growing up, though she wasn't very good at taking care of hers.
If you aren't familiar with Tamagotchis, there are various YouTube videos about them, here is a link to one:

`https://www.youtube.com/watch?v=cYRABCUoEB0`

Our version of Tamagotchis won't exactly match the original, but will share some of the characteristics. In particular, we will be able to do the following with our pets.

- Feed the pet, in order to keep it from going hungry.
- Bathe the pet, when it gets dirty.
- Play with the pet, to increase its happiness.
- As the pet is successfully cared for, it will grow from an egg, to a baby, to a child, to an adult.
- If the pet is consistently neglected, it will die.

In order to implement our Tamagotchi as a well-organized program, we will use classes and the principles of object-oriented design.

This project is due on Wednesday, May 20th, at 11pm.
Submissions more than a day late for this project will not be accepted.

Part 1: Programming a Pet

For this part of the project, you'll write the implementation for the `Pet` class, which will control all interactions with the Tamagotchi pet. This should be written in a file called `my_tamagotchi.py`. As you write the implementation for this class, **be sure to test your functions as you write them**. If you wait until all member functions are written before you test them, you're likely to produce lots of errors that will be difficult to track down. If you pay extra attention to the instructions and follow them exactly, this will save you a lot of time in Part 2 of the project. Suggestions for how to test your functions are given after the description of the `Pet` class.

The `Pet` class will need the following:

Member variables:

- `name`, the name of the pet.
- `fullness`, an integer between 1 and 10 indicating how well fed the pet is.
- `happiness`, an integer between 1 and 10 indicating how happy the pet is.
- `cleanliness`, an integer between 1 and 10 indicating how clean the pet is.
- `alive`, a boolean value indicating if the pet is alive.
- `stage`, a string giving the life stage of the pet ("egg", "baby", "child", "adult")
- `progress`, an integer between 1 and 20 tracking how close the pet is to progressing to the next stage.

Member functions:

- The constructor, takes a name for a pet, and creates a pet with the given name. `fullness`, `happiness`, `cleanliness` are all initialized to 8. `alive` is initialized to `True`, `stage` is initialized to "egg", and `progress` is initialized to 1.
- `feed`, adds 3 to the pet's `fullness`, up to a maximum of 10. If the pet's fullness is already 10 when this function is called, the pet spills food on itself, and its `cleanliness` drops by 2 (to a minimum of 1).
- `play`, adds 3 to the pet's `happiness`, up to a maximum of 10. If the pet's happiness is already 10 when this function is called, the pet gets hungrier from exercise, and its `fullness` drops by 2 (to a minimum of 1).
- `bathe`, adds 3 to the pet's `cleanliness`, up to a maximum of 10. If the pet's cleanliness is already 10 when this function is called, the pet gets annoyed by unnecessary bathing, and its `happiness` drops by 2 (to a minimum of 1).
- `age_up`, changes the value of `stage` to the next stage, and resets `progress` to 1.
- `status`, if `fullness`, `happiness`, and `cleanliness` are all greater than 5, returns the string "fine". If `fullness`, `happiness`, or `cleanliness` is 1, returns the string "dead" and changes the value of `alive` to `False`. If `fullness`, `happiness`, or `cleanliness` is less than or equal to 5 (but the pet isn't dead), returns the string "distress".
- `time_step`, randomly chooses one of `fullness`, `happiness`, and `cleanliness` to decrease by 1. Increases progress by 1. After the increase, if the progress is 20, calls the function `age_up`. Calls the function `status`, and returns the string returned by `status`.
  To randomly choose between `fullness`, `happiness`, and `cleanliness`, you'll need to use a function from the `random` module. The Python module `random` contains random number generators and other functions that produce random behavior, and we'll make use of the function `choice`. The function `choice` takes a list of possibilities, and randomly returns one of them. For example, the following code

prints a random integer between 1 and 3, inclusive.

```
>>> import random
>>> print(random.choice([1,2,3]))
```

Use `choice` to choose a random string among “fullness”, “happiness”, and “cleanliness”. If that random string is “fullness”, decrease `fullness`. If the random string is “happiness”, decrease `happiness`. If the random string is “cleanliness”, decrease `cleanliness`. Be careful that you don’t mix up strings and function names here - that’s an easy mistake to make!

To test your functions, it’s helpful to check the values of member variables before and after calling the functions. For example, to check that `feed` is adding to the fullness of the pet, you might try:

```
>>> my_pet = Pet(‘‘Mike’)
>>> my_pet.fullness
8
>>> my_pet.feed()
>>> my_pet.fullness
11
```

This would tell you that your `feed` function is increasing the fullness, but it’s going beyond the maximum value for fullness, which would need to be fixed. Think about how you can use various function calls to fully test your functions. This might include manually changing the values of member variables. For instance, you might try something like:

```
>>> my_pet = Pet(‘‘Mike’)
>>> my_pet.fullness = 10
>>> my_pet.cleanliness
8
>>> my_pet.feed()
>>> my_pet.cleanliness
8
```

This would tell you that your `feed` function isn’t decreasing the cleanliness of the pet when the fullness is already maxed out, which would need to be fixed.

Part 2: Playing the Game

You’ve written code for the `Pet` class, but it would be a lot more fun if there was a nice interface to interact with your pet! In this part, you’ll test out your `Pet` class using a class that Prof. Lynn wrote to display the Tamagotchi game. Prof. Lynn will post a video demonstrating how the game should run, if everything is done correctly.

Download the file `Tamagotchi.py` from moodle, and copy and paste the code from this file into your file `my_tamagotchi.py`. You are not allowed to make changes to this code, it must be copied into your file exactly. Then run your file, and try running the following commands in the shell:

```
>>> game = TamagotchiGame(‘‘George’’)
>>> game.run()
```

These commands create an instance of the class `TamagotchiGame`, with the name “George” for the pet. Then, they run the game. If you have implemented your `Pet` following the instructions above, these commands should launch a turtle window, and run the Tamagotchi game.

If your Tamagotchi game does not run, this means that you have an error in your `Pet` class, and you need to go back and check your implementation of the `Pet` class. Even if the Tamagotchi game seems to run correctly, it is still possible that you did not implement your `Pet` class according to the instructions, so it's very important that you thoroughly test your implementation from Part 1.

Part 3: One Last Function

Once you've verified that your `Pet` class is implemented correctly, and works with the `TamagotchiGame` class, you have one final task: writing a function that runs the Tamagotchi game. This function will get the name for a Tamagotchi pet from the user, and then create and run a Tamagotchi game for a pet with the given name.

To do this, we need to get user input, which we haven't done yet. Fortunately, this is pretty simple in Python - we just need to call the function `input`. For example, the following function prints a message asking a user for their name, then prints a message greeting the user.

```
# void -> void
def greet_user():
    print(‘‘What is your name?’’)
    name = input()
    print(‘‘Hi, ’’ + name + ‘‘, nice to meet you!’’)
```

The function `input` will wait for the user to type in a response and hit the return key, then `input` returns the string given as the response. For example, if I make the function call `greet_user()`, then type in “Melissa” and hit the return key, it will print “Hi, Melissa, nice to meet you!”. Test this function out before starting the implementation of `play_tamagotchi`, described below.

Now that you know how to get user input, we're ready to implement the function `play_tamagotchi`. This function will be relatively short, it just needs to do each of the following:

- Print the prompt, “What would you like to name your Tamagotchi?”
- Get the name as an input from the user.
- Create an instance of the class `TamagotchiGame` with the given name.
- Call the member function `run` for the instance of `TamagotchiGame`.

Extra Credit: Modifying `TamagotchiGame`

Before you start the extra credit, make a copy of your file `my_tamagotchi.py`, and call this copy `extra_credit.py`. The extra credit modifications must be done in a separate file from your main submission, and doing otherwise could result in losing points on the main part of the project.

Once you've copied your project, there are two ways you can earn extra credit on this project:

- **Adding Documentation.** Prof. Lynn did a terrible job of documenting `TamagotchiGame`, she hardly included any contracts or docstrings! If you add correct contracts and docstrings to the functions in this class, you'll earn 2 points of extra credit.
- **Improving the Graphics.** Prof. Lynn did a mediocre job of creating graphics for the TamagotchiGame, and we think that you can do much better! First, read through the code in TamagotchiGame to figure out how it works. Then, modify the functions to make the Tamagotchi game more visually appealing. This could include changing the appearance of the pet at the various stages, changing the animations for feeding, playing with, and bathing the pet, or anything else you can think of. Depending on how extensive your changes are, this can be worth up to 5 points of extra credit. However, if the game doesn't run properly because of your changes, you will not earn any extra credit points.

Submitting your work

For this project, you will need to submit the following:

- `my_tamagotchi.py`, which is the python file containing your implementation of the `Pet` class and `play_tamagotchi` function, along with the code copied from the file `Tamagotchi.py`.
- (optional) `extra_credit.py`, a Python file containing your customized Tamagotchi program.

You will be submitting your files using Moodle.

Grading

You will earn one point for each of the following accomplishments:

- (2 points) All of the functions have correct contracts and docstrings.
- You have written a constructor for the `Pet` class.
- Your constructor initializes member variables exactly as described in the instructions.
- You have written member functions `feed`, `play`, and `bathe` for the `Pet` class.

- Your functions `feed/play/bathe` increase the pet's fullness/happiness/cleanliness by 3.
- If this increase would make the fullness greater than 10, the fullness/happiness/cleanliness is set to 10.
- If the fullness/happiness/cleanliness is already 10, your functions `feed/play/bathe` correctly reduce the pet's cleanliness/fullness/happiness, to a minimum cleanliness/fullness/happiness of 1.
- You have written a member function `age_up` for the `Pet` class.
- Your function `age_up` correctly updates the stage of the pet, and resets the progress to 1.
- You have written a member function `status` for the `Pet` class.
- Your function `status` correctly determines if the pet is dead, in distress, or fine, and returns the string to report this, exactly as instructed.
- You have written a member function `time_step` for the `Pet` class.
- Your function `time_step` correctly uses a randomness to select which attribute to decrement.
- Your function `time_step` correctly updates the progress, ages up if necessary, and calls and returns the status.
- You have written a function `play_tamagotchi`.
- Your function `play_tamagotchi` correctly prompts and gets input from the user.
- Your function `play_tamagotchi` correctly creates an instance of `TamagotchiGame` with the given pet name, and runs the game.
- (2 points) Your code is well-organized, easy to understand, and not unnecessarily complicated. Any particularly tricky parts are explained with comments.
- (2 points, extra credit) In the file `extra_credit.py` (separate from your file `my_tamagotchi.py`), you have added contracts and docstrings to `TamagotchiGame`.
- (up to 5 points, extra credit) In the file `extra_credit.py` (separate from your file `my_tamagotchi.py`), you have changed the Tamagotchi artwork.