Python Project Virtual Pet

Assignment Overview

- Classes

Assignment Background

In this project, you will write a Python program to *implement a virtual pet*. You can decide the species, name, and gender of your pet. You can also interact with your pet by feeding your pet, playing with your pet, bathing your pet, etc. You will design a user-defined Class called 'Pet' to achieve that.

A pet has the following basic attributes:

- Species (we assume two species: dog and cat)
- Name
- Gender (we assume two genders: male and female)
- Fur color

As well as some other attributes that are related to physical or mental health status:

- Energy
- Hunger
- Thirst
- Loneliness
- Smell

These attributes are measured in the range from 0 to 10. For the 'energy' attribute, the ideal value is 10. For the other attributes ('hunger', 'thirst', 'loneliness', and 'smell'), the ideal value is 0. When a pet object is created, his or her initial status is randomized within an acceptable range. For example, a newly created pet may have a hunger/thirst/loneliness/smell level in the range of (0, 5), and have an energy level in the range of (5, 10).

Therefore, in your pet class, you will have attributes related to each feature that we mentioned. Make them as private attributes (i.e., the attribute name should start with a single underscore, like `self._name`)

The goal of this game is to take care of your pet by keeping your pet in a good status (i.e., lower hunger/thirst/loneliness/smell level, and higher energy level). You can take care of your pet by taking each of the following actions:

- Let your pet get some sleep
- Feed your pet some food
- Feed your pet some water

- Play with your pet
- Bathe your pet.

The health status of your pet will be updated each time you take a certain action:

- After you feed your pet food, his/her hunger level will be decreased.
- After you feed your pet water, his/her thirst level will be decreased.
- After you play with your pet, the loneliness level will be decreased, but the hunger level, thirst level, and smell level will be increased.
- After you bathe your pet, the smell level will be decreased.
- After your pet gets some sleep, the energy level will be increased.

After you take an action, your pet will give you feedback about the specific action you took. Next, your pet will report his/her status if some attributes are too low or too high. For example, after you play with your pet, your pet will first say 'Thanks for your company'. Next, if the hunger level of your pet is lower than a threshold (say 5), your pet will say 'I am hungry'. Please note that multiple attributes may need to be reported after an action. If your pet is both hungry and thirsty after playing with you, he/she will say two sentences: 'I am hungry,' and then 'I am thirsty'.

The game begins by prompting you to entering the species, name, gender, and fur color of your pet, each feature separated by a space. Once those inputs are validated, a pet object will be created. Then the program will prompt you to enter a command. Your command will either lead to an action taken or lead to a table that shows you the health status of your pet. The program will repeatedly prompt you to input a command, until you enter 'q' and then you will quit this game.

Project Specifications

You must implement the following classes and methods. *You are not allowed to add methods and you are only allowed to access attributes using the provided methods (that is no "name mangling" allowed).*

1. Design a class called **Pet** which initialize the following attributes in its `__init__()` function:

```
self._name  # string

self._species # string

self._gender # string

self._color # string

self._hunger # float

self._thirst # float

self._smell # float

self._loneliness # float
```

self._energy # float

self._edible_items # list

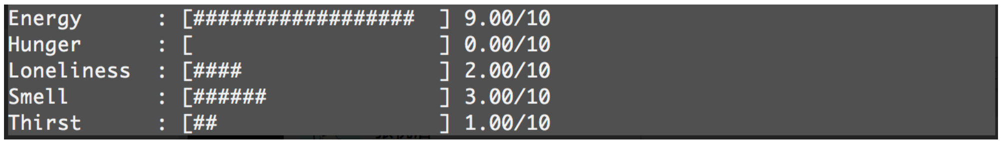
self._drinkable_items # list

- We provide a skeleton of the **Pet** class with all attributes in the `__init__` method initialized to `None`. You need to fill in appropriate values for initialization:
  - o The default values for `name`, `species`, `gender`, and `color` are `'Fluffy'`, `'Dog'`, `'Male'`, and `'White'` respectively. Force the attribute values to be capitalized (**hint**: use the *capitalize*`()` string method).
  - o The initial value for `hunger`, `thirst`, `smell`, and `loneliness` are a random integer number in the range of [0, 5], while the initial value for `energy` is a random integer number in the range of [5, 10]. (**hint**: import the **cse231_random** module and then use the ***cse231_randint()*** function).
  - o The initial value for *_edible_items* and *_drinkable_items* is an empty list. (We will append some items to these lists later on, so that *_edible_items* and *_drinkable_items* will be a list of different class types, and each class type is provided in the *edible.py* file. An example of a list of class types is given here: *a_list = [str, int, float]*)
- We provide a complete method called *_reply_to_master()*. After you initialize the attributes mentioned above, there is a call to the *_reply_to_master* method.

2. Complete the following class methods:
   a. *def get_hunger_level(self):* simply return the value of *self._hunger*
   b. *def get_thirst_level(self):* similar to previous.
   c. *def get_energy_level(self):* similar to previous.
   d. *def drink(self, liquid):* this method represents the action of 'feeding water'.
      i. this method takes an input parameter called `liquid`, which is an instance that belongs to a certain class type. This class type should be an element in the *self._drinkable_items* list. (For example, if liquid = 1 (or any other integer), then `int` must be an element in the *self._drinkable_items* list.) Otherwise, your program will output an error message. (**Hint**: use **isinstance**() function).
      ii. Call the *_time_pass_by* method with the default time (which is complete and does not need modification).
      ~~iii.~~ If the liquid is valid, you will decrease the *thirsty* value of your pet, based on the quantity of this liquid. (**hint**: read the *edible.py* file and figure out how to get the quantity attribute of the liquid). You must make sure that your pet's thirst level will not go below zero. For example, if the *_thirst* value is 3, and the quantity of the liquid is 2, then after you call the drink function, *_thirst* will be 1. If the quantity of the liquid is 4, then *_thirst* will become 0, not -1.~~ If all the liquid is not drunk, print "Too much drink to finish. I will leave some for you."~~
      iv. A pet will not drink if their thirst is less than 2. If they will not drink, print a message, e.g. "`Your pet is satisfied, no desire for sustenance now.`" if they are not hungry.
      v. If the liquid is invalid, print "`Not drinkable`"

   vi. After you update the *_thirst* value, call the *_reply_to_master()* method (which is also complete, but you need to figure out the input argument to this method).

e. *def feed(self, food):* the logic of this method is very similar to drink. The difference is that in this method, you will update the value of *_hunger.* You will also call the *_reply_to_master()* method with a different input argument than used with drink.
f. *def shower(self):* (provided)
   i. call the *_time_pass_by()* method. The shower takes a time=4. (If you wanted to create a more fun game, you could import random, assign a random time and use the cse231_random.randint(3,5) function. However, such a game program won't pass Mimir tests!).
   ii. this method changes two attributes of your pet: the *_smell* and the *_loneliness* . The smell level will always drop to zero. The loneliness level of your pet will be decreased depending on how long the shower takes. For example, if the time = 2 and the *loneliness* level was 3, then after the shower the loneliness level will drop to 1. Please note that the loneliness level cannot go below zero. Therefore, if time = 4 and loneliness level was 3, then it will drop to 0, instead of -1.
   iii. After those two attributes are updated, call the *reply_to_master()* method and then the *update_status()* method.
g. *def sleep(self):*
   i. Similar to the *shower* method*,* but sleeping takes a time=7, and the attribute modified by this method is *self._energy* which will increase. Please note that the energy level will not exceed MAX (10). Therefore, if time = 7 and the loneliness level was 4, then loneliness will rise to 10, instead of 11.
h. *def play_with(self):*
   i. Similar to the *shower* method*,* but *play_with* takes a time=4, and this method changes three attributes of your pet: *self._loneliness, self._smell,* and *self._energy.* The loneliness and energy level will decrease, and the smell level will increase, depending the time you have spent playing with your pet. Please note that all the attribute values will not exceed the range of MIN (0) and MAX (10).
i. *def show_status(self):*
   i. this method will display the hunger, thirst, smell, loneliness, and energy attributes of your pet as a table. Attribute values will be sorted in alphabetical order. An example is given below:



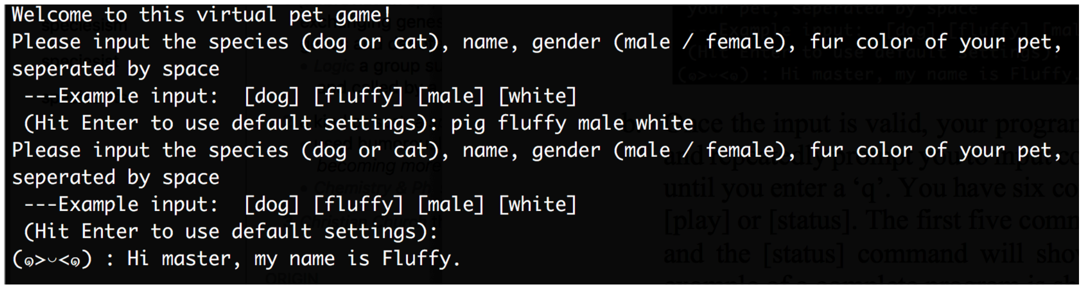
```
Energy      : [##################  ] 9.00/10
Hunger      : [                    ] 0.00/10
Loneliness  : [####                ] 2.00/10
Smell       : [######              ] 3.00/10
Thirst      : [##                  ] 1.00/10
```

The number of '#' is proportional to the attribute value, and it will occupy 20 spaces in total. A partially formatted string is in the provided file.

3. After you complete the *`Pet class`*, design two subclasses of Pet:
    a. `Class` **`Cat`**: this class inherits the **Pet** class. The `init` method of this class will inherit most of the features from the Pet class (by calling Pet's `__init__` method which is also a good place to set the species to `'cat'`). Besides that, the cat class will update two attributes (all within `__init__`):
        i. *`self._edible_items:`* it will be a list that contains elements from the *`'cat_edible_items'`* list (which is at the beginning of the template code.)
        ii. *`self._drinkable_items:`* it will be a list that contains elements from the *`'cat_drinkable_items'`*.
    b. *`Class`* ***`Dog`***: this class inherits the **Pet** class. The *`init`* function of this class will inherit most of the features from the Pet class (using Pet's `__init__` method). Besides that, the cat class will update two attributes (all within `__init__`):
        i. *`self._edible_items`*: it will be a list that contains elements from the *`'dog_edible_items'`* list (which is at the beginning of the template code.)
        ii. *`self._drinkable_items:`* it will be a list that contains elements from the *`'dog_drinkable_items'`*.

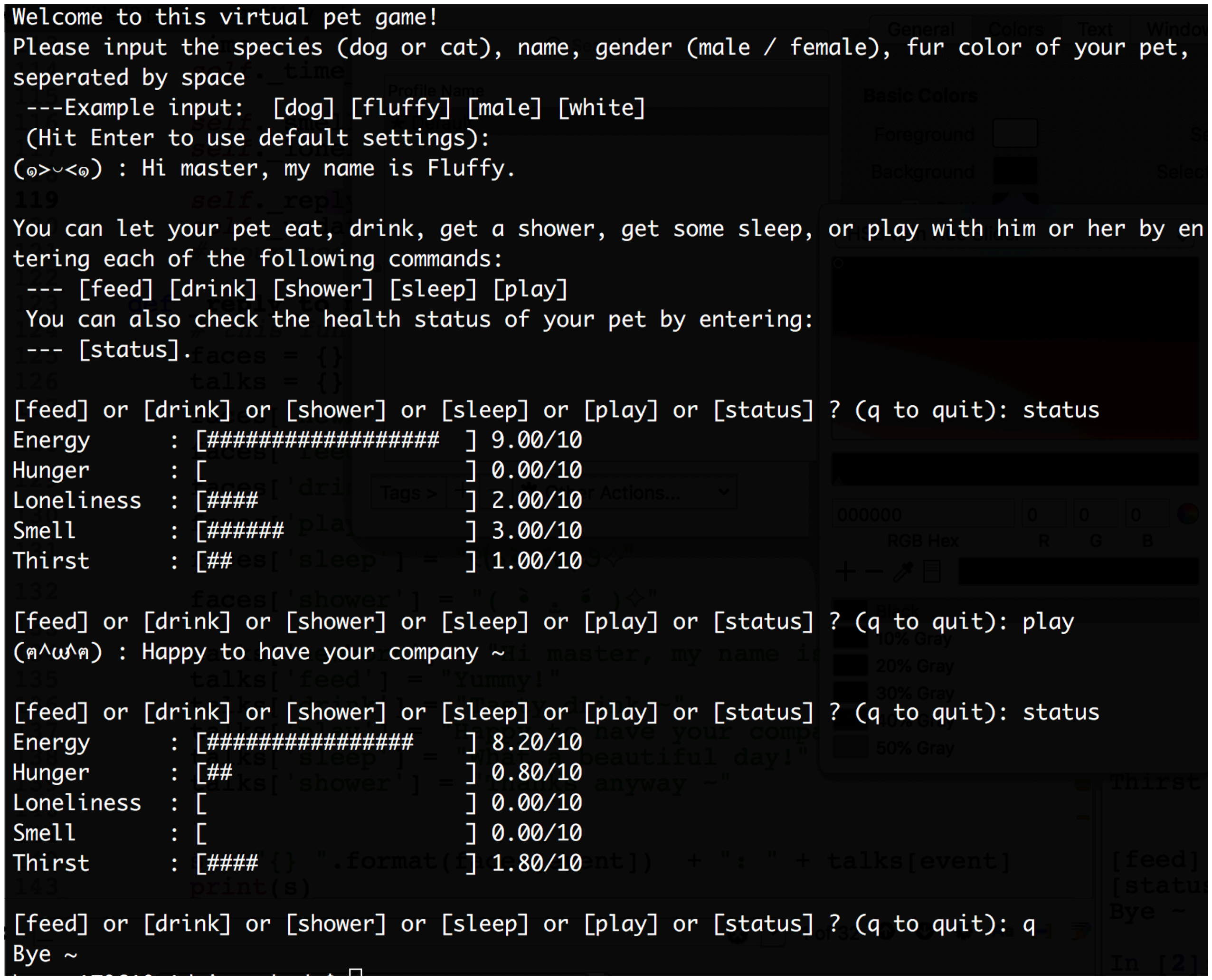
    Both of the above two class definitions should be less than 6 lines of code.

4. Design your main function:
    a. Prompt your user to enter the species, name, gender, fur color of the pet, separated by spaces. If the user hits the Enter key, the program will assume that your user wants to use default values (`species='dog',gender='male',color='white'`). A valid pet has species that is `'dog'` or `'cat'` and gender that is `'male'` or `'female'`. Otherwise, if the user inputs invalid values (say, `'pig fluffy male white'`), the program will repeat this prompt until the user input is valid. An example is given below:



```
Welcome to this virtual pet game!
Please input the species (dog or cat), name, gender (male / female), fur color of your pet,
seperated by space
 ---Example input:  [dog] [fluffy] [male] [white]
 (Hit Enter to use default settings): pig fluffy male white
Please input the species (dog or cat), name, gender (male / female), fur color of your pet,
seperated by space
 ---Example input:  [dog] [fluffy] [male] [white]
 (Hit Enter to use default settings):
(๑>◡<๑) : Hi master, my name is Fluffy.
```

    b. Once the pet input is valid, your program will create an instance of the Dog (or Cat) class, and repeatedly prompt you to input a valid command to interact with your Dog (or Cat) instance, until you enter a 'q'. You have six commands: [feed] or [drink] or [shower] or [sleep] or [play] or [status]. The first five commands are five actions you can take with your pet, and the [status] command will show a table of your pet's health status. A running example of a complete program is shown below.



```
Welcome to this virtual pet game!
Please input the species (dog or cat), name, gender (male / female), fur color of your pet,
seperated by space
 ---Example input:  [dog] [fluffy] [male] [white]
 (Hit Enter to use default settings):
(๑>◡<๑) : Hi master, my name is Fluffy.

You can let your pet eat, drink, get a shower, get some sleep, or play with him or her by en
tering each of the following commands:
 --- [feed] [drink] [shower] [sleep] [play]
 You can also check the health status of your pet by entering:
 --- [status].

[feed] or [drink] or [shower] or [sleep] or [play] or [status] ? (q to quit): status
Energy      : [##################  ] 9.00/10
Hunger      : [                    ] 0.00/10
Loneliness  : [####                ] 2.00/10
Smell       : [######              ] 3.00/10
Thirst      : [##                  ] 1.00/10

[feed] or [drink] or [shower] or [sleep] or [play] or [status] ? (q to quit): play
(ฅ^ω^ฅ) : Happy to have your company ~

[feed] or [drink] or [shower] or [sleep] or [play] or [status] ? (q to quit): status
Energy      : [################    ] 8.20/10
Hunger      : [##                  ] 0.80/10
Loneliness  : [                    ] 0.00/10
Smell       : [                    ] 0.00/10
Thirst      : [####                ] 1.80/10

[feed] or [drink] or [shower] or [sleep] or [play] or [status] ? (q to quit): q
Bye ~
```

Sample Output

Class Test `Pet` `tests the Pet ___init__ methods.`

Class Test `Dog` `tests the Dog __init__ methods.`

Class Test `Cat` `tests the Cat __init__ methods.`

Method Test Pet_Drink_Feed tests get_thirst_level(), get_hunger_level(), drink() and feed() methods and tests some defaults to _init_.

Method Test Pet_Shower_Sleep_Play tests get_energy_level(), shower(), sleep() and play_with() methods and tests some defaults to _init_.

Test 1 input1.txt and output1.txt are on web page

Test 2 includes tests of a number of errors

input2.txt and output2.txt are on web page