Homework 10 - Object Oriented Programming

CS 1301 - Intro to Computing - Fall 2024

Important

- Due Date: **Thursday, November 21st, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Ed Discussion
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

Python is one of many coding languages which uses object oriented programming (OOP). In OOP, classes can be created which contain certain attributes and methods which are shared by all objects of that class. This helps you create concise code which you can re-use. The goal of this homework is to understand OOP and its real world applications.

Written by: Priya Annapureddy

Helpful Information to Know

In this homework, you will be creating a pet adoption center simulation! There will be three main classes working together. Before implementing any of the classes, we highly recommend reading through this introduction and all class descriptions to fully understand how the classes interact.

The first class you will implement is the `Pet` class. A `Pet` object represents an animal, with various attributes such as species, age, health status, and adoption fee. Each pet has specific methods for interacting with potential adopters, tracking health updates, and even comparing itself to other pets based on characteristics like age and weight.

Next, you'll implement the `Owner` class, which represents the potential adopters. An `Owner` object keeps track of adopted pets and interacts with the adoption center. Each owner can adopt pets, check their current pets, and compare themselves to other owners based on the number of pets they've adopted.

Finally, these interactions happen in the `AdoptionCenter` class, which manages the overall pet adoption process. The `AdoptionCenter` facilitates pet adoptions, handles pet availability, keeps records, and manages the owners and pets in its system.

Due to the structure of the autograder test cases, we recommend implementing the `__init__()` methods for all three classes before attempting other methods. This will help you understand the purpose of all class attributes, which is essential for correctly implementing the other methods.

When implementing a class, some methods may require you to perform similar tasks multiple times. Instead of repeating the same code, you can call methods you've already written within other methods to make your code more consistent. For example, if another method in your class needs to remove a pet or check its adoptability, you can simply call remove_pet() or is_adoptable() instead of rewriting the same logic.

Pet

Attributes:

- name (`str`): the name of this Pet
- species (`str`): the species this Pet belongs to
- weight (`int`): the weight of this Pet
- fee (`float`): the adoption fee of this Pet
- age (`int`): age of this Pet
- happiness (`int`): how happy this Pet feels
- adopted (`bool`): whether this Pet is adopted
- isVaccinated (`bool`): whether this Pet has its shots
- owner (`str`): the name of the `Owner` object to which this Pet belongs.

Methods:

- **__init__()**
  - initializes the following attributes:
    - name
    - species
    - weight
    - fee
    - age - should default to `1` if no value is provided
    - happiness - should default to `50` if no value is provided
    - adopted - always initialized to `False`
    - isVaccinated - always initialized to `False`
    - owner - always initialized to `None`
- **is_adoptable()**
  - Return `False` if a pet is already adopted or has a `happiness` level less than or equal to 10.
  - Otherwise, return `True` .
- **update_happiness()**
  - Takes in one additional attribute, a happiness boost.
  - If Pet's `happiness` is less than 100, increase it by the happiness boost.
  - Return `"{pet_name}: happiness = {happiness}"` .

- **feed()**
    - Takes in one additional attribute, a type of animal food (e.g. `"Cat"`, `"Dog"`, etc.)
    - If the Pet's `happiness` is greater than or equal to 100, return `"{pet_name} is full."`.
    - If the `species` of the Pet does not match the species passed in, decrease its `happiness` by 30.
        - After decreasing, if the `happiness` of the Pet is less than or equal to 20, print `"{pet_name} needs to go to the vet."`.
        - Return `"{pet_species}s don't eat {species} food!"`.
    - Otherwise, increment its `happiness` by 20 and its `weight` by 2. Then, return `"{pet_name} is well fed."`.
- **checkup()**
    - If the Pet's happiness is greater than or equal to 100, return `"{pet_name}: Happiness - {happiness}"`.
    - Otherwise, set the Pet's `happiness` to 100 and `isVaccinated` to `True`.
- **up_for_adoption()** - We encourage you to implement this method after you implement `Owner.__init__()`.
    - Takes in two additional attributes, an `Owner` object and an adoption fee.
    - If the Pet is currently adopted:
        - Set the pet's adoption status to be False.
        - Perform a checkup on the Pet.
        - Set the Pet's `owner` to be `None`.
        - Set the Pet's `fee` to be the adoption fee that is passed in.
        - Removes this `Pet` from the Owner's pets.
- **__gt__()**
    - Takes in one additional attribute, another `Pet` object.
    - A `Pet` is considered greater than another if its `age` and `weight` is greater.
- **__str__()**
    - Return the string `"{pet_name} ({species}, {age} yrs, {happiness} happiness) - {isAdopted}, Fee: ${fee}"`, where `isAdopted` is either `"Adopted"` or `"Available"` depending on its adoption status.

Owner

Attributes:

- name (`str`): the name of this Owner
- budget (`float`): how much this Owner can spend on a new pet
- pets (`list`): the Pet objects that this Owner currently owns

Methods:

- **__init__()**
    - initializes the following attributes:
        - name
        - budget - should default to `150` if no value is provided
        - pets - always initialized to `[]`
- **can_afford()**
    - Takes in one additional attribute, a `Pet` object
    - Return `True` if an Owner's `budget` is greater than the Pet's `fee` and `False` otherwise.
- **adopt_pet()**
    - Takes in one additional attribute, a `Pet` object.
    - If the Owner can afford the `Pet` and the `Pet` is adoptable:
        - Set the Pet's `adopted` attribute to `True` .
        - Subtract the Pet's `fee` from the Owner's `budget` .
        - Append the `Pet` to the Owner's list of Pets.
        - Set the Pet's Owner to be the Owner's `name` .
        - Return `"{pet_name} has been adopted!"` .
    - Otherwise, return the string `"{pet_name} could not be adopted."`
- **__lt__()**
    - Takes in one additional attribute, another `Owner` object.
    - A `Owner` is less than another `Owner` if they own less pets and have a smaller budget.
- **__str__()**
    - Return `"{owner_name} has {num_pets} pets and a budget of ${budget}."` .

AdoptionCenter

Attributes:

- name (`str`): the name of this Adoption Center
- pets (`list`): the Pet objects up for adoption in this Adoption Center
- owners (`list`): the Owner objects of the Pets in this Adoption Center
- revenue (`float`): the total revenue of this Adoption Center

Methods:

- **__init__()**
  - initializes the following attributes:
    - name
    - pets - always initialized to `[]`
    - owners - always initialized to `[]`
    - revenue - always initialized to `0`
- **add_pet()**
  - Takes in one additional attribute, a `Pet` object.
  - Append the `Pet` to the Adoption Center's `pets` .
  - Return `"{pet_name} has been added to the adoption center."`
- **remove_pet()**
  - Takes in one additional attribute, a `Pet` object.
  - If the `Pet` belongs to the Adoption Center, remove it.
  - Do nothing if the `Pet` does not belong to the Adoption Center.
- **log_adoption()**
  - Takes in two additional attributes, a `Pet` and an `Owner` .
  - If the `Pet` is adoptable and the `Owner` can afford the `Pet` :
    - Append the `Owner` to the Adoption Center's `owners` , if the `Owner` is not already listed.
    - Remove the `Pet` from the Adoption Center's `pets` .
    - Increase the Adoption Center's `revenue` by the Pet's adoption `fee` .
    - Have the `Owner` adopt the `Pet` . Think about how you can use the adopt_pet() method in the Owner class here.
    - Return `"{owner_name} has adopted {pet_name}!"` .

    - Otherwise, return `"Sorry, {pet_name} cannot be adopted right now."`.

- **find_pet_by_species()**
    - Takes in one additional attribute, a `species`.
    - Returns a list of all Pet objects that are the that type of `species`, case insensitive.
- **adopted_pets_dict()**
    - Returns a dictionary mapping all `Owner` names of the Adoption Center to a list of their Pets.
- **happiness_of_adopted_pets()**
    - Returns the total happiness of all *adopted* Pets from all Owners associated with the Adoption Center.
- **__eq__()**
    - Takes in one additional attribute, another `AdoptionCenter`.
    - An Adoption Center is equal to another if it has the same name and revenue.
- **__str__()**
    - Return `"{center_name} has {num_pets_in_center} pets available for adoption."`.

Provided

The `HW10.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

The `"HW10 Test Cases".pdf` file has also been provided to you. There are no hidden test cases for this homework. Information about testing and scoring your homework are on this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW10.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the "Resubmit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW10.py` on Canvas.