Problem Set 7
Inheritance

Instructions

These exercises are designed to help cement the concepts from lecture and give you an opportunity to apply them yourself. You are welcome to discuss and share ideas with each other, but should write the answers and code yourself.
Some questions ask you to complete a checkoff, which means you need to come to office hours at this link and talk to a staff member. Staff members may complete checkoffs for multiple students at one time, meaning you may not answer every question yourself but instead should listen to what your peers have to say and ask questions! Collaboration is an important part of computer science.

1 Tasks

1.1 Understanding Inheritance - inheritance.py

Look at the **inheritance.py** file and use comments to write down your answers, so that you can explain your thinking to a course staff. You are encouraged to work together!

1. What are the parent and child classes here?
2. What does the code print out? (Try figuring it out without running it in Python.) Please specify which lines of code are printing which lines of output, and why they output what they do.
3. Which `get_description` method is called when `study_spell(Confundo())` is executed? Which `get_description` method is called when `study_spell(Accio())` is executed? Why?
4. What do we need to do so that `print(Accio())` will print the appropriate description (`'This charm summons an object to the caster, potentially over a significant distance'`)? Modify the code so that it prints the appropriate description for `Accio()`.

Checkoff #1—Discuss your answers with a member of the course staff. Suppose we want to add an additional spell, `Expecto Patronum`, which is a powerful protection charm. There are many different Patronus forms, which we want to set when we initialize this spell, so that we can include it in the description of the spell. Explain to a staff member how you would implement this.

1.2 Shape Inheritance - shape.py

In the **shape.py** file, you'll see a `Shape` class that contains functions for finding the area and perimeter of a shape. Below, there are a few shapes which inherit from a superclass, including `Rectangle`, `Square`, `Circle`, and `Triangle`. We will implement the area and perimeter values for all of these.

1. Let's start with the `Rectangle` shape. In the initialization of the shape, you'll see we have `length` and `width` passed into the object. Assign these as attributes of the Rectangle object.
2. Implement the area and perimeter functions for `Rectangle`.
3. Implement the `Square` object. Here, we can take advantage of the `Rectangle` class, so we do not have to write as much code. Hint: we only need to initialize the `Square`!
4. Implement the area and perimeter functions for `Circle`.
5. Implement the initialization and the area function for `Triangle`.

Run the **test_shape.py** file. **Do not change anything in this file.** If you have implemented everything correctly, you should see see that you are passing all the test cases.

Checkoff #2—Discuss your implementation of `Square` with a staff member. How did having the `Rectangle` implementation make that easier? In addition, we did not have a perimeter function for `Triangle`. Explain what would happen if we instantiate a `Triangle` option and call the perimeter function.

1.3 Classes for a Transcript - transcript.py

In this exercise, you will create a few classes for a transcript (pun intended). In the file `transcript.py`, you will find some functions, but no class definitions. Instead, we will be implementing class definitions from scratch! Let's walk through how to do this together.
Note that while we've provided some instructions, we have not specified *how* to implement the functions. Sometimes, the way you implement things does not matter as long as you have followed the instructions provided. As a result, this question is slightly more open-ended and you will have to think about the best way to implement certain functions!

1. First, start by creating a class called `Course`, which takes in a `name` of the course, `percent` representing the grade received in the course, and class `year` in which the course was taken. Write the initialization function for this class!
2. In order to calculate the grade point for the class using the percent grade, we first need to convert the percent into a letter grade. The letter grade percent cutoffs are provided as a comment in the `transcript.py` file. Create a method named `letter_grade` within the `Course` class that computes the letter grade for the course. You should not pass in any parameters (except for `self`). Instead, take advantage of objects' properties!
3. Now, using the letter grade, we can perform the grade point conversion using the unweighted letter-to-grade point conversion provided in the `transcript.py` file. Write a method called `grade_point` within the `Course` class that returns the grade point for the course! Again, you should leverage properties of classes such that you do not need to pass in any parameters except for `self`.
4. Finally, let's add a magic method in order to easily print the course as a string. Define the magic method `__str__` within the `Course` class, which displays the name of the course and the letter grade, separated by some spaces. **Bonus: can you think of a way to print the grades neatly in a single column?**
5. Next, let's create two additional classes: `Honors` and `AP`. These classes should be subclasses of the `Course` class. Override any functions necessary from the `Course` class. Grade points of honors classes should be increased by `0.5`, and grade points of AP classes should be increased by `1.0`. **Try implementing one class before the other! You can use this to debug before you try the second :)**

You should be able to run the `transcript.py` file and see the partial transcript and total GPA of the given transcript! Run the **test_transcript.py** file. **Do not change anything in this file.** If you have implemented everything correctly, you should see see that you are passing all the test cases.

Checkoff #3—Explain to a staff member some of the decisions you had to make when you were implementing these classes.

2 Optional Challenge[1]:

2.1 Working with Pets - pets.py

In the **pets.py** file, you'll see a `Pet` class as well as skeletons for `Dog` and `Cat` classes. Implement the missing parts of the `Dog` and `Cat` classes but don't make any change to the `Pet` class. You can test out your work by making new `Dog` and `Cat` objects and running `print_pet_sounds` and `print_dog_sounds` on them to see if they

[1]optional problems are provided for those who have further interests and want to explore more. You are not responsible for those questions, however.

do what you expect. Take note of the docstrings for each of the functions so you know what types of inputs are appropriate for each function. Then answer the questions that are written as comments at the bottom of the file.

Run the **test_pets.py** file. **Do not change anything in this file.** If you have implemented everything correctly, you should see see that you are passing all the test cases.

Submitting your PSET

After you've finished your PSET and checkoff: Log into your Canvas account, find the post for Problem Set 7 in Assignments, and submit all of the files that you created or edited. After you turn in your assignment, you're all done!