Homework for Week 5

UCSD Extension CSE-41273, Spring 2019.

Included with this instruction file is the file `HW5.py`, containing the skeleton code for the homework.

Please read everything and follow all the directions carefully!

Put your name in the appropriate comment at the top of the program file `HW5.py`. Turn in a zipped file as described in the *About Homework* document.

There are 2 separate parts to the homework: a Circle class and a BankAccount class.

Part 1. Circle class. 75 points total

Modify the Circle class given in 3 ways:
1 (a) Implement `__str__` and `__repr__` methods *exactly* as shown below.
1 (b) Add an attribute variable `radius_log` to the *instance*. Note that this is NOT a property. `radius_log` is a list to contain radius values that have belonged to the circle, where the last item in the list would be the same as the *current* value of the radius. In other words, I want you to keep a log of the changes to the radius of the circle object. Each time the radius changes, add the new value to the list. In order to do this, you will need to make the `radius` a property and add a radius "setter" property method.

Hint: You will need to store the *actual radius value* somewhere as an attribute that is not named `radius`, since the `radius` will now be a property method. I suggest using an attribute variable named `_radius` for this. Note that once you add the radius property and setter, they apply *everywhere*, including the other methods inside the class, which is why you need a separate attribute variable containing the actual radius value. **The two methods for getting and setting the radius should be the only methods to read/modify this attribute.** It should appear nowhere else. Other methods such as `__init__` should **not** reference `self._radius`. See Notes and FAQ at the end of this document for more explanation.

Think about what needs to happen whenever the `radius` is modified. It also needs to happen when the Circle instance is created, so keep in mind the DRY principle of Don't Repeat Yourself, and implement accordingly. Namely, it should only happen once in the code, in the radius setter property.

```
>>> from HW5 import Circle
>>> circle = Circle()
>>> circle
Circle(radius=1)
>>> circle.radius_log
[1]
>>> circle.radius = 2
>>> circle.diameter = 3
>>> circle
Circle(radius=1.5)
>>> print(circle)
Circle of radius 1.5
>>> circle.radius_log
[1, 2, 1.5]
>>> circle2 = Circle(radius=2)
>>> circle2.radius_log
[2]
>>> circle2.diameter = 4
>>> circle2.radius_log
```

```
[2, 2.0]
```

Note that `area` and `diameter` properties must continue to work!

1 (c) Once you have completed parts (a) and (b), modify the Circle class so that it will raise a `ValueError` if the `radius` or `diameter` is set to less than zero.

Everything from (a) and (b) should still work! And remember the DRY principle, the error should be raised from *only one* location in the code.

```
>>> from HW5 import Circle
>>> circle2 = Circle(-2)
Traceback (most recent call last):
  [... traceback information]
ValueError: Radius cannot be negative!

>>> circle = Circle()
>>> circle
Circle(radius=1)
>>> circle.radius = -1
Traceback (most recent call last):
  [... traceback information]
ValueError: Radius cannot be negative!
>>> circle
Circle(radius=1)
>>> circle.radius_log
[1]

>>> circle.diameter = -2
Traceback (most recent call last):
  [... traceback information]
ValueError: Radius cannot be negative!
>>> circle
Circle(radius=1)
>>> circle.radius_log
[1]
```

Part 2. BankAccount class. 25 points total

Here is our old friend BankAccount in a simplified version.
2 (a). Implement `__str__` and `__repr__` exactly as shown below.
2 (b). Implement *truthiness* for our BankAccount objects, such that an instance of the class BankAccount is truthy if the balance is greater than zero and falsey if the balance is less than or equal to zero.

```
>>> from HW5 import BankAccount
>>> account1 = BankAccount(100)
>>> account2 = BankAccount()
>>> account1
BankAccount(balance=100)
>>> print(account1)
Account with balance of $100
>>> account2
BankAccount(balance=0)
>>> bool(account1)
True
>>> bool(account2)
False
>>> account1.withdraw(200)
>>> bool(account1)
False
```

2 (c). Implement comparisons for our BankAccount objects, such that instances can be compared based on their balance. Use functools.total_ordering. *Note:* don't worry about checking for valid inputs, you don't need to implement an `is_valid_operand` method. You can assume you will get good input. In Real Life, you would want to do such checking, but that is not the purpose of this exercise. (making more work for yourself will not impress me or help your grade). **Note:** Keep in mind the purpose of `@total_ordering` and why we use it.

```
>>> from HW5 import BankAccount
>>> account1 = BankAccount(100)
>>> account2 = BankAccount()
>>> account3 = BankAccount(100)
>>> account1 == account2
False
>>> account1 == account3
True
>>> account1 != account3
False
>>> account1 != account2
True
>>> account1 < account2
False
>>> account1 > account2
True
>>> account1 <= account2
False
>>> account1 >= account2
True
>>> account1 <= account3
True
>>> account1 >= account3
True
```

Notes

You do not need to have any try/except blocks. If you really feel you need them, please email me with your code and tell me why you think you need to use a try/except so I can explain why not.

Notes for Circle Class

TL;DR for Circle class

`self._radius` (or whatever you name the variable for the actual radius value) and `self.radius_log` are both *attribute variables*, NOT properties. The **only** code to use/modify `self._radius` are the radius property methods.

The last value of the `self.radius_log` list should always be the same value as `self._radius`, as that is the current actual radius value.

Do not make any modifications to the diameter property methods.

FAQ for Part 1, Circle class

(From previous student questions)
Question. You wrote: “I suggest using an attribute named `_radius` for this. This attribute should only be seen in the radius methods, *not* in any other methods.”

I am a little confused about the reason why `_radius` should be used here? I've searched for it and I only find that it's a private variable and can't be accessed outside the class. Please explain more.

Answer. In order to implement the homework, you need to make the `radius` a property with the `@property` decorator and make an `@radius.setter` method, because *you have to do special things when the radius is modified.* Therefore you will need to store the *actual radius value* somewhere else, for example in another attribute variable that I suggest be named `_radius`. It could be called `actual_radius_value` or something else, but there is a strong convention for naming it with a single initial underscore character.

I repeat: **The two property methods for getting and setting the radius should be the only ones to read/modify this attribute variable that contains the actual radius value.**

Python does not have truly private attributes or methods in the way that other languages like C++ and Java do, and an attribute named `_radius` *can* be accessed outside the class if a user of the class knows about it. It is often said that Python is a programming language for consenting adults. This means that the users of our code (namely, other programmers) have enough rope to hang themselves if they want to be bad programmers. The underscore preceding the attribute is a *convention* that we use to tell users of the class (if they are looking at the code instead of the interface definition or documentation) that `_radius` is considered an "internal" attribute and a user of the class has no business messing with it & if they do, then they shouldn't complain if things don't work right. This is why I suggested it be called `_radius`, but you don't have to call it that. It's just that you can't call it `radius` because that is now a property of the class for interfacing with users of the class.

NOTE: In your search travels (because I know you will use Google instead of thinking, haha), you may find examples of making "private" variables that are prefixed with *two underscores* instead of one, which causes Python to make a weird replacement name. This is called *"name-mangling"* and its true purpose is for preventing potential name clashes or ambiguity in an inheritance hierarchy, not for making variables private (since there is no such thing in Python). Please resist the temptation to use name-mangling as it is completely unnecessary here and makes debugging difficult, should you need to do any debugging. **In addition, if I see name-mangling, I know that you don't understand what you are doing and are just copying something from the internet, so I will take points off.**

So a *user* of our Circle class *should* only use the `radius`, `area`, `diameter`, and `radius_log` attributes.

As I said, in order to implement the homework, you will need to implement the radius as property methods for getter and setter, *because you have to do special things when someone wants to change the radius value.*