CS112 Homework 8: – Groups Allowed – Classes/Exceptions

Due: Tuesday, December 3rd, 11:59pm

As this is a **Homework**, you can read any and all materials, ask us questions, talk with other students, and learn however you best learn in order to solve the task. Just create your own solution from those experiences, and turn in your work.

Notes

Classes allow us to define entirely new types. We might think of each type in Python as a set of values, and we use a class declaration to describe how to build and interact with all the values of this brand new type. We will often need at minimum a constructor __init__ (telling us what instance variables to always create), __str__ and __repr__ methods (telling Python how to represent the thing as a string), and then any extra methods that we deem useful ways of interacting with the new values.

- Remember, don't call `input()` or `print()` anywhere.
- Tester: https://cs.gmu.edu/~marks/112/homeworks/testerH8.py
- New! Instead of limiting to each function name, classes problematically often share method names. So we have provided the following "test-batch" names that you can use:
  - `Device`, `Outlet`, `Circuit` (tests the `__init__`, `__str__`, and `__repr__` methods of that class)
  - `eq  device_on_off_toggle`
  - `outlet_max_watts  outlet_watts_now  add_device  remove_device  outlet_turn_off_all`
  - `add_outlet  circuit_max_watts  circuit_watts_now  circuit_turn_off_all`

- Run it:
```
python3 testerH8.py gmason76_230_H8.py
python3 testerH8.py gmason76_230_H8.py Device eq device_on_off_toggle
python3 testerH8.py gmason76_230_H8.py outlet_max_watts
```

Turning It In

Add a comment at the top of the file that indicates your name, userID, G#, lab section, a description of your collaboration partners, as well as any other details you feel like sharing. Please also mention what was most helpful for you. Once you are done, run the testing script once more to make sure you didn't break things while adding these comments. If all is well, go ahead and turn in **just your .py file** you've been working on, named with our usual convention (see above), over on BlackBoard to the correct assignment. Please don't turn in the tester file or any other extra files, as it will just slow us down.

Grading Rubric

```
Pass shared test cases       100      Deductions possible for violating the
------------------------------        restrictions or hardcoding to the tester.
TOTAL:                       100
```

Restrictions:

- you can't `import` anything.
- You can use all the built-in functions and types (and their methods) as you like! ☺

`class Device:` Represents an electric device that can be plugged into an Outlet to receive power.

- `def __init__(self, name, watts=100, on=False):` assigns all non-`self` parameters to fields of the same name (this is the most common kind of python constructor)
- `def __str__(self):` returns a str that follows these patterns:
  when on, `"(+100W: lightbulb1)"` or, when off, `"(+0W: blender)"`
- `def __repr__(self):` returns a str that follows this pattern (note the single quote usage):
  `"Device('coffeepot', 200, True)"`
- `def __eq__(self,other):` both parameters are `Device`s. returns `True` when they have identical values for all three fields, `False` otherwise. (overrides the meaning of `==` for `Device`s!)
- `def turn_on (self):` Ensures the device is `on`.
- `def turn_off(self):` Ensures the device is not `on`.
- `def toggle  (self):` Changes the device's `on` status between `True` and `False`.

`class Outlet:` Represents an electrical outlet (the thing you plug devices into to get electricity). It might have some `Device`s plugged into it, which may be on or not.

- `def __init__(self, devices=None):` ensures there's a field named `devices` for this object. Uses the provided value; but if `devices==None`, a new empty list is created as the value.
- `def __str__(self):` returns a str that follows this pattern. Note, you ***should*** try to use the str definition from `Device`s here (that's the learning goal).
  `"Outlet([(+100W:lightbulb1), (+0W: blender)])"`
- `def __repr__(self):` returns a str that follows this pattern. Note, you ***should*** try to use the repr definition from `Device`s here (that's the learning goal).
  `"Outlet([Device('lightbulb1', 100, True), Device('blender', 200, False)])"`
- `def __eq__(self,other):` both parameters are `Outlet`s. returns `True` when they have identical values for their field, `False` otherwise. (overrides the meaning of `==` for `Outlet`s!)
- `def max_watts(self):` returns the maximum watts on this outlet if all devices are on.
- `def watts_now(self):` returns the actual watts being used by this outlet at the moment.
- `def add_device(self, device):` adds the given `device` to the end of the devices list.
- `def remove_device(self, name):` finds and removes the (first) `device` with the indicated `name`. (a `string`, not the whole `Device` object). Returns the device if found, or `None` when no device of that name was found.
- `def turn_off_all(self):` ensures that all devices on this outlet are turned off.

`class Circuit:` Represents multiple outlets that are all connected together (behind the walls of your house), to a single breaker in the building.

- `def __init__(self, outlets=None):` ensures there's a field named `outlets` for this object. Uses the provided value; but if `outlets==None`, a new empty list is created as the value.
- `def __str__(self):` returns a str that follows this pattern. Note, you ***should*** try to use the str definition from `Outlet`s here (that's the learning goal).
  `"Circuit([Outlet([(+0W: blender)]), Outlet([(+15W: LED), (+0W: curling iron)])])"`
- `def __repr__(self):` returns a str that follows this pattern. Note, you ***should*** try to use the str definition from `Outlet`s here (that's the learning goal).
  `"Circuit([Outlet([Device('lightbulb1', 100, True)])])"`
- `def __eq__(self,other):` both parameters are `Circuit`s. returns `True` when they have identical values for their field, `False` otherwise. (overrides the meaning of `==` for `Circuit`s!)
- `def add_outlet(self, outlet):` adds the given `outlet` to the end of the outlets list.
- `def max_watts(self):` returns the maximum watts on this circuit if all devices were on.
- `def watts_now(self):` returns the actual watts being used by this circuit at the moment.
- `def turn_off_all(self):` ensures that all devices on this circuit are turned off.