CIS 211

Spring 2020 Midterm 1

This exam contains three programming problems. For each programming problem, there is an accompanying file of starter code in this same archive. There is also a "quiz" question in Canvas corresponding to each problem. You should edit the the starter code and test it, then upload the edited file to the corresponding quiz question.

You must complete this exam by yourself, without help.

- You may use printed and online materials including tutorials and the documentation on Python.org.
- You may not pose a question pertaining to the exam, in person or through any communication medium, with the single exception that you may pose clarifying questions as *private* posts on Piazza.

You will have two hours to complete the exam, within a three day window.

This exam is not designed to test how fast you can type or how fast you can reproduce memorized examples. Do keep track of your time, but don't rush. If you get stuck on a problem, go on to the next one and come back.

Each problem comes with a test suite. However, passing the test cases in the test suite does not guarantee correctness, and correctness does not guarantee full points. In addition to being correct, your code must comply with the coding standards of `https://uo-cis211.github.io/reference/CodingStandards.html`. Also it must be *clean, concise, well-designed code.* Code that is correct but inelegant will not earn full points.

1. [20 points]

The starter code for this problem is `q1_intervals.py`.

A closed interval [m,n] represents the set of numbers that are at least m and at most n. Note that a closed interval includes both of its bounds, i.e., $m \in [m, n]$ and $n \in [m, n]$.

Objects of class `Interval` represent closed intervals of integers.

Class `IntervalCollection` is a wrapper for a kind of list whose elements are `Interval` objects. An integer is contained in an `IntervalCollection` if and only if it is contained in any of the `Interval` objects in the `IntervalCollection`.

You must write:

- the `contains` method of class `Interval`
- the `append` method of class `IntervalCollection`
- and the `contains` method of class `IntervalCollection`

Your solution must pass the tests in `test_intervals.py`.

2. [20 points]

The starter code for this question is `q2_color_tiles.py`. The accompanying test suite is `test_color_tiles.py`.

In this problem you must complete classes `Tile` and `Row`.

A `Tile` object has a color, which is either red or blue. Two 'Tile' objects are considered equal if they have the same color. The printed representation of a 'Tile' object is a letter indicating its color, either "r" for red or "b" for blue.

A `Row` object is a wrapper for a list of `Tile` objects. Initially a `Row` object is empty (it represents a row of no tiles). We can place tiles in a `Row` in two ways, all at once with `from_abbrevation` or one by one with `append`.

Two `Row` objects are equal if they have the same length and their corresponding elements are equal.

`append` works just like the `append` method of a list.

`from_abbrevation` replaces the current contents of the `Row` by a new list of `Tile` objects. For example, if `r` is a `Row` object, then `r.from_abbreviation("rbr")` will cause the elements of `r` to be a red tile, a blue tile, and a red tile, in that order.

The string representation of a row of tiles is a string of letters indicating the colors of its tiles. For example, consider the `row` object in the prior paragraph with a red tile, a blue tile, and a red tile. `str(r)` would be `"rbr"`.

Your completed file `q2_color_tiles.py` must pass the tests in `test_color_tiles.py`, comply with CIS 211 coding style, and be clean, concise, readable code.

3. [20 points]

The starter code for this problem is `q3_shapes.py`. The accompanying test suite is `test_shapes.py`.

You must make the `above` and `below` methods in class `Shape` work correctly. You should

- implement methods `y_min` and `y_max` in class `Rect`. These will override methods of `Shape`.
- complete methods `above` and `below` in class `Shape`. These will be inherited in classes `Circle` and `Rect`.
- Do *not* override methods `above` and `below` in classes `Circle` and `Rect`. Let them inherit these methods from `Shape`.