CSE 4256

Due Sunday Mar 13 11:59 PM EST

Dr. Alan Weide

Spring Term 2022

Homework 7

Code Quality

Any submitted homework that raises any errors when I invoke Python with your submitted `main.py` file will receive no credit. As discussed in class, I will enforce the *spirit* of this rule and not the *letter* of this rule. In particular, simply deleting all code from `main.py` will result in no credit being given.

Beyond that, while you do not need to correctly answer *every* question in order to earn a grade of "satisfactory" on assignments in this course, please at least *attempt* every question. You may be surprised how quickly things click once you start working on them!

Assignment

The purpose of this assignment is to review the Python language features we have studied so far this semester. Note that it is due **two weeks** from the day it was assigned. The extended deadline is for two reasons:

- There is a lot of work to be done to complete this assignment.
- The last question asks you to apply topics we will cover next week.

Therefore, I *strongly* suggest that you do not wait a week before getting started. At the same time, please recognize that I don't expect you to complete it within a single week.

Many of the activities in this assignment are modified from challenge activities in previous assignments. You should complete each activity as it appears in *this* document, and not as it appeared in any previous documents.

This assignment should be completed as a collection of several Python source (`.py`) files, one of which should be named `main.py` and should contain a variety of test cases for the code in your other files, and another of which should be named `tests.py` and should contain test cases written with either the `unittest` or `pytest` framework.

(1) Complete the "Challenge Activities" numbers 2, 3, and 5 from Homework 2 in a file named `triangles.py`. They are reproduced here, modified slightly.

- Write a function called `triangles(n)` that, given a positive integer `n`, produces a list of the first `n` triangle numbers. You may assume that `n` is a positive integer.
- Write a function called `ctriangles(n)` that produces a list identical to `triangles(n)`, but does so using a single-line list comprehension. **Hint**: the built-in function **`sum`**`(s)` is probably useful here.
- Write a function called `pascal(r)` that, given a positive integer `r`, produces a list containing the first `r` rows of Pascal's Triangle (more precisely, the function should only generate all *non-zero* entries in the triangle). You may assume that `r` is a positive integer. You *should not import any modules* to complete this task, including the `math` module.

  Recall: The kth entry in the nth row of Pascal's Triangle has the value $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. It has the nice property that $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

(2) Complete the "Challenge Activities" numbers 18 and 19 from Homework 4. They are reproduced here, modified slightly.

- The `diceroller` function in the Homework 4 template file is fine, but the output leaves something to be desired. Copy the `diceroller` function from the Homework 4 template into a file named `dice.py`. First, replace the `mcg` pseudorandom number generator with a better one, such as the `random` module. Next, add a new function called `print_bar_chart(data: `**`dict`**`)` that takes, as input, the dictionary returned by `diceroller` and prints to the console a horizontal bar chart made of unicode characters. Do not import any charting modules—use only the built-in functions for **`dict`** and **`str`** and, if you want, f-strings. A useful character to use as you build each bar is the Unicode character 'FULL BLOCK' (U+2588), which can be included in Python source as the **`str`** literal `"\u2588"`. Here are the requirements for the printed chart:

  - There should be one line for each possible roll value, and they should be sorted in increasing order of roll value.
  - Each line should begin with a label (*i.e.*, a roll value) followed by at least one space, followed by the bar for that value.
  - Each bar should be left-aligned in the console.
  - Each line should be no more than 80 characters long, including the label.

  In listing 1, see an example output based on the dictionary returned by `diceroller(6)` (it uses # as the bar character because of limitations in my LaTeX installation, and the line width is only 60 characters because of space limitations on this page).

LISTING 1. Example run of `python3 -i dice.py`.

```
>>> for line in diceroller(6, 5000):
...     print(line)
...
1 ############################################################
2 ####################################################
3 ################################################################
4 ########################################################
5 ##################################################
6 ############################################################
```

- Copy the generator function `words(s)` from your solution to Homework 4 into a file named `words.py`. If it is not correct, make it correct. Use that generator function to write a function called `firstlines(filename)` that builds a dictionary of every word that appears in a file mapped to the line number on which it first appears. For example, suppose the contents of the file `foo.txt` are

  ```
  this is
  ```

```
my file
this is not
your file
```

Then a call to `firstlines("foo.txt")` should produce the dictionary `{'this': 0, 'is': 0, 'my': 1, 'file': 1, 'not': 2, 'your': 3}`.

(3) Complete the "Challenge Activities" numbers 11 and 12 from Homework 5 in a file named `fraction.py`. They are reproduced here, modified slightly.

- Augment the `Fraction` class with a class method called `from_str(cls, str_rep: str) -> Fraction` with the following definition.

```
@classmethod
def from_str(cls, str_rep: str) -> 'Fraction':
    """Produces a fraction from string str_rep.

    Requires str_rep is in one of two forms.
    Either str_rep is the string representation of a fraction
        (e.g., '5/3' or '-18/36'),
    or str_rep is the string representation of a decimal number
        (e.g., '47.625' or '-8.3333').
    The returned fraction should be in reduced form and have the
        value one would "expect" from the input string.
    """
```

- Modify the `__init__` method of the `Fraction` class to take either two **`int`** arguments or a single **`str`** argument, treated as it is in `from_str`. **Hint**: use default parameter values to detect whether the function was called with one or two arguments.

(4) Modify the `from_str` method you just wrote to raise a `ValueError` if the string passed as an argument is not in one of the acceptable forms.

(5) Starting from your solution to Homework 6, complete the "Challenge Activities" numbers 4 and 5 from Homework 6 in a file named `graph.py`. They are reproduced here, modified slightly.

- Implement the other two subclasses of `Graph`.
- There are two (non-abstract) functions in the `Graph` class that are marked with `TODO (challenge)` comments: `depth_first_search` and `breadth_first_search`. Implement them as generator functions.

(6) Create a file named `main.py` that **`import`**s the files in which you completed the rest of the activities, and fill it with test cases for the rest of your code.

(7) Create a file called `tests.py` that contains test cases for all of your code written in one of the frameworks we discussed (either `unittest` or `pytest`). `tests.py` should execute all of the test cases when it is run via `python3 test.py`.

Challenge Activities

Some homeworks (such as this one) will have additional challenge activities. These activities **do not contribute to your grade**, but they are problems that I find interesting or challenging.

(8) Modify your `main.py` file to execute all of your test cases when it is run via `python3 main.py`.

(9) Keeping all of your test cases in a single file is less than ideal, especially when working in a large project. Reorganize your test cases so that it will remain manageable even when your project grows. Create a shell script called `runtests.sh` that, when executed at the command line, runs all of the test cases in your project. As an added challenge, do it with a *single* command (this may involve creating or modifying Python source files).

(10) Complete the rest of the challenge activities from Homeworks 1–6.

(11) Write a program that takes as input the name of a directory containing a bunch of Python source files that all expose the same set of functions and classes, and runs a suite of test cases on each of those files, pausing after each run to allow the user to inspect the result.

Submission

To submit this assignment, upload a **`.zip`** file containing all of your Python files to the "Homework 7" assignment on Carmen. As always, be sure to note all group members who contributed to the assignment and what those contributions were.