COMP 1510 Programming Methods Lab 07

Christopher Thompson
`cthompson98@bcit.ca`

BCIT CST — October 2019

Welcome!

This lab is due at the end of your lab period.

Welcome back. There are six weeks left in the term. We've got a lot to cover. But first, let's take one final look at your midterm before moving on.

You will need a copy of your midterm for this lab. Do not forget it at home.

1 Grading

Figure 1: This lab is graded out of 5

This lab will be marked out of 5. For full marks this week, you must:

1. (3 points) Correctly implement the coding requirements in this lab.
2. (1 point) Correctly format and comment your code.
3. (1 point) Correctly generate unit tests for your code.

2 Project Setup

Please complete the following:

1. For your seventh lab, all files must go into the Lab07 folder.
2. After you complete each task, commit and push your change to version control. In order to earn full marks, you must commit and push after each function is complete. Your commit message must be a short message to me that describes what you completed for this commit, i.e., "deconstructed base_conversion", or "debugged function_name", etc.
3. Since I am already a collaborator I will pull your future work automatically.

3 Requirements

Please complete the following:

1. Implement the following functions. The name you must use is below. Do not modify the name I have provided.

2. Include correctly formatted docstrings that include comprehensive doctests (where possible) for each function. These docstrings must explain to the user how the function is meant to be used, and what the expected output will be when input is provided.

3. Each function must also be tested in a separate unit test file by a separate collection of unit tests. Ensure your test cases and functions have descriptive, meaningful names, and ensure that every possible valid input is tested for each function.

4. Ensure that the docstring for each function you write has the following components (in this order):

   (a) Short one-sentence description that begins with a verb in imperative tense

   (b) One blank line

   (c) Additional comments if the single line description is not sufficient (this is a good place to describe in detail how the function is meant to be used)

   (d) PARAM statement for each parameter that describes what it should be, and what it is used for

   (e) PRECONDITION statement for each precondition which the user promises to meet before using the function

   (f) POSTCONDITION statement for each postcondition which the function promises to meet if the precondition is met

   (g) RETURN statement that describes what the function returns

   (h) One blank line

   (i) And finally, the doctests. Here is an example:

   ```
   def my_factorial(number):
       """Calculate factorial.

       A simple function that demonstrates good comment construction.
       PARAM: number, a positive integer
       PRECONDITION: number must be a positive integer
       POSTCONDITION: calculates the correct factorial
       RETURN: correctly calculated factorial as an int

       >>> my_factorial(0)
       1
       >>> my_factorial(1)
       1
       >>> my_factorial(5)
       120
       """
       return math.factorial(number)
   ```

5. Part C of the midterm consisted of six questions. I'd like to make sure you can answer these questions correctly.

6. **Create a new file called midterm.py**.

7. Inside midterm.py, **implement correct, efficient solutions to all six of the midterm coding questions**.

8. Your solutions must **include docstrings. Add doctests if necessary** that demonstrate the correct use of the code to potential users.

9. Of course we need to write some unit tests, too. (Of course). I'd like you to **write a small suite of unit tests for three of the functions, to wit: list_tagger, cutoff, and prepender**:

   (a) for list_tagger, test the following disjointed equivalency partitions:

      i. empty list

    ii. list of length 1
    iii. a non-empty list that contains more than one item.
  (b) for cutoff, test the following disjointed equivalency partitions:
    i. [], 0 (empty list and zero)
    ii. [], 5 (empty list and five)
    iii. [0], 0 (list of length one containing a zero, and zero)
    iv. [0], 5 (list of length one containing a zero, and five)
    v. [2], 2 (list of length one containing a two, and two)
    vi. [2], 4 (list of length one containing a two, and four)
    vii. [1, 2, 3, 4, 5], 0 (list of five positive numbers, and zero)
    viii. [1, 2, 3, 4, 5], 2 (list of five positive integers, and a number that divides evenly into some of them)
    ix. [2, 2, 2, 2, 2], 2 (list of five identical positive integers, and a number that divides evenly into all of them)
    x. [2, 2, 2, 2, 2], 10 (list of five identical positive integers, and a number that doesn't divide evenly into any of them)
    xi. [3, 6, 9, 12, 15], 3 (list of five different integers, and a number that divides evenly into all of them)
    xii. Can you think of any other combinations of input that are unrelated to the ones I have listed here?
  (c) for prepender, test the following disjointed equivalency partitions:
    i. [], "" (an empty list and a string that contains no characters)
    ii. [], "Python" (an empty list and a string that contains some characters)
    iii. ["Python"], "" (a list of length one and a string that contains no characters)
    iv. ["Python"], "I love " (a list of length one and a string that contains characters)
    v. ["Python", "is", "better", "than", "JavaScript"], "" (a list of length greater than one, and an empty string)
    vi. ["Python", "is", "better", "than", "JavaScript"], "Umm... " (a list of length greater than one, and a non-empty string)
    vii. Can you think of any other combinations of input that are unrelated to the ones I have listed here?

10. That's it. Not so hard, was it?