CSC108H Lab 9: Testing and Debugging

As usual, pick a partner and complete the lab using pair programming (taking turns driving and navigating). At the end of the lab, please return this handout to your TA. We will post the handout on the course website at the end of the week.

Objectives

• Practice selecting good test cases.

• Practice implementing a set of test cases using `nose`.

Testing

In this lab, you will write a nose test suite to thoroughly test several functions, then use it to identify and fix any bugs that you find.

1. Download `buggy_functions.py` and `test_buggy_functions.py` from the lab page. The module `buggy_functions.py` contains `is_lowercase` as well as a handful of other functions for which you will write tests. `test_buggy_functions.py` already contains a set of `nose` tests for `is_lowercase`. Take a look at the structure of the file. Each of the tests contains a single `assert` statement that checks that the function being tested returns the correct value, and each test is named to inform you that it is a test, that it is testing a specific function, and that it is testing a specific input to that function. The `__main__` block contains a single line that causes all of the test functions in the module to be called.

2. Run the test module. You'll find that at least one of the tests for `is_lowercase` fails. Look at the output to determine which case is triggering the error and then edit `buggy_functions.py` to correct the error. Re-run the tests to verify that your fix is correct.

3. The table on the next page describes the remaining functions in the module. Several (perhaps all) of the functions in `buggy_functions.py` have at least one error. Your job is to find them. Use the following steps for each of the functions:

(a) Write down the inputs to the function, and make a list of the "attributes" (or features) of these inputs that you could vary across your test cases. For example, if one parameter is a list, you could vary its length.

(b) Use these ideas to generate a table of interesting test cases for the function. It should look like this (but with lots of rows):

Input Values	Expected Result	Purpose

Each row represents a single test cases. Make sure that, among the rows of the table, you cover the issues that you raised above, E.g., for a function with a list as a parameter, make sure you include tests cases with different list lengths.

(c) Write a `nose` test in `test_buggy_functions.py` for each test case in your table. Each of your tests should use assert to check the result of the function on the input with the expected output. Make sure to include a descriptive message for each assert statement so that you know which test case fails.

(d) Run your tests. If you find any errors, debug and correct them.

(e) Switch driver and navigator.

`evens(list)`	Given a list, return a new list consisting of those members of the given list whose index is even. Items in the new list should be in the same order as they appear in the given list. Do not modify the original list.
`reverse(list)`	Given a list, return a new list that contains the items from the original list in reverse order. Do not modify the original list, and do not use `list.reverse`.
`left_strip(str, str)`	Given two strings, where the second is a single-character string, return a new string identical to the first string but with any occurrences of the second string removed from the beginning. For example, `left_strip("fffabcdefffg", "f")` should return `"abcdefffg"`. (This is much like `str.lstrip`.) Do not use any `str` methods to implement this function.
`halve_my_digits(str)`	Given a string of digits, return a string that represents the number obtained by dividing each digit in the given string by 2 (integer division). For example, `halve_my_digits("123456")` should return the string `"011223"`.