UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephone: (306) 966-4886, Facimile: (306) 966-4884

Assignment 9

Testing and Debugging

Date Due: Wednesday, November 17, 11:59pm **Total Marks: 35**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M (yes, sometimes typos happen – do not be confused). Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you. Do not submit folders, or zip files, even if you think it will help.
- **Programs must be written in Python 3.** Marks will be deducted with severity if you submit code for Python 2.
- **Assignments must be submitted to Canvas.** There is a link on the course webpage that shows you how to do this.
- **Canvas will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Questions are annotated use descriptors like "easy" "moderate" and "tricky". All students should be able to obtain perfect grades on "easy" problems. Most students should obtain perfect grades on "moderate" problems. The problems marked "tricky" may require significantly more time, and only the top students should expect to get perfect grades on these. We use these annotations to help students be aware of the differences, and also to help students allocate their time wisely. Partial credit will be given as appropriate, so hand in all attempts.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (14 points):

Purpose: To practice testing functions you wrote yourself.

Degree of Difficulty: Easy to Moderate

(a) Write the following functions and their docstrings:

- `between(num1, num2, num3)` takes 3 integer arguments and returns `True` if `num2` is between `num1` and `num3`. It is not between them if it is equal to either of the other two. For example, given the inputs `5`, `3` and `0`, the value returned should be `True`. Given the inputs `-2`, `2` and `2`, `False` should be returned. Note that there is no restriction that `num1` must be less than `num3`.
- `majorityEven(num_list)` returns `True` if `more than half of the integers` in the `num_list` are divisible by 2, with no remainder, otherwise it returns `False`. The list can be of any size. Recall that zero is divisible by 2 with no remainder. For example, the function should return `False` for the list `[1,2,3,6]` (as only two of the four numbers are divisible by 2) and `True` for the list `[0,1,-4]` (as two of the three numbers are divisible by 2).

(b) Generate at least six test cases for each function you wrote in part (a). You may use white-box and/or black-box test case generation. Carefully and strategically select your white-box test cases. As you choose your cases, try to think about all the different paths of execution through your code and identify cases that cause the execution of each of those paths at least once.

(c) Implement a test driver which thoroughly tests both functions using the tests identified in part (b). You can use either plain if-statements (as demonstrated in readings 15.2.4) or a list-of-dictionaries to implement your test driver (as demonstrated in class during chapter 15 exercise 3).

What to Hand In

- A document called `a9q1.py` containing your functions.
- A document called `a9q1_testing.py` containing your test cases and test driver implementation for each function.

Be sure to include your name, NSID, student number and instructorâĂŹs name at the top of all documents.

Evaluation

- 1 mark: Your function `between()` takes three integers and correctly returns `Boolean` value.
- 1 mark: The docstring for `between()` is sufficiently descriptive, indicating its purpose and requirements (because docstrings are important for black-box test case generation!).
- 3 marks: You designed at least six appropriate black-box or white-box test cases for `between()` including some that would return `True` and some that would return `False`.
- 2 marks: Implemented a test driver that would reveal faults in `between()`.
- 1 mark: Your function `majorityEven()` takes a list of integers and correctly returns Boolean value.
- 1 mark: The docstring for `majorityEven()` is sufficiently descriptive, indicating its purpose and requirements.
- 3 marks: You designed at least six appropriate black-box or white-box test cases for `majorityEven()`, including some that would return `True` and some would return `False`.
- 2 marks: Implemented a test driver that would reveal faults in `majorityEven()`.
- **-1 mark if the student did not include their name, NSID, student number and instructorâĂŹs name at the top of the submitted file.**

Question 2 (12 points):

Purpose: To practice testing and debugging code that someone else wrote.

Degree of Difficulty: Moderate

In the provided file `a9q2_functions.py` you'll find two functions:

- `isValidPhoneNumber()`: Returns `True` if the provided phone number (represented as a string) is correctly formatted, otherwise `False`. To be considered correctly formatted, phone numbers must be written as `###-###-####`, where # is a digit between 0 and 9.
- `validatePhoneBook()`: A phone book is a list where each entry is a phone book record (represented as a dictionary; see below for more details). This function checks each phone book record in the phone book for correctly formatted phone numbers.
  A phone book record is a dictionary which initially has two keys: the key `"name"` mapped to the contact's name (string) and the key `"phone number"` mapped to that contact's phone number (also a string). `validatePhoneBook()` adds a new key `"valid"` to the record with the value `True` if the phone number is formatted correctly, otherwise `False`.

Here is how a sample phone book might look before and after `validatePhoneBook()`:

```
# before
[ { "name" : "Department of Computer Science",
    "phone number" : "306-966-4886" },
  { "name" : "Department of History",
    "phone number" : "306.966.8712" } ]
# after
[ { "name" : "Department of Computer Science",
    "phone number" : "306-966-4886",
    "valid" : True },
  { "name" : "Department of History",
    "phone number" : "306.966.8712",
    "valid" : False } ]
```

The provided functions contain errors. Your task will be to find the errors by systematically testing. You'll hand in an explanation of what the errors are, and how you fixed them, but you will not hand in the corrected code.

Part of the exercise is figuring out what the code is supposed to do based on the documentation!

1. Write white-box and black-box tests for the function `isValidPhoneNumber()`. You should do this without Python, either on paper, or in a simple document. Don't worry about running your tests until you have thought about which test cases you want. Make sure you have **at least** 3 black-box tests and 3 white-box tests. More tests may be useful.
2. Implement a test driver using your test cases in a document named `a9q2_testing.py`. This should be done using the techniques seen in the textbook (Chapter 15) and as demonstrated in class (Chapter 15 Exercise 3).
3. Run your tests and copy/paste the console output to a document named `a9q2_output.txt`. The console output you hand in should come from the program before you try to fix it!
4. Try to deduce the problems with the `isValidPhoneNumber()` function from the output of your testing. Then fix the error(s) in the function!
5. In the document `a9q2_output.txt`, describe the error(s) you found and how you fixed it (them). You do not need to write a lot; a sentence or two for each error is all we want. Write your explanation as if you are explaining the error to a colleague of yours who wrote the functions.
6. Repeat the above steps for the second function `validatePhoneBook()`.

Note: When writing your test driver for `validatePhoneBook()`, you can assume existing keys and values in a phone book will be unchanged. In other words, it is sufficient to only check whether the key "valid" exists and is paired with the correct value in each phone book record after `validatePhoneBook()` is called.

What to Hand In

Remember: for this question, you do not need to hand in the fixed code!

- A document named `a9q2_testing.py` containing your testing code as described above.
- A document named `a9q2_output.txt` (RTF and PDF formats are also allowable) with the console output of your testing copy/pasted into it and the explanations of the errors you found.

Be sure to include your name, NSID, student number, course number, lecture section and laboratory section at the top of all documents submitted.

Evaluation

- 3 marks: Your test driver for `isValidPhoneNumber()` contains at least 6 appropriate test cases.
- 3 marks: Your test driver for `validatePhoneBook()` contains at least 6 appropriate test cases.
- 2 marks: The output of your test drivers reveal faults in the starter code.
- 3 marks: You correctly identified 3 errors in `isValidPhoneNumber()` and explained briefly how you fixed them.
- 1 mark: You correctly identified 1 error in `validatePhoneBook()` and explained briefly how you fixed it.

Question 3 (9 points):

Purpose: More practice writing test-cases and implementing test-drivers.

Degree of Difficulty: Moderate

In the document `a9q3_functions.py` you'll find the function:

- `medianOfTwoLists()`: Given 2 lists whose elements are in increasing order, return the **median** of the two lists.
- **median**: The median is the middle number in an ascending or descending list of numbers. If there is an odd amount of numbers in the list, the median value is the number that is in the middle; if there is an even amount of numbers in the list, the median is the average of the middle pair numbers. It means you need to get the middle pair numbers of the list, add them together, and divide the sum by two to find the median value. It is notable that the median is undefined in empty lists.
- The idea of the function is merging the two list first, then find the median value of the merged list whose elements should also be in increasing order.
- The idea of merging the two increasing list is to step through both lists one element at a time. Compare two elements, one from each list, and put the smaller into the merged list. If you reach the end of either list, add the rest of the other list to the end of the merged result.

```
1  def medianOfTwoLists(ls1, ls2):
2      '''
3      Given 2 lists whose elements are in increasing order,
4      merge them into one list of elements in increasing order,
5      and return the median of the merged list, which is also
6      the median of the two input lists.
7      :param ls1: a list in increasing order
8      :param ls2: a list in increasing order
9      :return: the median of the two input lists
10     '''
11     merged = []
12     index1 = 1
13     index2 = 1
14
15     while index1 < len(ls1) and index2 < len(ls2):
16         # step through the lists one element at a time
17         # put the smaller element in the result
18         if ls1[index1] < ls2[index2]:
19             merged.append(ls2[index2])
20             index1 = index1 + 1
21         else:
22             merged.append(ls2[index2])
23             index2 = index2 + 1
24
25     # add the remaining elements
26     if index1 < len(ls1):
27         merged.extend(ls1[index1:])
28
29     # find the median
30     if len(merged) == 0:
31         return "The median is undefined for empty lists!"
32     else:
33         return merged[len(merged) // 2]
```

This function contains errors. Your task will be to find the errors using testing. You'll hand in an explanation of what the errors are, and how you fixed them, but you will not hand in the corrected code.

1. Write white-box and black-box tests for the function `medianOfTwoLists()`. You should do this without Python, either on paper, or in a simple document. Don't worry about running your tests until you have thought about which tests you want. Make sure you have at least 3 black-box tests, and 3 white-box tests, as a minimum. More tests may be useful.
2. Implement your test cases in a document named `a9q3_testing.py`. You can use either plain if-statements (as demonstrated in readings 15.2.4) or a list-of-dictionaries to implement your test driver (as demonstrated in class during chapter 15 exercise 3).
3. Run your tests, and copy/paste the console output to a document named `a9q3_output.txt`. The console output you hand in should come from the program before you try to fix it!
4. Try to deduce the problems with the `medianOfTwoLists()` function from the output of your testing. Then fix the errors in the function, so that your test driver does not indicate any faults.
5. In the document `a9q3_output.txt`, describe the error(s) you found, and how you fixed it (them). You do not need to write a lot; a sentence or two for each error is all we want. Write your explanation as if you are explaining the error to a colleague of yours who wrote the function.

However, since there are two ways when you writing the text driver:

- you can copy/pasting the code of function of `medianOfTwoLists()` into the test driver file. In this case, you are allowed to include (and hand in) the code of function `medianOfTwoLists()` either fixed or unfixed.
- you can also use the `import` statement to include the function in the test driver. In this case, you are **NOT** need to hand in the fixed code.

What to Hand In

Remember, for this question, you do not need to hand in the fixed code!

- A document called `a9q3_testing.py` containing your testing code as described above.
- A document called `a9q3_output.txt` (rtf and pdf formats are also allowable) with the console output of your testing copy/pasted into it, and the explanations of the errors you found.

Be sure to include your name, NSID, student number and instructorâĂŹs name at the top of all documents.

Evaluation

- 3 marks: Your test-driver for `medianOfTwoLists()` contains at least 6 test cases.
- 2 marks: The output of your test driver reveals faults in the given function.
- 4 marks: You correctly identified four errors in `medianOfTwoLists()`, and explained briefly how you fixed them.
- **-1 mark if the student did not include their name, NSID, student number and instructorâĂŹs name at the top of the submitted file.**