Assignment 8

Testing and Debugging

Date Due: Friday, November 22, 6:00pm **Total Marks: 26**

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 Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle 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: Moderate

In digital advertising, data is extremely important to measure success of a campaign. In this assignment, we are going to calculate two commonly used digital advertising measurements, write test cases, and create a test driver for those test cases. The equations for the measurements are below:

eCPM (effective cost per mille)

$$eCPM = (\ Total\ Ad\ Spend\ /\ Total\ Ad\ Impressions\ ) * 1000$$

ARPDAU (average revenue per daily active user)

$$ARPDAU = (\ Revenue\ /\ Active\ Users\ /\ Days\ )$$

Write the following functions:

- `calculate_ecpm(ad_spend, ad_impresions)`
  takes 2 **integer** parameters and returns a **float** representing calculated **eCPM**. Your function should return 0.0 for incorrect input.
  For Example:

```
> calculate_ecpm(152, 1013)
> 150.05
```

- `calculate_arpdau(user_list, revenue_list)`
  takes 2 **list** parameters and returns a **float** representing calculated **ARPDAU**. Both lists should be the same length and include numbers only. You will need to **sum** the values in each list. Your function should return 0.0 for incorrect input.
  For Example:

```
> calculate_arpdau([54,36,30,96,112,130],[40.05,15.11,16.07,26.90,79.68,114.12])
> 0.1062
```

Inside both functions, use the **round()** method to limit the number of decimal places shown in the results. **calculate_ecpm** should round to 2 decimal places while **calculate_arpdau** should round to 4 decimal places. An example of using **round()**:

```
> round(1.259483,2)
> 1.25
```

To show that your functions are working correctly, implement a test driver which thoroughly tests both functions with a variety of white-box tests (at least six per function). 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 `a8q1.py` containing your functions and testing code as described above. Your functions and test driver should be written and submitted in the same file.

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

Evaluation

- 2 marks: Your function `calculate_ecpm()` takes two integers and correctly calculates the eCPM.
- 1 mark: The docstring for `calculate_ecpm()` is sufficiently descriptive, indicating its purpose and requirements.
- 3 marks: You designed at least six appropriate white-box test cases and implemented a test driver that would reveal faults in `calculate_ecpm()`.
- 2 marks: Your function `calculate_arpdau()` takes two lists as input and correctly calculates the ARPDAU.
- 1 mark: The docstring for `calculate_arpdau()` is sufficiently descriptive, indicating its purpose and requirements.
- 3 marks: You designed at least six appropriate white-box test cases and implemented a test driver that would reveal faults in `calculate_arpdau()`.
- 2 marks: Your chosen test cases show you have a clear understanding of white-box testing.

Question 2 (12 points):

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

Degree of Difficulty: Moderate

In the provided file `a8q2_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 `a8q2_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 `a8q2_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 `a8q2_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 `a8q2_testing.py` containing your testing code as described above.
- A document named `a8q2_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.