Guidelines on

Designing Test Harnesses

Here are a few guidelines to help you improve the quality of your tests;

- Make sure you're your resulting test suite is providing **exhaustive coverage**. His means that every path of execution in your program needs to be executed during the tests. E.g.
    - The else part of a selection statements should not be the one always executed in all your tests. That would prevent you from ever knowing if the other part of the selection statement is buggy.

- For each path of execution, make sure that your tests are **stressing** enough to catch as many possible implementation bugs as feasible. This means not only catching the bugs you had or have in your own code but anticipating what might be a situation which would fail another students' solution. E.g. don't forget to test "extreme cases"
    - Enter a sentinel of -1 as the first grade in a grade-average program and making sure there are no divisions by zero
    - Provide input which prevents entering a loop at all
    - Provide input which makes a program exhaust all tries it allows a player

- Your suite of tests also needs to be **minimal**; avoid redundant tests which are not bringing anything more. E.g.
    - If you're testing a program displaying the largest of two scanned integers, it is useless to test both {42,99} and {5,9}.
    - What matters is to test whether the program is able to find the largest number when it is the first, the second, or when both are equal.

- Make sure you **comment** your tests to justify them; that will probably help you eliminate redundancy and realize what you are not testing.

- You are not expected to perform **robustness** testing of your program vs. badly formatted user input. E.g.
    - If you program prompts the user for an integer value, you will not have to test what happens if you enter text.
    - We assume the user follows the directions provided by the program.