Week 1 Practice Quiz

1. Design and prototype walk-throughs with users are examples of:

   - Verification
   - Validation *{We are soliciting feedback from users to see if our ideas about the system match their expectations, so this is a validation activity.}*

2. Unit tests mapping requirements (or User Stories in Agile development) to class behaviors are examples of:

   - Validation
   - Verification *{We are determining whether a system matches its requirements.}*

3. User interviews to see if requirements/User Stories match the user's expectations of how the system will perform are examples of:

   - Verification
   - Validation

4. Refactoring is a technique that modifies the structure of software to improve the design (or some non-functional attribute such as performance) without changing the functionality of the software.

   Regression testing after refactoring the system to determine whether it behaves the same as the non-refactored version is an example of:

   - Validation
   - Verification

5. Which of the following are strengths of testing? (Check all that apply)

   - It checks the whole system, including software that you didn't write.
   - It can conclusively determine whether the software is correct.
   - It documents system behavior.

On the difficulty of software testing

1. We said that software testing is different than testing in many disciplines because software is discontinuous. In mathematics, continuity means that if you know the value of a function at one point, you can make claims about its value at points nearby. Why does this matter for testing?

   - There is no way to test software thoroughly.
   - Computer systems are easier to test than continuous systems because continuous systems have an infinite number of points to test while software is finite.
   - Continuous systems are simpler to test because you can often extrapolate test results from one point to points 'nearby'.

2. We said that the Zune needed tests for boundary conditions. These are values that check that the boundaries of arithmetic or relational expressions (like ==, !=, <, >, <=, >=) are tested. What are examples of boundary condition tests for the following function?

```
1   int flipSome(int A , int N, int X)
2   {
3     int i=0;
4     while (i<N)
5     {
6       if (A<X)
7         A = -A;
8       i++;
9     }
10    return(1);
11  }
```

- A = 1, X = 10, N=1
- A = 1, N=5, X=5
- A = 5, X=5, N=5

3. Is testing a (primarily) optimistic or pessimistic verification technique?

- Pessimistic
- Optimistic *{That's correct! Testing only checks certain program paths, so it is (usually) optimistic. Testing in very rare cases can be pessimistic, but only if the test outcome is incorrectly defined.}*

4. Is it always possible to prove whether any program meets its requirements in all cases, given enough effort?
   - Yes
   - No

What is a Test?

1. An automated oracle is:

   - A program that automatically generates test-cases.
   - A program that automatically matches the actual program output to the expected output of the test case.
   - A program that automatically generates input test data.
   - All of the above.

2. The setup step for a test case is important to establish a specific state for the program to run the test.
   - True
   - False

3. Tasks that can be part of the tear down phase are (choose two answers):
   - Initialize test case values
   - Remove data you added after testing is done
   - Open connection for testing
   - Close connection after testing is done

4. Assessment is the phase where:
   - Software output is checked by an oracle.
   - Test cases are run.
   - Test data is initialized.

   {That is right. Part of the assessment is to compare the output of the software under test with the expected output. This can be done manually, with a human checking that the result meets his/her expectations, or automatically, with an oracle.}

Automation: Using a Test Framework

1. Testing execution frameworks (e.g., JUnit) are important because:
   - They allow automated checks against an oracle (e.g., Do the outputs of the program match expectations?) to determine whether or not a test passes. *{That is right! A testing framework is a tool that allows testers to create test cases and execute them against the program (called the system under test or SUT). During test execution, the framework automatically checks that the output generated by the SUT conforms to the expected output, providing feedback to the tester on test case success or failure.}*
   - They run all test cases and provide feedback on which test cases passed and which failed.
   - They generate tests for you.
   - They allow programmers to unit test each method.
   - Correct

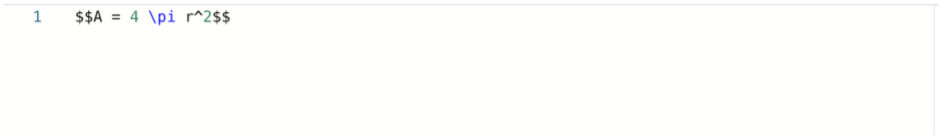
2. To test a main method you need to:
   - Redirect input to be entered by the test case.
   - Redirect output to be generated by the test case.
   - Invoke the main method with the right parameters.

3. In a test framework, we write test cases:

   - Inside the executed method and we annotate that this part is for testing.
   - Inside the class to be tested; we annotate that this part is for testing.
   - In a separate class---usually, with one or more test classes per class under test.
   - All of the above.

Automation: Writing JUnit Tests

1. Given a program that calculates the surface area of sphere A using the equation below where r is the radius of the sphere and is provided by the user, which of the following are valid test cases?



   - input r = 1/2, expected output = 3.14
   - input r = -1/2, expected output = invalid input
   - input r = 0, expected output = invalid input
   - input r = 30, expected output = 11309.73
   - input r = -1, expected output = invalid input

2. Redirection of system input and output for the Coffee Maker example is done in order to test:
   - The main method.
   - Any method for a domain class (Coffee Maker, Recipe, etc.)
   - All the domain classes in the system.

3. Testing is all about corner cases.
   - True.
   - False. *{That is right! While it is important to test corner cases, we must also check that the system functions correctly under normal use. Test cases that check the normal use of the system (called nominal test cases) are very important because they check the code that most users will regularly execute.}*

4. The strategy of writing test cases before implementation is:
   - A bad idea. Implementation needs to be there to know which test cases you should include.
   - A good idea. Test cases should be built based on the expected output and should not be influenced by the implementation.

5. When you have a new test case:
   - You need to run it alone to check that it has passed. There is no need to run other, older test cases since they previously passed and they may take a while to rerun.
   - All test cases need to be run including the new one.

{True! You need to run all the test cases even though they might have passed on a previous step because the new test case might cause other test results to change. This occurs, for example, when a test case doesn't correctly clean-up after itself causing structures to be reused across tests. This can be masked in previous test executions, but that could change with the addition of the new test case causing a test to fail. Therefore, all tests need to be rerun when a new test case is added to the test suite.}