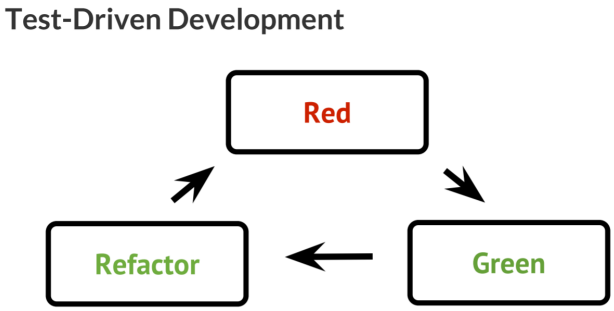
Part 3: Unit testing and Test Driven Development

In this part, use **Cyber-Dojo** to solve programming exercises. Solve at least *two* programming tasks with varying degrees of difficulty. Start with an easy task, e.g. *Bowling game*, *Closest To Zero*, or *Remove Duplicates* , and then choose one that is more difficult. (Do not choose *Leap Years*, *FizzBuzz*, or any examples reviewed in the lecture or seminars/group sessions.)

It is important that you follow the principles of **TDD**, with iterations of Red, Green, and Refactor:



1. List the requirements of the program
A key benefit of TDD is that it makes the developer focus on requirements before writing code. This is in contrast with the usual practice, where unit tests are only written after code.

2. Write a test for an item on the list
Write an automated test that would pass if the requirement is met.

3. Run all tests. The new test should fail – for expected reasons
This shows that new code is actually needed for the desired feature. It validates that the test harness is working correctly. It rules out the possibility that the new test is flawed and will always pass.

4. Write the simplest code that passes the new test
Inelegant code and hard coding is acceptable. The code will be honed in Step 6. No code should be added beyond the tested functionality.

5. All tests should now pass
If any fail, fix failing tests with minimal changes until all pass.

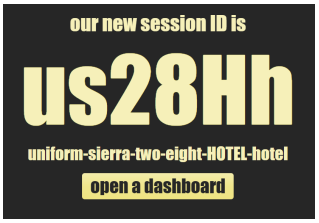
6. Refactor as needed while ensuring all tests continue to pass
Code is refactored for readability and maintainability. In particular, hard-coded test data should be removed from the production code. Running the test suite after each refactor ensures that no existing functionality is broken. Examples of refactoring:

- moving code to where it most logically belongs
- removing duplicate code
- making names self-documenting

- splitting methods into smaller pieces

Repeat the process, starting at step 2, with each requirement on the list until all tests are implemented and passing. Each test should be small, and commits made often. If new code fails some tests, the programmer can undo or revert rather than debug excessively. For more information, see the lecture slides.

The delivery of this part is the test-suite and the code developed for both programming exercises, totally four files. Do also add **screenshots of the session website** codes, or plain text of the two codes:



NB! You shall not use this concrete number, but your own sessions IDs.