1. Create list of potential tests
2. Start with the simplest test case
3. Use index cards to keep track of tasks and test cases, rip up when complete
4. Do the simplest thing to make it work even if the results are hard coded
5. Do a small amount of work and check in to source control
6. Tear up the card
7. Identify the next small item to work on
8. Allow API to be defined as you go, don’t design up front too much
9. Begin with the assertion and work backwards
10. Make small incremental changes
11. Red-green-refactor
12. Triangulate
13. Taking a break? Leave on a failing test so there is something to return to
14. One thing at a time, use index cards to capture distractions that can be addressed at a later point in time
15. Before implementing the production code behind a test, ensure you see the test fail first
16. Before committing view the diffs to ensure everything you want to commit will be committed
17. Strive for a single assertion per test
18. Use setup and teardown
19. Avoid integration with external systems such as disk, database, network and other processes
20. Keep tests light weight
21. Ensure tests are not dependent on the order of their execution
22. Ensure tests names reflect the use case
23. Check code coverage, which does not reflect on the quality of tests
24. Avoid static code because it is difficult to test and does not facilitate dependency injection
25. Write tests to expose existing bugs
26. Write once, duplicate, duplicate again and then refactor
27. Duplication comes in two flavors: code duplication and data duplication
28. Use tests to drive the creation of the interface you wish you had
29. Strive for low coupling and high cohesion
30. Commit often
31. Test:
    1. Conditionals
    2. Loops
    3. Operations
    4. Polymorphism
32. Test bugs in 3rd party libraries such that when an upgrade fixes the bug, the test breaks so you can be alerted to the fact and then remove any work arounds in place
33. Ensure all existing tests are passing before adding new tests or refactoring existing code
34. Work in small tiny increments
35. Start with broken code, that is code where the classes don’t exist
36. TDD rules:
    1. Add a little test
    2. Run all tests and fail
    3. Make a little change
    4. Run the tests and succeed
    5. Refactor to remove duplication
37. Apply the two patterns:
    1. Fake It
    2. Use Obvious Implementation