Test-Driven Development
Best Practices

Always Do the Next Simplest Test Case

- Doing the next simplest test case allows you to gradually increase the complexity of your code.
- If you jump into the complex test cases too quickly you will find yourself stuck writing a lot of functionality all at once.
- Beyond just slowing you down, this can also lead to bad design decisions.

Use Descriptive Test Names

- Code is read 1000 times more than it’s written. Make it clear and readable!
- Unit tests are the best documentation for how your code works. Make them easy to understand.
- Test suites should name the class or function under test and the test names should describe the functionality being tested.

Keep Test Fast

- One of the biggest benefits of TDD is the fast feedback on how your changes have affected things.
- This goes away if your unit tests take more than a few seconds to build and run.
- To help your test stay fast try to:
  - Keep console output to a minimum. This slows things down and can clutter up the testing framework output.
  - Mock out any slow collaborators with test doubles that are fast.

Use Code Coverage Tools

- Once you have all your test cases covered and you think you’re done run your unit test through a code coverage tool
- This can help you identify any test cases you may have missed (i.e. negative test cases).
- You should have a goal of 100% code coverage in functions with real logic in them (i.e. not simple getters/setters).
- Istanbul is easy to install (npm install —save-dev nyc) and can generate an easy to use html output.

Run Your Tests Multiple Times and In Random Order

- Running your tests many times will help ensure that you don’t have any flaky tests that fail intermittently.
- Running your tests in random order ensures that your tests don’t have any dependencies between each other.
- The “choma” plugin for Mocha provide randomization of the execution order of the tests in Mocha.

Use a Static Code Analysis Tool

- Static code analysis is a core requirement for ensuring code quality.
- JSHint is an excellent open source static code analysis tool that will find errors in your code that you may have missed in your testing.
- JSHint can verify your javascript code meets your team’s coding standard.