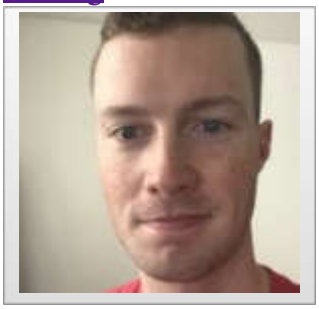
codecademy / PRO

Learn
Catalog



Build Websites from Scratch
Introduction and Setup
7 / 7
Unit 1: HTML (5 days)
15 / 15
Unit 2: CSS (6 days)
20 / 20
Unit 3: Command Line and Git (7 days)
17 / 17
Unit 4: Display and Positioning (16 days)
22 / 22
Unit 5: Responsive Design and Browser Compatibility (12 days)
17 / 17
Unit 6: CSS Documentation and Debugging (3 days)
5 / 5
Unit 7: Design and UI Feedback (7 days)
21 / 21
Unit 8: Capstone Project - Colmar Academy (7 days)
10 / 11
Unit 9: Next Steps - Bootstrap, JavaScript, and jQuery (7 days)
9 / 15
Overview
Feedback
Day 1
Lesson: Introduction to Bootstrap
Day 2
Lesson: Introduction to JavaScript
Day 3
Lesson: Introduction to jQuery
Days 4, 5, 6, and 7
Article: Test-Driven Development
Project Overview
Project: Match Game
Project: Trackster
Project: Jumpstart
Congratulations!
Article
Test-Driven Development
Reading Time: About 5 minutes
Mark Complete

Web developers have different workflows for creating and maintaining projects. For example, when implementing responsive design, some developers will program "mobile-first" by starting with the mobile view and scaling up, while others will start with the desktop view and update as necessary on smaller and smaller screens.

In this article we will discuss a very popular development workflow called *test-driven development* (TDD). TDD allows developers to take time to specify what their projects should be able to do before writing any code. They then work their way through each feature and implement them one at a time, using the tests to verify that each feature works the way it was intended to.

What are tests?

A *test* is a piece of code that checks to see if project code can complete a specific task. For example, a Go Fish app should have a test that makes sure that each player starts with seven cards. This test will start the game and check each player's hand. If a player doesn't have seven cards, the test will fail.

When a test fails, it returns a message to the developer explaining why that test failed. In the case of our Go Fish test, the test would return something like, "Each player should start the game with seven cards." A good test will describe the situation with the word "should", making it clear which part of your code is not working as intended.

In test-driven development, developers create these "should" statements for each feature of their project. They then write test code for each statement. Once all the tests are set up, developers will begin writing the code to make them pass.

A typical TDD workflow will look like this:

1. Run tests and select a failing test to fix.
2. Write code that should pass that test.
3. Run tests again. If selected test still fails, keep attempting to fix until test passes.
4. Select a new failing test and start over.

There are many benefits to using TDD:

- TDD causes developers to focus on what their app should do rather than coding without planning.
- TDD ensures that adding new features doesn't break old ones (old tests will fail if their corresponding features get broken).
- TDD makes it easy to identify where bugs have appeared in code through clear error messaging.

How to Run Tests

Let's try out TDD by running tests and fixing code on your own computer.

1. Download and unzip the project located here.
2. Open *index.html* in Google Chrome and open the JavaScript Console in Chrome DevTools (located in View > Developer > JavaScript Console).Verify that you are in the "Console" tab.
3. In the console, enter the command to run the tests for this project, `Tests.runAllTests()`. As you can see, there are multiple failing tests. Let's check them out.
4. Open the project in a text editor. Open the *tests.js* file. Looking at `Tests.runAllTests`, we can see that it runs through all of our tests — in this case `Tests.testReturnsTrue`, `Tests.testReturnsHello`, and `Tests.testReturnsTheMeaningOfLife`. Each test runs its corresponding method and checks to see if the method returns the expected value. Since the tests are failing, we can infer that they are not implemented properly. Let's look at the code.

5. Open *main.js* and examine the `Main.returnsTrue` method. As you might notice, this method returns `false`. Change `return false;` to `return true;`.
6. Since we believe we have now fixed the method, go back to Chrome and re-run `Tests.runAllTests()`. The 'returnsTrue' test should now pass!
7. Follow the same process for the `Main.returnsHello()` method. Fix what you believe is causing the bug and re-run the tests to see if the 'returnsHello' test now passes. Iterate until the test passes.
8. Once you have fixed the `.returnsHello()` method, fix the `.returnsTheMeaningOfLife()` method. This one is a little trickier since it actually has two failing tests. One fix should solve both of them though. Iterate until you get all of the tests to pass.

Congratulations! You now have a working test suite for all of these methods. As you can imagine, when using TDD to produce full web pages there will be many more tests to run and methods to write/fix. However, the steps will always remain the same: run, implement, and run again. Making TDD a permanent part of your development workflow will help ensure your code always works as intended and gives you incentive to keep implementing features!

Mark Complete