Week 6: Dev Tools, Debugging, & Unit Tests

Agenda

- Introductions
- Questions?
- Dev Tools
- Unit Tests
- Test Driven Development
- Next Week: HTML

What questions do you have?

The Muscles of Programming

1. **Researching** - the ability to google & read documentation when you don't know how to do something, find the answer, and apply it to your situation
2. **Writing** - the ability to break a problem down, plan things out, and turn it into code
3. **Debugging** - the ability to find and fix problems in your code
4. **Reading** - the ability to read code (someone else's or your own) and tell what the computer will do at each line
5. **Testing** - the ability to discover problems in your code and/or confirm that it works correctly in all possible ways

What makes you a developer is not what you know, it's **what you can do**.

Knowing the syntax and terminology and concepts is an important foundation to all these abilities. But that **knowledge** will not make you a successful developer.

You become a developer as you **practice with these muscles** and slowly make them stronger and stronger.

In the time you spend working to become a developer, make sure what you're focused on is **building muscle**, not just collecting knowledge.

Dev Tools

Elements

Debugging HTML & CSS

Console

Debugging Javascript

Checking for exceptions, logs, running code

Sources

Debugging Javascript

Breakpoints, stepping through line-by-line

Network

Debugging AJAX requests

Debugging Tips

- Make sure you're actually looking at the latest version of your code in the browser. Don't assume you are!
    - Particularly if you're working with alerts, you have to completely close out the alert for the browser to load the latest version.
    - Check that you saved your files after you made the change. If there are any white dots next to file names then they have not been saved.
    - Sometimes the browser just doesn't pull the latest version even if it seems like it did. You can force a hard reload in Chrome by holding Ctrl and clicking the refresh button.
- Check the console for errors
    - Don't just check when the page loads in, check if any pop up as you're using your app
    - If you don't understand the error, Google the first 5-10 words of it. Sometimes the same error can be caused by many different issues, so you may need to add details to your Google search to get relevant results. You also may need to dig through many stack overflow answers to find one that applies to you.
- Use the sources tab to step through your code and check your variables at each step to see where it goes wrong.

Researching Tips

- Include the **name of the language/library** you're working with (ex: "for loop javascript", "chai expect not equal")
- Take note of the **terms** that are used to describe programming concepts. Not necessarily the super technical ones, but the ones that are used frequently by other programmers ("array method", "declare a variable", "pass to the function")
- Hone the ability to express what's going wrong with your code **specifically** ("the button doesn't work" vs "the function isn't being called")
- Check the **dates** of answers and articles to make sure they aren't out of date. What is out of date will depend on the last time there was a major update to that language/library (ES6 came out in 2015 and became mainstream a year or two after)
- Look at the **comments** on Stack Overflow answers and articles. It's not true just because someone said it online, and comments often point out issues or helpful additions

Unit Tests

A test for a **small part** of the code (often one function/method/class)

Make sure you're **testing for edge cases** and weird cases - often using random testing helps catch problems you didn't think of

It allows you to build **knowing your foundation is solid**. If something goes wrong, you know it's in *how* you're using your parts, not in the parts themselves.

If **someone else needs to change a part you wrote**, they can rerun the test cases to make sure they didn't break anything.

Installing

```
npm init
npm install mocha
npm install chai
```

Mocha

Sets everything up with `describe()` and `it()`

Chai

Better testing of values with `expect()` and `should()` and `assert()`

AAA Pattern

Arrange

Set everything up
Create the testing data (objects, arrays)
Get all your variables ready

Act

Call the function or method you're testing

Assert

Check that it did what it should

Test Driven Development

Steps of TDD:

1. Write a unit test to test one small feature
2. Run the test and make sure it fails
3. Write the code for the small feature in the simplest way
4. Run the test and make sure it passes
5. Refactor to make it more efficient, readable, etc
6. Run the test and make sure it passes

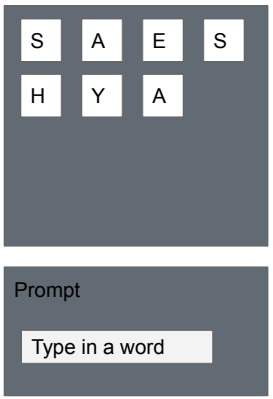
Test Driven Development

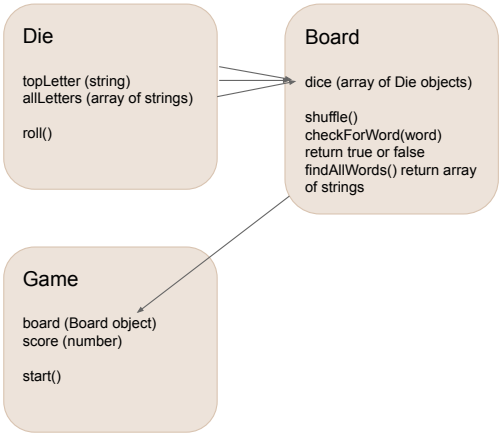
Benefits of TDD:

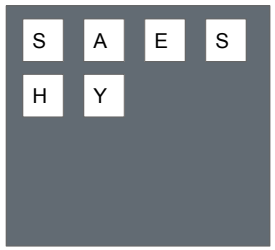
- **Encourages modular design** - each small feature is written on its own, then connected in
- **Easier to maintain and refactor** - you can just run the tests again to make sure you didn't break anything
- **The tests document the code** - each test gives examples of how that part of the code can be used

Drawbacks of TDD:

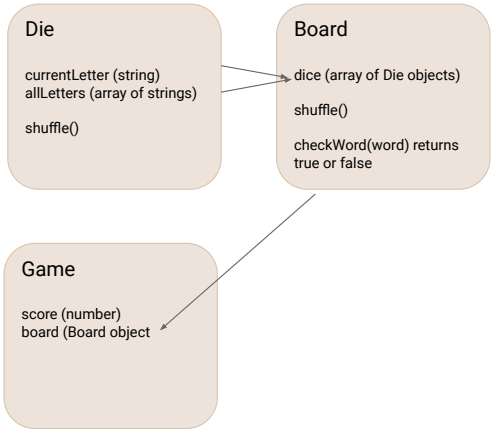
- **Takes longer** - Writing that many unit tests eats up a lot of time
- **Harder to change functionality** - Anytime you change functionality, you have to change all the tests that touch that functionality
- **It doesn't always work** - If you write the test and you write the code, you might just introduce the same bug into both of them







Type in a word



Additional Resources

Mocha

https://mochajs.org/

Chai

https://www.chaijs.com/

Jest

https://jestjs.io/

Short Video on Test Driven Development

https://www.youtube.com/watch?v=uGaNkTahrlw

Deeper Video on Unit Testing (uses C# example)

https://youtu.be/3kzHmaeozDI