Crash Course in Automated Testing

- Having a program run parts or all of code
- Confirm tested parts meet expectations
- Expected by almost all coding jobs
- Could be an entire course by itself!
    - We'll cover the most important highlights
    - But actual learning requires practice

Why Automated Testing?

Multiple Reasons. Values differ for different teams!

1. **Confirm code initially works** (least value)
    - Should really be done anyway
    - Initial tests + code likely same mistakes
2. **Functions as "real" documentation**
    - Anything not tested may not be true
3. **Confirms code works after changes**
    - Allows changes with confidence!
4. **Confirms code works in different environments**
    - Reduces bad deploys that "worked for me!"

The "Testing Pyramid"

1. Tip, few: **End-to-End** (E2E, UI testing)
    - Test code interacts with app like a user
    - Very Slow to run (5-30 mins common)
    - "Brittle" tests
2. Middle, some: **Integration Tests**
    - Test code calls combinations of code
    - Slow to run (1-5 mins common)
3. Bottom, many: **Unit Tests**
    - Test code tests individual "units"
    - Very Fast to run (0-3 seconds common)
    - Tests very durable

Testing Pyramid Caveats

- 3-5 layers, sometimes different names
    - Often based on area of coding
- Frontend Web Dev harder to make durable tests
    - Often more Integration Tests
- Writing test code takes time
    - May require changes when code changes
- MANY lengthy debates about
    - **What code to test**
    - **What about the code to test**
    - **How to write your tests**
    - **When to write your tests**

Automated Unit Testing

What is a **unit**?

- The smallest piece of usefully testable code
- Function, object, or module that provides a defined, indivisible, and self-contained functionality

Unit test will NOT involve other systems

- No database
- No web calls
- No network

Integration Testing

- Combines some units, tests combination
- Should NOT repeat the unit tests
    - Repeated code is just more to maintain
- Validates the *assumptions of* unit tests
    - Things unit tests couldn't test
    - But trying to minimize brittle E2E tests
    - Confirm integrations work given correct data
- Allows for external systems (database, web)
    - Often tries to focus on one at a time
- Not yet end-to-end

End-to-End Testing

- Runs test code against product as user
- Example: Controls browser, clicks/types as user
- Pro: Confirms app actual user experience (mostly)
- Con: Tests tedious to write/maintain
- Con: Tests slow to run
- Con: Tests very brittle
- Should *not* repeat unit/integration tests
    - Confirms if/when user sees expected results