TESTING IN RUST

- Language Features
- Rust Libraries for Testing
  - e.g. proptest, netsim
- Language-agnostic tools
  - e.g. Jepsen, TLA+

TESTING LANGUAGE FEATURES

- Unit Tests
- Integration Tests
- Performance Tests

THE "TEST" BUILD CONFIGURATION

- Build/run using "cargo test"
- *Not* built with "cargo build"
- Mark test-only code sections using the #[cfg(test)] attribute
- Only compiled in test configurations

UNIT TESTS

- To test a *single* module
- By convention appended to the same file to be tested
- By convention in a submodule named "tests"
- Simply mark a Rust function with the [#test] attribute
- Debug configuration by default (--release to override)
- Rust allows testing of private functions

INTEGRATION TESTS

- To test multiple modules in combination
- Located in a "tests" folder alongside the "src" folder
- No need to mark with the #[cfg(test)] attribute
- Tested crate needs to be imported
- Each integration test source file compiled as separate crate

TEST FILTER OPTIONS

- Run a specific unit/performance test specifying the test name
    - Name needs to include the complete module path
    - Module path alone runs all tests below that module
- Run a specific integration test using "--test "
- Use "--tests" option to only run unit tests

PERFORMANCE TESTS

- Requires the "test" feature and crate import

```
#![feature(test)]
extern crate test;
```

- Run with "cargo bench"
- Use the "#[bench]" attribute on the test function

PROPTEST

The best tests are those you do not need to write yourself. - Tyler Neely

- Based on Python's "Hypothesis" module
  - Which in turn is based on Haskell's "QuickCheck"
- Randomly checks over an input range
- Automatically reduces to a *minimal* test case
- Writes seed for random number generation in case of failure
  - Makes random failures repeatable
- Combine with traditional Unit Testing for edge cases

USAGE

- Import proptest crate with using_macros and test attributes

```
#[cfg(test)]
#[macro_use]
extern crate proptest;
```

- Then use the proptest! macro to define tests

EXAMPLE

```
proptest! {
    #[test]
    fn a_proptest(a in (0i32..100),
                  b in (0i32..100)) {
        assert!(a * b <= 10000);
    }
}
```

STRING PATTERNS

- proptest supports more advanced range patterns
  - patterns extensible as "Strategies"

```
#[test]
fn parses_all_valid_dates(s in "[0-9]{4}-[0-9]{2}-[0-9]{
    parse_date(s).unwrap();
}
```