Maintainable Code

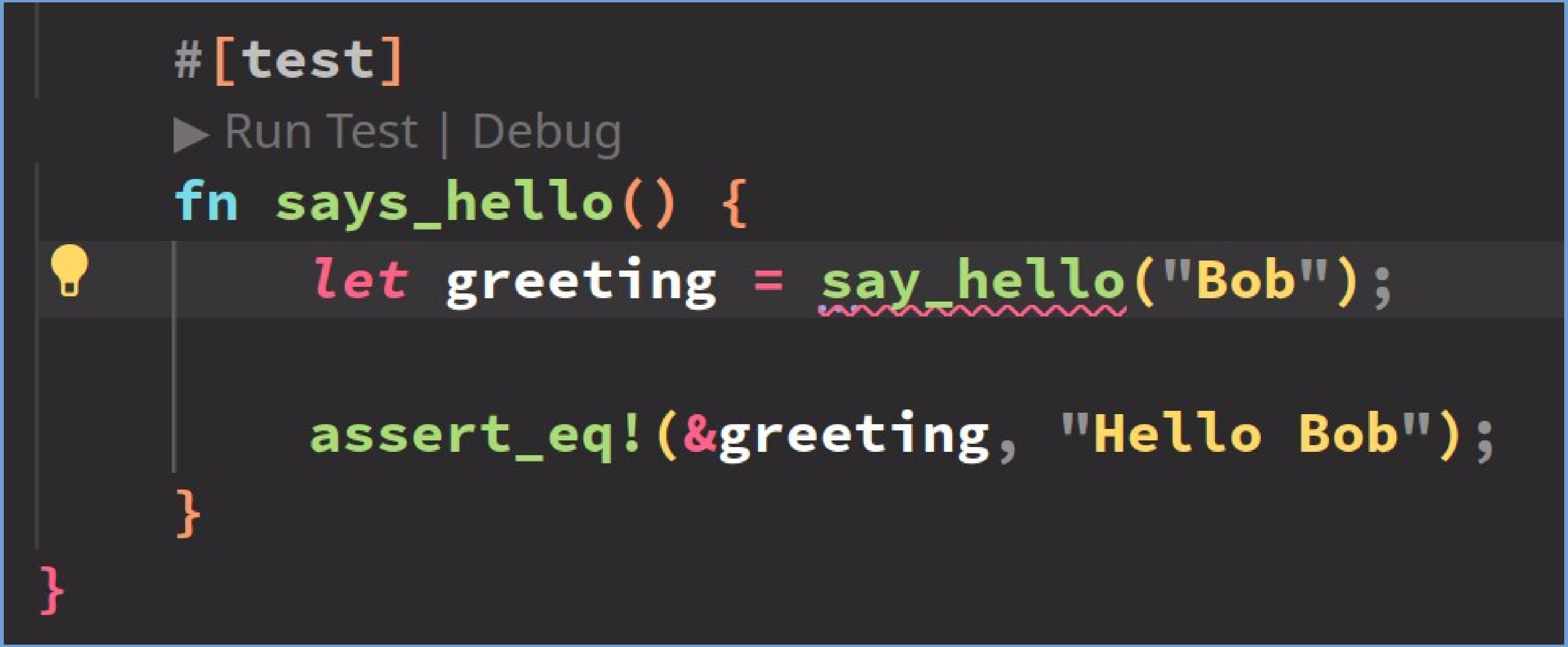
Red-Green-Refactor

Red-Green-Refactor

- Method of writing tests

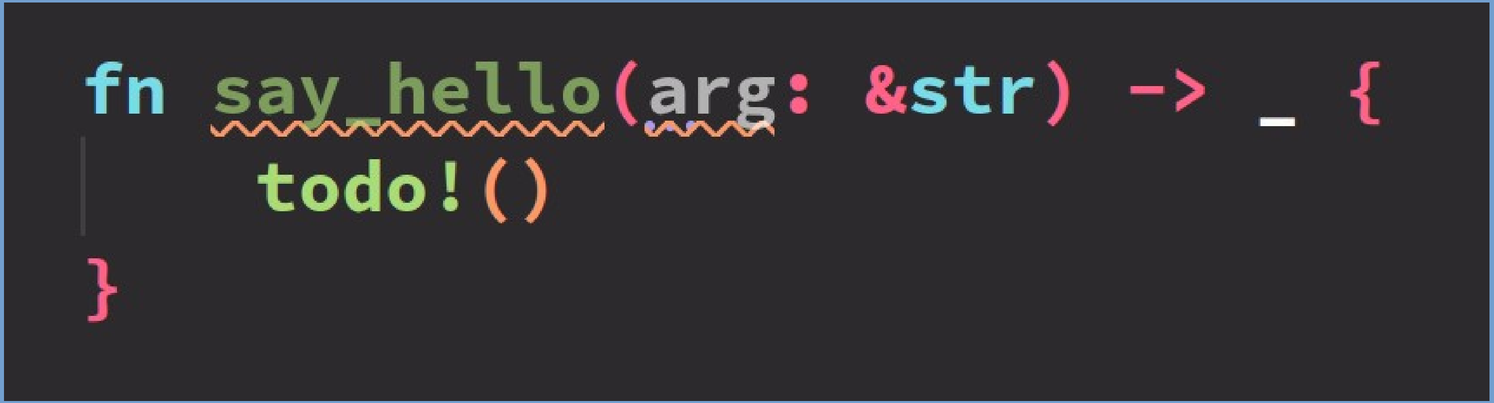
1. **RED**
   - Write the test ***then run it*** so it fails
2. **GREEN**
   - Implement the feature so it passes the test
3. **REFACTOR**
   - Cleanup implementation code
   - Reduce test duplication (DRY)

Process: RED (1)

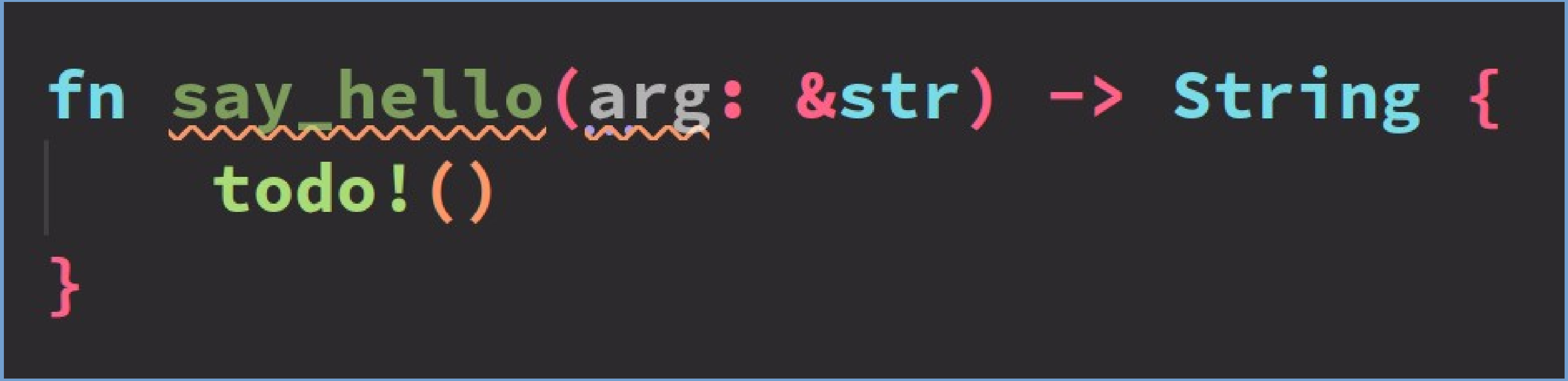


```
    #[test]
    ▶ Run Test | Debug
    fn says_hello() {
        let greeting = say_hello("Bob");

        assert_eq!(&greeting, "Hello Bob");
    }
}
```

Process: RED (2)

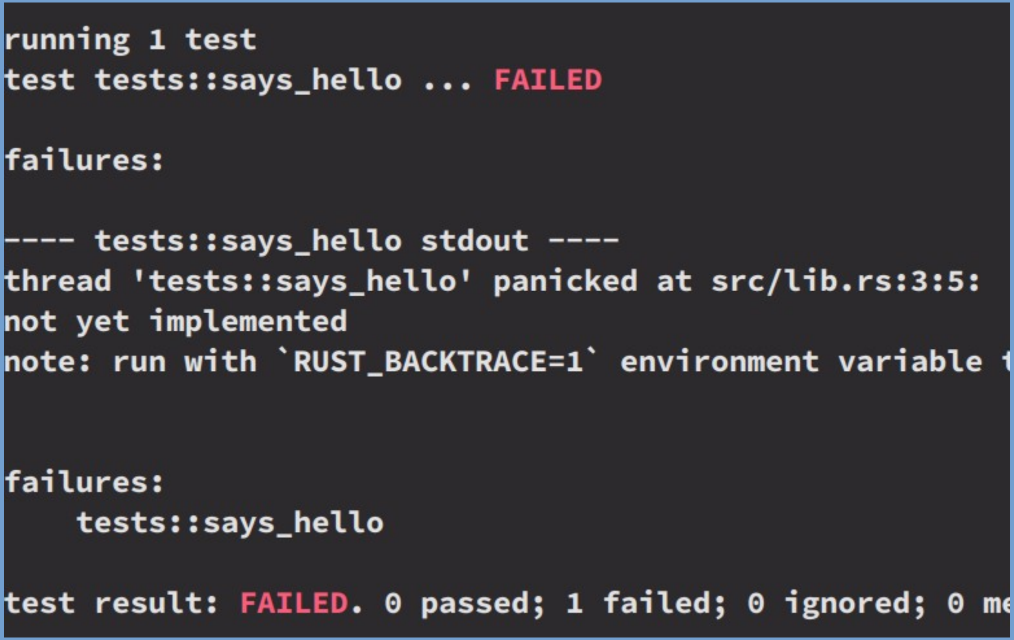

```
fn say_hello(arg: &str) -> _ {
    todo!()
}
```

Process: RED (3)


```rust
fn say_hello(arg: &str) -> String {
    todo!()
}
```

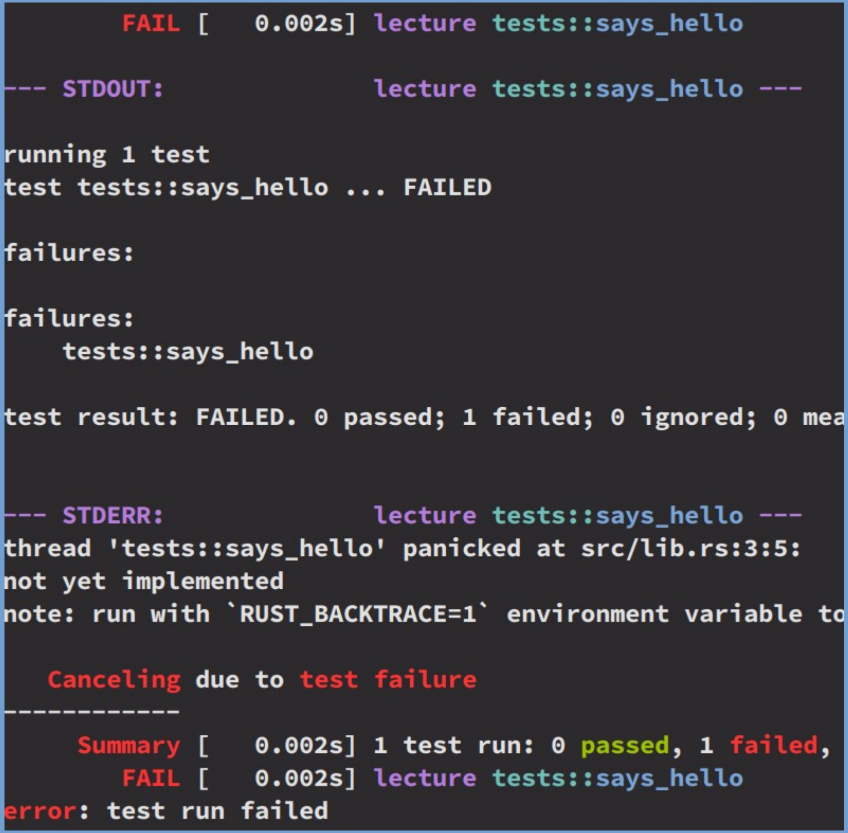
Process: RED (4)

cargo test



```
running 1 test
test tests::says_hello ... FAILED

failures:

---- tests::says_hello stdout ----
thread 'tests::says_hello' panicked at src/lib.rs:3:5:
not yet implemented
note: run with `RUST_BACKTRACE=1` environment variable t


failures:
    tests::says_hello

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 me
```

cargo nextest run



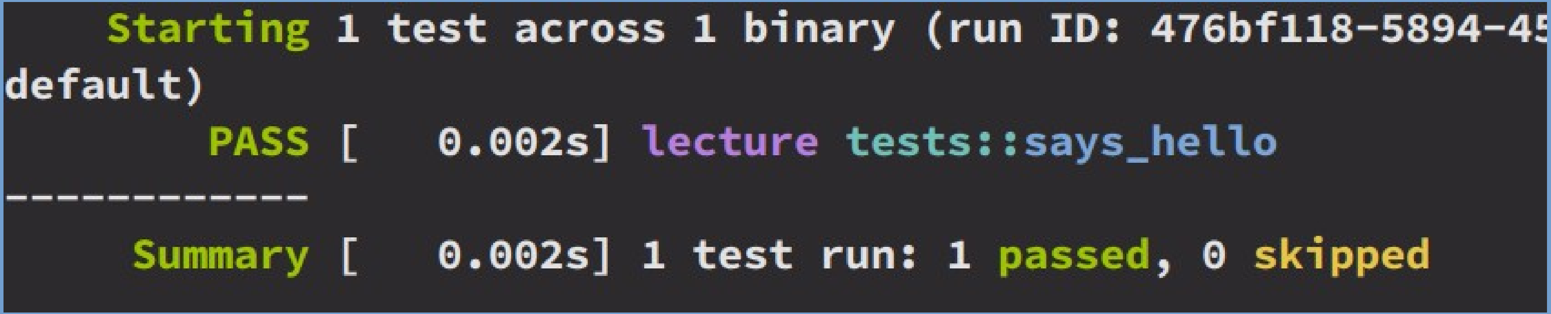
```
        FAIL [   0.002s] lecture tests::says_hello

--- STDOUT:              lecture tests::says_hello ---

running 1 test
test tests::says_hello ... FAILED

failures:

failures:
    tests::says_hello

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 mea


--- STDERR:              lecture tests::says_hello ---
thread 'tests::says_hello' panicked at src/lib.rs:3:5:
not yet implemented
note: run with `RUST_BACKTRACE=1` environment variable to

   Canceling due to test failure
------------
     Summary [   0.002s] 1 test run: 0 passed, 1 failed,
        FAIL [   0.002s] lecture tests::says_hello
error: test run failed
```

Process: GREEN (1)

```rust
fn say_hello(name: &str) -> String {
    format!("Hello {name}")
}
```

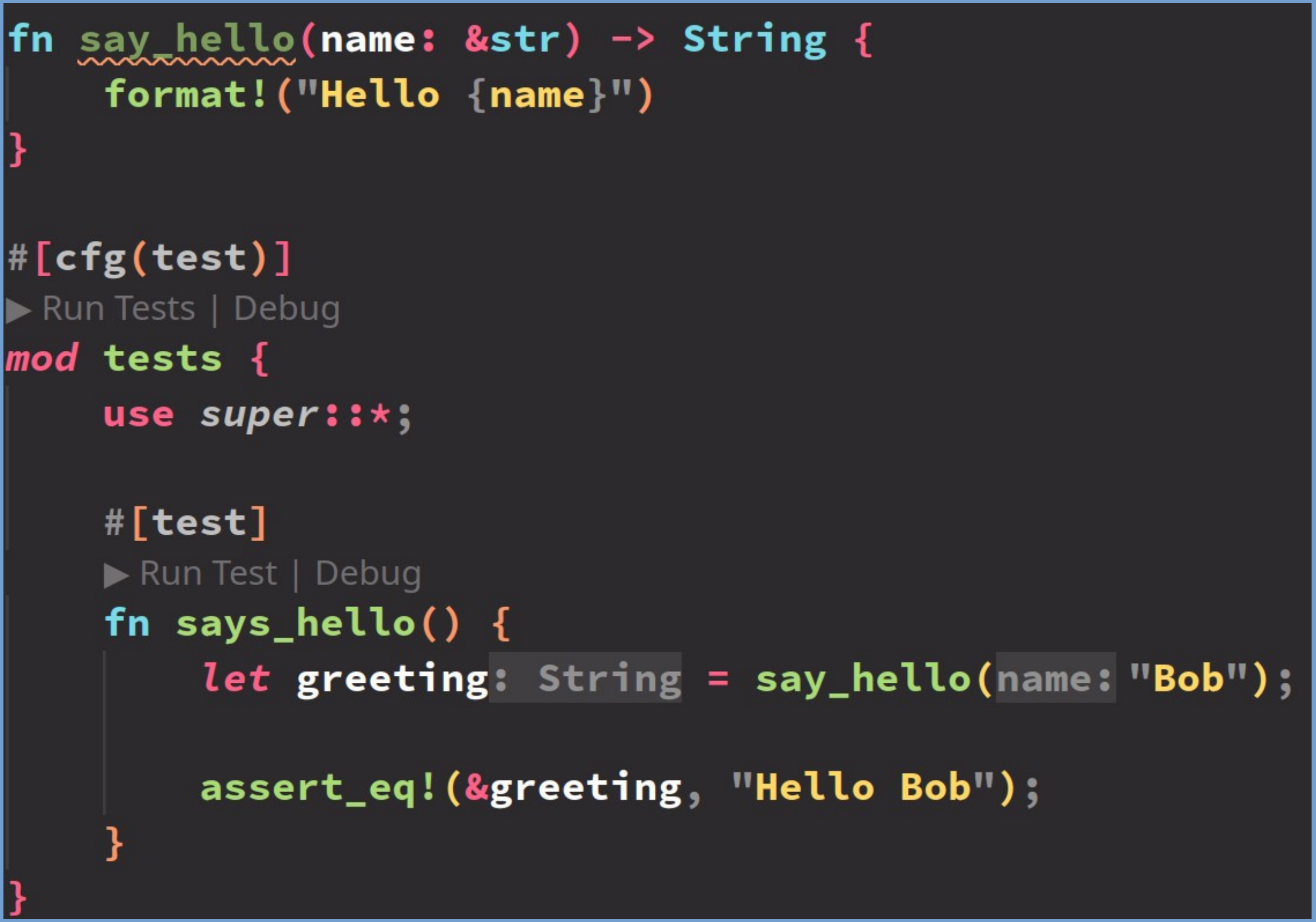
```rust
assert_eq!(&greeting, "Hello Bob");
```

Process: GREEN (2)



```
    Starting 1 test across 1 binary (run ID: 476bf118-5894-45
default)
        PASS [   0.002s] lecture tests::says_hello
------------
     Summary [   0.002s] 1 test run: 1 passed, 0 skipped
```

Process: REFACTOR (1)



```
fn say_hello(name: &str) -> String {
    format!("Hello {name}")
}

#[cfg(test)]
▶ Run Tests | Debug
mod tests {
    use super::*;

    #[test]
    ▶ Run Test | Debug
    fn says_hello() {
        let greeting: String = say_hello(name: "Bob");

        assert_eq!(&greeting, "Hello Bob");
    }
}
```

Supplemental tools

- **`cargo install cargo-nextest --locked`**
  - Run tests with:

    ```
    cargo nextest run
    ```

- **`cargo install --locked watchexec-cli`**
  - Automatically run tests when saving:

    ```
    watchexec --clear --debounce 200ms cargo nextest run
    ```

Recap

- Red-Green-Refactor ensures that your code is being tested
- Steps:
  1. Write one failing test
  2. Write passing implementation
  3. Code cleanup
- Prefer using `` `**cargo nextest run**` `` for tests
- Automatically run tests with `` `**watchexec**` ``