In this lecture, we will discuss...

♦ Unit testing



Why Write Tests?

- ♦ How do you know that your code works?
 - You really have no idea until you run it...
- How do you refactor with confidence that you didn't break anything?
- Serves as documentation for developers



Enter Test::Unit

- Ruby takes testing very seriously and ships with Test::Unit
- In Ruby 1.8 Test::Unit was bloated with extra libraries that included unnecessary code
- Ruby 1.9 stripped Test::Unit to a minimum

(The new framework is officially called MiniTest, but it's a drop-in replacement, so no changes are required to Test::Unit code)



Enter Test::Unit

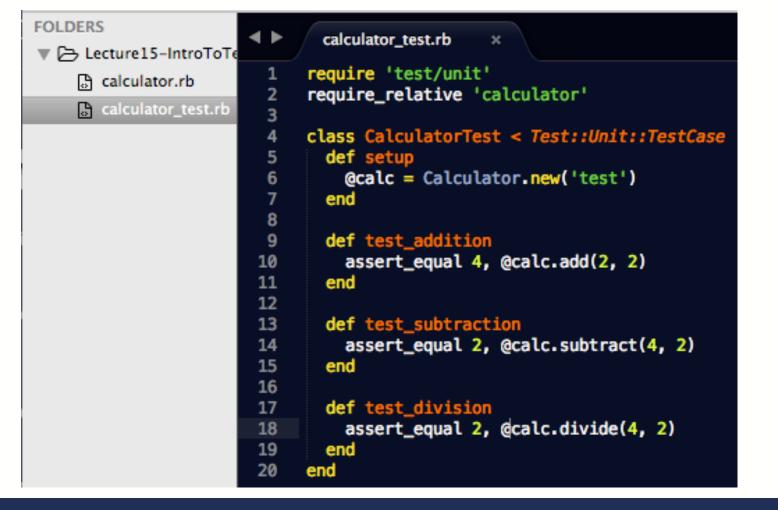
- ♦ Member of the XUnit family (JUnit, CppUnit)
- ♦ Basic Idea extend Test::Unit::TestCase
- Prefix method names with test_
- If one of the methods fails others keep going (this is a good thing)
- Can use setup() and teardown() methods for setting up behavior that will execute before every test method



calculator.rb

```
class Calculator
 attr_reader :name
 def initialize(name)
    @name = name
 end
 def add(one, two)
    one - two
 end
 def subtract(one, two)
    one + two
 end
 def divide(one, two)
    one / two
 end
end
```



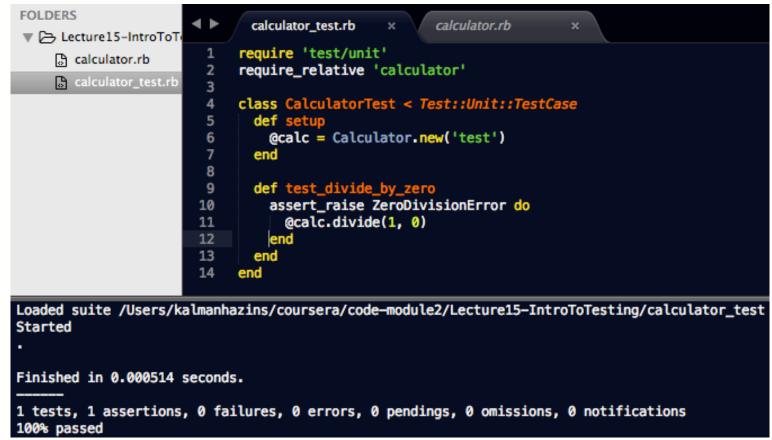




```
~/coursera/code-module2/Lecture15-IntroToTesting$ ruby calculator_test.rb
Loaded suite calculator_test
Started
Failure: test_addition(CalculatorTest)
calculator_test.rb:10:in `test_addition'
     7: end
      8:
     9: def test_addition
  ⇒ 10:
            assert_equal 4, @calc.add(2, 2)
    11: end
    12:
    13: def test subtraction
expected but was
Failure: test_subtraction(CalculatorTest)
calculator_test.rb:14:in `test_subtraction'
    11: end
    12:
    13: def test_subtraction
  ⇒ 14:
            assert_equal 2, @calc.subtract(4, 2)
    15: end
    16:
    17: def test_division
expected but was
Finished in 0.006305 seconds.
3 tests, 3 assertions, 2 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
33.333% passed
475.81 tests/s, 475.81 assertions/s
```



Testing Failures





Summary

- ♦ Assertions allow you to exercise your code
- Unit tests give you confidence to restructure/refactor your code

What's Next?

RSpec

