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

```ruby
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
```

FOLDERS

- Lecture15-IntroToTe
  - calculator.rb
  - calculator_test.rb

calculator_test.rb

```ruby
require 'test/unit'
require_relative 'calculator'

class CalculatorTest < Test::Unit::TestCase
  def setup
    @calc = Calculator.new('test')
  end

  def test_addition
    assert_equal 4, @calc.add(2, 2)
  end

  def test_subtraction
    assert_equal 2, @calc.subtract(4, 2)
  end

  def test_division
    assert_equal 2, @calc.divide(4, 2)
  end
end
```

```
~/coursera/code-module2/Lecture15-IntroToTesting$ ruby calculator_test.rb
Loaded suite calculator_test
Started
F
================================================================================
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
<4> expected but was
<0>
================================================================================
.F
================================================================================
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
<2> expected but was
<6>
================================================================================


Finished in 0.006305 seconds.
--------------------------------------------------------------------------------
3 tests, 3 assertions, 2 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
33.3333% passed
--------------------------------------------------------------------------------
475.81 tests/s, 475.81 assertions/s
```

Testing Failures

FOLDERS

Lecture15-IntroToT

calculator.rb

calculator_test.rb

calculator_test.rb | calculator.rb

```ruby
1  require 'test/unit'
2  require_relative 'calculator'
3
4  class CalculatorTest < Test::Unit::TestCase
5    def setup
6      @calc = Calculator.new('test')
7    end
8
9    def test_divide_by_zero
10     assert_raise ZeroDivisionError do
11       @calc.divide(1, 0)
12     end
13   end
14 end
```

```
Loaded suite /Users/kalmanhazins/coursera/code-module2/Lecture15-IntroToTesting/calculator_test
Started
.

Finished in 0.000514 seconds.
------
1 tests, 1 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
100% passed
```

Summary

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

What's Next?

RSpec