Unit Testing
Testing

• How do you know if your code is correct?

• Submit to AutoLab?
  • Does not exist outside of class
  • Does not exist for your project
Recall

• How do we test this function to verify that it's correct?
package example

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

  def main(args: Array[String]): Unit = {
    println(computeSize(70.0))
    println(computeSize(50.0))
    println(computeSize(10.0))
  }

  
  • Call the method from main
  • Print the results
  • Manually verify
}
What About Large Projects?

• There may be 100's of files and 1000's of methods

• Any change in a function might break any code that calls that function

• Will you manually verify all that code for each change?

• Unit Testing
  • Automate testing
  • Provide structure to testing
Unit Testing

- Run a series of tests on your code

- If the code is correct, all tests should pass

- If the code is incorrect, at least one test should fail

- A set of tests should test every possible error that could occur
Scala Unit Testing

```scala
package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Doubles are checked for size in each category") {
    val largeDouble: Double = 70.0
    val mediumDouble: Double = 50.0
    val smallDouble: Double = 10.0

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDouble) == "medium", mediumDouble)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}
```

Use Maven to download scalatest (see pom.xml on the course website)

Click Maven in the IntelliJ sidebar to interact with pom.xml
Scala Unit Testing

package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Doubles are checked for size in each category") {
    val largeDouble: Double = 70.0
    val mediumDouble: Double = 50.0
    val smallDouble: Double = 10.0

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDouble) == "medium", mediumDouble)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}

Import everything from the org.scalatest package

_ is a Scala wildcard
Scala Unit Testing

```scala
package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Doubles are checked for size in each category") {
    val largeDouble: Double = 70.0
    val mediumDouble: Double = 50.0
    val smallDouble: Double = 10.0

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDouble) == "medium", mediumDouble)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}

Create a new class of type FunSuite (Function Suite)

*More details on this syntax next week. This is inheritance
Create a new test that will be executed when this file is ran

No main method

FunSuite controls execution instead of main
Scala Unit Testing

package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Doubles are checked for size in each category") { 
    val largeDouble: Double = 70.0
    val mediumDouble: Double = 50.0
    val smallDouble: Double = 10.0

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDouble) == "medium", mediumDouble)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }

  }

Call assert to test values

First argument is a boolean that must be true for the test to pass
-Should return false if the code is not correct
Second argument is optional. Is printed if the test fails
Scala Unit Testing

package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Doubles are checked for size in each category") {
    val largeDouble: Double = 70.0
    val mediumDouble: Double = 50.0
    val smallDouble: Double = 10.0

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDouble) == "medium", mediumDouble)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}

This class tests if the inputs 70.0, 50.0, and 10.0 return "large", "medium", and "small" respectively

Is this enough testing?
Correct Solution

```scala
package example

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

}
```
Incorrect Solution
-Passes the tests-

package example

object Conditional {

    def computeSize(input: Double): String = {
        val large: Double = 65.0
        val medium: Double = 20.0
        if (input >= large) {
            "large"
        } else if (input >= medium) {
            "medium"
        } else {
            "small"
        }
    }

}
Scala Unit Testing

package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Size boundaries are checked"){
    val largeDouble: Double = 60.0
    val mediumDoubleUpperBound: Double = 59.99
    val mediumDoubleLowerBound: Double = 30.0
    val smallDouble: Double = 29.99

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDoubleUpperBound) == "medium", mediumDoubleUpperBound)
    assert(Conditional.computeSize(mediumDoubleLowerBound) == "medium", mediumDoubleLowerBound)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}

Check the boundaries for more accurate testing

Is this enough testing?
Scala Unit Testing

package tests

import org.scalatest._
import example.Conditional

class TestComputeSize extends FunSuite {

  test("Size boundaries are checked"){
    val largeDouble: Double = 60.0
    val mediumDoubleUpperBound: Double = 59.99
    val mediumDoubleLowerBound: Double = 30.0
    val smallDouble: Double = 29.99

    assert(Conditional.computeSize(largeDouble) == "large", largeDouble)
    assert(Conditional.computeSize(mediumDoubleUpperBound) == "medium", mediumDoubleUpperBound)
    assert(Conditional.computeSize(mediumDoubleLowerBound) == "medium", mediumDoubleLowerBound)
    assert(Conditional.computeSize(smallDouble) == "small", smallDouble)
  }
}

Check the boundaries for more accurate testing

Is this enough testing?

We could reasonable stop here.. but we could do more thorough testing
Use data structures to run many test cases
Unit Testing Objectives

- Each homework, and other places in the course, will have objectives that require thorough testing
- When these objectives are graded, your test suite is ran:
  - Against your solution
  - Against a correct solution stored on the server
  - Against a variety of incorrect solution stored on the server
- Your test suite should pass on both your solution and the correct solution
- Your test suite should fail on all the incorrect solutions
Maven: Dependency Management

- To run this testing code, we used an external library named Scalatest
  - Scalatest does not come with Scala
  - We must download it before running tests
- To manage external libraries, we'll use Maven
  - List all dependancies (libraries) in a file named pom.xml
  - Save pom.xml in the root directory of your project
  - Use Maven to download all dependancies
- The pom.xml is similar to the requirements.txt file we used in Python
Maven Demo