

# Comp 311

# Functional Programming

Nick Vrvilo, Two Sigma Investments  
Robert “Corky” Cartwright, Rice University

September 6, 2018

# Announcements

- Homework 0 is “due” today.  
*(You should now know how to use the SVN repo.)*
- Homework 1 assignment has been moved to Tuesday, and the due date will shift likewise.

# Class Methods

- Methods are functions defined in the body of a class definition. They have direct access to the members of a class instance
- Syntactically, they are placed between braces, after the class parameters

# Class Methods

```
case class C(field1: Type1, ..., fieldN: TypeN) {  
  def m1(x11: TypeP11, ..., xK1: TypePK1): TypeR11 =  
    expr  
  
  ...  
  def mJ(x1J: TypeP1J, ..., xKJ: TypePKJ): TypeR1J =  
    expr  
}
```

# Method Definitions

```
case class Coordinate(x: Int, y: Int) {  
  def magnitude() = x*x + y*y  
}
```

# Applying a Class Method

- Given a class definition:

```
class C(p1: T1, ..., pk: Tk) { ...  
    def m(param1: T11, paramN: T1N): T = e  
    ...  
}
```

- To reduce the application of a method:

$$C(v_1, \dots, v_k).m(\arg_1, \dots, \arg_N)$$

- Reduce the receiver and arguments, left to right
- Reduce the body of  $m$ , replacing constructor parameters with constructor arguments and method parameters with method arguments

# Applying a Class Method

`Coordinate(5,3).magnitude()`  $\mapsto$

$5*5 + 3*3$   $\mapsto$

$25 + 9$   $\mapsto$

34

# Compound Value Patterns

```
def dotProduct(c1: Coordinate, c2: Coordinate) = {  
  (c1, c2) match {  
    case (Coordinate(x1,y1), Coordinate(x2,y2)) =>  
      x1*x2 + y1*y2  
  }  
}
```



# Patterns in Assignments

Patterns in Scala may also be used for destructuring assignments:

```
def dotProduct(c1: Coordinate, c2: Coordinate) = {  
  val Coordinate(x1, y1) = c1  
  val Coordinate(x2, y2) = c2  
  x1*x2 + y1*y2  
}
```

# Symbols in Patterns: Binding or Constant?

- A symbol with a *lower-case* first character is a binding symbol
- A character with an *upper-case* first character is a value
- You can make a variable a constant using `backticks`

```
val pi = 3.14
```

```
val One = 1.0
```

```
expr match {
```

```
  case pi => "Pi"
```

```
  case One => "One"
```

```
}
```

# Singleton Objects

# Singleton Objects

- Also, we often would like to organize identifiers and functions together into a single entity
- When *compiling* a Scala file, it is *required* that all constant and function definitions are placed inside a class or object
- For this purpose, we can make use of *singleton objects*

# Singleton Objects

```
object IncomeTax {  
  
  val cutoff0 = 0  
  val bracket0 = 0  
  
  val bracket1 = 100  
  val cutoff1 = 9075  
  ...  
  
  def incomeTaxForBracket(income: Int, cutoff: Int, bracket: Int) = {  
    require(income >= 0)  
    (income - cutoff) * bracket / divisor + incomeTax(cutoff)  
  } ensuring (_ >= 0)  
}
```

# Syntax for Singleton Objects

```
object Name {  
    valDefs*  
  
    functionDefs*  
}
```

# We Can Refer to the Constants and Functions in the Object Using Dot Notation

`IncomeTax.bracket1`

`↳`

`100`

# We Can Refer to the Constants and Functions in the Object Using Dot Notation

`IncomeTax.incomeTax(100000)`

$\mapsto$

`21174`



# Case Objects

- Declaring a *case object* denotes your intent: You will use this object as a *value* and use it as a pattern in *match* expressions.
- Using a normal *object* denotes a container for “static” methods declarations, or a value that won’t be *matched*.

```
case object Name {  
    valDefs*  
    functionDefs*  
}
```

# Homework

# Homework Grading Criteria

- Style: 50%
- Correctness: 50%

# Style of Program Code and Test Code

- Clarity
- Comments
- Contracts
- Design Principles

# Clarity: Is the Program Easy to Read?

- Is the program concise?

*“Make every word say.”*

*(Strunk and White, *The Elements of Style*)*

- Are functions kept relatively small, with sub-parts broken up according to the problem domain?

Think of the *profit, revenue, and cost* example from Lecture 2

# Clarity: Is the Program Easy to Read?

- Are the names of functions and variables syntactically consistent?
  - For instance, do they all use camelCase?
  - Are similar functions given names of similar length?

# Clarity: Is the Program Easy to Read?

- Are names adequately descriptive and appropriate?
  - For example, using single letter names for public functions is not appropriate
- Are consistent metaphors used for functions that work together?

# Clarity: Is the Program Easy to Read?

- Is the program consistent in its indentation and whitespace?
  - This can affect readability
- Is there appropriate spacing?
  - Code that is too close together can be hard to read



# Comments

- Does each function include a statement of purpose?
- Are the comments excessive?
  - Comments embedded in program should be used only for cases where it is not clear locally why the program is doing what it does
  - The reader should be expected to know the language the text is written in

# Contracts

- Do the parameter types and return types of all functions and variables make sense?
- Are `require` and `ensuring` clauses included when necessary?
- Are the included `require` and `ensuring` clauses defined appropriately?
- Are requirements that cannot be expressed in `require` and `ensuring` clauses defined as documentation?

# Design Principles

- Does the program stick to the constructs covered in class so far?
- Is the program purely functional?

# Design Principles

- Does the program follow templates provided in class when appropriate?
  - For instance, is the function body a simple algebraic expression?
  - Is it a series of `if-else` expressions breaking up sub-ranges?
  - Is it a `match` expression breaking up an abstract datatype?

# Design Principles

- Does the program include abstractions to factor out common code? (DRY)
  - Copy-and-paste coding should be strongly avoided
- Does the program avoid unnecessary complexity? (KISS)

# Correctness

- Does the program compile?
- Do all student submitted tests pass?
- Does the program include all entry points required by the assignment?
- Are all tests automated? Tests should indicate on their own that either they pass or fail

# Correctness

- Example Tests: Are simple examples included in the tests showing how the function behaves under usually circumstances?
- Stress Tests: Are there additional tests ensuring that the function behaves appropriately when given extreme data values

0, 1, -1, PositiveInfinity,  
NegativeInfinity, NaN, etc.

# Correctness

- Persuasive Tests: Is there adequate coverage to convince the reader that the program behaves as expected?
- Does the program perform correctly when subjected to additional testing provided by the course staff?



# Expected Test Structure

- All tests in a program should be captured in a *test suite*
- For each component of a program, there should be a corresponding test class
- For each function, there should be a corresponding test function
- For each test function, there should be multiple tests, checking both common and extreme cases

# Example: Testing Our Theater Profit Calculator

```
class TheaterProfitTest(name: String) extends TestCase(name) {  
    def testAttendance() = {  
        ...  
    }  
    def testCost() = {  
        ...  
    }  
    def testProfit() = {  
        ...  
    }  
    def testRevenue() = {  
        ...  
    }  
    def testMax() = {  
        ...  
    }  
}
```

# Example: Testing Our Theater Profit Calculator

```
class TheaterProfitTest(name: String) extends TestCase(name) {  
  
  def testAttendance() {  
    assertEquals(120, attendance(500))  
    assertEquals(135, attendance(490))  
    assertEquals(165, attendance(470))  
    assertEquals(0, attendance(1000))  
    assertEquals(0, attendance(580))  
    assertEquals(2, attendance(579))  
    assertEquals(870, attendance(0))  
  }  
  ...  
}
```

# Example: Testing Our Theater Profit Calculator

```
class TheaterProfitTest(name: String) extends TestCase(name) {  
  ...  
  def testRevenue() {  
    assertEquals(0, revenue(0))  
    assertEquals(0, revenue(1000))  
    assertEquals(53550, revenue(510))  
  }  
  ...  
}
```

# Using DrScala

# DrScala

- Available from the course homepage:

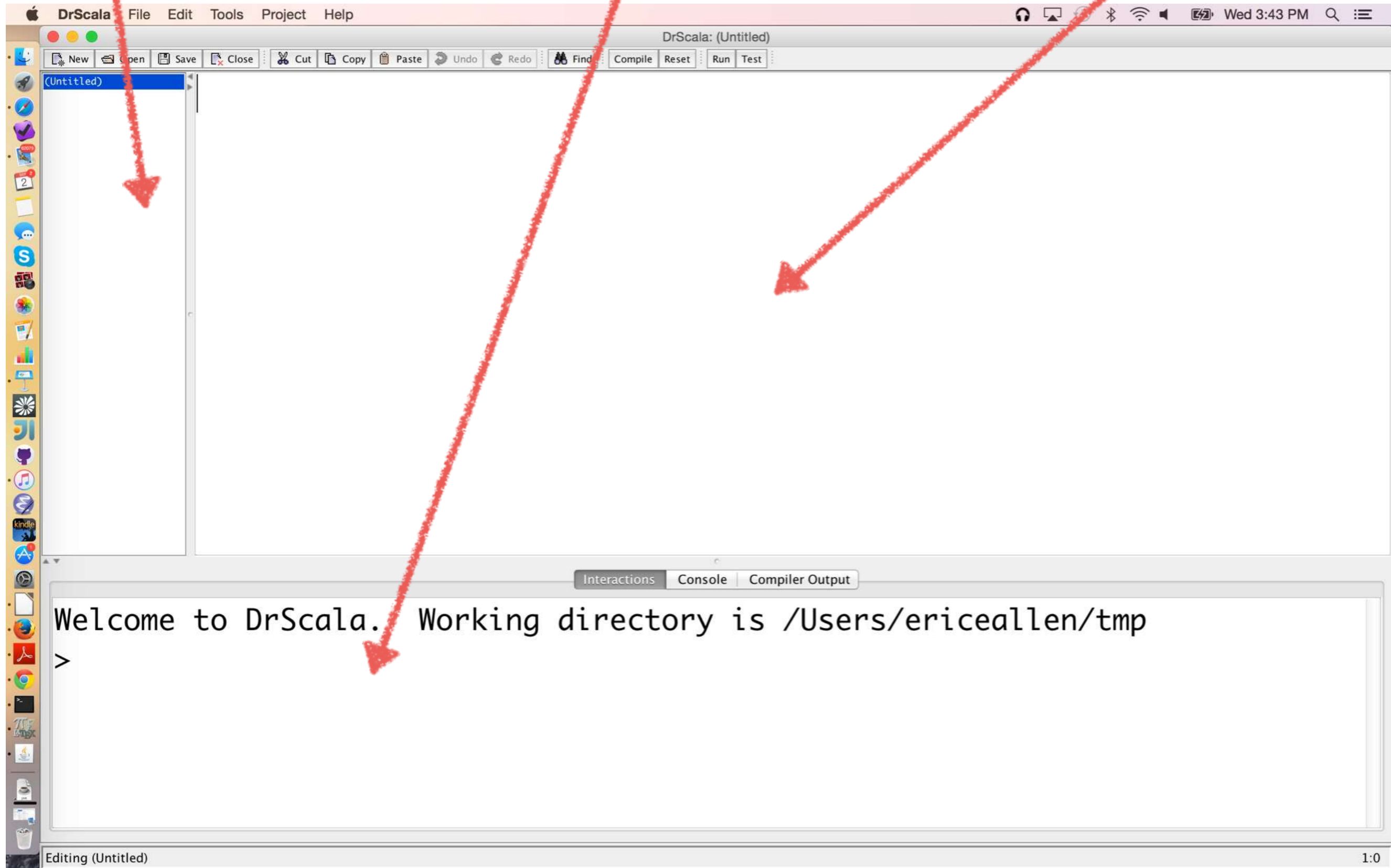
<https://comp311.rice.edu>

- A lightweight development environment well-suited to the exercises we will do in this class

# Open Files

# Interactions Pane

# Definitions Pane



# Define your program in the definitions pane

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The title bar indicates the current file is 'DrScala: /Users/ericeallen/tmp/IncomeTax.scala'. The toolbar contains icons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. The main editor area displays the following Scala code:

```
object IncomeTax {  
  |  
  val cutoff0 = 0  
  val bracket0 = 0  
  
  val bracket1 = 100  
  val cutoff1 = 9075  
  
  val bracket2 = 150  
  val cutoff2 = 36900  
  
  val bracket3 = 250  
  val cutoff3 = 89350  
}
```

A red arrow points from the title 'Define your program in the definitions pane' to the code in the editor. Below the editor, the 'Interactions' pane shows the following text:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

The status bar at the bottom indicates 'Editing /Users/ericeallen/tmp/IncomeTax.scala \*' and '2:3'.



# A prompt to save your program after hitting the Compile button

The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code:

```
object IncomeTax {  
  
  val cutoff0 = 0  
  val bracket0 = 0  
  
  val bracket1 = 100  
  val cutoff1 = 9075  
  
  val bracket2 =  
  val cutoff2 = 3  
  
  val bracket3 =  
  val cutoff3 = 89350
```

A dialog box titled "Must Save All Files to Continue" is overlaid on the code. The dialog contains the following text:

To compile, you must first save ALL modified files.  
Would you like to save and then compile?

Always save before compiling

No Yes

The console window at the bottom of the IDE shows the following output:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

At the bottom of the IDE, the status bar indicates "Editing /Users/ericeallen/tmp/IncomeTax.scala \*".

The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code for an object named `IncomeTax`:

```
object IncomeTax {  
  val cutoff0 = 0  
  val bracket0 = 0  
  
  val bracket1 = 100  
  val cutoff1 = 9075  
  
  val bracket2 = 150  
  val cutoff2 = 36900  
  
  val bracket3 = 250  
  val cutoff3 = 89350  
}
```

Below the code editor, the **Compiler Output** tab is active, showing the message: "Compilation completed. Output directory is: /Users/ericeallen/tmp". To the right of this message, a small panel shows the compiler version as "Scala version 2.12.0-M2" and a checked checkbox for "Highlight source". A red arrow points from the text "Successful compilation reported in the Compiler Output tab" to the Compiler Output tab.

Successful compilation reported  
in the Compiler Output tab

Interactions Console **Compiler Output**

Compilation completed. Output directory is:  
/Users/ericeallen/tmp

Compiler  
Scala version 2.12.0-M2

Highlight source

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

```
IncomeTax.scala
object IncomeTax {

    val cutoff0 = 0
    val bracket0 = 0

    val bracket1 = 100
    val cutoff1 = 9075

    val bracket2 = 150
    val cutoff2 = 36900

    val bracket3 = 250
    val cutoff3 = 89350
}
```

Interactions Console Compiler Output

Editing /Users/ericeallen/tmp/IncomeTax.scala

2:3

Console output from running  
a program printed here



The image shows a screenshot of the DrScala IDE. The main editor window displays the following Scala code for an `IncomeTax` object:

```
object IncomeTax {  
  
  val cutoff0 = 0  
  val bracket0 = 0  
  
  val bracket1 = 100  
  val cutoff1 = 9075  
  
  val bracket2 = 150  
  val cutoff2 = 36900  
  
  val bracket3 = 250  
  val cutoff3 = 89350  
}
```

Below the code editor is an interactive console pane with tabs for "Interactions", "Console", and "Compiler Output". The "Interactions" tab is active, showing the following text:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

A red arrow points from the text "We can interact with the functions in our program directly in the interactions pane" to the console pane.

We can interact with the functions  
in our program directly in the  
interactions pane



The screenshot shows the DrScala IDE interface. The 'File' menu is open, with 'New JUnit Test Case...' highlighted. A red arrow points from this menu item to the code editor. The code editor contains the following Scala code:

```
IncomeTax {  
  toff0 = 0  
  bracket0 = 0  
  bracket1 = 100  
  toff1 = 9075  
  bracket2 = 150  
  toff2 = 36900  
  val bracket3 = 250  
  val cutoff3 = 89350  
}
```

The console at the bottom displays the following text:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>
```

Create a new JUnit TestCase class

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

```
IncomeTax.scala
object IncomeTax {

    val cutoff0 = 0
    val bracket0 = 0

    val bracket1 = 100
    val cutoff1 = 9075

    val bracket2 = 15
    val cutoff2 = 369

    val bracket3 = 25
    val cutoff3 = 89350
}
```

We are prompted for a name. Let's call it IncomeTaxTest

New JUnit Test Case

Please enter a name for the test class:

IncomeTaxTest

Cancel OK

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing /Users/ericeallen/tmp/IncomeTax.scala

2:3

DrScala File Edit Tools Project Help Wed 3:54 PM

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
import junit.framework.TestCase
import junit.framework.Assert._

/**
 * A JUnit test case class.
 * Every method starting with the word "test" will be called when ru
 * the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * A test method.
   * (Replace "X" with a name describing the test. You may write as
```

A new test class is created

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\* 26:0



DrScala File Edit Tools Project Help

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
import junit.framework.TestCase
import junit.framework.Assert._

/**
 * A JUnit test case class.
 * Every method starting with the word "test" will be called when ru
 * the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * A test method.
   * (Replace "X" with a name describing the test. You may write as
```

Note that this is *not* a case class

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\*

26:0



DrScala File Edit Tools Project Help

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test


(Untitled)\*  
IncomeTax.scala

```
import junit.framework.TestCase
import junit.framework.Assert._

/**
 * A JUnit test case class.
 * Every method starting with the word "test" will be called when ru
 * the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * A test method.
   * (Replace "X" with a name describing the test. You may write as
```

Ignore the extends clause for now



Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\* 26:0

DrScala File Edit Tools Project Help

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
import junit.framework.TestCase
import junit.framework.Assert._

/**
 * A JUnit test case class.
 * Every method starting with the word "test" will be called when ru
 * the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * A test method.
   * (Replace "X" with a name describing the test. You may write as
```

**Ignore the import statements for now**

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\* 26:0



DrScala File Edit Tools Project Help Wed 3:54 PM

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
* many "testSomething" methods in this class as you wish, and eac
* one will be called when running JUnit over this class.)
*/
def testX() {
}

/** Sample test method which tests no program code. */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
}
```

The provided tests don't do very much

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\* 26:0

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The toolbar contains icons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. The main editor window displays Scala code for a class named 'IncomeTax.scala'. The code includes a comment about 'testSomething' methods, a 'testX()' method definition, and a 'testNothing()' method definition. A red arrow points from the text 'All functions with names starting with "test" are treated as tests' to the 'testX()' method definition. The bottom panel shows the 'Test Output' tab with the message 'Welcome to DrScala. Working directory is /Users/ericeallen/tmp' and the output 'TESTING Nothing'.

```
DrScala: (Untitled) *
File Edit Tools Project Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
IncomeTax.scala
*/
def testX()
}

/** Sample test method which tests no program code. */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
}

All functions with names starting
with "test" are treated as tests

Welcome to DrScala. Working directory is /Users/ericeallen/tmp
TESTING Nothing
>
```

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The toolbar contains icons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. The main editor window displays the following Scala code:

```
* many "testSomething" methods in this class as you wish, and eac
* one will be called when running JUnit over this class.)
*/
def testX() {
}

/** Sample test method which tests no program code. */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
}
```

A red arrow points from the text 'The assertTrue function is available to us in our tests.' to the `assertTrue` call in the code. The bottom panel shows the 'Test Output' tab with the following text:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp
TESTING Nothing
>
```

The status bar at the bottom left indicates 'Editing (Untitled) \*' and the bottom right shows '26:0'.

The assertTrue function is available to us in our tests.



DrScala File Edit Tools Project Help

DrScala: (Untitled)\*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
* many "testSomething" methods in this class as you wish, and eac
* one will be called when running JUnit over this class.)
*/
def testX() {
}

/** Sample test method which tests no program code. */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
}
```

The optional String is printed if the test fails.

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled)\*

Wed 3:54 PM

26:0

DrScala File Edit Tools Project Help Wed 3:54 PM

DrScala: (Untitled) \*

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala

```
* many "testSomething" methods in this class as you wish, and eac
* one will be called when running JUnit over this class.)
*/
def testX() {
}

/** Sample test method which tests no program code. */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
}
```

The test fails if this argument does not reduce to true.

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
>

Editing (Untitled) \* 26:0



# assertEquals fails if its two arguments are not equal

The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code:

```
/**  
 * Testing simple income tax computations.  
 */  
def testIncomeTax() {  
  assertEquals(100, IncomeTax.incomeTax(1000))  
  assertEquals(907, IncomeTax.incomeTax(9075))  
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))  
}  
  
/** Sample test method which tests no program code. */  
def testNothing() {  
  assertTrue("Dummy Test", true)  
  println("TESTING Nothing")  
}
```

Red arrows point from the title to the `assertEquals` calls in the `testIncomeTax` function. A large red text overlay says "Add many more test functions" with an arrow pointing to the `testIncomeTax` function definition.

The bottom panel shows the Test Output tab with the following text:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
TESTING Nothing  
> |
```

At the bottom left, the status bar shows "Bracket matches: def testIncomeTax() {" and the time "18:3".



# Hitting the Test button prompts us to compile

The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code for `IncomeTaxTest.scala`:

```
class IncomeTaxTest(name: String) extends TestCase(name) {  
  
  /**  
   * Testing simple income tax computations.  
   */  
  def testInc  
    assertEquals  
    assertEquals  
    assertEquals  
  
}
```

A dialog box titled "Must Compile All Source Files to Run Unit Tests" is overlaid on the code. It contains the following text:

Before you can run unit tests, you must first compile all out of sync source files. The files below are out of sync. Would you like to compile all files and run the specified test(s)?

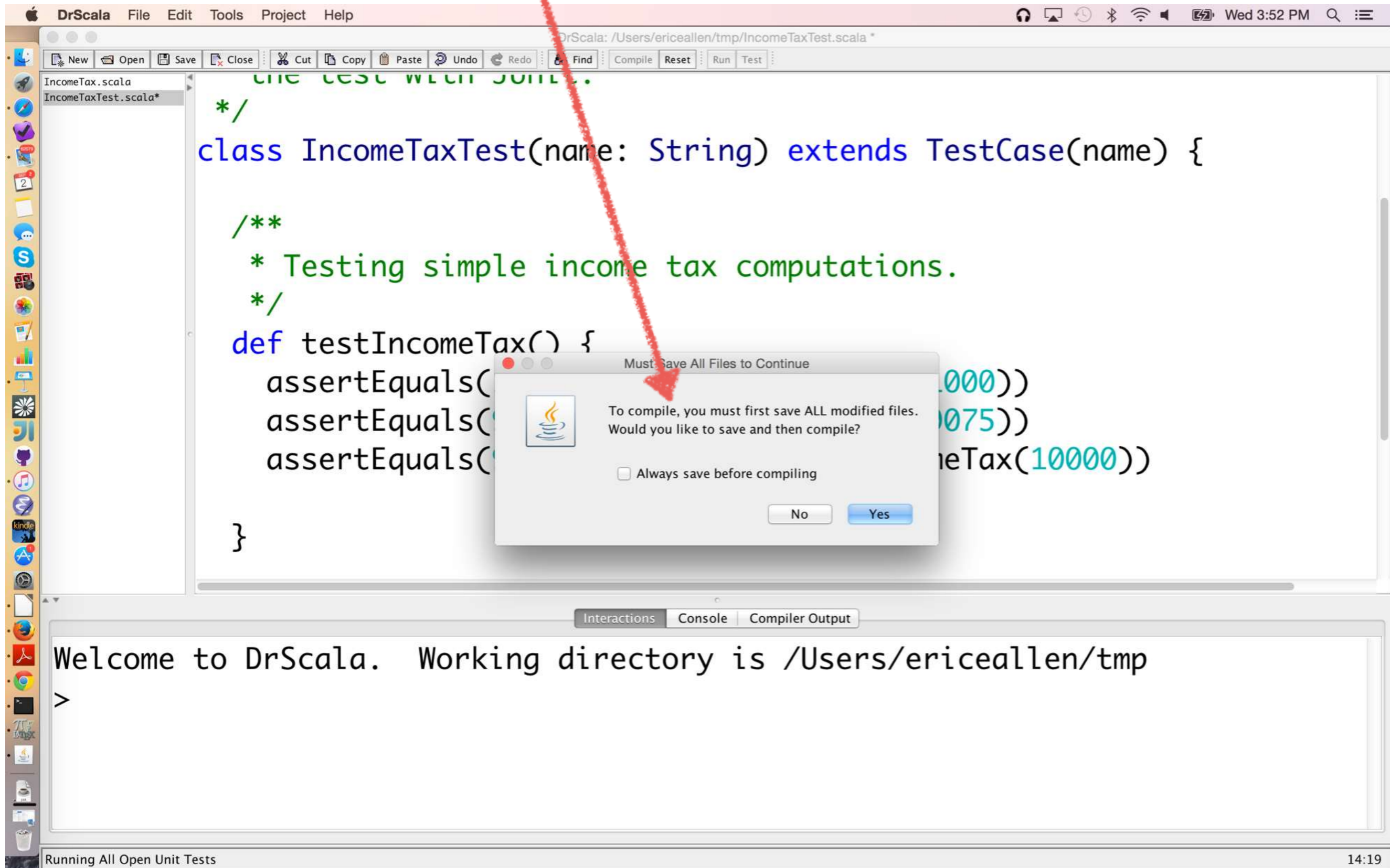
The dialog lists `IncomeTaxTest.scala` as the file to be compiled. At the bottom of the dialog are "Yes" and "No" buttons. A red arrow points from the "Test" button in the IDE's toolbar to the "Yes" button in the dialog.

Below the editor, the console shows the following output:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

The status bar at the bottom of the IDE displays "Running All Open Unit Tests" and the time "14:19".

# Agreeing to compile prompts us to save



The screenshot shows the DrScala IDE interface. The main editor window displays Scala code for a test class:

```
class IncomeTaxTest(name: String) extends TestCase(name) {  
  
  /**  
   * Testing simple income tax computations.  
   */  
  def testIncomeTax() {  
    assertEquals(  
    assertEquals(  
    assertEquals(  
  
  }  
}
```

A dialog box titled "Must Save All Files to Continue" is overlaid on the code. It contains the following text:

To compile, you must first save ALL modified files.  
Would you like to save and then compile?

Always save before compiling

Buttons: No, Yes

The console at the bottom of the IDE shows the following output:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

The status bar at the bottom left indicates "Running All Open Unit Tests" and the bottom right shows the time "14:19".

# A green bar indicates that all tests passed

The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code:

```
class IncomeTaxTest(name: String) extends TestCase(name) {  
  
  /**  
   * Testing simple income tax computations.  
   */  
  def testIncomeTax() {  
    assertEquals(100, IncomeTax.incomeTax(1000))  
    assertEquals(907, IncomeTax.incomeTax(9075))  
    assertEquals(907 + 138, IncomeTax.incomeTax(10000))  
  }  
}
```

Below the editor, the Test Output panel is active, showing the following text:

```
All tests completed successfully.  
IncomeTaxTest  
  testNothing  
  testIncomeTax
```

On the right side of the Test Output panel, there is a 'Test Progress' section with a green progress bar that is completely filled, indicating that all tests passed. A red arrow points from the title text at the top of the slide to this green bar.

At the bottom of the IDE window, the status bar shows 'Editing /Users/ericallen/tmp/IncomeTaxTest.scala' and the time '14:19'.



DrScala File Edit Tools Project Help Wed 8:15 PM

DrScala: /Users/ericeallen/tmp/IncomeTaxTest.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

IncomeTax.scala  
IncomeTaxTest.scala

```
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

def testThatFails() {
  assertTrue(false)
}

/** Sample test method which tests no program code.
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
```

A red bar indicates a test failure

Interactions Console Compiler Output Test Output

IncomeTaxTest  
testIncomeTax  
testThatFails  
testNothing

Test Progress  
Show Stack Trace  
 Highlight source

Editing /Users/ericeallen/tmp/IncomeTaxTest.scala 21:0

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The title bar indicates the file path: 'DrScala: /Users/ericeallen/tmp/IncomeTaxTest.scala'. The toolbar contains buttons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. The main editor area displays the following Scala code:

```
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

def testThatFails() {
  assertTrue(false)
}

/** Sample test method which tests no program code.
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
```

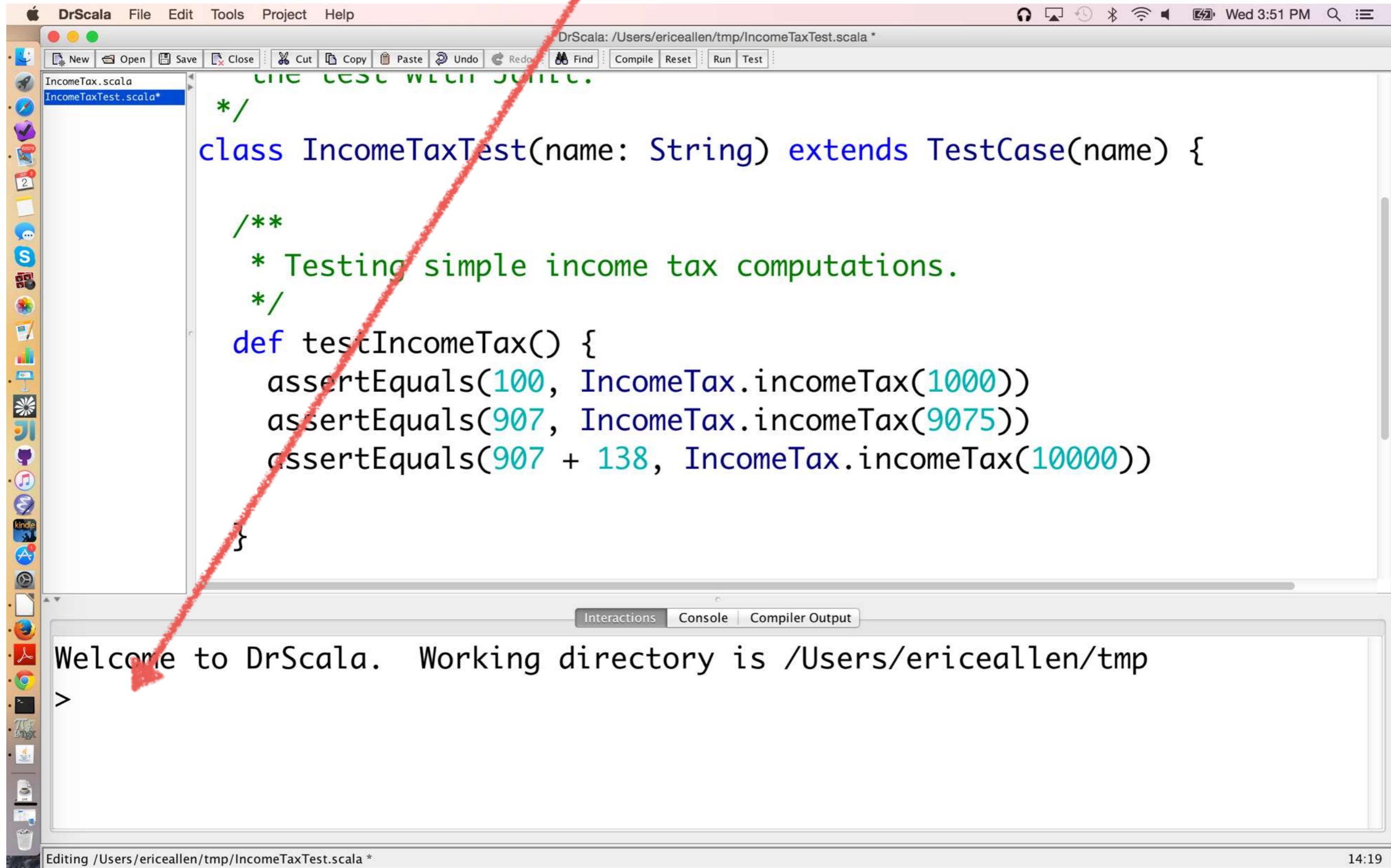
The line `assertTrue(false)` in the `testThatFails` method is highlighted in yellow. A red arrow points from the text 'The failing test is highlighted in yellow' to this line. The bottom panel shows the 'Test Output' tab with the following results:

```
IncomeTaxTest
testIncomeTax
testThatFails
testNothing
```

On the right side of the bottom panel, there is a 'Test Progress' indicator with a red bar, a 'Show Stack Trace' button, and a checked 'Highlight source' checkbox. The status bar at the bottom left shows 'Editing /Users/ericeallen/tmp/IncomeTaxTest.scala' and the bottom right shows '21:0'.



# To interact with our program, we use the Interactions Pane



The screenshot shows the DrScala IDE interface. The main editor window displays the following Scala code:

```
class IncomeTaxTest(name: String) extends TestCase(name) {  
  
  /**  
   * Testing simple income tax computations.  
   */  
  def testIncomeTax() {  
    assertEquals(100, IncomeTax.incomeTax(1000))  
    assertEquals(907, IncomeTax.incomeTax(9075))  
    assertEquals(907 + 138, IncomeTax.incomeTax(10000))  
  }  
}
```

The Interactions pane at the bottom shows the following output:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
>
```

A red arrow points from the top of the slide to the Interactions pane.

DrScala File Edit Tools Project Help Wed 5:02 PM

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
> 2 + 2  
res0: Int = 4  
> |
```

We can enter arbitrary Scala expressions

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
> 2 + 2  
res0: Int = 4  
> |
```

The value our expression reduces to is displayed

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3



DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
> 2 + 2  
res0: Int = 4  
> |
```

As is its type

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
> 2 + 2  
res0: Int = 4  
> |
```

And the value is bound to a fresh identifier

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp  
> 2 + 2  
res0: Int = 4  
> IncomeTax.incomeTax(100000)  
res1: Int = 21174  
>
```

The classes we have compiled in  
Definitions are in scope in Interactions

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3



The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The title bar indicates the current file is 'IncomeTax.scala' located at '/Users/ericeallen/tmp/IncomeTax.scala'. The main editor window contains the following Scala code:

```
/* Given an income in U.S. Dollars,
 * returns the dollar value of tax
 * owed for a single tax payer, using
 * 2014-2015 IRS tax brackets.
 */
def incomeTax(income: Int): Int = {
  require(income >= 0)

  if (income <= cutoff0) {
    bracket0
  }
}
```

Below the editor is a REPL window with tabs for 'Interactions', 'Console', 'Compiler Output', and 'Test Output'. The REPL shows the following interactions:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp
> 2 + 2
res0: Int = 4
> IncomeTax.incomeTax(100000)
res1: Int = 21174
> res0 * res1
res2: Int = 84696
>
```

A red arrow points from the text 'We can refer to previously bound identifiers in subsequent expressions' to the 'res1' value in the REPL output.

We can refer to previously bound identifiers in subsequent expressions

DrScala File Edit Tools Project Help Wed 5:04 PM

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

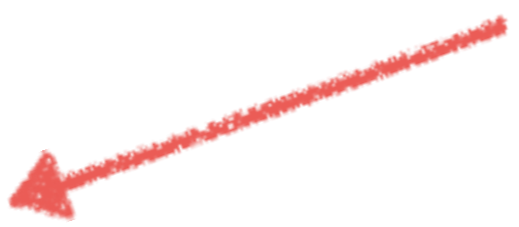
(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
res0: Int = 4  
> IncomeTax.incomeTax(100000)  
res1: Int = 21174  
> res0 * res1  
res2: Int = 84696  
> val pi = 3.14  
pi: Double = 3.14  
>
```

We can also bind new identifiers directly



DrScala File Edit Tools Project Help Wed 5:04 PM

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
res1: Int = 21174  
> res0 * res1  
res2: Int = 84696  
> val pi = 3.14  
pi: Double = 3.14  
> res1 * pi  
res3: Double = 66486.36  
> |
```

And compute with them

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3



DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
res2: Int = 84696  
> val pi = 3.14  
pi: Double = 3.14  
> res1 * pi  
res3: Double = 66486.36  
> def square(x: Double) = x * x  
square: (x: Double)Double  
>
```

We can also define new functions

Editing /Users/ericeallen/tmp/IncomeTax.scala

2:3

DrScala File Edit Tools Project Help Wed 5:05 PM

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/* Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

```
> val pi = 3.14  
pi: Double = 3.14  
> res1 * pi  
res3: Double = 66486.36  
> def square(x: Double) = x * x  
square: (x: Double)Double  
> def abs(x: Double) =  
  |
```

For definitions that are not syntactically complete, we are given a new line, indicated by a vertical bar

Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3



DrScala File Edit Tools Project Help Wed 5:07 PM

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/**  
 * Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
  
  if (income <= cutoff0) {  
    bracket0
```

Interactions Console Compiler Output Test Output

Welcome to DrScala. Working directory is /Users/ericeallen/tmp

```
> def abs(x: Double) =  
  | if (x < 0) -x else  
  | x  
abs: (x: Double)Double
```

>

Resetting Interactions

2:3

The function is bound and an arrow type is displayed

DrScala File Edit Tools Project Help

DrScala: /Users/ericeallen/tmp/IncomeTax.scala

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test

(Untitled)\*  
IncomeTax.scala  
IncomeTaxTest.scala\*

```
/**
 * Given an income in U.S. Dollars,
 * returns the dollar value of tax
 * owed for a single tax payer, using
 * 2014-2015 IRS tax brackets.
 */
def incomeTax(income: Int): Int = {
  require(income >= 0)

  if (income <= cutoff0) {
    bracket0
```

Interactions Console Compiler Output Test Output

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp
> def abs(x: Double) =
  | if (x < 0) -x else
  | x
abs: (x: Double)Double
> abs(-5.0)
res0: Double = 5.0
>
```

Resetting Interactions

2:3

And we can refer to this function in subsequent expressions



# We can click on the file to appear in Definitions

The screenshot shows the DrScala IDE interface. The title bar reads "DrScala: /Users/ericeallen/tmp/IncomeTax.scala". The menu bar includes "DrScala", "File", "Edit", "Tools", "Project", and "Help". The toolbar contains buttons for "New", "Open", "Save", "Close", "Cut", "Copy", "Paste", "Undo", "Redo", "Find", "Compile", "Reset", "Run", and "Test".

The file explorer on the left shows a list of files: "(Untitled)\*", "IncomeTax.scala", and "IncomeTaxTest.scala\*". A red arrow points from the title "We can click on the file to appear in Definitions" to the "IncomeTax.scala" file name.

The main editor displays the following Scala code:

```
// Brackets are in tenths of percentage points,  
// thus percentage divisor must be 1000  
val divisor = 1000  
  
/**  
 * Given an income in U.S. Dollars,  
 * returns the dollar value of tax  
 * owed for a single tax payer, using  
 * 2014-2015 IRS tax brackets.  
 */  
def incomeTax(income: Int): Int = {  
  require(income >= 0)  
}
```

At the bottom, the console shows the message: "Welcome to DrScala. Working directory is /Users/ericeallen/tmp" followed by a prompt ">".

The status bar at the bottom left indicates "Editing /Users/ericeallen/tmp/IncomeTax.scala" and the bottom right shows "2:3".

# Files that have not been saved include an asterisk

The screenshot shows the DrScala IDE interface. The title bar reads "DrScala: /Users/ericeallen/tmp/IncomeTaxTest.scala \*". The menu bar includes "DrScala", "File", "Edit", "Tools", "Project", and "Help". The toolbar contains "New", "Open", "Save", "Close", "Cut", "Copy", "Paste", "Undo", "Redo", "Find", "Compile", "Reset", "Run", and "Test".

The file explorer on the left shows a list of files: "(Untitled)\*", "IncomeTax.scala", and "IncomeTaxTest.scala\*", with the latter selected. A red arrow points from the asterisk in the title bar to the asterisk in the filename.

```
* Testing simple income tax computations.
*/
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

/** Sample test method which tests no program code.
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
```

The console at the bottom shows the message: "Welcome to DrScala. Working directory is /Users/ericeallen/tmp" followed by a prompt ">".

At the bottom left, it says "Editing /Users/ericeallen/tmp/IncomeTaxTest.scala \*". At the bottom right, the time is "18:9".



# Reset resets the Interactions session

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The toolbar contains buttons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. A red arrow points to the 'Reset' button. The main editor window displays the following Scala code:

```
(Untitled)*
IncomeTax.scala
IncomeTaxTest.scala*

/* Testing simple income tax computations.
 */
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

/** Sample test method which tests no program code.
 */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
```

Below the editor, the 'Interactions' tab is active, showing the following output:

```
Welcome to DrScala. Working directory is /Users/ericeallen/tmp
>
```

The status bar at the bottom indicates 'Editing /Users/ericeallen/tmp/IncomeTaxTest.scala \*' and the time '18:9'.

# Run executes Definitions

The screenshot shows the DrScala IDE interface. The top menu bar includes 'DrScala', 'File', 'Edit', 'Tools', 'Project', and 'Help'. The title bar indicates the current file is 'DrScala: /Users/ericeallen/tmp/IncomeTaxTest.scala \*'. The toolbar contains buttons for 'New', 'Open', 'Save', 'Close', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', 'Find', 'Compile', 'Reset', 'Run', and 'Test'. A red arrow points to the 'Run' button.

The main editor area displays the following Scala code:

```
(Untitled)*
IncomeTax.scala
IncomeTaxTest.scala*

/* Testing simple income tax computations.
 */
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

/** Sample test method which tests no program code.
 */
def testNothing() {
  assertTrue("Dummy Test", true)
  println("TESTING Nothing")
}
```

At the bottom, the console shows the output of the 'Run' command:

```
Interactions Console Compiler Output Test Output
Welcome to DrScala. Working directory is /Users/ericeallen/tmp
>
```

The status bar at the bottom indicates 'Editing /Users/ericeallen/tmp/IncomeTaxTest.scala \*' and the time '18:9'.