Comp 311
Functional Programming

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

September 12, 2017

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(field_1: Type_1, ..., field_N: Type_N) {
  def m_1(x_{11}: TypeP_{11}, ..., x_{K1}: TypeP_{k1}): TypeR_{11} =
      expr
  ...
  def m_J(x_{1J}: TypeP_{1J}, ..., x_{KJ}: TypeP_{kJ}): TypeR_{1J} =
      expr
  }
```

Method Definitions

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

Applying a Class Method

- Given a class definition:

```
class C(p_1: T_1, ..., p_k: T_k) { ...

    def m(param_1: T_11, param_N: T_1N): T = e

    ...

}
```

- To reduce the application of a method:

```
C(v_1, ..., v_k).m(arg_1, ..., 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() ↦

5*5 + 3*3 ↦

25 + 9 ↦

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
}
```

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)
↦
21174
```

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:

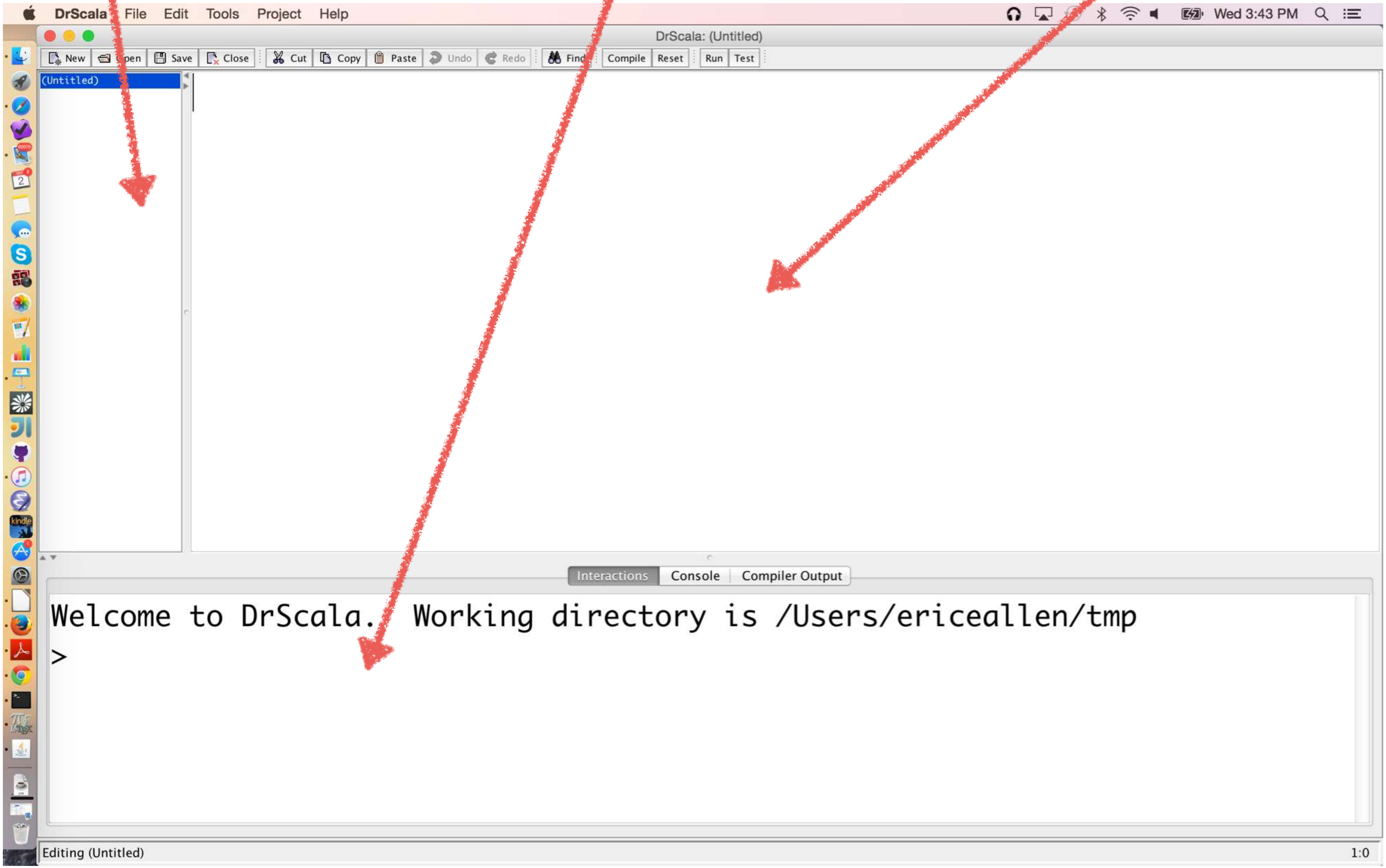
  https://comp311.rice.edu

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

Open Files

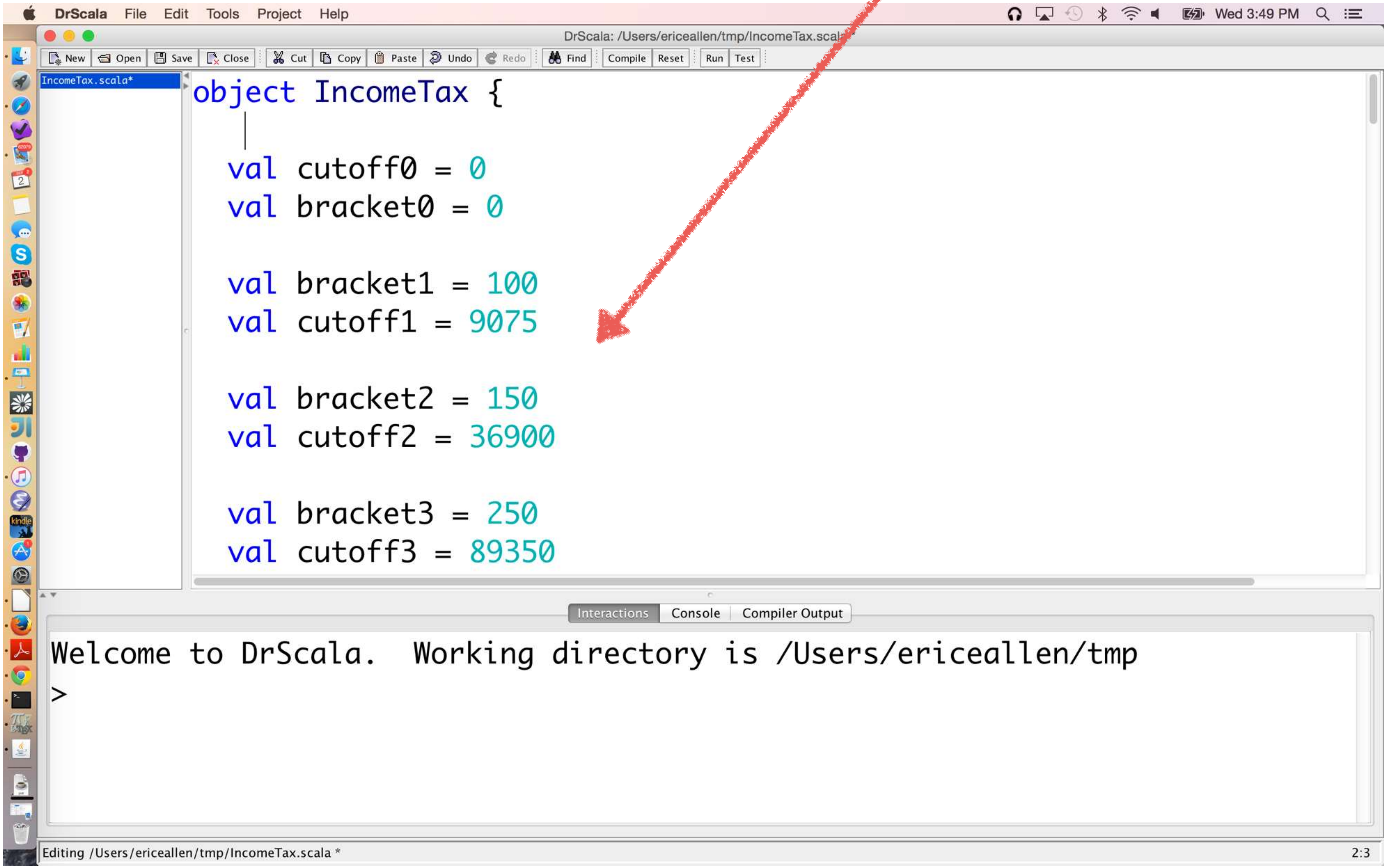
Interactions Pane

Definitions Pane



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

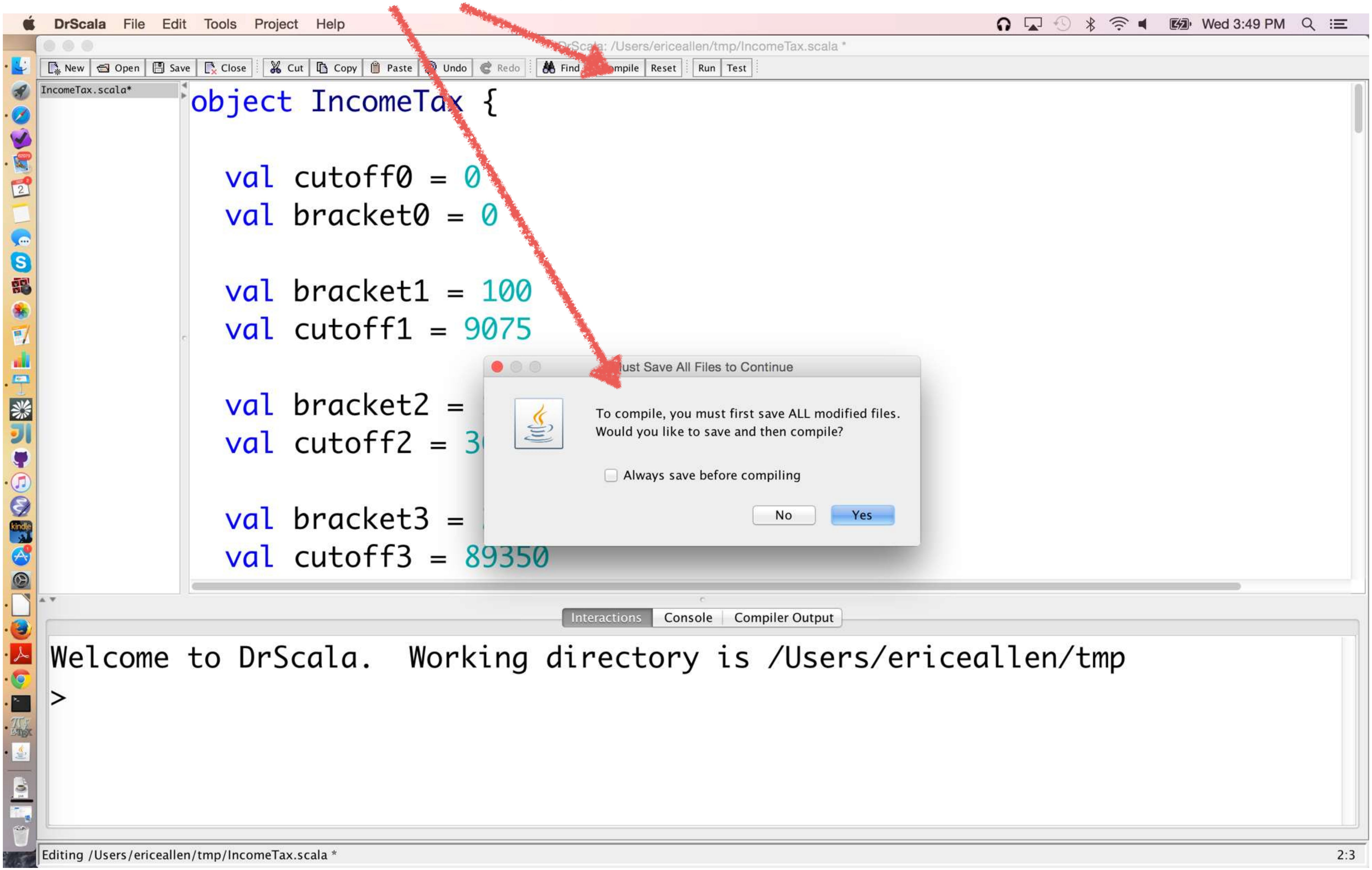
Define your program in the definitions pane



```
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
```

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

A prompt to save your program after hitting the Compile button


DrScala File Edit Tools Project Help
Wed 3:49 PM
DrScala: /Users/ericeallen/tmp/IncomeTax.scala *
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
IncomeTax.scala*
Interactions Console Compiler Output

```
object IncomeTax {

  val cutoff0 = 0
  val bracket0 = 0

  val bracket1 = 100
  val cutoff1 = 9075

  val bracket2 =
  val cutoff2 = 3

  val bracket3 =
  val cutoff3 = 89350
```

Must Save All Files to Continue

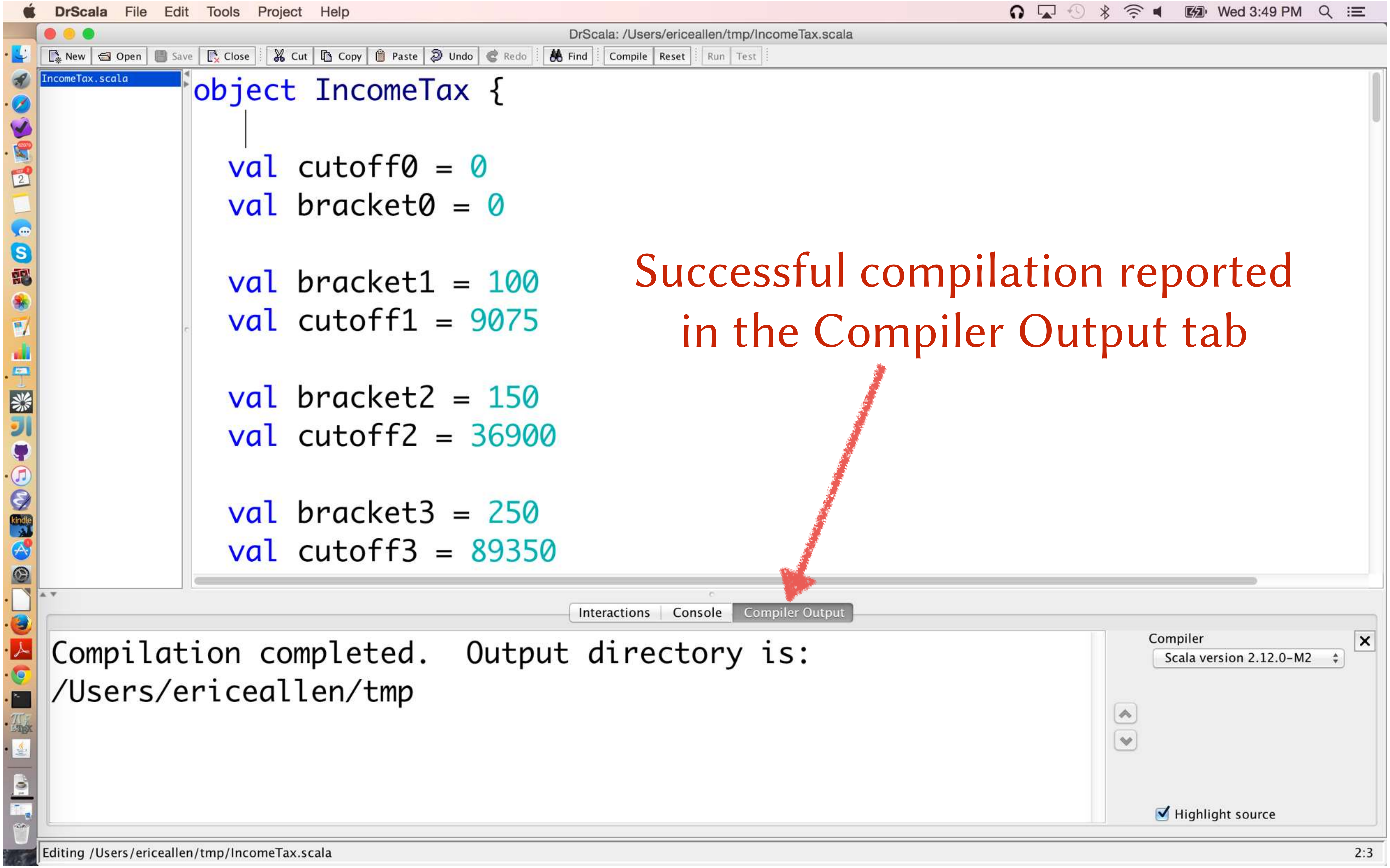
To compile, you must first save ALL modified files.
Would you like to save and then compile?

□ Always save before compiling

No Yes

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

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



```
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
```

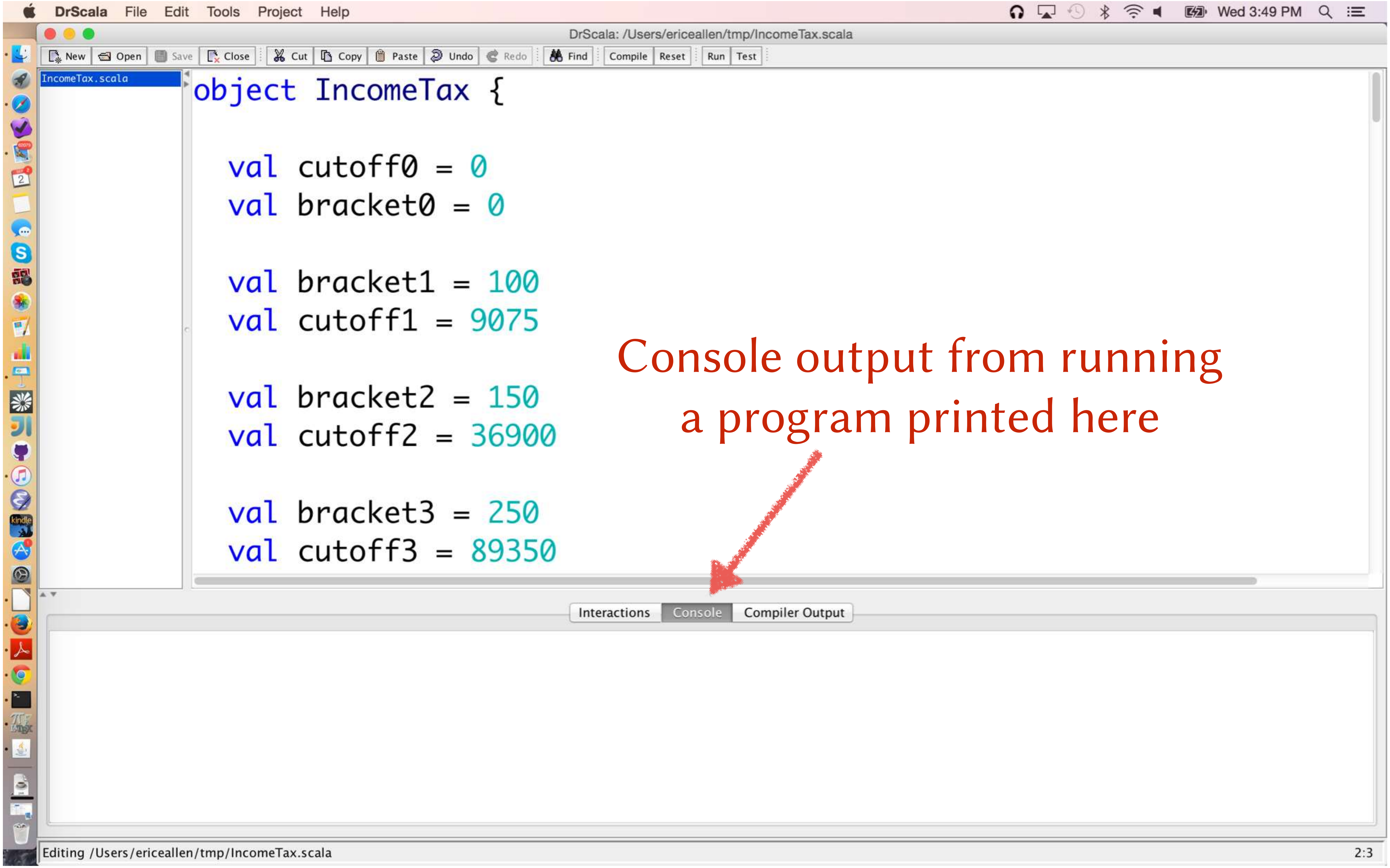
Successful compilation reported in the Compiler Output tab

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

Compiler: Scala version 2.12.0-M2

☑ Highlight source

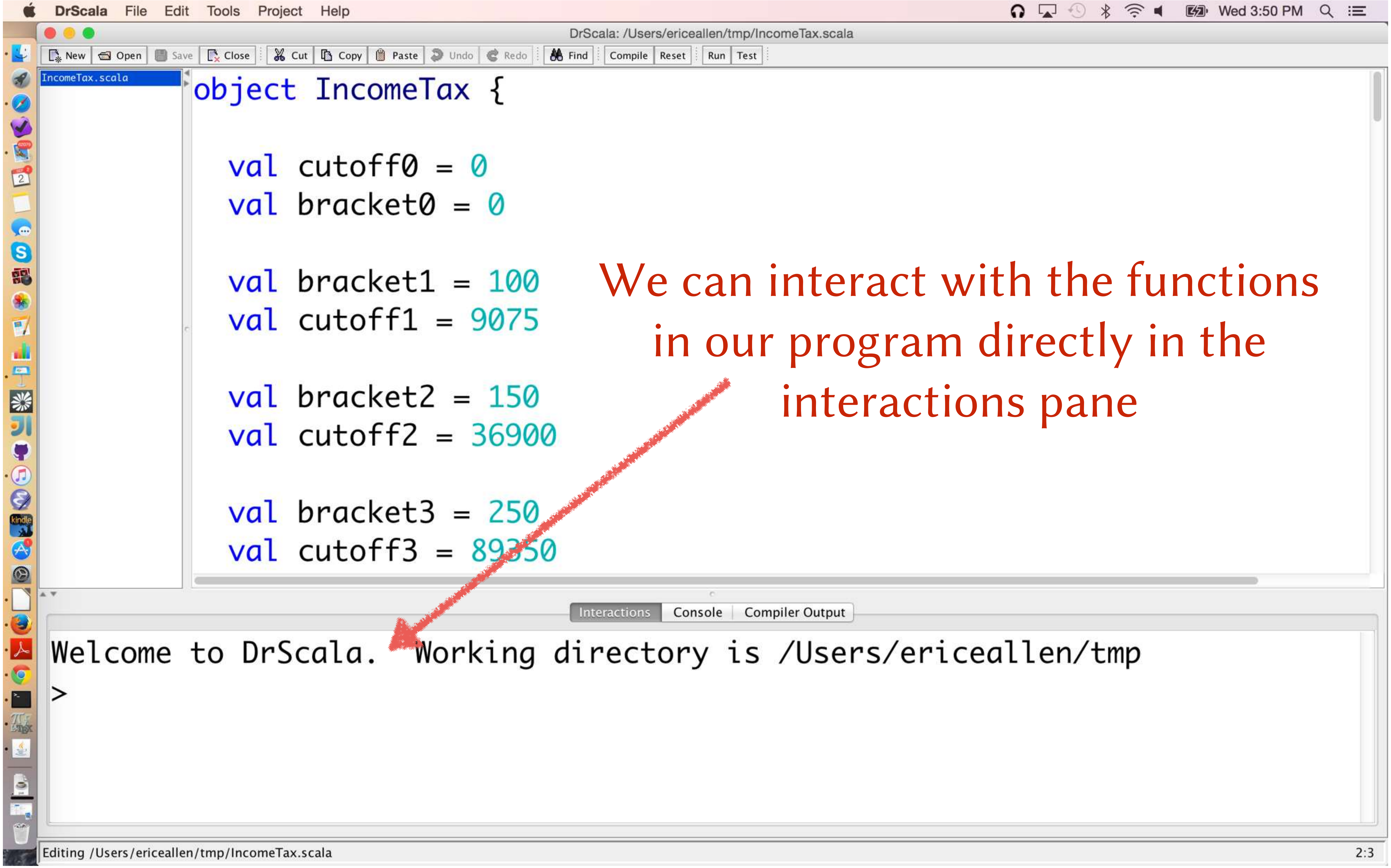
Editing /Users/ericeallen/tmp/IncomeTax.scala 2:3



```
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
```

Editing /Users/ericeallen/tmp/IncomeTax.scala

Console output from running a program printed here



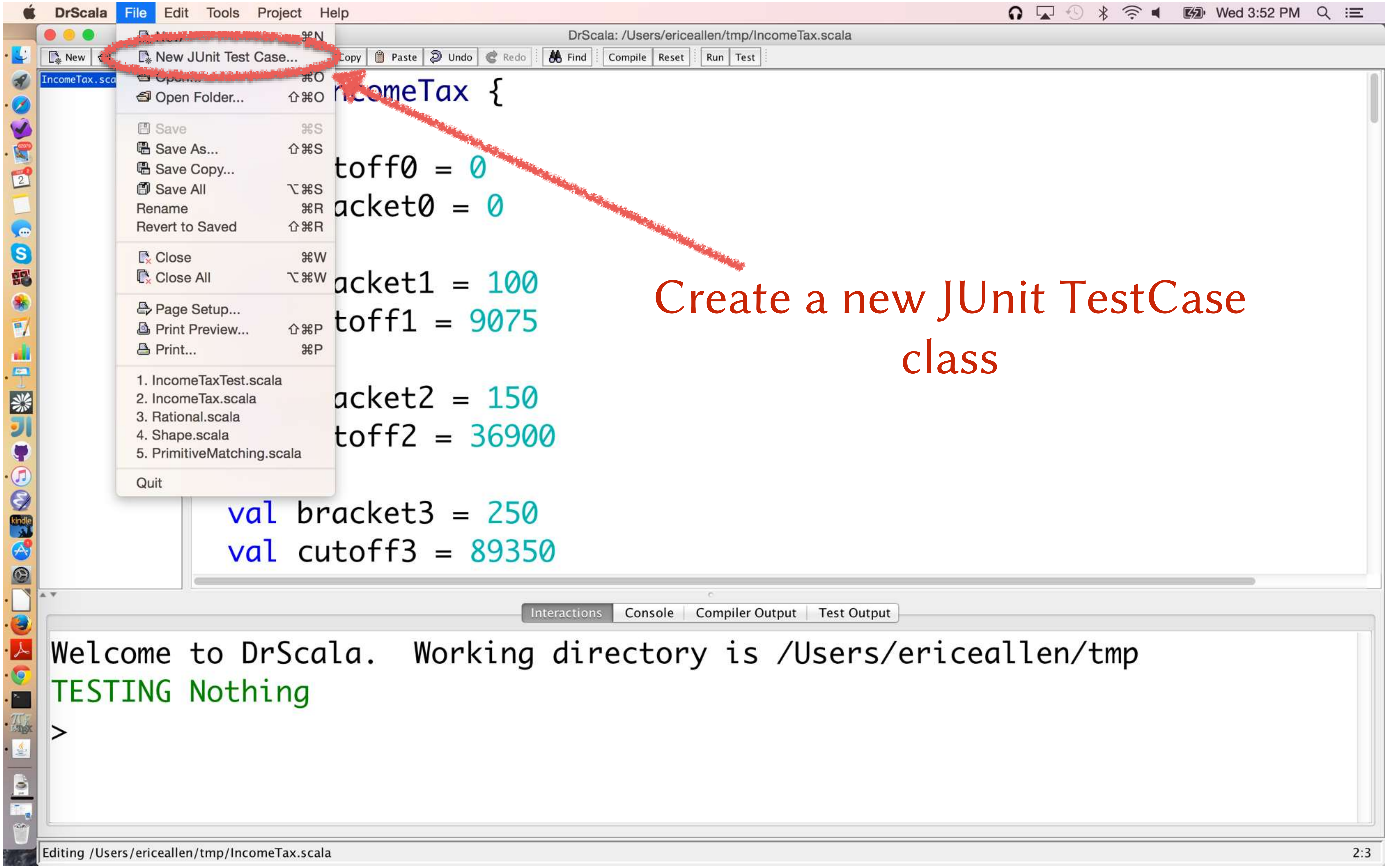
DrScala File Edit Tools Project Help
Wed 3:50 PM
DrScala: /Users/ericeallen/tmp/IncomeTax.scala
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
IncomeTax.scala
Interactions Console Compiler Output
2:3

```
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
```

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

Editing /Users/ericeallen/tmp/IncomeTax.scala

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

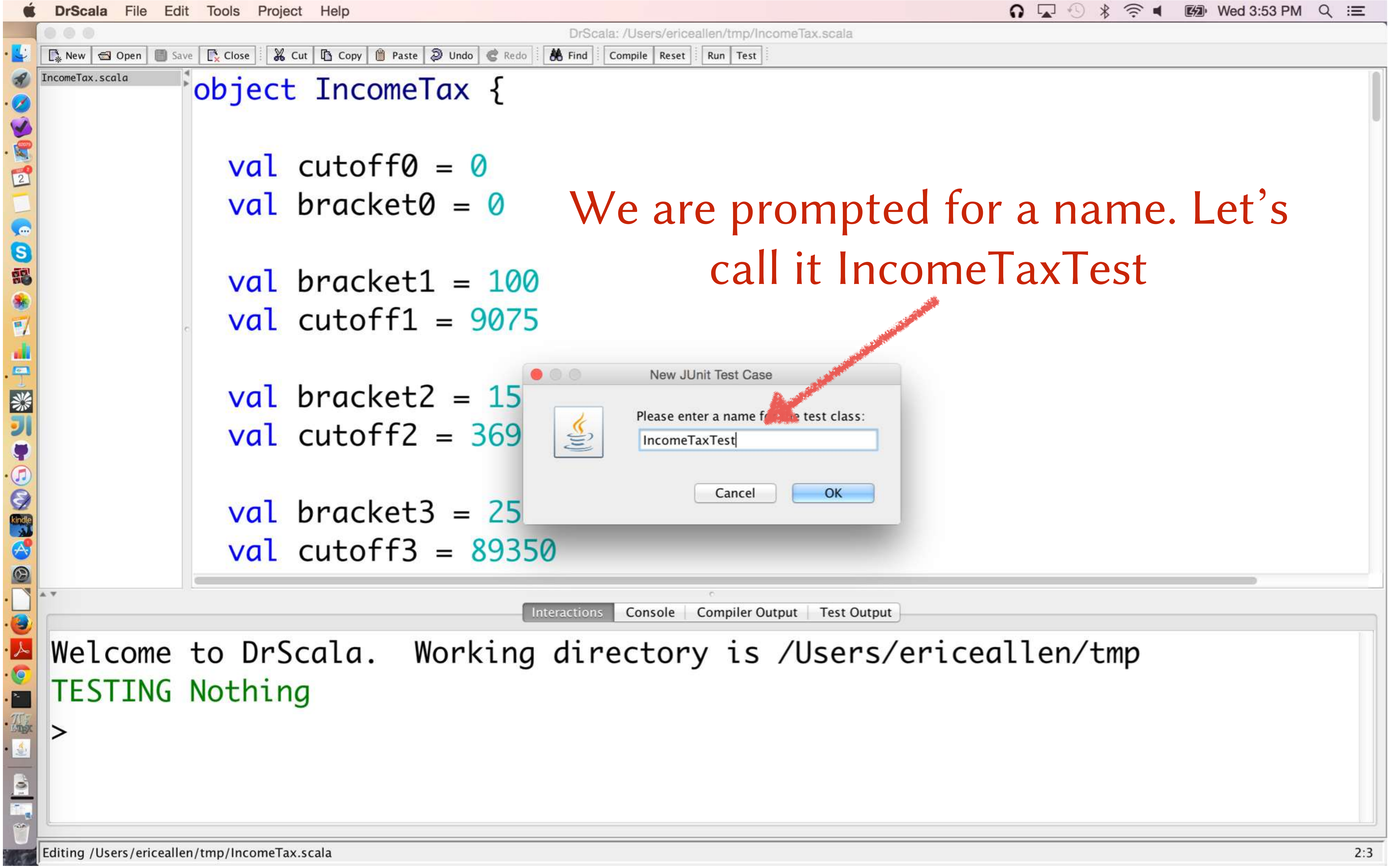


```
ncomeTax {

toff0 = 0
acket0 = 0

acket1 = 100
toff1 = 9075

acket2 = 150
toff2 = 36900

  val bracket3 = 250
  val cutoff3 = 89350
```

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

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

Create a new JUnit TestCase class



```
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
```

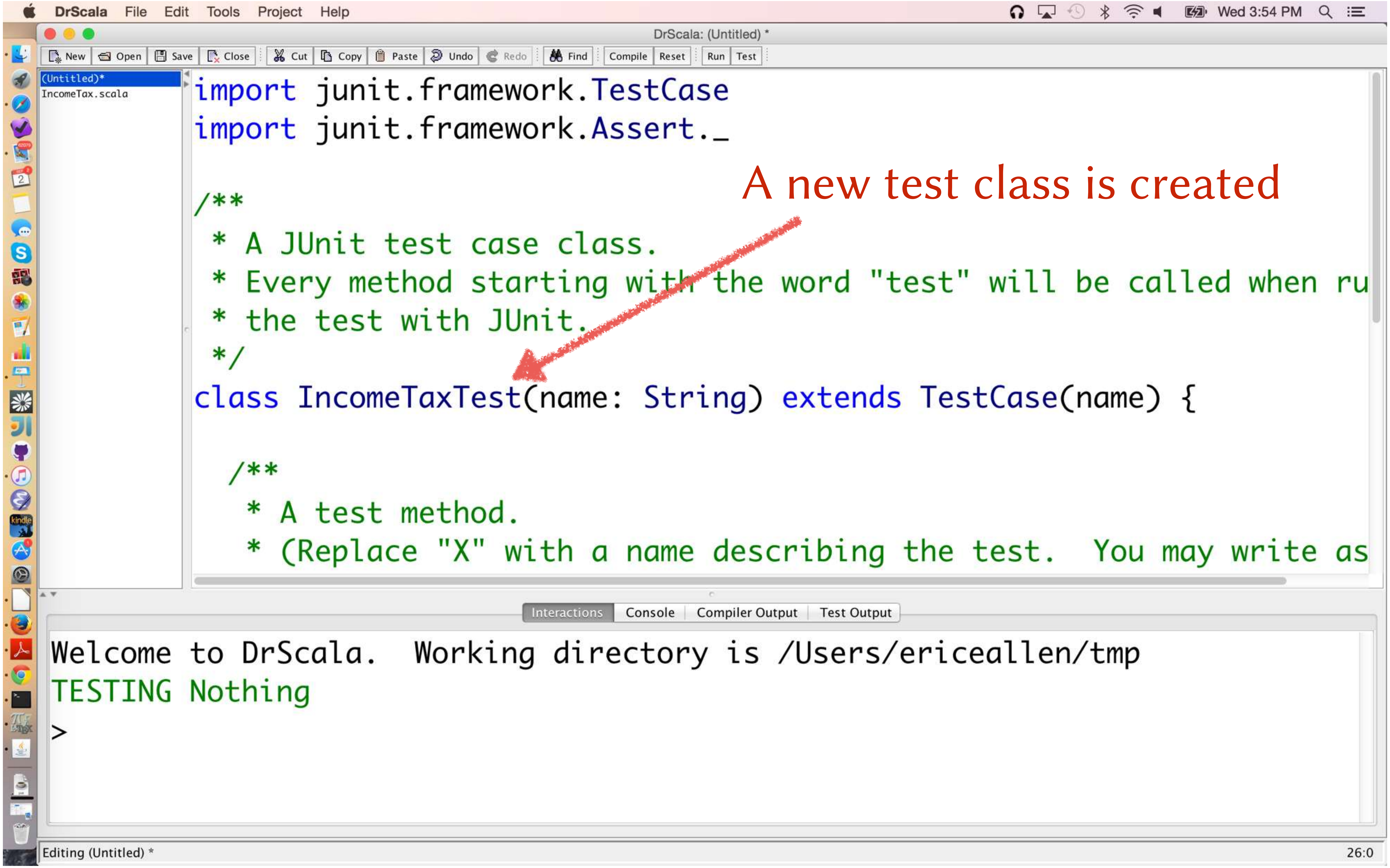
Please enter a name f[illegible]e test class:

IncomeTaxTest

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

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

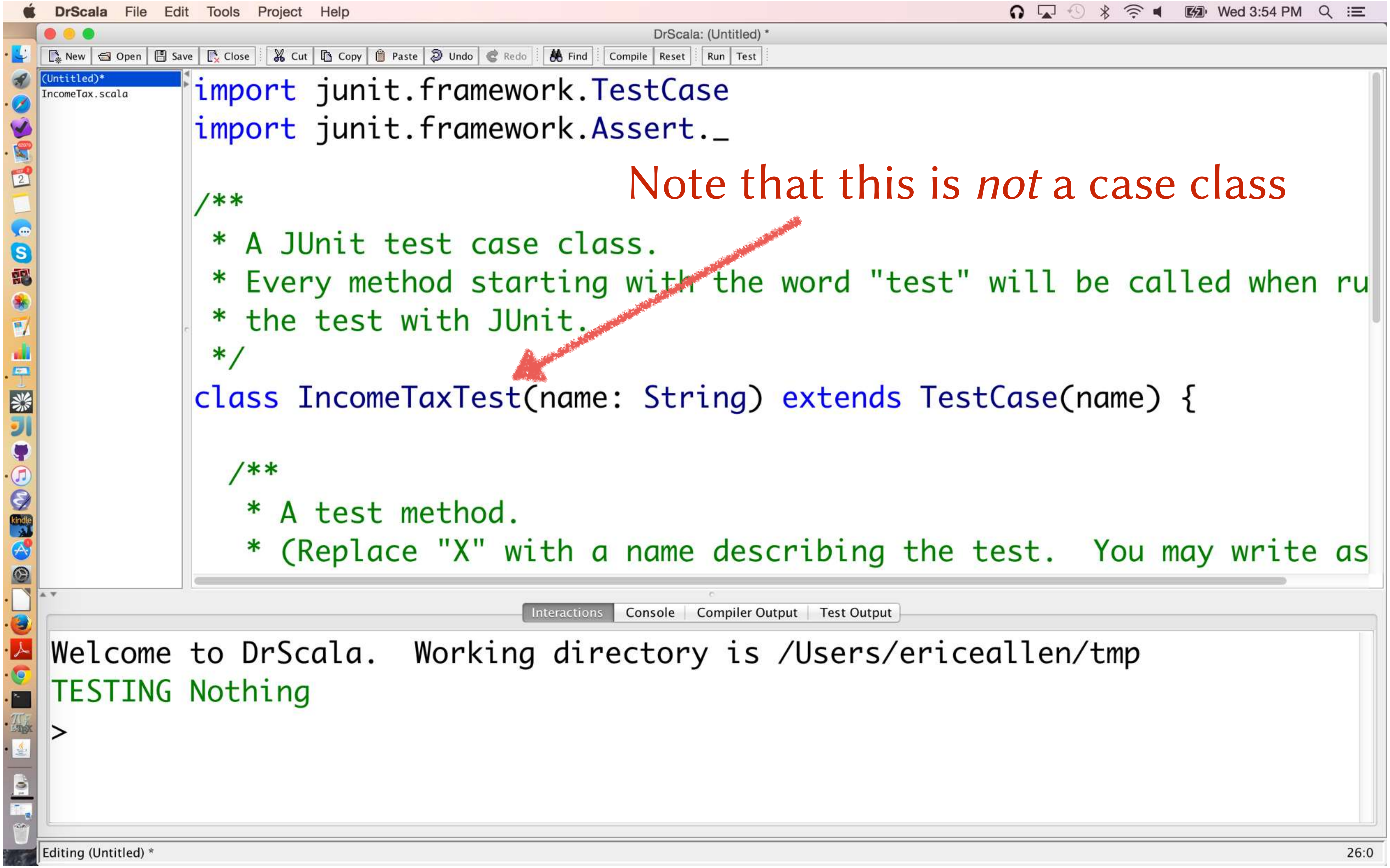
We are prompted for a name. Let's call it IncomeTaxTest



A new test class is created

```
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
```

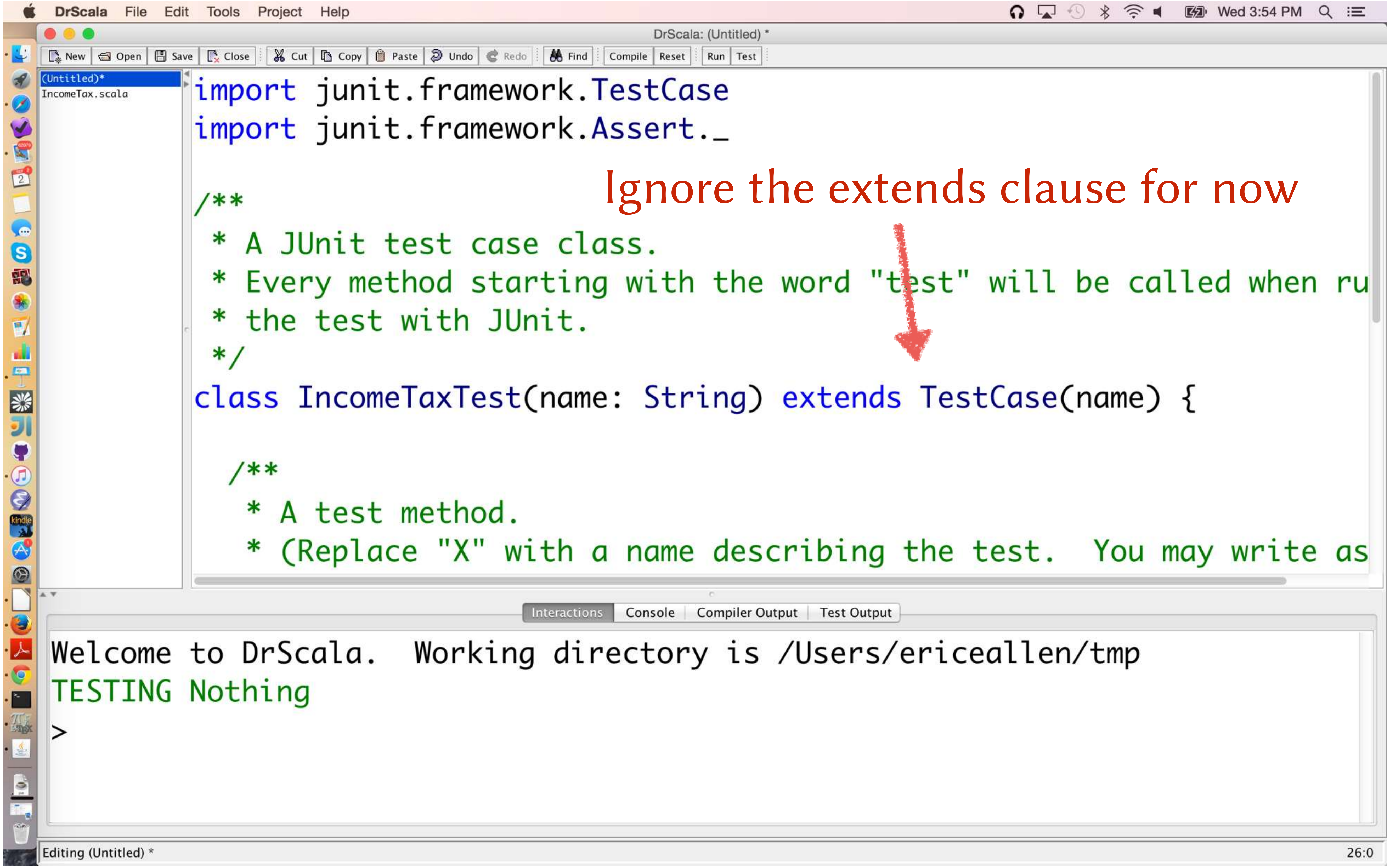
```
Welcome to DrScala.  Working directory is /Users/ericeallen/tmp
TESTING Nothing
>
```



```
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
```

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

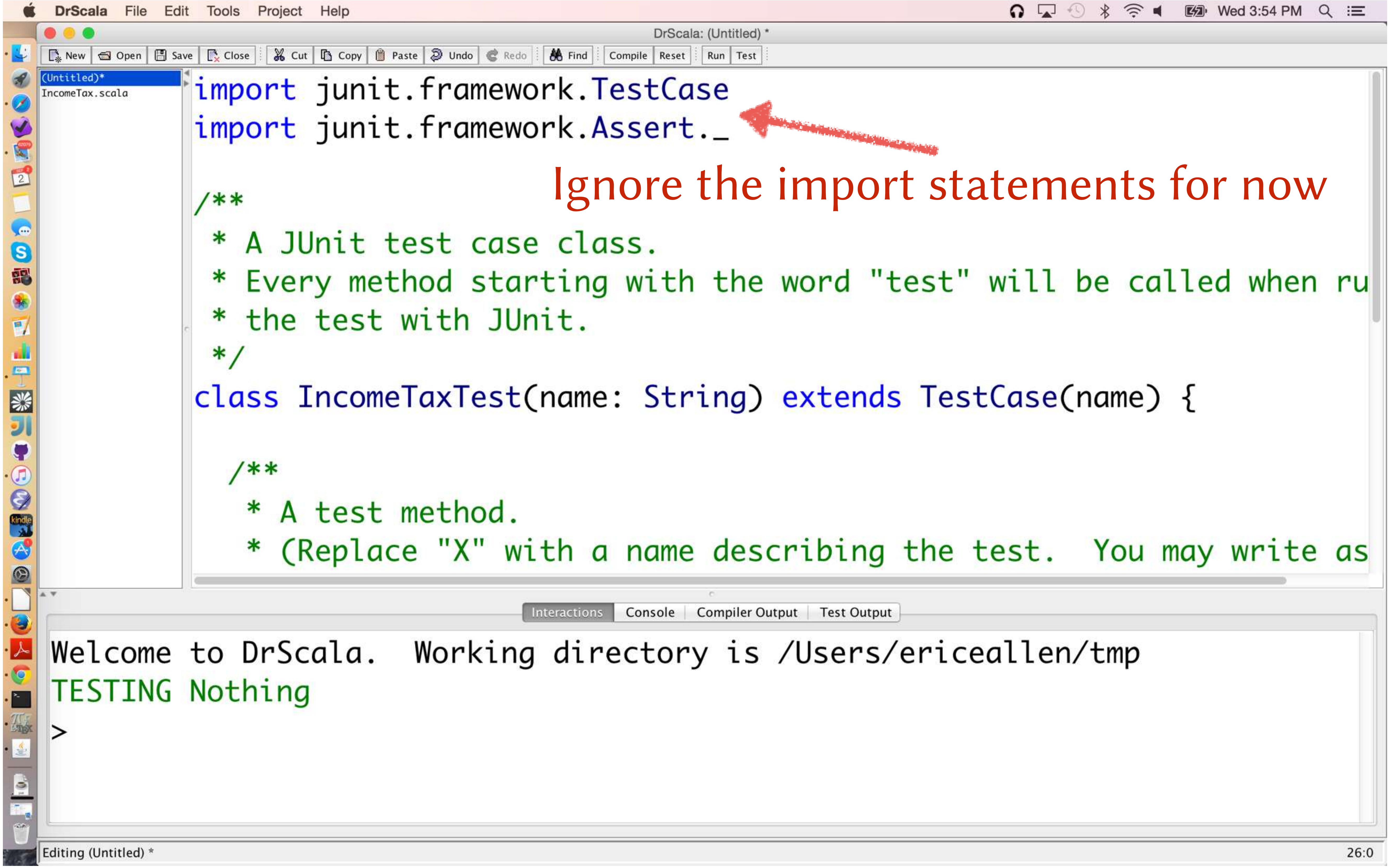
Note that this is *not* a case class



```
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
```

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

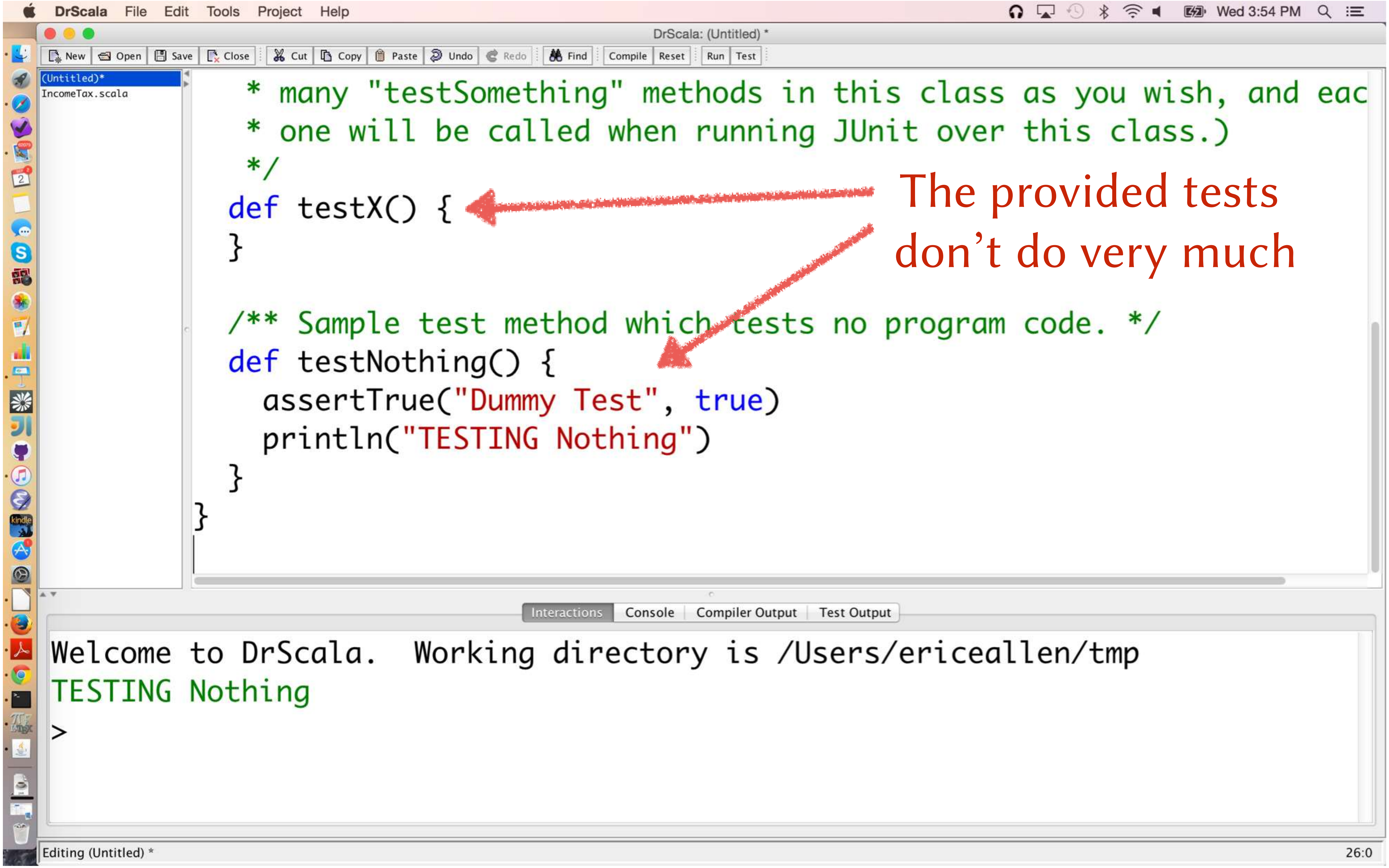
Ignore the extends clause for now



```
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

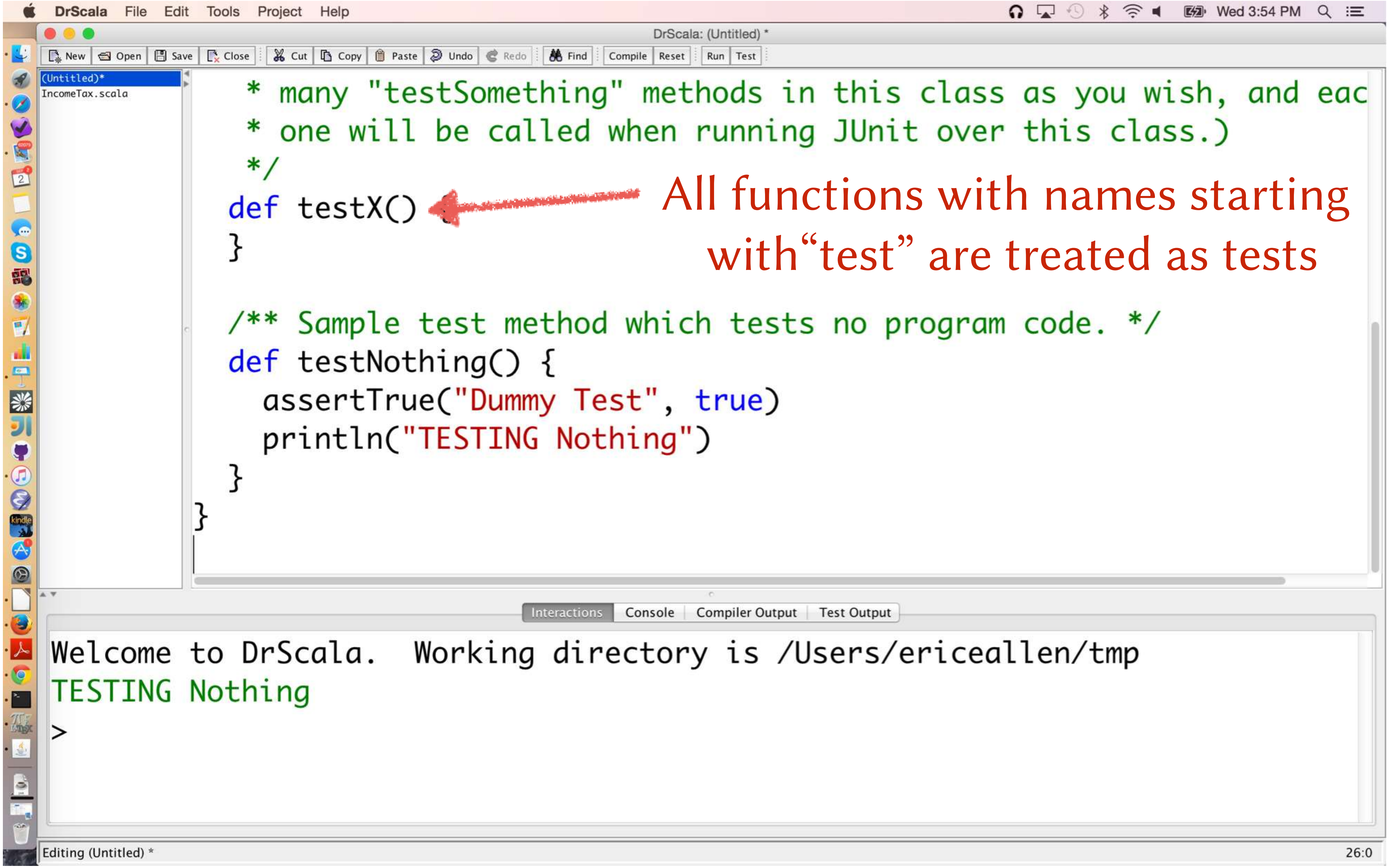
```
Welcome to DrScala.  Working directory is /Users/ericeallen/tmp
TESTING Nothing
>
```



```
   * 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")
  }
}
```

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

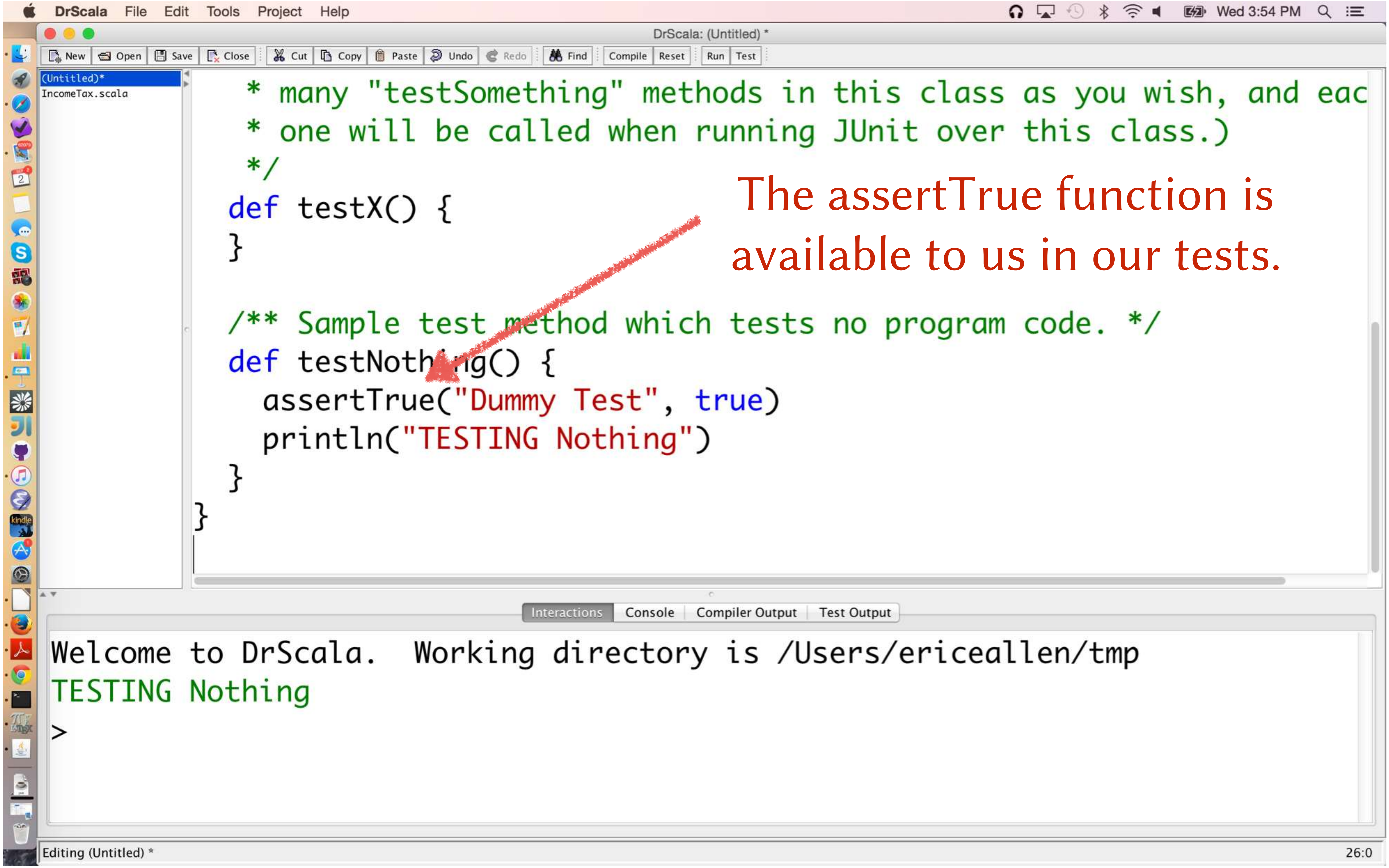
The provided tests don’t do very much


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
Interactions Console Compiler Output Test Output
Editing (Untitled) *
26:0

```
  * 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")
 }
}
```

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

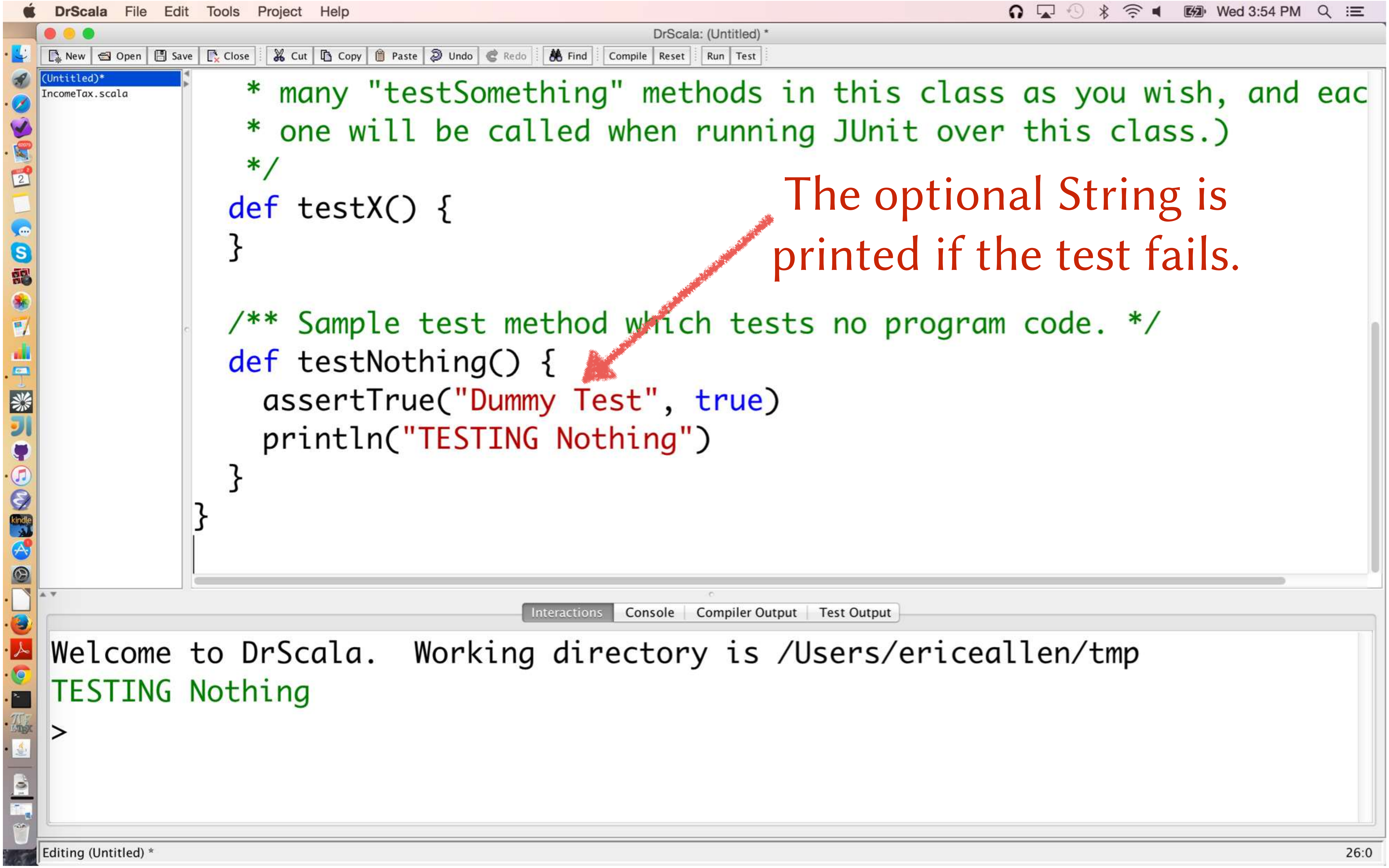
```
Welcome to DrScala.  Working directory is /Users/ericeallen/tmp
TESTING Nothing
>
```



```
   * 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")
  }
}
```

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

The assertTrue function is available to us in our tests.

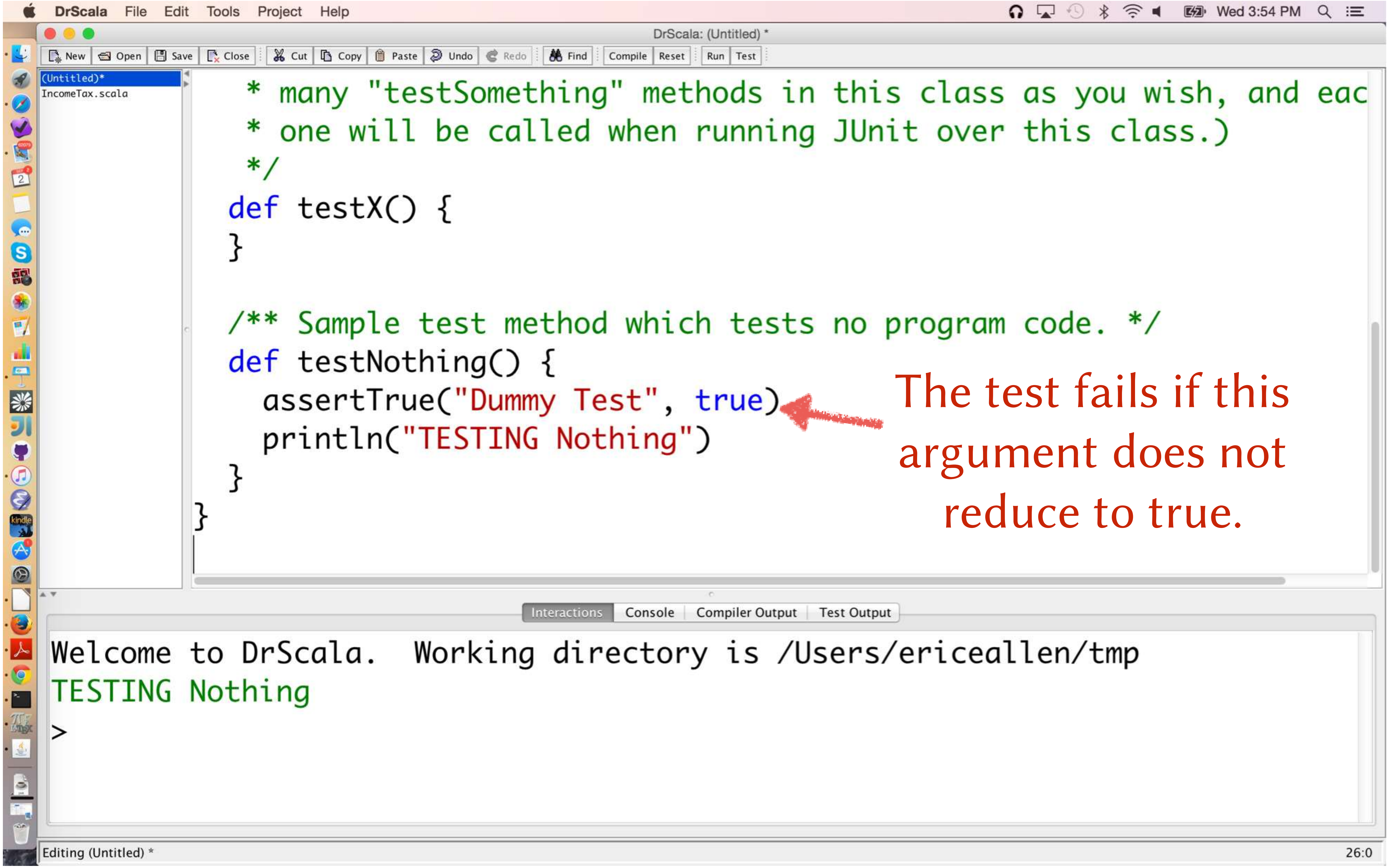


DrScala File Edit Tools Project Help
Wed 3:54 PM
DrScala: (Untitled) *
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
(Untitled)*
IncomeTax.scala
Interactions Console Compiler Output Test Output
Editing (Untitled) *
26:0

```
   * 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")
  }
}
```

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

The optional String is printed if the test fails.

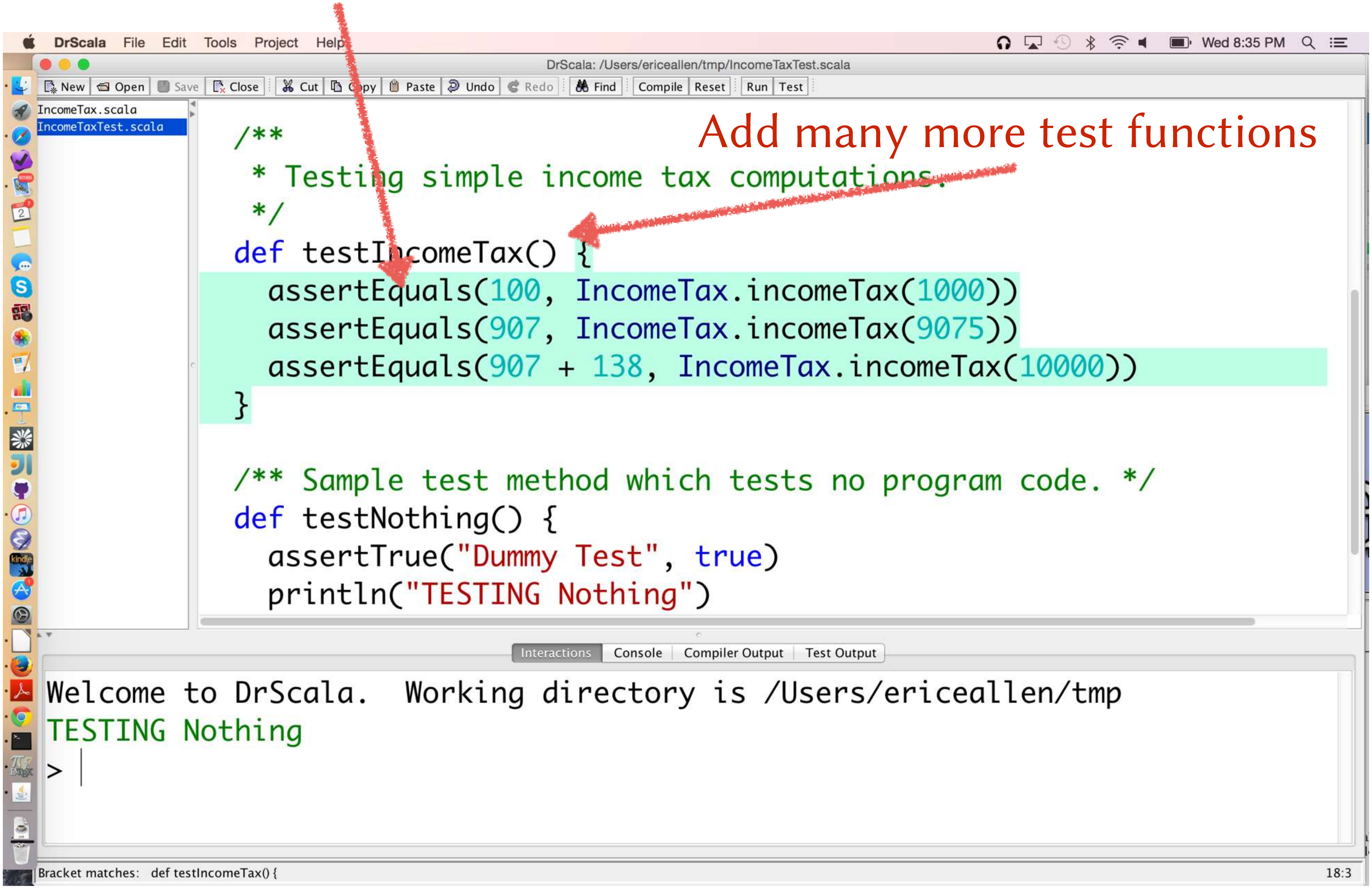


```
   * many "testSomething" methods in this class as you wish, and eac
   * one will be called when running JUnit over this class.)
   */
  def testX() {
  }

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

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

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

assertEquals fails if its two arguments are not equal



Add many more test functions

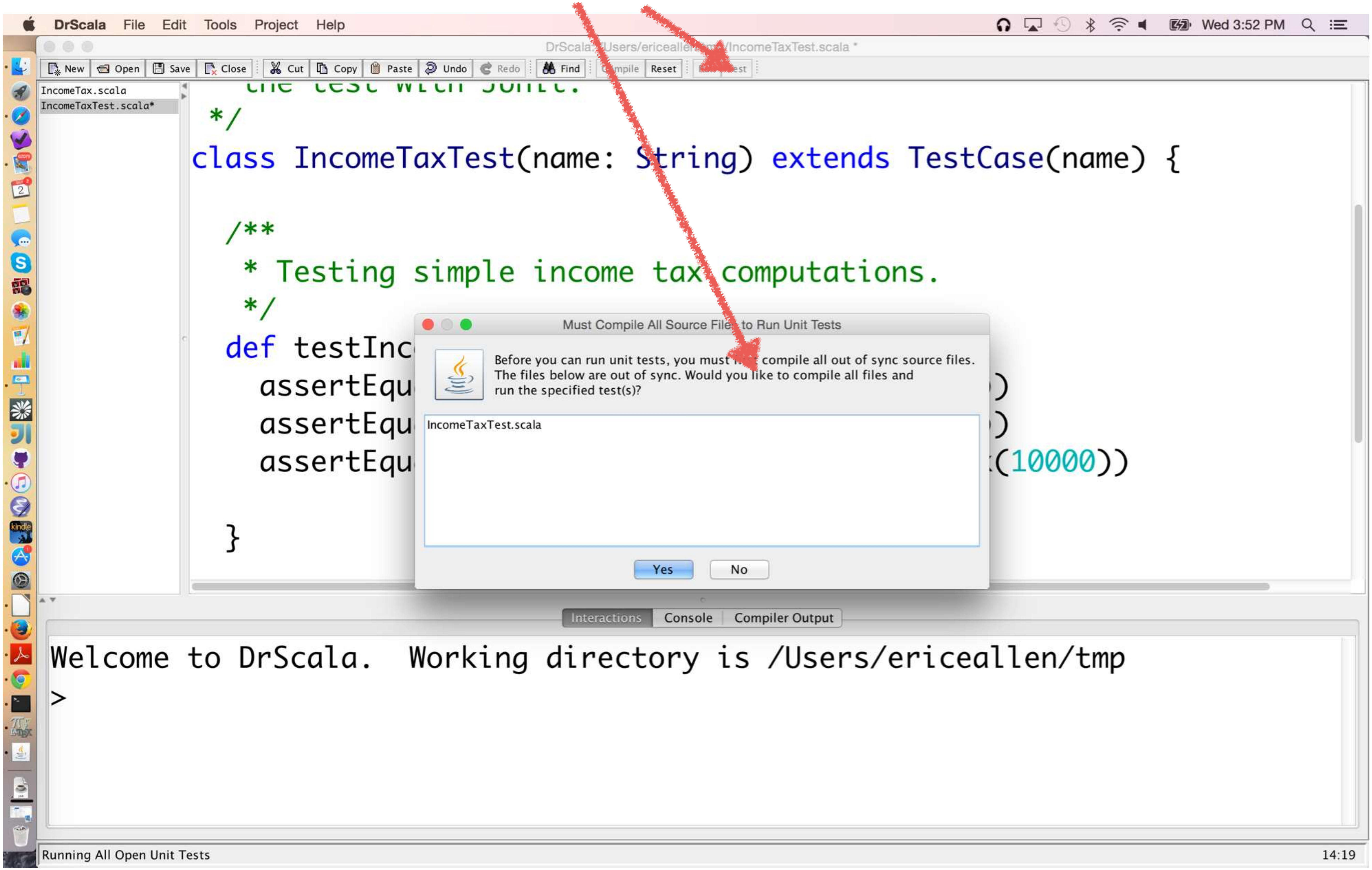
```
/**
 * Testing simple income tax computations.
 */
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000))
}

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

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

Bracket matches: def testIncomeTax() {

18:3

Hitting the Test button prompts us to compile



```
  the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * Testing simple income tax computations.
   */
  def testInc
    assertEqu                                              )
    assertEqu                                              )
    assertEqu                                              (10000))

  }
```

Must Compile All Source Files to Run Unit Tests

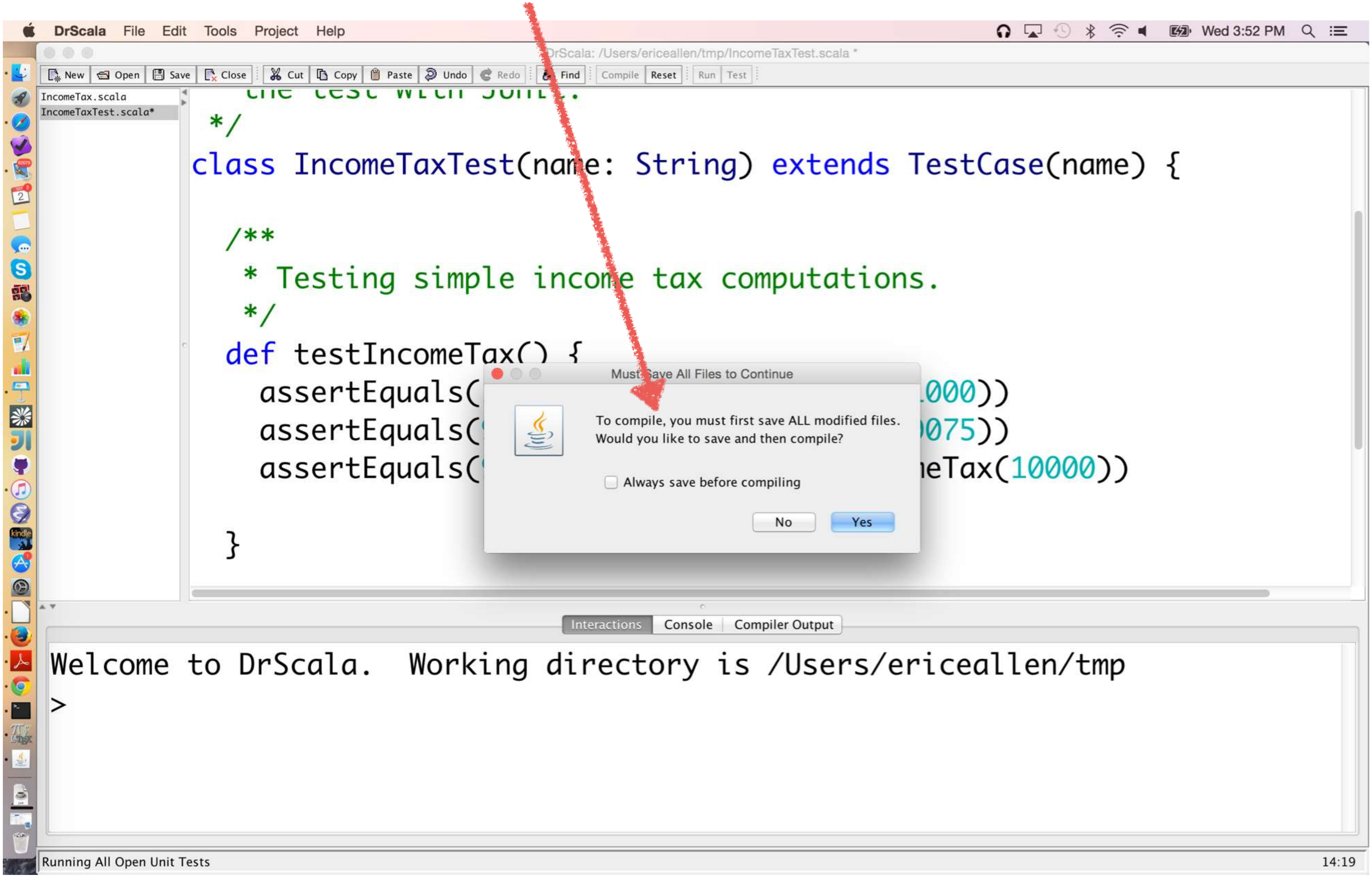
Before you can run unit tests, you must first compile all out of sync source files. The files below are out of sync. Would you like to compile all files and run the specified test(s)?

IncomeTaxTest.scala

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

Running All Open Unit Tests 14:19

Agreeing to compile prompts us to save



```
  the test with JUnit.
 */
class IncomeTaxTest(name: String) extends TestCase(name) {

  /**
   * Testing simple income tax computations.
   */
  def testIncomeTax() {
    assertEquals(                          000))
    assertEquals(                          075))
    assertEquals(                          eTax(10000))

  }
```

Must Save All Files to Continue

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

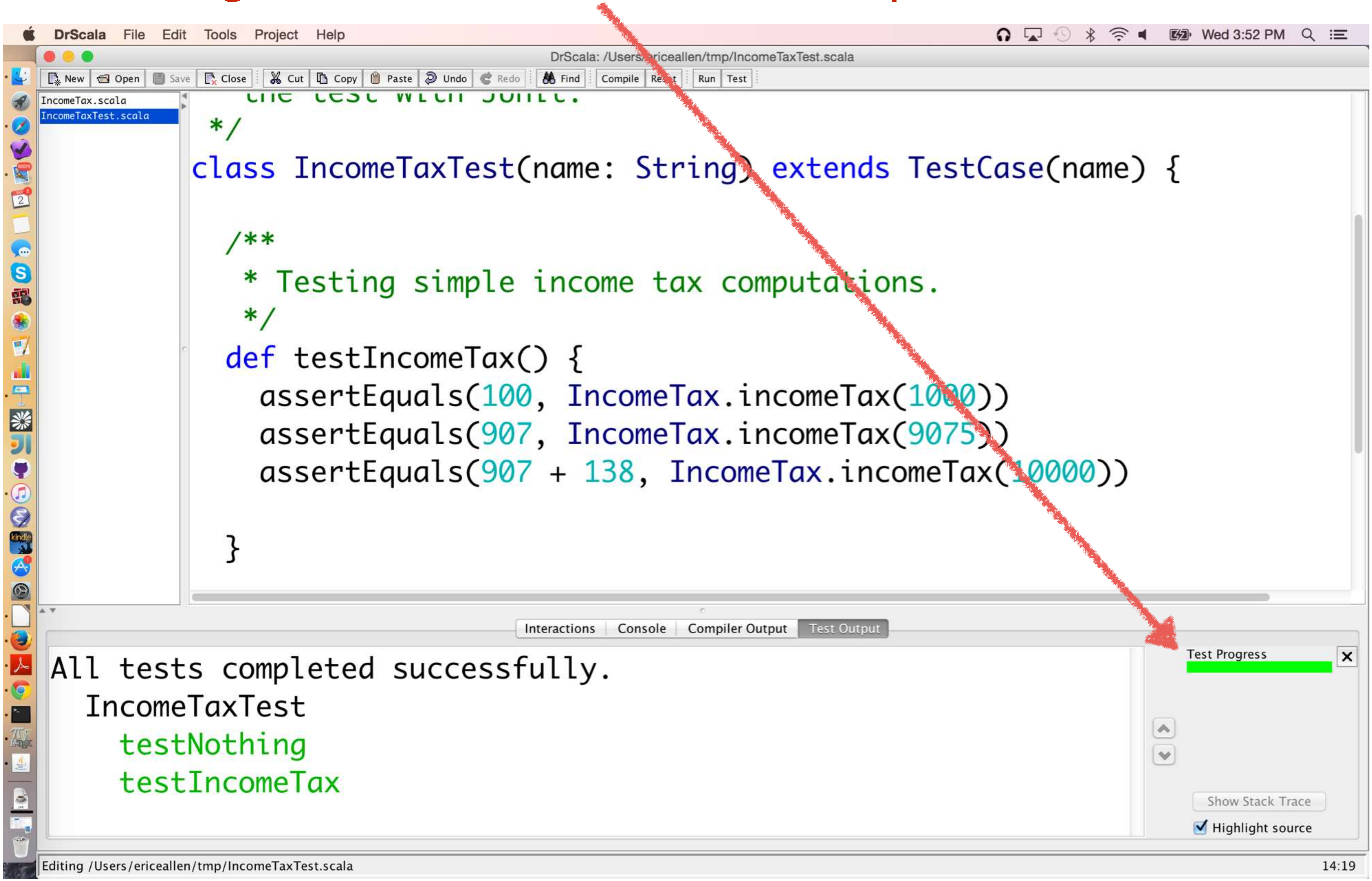
□ Always save before compiling

No | Yes

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

Running All Open Unit Tests 14:19

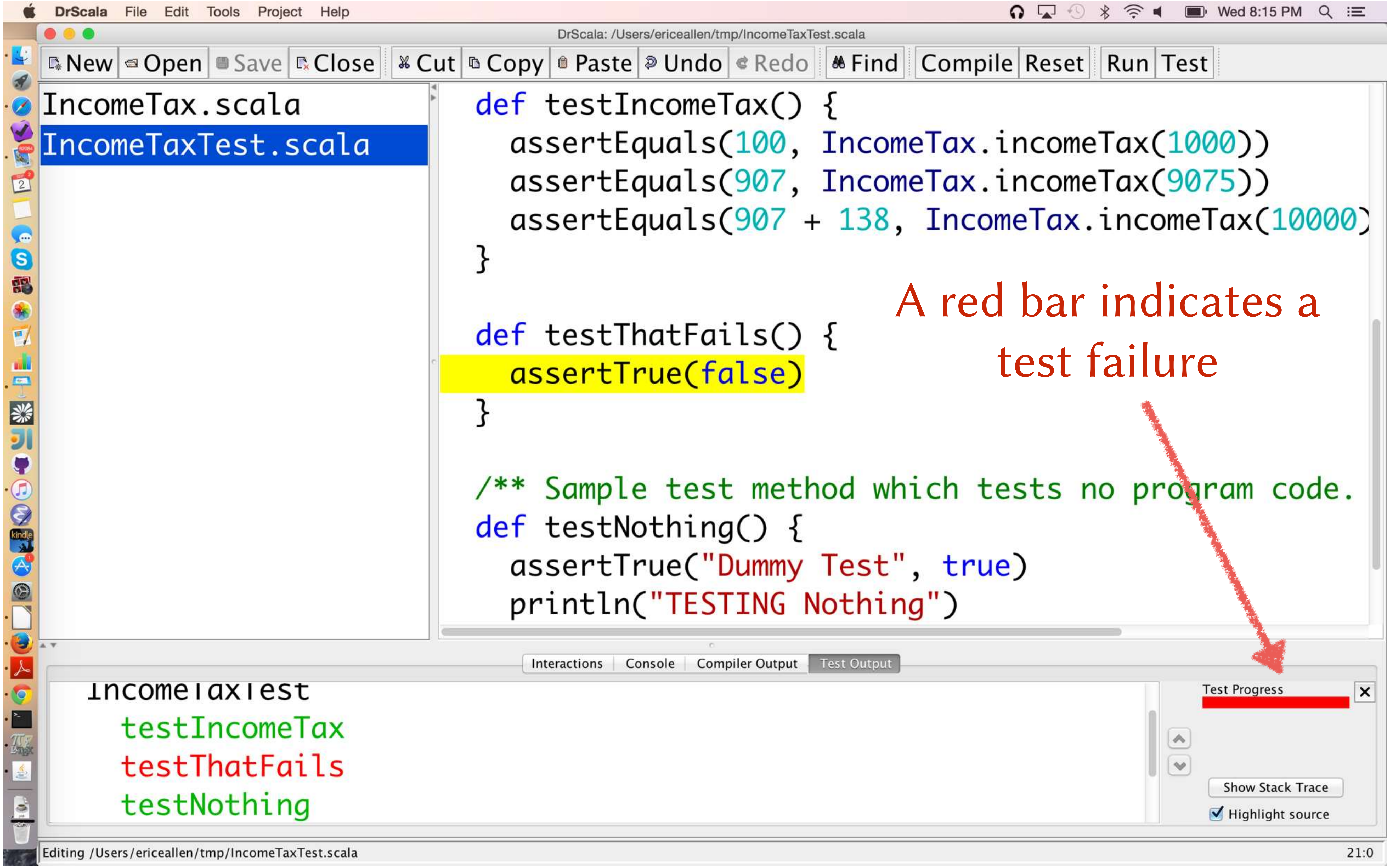
A green bar indicates that all tests passed



```
  the test with JUnit.
 */
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))

  }
```

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

Editing /Users/ericeallen/tmp/IncomeTaxTest.scala 14:19



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

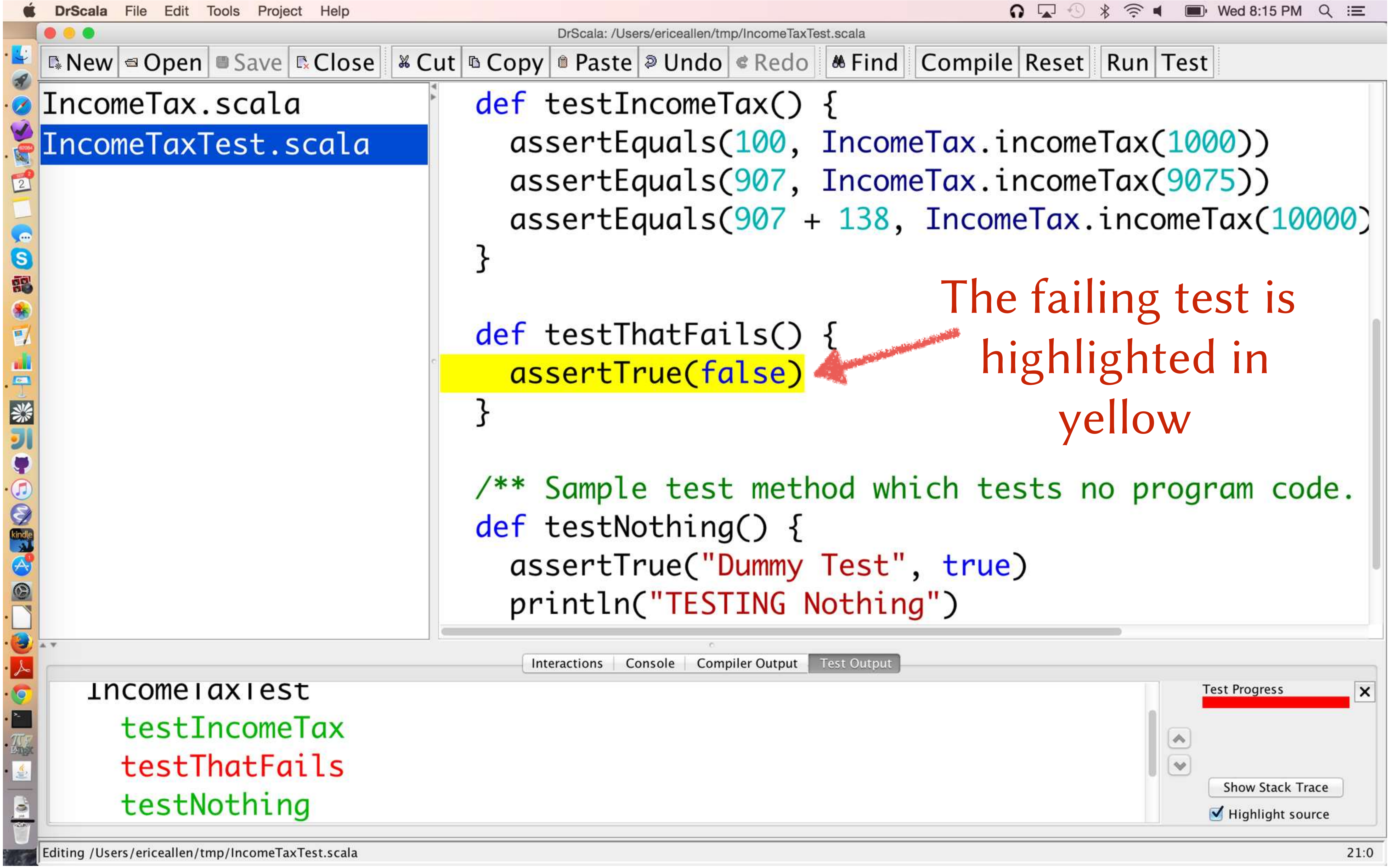
IncomeTax.scala
IncomeTaxTest.scala

```
  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")
```

```
IncomeTaxTest
  testIncomeTax
  testThatFails
  testNothing
```

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

A red bar indicates a test failure



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
Interactions Console Compiler Output Test Output
Test Progress
Show Stack Trace
☑ Highlight source

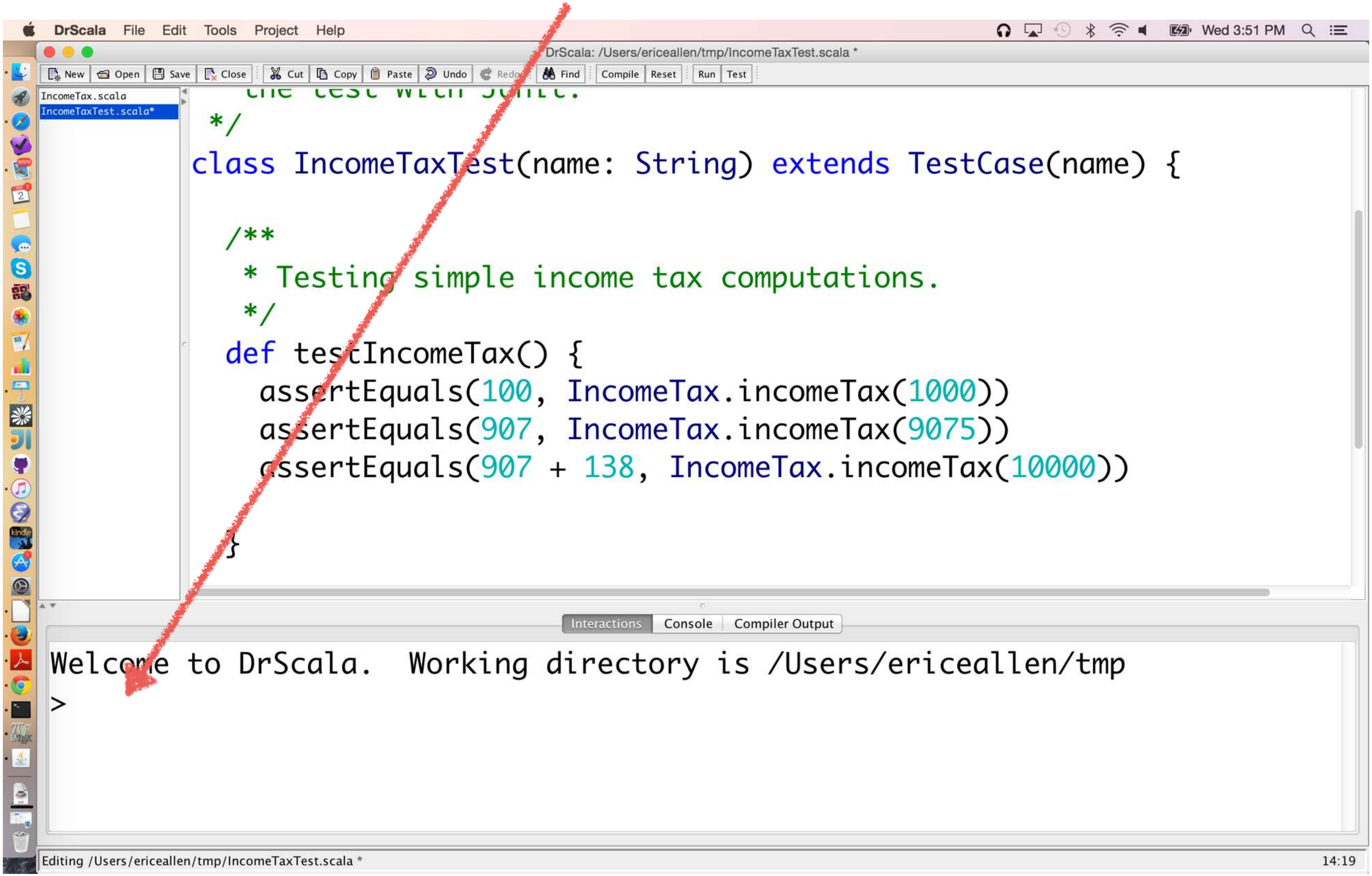
```
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000)
}

def testThatFails() {
  assertTrue(false)
}

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

```
IncomeTaxTest
  testIncomeTax
  testThatFails
  testNothing
```

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

The failing test is highlighted in yellow

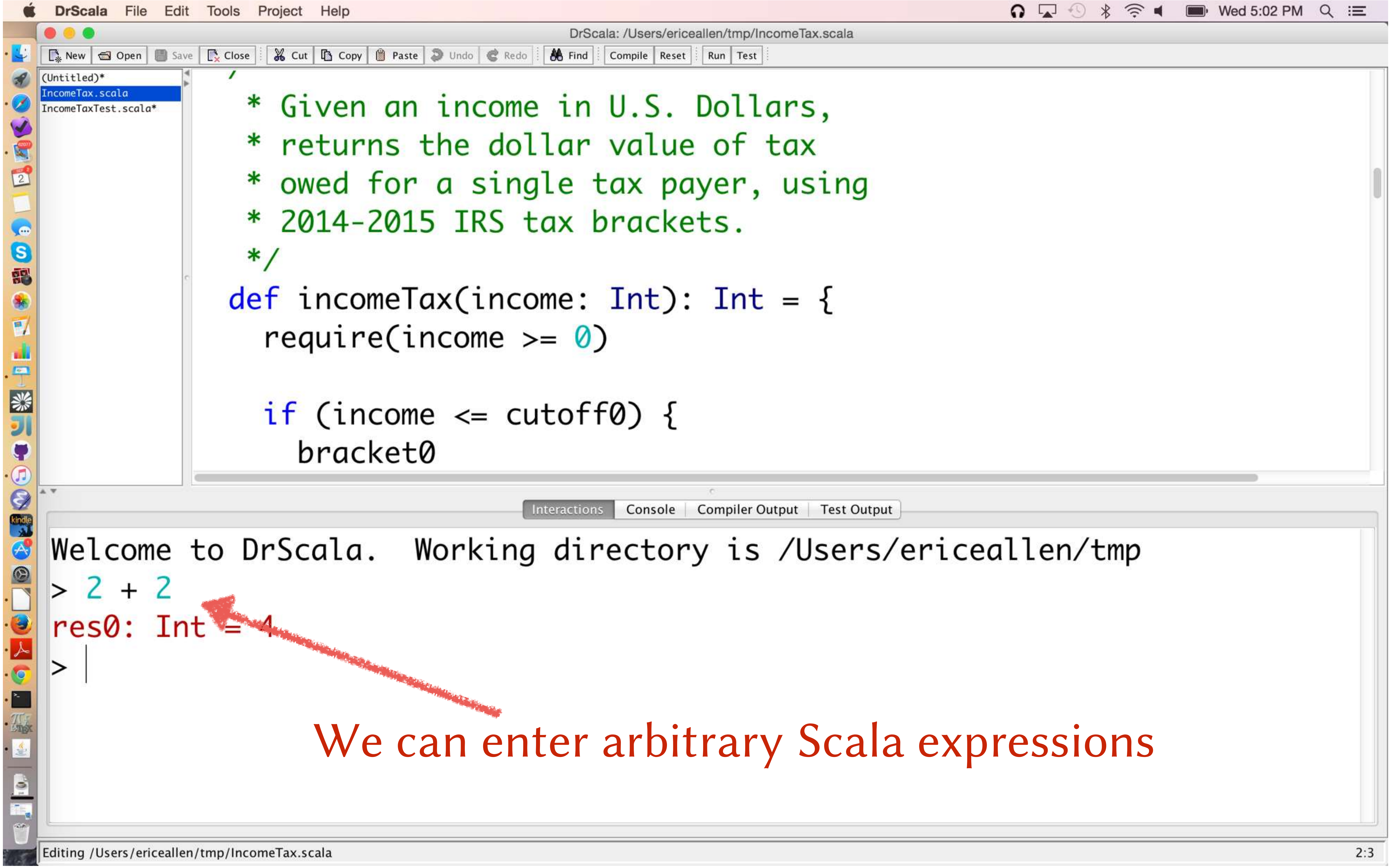
To interact with our program, we use the Interactions Pane



```
  the test with JUnit.
 */
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))

  }
```

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

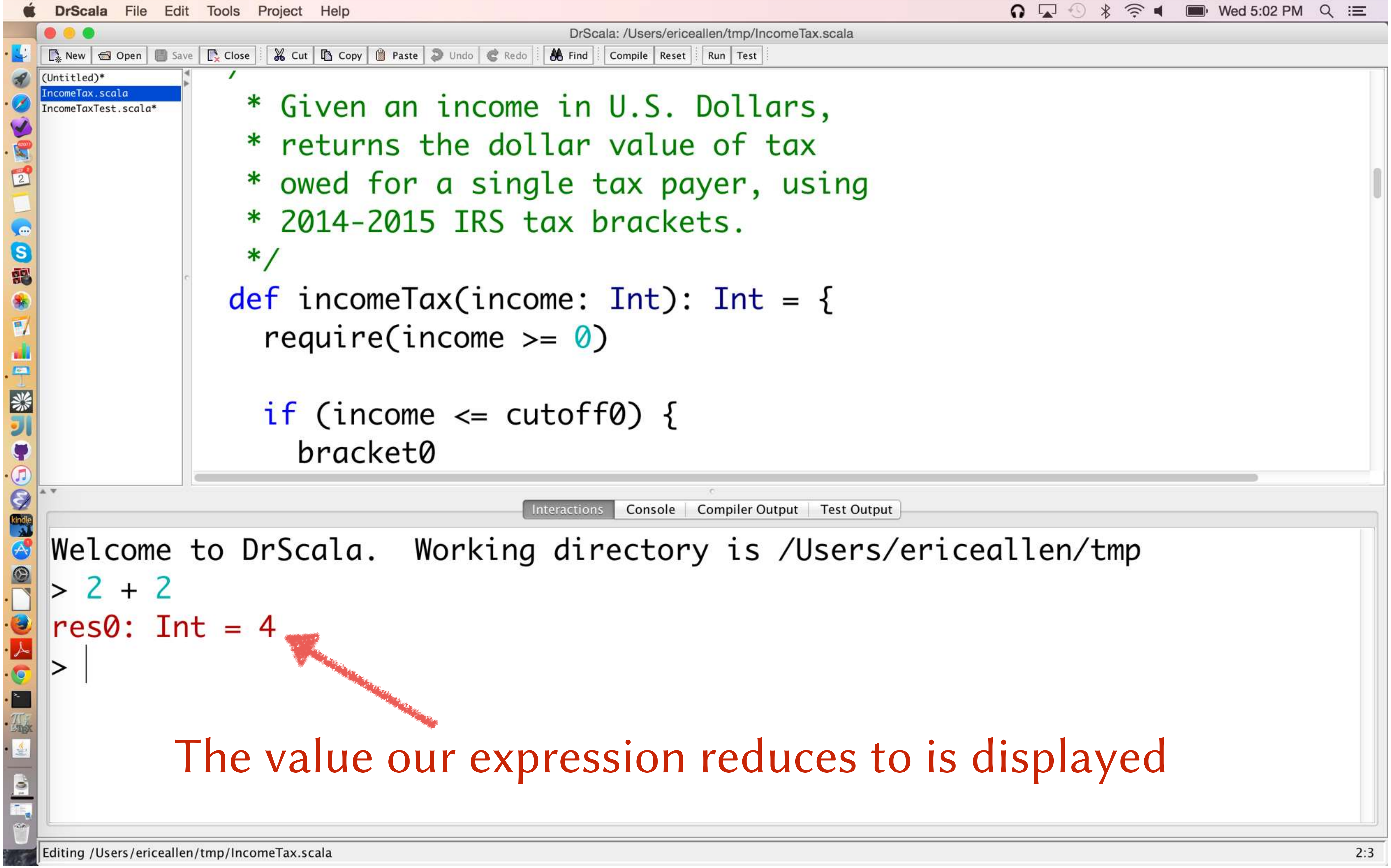
Editing /Users/ericeallen/tmp/IncomeTaxTest.scala * 14:19



```
/
 * 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
```

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

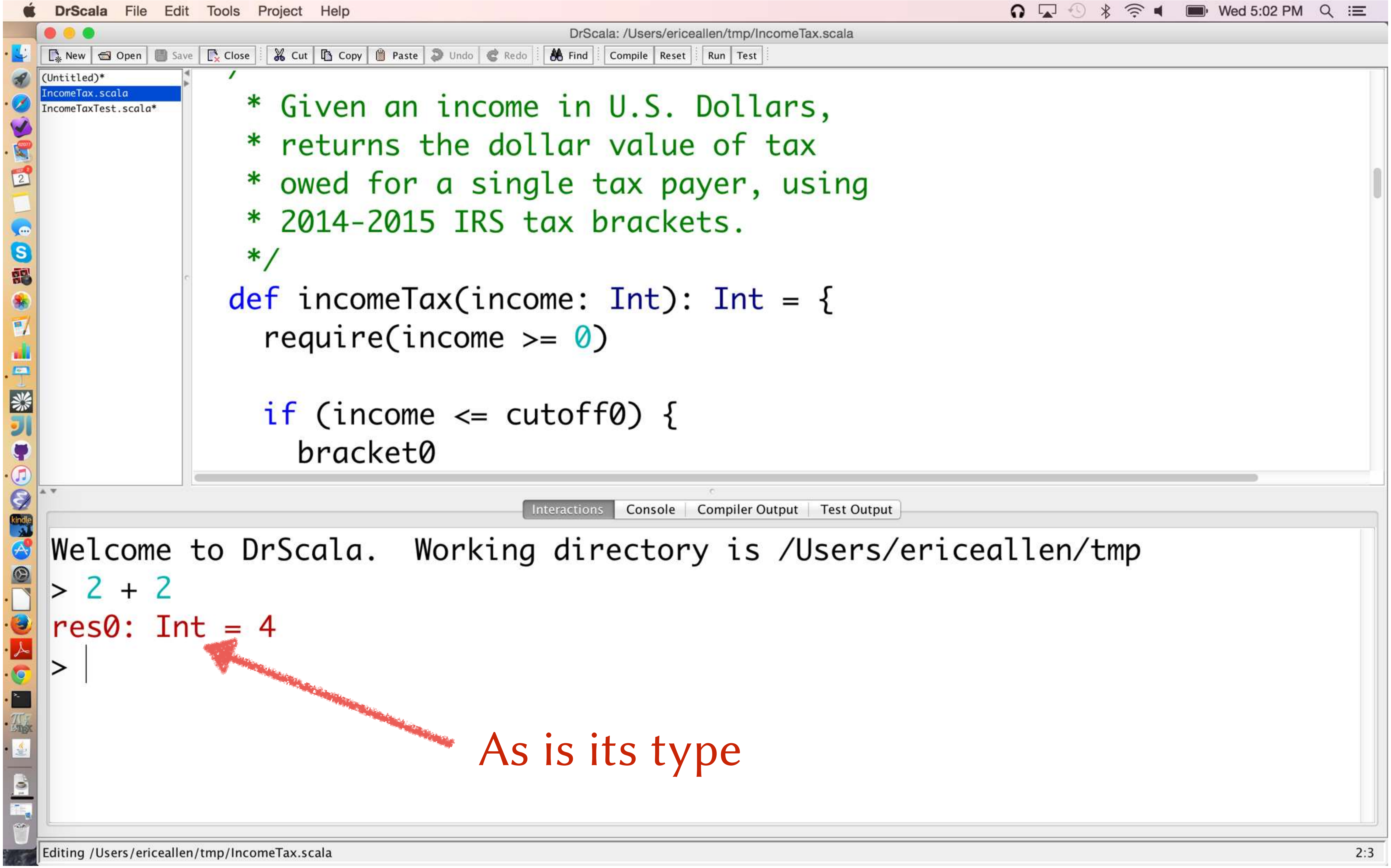
We can enter arbitrary Scala expressions



```
 /
  * 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
```

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

The value our expression reduces to is displayed



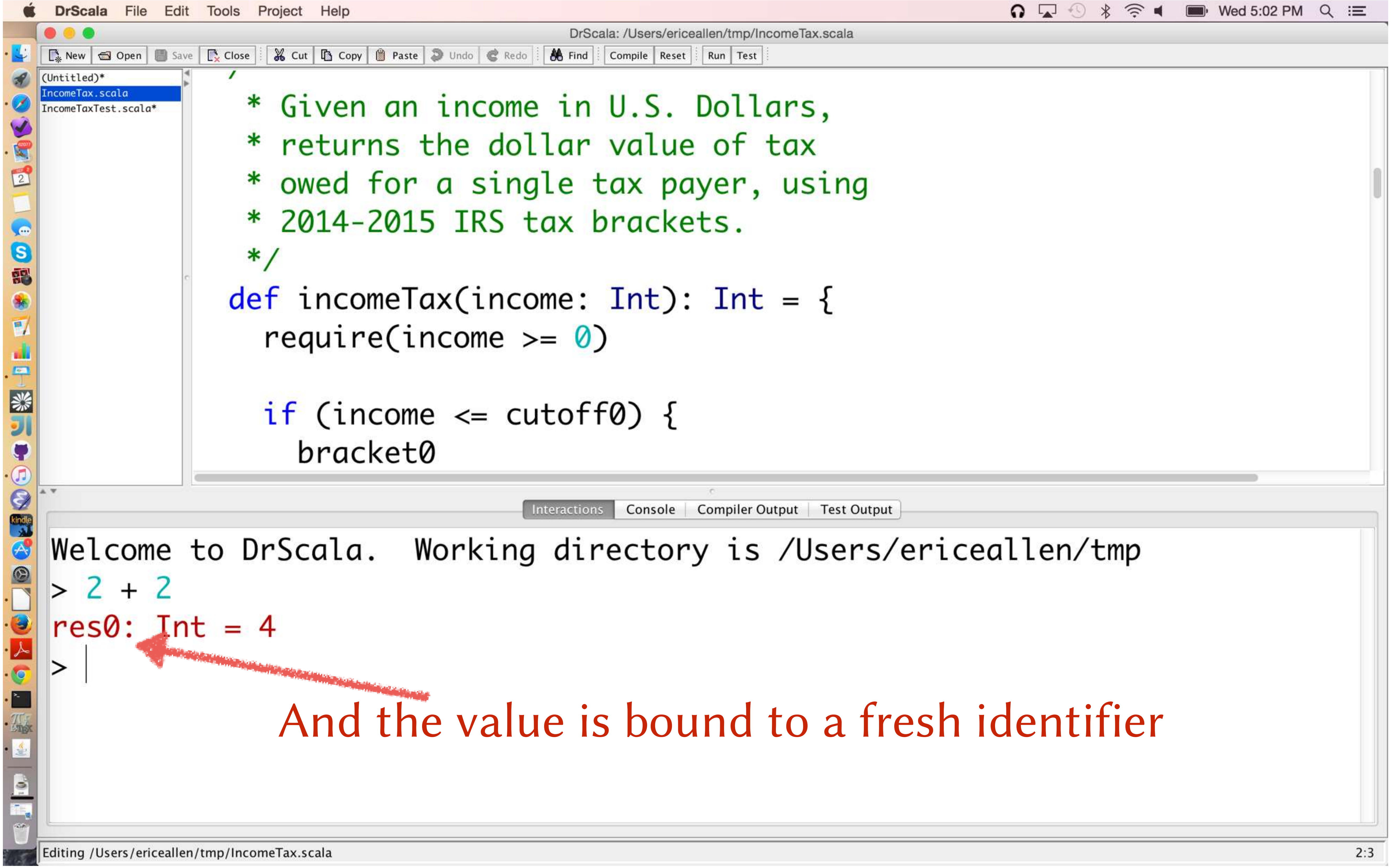
```
/
  * Given an income in U.S. Dollars,
  * returns the dollar value of tax
  * owed for a single tax payer, using
  * 2014-2015 IRS tax brackets.
  */
def incomeTax(income: Int): Int = {
  require(income >= 0)

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

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

Editing /Users/ericeallen/tmp/IncomeTax.scala

2:3

As is its type

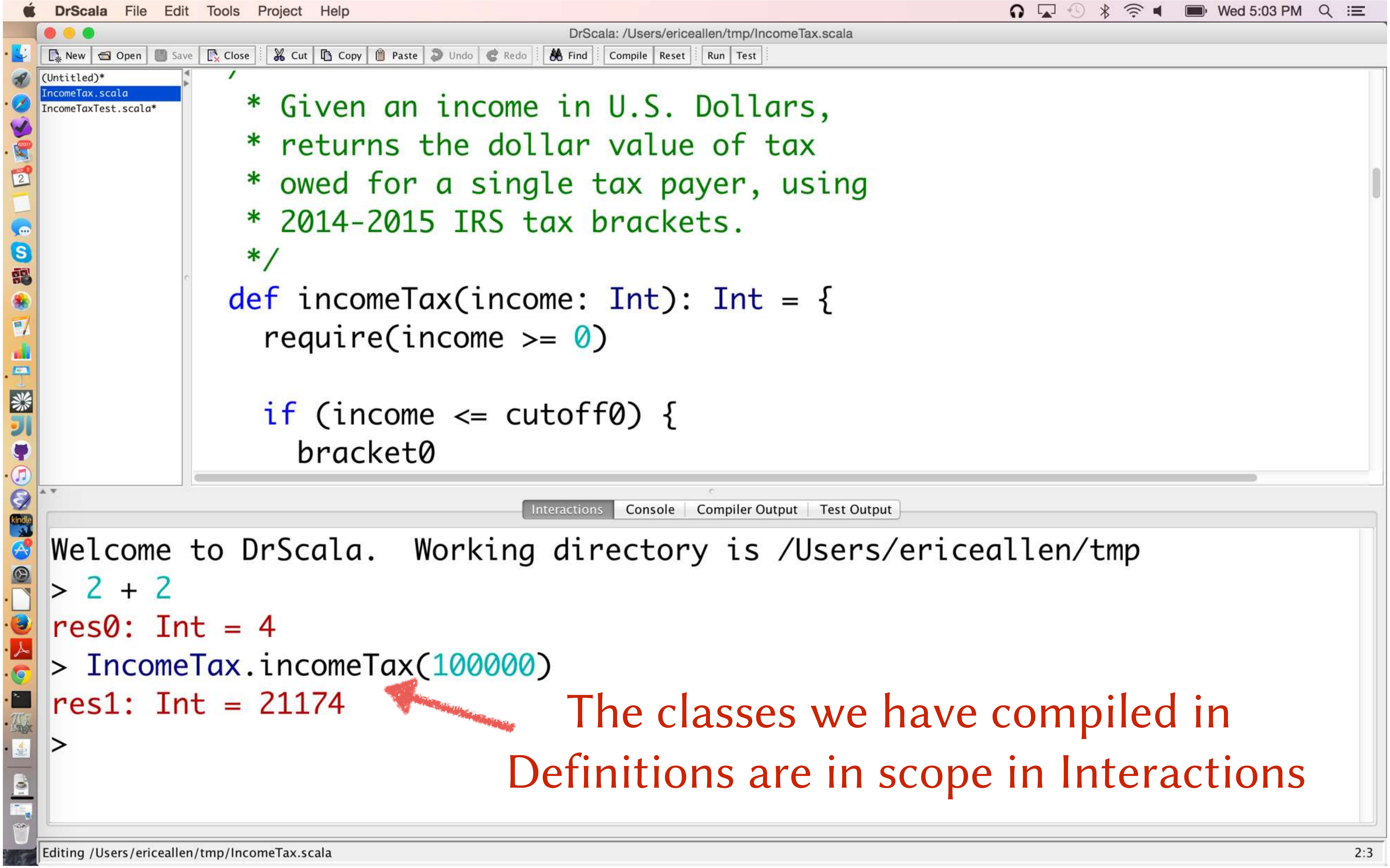


```
 /
  * Given an income in U.S. Dollars,
  * returns the dollar value of tax
  * owed for a single tax payer, using
  * 2014-2015 IRS tax brackets.
  */
def incomeTax(income: Int): Int = {
  require(income >= 0)

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

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

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

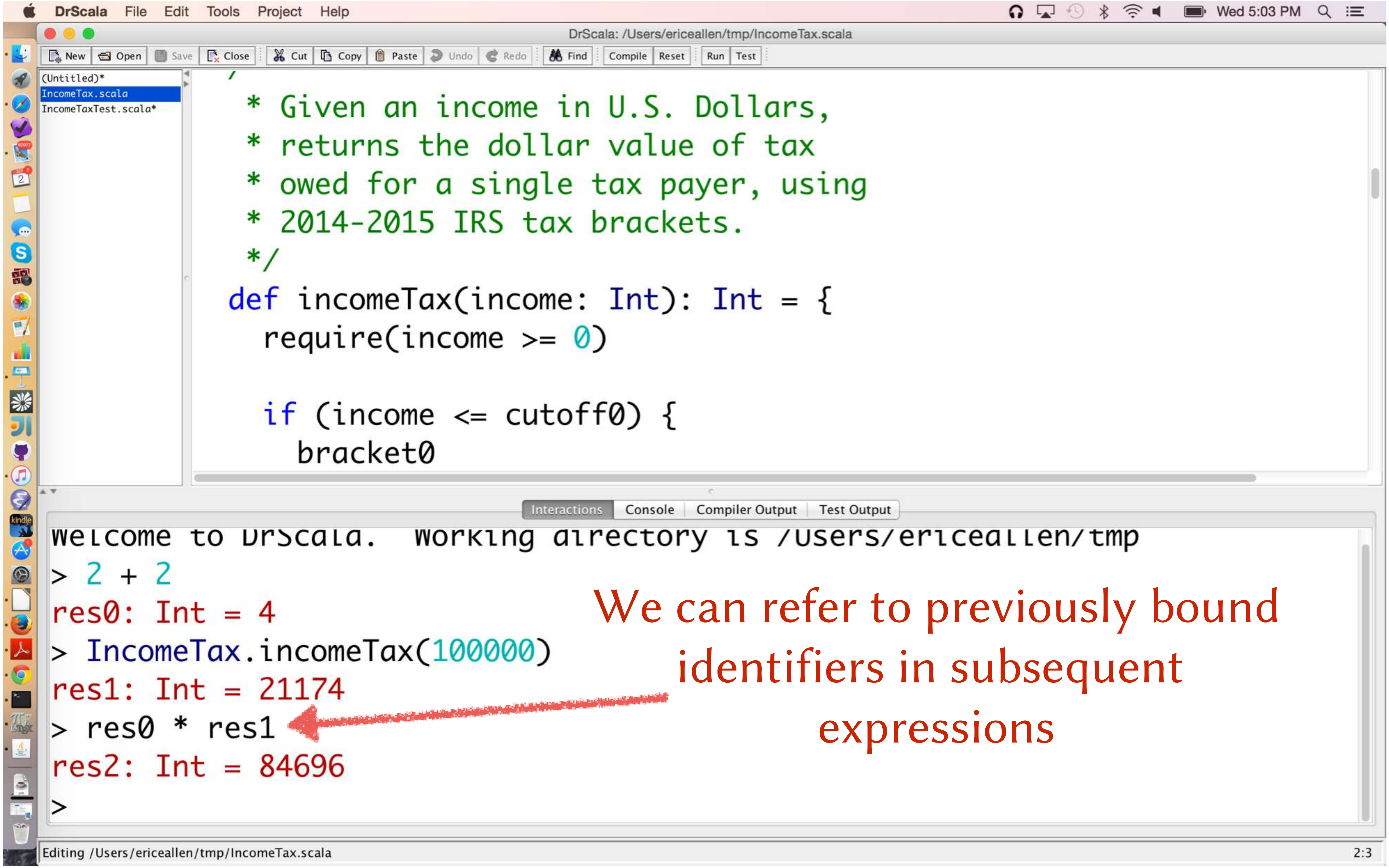
And the value is bound to a fresh identifier



```
/
 * 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
```

```
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

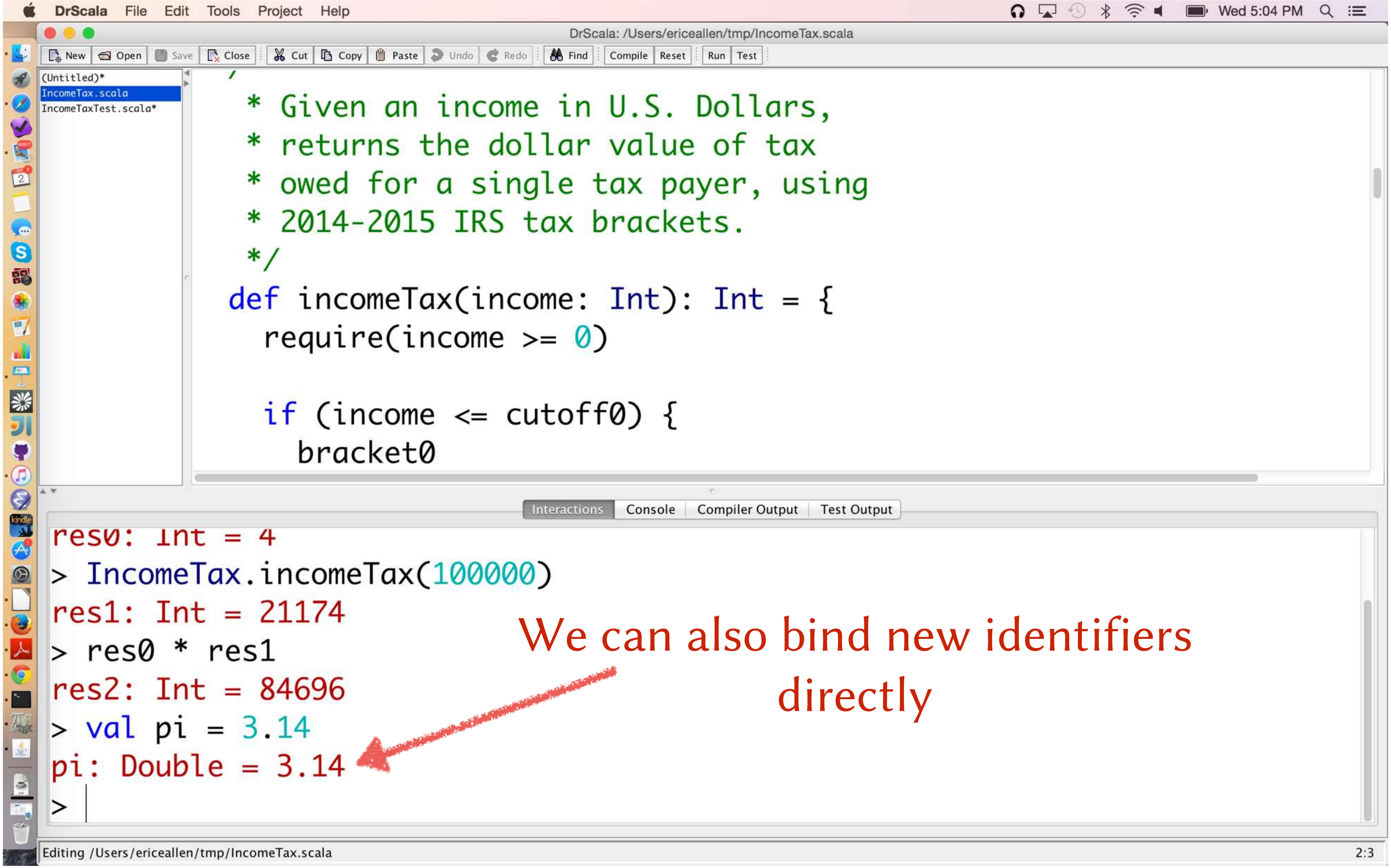

DrScala File Edit Tools Project Help
Wed 5:03 PM
DrScala: /Users/ericeallen/tmp/IncomeTax.scala
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
(Untitled)*
IncomeTax.scala
IncomeTaxTest.scala*
Interactions Console Compiler Output Test Output

```
 * 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
```

```
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
>
```

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

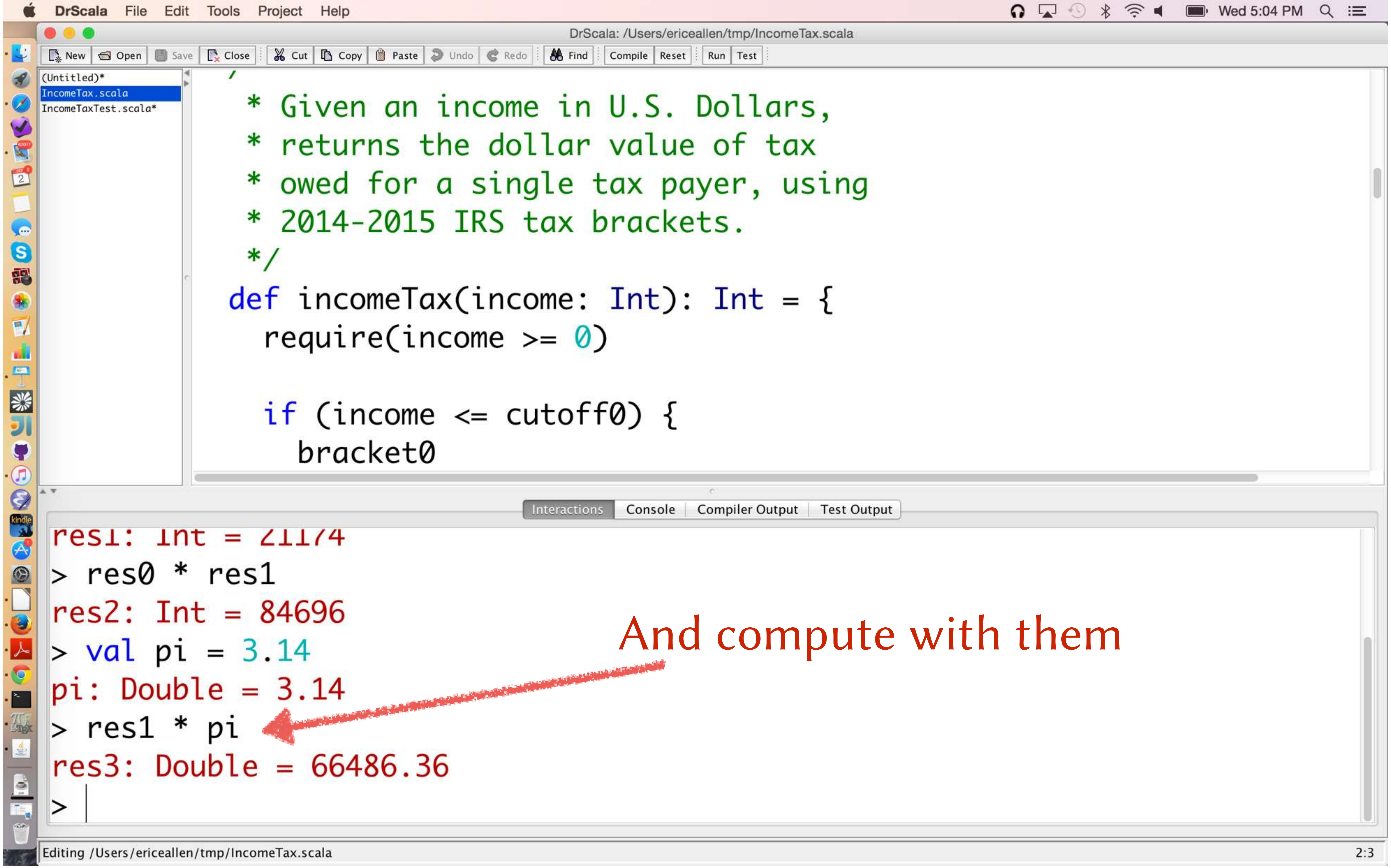
We can refer to previously bound identifiers in subsequent expressions



```
  * 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
```

```
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

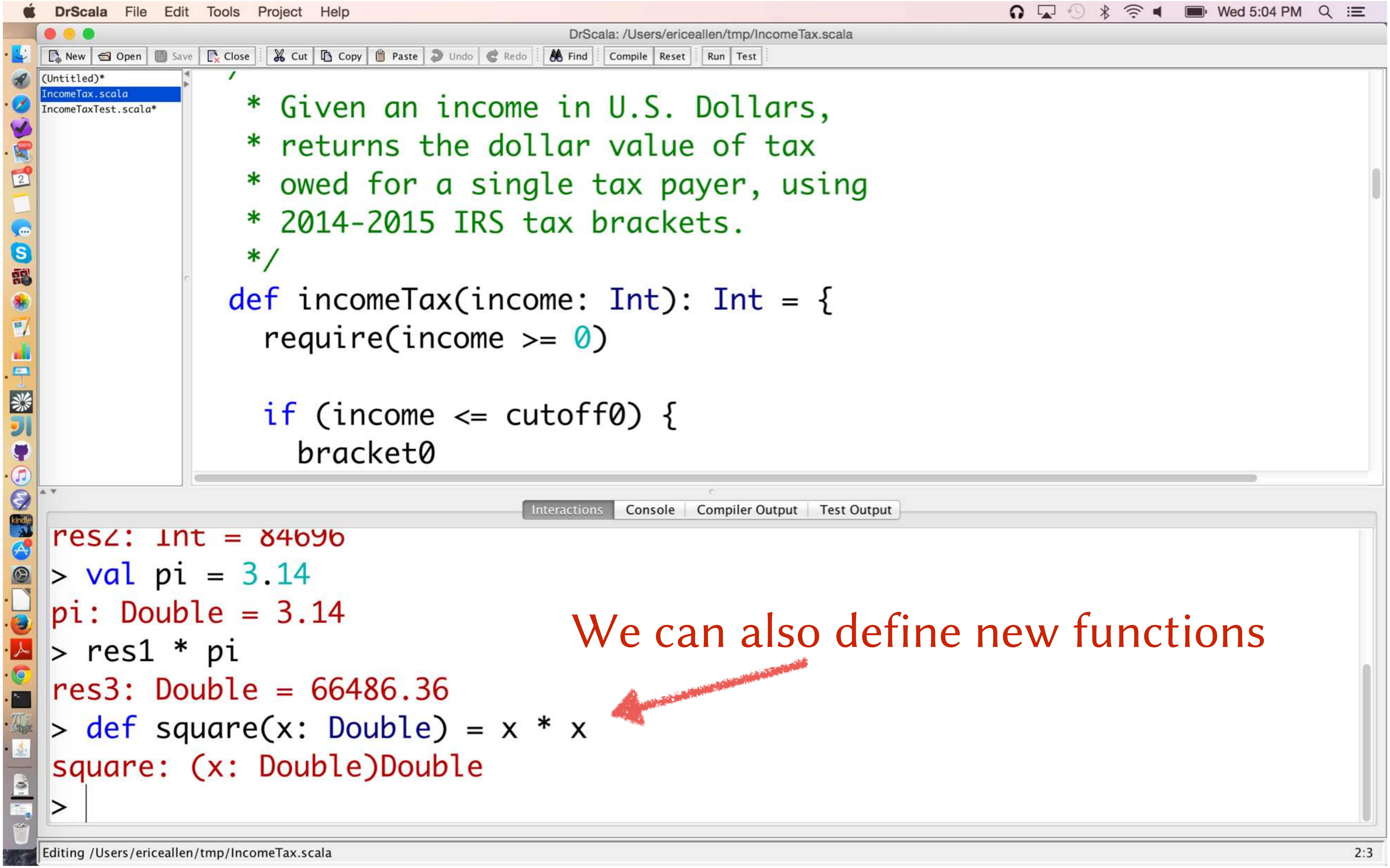


```
 * Given an income in U.S. Dollars,
 * returns the dollar value of tax
 * owed for a single tax payer, using
 * 2014-2015 IRS tax brackets.
 */
def incomeTax(income: Int): Int = {
  require(income >= 0)

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

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

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

And compute with them

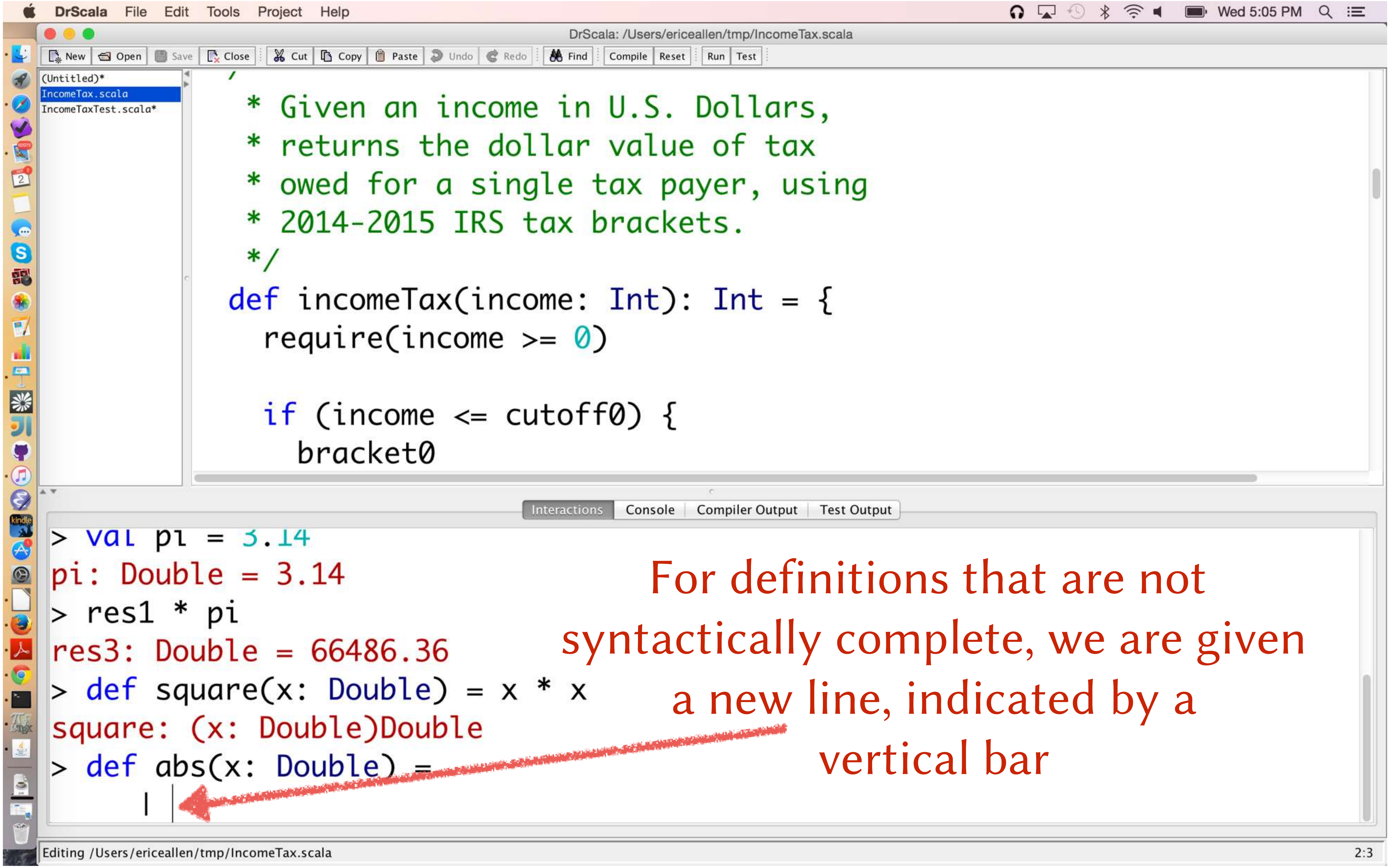


```
/
 * Given an income in U.S. Dollars,
 * returns the dollar value of tax
 * owed for a single tax payer, using
 * 2014-2015 IRS tax brackets.
 */
def incomeTax(income: Int): Int = {
  require(income >= 0)

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

```
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
>
```

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

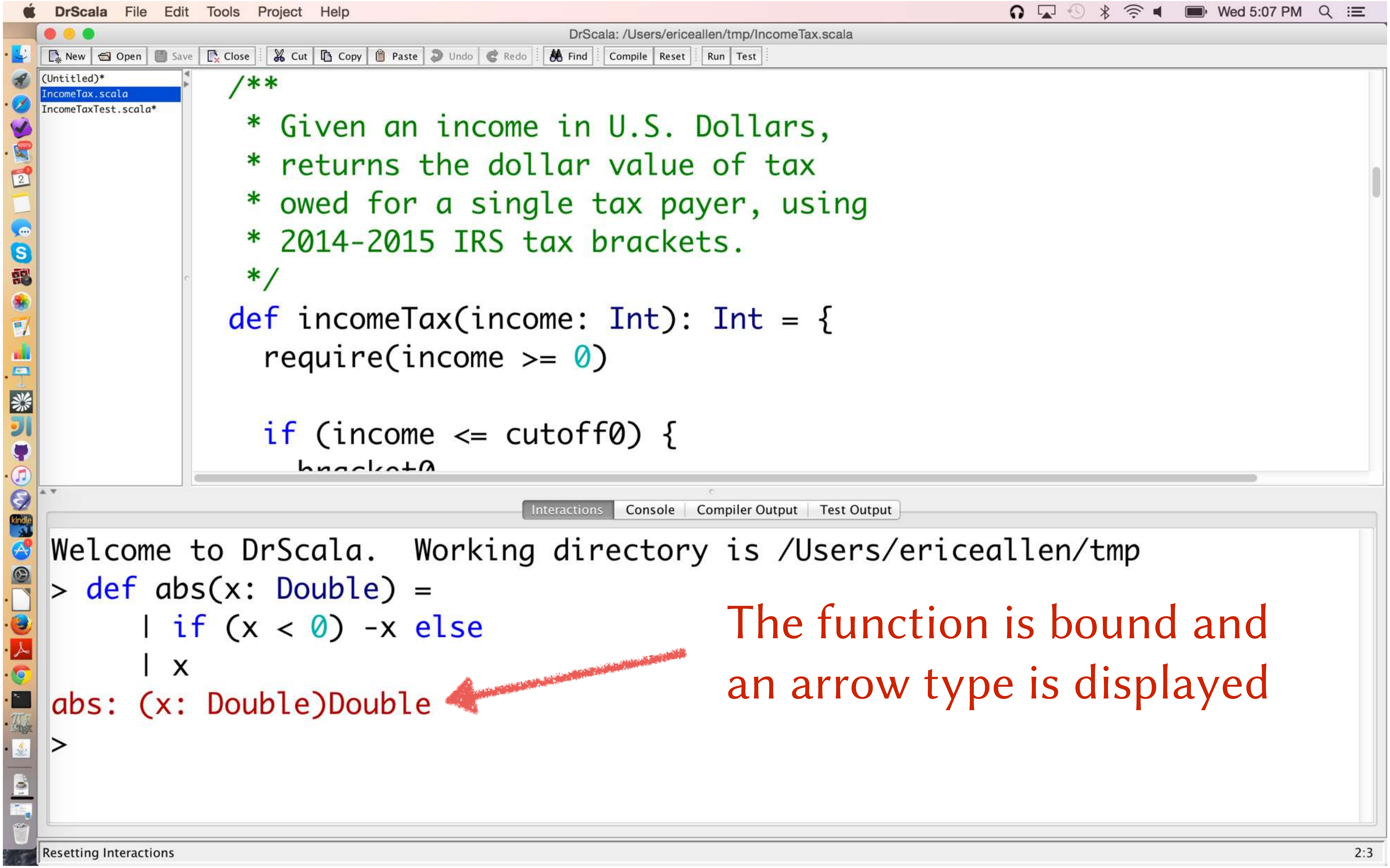
We can also define new functions



```
  * 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
```

```
> 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) =
      |
```

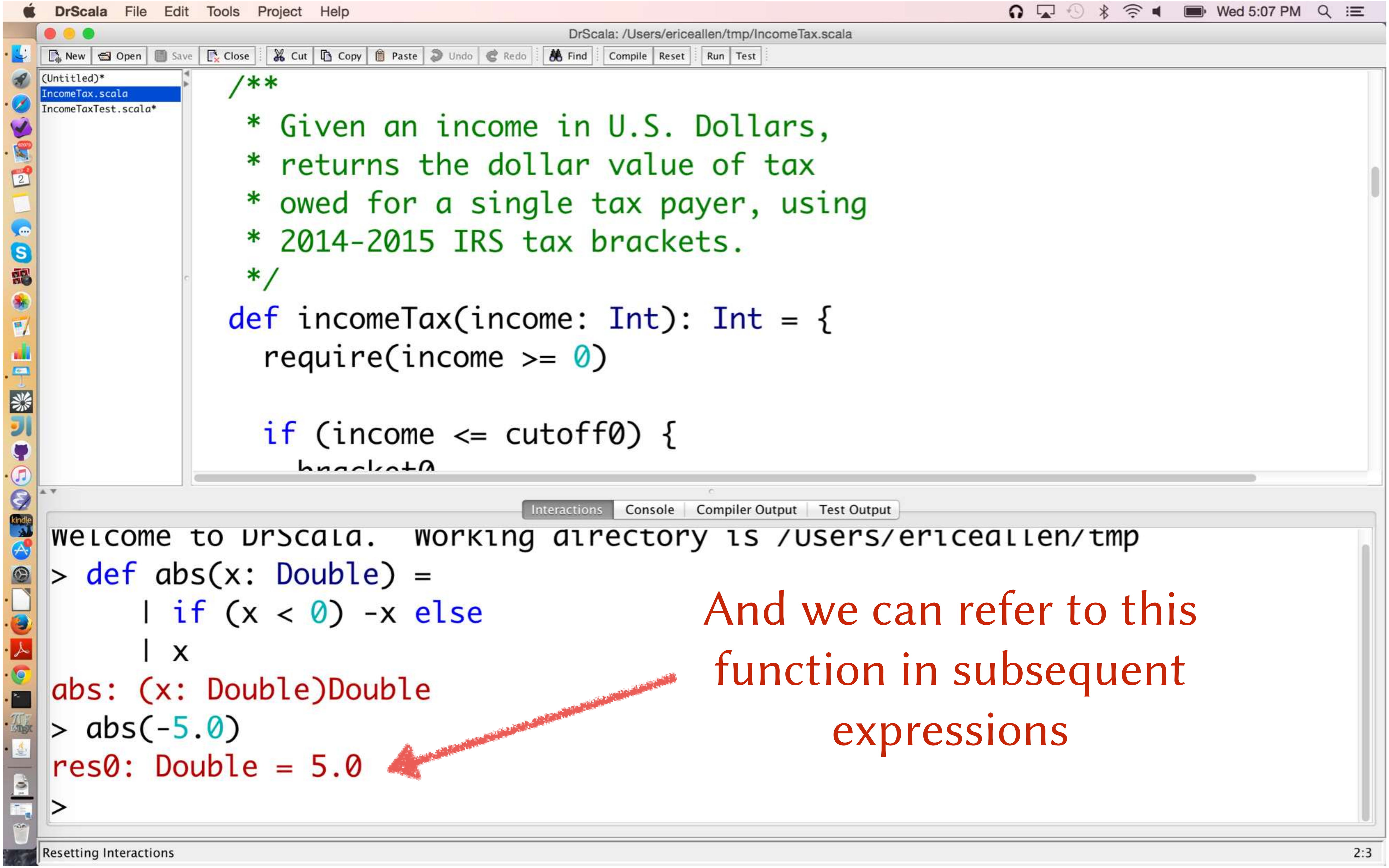
For definitions that are not syntactically complete, we are given a new line, indicated by a vertical bar



```
/**
 * 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
```

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

The function is bound and an arrow type is displayed

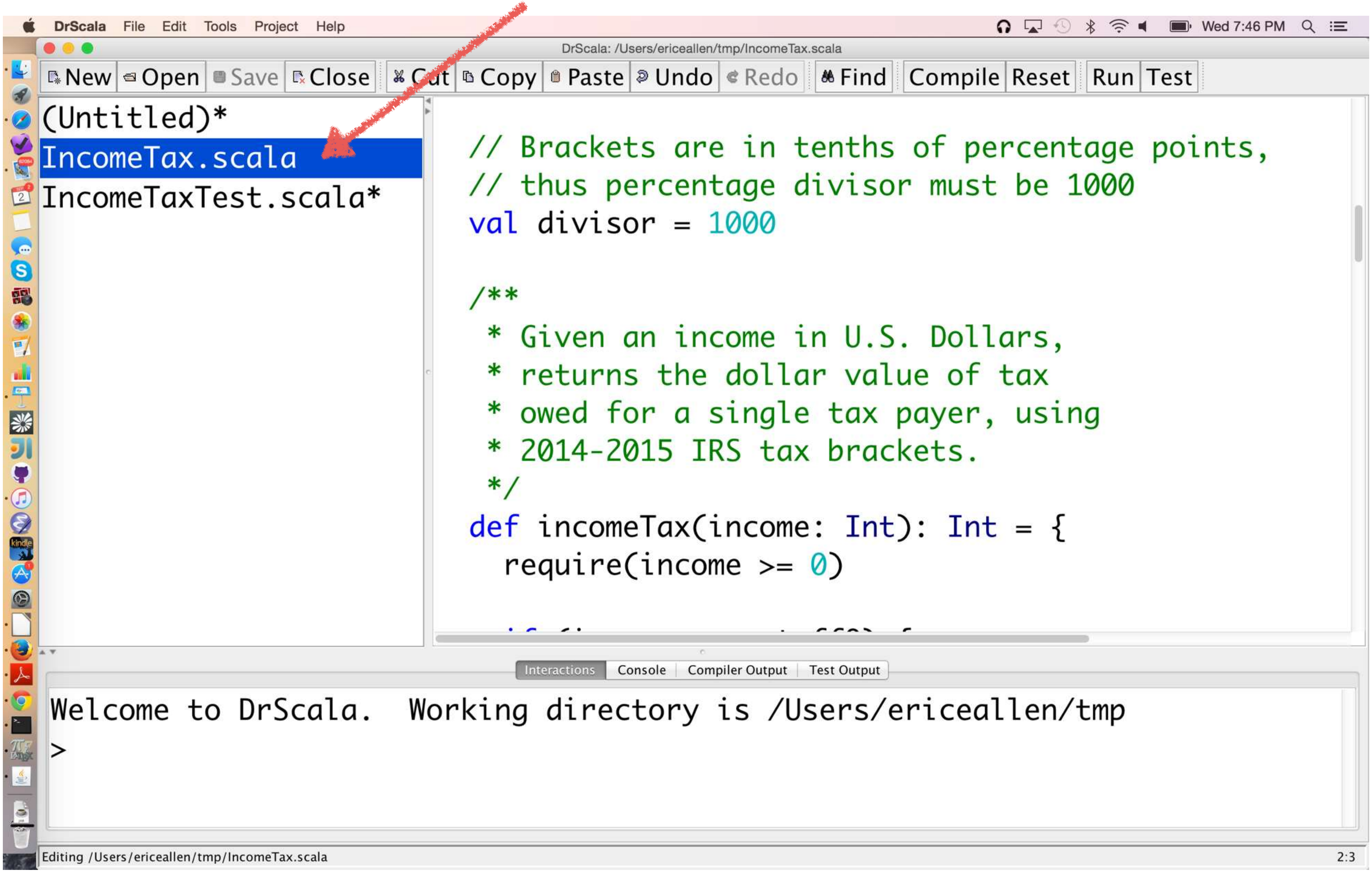


```scala
/**
 * Given an income in U.S. Dollars,
 * returns the dollar value of tax
 * owed for a single tax payer, using
 * 2014-2015 IRS tax brackets.
 */
def incomeTax(income: Int): Int = {
  require(income >= 0)

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

```
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
>
```

And we can refer to this function in subsequent expressions

We can click on the file to appear in Definitions

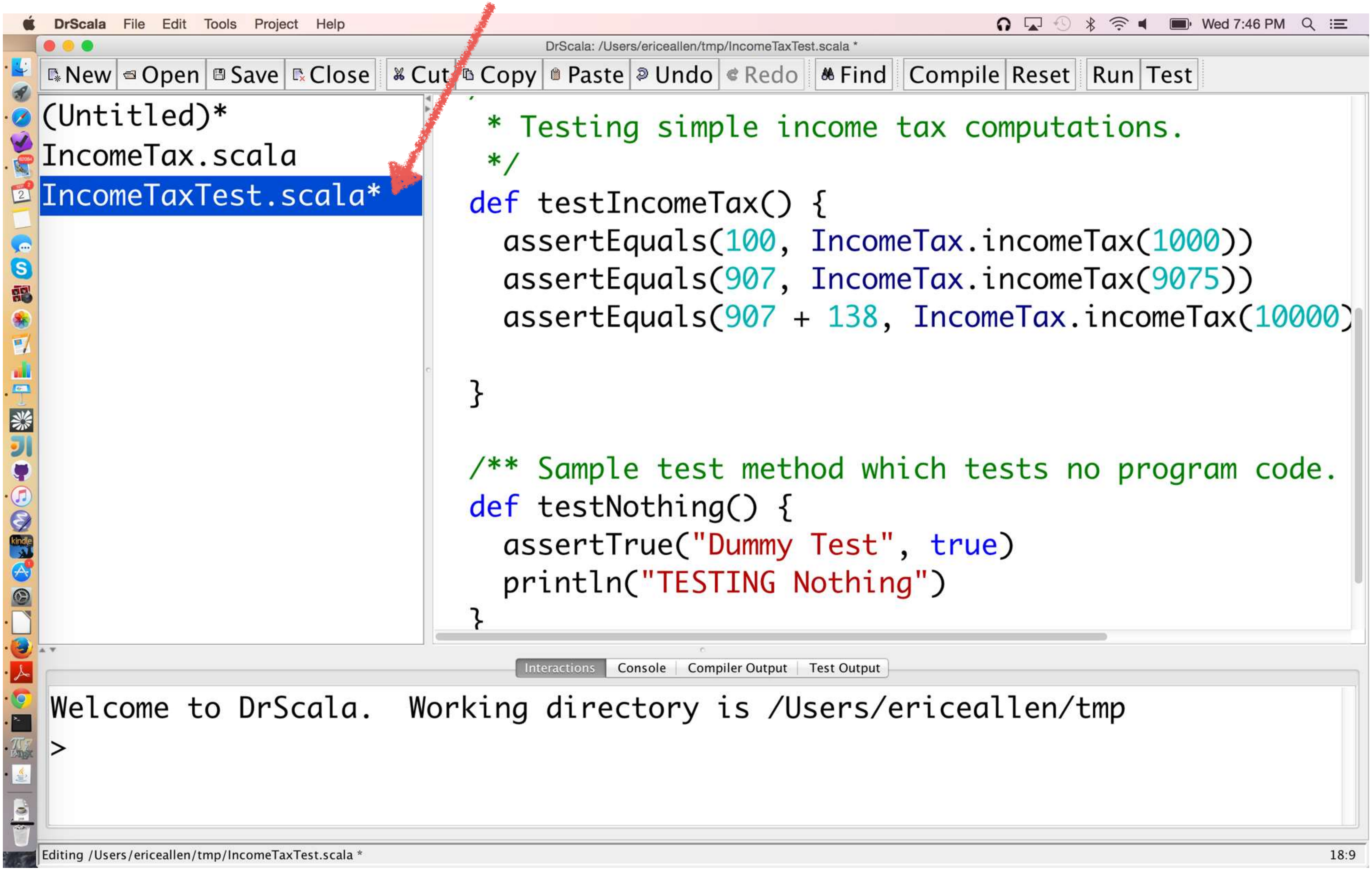


```
// Brackets are in tenths of percentage points,
// thus percentage divisor must be 1000
val divisor = 1000

/**
 * 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)
```

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

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

Files that have not been saved include an asterisk

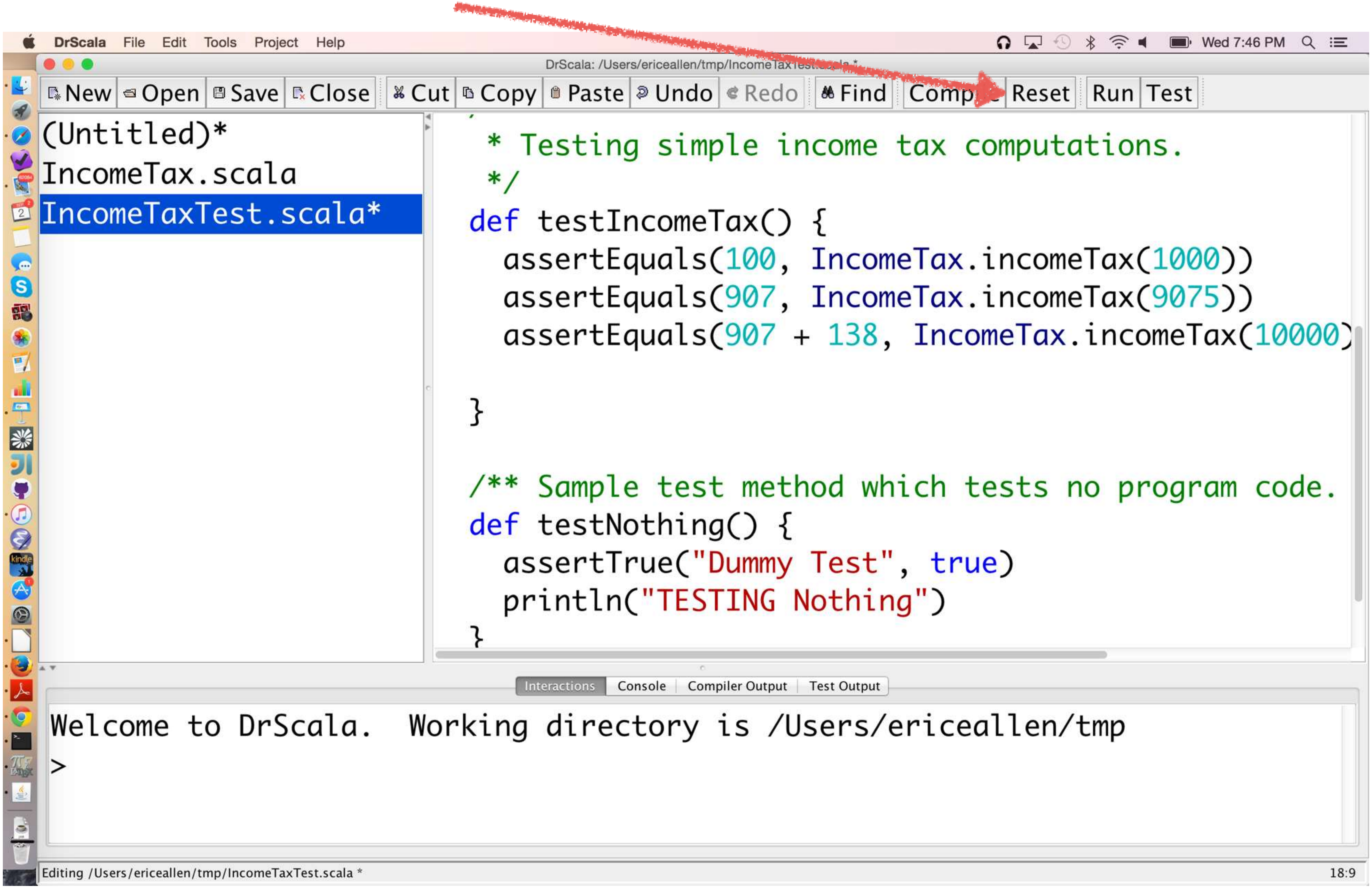

DrScala File Edit Tools Project Help
Wed 7:46 PM
DrScala: /Users/ericeallen/tmp/IncomeTaxTest.scala *
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test
(Untitled)*
IncomeTax.scala
IncomeTaxTest.scala*
Interactions Console Compiler Output Test Output

```
  * 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")
}
```

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

Editing /Users/ericeallen/tmp/IncomeTaxTest.scala * 18:9

Reset resets the Interactions session

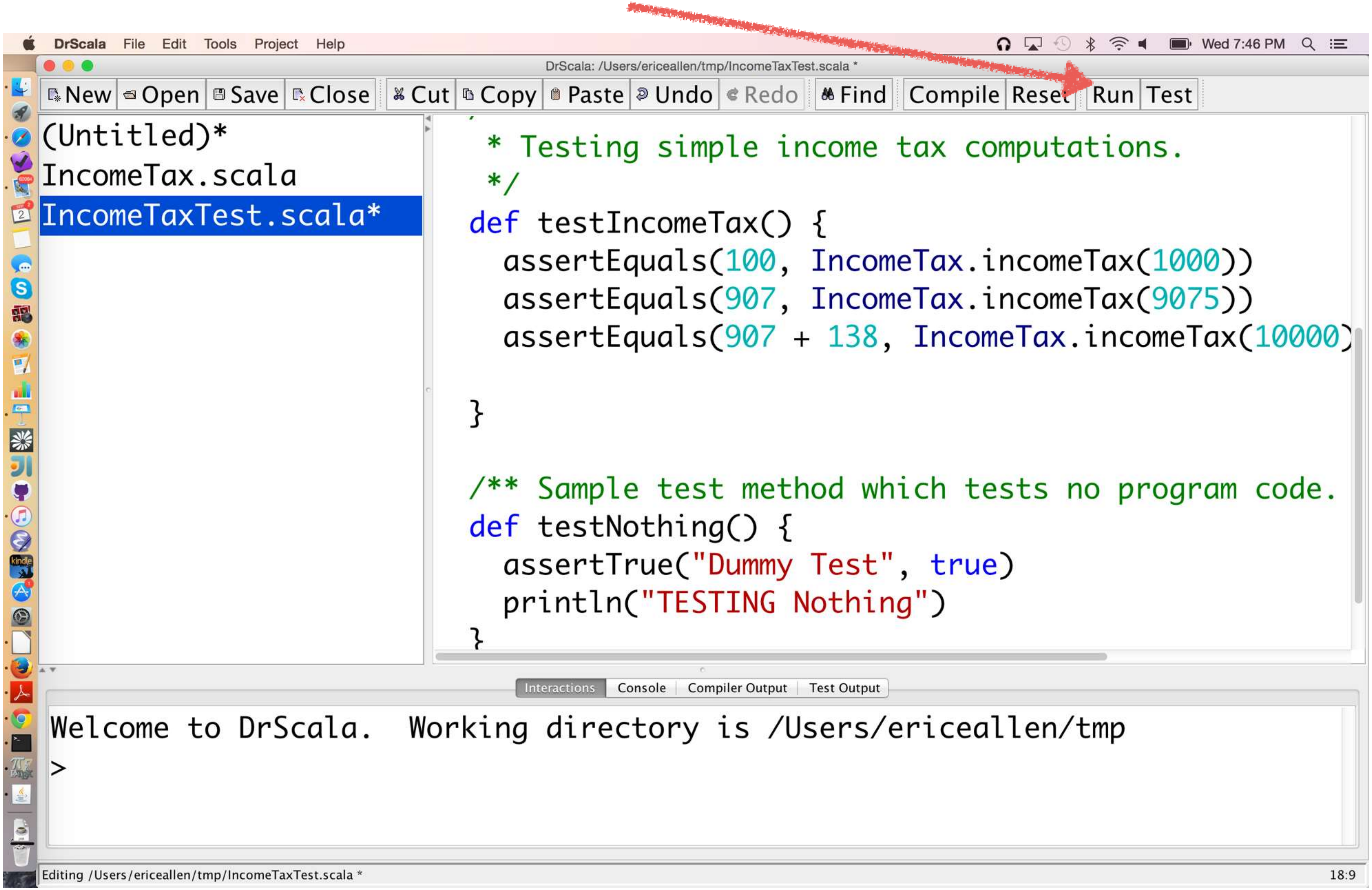


```
  * Testing simple income tax computations.
  */
def testIncomeTax() {
  assertEquals(100, IncomeTax.incomeTax(1000))
  assertEquals(907, IncomeTax.incomeTax(9075))
  assertEquals(907 + 138, IncomeTax.incomeTax(10000)

}

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

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

Editing /Users/ericeallen/tmp/IncomeTaxTest.scala *

Run executes Definitions



```
  * 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")
}
```

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

Editing /Users/ericeallen/tmp/IncomeTaxTest.scala *

18:9