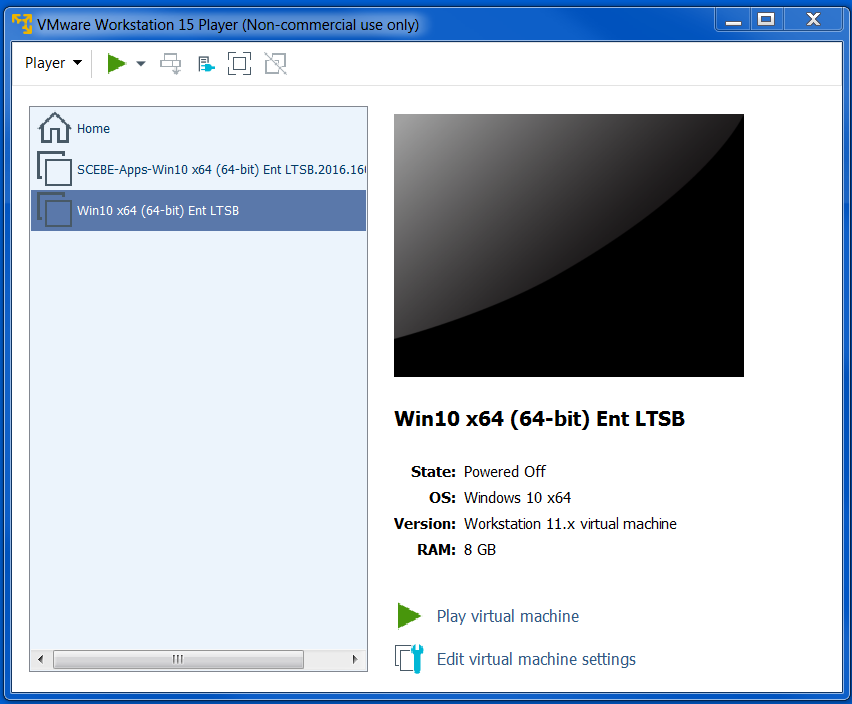
## LAB 1: WORKING WITH SCALA

In this lab you will practice working with Scala from the command line and using an IDE, and practice using some basic Scala syntax. In the university labs Scala is located in the Windows 10 VMware image.

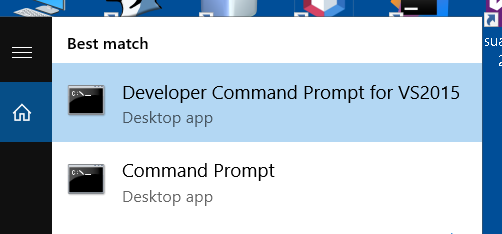


### Task 1. Working with the REPL

In order to use the Scala REPL you need to have a Java JDK and Scala installed and your OS environment correctly configured. The environment variable JAVA\_HOME should be set to the root directory of the JDK, the PATH environment should include the bin directory of your Scala installation and SCALA\_HOME should be set to the root directory of your Scala installation.

Scala should be configured already in a VM on the lab PCs. You can find installation instructions at <http://www.scala-lang.org/download/install.html> if you need to install on your own computer. Scala runs on Windows, MacOS and Linux – the lab PCs run Windows.

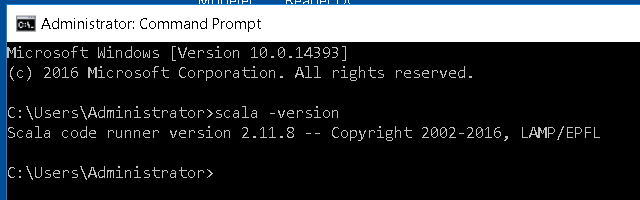
1. Open a system command prompt.



Enter the following command:

scala –version

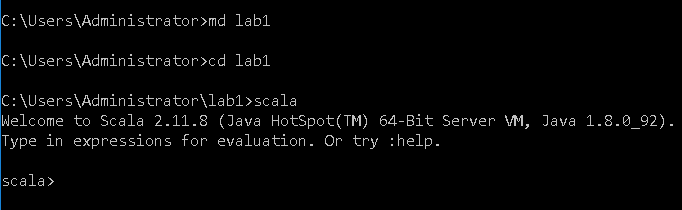
You should see a message confirming the version of Scala that is installed, e.g.



1. Create a folder where you want to store your working files for this lab and navigate to this in the command window. Enter the command:

scala

This should start the Scala REPL. When it is ready you should see a message and a scala> prompt, e.g.



(Note that when you are finished with the REPL and want to quit to the OS command prompt you enter the command :q)

1. Enter a simple variable declaration statement, e.g.

val x = 10

The result of evaluating this should be shown:

x: Int = 10

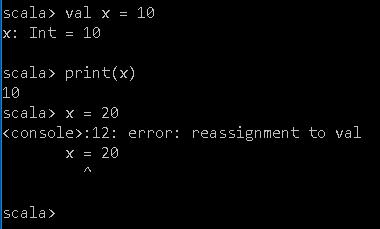
1. The REPL shows the result of each expression, but you can also explicitly print to the REPL output. Enter the following:

print(x)

1. Try changing the value of x by entering:

x = 20

What happens? Why? How can you make it possible to change the value of a variable?



1. Declare another variable y with the value 5. Enter an expression that adds x and y. Enter a statement that prints the result of adding x and y.
2. Operators in Scala are actually methods. Enter the following expression that adds x and y – the result should be the same as step 6:

x.+(y)

1. Enter the following function definition. This function calculates the nth power of a number, e.g. 24 = 16. Note that the REPL allows you to enter multiline statements like this and recognises when the statement is completed:

def power(value:Int, power:Int):Int = {

var result = 1

for(i <- 0 to power-1) {

note that this is not a very “functional” function – you will see why later in the module

result = result \* value

}

result

}

Pay careful attention to the syntax, if you don’t get it right you will have to start again. When you have entered the function definition correctly you should see the result – a function definition is evaluated to a description of the signature of the function:

power: (value: Int, power: Int)Int

1. Enter an expression that calls the power function, e.g.

power(2,4)

Did you get the correct result? Try some more calls to power, with different values. Do you always get the result you expect?

1. You can define modules (classes/singleton objects) in the REPL. Enter the following definition for a singleton object

object HelloWorld{

def main(args: Array[String]) {

println("Hello " + args(0))

}

}

Call the main method of the object:

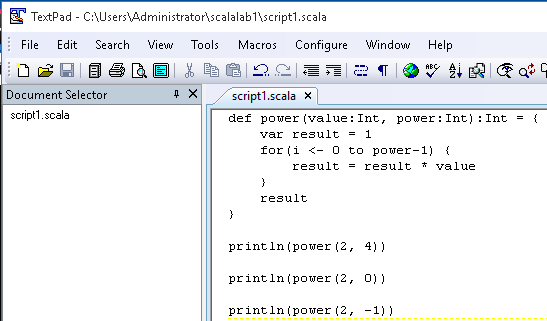
HelloWorld.main(Array("Jim"))

1. Exit the REPL using :q to return to the OS command prompt. Keep the command window open.

### Task2. Running scripts and compiling classes

The REPL is useful for experimenting interactively with Scala. However, it can be more convenient sometimes to write your Scala code in a file in a text editor, and run it as a script. That way, if you have made an error in your code, you can just edit it and run the script again.

1. Use a text editor (e.g. Notepad or TextPad) to create a file *script1.scala* in your working folder.
2. In the editor, enter the definition of the power method from Task 1, and some println statements which print the result of calling the function with various values. Save the script file.



1. At the command prompt, run the script with the command:

scala script1.scala

You should see the output from the print statements in your script. Note this will not work if your command prompt current folder is not the one where your script file was saved.

1. Create a new file *Classes.scala* in your working folder and use your text editor to enter the following in that file:

class Point(var x: Int, var y: Int) {

def move(dx: Int, dy: Int): Unit = {

x = x + dx

y = y + dy

}

override def toString: String =

"(" + x + ", " + y + ")"

}

object Classes extends App {

val pt = new Point(1, 2)

println(pt)

pt.move(10, 10)

println(pt)

}

This defines a class called Point that represents a point in 2D space, and a singleton object called Classes with a main method as an entry point to run this as a program. Note that here, the object extends App, which means you don’t have to put code inside a main method.

1. At the command prompt, compile this code with the following command:

scalac Classes.scala

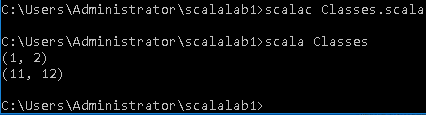
If there are no errors you should see no message – no news is good news!

Look at the contents of your working folder. You should see files *Classes.class* and Point.class. These are Java class files that have been create by the Scala compiler – Scala is compiled to Java bytecode.

1. Run this code from the command prompt with the command:

scala Classes

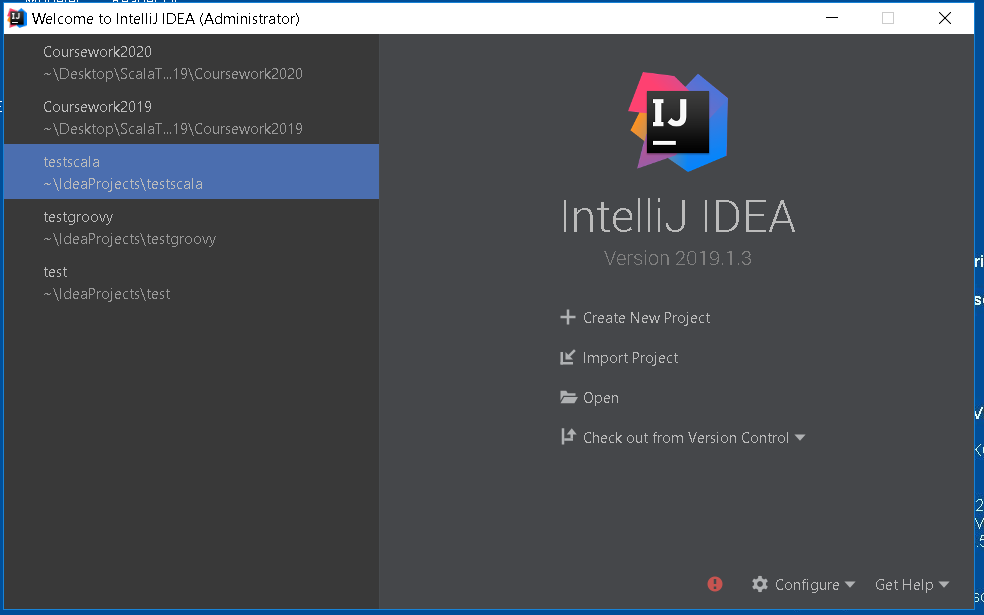
Do you see the output you expect? Note that here you are running compiled code, rather than a script, and you need to have code with complete classes rather than fragments of code as you can have in a script.



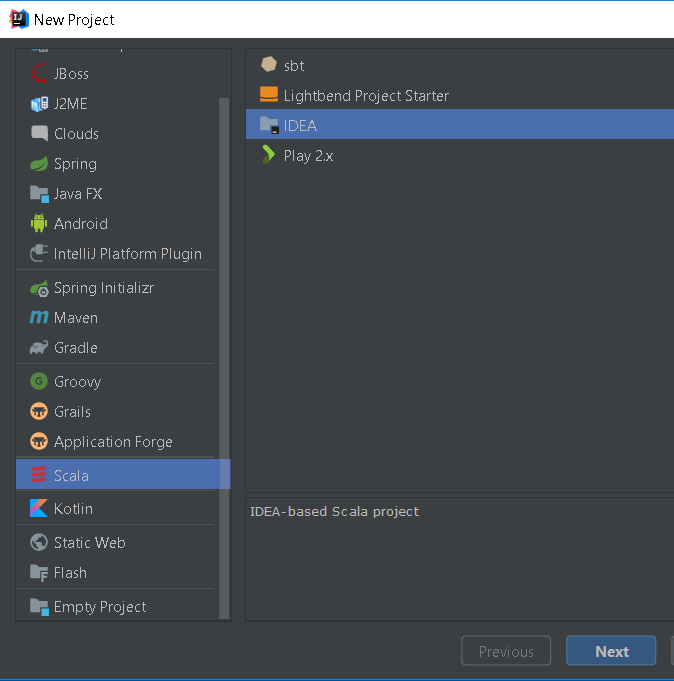
### Task 3. Working with an IDE

Some developers like to work with the command prompt, and it is possible to do so even with large projects, with the help of the *sbt* build tool, which can be used in this way. Others prefer to use an IDE, which provides many features to help developer productivity. Here, you will look at the IntelliJ IDE from JetBrains. In order to use the IntelliJ for Scala you need to have the Scala plugin installed in the IDE and configured correctly. IntelliJ should be configured already in the VM on the lab PCs.

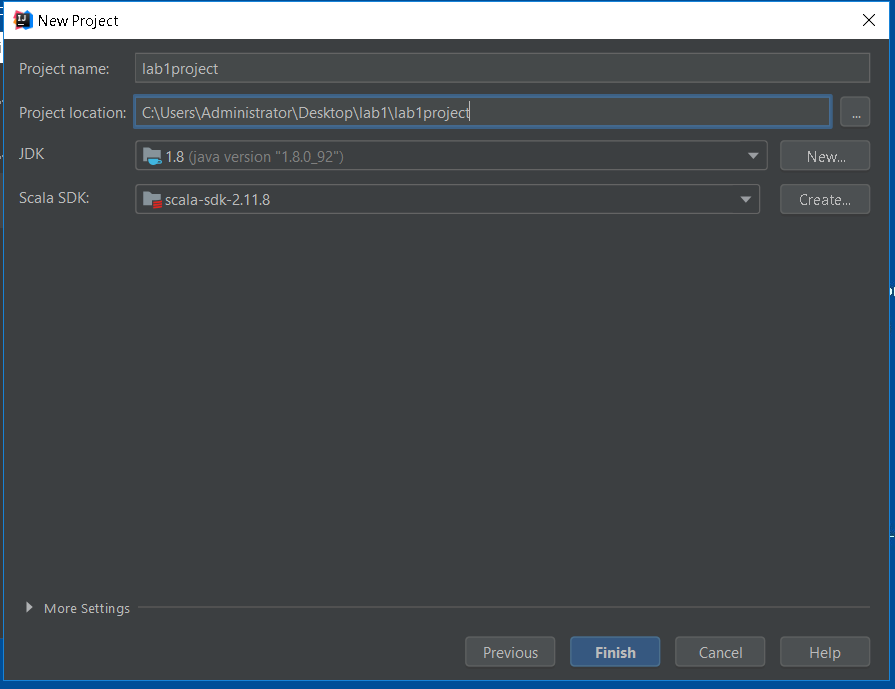
1. Start IntelliJ. You should see a Welcome window which gives you the option to create a new project. Click Create New Project.



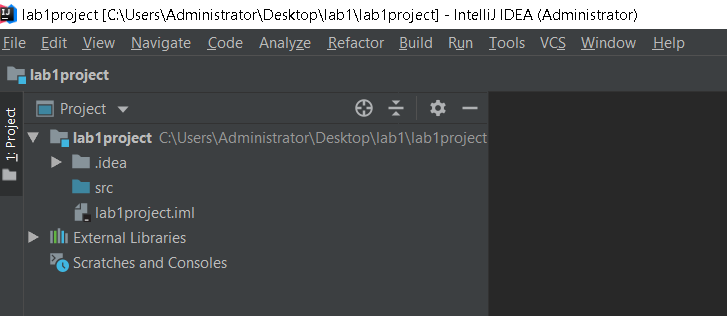
1. In the New Project window select Scala in the list on the left, and Scala in the main pane, and click Next.



1. In the next step, navigate to your working folder and add \lab1project to the path name in the Location box. For example, if your folder is c:\lab1 your location should be *c:\lab1\lab1project*. The project name should then be set automatically to *lab1project*. The Project SDK and Scala SDK boxes should be populated already – if not you will have to select the root folder of either or both of the JDK and Scala installation on your machine. Click Finish. Click OK if you get a prompt to create a new directory.



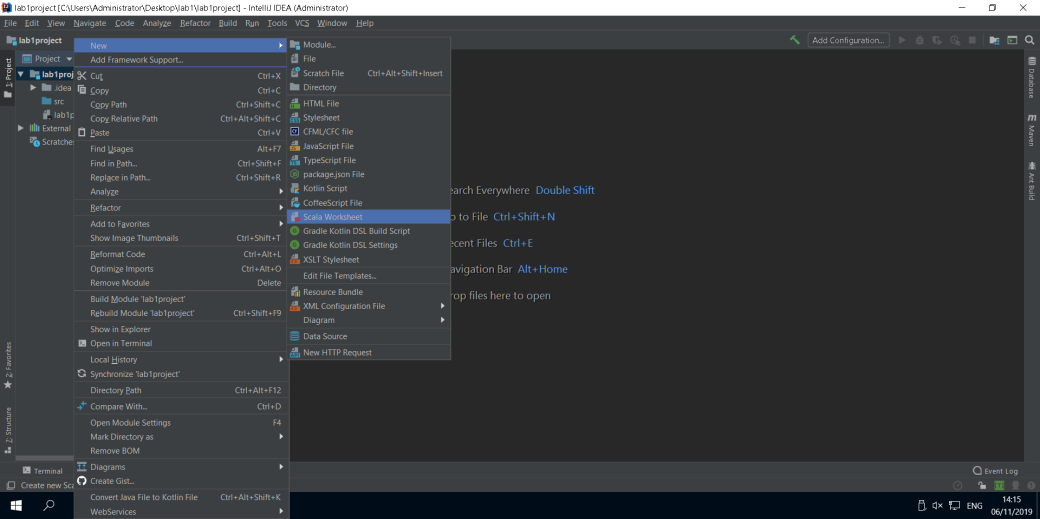
The new project should be created, and the project window in IntelliJ should look like this



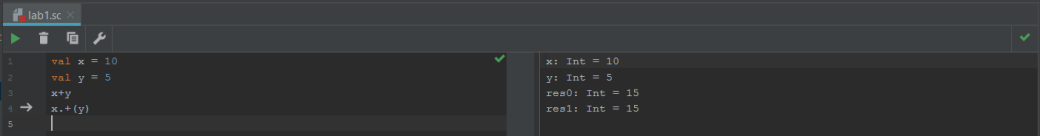
### Scala worksheet

The IDE provides an alternative way to work interactively with code, Scala worksheets. A worksheet allows you to enter Scala expressions and have them evaluated immediately.

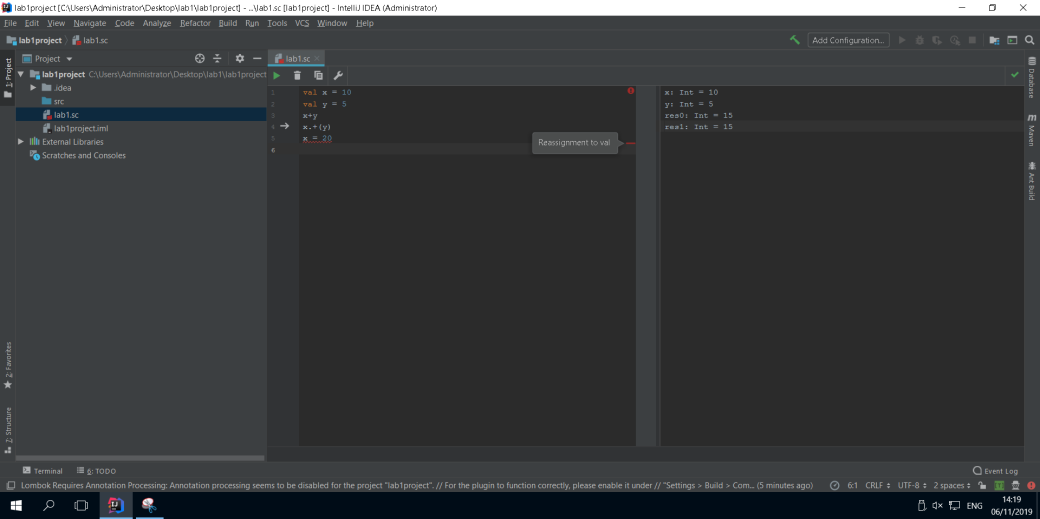
1. Right-click on the project root in the project window and select New > Scala Worksheet from the pop-up menus. Enter the name *lab1* for the worksheet. It will appear in the project window as *lab1.sc* (worksheets have the extension .*sc*) and open in the editor.



1. Enter the same simple variable declarations as you did in the REPL in task 1. When you enter an expression it should be evaluated (almost) immediately, and the result display in a separate pane alongside the code. This may take a few moments the first time you enter an expression, and you may have to click the green Evaluate Worksheet button above the code to start with - make sure Interactive mode is checked, and evaluation should happen automatically after that.



If the code in the worksheet contains no errors then you should see green ticks in the code pane and above the results pane. If there are errors, these will be indicated. You can go back and edit the code in the worksheet and it will then be evaluated again.



1. Repeat the remaining exercises that you did in the REPL, this time in the worksheet. When you finish, the worksheet will contain all the code you entered, with all the results shown. This is like a REPL session that you can go back into and re-run and change as you like without having to re-enter code.

* Enter the power function, and a call to this function
* Enter the HelloWorld object, and a call to its main method. In this case, you should see both the output from the println statement and the return value of the method – what is the return value and type? What does this mean?

### Scala classes

The Scala worksheet is useful for experimenting and learning, but when you are building an application you will instead create Scala classes. These normally go in the project’s *src* folder. In larger projects they will be organised into packages in the same way as you do with Java.

1. Right-click on the src folder in the project window and select New > Scala class from the menu. Enter the name Point for the class. It will appear in the project window as Point and open in a new editor tab. Note the icon in the project window which indicates that this is a Scala class.
2. Replace the code in the editor with the code for the Point class that you created in task 2.
3. Create another class in the *src* folder, with the name Classes. In the Create New Scala class dialog where you enter the name, select Object from the drop-down box instead of Class. Note the icon in the project window which indicates that this is a Scala class.
4. Replace the code in the editor with the code for the Classes singleton object that you created in task 2.
5. The object Classes contains the main method, so you can run it. Note that the main method uses the Point class that you created separately. Right-click on Classes in the project window and select Run ‘Classes’ from the menu. Your code will be compiled and run, and the output will be shown in the output window which is normally below the editor.
6. Running Classes once like this creates a run configuration. Run configurations can be selected, edited and run and debugged from the controls above the editor.



### Task 4. Practicing Scala

You have now tried a number of tools of ways of working with Scala. In this task you will now practice writing some Scala expressions yourself. You can use whichever of these tools you prefer to attempt the following exercises. If you choose to use a Scala worksheet you can continue with the same worksheet as in task 3, or start a new one. You should use your lecture notes and references listed under ‘Further reading’ to help you.

1. Write and test a function called *square* that takes a parameter of type Double and returns a Double that is the square of the parameter
2. Write and test a function called *distance* that takes parameters x1,y1,x2 and y2, all of type Double as parameters. These represent the coordinates of two points in 2D space (x1,y1) and (x2,y2). Your function should return the distance between these points, using the distance formula:

For example, distance(0,0,1,1) should return 1.414 (approximately)

You can make use of your square function in the distance function. You will also need to use Scala’s *sqrt* function, which is in the *scala.math* package, so you will need the following statement:

import scala.math.sqrt

1. Modify your distance function to use default and named parameters so that it can be called with two parameters only, and the distance from (0,0) to that point will be calculated, e.g.

distance(x2=1,y2=1)

1. Write an expression to create an array of integers containing the values 4,7,3,2,6
2. Write a function called *max* that takes an array of integers as a parameter and returns the highest value in the array. Use a for expression and an if expression in your function so that you can practice using these. Note that there are more “functional” ways of doing this computation, which you will see later on in the module.
3. Create a new class called *Circle* with two constructor parameters:

* center, of type Point (you will need the Point class you created previously)
* radius, of type Integer

Add a method called *area* which returns the area of the circle (πr2) as a Double

Add a method *toString*, similar to the one in Point, that returns the details of the Circle as a string, for example *(10,10), radius = 5*. Use the Point class as an example to help you.

1. Test your Circle class by adding the following code to the Classes object you created previously and run it:

val circ = new Circle(pt,5)  
println(circ)  
println(circ.area)  
println(circ.radius)  
circ.radius = 10  
println(circ)

Note that when you wite circ.radius you are using a getter or setter method created by the compiler – the notation for this is similar to properties in C# rather than getter/setter methods in Java.

What happens if you use each of the following for the radius parameter in your Circle class? Why?

* var
* val
* neither var nor val