Scala Tutorial

Axel Jacquin

January 16, 2017

The objective of this exercise is to initiate you to the Scala language which is the most common language for Spark.

This introduction to Scala language is obviously not an exhaustive course but will only serve as a toolbox for the upcoming TPs.

This introduction is freely inspired by `https://www.tutorialspoint.com/scala`

1 Overview

Scala, short for Scalable Language, is a hybrid functional programming language. It was created by Martin Odersky. Scala smoothly integrates the features of object-oriented and functional languages. Scala is compiled to run on the Java Virtual Machine. Many existing companies, who depend on Java for business critical applications, are turning to Scala to boost their development productivity, applications scalability and overall reliability.

1.1 Scala is object-oriented

Scala is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes which will be explained in subsequent chapters.

1.2 Scala is functional

Scala is also a functional language in the sense that every function is a value and every value is an object so ultimately every function is an object.

1.3 Scala runs on the JVM

Scala is compiled into Java Byte Code which is executed by the Java Virtual Machine (JVM). This means that Scala and Java have a common runtime platform. You can easily move from Java to Scala.

The Scala compiler compiles your Scala code into Java Byte Code, which can then be executed by the 'scala' command. The 'scala' command is similar to the java command, in that it executes your compiled Scala code.

Scala can execute Java code. Scala enables you to use all the classes of the Java SDK and also your own custom Java classes, or your favorite Java open source projects.

2 Exploring Scala Shell and Scala Basics

For this section, we are going to use scala shell.

2.1 Environment setup

Install VMWare WorkStation Player to use the new VM.

Connect on the VM called Cloudera-Training-SparkDev with training/training credentials and use the spark-shell as a scala-shell.
1. Open a new Scala shell typing spark-shell in a terminal Window

2.2 Basic operations

Let's start with some basic arithmetic operation. Just like most languages, simple arithmetic operations are part of the language.

2. Enter

```
scala> 1+2
```

You should see a new line like:

```
res0: Int = 3
```

where :

- res0 is the name of the variable where Scala put the output in. You can now reuse the output of 1+2 through the variable named res0.
- Int is the type of the output.
- 3 is the value of the output.

2.3 Data types

3. What is the type of the following operations's outputs :

- 1*3
- 5/2
- 5.3 - 4.3
- 'e'
- "Hello World"
- true

- 1 == 1
- 1 == 2
- 1 != 2
- "My" + " name" + " is" + " Axel"
- println(res0)
- "My age is " + 23

2.4 Variables

2.4.1 Mutable variables

In scala, variables are declared using the **var** keyword :

```
scala> var myAge: Int = 23
```

- myAge is the name of my variable
- Int is the type of my variable
- 23 is the value of my variable

In many cases, the type information can be omitted, thanks to Scala's type inference:

```
scala> var myAge = 23
```

You can update a variable value like :

```
scala> myAge = 24
```

Like in Java, in Scala, first letter of a variable's name is always lower case.

4. Create any String variable you want. Then print the value of this variable using println.

5. Update the variable you have just created then print the new value.

2.4.2 Final variables

Final variables are declared using the val keyword (a final variable is a variable that can't be reassigned)

```
scala> val myName = "Axel"
```

Prefer using '**val**' over '**var**' (and immutable objects over mutable ones). There are many benefits that are out of the scope of this small tour.

6. Create any final variable you want. Then print the value of this variable using println.

7. Try to update the value of your variable.

3 Run a Scala Application through IntelliJ

3.1 Environment setup

For this part, you can use Maven and IntelliJ that are already installed on the VM

3.1.1 Create a new Scala Application

9. Create a new Scala project.
The easiest way to create new projects is using an "archetype". An archetype is a general skeleton structure, or template for a project.
You run the archetype plugin like this:

```
$ mvn archetype:generate
```

If this is your first time, you'll notice that Maven is downloading many jar files. Maven resolves dependencies and downloads them as needed (and only once). Right now, Maven is downloading its core plugins.
Afterwards, it should give you a list of archetypes (in the hundreds).
Choose **net.alchim31.maven:scala-archetype-simple** (You can type "scala" (or something else) to filter the results.) Then choose 1.5 version.
Type 'dsti' for groupeId
Type 'scala-tutorial' for artifactId
Type enter for version and package.

Then Maven will create a new Maven project under scala-tutorial directory

10. cd to this directory

```
$ cd scala-tutorial
```

then run

```
$ mvn package
```

You'll see Maven downloading dependencies including the Scala library.

11. Open your project with IntelliJ doing
File > Importing Project
and importing from external model Maven then pointing to the freshly created project.

IntelliJ should open the new project containing these following :

- .idea folder which contains some IntelliJ configuration files
- src folder which contains the source code of your application (.java, .scala files)
- a file pom.xml which will contains all dependencies of your application

This archetype provides you a App object containing a main function.
This main function is the entry point of your program.

12. Run the App class to check if everything is correct. You should see :
Hello World!
concat arguments =

3.2 Functions

In Scala functions have the following form :

```
def functionName ([list of parameters]) : [return type] = {
   function body
   return [expr]
}
```

For instance,

```
object App {
def addition( a:Int, b:Int ) : Int = {
      val sum = a + b
      return sum
}
}
```

Like in Java, in Scala, first letter of a function's name is always lower case.

Functions are called from the main function like :

```
object App {
    def main(args : Array[String]) {
         val mysSum = addition(3+2)
    }
}
```

13. Create a new function *presentation* in App (before the main function) that takes a name as parameter. *presentation* must return a sentence that give the name entered as parameters like "My name is .."

14. Call *presentation* under the main function with your name.

15. Change the function *presentation* in order to take a name and an age as parameters. The output must be the sentence : "My name is ... and my age is ..."

16. Call *presentation* under the main function with your name and your age.

A function, that does not return anything can return a Unit that is equivalent to void in Java and indicates that function does not return anything. The functions which do not return anything in Scala, are called procedures.

17. Transform *presentation* function into a procedure.

You can return multiple variable using tuples :

```
def swap(x:String, y:String) = (y, x)  // Note that the brackets are optionals
                                       //in one-line functions
(a,b) = swap(x:String,y:String)
println (a,b)
```

18. Transform *presentation* in order to return a tuple where the first element is the name and the second is the age.

3.3 Classes

A class is a blueprint for objects. Once you define a class, you can create objects from the class blueprint with the keyword new. Through the object you can use all functionalities of the defined class.

Classes are defined by attributes and functions. An object can be created according to the class attributes and functions of the class can be applied to the object

Classes are defined according to the following form:

```
class ClassName ([list of attributes]) {
   functions definitions
}
```

Like in Java, in Scala, first letter of a class's name is always upper case.

Then object are created like :

```
val myObject = new ClassName (list of attributes)
```

For instance :

```
class Movie (name: String, year:Int, director:String) {
   def presentation(): Unit{
         println("The movie is" + name
     +"and has been realised in " + year+
     "by " + director )
   }
}
val myFilm = new Film("Inception",2010,"Nolan")
myFilm.presentation()
```

19. Create a new class Person defined by two attributes : a name and an age.

20. Change the *presentation* function in order to take a Person as a parameter. Test it !

21. Now, move the *presentation* function into the class Person definition. *presentation* is now part of the class definition, you can apply the *presentation* function to a Person object.
Modify the main function to work with this new definition

22. In the main function, create a new Person and print it using println()

The output of println should be unreadeable. Scala use a toString default function to print object. We are going to redefine this default functions for Person objects.

23. Add the following to the Person class:

```
override def toString: String = presentation()
```

Now, when you will print Person, scala is going to use *presentation* function. Test it.

3.4 Array

Arrays are constructed simply using Array(element1,element2, ..)

Arrays are mutable (can't change it's size once created, but can modify it's elements)

Array elements can be of any type, but the Array's final type will be the lowest common denominator. (for instance Array(1 , 1.2) will be an Array[Double])

You access items using (index) (not [index])

24. Create an Array[Int] named *ages* of 5 elements.

25. Print the array ages thanks to println.
The output should not be readable. To print an array, the most common way is to use .mkString(',') on the array. Test it.

26. Create an Array[String] named *names* of 5 elements

3.5 Loops

3.5.1 For Loops

For loops in Scala are powerful and deserve a separate discussion. But they can be used to mimic a c / Java like for loop as well. The syntax is :

```
for ( i <- 0 until 10) {
  println(i)
}
```

27. Create an empty array of Person named persons of 5 elements

```
val persons = new Array[Student](5)
```

28. Populate persons thanks to ages, names and a for loop.

29. Use foreach function on persons in order to print each Person.

3.5.2 While Loops

The syntax of a while loop is :

```
var i
while ( i < 10) {
  println(i)
  i+=1
}
```

30. In order to practice, calculate 10! using a while loop.

3.6 Conditions

Syntax:

```
if ( boolean ) {
 instructions
} else {
 instructions
}
```

31. Create a new function *isTooYoung*() in the Person class returning *true* if the Person is under 18 and *false* if not. Test it.