# Introduction to Scala

October 9, 2017

## 1 Learning Outcomes

- Start Scala REPL in Scala application development
- Execute and observe Scala programs Scala application development
- Comprehend all the Scala languages features and the program semantics when reviewing Scala source codes
- Develop data transformation scripts using Scala

## 2 Scala Features

1. Scala is an o_________ oriented and f_________ language.
2. Scala is a _________ typed language.

## 3 First Scala Program - Hello World

1. Check out the source codes.
   (a) go to Github and download the `scala.zip` from

   ```
   $ cd learning-scala/codes
   ```

   (b) Examine the script `Script.scala` in `helloworld`.

   (c) Execute the script with the following

   ```
   $ scala Script.scala
   ```

   (d) Examine the code `Main.scala` in `helloworld`.

   (e) Compile the code

   ```
   $ scalac Main.scala
   ```

   (f) Execute the compiled code

   ```
   $ scala Main
   ```

## 4 Scala REPL

(a) Start a terminal in Linux or command line in Windows, type

```
$ scala
```

Note that the $ sign is the command prompt, you should not include it as part of the command.

(b) Exit Scala REPL by typing

```
scala> :quit
```

Note that the `scala>` sign is the Scala REPL prompt, you should not include it as part of the command.

## 5 Variables, Values and Assignment Statement

In a Scala REPL

(a) Declare a variable with name "first_name" and assign a string value as "robin".

(b) Declare a value with name "last_name" and assign a string value as "Williams".

(c) Update the variable "first_name" to a new string value "Robin"

(d) If you were to update the value "last_name" to a new string "Hood", what will happen?

## 6 Print Statement

In a Scala REPL

(a) Print the variable "first_name" and value "last_name" individually

(b) Use template, print the following

```
Robin William (1951 - 2014)
```

You need to make use of the variable "first_name" and value "last_name", and put 1951 and 2014 into the two additional variables. For instance, assuming you have defined "first_name" and "last_name".

```
val bYear = 1951
val dYear = 2014
println(s"$first_name $last_name ($bYear - $dYear)")
```

## 7 If-else

(a) Type the following code snippet in the Scala REPL and observe the output.

```
val i = 1
if (i / 2 >= 0.5) {
  println(s" ${i} / 2 is greater than or equal to  0.5") }
else {
   println(s"${i} / 2 is less than 0.5")
}
```

## 8 List and List operation

(a) Declare a list of integer `l1` with integers 1, 2, 3 and 4.

(b) Declare a second list `l2` whose elements are the odd values of `l1` incremented by 1.

(c) Find out the head and the tail of `l2`.

(d) Reverse `l2`.

(e) Concatenate `l1` and `l2`

(f) Compute the sum of `l1`

## 9 Object Oriented Programming

(a) In the terminal, change the working directory to `/git/learning-scala/codes/oop`.

(b) Examine the code `OOP.scala`, are you able to identify the class constructors, member fields, member methods? Are you able to identify the class inheritence?

```
class Person(n:String,i:String) {
        private val name:String = n
        private val id:String   = i
        def getName():String = name
        def getId():String = id
}

trait NightOwl {
        def stayUpLate():Unit
}

class Student(n:String, i:String, g:Double) extends Person(n,i) with NightOwl {
        private var gpa = g
        def getGPA() = gpa
        def setGPA(g:Double) =
        {
                gpa = g
        }
        override def stayUpLate():Unit =
        {
                println("woohoo")
        }
}

class Staff(n:String, i:String, sal:Double) extends Person(n,i) {
        private var salary = sal
        def getSalary() = salary
        def setSalary(sal:Double) =
        {
                salary = sal
        }
}
```

(c) Load the class in the Scala REPL and test it out

```
scala> :load OOP.scala
Loading OOP.scala...
defined class Person
defined trait NightOwl
defined class Student
defined class Staff

scala> val tom = new Student("Tom", "X1235", 4.0)
tom: Student = Student@601c1dfc

scala> val jerry = new Staff("Jerry", "T0001", 500000.0)
jerry: Staff = Staff@650fbe32

scala> tom.stayUpLate
woohoo
```

## 10 Functional Programming in Scala

(a) In the terminal, change the working directory to `/git/learning-scala/codes/fp`.

(b) Examine the code `Exp.scala`, are you able to identify the sealed trait, the case class, and the pattern matching?

```
sealed trait Exp
case class Val(v:Int) extends Exp
case class Plus(e1:Exp, e2:Exp) extends Exp
```

```
def simp(e:Exp):Exp = e match
{
        case Val(v) => e
        case Plus(Val(0), e2) => e2
        case Plus(e1,e2) => Plus(simp(e1), simp(e2))
}
```

(c) Run it with Scala REPL

```
$ scala
scala> :load Exp.scala
scala> val e = Plus(Val(0), Plus(Val(1), Val(2)))
e: Plus = Plus(Val(0),Plus(Val(1),Val(2)))

scala> simp(e)
res0: Exp = Plus(Val(1),Val(2))
```

(d) Note that $x - 0 = x$, $x * 1 = x$, $x/1 = x$ for all $x$, can we extend our `Exp` data type and the simplification `simp` to handle minus, multiplication, and division?

(e) Note that the simplification is not througout, e.g.

```
scala> val e2 = Plus(Val(0), Plus(Val(0),Val(2)))
e2: Plus = Plus(Val(0),Plus(Val(0),Val(2)))

scala> simp(e2)
res1: Exp = Plus(Val(0),Val(2))
```

How can we fix it?