ADVANCED FUNCTIONAL THINKING

DR SURBHI SARASWAT
SCHOOL OF COMPUTER SCIENCE

FIRST SCALA PROGRAM

We can execute a Scala program in two modes: one is interactive mode and another is script mode.

Interactive Mode

Open the command prompt and use the following command to open Scala.

```
\>scala
```

If Scala is installed in your system, the following output will be displayed –

```
Welcome to Scala version 2.9.0.1
Type in expressions to have them evaluated.
Type :help for more information.
```

Type the following text to the right of the Scala prompt and press the Enter key –

```
scala> println("Hello, Scala!");
```

It will produce the following result –

```
Hello, Scala!
```

FIRST SCALA PROGRAM

Script Mode

Use the following instructions to write a Scala program in script mode. Open notepad and add the following code into it.

```
object HelloWorld {
   /* This is my first java program.
   * This will print 'Hello World' as the output
   */
   def main(args: Array[String]) {
      println("Hello, world!") // prints Hello World
   }
}
```

Save the file as – **HelloWorld.scala**

Use the following command to compile and execute your Scala program.

```
\> scalac HelloWorld.scala
\> scala HelloWorld
```

BASIC SYNTAX

The following are the basic syntaxes and coding conventions in Scala programming.

- **Case Sensitivity** – Scala is case-sensitive, which means identifier **Hello** and **hello** would have different meaning in Scala.

- **Class Names** – For all class names, the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  Example – class MyFirstScalaClass.

- **Method Names** – All method names should start with a Lower Case letter. If multiple words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  Example – def myMethodName()

- **Program File Name** – Name of the program file should exactly match the object name. When saving the file you should save it using the object name (Remember Scala is case-sensitive) and append '**.scala**' to the end of the name. (If the file name and the object name do not match your program will not compile).
  Example – Assume 'HelloWorld' is the object name. Then the file should be saved as 'HelloWorld.scala'.

- **def main(args: Array[String])** – Scala program processing starts from the main() method which is a mandatory part of every Scala Program.

SCALA IDENTIFIERS

Identifiers in a programming language are the names given to a class, method, variable or object to identify them in the program.

Here is a sample Scala code

```scala
object myObject {
    def main(args: Array[String]) {
        var value1 = 54
        var value2 = 65

        println("The sum of two values is " + (value1 + value2))
    }
}
```

Output:

```
The sum of two values is 119
```

All identifiers in this program are, **myObject, main, args, value1, value2.**

SCALA IDENTIFIERS

The declaration of these identifiers is based on certain rules. And programs violating these rules will return a compiler-Time error on compilation.

Rules to declare an identifier

- You cannot use a Scala keyword as an identifier in Scala. For example, class is not a valid identifier.
- Reserved words cannot be used as identifiers in Scala. For example, $ cannot be an identifier.
- There is no upper limit to the number of characters used in the identifier. WelcomeToScalaProgrammingLanguage is also a valid identifier.
- An identifier cannot start with digits. For example, 2421Scala is not a valid identifier.
- Scala identifiers are case-sensitive. For example, isgreater and isGreater are different identifiers.

SCALA IDENTIFIERS

Scala supports four types of identifiers.

Alphanumeric Identifiers

An alphanumeric identifier starts with a letter or an underscore, which can be followed by further letters, digits, or underscores. The '$' character is a reserved keyword in Scala and should not be used in identifiers.

The following are legal alphanumeric identifiers –

age, salary, _value, __1_value

The following are illegal identifiers –

$salary, 123abc, -salary

Example:

```
object myObject {
    def main(args: Array[String]) {
        var _value1 = 54
        var value_2 = 65

        println("The sum of two values is " + (_value1 + value_2))
    }
}
```

SCALA IDENTIFIERS

Operator Identifiers

An operator identifier consists of one or more operator characters. Operator characters are printable ASCII characters such as +, :, ?, ~ or #.

The following are legal operator identifiers –

```
+ ++ ::: <?> :>
```

The Scala compiler will internally "mangle" operator identifiers to turn them into legal Java identifiers with embedded $ characters.

For instance, the identifier :-> would be represented internally as $colon$minus$greater.

Example:

```
object myObject {
    def main(args: Array[String]) {
        var _value1 = 54
        var value_2 = 65

        println("The result after using '+' identifier is " + (_value1 + value_2))
    }
}
```

Output:

```
The result after using '+' identifier is 119
```

SCALA IDENTIFIERS

Mixed Identifiers

A mixed identifier consists of an alphanumeric identifier, which is followed by an underscore and an operator identifier.

The following are legal mixed identifiers –

```
unary_+, myvar_=
```

Here, unary_+ used as a method name defines a unary + operator and myvar_= used as a method name defines an assignment operator (operator overloading).

Example:

```scala
object myObject {
    def main(args: Array[String]) {
        var _value1 = 54
        var value_2 = 65

        var val_+ = _value1 + value_2

        println("The sum is " + (val_+))
    }
}
```

Output:

```
The sum is 119
```

SCALA IDENTIFIERS

Literal Identifiers
A string literal used as an identifier in Scala is a literal identifier.

These are strings enclosed inside ticks ('...').

Example:

```
`scala`, `value`
```

Program:

```
object myObject {
    def main(args: Array[String]) {
        var `firstValue` = 54
        var `secondValue` = 65

        var `result` = `firstValue` + `secondValue`

        println("The result of sum is " + (`result`))
    }
}
```

Output:

```
The result of sum is 119
```

SCALA KEYWORDS

abstract	case	catch	class
def	do	else	extends
false	final	finally	for
forSome	if	implicit	import
lazy	match	new	Null
object	override	package	private
protected	return	sealed	super
this	throw	trait	Try
true	type	val	Var
while	with	yield	
-	:	=	=>
<-	<:	<%	>:
#	@		

SCALA VARIABLES

A variable is a name which is used to refer to memory location. You can create a mutable and immutable variable in scala.

Let's see how to declare variable.

Mutable Variable

You can create a mutable variable using the **var** keyword. It allows you to change the value after the declaration of a variable.

```
var data = 100
data = 101                // It works, No error.
```

In the above code, **var** is a keyword and data is a variable name. It contains an integer value 100. Scala is a type infers language so you don't need to specify data type explicitly. You can also mention the data type of variable explicitly as we have used in below.

Another example of variable

```
var data:Int = 100          // Here, we have mentioned Int followed by : (colon)
```

SCALA VARIABLES

A variable is a name which is used to refer to memory location. You can create a mutable and immutable variable in scala.

Let's see how to declare variable.

Immutable Variable

You can create an immutable variable using the **val** keyword.

```
val data = 100
data = 101 // Error: reassignment to val
```

The above code throws an error because we have changed the content of an immutable variable, which is not allowed. So if you want to change content then it is advisable to use var instead of **val**.

SCALA VARIABLES

Variable Type Inference:

When you assign an initial value to a variable, the Scala compiler can figure out the type of the variable based on the value assigned to it. This is called variable type inference.

Therefore, you could write these variable declarations like this:

```
var myVar = 10;

val myVal = "Hello, Scala!";
```

Here by default myVar will be Int type and myVal will become String type variable.

Multiple assignments:

Scala supports multiple assignments. If a code block or method returns a Tuple, the Tuple can be assigned to a val variable.

```
val (myVar1: Int, myVar2: String) = Pair(40, “Shoes")
```

And the type inference gets it right:

```
val (myVar1, myVar2) = Pair(40, “Shoes")
```

SCALA DATA TYPES

Data Type	Default Value	Size
Boolean	FALSE	True or false
Byte	0	8 bit signed value (-2^7 to 2^7-1)
Short	0	16 bit signed value(-2^{15} to 2^{15}-1)
Char	\u0000'	16 bit unsigned Unicode character(0 to 2^{16}-1)
Int	0	32 bit signed value(-2^{31} to 2^{31}-1)
Long	0L	64 bit signed value(-2^{63} to 2^{63}-1)
Float	0.0F	32 bit IEEE 754 single-precision float
Double	0.0D	64 bit IEEE 754 double-precision float
String	Null	A sequence of characters

COMMENTS

Scala supports single-line and multi-line comments very similar to Java.

Multi-line comments may be nested, but are required to be properly nested.

All characters available inside any comment are ignored by the Scala compiler.

```
object HelloWorld {
  /* This is my first java program.
   * This will print 'Hello World' as the output
   * This is an example of multi-line comments.
   */
  def main(args: Array[String]) {
    // Prints Hello World
    // This is also an example of single line comment.
    println("Hello, world!")
  }
}
```