Scala – Scalable Languages

Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way. Scala has been created by Martin Odersky and he released the first version in 2003. Scala smoothly integrates the features of object-oriented and functional languages. This tutorial explains the basics of Scala in a simple and reader-friendly way.

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 and traits which will be explained in subsequent chapters.

Classes are extended by **subclassing** and a flexible **mixin-based composition** mechanism as a clean replacement for multiple inheritance.

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.

Scala provides a lightweight syntax for defining **anonymous functions**, it supports **higher-order functions**, it allows functions to be **nested**, and supports **currying**. These concepts will be explained in subsequent chapters.

Scala is statically typed

Scala, unlike some of the other statically typed languages (C, Pascal, Rust, etc.), does not expect you to provide redundant type information. You don't have to specify a type in most cases, and you certainly don't have to repeat it.

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.

Scala can do Concurrent & Synchronize processing

Scala allows you to express general programming patterns in an effective way. It reduces the number of lines and helps the programmer to code in a type-safe way. It allows you to write codes in an immutable manner, which makes it easy to apply concurrency and parallelism (Synchronize).

Scala vs Java

Scala has a set of features that completely differ from Java. Some of these are −

* All types are objects
* Type inference
* Nested Functions
* Functions are objects
* Domain specific language (DSL) support
* Traits
* Closures
* Concurrency support inspired by Erlang

## Scala Web Frameworks

Scala is being used everywhere and importantly in enterprise web applications. You can check a few of the most popular Scala web frameworks −

* The lift framework
  + Lift is the most powerful, most secure web framework available today. There are [Seven Things](http://seventhings.liftweb.net/)that distinguish Lift from other web frameworks.
  + **Lift applications are:**
  + **Secure** -- Lift apps are resistant to common vulnerabilities including many of the OWASP Top 10
  + **Developer centric** -- Lift apps are fast to build, concise and easy to maintain
  + **Designer friendly** -- Lift apps can be developed in a totally designer friendly way
  + **Scalable** -- Lift apps are high performance and scale in the real world to handle insane traffic levels
  + **Modular** -- Lift apps can benefit from, easy to integrate, pre built modules
  + **Interactive like a desktop app** -- Lift's Comet support is unparalleled and Lift's ajax support is super-easy and very secure
* The Play framework

# Play Framework makes it easy to build web applications with Java & Scala.

Play is based on a lightweight, stateless, web-friendly architecture.

Built on Akka, Play provides predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications.

## Developer friendly.

Make your changes and simply hit refresh! All you need is a browser and a text editor.

* Hit refresh workflow
* Powerful console & build tools
* Type safety
* Built in testing tools
* IDE support for Eclipse and IntelliJ IDEA
* The Bowler Framework

# Bowler SBT Quickstart

The Bowler Quickstart is a Skeleton sbt project and Bowler app that gets you up and running with a minimal Bowler app that you can adapt in no time.

Scala Installation:

## Step 1: Verify Your Java Installation

Scala can be installed on any UNIX flavored or Windows based system. Before you start installing Scala on your machine, you must have Java 1.8 or greater installed on your computer.

|  |  |  |
| --- | --- | --- |
| **Platform** | **Command** | **Sample Output** |
| Windows | Open Command Console and type −  **\>java –version** | Java version "1.8.0\_31"  Java (TM) SE Run Time  Environment (build 1.8.0\_31-b31)  Java Hotspot (TM) 64-bit Server  VM (build 25.31-b07, mixed mode) |

## Step 2: Set Your Java Environment

Set the environment variable JAVA\_HOME to point to the base directory location where Java is installed on your machine. For example,

|  |  |
| --- | --- |
| **Sr.No** | **Platform & Description** |
| 1 | **Windows**  Set JAVA\_HOME to C:\ProgramFiles\java\jdk1.7.0\_60 |

Append the full path of Java compiler location to the System Path.

|  |  |
| --- | --- |
| **Sr.No** | **Platform & Description** |
| 1 | **Windows**  Append the String "C:\Program Files\Java\jdk1.7.0\_60\bin" to the end of the system variable PATH. |

## Step 3: Install Scala

You can download Scala from [http://www.scala-lang.org/downloads](http://www.scala-lang.org/download/). At the time of writing this tutorial, I downloaded ‘scala-2.11.5-installer.jar’. Make sure you have admin privilege to proceed. Now, execute the following command at the command prompt −

|  |  |  |
| --- | --- | --- |
| **Platform** | **Command & Output** | **Description** |
| Windows | \>java –jar scala-2.11.5-installer.jar\> | This command will display an installation wizard, which will guide you to install Scala on your windows machine. During installation, it will ask for license agreement, simply accept it and further it will ask a path where Scala will be installed. I selected default given path *“C:\Program Files\Scala”,* you can select a suitable path as per your convenience. |

Finally, open a new command prompt and type **Scala -version** and press Enter. You should see the following −

|  |  |  |
| --- | --- | --- |
| **Platform** | **Command** | **Output** |
| Windows | \>scala -version | Scala code runner version 2.11.5 -- Copyright 2002-2013, LAMP/EPFL |

If you have a good understanding on Java, then it will be very easy for you to learn Scala. The biggest syntactic difference between Scala and Java is that the ';' line end character is optional.

When we consider a Scala program, it can be defined as a collection of objects that communicate via invoking each other’s methods. Let us now briefly look into what do class, object, methods and instance variables mean.

* **Object** − Objects have states and behaviors. An object is an instance of a class. Example − A dog has states - color, name, breed as well as behaviors - wagging, barking, and eating.
* **Class**− A class can be defined as a template/blueprint that describes the behaviors/states that are related to the class.
* **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
* **Fields** − Each object has its unique set of instance variables, which are called fields. An object's state is created by the values assigned to these fields.
* **Closure** − A **closure** is a function, whose return value depends on the value of one or more variables declared outside this function.
* **Traits** − A trait encapsulates method and field definitions, which can then be reused by mixing them into classes. Traits are used to define object types by specifying the signature of the supported methods.

## 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!

## Script Mode

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

### Example

[Live Demo](http://tpcg.io/xmOtWe)

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**.

Open the command prompt window and go to the directory where the program file is saved. The ‘**scalac**’ command is used to compile the Scala program and it will generate a few class files in the current directory. One of them will be called **HelloWorld.class**. This is a bytecode which will run on Java Virtual Machine (JVM) using ‘**scala**’ command.

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

\> scalac HelloWorld.scala

\> scala HelloWorld

### Output

Hello, World!

## 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

All Scala components require names. Names used for objects, classes, variables and methods are called identifiers. A keyword cannot be used as an identifier and identifiers are case-sensitive. 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.

Following are **legal alphanumeric identifiers** −

age, salary, \_value, \_\_1\_value

Following are **illegal identifiers** −

$salary, 123abc, -salary

Operator Identifiers

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

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.

Mixed Identifiers

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

Following are legal mixed identifiers −

unary\_+, myvar\_=

Here, unary\_+ used as a method name defines a unary + operator and myvar\_= used as method name defines an assignment operator (operator overloading).

Literal Identifiers

A literal identifier is an arbitrary string enclosed in back ticks (` . . . `).

Following are legal literal identifiers −

`x` `<clinit>` `yield`

Scala Keywords

The following list shows the reserved words in Scala. These reserved words may not be used as constant or variable or any other identifier names.

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |  |
| - | : | = | => |
| <- | <: | <% | >: |
| # | @ |  |  |

Comments in Scala

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 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!")

}

}

Blank Lines and Whitespace

A line containing only whitespace, possibly with a comment, is known as a blank line, and Scala totally ignores it. Tokens may be separated by whitespace characters and/or comments.

Newline Characters

Scala is a line-oriented language where statements may be terminated by semicolons (;) or newlines. A semicolon at the end of a statement is usually optional. You can type one if you want but you don't have to if the statement appears by itself on a single line. On the other hand, a semicolon is required if you write multiple statements on a single line. Below syntax is the usage of multiple statements.

val s = "hello"; println(s)

Scala Packages

A package is a named module of code. For example, the Lift utility package is net.liftweb.util. The package declaration is the first non-comment line in the source file as follows −

package com.liftcode.stuff

Scala packages can be imported so that they can be referenced in the current compilation scope. The following statement imports the contents of the scala.xml package −

import scala.xml.\_

You can import a single class and object, for example, HashMap from the scala.collection.mutable package −

import scala.collection.mutable.HashMap

You can import more than one class or object from a single package, for example, TreeMap and TreeSet from the scala.collection.immutable package −

import scala.collection.immutable.{TreeMap, TreeSet}

Apply Dynamic

A marker trait that enables dynamic invocations. Instances x of this trait allow method invocations x.meth(args) for arbitrary method names meth and argument lists args as well as field accesses x.field for arbitrary field namesfield. This feature is introduced in Scala-2.10.

If a call is not natively supported by x (i.e. if type checking fails), it is rewritten according to the following rules −

foo.method("blah") ~~> foo.applyDynamic("method")("blah")

foo.method(x = "blah") ~~> foo.applyDynamicNamed("method")(("x", "blah"))

foo.method(x = 1, 2) ~~> foo.applyDynamicNamed("method")(("x", 1), ("", 2))

foo.field ~~> foo.selectDynamic("field")

foo.varia = 10 ~~> foo.updateDynamic("varia")(10)

foo.arr(10) = 13 ~~> foo.selectDynamic("arr").update(10, 13)

foo.arr(10) ~~> foo.applyDynamic("arr")(10)

Variables are nothing but reserved memory locations to store values. This means that when you create a variable, you reserve some space in memory.

## Variable Declaration

Scala has a different syntax for declaring variables. They can be defined as value, i.e., constant or a variable. Here, myVar is declared using the keyword var. It is a variable that can change value and this is called **mutable variable**. Following is the syntax to define a variable using **var** keyword −

### Syntax

var myVar : String = "Foo"

Here, myVal is declared using the keyword val. This means that it is a variable that cannot be changed and this is called **immutable variable**. Following is the syntax to define a variable using val keyword −

### Syntax

val myVal : String = "Foo"