SCALA

programming language

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com


https://www.facebook.com/tutorialspointindia


https://twitter.com/tutorialspoint

About the Tutorial

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.

Audience

This tutorial has been prepared for beginners to help them understand the basics of Scala in simple and easy steps. After completing this tutorial, you will find yourself at a moderate level of expertise in using Scala from where you can take yourself to next levels.

Prerequisites

Scala Programming is based on Java, so if you are aware of Java syntax, then it's pretty easy to learn Scala. Further if you do not have expertise in Java but if you know any other programming language like C, C++ or Python then it will also help in grasping Scala concepts very quickly.

Disclaimer & Copyright

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

tutorialspoint
SIMPLYEASYLEARNING

Table of Contents

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

1. SCALA – 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.

Here we have presented a few points that makes Scala the first choice of application developers.

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 functions**. 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
- The Play framework
- The Bowler framework

tutorialspoint
SIMPLYEASYLEARNING

2. SCALA – ENVIRONMENT

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.

Follow the steps given below to install Scala.

Step 1: Verify Your Java Installation

First of all, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute any of the following two commands depending on the platform you are working on.

If the Java installation has been done properly, then it will display the current version and specification of your Java installation. A sample output is given in the following table.

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)
Linux	Open Command terminal and type: **$java –version**	Java version "1.8.0_31" Open JDK Runtime Environment (rhel-2.8.10.4.el6_4-x86_64) Open JDK 64-Bit Server VM (build 25.31-b07, mixed mode)

tutorialspoint
SIMPLYEASYLEARNING

We assume that the readers of this tutorial have Java SDK version 1.8.0_31 installed on their system.

In case you do not have Java SDK, download its current version from http://www.oracle.com/technetwork/java/javase/downloads/index.html and install it.

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,

Platform	Description
Windows	Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60
Linux	Export JAVA_HOME=/usr/local/java-current

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

Platform	Description
Windows	Append the String "C:\Program Files\Java\jdk1.7.0_60\bin" to the end of the system variable PATH.
Linux	Export PATH=$PATH:$JAVA_HOME/bin/

Execute the command **java -version** from the command prompt as explained above.

Step 3: Install Scala

You can download Scala from http://www.scala-lang.org/downloads. 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

tutorialspoint
SIMPLYEASYLEARNING

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.
Linux	**Command:** $java –jar scala-2.9.0.1-installer.jar **Output:** Welcome to the installation of Scala 2.9.0.1! The homepage is at: http://Scala-lang.org/ press 1 to continue, 2 to quit, 3 to redisplay 1 ..... [Starting to unpack] [Processing package: Software Package Installation (1/1)] [Unpacking finished] [Console installation done]	During installation, it will ask for license agreement, to accept it type 1 and it will ask a path where Scala will be installed. I entered */usr/local/share*, 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

tutorialspoint
SIMPLYEASYLEARNING

Windows	\>scala -version	Scala code runner version 2.11.5 -- Copyright 2002-2013, LAMP/EPFL
Linux	$scala -version	Scala code runner version 2.9.0.1 – Copyright 2002-2013, LAMP/EPFL

tutorialspoint
SIMPLYEASYLEARNING

3. SCALA – BASICS

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 instant 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 object of its type support.

- **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 instant 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:

tutorialspoint
SIMPLYEASYLEARNING

```
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.

```
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:

tutorialspoint
SIMPLYEASYLEARNING

```
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**:

tutorialspoint
SIMPLYEASYLEARNING

```
$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

tutorialspoint
SIMPLYEASYLEARNING

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.

tutorialspoint
SIMPLYEASYLEARNING

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")
```


tutorialspoint
SIMPLYEASYLEARNING

```
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)
```


tutorialspoint
SIMPLYEASYLEARNING

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**

tutorialspoint
SIMPLYEASYLEARNING