Scala stands for Scalable Language.Scala was created by Martin Odersky.Scala smoothly **integrates the features of object-oriented and functional languages**. It is a hybrid functional programming language. It is a multi-paradigm programming language. Scala is compiled to run on the Java Virtual Machine.

# Scala - Environment Setup:

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

|  |  |  |
| --- | --- | --- |
| **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) |

If 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,

|  |  |
| --- | --- |
| **Sr.No** | **Platform & Description** |
| 1 | **Windows**  Set JAVA\_HOME to C:\ProgramFiles\java\jdk1.7.0\_60 |
| 2 | **Linux**  Export JAVA\_HOME=/usr/local/java-current |

Append the full path of Java compiler locationto 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. |
| 2 | **Linux**  Export PATH=$PATH:$JAVA\_HOME/bin/ |

Execute the command **java -version** from the command prompt.

Step 3: Install Scala

Download Scala from [http://www.scala-lang.org/downloads](http://www.scala-lang.org/download/).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. |
| 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/](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. |

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 |
| Linux | $**scala–version** | Scala code runner version 2.9.0.1 – Copyright 2002-2013, LAMP/EPFL |

If you know 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.

Scala program: it can be defined as a collection of objects that communicate via invoking each other’s methods. Let us see 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 thatdescribes 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 dependson 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.

**Scala Program Example**:

Execute a Scala program in two modes:

* **interactive mode** : use command prompt to execute one statement at a time.

Hadoop\_user]$scala

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

Output: Hello, Scala!

* **script mode**: write a Scala program in script mode. Open notepad and add the following code into it.

object HelloWorld {

def main(args: Array[String]) {

println("Hello, world!") // prints Hello World

}

}

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

\>scalacHelloWorld.scala

\>scala HelloWorld

Output:

Hello, World!

‘**scalac**’ command: used to compile the Scala program and it will generate dot class files (**HelloWorld.class bytecode)** which can run on Java virtua machine(JVM) using **scala**.

**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** − defmyMethodName()

* **Program File Name** − Name of the program file should exactly match the object name.

**Example** − 'HelloWorld' is the object name.

File should be saved as 'HelloWorld.scala'.

* **defmain(args: Array[String])** − Scala program processing starts from the **main() method which is amandatory 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 identifierand 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**.

Example: age, salary, \_value, \_\_1\_value

Following are **illegal identifiers** −

$salary, 123abc, -salary

**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 //

multi-line comments very similar to Java. /\* \*/

Scala has all the same data types as Java, with the same memory footprint and precision. All the data types listed are objects. There are no primitive types like in Java

|  |  |
| --- | --- |
| **Data Types** | **Description** |
| **Byte** | 8 bit signed value.  Range from -128 to 127 |
| **Short** | 8 bit signed value. Range from -128 to 127 |
| **Int** | 32 bit signed value. Range -2147483648 to 2147483647 |
| **Long** | 64 bit signed value. -9223372036854775808 to 9223372036854775807 |
| **Float** | 32 bit IEEE 754 single-precision float |
| **Double** | 64 bit IEEE 754 double-precision float |
| **Char** | 16 bit unsigned Unicode character. Range from U+0000 to U+FFFF |
| **String** | A sequence of Chars |
| **Boolean** | Either the literal true or the literal false |
| **Unit** | Corresponds to no value |
| **Null** | null or empty reference |
| **Nothing** | The subtype of every other type; includes no values |
| **Any** | The supertype of any type; any object is of type *Any* |
| **AnyRef** | The supertype of any reference type |

All the data types listed above are objects. There are no primitive types like in Java. This means that you can call methods on an Int, Long, etc.

**Variables in Scala**

Variables are reserved memory locations where values are stored. Which can be referred later in the program.

Scala has two kinds of variables.

1. Immutable

2. Mutable

Immutable variables – These kinds of variables can’t be changed. It is declared using the Val keyword.

Example

Val name:String=”Rahul”

Mutable variables – These kinds of variables can be changed. It is declared using var keyword.

Example

varname:String=”Rahul”

**Class and Objects:**

import java.io.\_

class Point(xc: Int, yc: Int) {

var x: Int = xc

var y: Int = yc

def move(dx: Int, dy: Int) {

x = x + dx

y = y + dy

println ("Point x location : " + x);

println ("Point y location : " + y);

}

}

object Demo {

defmain(args: Array[String]) {

val pt = new Point(10, 20);

pt.move(10, 10);

}

}

Save the above program in Demo.scala.

\>scalacDemo.scala

\>scala Demo

Point x location : 20

Point y location : 30

class Location(override val xc: Int, override valyc: Int,

valzc :Int) extends Point(xc, yc){

var z: Int = zc

defmove(dx: Int, dy: Int, dz: Int) {

x = x + dx

y = y + dy

z = z + dz

println ("Point x location : " + x);

println ("Point y location : " + y);

println ("Point z location : " + z);

}

}

Val loc = new Location(10, 20, 15);

loc.move(10, 10, 5);

A function is a group of statements that perform a task. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically, the division usually is so that each function performs a specific task.

Scala has both functions and methods and we use the terms method and function interchangeably with a minor difference. A Scala method is a part of a class which has a name, a signature, optionally some annotations, and some bytecode where as a function in Scala is a complete object which can be assigned to a variable. In other words, a function, which is defined as a member of some object, is called a method.

A function definition can appear anywhere in a source file and Scala permits nested function definitions, that is, function definitions inside other function definitions. Most important point to note is that Scala function's name can have characters like +, ++, ~, &,-, --, \, /, :, etc.

Function Declarations

A Scala function declaration has the following form −

Def functionName ([list of parameters]) : [return type]

Methods are implicitly declared abstract if you don’t use the equals sign and the method body.

Function Definitions

A Scala function definition has the following form −

Syntax

Def functionName ([list of parameters]) : [return type] = {

function body

return [expr]

}

Here, return type could be any valid Scala data type and list of parameters will be a list of variables separated by comma and list of parameters and return type are optional. Very similar to Java, a return statement can be used along with an expression in case function returns a value. Following is the function which will add two integers and return their sum −

Syntax

object add {

def addInt( a:Int, b:Int ) : Int = {

var sum:Int = 0

sum = a + b

return sum

}

}

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, they are called procedures.

Syntax

Here is the syntax −

object Hello{

def printMe( ) : Unit = {

println("Hello, Scala!")

}

}

Calling Functions

Scala provides a number of syntactic variations for invoking methods. Following is the standard way to call a method −

functionName( list of parameters )

If a function is being called using an instance of the object, then we would use dot notation similar to Java as follows −

[instance.]functionName( list of parameters )

Try the following example program to define and then call the same function.

Example

object Demo {

def main(args: Array[String]) {

println( "Returned Value : " + addInt(5,7) );

}

defaddInt( a:Int, b:Int ) : Int = {

varsum:Int = 0

sum = a + b

return sum

}

}

Save the above program in Demo.scala. The following commands are used to compile and execute this program.

Command

\>scalacDemo.scala

\>scala Demo

Output

Returned Value : 12

Scala functions are the heart of Scala programming and that's why Scala is assumed as a functional programming language. Following are few important concepts related to Scala functions which should be understood by a Scala programmer.

**Pattern matching:** is the second most widely used feature of Scala, after function values and closures. Scala provides great support for pattern matching, in processing the messages.

**It works same as switch case in other programming languages.** It matches best case available in the pattern.A pattern match includes a sequence of alternatives**, each starting with the keyword case**. Each alternative includes a **pattern** and one or more **expressions**, which will be evaluated if the pattern matches. An **arrow symbol => separates the pattern from the expressions**.

Example to match against an integer value:

object MainObject {

defmain(args: Array[String]) {

var a = 1

a match{

case 1 =>println("One")

case 2 =>println("Two")

case \_ =>println("No") //Default case

}

}

}

Compile: scala>scalacMainObject.scala

Scala>scalaMainObject

Output: One

Example:

object Demo {

defmain(args: Array[String]) {

println(matchTest(3))

}

Def matchTest(x: Int): String = x match {

case 1 => "one"

case 2 => "two"

case \_ => "many"

}

}

Output: many

Example:

object MainObject {

defmain(args: Array[String]) {

var result = search ("Hello")

print(result)

}

def search (a:Any):Any = a match{

case 1 =>println("One")

case "Two" =>println("Two")

case "Hello" =>println("Hello")

case \_ =>println("No")

}

}

Ouput: Hello

The block with the case statements defines a function, which maps integers to strings. Try the following example program, which matches a value against patterns of different types.

Example:

object Demo {

defmain(args: Array[String]) {

println(matchTest("two"))

println(matchTest("test"))

println(matchTest(1))

}

defmatchTest(x: Any): Any = x match {

case 1 => "one"

case "two" => 2

case y: Int => "scala.Int"

case \_ => "many"

}

}

Save the above program in **Demo.scala**. The following commands are used to compile and execute this program.

Command

\>scalacDemo.scala

\>scala Demo

Output

2

many

one

**Matching using Case Classes:**The **case classes** are special classes that are used in pattern matching with case expressions. Syntactically, these are standard classes with a special modifier: **case**.

Simple pattern matching example using case class.

Example

object Demo {

defmain(args: Array[String]) {

val alice = new Person("Alice", 25)

val bob = new Person("Bob", 32)

val charlie = new Person("Charlie", 32)

for (person <- List(alice, bob, charlie)) {

person match {

case Person("Alice", 25) =>println("Hi Alice!")

case Person("Bob", 32) =>println("Hi Bob!")

case Person(name, age) =>println(

"Age: " + age + " year, name: " + name + "?")

}

}

}

case class Person(name: String, age: Int)

}

Save the above program in **Demo.scala**. The following commands are used to compile and execute this program.

Command

\>scalacDemo.scala

\>scala Demo

Output

Hi Alice!

Hi Bob!

Age: 32 year, name: Charlie?