**Introduction:**

* Scala is an object-oriented and functional programming language.
* Scala is a general-purpose programming language. It supports object oriented, functional and imperative programming approaches. It is a strong static type language.
* In scala**, everything is an object whether it is a function or a number. It does not have concept of primitive data.**
* It was designed by Martin Odersky. It was officially released for java platform in early 2004 and for .Net framework in June 2004. Later on, Scala dropped .Net support in 2012.
* Scala is influenced by Java.
* File extension of scala source file may be either .scala or .sc
* You can create any kind of application like web application, enterprise application, mobile application, desktop based application etc.

Scalable:

* Web applications
* Utilities and libraries
* Parallel batch processing
* Concurrency and distributed application
* Data analysis with Spark

Scala Program Example

**object** MainObject{

**def** main(args:Array[String]){

        print("Hello Scala")

    }

}

* Scala is not an extension of Java, but it is completely interoperable with it. While compilation, Scala file translates to Java bytecode and runs on JVM (Java Virtual machine).
* Scala was designed to be both object-oriented and functional. It is a pure object-oriented language in the sense that every value is an object and functional language in the sense that every function is a value. The name of scala is derived from word scalable which means it can grow with the demand of users.

**why scala**

Popularity of Scala

* Twitter have announced that it had switched large portions of its backend from Ruby to Scala and intended to convert the rest.
* Apple Inc. uses Scala in certain teams, along with Java and the Play framework.
* The New York Times revealed in 2014 that its internal content management system Blackbeard is built using Scala, Akka and Play Framework.
* There are teams within Google that use Scala, mostly due to acquisitions such as Firebase and Nest.
* The Walmart Canada Uses Scala for their back end platform.

Features of Scala

Type Inference

In Scala, you don't require to mention data type and function return type explicitly. Scala is enough smart to deduce the type of data. The return type of function is determined by the type of last expression present in the function.

Singleton object

In Scala, there are no static variables or methods. Scala uses singleton object, which is essentially class with only one object in the source file. Singleton object is declared by using object instead of class keyword.

Immutability

Scala uses immutability concept. Each declared variable is immutable by default. Immutable means you can't modify its value. You can also create mutable variables which can be changed.

Immutable data helps to manage concurrency control which requires managing data.

Lazy Computation

In Scala, computation is lazy by default. Scala evaluates expressions only when they are required. You can declare a lazy variable by using lazy keyword. It is used to increase performance.

Case classes and Pattern matching

Scala case classes are just regular classes which are immutable by default and decomposable through pattern matching.

All the parameters listed in the case class are public and immutable by default.

Case classes support pattern matching. So, you can write more logical code.

Concurrency control

Scala provides standard library which includes the actor model. You can write concurrency code by using actor. Scala provides one more platform and tool to deal with concurrency known as Akka. Akka is a separate open source framework that provides actor-based concurrency. Akka actors may be distributed or combined with software transactional memory.

String Interpolation

Since Scala 2.10.0, Scala offers a new mechanism to create strings from your data. It is called string interpolation. String interpolation allows users to embed variable references directly in processed string literals. Scala provides three string interpolation methods: s, f and raw.

Higher Order Functions

Higher order function is a function that either takes a function as argument or returns a function. In other words, we can say a function which works with another function is called higher order function.

Higher order function allows you to create function composition, lambda function or anonymous function etc.

Traits

A trait is like an interface with a partial implementation. In Scala, trait is a collection of abstract and non-abstract methods. You can create trait that can have all abstract methods or some abstract and some non-abstract methods.

Traits are compiled into Java interfaces with corresponding implementation classes that hold any methods implemented in the traits.

Rich Set of Collection

Scala provides rich set of collection library. It contains classes and traits to collect data. These collections can be mutable or immutable. You can use it according to your requirement. Scala.collection.mutable package contains all the mutable collections. You can add, remove and update data while using this package.

Scala.collection.immutable package contains all the immutable collections. It does not allow you to modify data.

**Tool setup**

Installing Scala : <https://www.scala-lang.org/download/>

To write scala program you need to install scala on your machine. You must have latest jdk installed on your machine because scala compiler creates .class file which is a byte code. Scala interpreter executes this byte code by using jvm (Java Virtual Machine).

**Compiling and running a code/script with scala**

Simple program on scala

Hello Scala

**object** ScalaExample{

**def** main(args:Array[String]){

        println "Hello Scala"

    }

}

 we have created an object ScalaExample. It contains a main method and display message using println method.

This file is saved with the name **ScalaExample.scala.**

Command to compile this code is: **scalac ScalaExample.scala**

Command to execute the compiled code is: **scala ScalaExample**

After executing code it yields the following output.

Output:

Hello Scala

You can also use IDE (Integrated Development Environment) for executing scala code.

The above example is written using object oriented approach. You can also use functional approach to write code in scala.

Scala Example: Hello Scala

Below is the example by using functional approach.

def scalaExample{

println("Hello Scala")

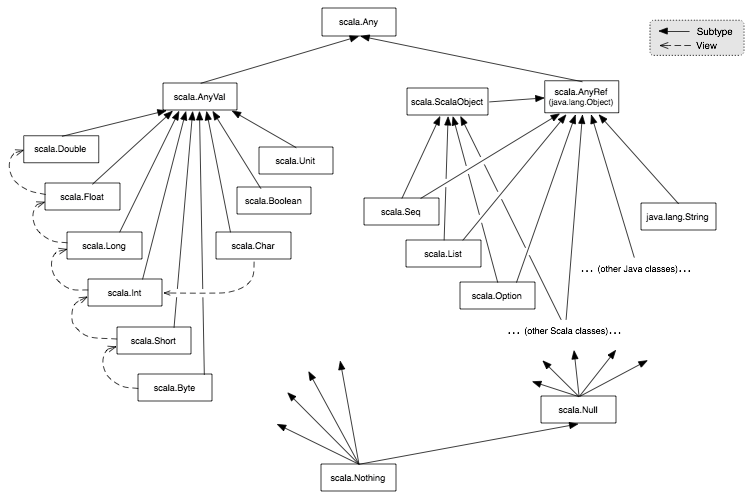
}

scalaExample // Calling of function

Output:

Hello Scala

**Type Hierarchy**



AnyVal is the root class of all *value types*, which describe values not implemented as objects in the underlying host system. The value classes are specified in SLS 12.2.  
The standard implementation includes nine AnyVal subtypes:  
Double, Float, Long, Int, Char, Short, and Byte are the *numeric value types*.  
Unit and Boolean are the *non-numeric value types*.

Scala Variables and Data Types

Variable is a name which is used to refer memory location. You can create mutable and immutable variable in scala. Let's see how to declare variable.

Mutable Variable

You can create mutable variable using var keyword. It allows you to change value after declaration of 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 data type of variable explicitly as we have used in below.

Another example of variable

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

Immutable Variable

val data = 100

data = 101 // Error: reassignment to val

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

Data Types in Scala

Data types in scala are much similar to java in terms of their storage, length, except that in scala there is no concept of primitive data types every type is an object and starts with capital letter. A table of data types is given below.

Data Type Default Value Size

Boolean False True or false

Byte 0 8 bit signed value (-27 to 27-1)

Short 0 16 bit signed value(-215 to 215-1)

Char '\u0000' 16 bit unsigned Unicode character(0 to 216-1)

Int 0 32 bit signed value(-231 to 231-1)

Long 0L 64 bit signed value(-263 to 263-1)

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

Scala Conditional Expressions

Scala provides if statement to test the conditional expressions. It tests boolean conditional expression which can be either true or false. Scala use various types of if else statements.

If statement

If-else statement

Nested if-else statement

If-else-if ladder statement

Scala if statement

The scala if statement is used to test condition in scala. If block executes only when condition is true otherwise execution of if block is skipped.

Syntax

if(condition){

// Statements to be executed

}

Ex:

var age:Int = 20;

if(age > 18){

println ("Age is greate than 18")

}

Output:

Age is greate than 18

Scala If-Else Statement

The scala if-else statement tests the condition. If the condition is true, if block executes otherwise else block executes.

Syntax

if(condition){

// If block statements to be executed

} else {

// Else bock statements to be executed

}

Ex:

var number:Int = 21

if(number%2==0){

println("Even number")

}else{

println("Odd number")

}

Output:

Odd number

Scala If-Else-If Ladder Statement

The scala if-else-if ladder executes one condition among the multiple conditional statements.

Syntax

if (condition1){

//Code to be executed if condition1 is true

} else if (condition2){

//Code to be executed if condition2 is true

} else if (condition3){

//Code to be executed if condition3 is true

}

...

else {

//Code to be executed if all the conditions are false

}

Ex:

var number:Int = 85

if(number>=0 && number<50){

println ("fail")

}

else if(number>=50 && number<60){

println("D Grade")

}

else if(number>=60 && number<70){

println("C Grade")

}

else if(number>=70 && number<80){

println("B Grade")

}

else if(number>=80 && number<90){

println("A Grade")

}

else if(number>=90 && number<=100){

println("A+ Grade")

}

else println ("Invalid")

Output:

A Grade

Scala If Statement as better alternative of Ternary Operators

In scala, you can assign if statement result to a function. Scala does not have ternary operator concept like C/C++ but provides more powerful if which can return value. Let's see an example

Example

object MainObject {

def main(args: Array[String]) {

val result = checkIt(-10)

println (result)

}

def checkIt (a:Int) = if (a >= 0) 1 else -1 // Passing a if expression value to function

}

Output:

-1

Scala Pattern Matching

Pattern matching is a feature of scala. It works same as switch case in other programming languages. It matches best case available in the pattern.

Let's see an example.

Scala Pattern Matching Example

object MainObject {

def main(args: Array[String]) {

var a = 1

a match{

case 1 => println("One")

case 2 => println("Two")

case \_ => println("No")

}

}

}

In the above example, we have implemented a pattern matching.

Here, match using a variable named a. This variable matches with best available case and prints output. Underscore (\_) is used in the last case for making it default case.

Output:

One

Match expression can return case value also. In next example, we are defining method having a match with cases for any type of data. Any is a class in scala which is a super class of all data types and deals with all type of data. Let's see an example.

Scala Pattern Matching Example2

object MainObject {

def main(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")

}

}

Output:

Hello

Scala while loop

In Scala, while loop is used to iterate code till the specified condition. It tests boolean expression and iterates again and again. You are recommended to use while loop if you don't know number of iterations prior.

Syntax

while(boolean expression){

// Statements to be executed

}

Flowchart:

Scala While loop 1

Scala while loop Example

object MainObject {

def main(args: Array[String]) {

var a = 10; // Initialization

while( a<=20 ){ // Condition

println(a);

a = a+2 // Incrementation

}

}

}

Output:

10

12

14

16

18

20

Scala Infinite While Loop Example

You can also create an infinite while loop. In the below program, we just passed true in while loop. Be careful, while using infinite loop.

object MainObject {

def main(args: Array[String]) {

var a = 10; // Initialization

while( true ){ // Condition

println(a);

a = a+2 // Incrementation

}

}

}

Output:

10

12

14

16

?

Ctr+Z // To stop execution

Scala do-while loop example

object MainObject {

def main(args: Array[String]) {

var a = 10; // Initialization

do {

println( a );

a = a + 2; // Increment

}

while( a <= 20 ) // Condition

}

}

Output:

10

12

14

16

18

20

Scala Infinite do-while loop

In scala, you can create infinite do-while loop. To create infinite loop just pass true literal in loop condition.

Let's see an example.

object MainObject {

def main(args: Array[String]) {

var a = 10; // Initialization

do {

println( a );

a = a + 2; // Increment

}

while( true) // Condition

}

}

Output:

10

12

14

16

?

Ctrl+Z // To stop execution of program

Scala for loop

In scala, for loop is known as for-comprehensions. It can be used to iterate, filter and return an iterated collection. The for-comprehension looks a bit like a for-loop in imperative languages, except that it constructs a list of the results of all iterations.

Syntax

for( i <- range){

// statements to be executed

}

In the above syntax, range is a value which has start and end point. You can pass range by using to or until keyword.

Scala for-loop example by using to keyword

object MainObject {

def main(args: Array[String]) {

for( a <- 1 to 10 ){

println(a);

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

In the below example, until is used instead of to. The major difference between until and to is, to includes start and end value given in the range, while until excludes last value of the range. So, the below example will print only 1 to 9.

Scala for-loop Example by using until keyword

object MainObject {

def main(args: Array[String]) {

for( a <- 1 until 10 ){

println(a);

}

}

}

Output:

1

2

3

4

5

6

7

8

9

It is helpful to apply until keyword when you are iterating string or array, because array range is 0 to n-1. until does not exceed to n-1. So, your code will not complain of upper range.

Scala for-loop filtering Example

You can use for to filter your data. In the below example, we are filtering our data by passing a conditional expression. This program prints only even values in the given range.

object MainObject {

def main(args: Array[String]) {

for( a <- 1 to 10 if a%2==0 ){

println(a);

}

}

}

Output:

2

4

6

8

10

Scala for-loop Example by using yield keyword

In the above example, we have used yield keyword which returns a result after completing of loop iterations. The for use buffer internally to store iterated result and after finishing all iterations it yields the final result from that buffer. It does not work like imperative loop.

object MainObject {

def main(args: Array[String]) {

var result = for( a <- 1 to 10) yield a

for(i<-result){

println(i)

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

Scala for-loop in Collection

In scala, you can iterate collections like list, sequence etc, either by using for each loop or for-comprehensions.

Let's see an example.

Scala for- loop Example for Iterating Collection

object MainObject {

def main(args: Array[String]) {

var list = List(1,2,3,4,5,6,7,8,9) // Creating a list

for( i <- list){ // Iterating the list

println(i)

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

Scala for-each loop Example for Iterating Collection

In the below code we have use three approaches of for-each loop. You can implement any of them according to your need.

object MainObject {

def main(args: Array[String]) {

var list = List(1,2,3,4,5,6,7,8,9) // Creating a list

list.foreach{

println // Print each element

}

list.foreach(print)

println

list.foreach((element:Int)=>print(element+" ")) // Explicitly mentioning type of elements

}

}

Output:

1

2

3

4

5

6

7

8

9

123456789

1 2 3 4 5 6 7 8 9

Scala for-loop Example using by keyword

In the above example, we have used by keyword. The by keyword is used to skip the iteration. When you code like: by 2 it means, this loop will skip all even iterations of loop.

object MainObject {

def main(args: Array[String]) {

for(i<-1 to 10 by 2){

println(i)

}

}

}

Output:

1

3

5

7

9

Scala Break

Break is used to break a loop or program execution. It skips the current execution. Inside inner loop it breaks the execution of inner loop.

In scala, there is no break statement but you can do it by using break method and by importing scala.util.control.Breaks.\_ package. Let's see an example.

Scala Break Example

import scala.util.control.Breaks.\_ // Importing package

object MainObject {

def main(args: Array[String]) {

breakable { // Breakable method to avoid exception

for(i<-1 to 10 by 2){

if(i==7)

break // Break used here

else

println(i)

}

}

}

}

Output:

1

3

5

Scala Break Example: Break inner Loop

You can use break statement to terminate execution of inner loop in nested loop.

Let's see an example.

import scala.util.control.Breaks.\_

object MainObject {

def main(args: Array[String]) {

for(i <- 1 to 3){

breakable {

for(j <- 1 to 3){

if(i == 2 & j == 2 )

break

println(i+" "+j)

}

}

}

}

}

Output:

1 1

1 2

1 3

2 1

3 1

3 2

3 3

Scala Comments

The scala comments are statements which are not executed by the compiler or interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code details.

In scala, there are three types of comments

Single line comment

Multiline comment

Documentation comment

Scala Single Line Comment Example

Single line comment is used to comment single line of code.

// Example of single line comment.

object MainObject {

def main(args: Array[String]) {

var a = 1 // Here, a is a variable

println(a)

}

}

Output:

1

Scala Multiline Comment

Multiline comment is used to comment multiple lines of code in the program.

// Example of multi line comment.

object MainObject {

def main(args: Array[String]) {

var a = 1

println(a)

}

/\*

In the main method, we have created a variable named a

and printed it

\*/

}

Output:

1

Scala Documentation Comment Example

// Example of documentation comment.

object MainObject {

def main(args: Array[String]) {

var a = 1

println(a)

}

/\*\*

\* In the main method, we have created a variable named a

\* and printed it

\*

\*/

}

Output:

1

Scala Functions

Scala supports functional programming approach. It provides rich set of built-in functions and allows you to create user defined functions also.

In scala, functions are first class values. You can store function value, pass function as an argument and return function as a value from other function. You can create function by using def keyword. You must mention return type of parameters while defining function and return type of a function is optional. If you don't specify return type of a function, default return type is Unit.

Scala Function Declaration Syntax

def functionName(parameters : typeofparameters) : returntypeoffunction = {

// statements to be executed

}

In the above syntax, = (equal) operator is looking strange but don't worry scala has defined it as:

You can create function with or without = (equal) operator. If you use it, function will return value. If you don't use it, your function will not return anything and will work like subroutine.

Scala functions don?t use return statement. Return type infers by compiler from the last expression or statement present in the function.

Scala Function Example without using = Operator

The function defined below is also known as non parameterized function.

object MainObject {

def main(args: Array[String]) {

functionExample() // Calling function

}

def functionExample() { // Defining a function

println("This is a simple function")

}

}

Output:

This is a simple function

Scala Function Example with = Operator

object MainObject {

def main(args: Array[String]) {

var result = functionExample() // Calling function

println(result)

}

def functionExample() = { // Defining a function

var a = 10

a

}

}

Output:

10

Scala Parameterized Function Example

when using parameterized function you must mention type of parameters explicitly otherwise compiler throws an error and your code fails to compile.

object MainObject {

def main(args: Array[String]) = {

functionExample(10,20)

}

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

var c = a+b

println(c)

}

}

Output:

30

Scala Recursion Function

In the program given below, we are multiplying two numbers by using recursive function.

In scala, you can create recursive functions also. Be careful while using recursive function. There must be a base condition to terminate program safely.

object MainObject {

def main(args: Array[String]) = {

var result = functionExample(15,2)

println(result)

}

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

if(b == 0) // Base condition

0

else

a+functionExample(a,b-1)

}

}

Output:

30

Function Parameter with Default Value

Scala provides a feature to assign default values to function parameters. It helps in the scenario when you don't pass value during function calling. It uses default values of parameters.

Let's see an example.

Scala Function Parameter example with default value

object MainObject {

def main(args: Array[String]) = {

var result1 = functionExample(15,2) // Calling with two values

var result2 = functionExample(15) // Calling with one value

var result3 = functionExample() // Calling without any value

println(result1+"\n"+result2+"\n"+result3)

}

def functionExample(a:Int = 0, b:Int = 0):Int = { // Parameters with default values as 0

a+b

}

}

Output:

17

15

0

Scala Function Named Parameter Example

In scala function, you can specify the names of parameters during calling the function. In the given example, you can notice that parameter names are passing during calling. You can pass named parameters in any order and can also pass values only.

Let's see an example.

object MainObject {

def main(args: Array[String]) = {

var result1 = functionExample(a = 15, b = 2) // Parameters names are passed during call

var result2 = functionExample(b = 15, a = 2) // Parameters order have changed during call

var result3 = functionExample(15,2) // Only values are passed during call

println(result1+"\n"+result2+"\n"+result3)

}

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

a+b

}

}

Output:

17

17

17

Scala Higher Order Functions

Higher order function is a function that either takes a function as argument or returns a function. In other words we can say a function which works with function is called higher order function.

Higher order function allows you to create function composition, lambda function or anonymous function etc.

Let's see an example.

Scala Example: Passing a Function as Parameter in a Function

object MainObject {

def main(args: Array[String]) = {

functionExample(25, multiplyBy2) // Passing a function as parameter

}

def functionExample(a:Int, f:Int=>AnyVal):Unit = {

println(f(a)) // Calling that function

}

def multiplyBy2(a:Int):Int = {

a\*2

}

}

Output:

AnyVal is a root class

Double, Short

60

Scala Example: Function Composition

In scala, functions can be composed from other functions. It is a process of composing in which a function represents the application of two composed functions.

Let's see an example.

object MainObject {

def main(args: Array[String]) = {

var result = multiplyBy2(add2(10)) // Function composition

println(result)

}

def add2(a:Int):Int = {

a+2

}

def multiplyBy2(a:Int):Int = {

a\*2

}

}

Output:

24

Scala Anonymous (lambda) Function

Anonymous function is a function that has no name but works as a function. It is good to create an anonymous function when you don't want to reuse it latter.

You can create anonymous function either by using => (rocket) or \_ (underscore) wild card in scala.

Let's see an example.

Scala Anonymous function Example

object MainObject {

def main(args: Array[String]) = {

var result1 = (a:Int, b:Int) => a+b // Anonymous function by using => (rocket)

var result2 = (\_:Int)+(\_:Int) // Anonymous function by using \_ (underscore) wild card

println(result1(10,10))

println(result2(10,10))

}

}

Output:

20

20

Scala Multiline Expression

Expressions those are written in multiple lines are called multiline expression. In scala, be carefull while using multiline expressions.

The following program explains about if we break an expression into multiline, the scala compiler throw a warning message.

Scala Multiline Expression Example

def add1(a:Int, b:Int) = {

a

+b

}

The above program does not evaluate complete expression and just return b here. So, be careful while using multiline expressions.

Output:

MainObject.scala:3: warning: a pure expression does nothing in statement

position; you may be omitting necessary parentheses

a

^

one warning found

10

You can apply following ways to avoid above problem.

Scala Example Multiline Expression

object MainObject {

def add2(a:Int, b:Int) = {

a+

b

}

def add3(a:Int, b:Int) = {

(a

+b)

}

def main(args: Array[String]) = {

var result2 = add2(10,10)

var result3 = add3(10,10)

println(result2+"\n"+result3)

}

}

Output:

20

20

Scala Function Currying

In scala, method may have multiple parameter lists. When a method is called with a fewer number of parameter lists, then this will yield a function taking the missing parameter lists as its arguments.

In other words it is a technique of transforming a function that takes multiple arguments into a function that takes a single argument.

Scala Function Currying Example

object MainObject {

def add(a:Int)(b:Int) = {

a+b

}

def main(args: Array[String]) = {

var result = add(10)(10)

println("10 + 10 = "+result)

var addIt = add(10)\_

var result2 = addIt(3)

println("10 + 3 = "+result2)

}

}

Output:

20

Scala Nested Functions

Scala is a first class function language which means it allows you to passing function, returning function, composing function, nested function etc. An example below explain about how to define and call nested functions.

Scala Nested Functions Example

object MainObject {

def add(a:Int, b:Int, c:Int) = {

def add2(x:Int,y:Int) = {

x+y

}

add2(a,add2(b,c))

}

def main(args: Array[String]) = {

var result = add(10,10,10)

println(result)

}

}

Output:

30

Scala Function with Variable Length Parameters

In scala, you can define function of variable length parameters. It allows you to pass any number of arguments at the time of calling the function.

Let's see an example.

Scala Example: Function with Variable Length Parameters

def add(args: Int\*) = {

var sum = 0;

for(a <- args) sum+=a

sum

}

var sum = add(1,2,3,4,5,6,7,8,9);

println(sum);

Output:

45