Scala: Scala is JVM programming Language, multi-paradigm programming language (Integrates object oriented and functional programming smoothly), object-oriented language. Scala runs on top of the java virtual machine. So that java compiler compiles our scala code to the same bytecode that java runs. We need to install scala plugin in IntelliJ IDE.

Created sample file in scala Playground.scala

package playground  
object Playground {  
 def main(args: Array[String]): Unit = {  
 *println*("learn scala")  
 }  
}

extends App is equivalent to creating a main function which makes this application runnable.

object Playground extends App {  
}

**Values Variables and Types:**

val *x*: Int =42  
*println*(*x*)

x=2 //getting error

Declared with the val keyword cannot be reassigned. val are immutable similar to final in java.

* Types of the val is optional.

val x=42  
 println(x)

* var *empName* : String ="lalitha"  
  *println*(*empName*.take(4)) o/p: lali

Initializing variables to default values: The variable can be initialized to default values using “\_”. “\_” can be used only with var and not val.

**Types in Scala:**

Int,String,Boolean,Char,Short,Long,Float,Double

* Declared with the var keyword is mutable.

var y : Int=32   
y=33  
println(y)

* Compiler automatically infers types when omitted
* Everything in Scala is an expression but not instruction.

**If Expression:**

var *aCondition*=true  
var *aConditionValue* = if(*aCondition*) 5 else 3  
*println*(*aConditionValue*)

o/p:5

**Unit:**  Unit is a special type in scala, which is equivalent to void in other languages.

var *x*=3  
var *aValue* = (*x*=3)  
*println*(*aValue*);

o/p: ()

Side effects: println(),whiles,reassigning (these return unit)

**Code Blocks:**

The value of the block is the value of its last expression.

val codeBlock = {  
 val y=2  
 val z=y+1  
 if(z>2) "hello" else "goodbye"  
 if(z==3) 42 else 5  
}  
println(codeBlock)

o/p: 42

Difference between “hello world” and println(“hello world”)

“hello world” is literal and the type is string but println(“hello world”) type is unit.

val *codeBlock* = {  
 val y=2  
 val z=y+1  
 if(z>2) "hello" else "goodbye"  
 if(z==3) 42 else 5  
 10  
}  
*println*(*codeBlock*)

o/p: 10

**Functions:** Every function is an object.

* def aFunction(a:String, b:Int) :String={  
   a+" "+b  
  }  
  *println*(*aFunction*("hello",3)) o/p: hello 3
* def parameterLessFunction : Int =42  
  *println*(*parameterLessFunction*) o/p:42

Concatenating the string n times:

(when we will need loops we will use recursion)

def repeatedFunction(aString :String, n:Int) :String={  
 If(n==1) aString  
 else aString+repeatedFunction(aString,n-1)  
 }  
 println(repeatedFunction("hello",3))

if we delete the return type of the recursive function compiler will complain.

def biggerFunction(n: Int): Int={  
 def smallerFunction(a:Int, b:Int) : Int=a+b  
 smallerFunction(n,n-1)  
 }  
 println(biggerFunction(3))

**Stack and Tail Recursion:**

Factorial of number:

def factorial(n:Int):Int={

if(n<=1) 1

else n\*factorial(n-1)

}

println(factorial(5)) o/p:120

println(factorial(5000)) o/p: stack overflow error

@tailrec

def anotherFactorial(n:Int,accumulator:BigInt) :BigInt={  
 if(n<=1) accumulator  
 else *anotherFactorial*(n-1,n\*accumulator)  
}  
*println*(*anotherFactorial*(5000,1))

Fibonacci of a number:

def fibonacci(n:Int): Int={  
 def findFib(i:Int,last:Int,nextToLast :Int):Int={  
 if(i>=n) last  
 else findFib(i+1,last+nextToLast,last)  
 }  
 if(n<=2) 1  
 else findFib(2,1,1)  
}  
*println*(*fibonacci*(8))

**Lazy Values:** initializatiom of these variables that is deferred until they are first accessed

lazy val empMessage =” hi”

**Variable Scopes:**

Scala provides three variable scopes fields,method parameters, local variables.

**Input:** readLine(),readInt(),readFloat()……

**String Interpolators:**

S-interpolators

val *name*="david"  
val *age*=25  
val *greeting*= s"hello my name is **$***name* and I am **$**{*age*+1} years old"  
*println*(*greeting*)

F-interpolators

val *speed*=1.2f  
val *myth*=f"**$***name* can eat **$***speed*%2.2f burgers per min"  
*println*(*myth*)

raw-interpolators

*println*(raw"This is a \n newline")  
val *escaped* = "This is a \n newline"  
*println*(raw"**$***escaped*")

**Call-by-Name and Call-by-Value**

def calledByValue(x: Long) :Unit={  
 *println*("by value: "+x)  
 *println*("by value: "+x)  
}  
  
def calledByName(x: =>Long) :Unit={  
 *println*("by name: " + x)  
 *println*("by name: " + x)  
}  
*calledByValue*(System.*nanoTime*())  
*calledByName*(System.*nanoTime*())

o/p: by value: 90754975571300

by value: 90754975571300

by name: 90755068138400

by name: 90755069418500

In call by name , by name parameter delays the evaluation of the expression passed here until it’s used.

def infinite(): Int =1+*infinite*()  
def printFirst( x: Int, y: =>Int) =*println*(x)  
//printFirst(infinite(),34) stack-overflow error  
*printFirst*(34,*infinite*())

In call by value, value is computed before call. Same value used everywhere.

In call by name expression is passed literally, expression is evaluated at every use within the definition.

**Default and Named arguments:**

def trFact(n:Int, acc:Int=1) :Int={  
 if(n<=1) acc  
 else *trFact*(n-1,n\*acc)  
}  
*println*(*trFact*(4))  
  
def savePicture(format:String="jpg",width:Int=1920,height:Int=1000):Unit=*println*("saving picture")  
*savePicture*(800) //error  
*savePicture*(height=600) =>Named Argument

**Object Oriented Programming in Scala:**