Scala Interpreter and Scala Coding Intro

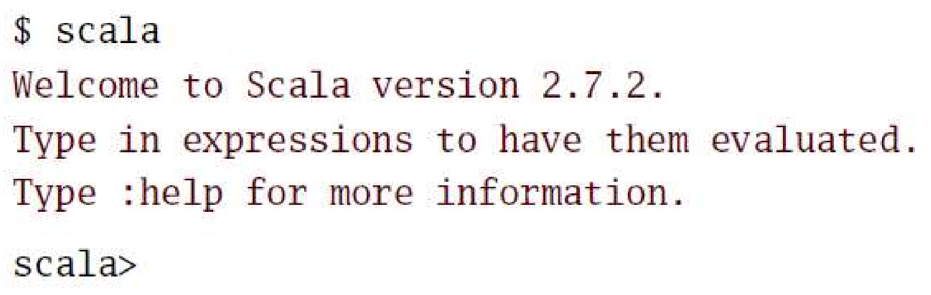
Principles of Programming Languages

Prerequisite

- You should have a standard Scala installation.
- You can also use a Scala plugin for Eclipse, IntelliJ, or NetBeans

Scala Interpreter

- Scala provides a Read Eval Print Loop (REPL) – an interactive shell that compiles Scala code and returns result/type immediately.
- The Scala REPL is instantiated by running scala on the co.mmand line



```
$ scala
Welcome to Scala version 2.7.2.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

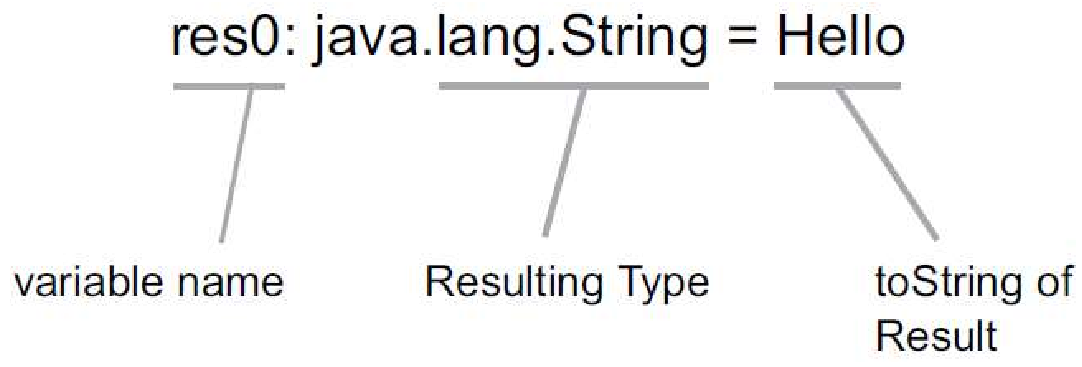
Through REPL

- Observe the following code:

```
scala> "Hello"
res0: java.lang.String = Hello
```

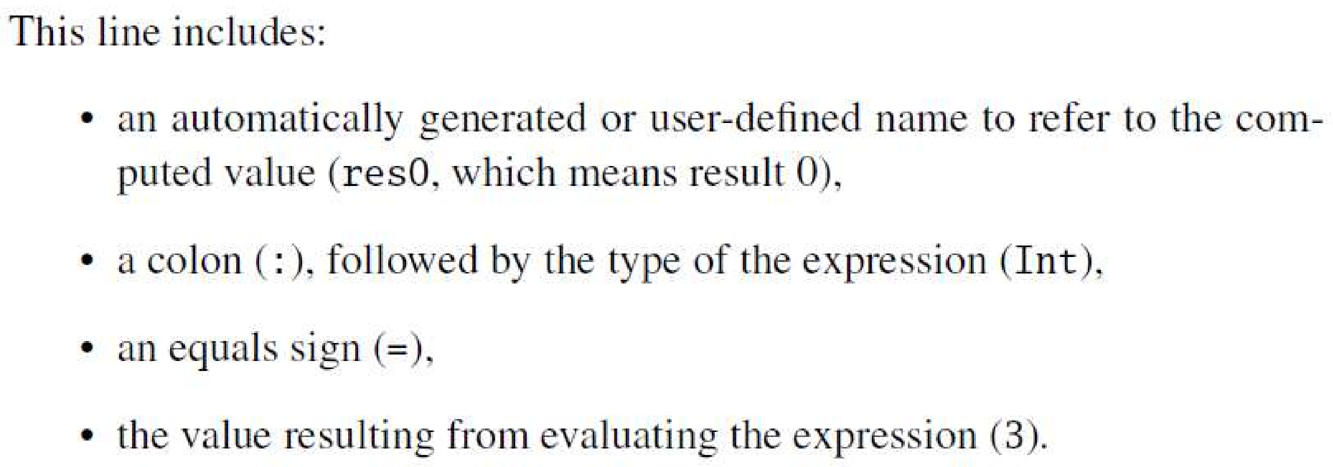
- Notice that after the statement we enter into the interpreter, it prints a line like res0: java.lang.String = Hello

The first part of this expression is a variable name for the expression. The next part of the result expression (after the :) is the static type of the expression. The last part of the result expression is the stringified value of the result.



Scala Interpreter

- After you type an expression, such as 1 + 2, and hit enter: `scala> 1 + 2`
- The interpreter will print: `res0: Int = 3`



This line includes:

- an automatically generated or user-defined name to refer to the computed value (`res0`, which means result 0),
- a colon (`:`), followed by the type of the expression (`Int`),
- an equals sign (`=`),
- the value resulting from evaluating the expression (3).

Scala Interpreter

- The resX identifier may be used in later lines. For instance, since res0 was set to 3 previously, res0 * 3 will be 9:

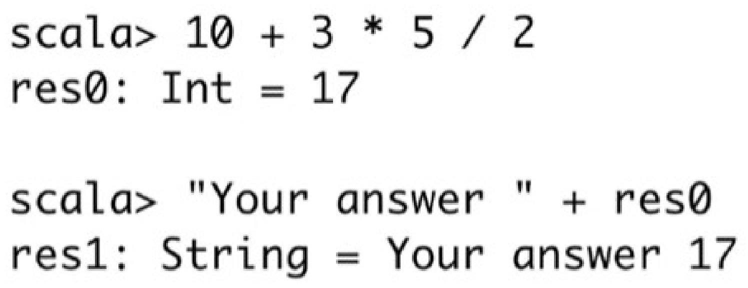
```
scala> res0 * 3
res1: Int = 9
```

- To print the necessary, but not sufficient, Hello, world! greeting, type:

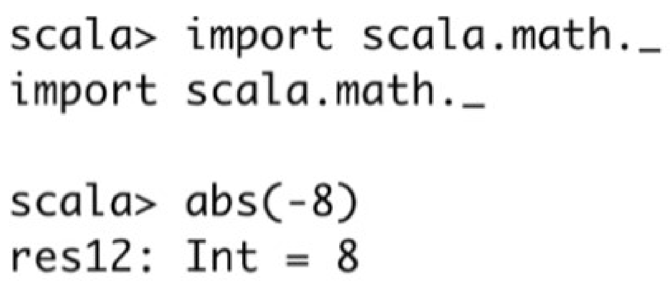
```
scala> println("Hello, world!")
Hello, world!
```

- Similar to System.out.println in Java.

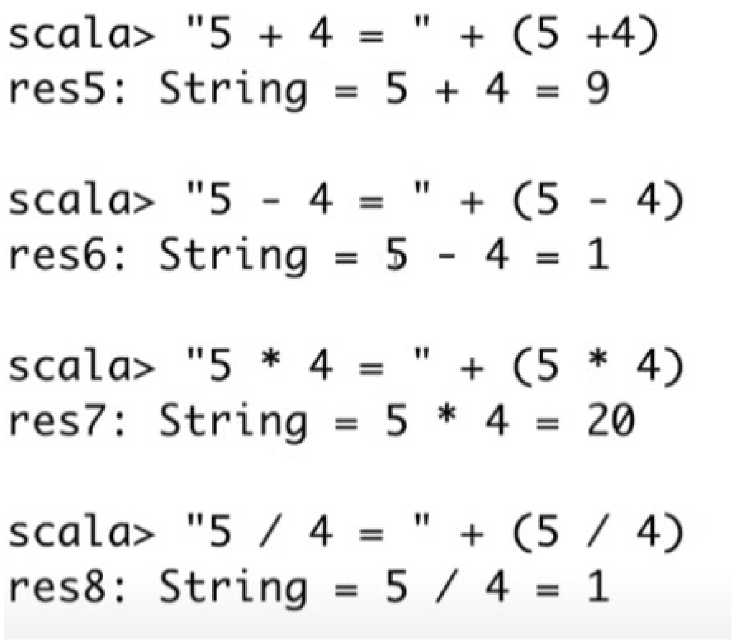
Scala Interpreter



```
scala> 10 + 3 * 5 / 2
res0: Int = 17

scala> "Your answer " + res0
res1: String = Your answer 17
```

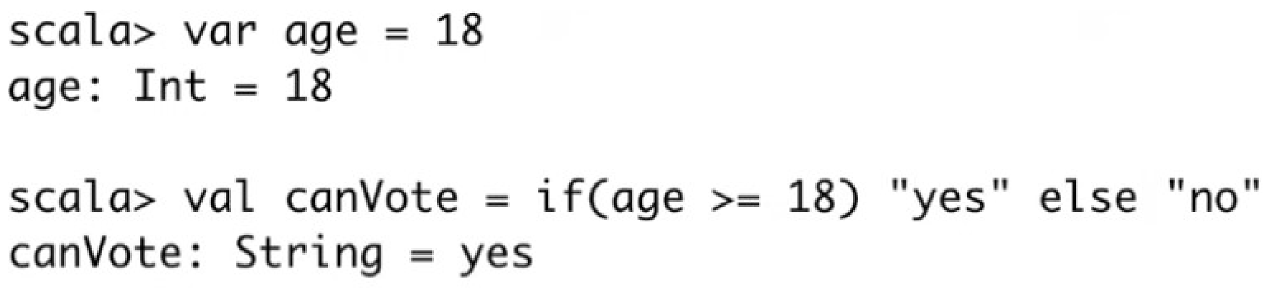


```
scala> import scala.math._
import scala.math._

scala> abs(-8)
res12: Int = 8
```

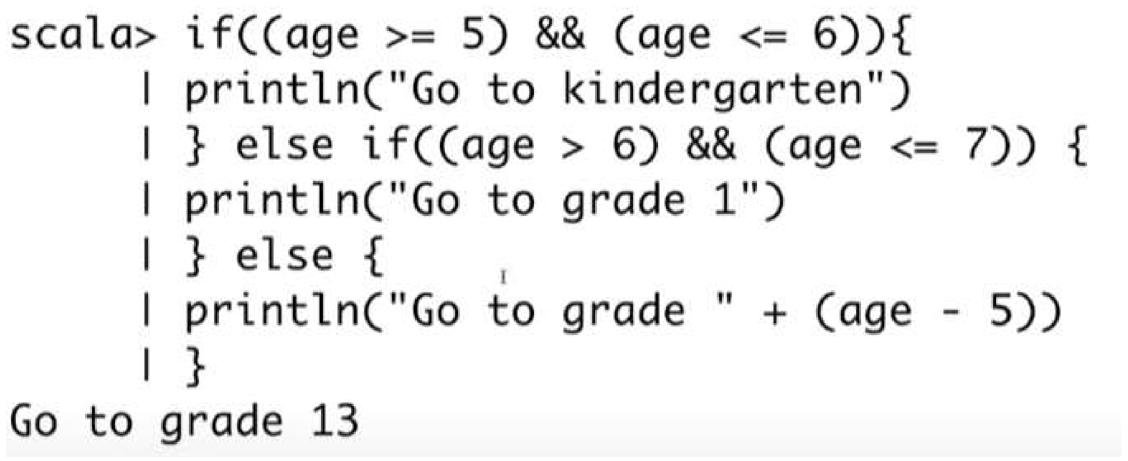


```
scala> "5 + 4 = " + (5 +4)
res5: String = 5 + 4 = 9

scala> "5 - 4 = " + (5 - 4)
res6: String = 5 - 4 = 1

scala> "5 * 4 = " + (5 * 4)
res7: String = 5 * 4 = 20

scala> "5 / 4 = " + (5 / 4)
res8: String = 5 / 4 = 1
```

Using conditional statements



```
scala> var age = 18
age: Int = 18

scala> val canVote = if(age >= 18) "yes" else "no"
canVote: String = yes
```



```
scala> if((age >= 5) && (age <= 6)){
     | println("Go to kindergarten")
     | } else if((age > 6) && (age <= 7)) {
     | println("Go to grade 1")
     | } else {
     | println("Go to grade " + (age - 5))
     | }
Go to grade 13
```

Running Scala Scripts

- A script is sequence of statements in a file, interpreted sequentially.

hello.scala:

```
println("Hello, World!")
```

```
$ scala hello.scala
Hello, World!
```

Compiling and Running Scala Code

$$\boxed{\texttt{Test.scala}} \rightarrow \texttt{scalac} \rightarrow \boxed{\texttt{Test.class}} \rightarrow \texttt{scala}$$

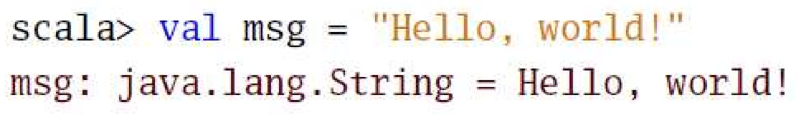
Hello.scala:

```
object Hello {
    def main(args: Array[String]) = println("Hello, World!");
}
```

```
$ scalac Hello.scala
$ ls
Hello$.class    Hello.class   Hello.scala
$ scala Hello
Hello, World!
```

Define some variables

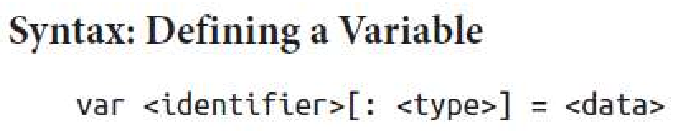
- Scala has two kinds of variables, vals and vars.
  - A val is like a final variable in Java. Once initialized, a val can never be reassigned.
  - A var, by contrast, is similar to a non-final variable in Java. A var can be reassigned throughout its lifetime.
- Example of val:



```
scala> val msg = "Hello, world!"
msg: java.lang.String = Hello, world!
```

Variable

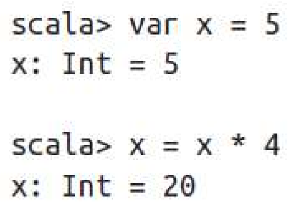
- The var keyword is used to define a variable with a given name, type, and assignment.



Syntax: Defining a Variable

```
var <identifier>[: <type>] = <data>
```

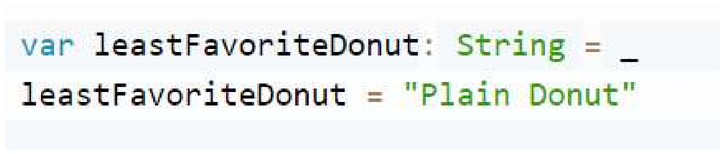
- If no type is specified, the Scala compiler will use type inference to determine the correct type to assign to your variable.



```
scala> var x = 5
x: Int = 5

scala> x = x * 4
x: Int = 20
```

Declare a variable with no initialization

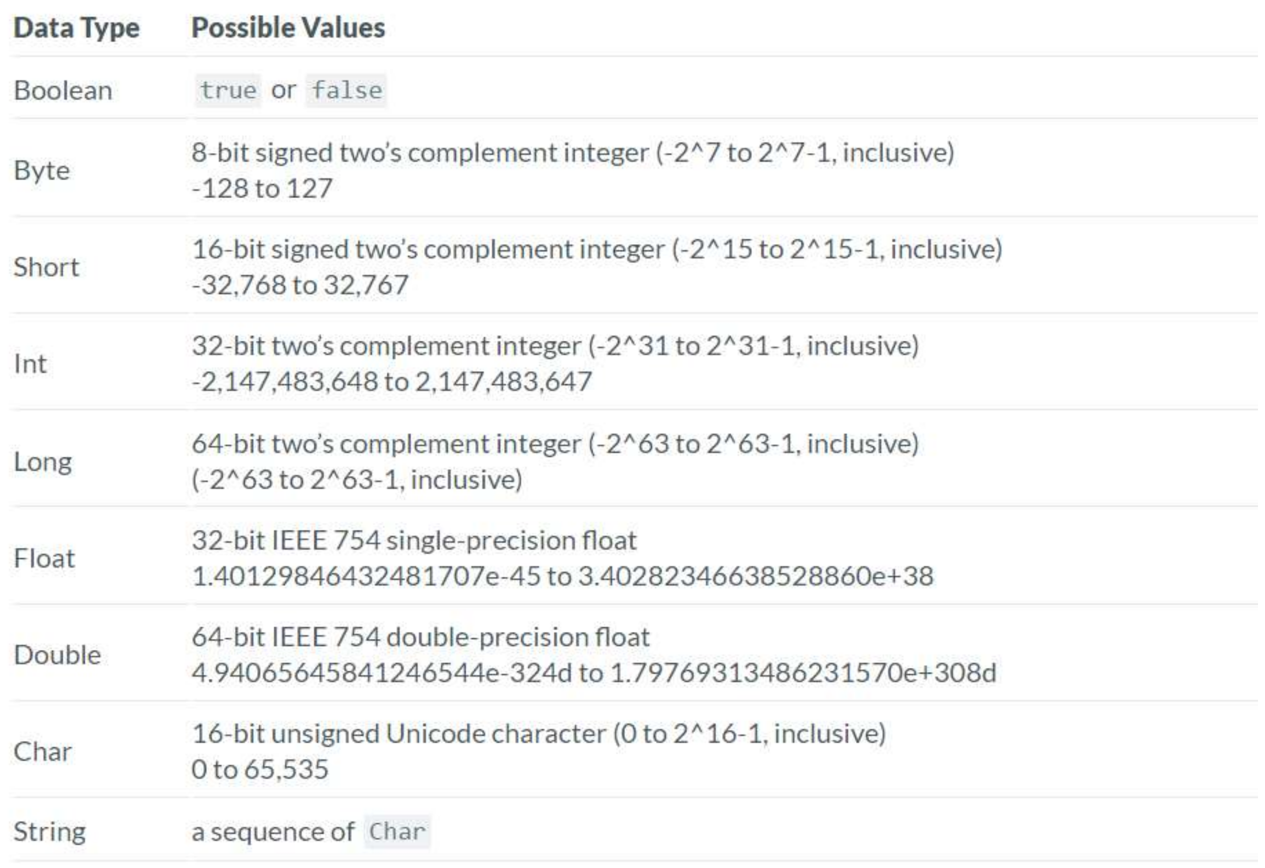
Sometimes you may not know the value of your variable immediately. You can only assign your variable's value at some later point in time during the execution of your application.



```
var leastFavoriteDonut: String = _
leastFavoriteDonut = "Plain Donut"
```

- We've used the wildcard operator _ when defining our variable.

Built in types



Data Type	Possible Values
Boolean	`true` or `false`
Byte	8-bit signed two's complement integer (-2^7 to 2^7-1, inclusive) -128 to 127
Short	16-bit signed two's complement integer (-2^15 to 2^15-1, inclusive) -32,768 to 32,767
Int	32-bit two's complement integer (-2^31 to 2^31-1, inclusive) -2,147,483,648 to 2,147,483,647
Long	64-bit two's complement integer (-2^63 to 2^63-1, inclusive) (-2^63 to 2^63-1, inclusive)
Float	32-bit IEEE 754 single-precision float 1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64-bit IEEE 754 double-precision float 4.94065645841246544e-324d to 1.79769313486231570e+308d
Char	16-bit unsigned Unicode character (0 to 2^16-1, inclusive) 0 to 65,535
String	a sequence of `Char`

Anonymous Functions

- The Scala language has anonymous functions, which are also called function literals. Scala being a functional language often means developers break down large problems into many small tasks and create many functions to solve these problems

```
val multiplyByTwo = (n:Int) => n * 2
def multiplyByThree = (n:Int) => n *3
```

```
multiplyByTwo(3)

//6

multiplyByThree(4)

//12
```

These methods are not limited to functions with arguments and can be used to instantiate methods that don’t take in any arguments.

```
val sayHello = ()=>{ println("hello") }
```

Anonymous Functions

- Inline functions - Using anonymous functions is a common pattern which is used pervasively in the collections library to perform quick actions over a collection.
- For instance, we have the filter method that takes an inline function / anonymous function to create another collection with only elements that meet the criteria we define in the anonymous function.

```
val myList = List(1,2,3,4,5,6,7)

val myEvenList = myList.filter((n: Int) => n % 2 == 0)
//List(2,4,6)

val myOddList = myList.filter((n:Int) => n % 2 != 0)
//List(1,3,5,7)
```

Anonymous Functions

- In Scala, it’s also possible to use wildcards where our anonymous function’s parameter aren’t named.

```
var timesTwo = (_:Int)*2

timesTwo(5)
//10
```

Lazy Evaluation

- In Scala, we have a keyword called lazy, which helps in dealing with values we don’t want to be evaluated until they’re referenced.
- A variable marked as lazy won’t be evaluated where it is defined, that’s commonly known as eager evaluation, it will only be evaluated when it’s referenced at some later in the code.

```
lazy val myExpensiveValue = expensiveComputation

def runMethod()={
    if(settings == true){
        use(myExpensiveValue)
    }else{
        use(otherValue)
    }
}
```

Type Inference

- In Scala, you don’t have to declare types for every variable you create. This is because the Scala compiler can do type inference on types based on evaluation of the right-hand side.

```
var first:String = "Hello, "
var second:String = "World"
var third = first + second
//the compile infers that third is of type String
```

Pattern Matching

- Scala has a powerful inbuilt mechanism for helping us check a whether a variable matches up to certain criteria, much like we would do in a switch statement in Java or in a series of if/else statements.

```
myItem match {
  case true => //do something
  case false => //do something else
  case  _ => //if none of the above do this by default
}
```

Classes and Objects

- Objects are the real-world entities, and class is a template that defines objects.
- Classes have both state and behaviors. The states are either values or variables. The behaviors are the methods in Scala.

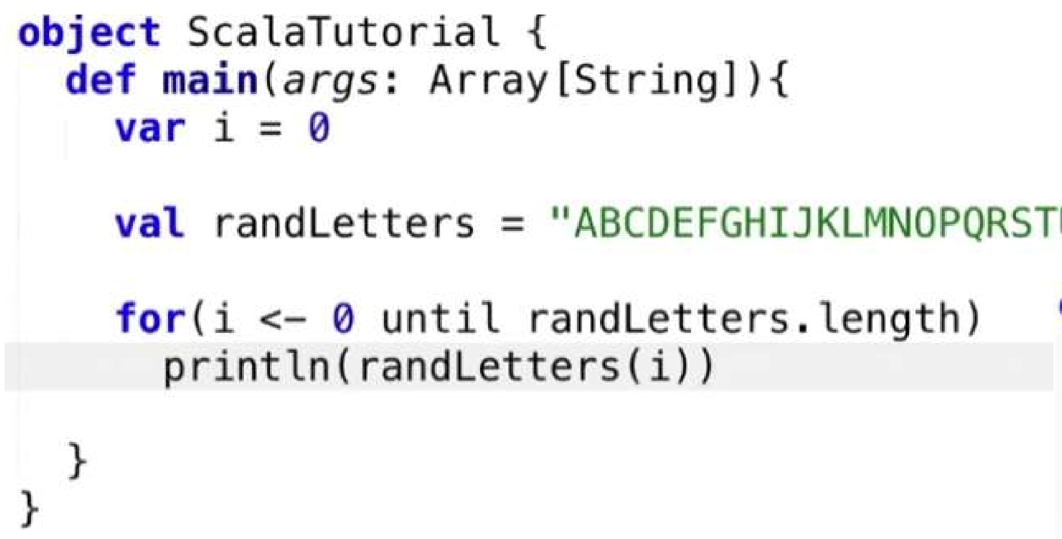
How to define class in Scala?

The class called Rectangle, which has two variables and two functions. You can also use the parameters l and b directly as fields in the program. You have an object which has a main method and has instantiated the class with two values.

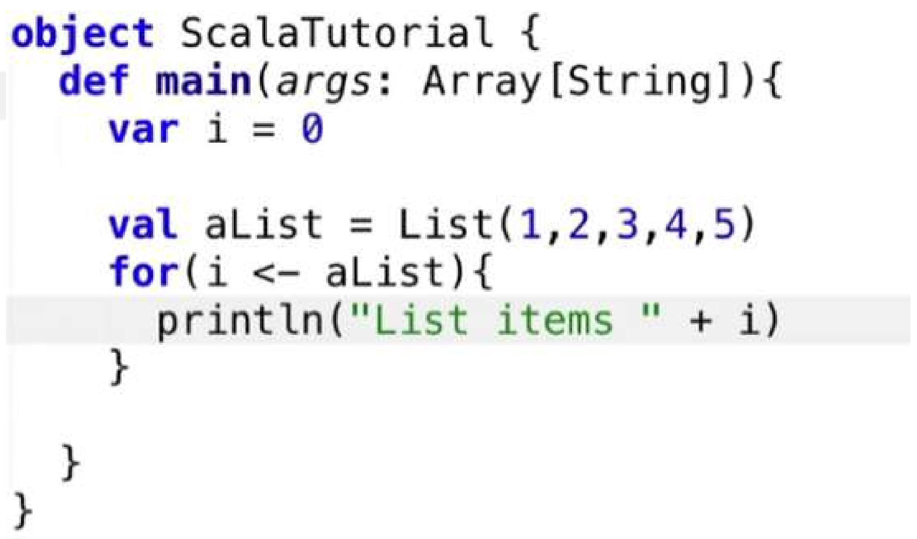
```
class Rectangle( l: Int,  b: Int) {
  val length: Int = l
  val breadth: Int = b
  def getArea: Int = l * b
  override def toString = s"This is rectangle with length as $length and breadth as
$breadth"
  }
object RectObject {
  def main(args: Array[String]) {
    val rect = new Rectangle(4, 5)
    println(rect.toString)
    println(rect.getArea)
  }
}
```

All the fields and method are by default public in Scala.
It is essential to use override because toString method is defined for Object in Scala.

Using for Loops

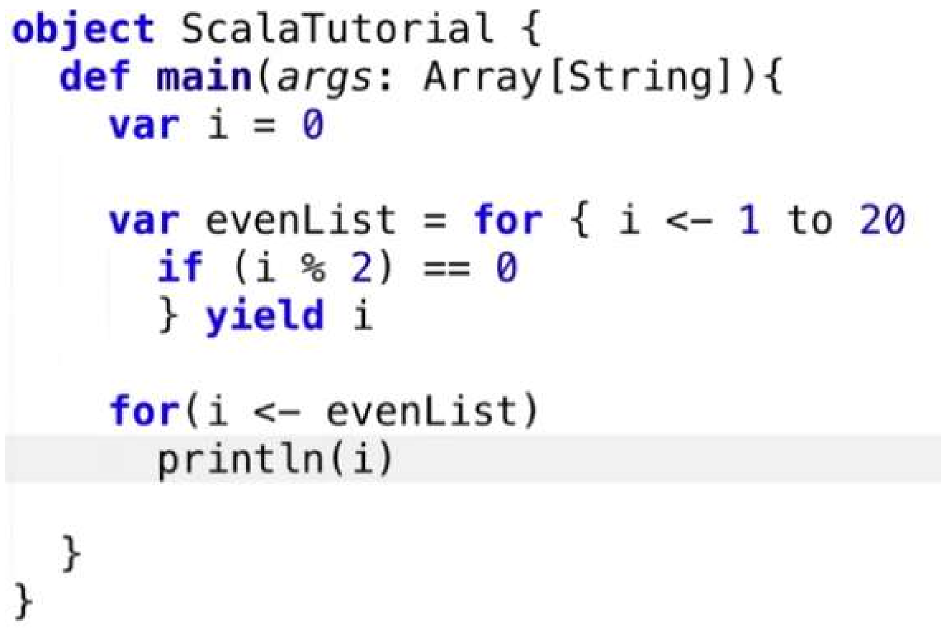


```scala
object ScalaTutorial {
  def main(args: Array[String]){
    var i = 0

    val randLetters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    for(i <- 0 until randLetters.length)
      println(randLetters(i))

  }
}
```

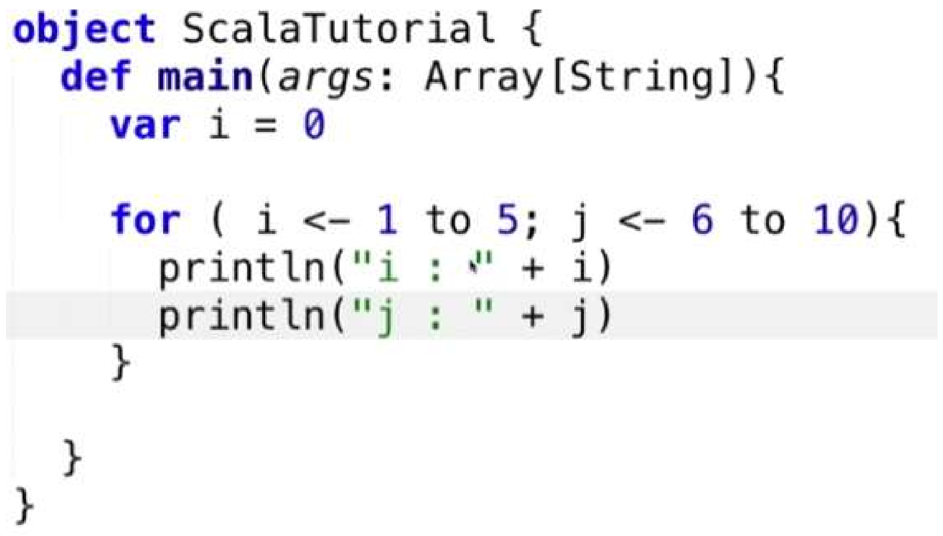


```scala
object ScalaTutorial {
  def main(args: Array[String]){
    var i = 0

    val aList = List(1,2,3,4,5)
    for(i <- aList){
      println("List items " + i)
    }

  }
}
```

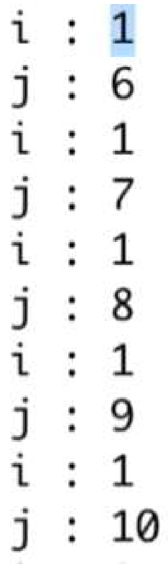
Using for Loops



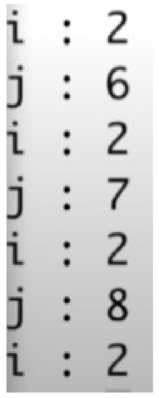
```
object ScalaTutorial {
  def main(args: Array[String]){
    var i = 0

    var evenList = for { i <- 1 to 20
      if (i % 2) == 0
      } yield i

    for(i <- evenList)
      println(i)

  }
}
```



```scala
object ScalaTutorial {
  def main(args: Array[String]){
    var i = 0

    for ( i <- 1 to 5; j <- 6 to 10){
      println("i : " + i)
      println("j : " + j)
    }

  }
}
```



```
i : 1
j : 6
i : 1
j : 7
i : 1
j : 8
i : 1
j : 9
i : 1
j : 10
```



```
i : 2
j : 6
i : 2
j : 7
i : 2
j : 8
i : 2
```

Command line I/O - Printing

- Like Java, write output to standard out (STDOUT) using println

```
println("Hello, world")
```

- if you don’t want to add a new line, just use print instead

```
print("Hello without newline")
```

- When needed, you can also write output to standard error (STDERR) like this:

```
System.err.println("yikes, an error happened")
```

Command line I/O - Reading

- The easiest way is to use the readLine method in the scala.io.StdIn package. To use it, you need to first import it, like this

```
import scala.io.StdIn.readLine
```

- Consider a file named HelloInteractive.scala

```
import scala.io.StdIn.readLine

object HelloInteractive extends App {

    print("Enter your first name: ")
    val firstName = readLine()

    print("Enter your last name: ")
    val lastName = readLine()

    println(s"Your name is $firstName $lastName")

}
```

```
$ scalac HelloInteractive.scala
```

```
$ scala HelloInteractive
Enter your first name: Alvin
Enter your last name: Alexander
Your name is Alvin Alexander
```

Next – Coding -II