Scala Tutorial

Defining a variable:

Var: the value of that variable can change
Val: the value of that variable CANNOT be changed

Comment:

// or /*...*/

Other data types: (All these data types are objects in Scala)

Byte
Boolean
Char
Short
Int
Long
Float
Double
BigInt: val largePrime = BigInt("622750757456495624956435640564305643053465 04")
BigDecimal

Import:

Import scala.math._ // this will import all functions from math
(random*n).toInt // random integer between 0(inclusive) to n(exclusive)
random*(11-1)   //random number between 1-10(both inclusive)

Print:

println("this will print a line")

Conditional Statements:

if(){} else if(){} else{}

Exit scala Terminal:

:q

Loop:

for (i <- 1 to 10) {} or for(i <- 1 until randString.length){}
while() {}
do{}while()
For array index/loop, use round brackets instead of square ones. Eg: println(randString(i))

List comprehension:

var evenList = for {i <- 1 to 20 if (i%2)==0 } yield i
Multiple for loops:
for (i <- 1 to 5; j <- 6 to 10) {} // this like j loop inside i loop

Break/continue:

There is no break/continue in Scala. You can use “return” instead of break.

List:

val primeList = List(1, 2, 3, 4)

Functions:

```
def funcName(param1:dataType, param2:dataType) : returnType = {
	Function body
	return valueToReturn
}
```

When you don’t return anything, you put “Unit”

```
def getSum(args: Int*): Int = {
	var sum: Int = 0
	for(num <- args){
		sum += num
	}
	sum
}
```

Array and ArrayBuffer:

```
val favNums = new Array[Int](20)
val friends = Array(“Bob”, “Tom”)
friends(0) = “Sue”
val friends2 = ArrayBuffer[String]()
friends.insert(0, “Phill”)
friends2 += “Mark”
friends2 ++= Array(“Susy”, “Paul”)
friends2.insert(1, “Mike”, “Sally”, “Sam”)
friends2.remove(1,2) // starting index and the number of items to remove
var friend : String = “ “
for (friend <- friends){
	println(friend)
}

for(j<-0 to (favNums.length -1)){
	favNums(j) = j
}

val favNumsTimes2 = for(num<-favNums) yield 2*num

favNumsTimes2.foreach(println)

var temp = for(num<-favNums if num%4==0) yield num

var multTable = Array.ofDim[Int](10,10)
for(i<- 0 to 9) {
	for(j<- 0 to 9){
		printf(“%d:%d\n”, i, j, multTable(i)(j))
```

```
        }
}

favNums.sum
favNums.min
favNums.max
favNums.sortWith(_>_) //desc
favNums.sortWith(_<_) //asce

sortedNums.deep.mkString(“,”)
```

Maps:

```
val employees = Map(“Manager”->”Bob”, “Secretary”->”Sue”) //immutable
if(employees.contains(“Manager”){
        employees(“Manager”)
}

val customers = collection.mutable.Map(100-> “Paul”) //mutable
customers(100)
customers(100) = “Tom” //changing values
customers(101) = “Sue” //adding values
for((k,v)-<customers)
        printf(“%d:%s\n”, k, v)
```

Tuples:

(normally immutable)

```
var tupleMarge = (102, “Marge”, 10.23)
printf(“%s owes us $%.2f\n”, tupleMarge._2, tupleMarge._3)
tupleMarge.productIterator.foreach{i=>println(i)} // prints all items in a separate line
tupleMarge.toString()
```

Classes:

Usually defined outside the main function but within the constructor of the program.
There are no static methods/variables in scala

```
object ScalaTutorial{
        def main(args: Array[String]){
```

```
        val rover = new Animal
        rover.setName(“Rover”)
        rover.setSound(“Woof”)
        printf(“%s says %s\n”, rover.getName, rover.getSound)

        val whiskers =  new Animal(“Whiskers”, “Meao”)
        println(s”${whiskers.getName} with id ${whiskers.id} says ${whiskers.getSound}”)

        println(whiskers.toString)

        val spike = new Dog(“Spike”, “Woof”, “GHrrr”)
        println(spike..toString)
} //end of main

class Animal(var name: String, var sound: String){
        this.setName(name)

        val id = Animal.newIdNum
//      protected var name = “No Name” //protected variables can be accessed only by
the class or the subclass

        def getName(): String = name
        def getSound(): String = sound

        def setName(name :  String){
                if(!(name.matches(“.*\\d+.*”))) //check if the variable contains only
non-numeric string
                        This.name = name
                else
                        This.name = “No Name”
        }

        def setSound(sound: String){
                this.sound = sound
        }

        def this(name: String){ //this is a constructor for this class in case it is called
without any specific arguments
                this(“No Name”, “No sound”)
                this.setName(name)
        }

        def this(){
                this(“No name”, “No Sound”)
        }

        override def toString() : String = { // function to override an existing function
                return “%s with the id %d says %s”.format(this.name, this.id, this.sound)
        }
}

// outside class
// create a companion object for the above class where you can get the static variables
and methods
object Animal { //it should have same name as that of the class
        private var idNumber = 0
        private def newIdNum = { idNumber += 1 ; idNumber}
        }
}
```

Inheritance

If you don’t want a class to be inherited, then declare it a “final”
// class final Animal

```
class Dog(name: String, sound: String, growl: String) extends Animal(name, sound){
        def this(name: String, sound: String){
                this(“No name”, sound, “No Growl”)
                this.setName(name)
        }

        def this(name: String){
                this(“No Name”, “No Sound”, “No growl”)
                this.setName(name)
        }

        def this(){
                this(“No Name”, “No Sound”, “No growl”)
        }

        // overriding methods from superclass
        override def toString(): String={
                return “%s with the id %d says %s or %s”.format(this.name, this.id, this.sound,
        this.growl)
        }
}
```

Eg of how to call dog class is defined in the main above

Abstract Classes:

```
abstract class Mammal(val name: String){
	var moveSpeed : Double

	def move: String
}

class Wolf(name: String) extends Mammal(name){
	var moveSpeed = 35.7

	def move = “The wolf %s runs %.2f mph”.format(this.name, this.moveSpeed)
}
```

In main function:

```
	val fang = new Wolf(“Fang”)
	fang.moveSpeed = 36.0
	println(fang.move)
```

Traits:

They are more like Java interface, in which a class can extend more than one class, except that we will be able to define concrete methods as well

```
trait Flyable{
	def fly: String
}
trait BulletProof{
	def hitByBullet : String

	def ricochet(startSpeed :  Double) : String = {
		“The bullet ricochets at a speed of %.1f ft/sec”.format(startSpeed * .75)
	}
}
class Superhero(val name: String) extends Flyable with BulletProof{
	def fly = “%s flys through the air”.format(this.name)
```

```
        def hitByBullet = “The bullet bounces off the %s”.format(this.name)
}
```

In main:

```
        val superman = new Superhero(“Superman”)
        println(superman.fly)
        println(superman.hitByBullet)
        println(superman.ricochet(2500))
```

Higher Order Functions:

Functions can be passed like the variables

```
val log10Func = log10 _       //underscore represent that we passed a function and not a variable
println(log10Func(1000))
List(1000.0, 10000.0).map(log10Func).foreach(println)
List(1, 2, 3, 4, 5).map((x: Int) => x*50).foreach(println)
List(1, 2, 3, 4, 5).filter(_ % 2 ==0).foreach(println)     // here underscore is to indicate each value
in the loop
```

How to pass different functions into a function:

```
val log10Func = log10 _
def times3(num: Int) =  num*3
def times4(num: Int) =  num*4

def mutl( func: (Int) => Double, num : Int) = {          //this function takes another function and an
                                                          int as the args. The function it uses, takes
                                                   int and returns Double
        func(num)
}
printf(“4*100= %.1f\n”, mult(times4, 100))
```

Closures:

It is a function that is dependent on a variable defined outside of the function

```
val divisorVal = 5
val divisor5 = (num: Double) => num/divisorVal
```

```
println(“5/5=” + divisor5(5.0))
```

File IO:

```
import  java.io.PrintWriter
import scala.io.Source

val writer = PrintWriter(“test.txt”)
writer.write(“Just some random text”)
writer.close()

val textFromFile = Source.fromFile(“test.txt”, “UTF-8”)
val lineIterator = textFromFile.getLines
for(line <- lineIterator)
        println(line)
textFromFile.close()
```

Exception Handling:

```
def divideNums(num1: Int, num2 = Int) = try
{
        (num1/num2)
} catch{
        case ex : java.lang.ArithmeticException => “can’t divide by 0”
} finally {
        // clean up after exception
}

println(“3/0=” + divideNums(3, 0))
```

CheatSheet:

http://goo.gl/O1CuGM