```scala
//------- Scala+reqT Crash Course "Getting started with Scala and reqT"
// ### Contents:
// ### Part 1: Scala crash course
// ### Part 2: Some reqT basics
// Next steps after this crash course:
//   check out docs at reqT.org and do the reqT Lab 1:
//   https://github.com/reqT/reqT/blob/3.0.x/doc/lab1/lab1.pdf
//Prerequisites: basic programming skills in Java or similar.

//########################## Part 1 Scala crash course ##########################
//start reqT with this command: java -jar /path/to/the/reqT.jar
//run the below statements in the reqT shell line by line

//declare integer variable:
var myVar: Int = 0  //corresponding Java:  int myVar = 0;

//type inference allow us to skip the type annotation:
var x = 0

//assignment:
x = x + 1
x += 1
x -= 10
println(x)

//declare a constant:
val k = 20
k = k + 1 //Compile error:reassignment to val

//declare a function
def inc(x: Int):Int = x + 1

//type inference allow us to skip the return type
def inc(x: Int) = x + 1

//function call:
inc(41)

//create a Vector:
val xs = Vector(5,6,7,8)

//map inc over all elements and make a new Vector:
val ys = xs.map(inc)

//collect some specific elements in a new vector:
val zs = xs.collect{case x if x > 6 => x}

//for loop:
for (i <- 0 to 2) { println(xs(i)) }

//same as above:
(0 to 2).foreach(i => println(xs(i)))

//shorter but same as above:
xs.take(3).foreach(println)

```

```
//for comprehension:
val incxs = for (i <- 0 to 2) yield xs(i) + 1

//create a singelton object (exactly one instance, no new)
object obj { def dec(x: Int) = x - 1 }

//dot notation:
obj.dec(41)

//import all public members of an object:
import obj._
dec(43)

//functions are actually objects with apply method(s):
object inc2 { def apply(x: Int) = x + 1}
inc2.apply(41)
inc2(41)  //the complier injects the .apply method call

//Every value is an object:
41 + 1  //this is actually simplified dot notation
41.+(1)  //41 is an object
inc2 apply 41  //operator notation on object inc2

//declare a class with a default constructor:
class Banana(gram: Int) {
  def kilo = gram / 1000.0
}

//create an object and store the reference in a constant:
val b1 = new Banana(420)

//gram is private:
b1.gram //Compile error:value gram is not a member of Banana

//methods are public by default:
b1.kilo

//if you add val before class parameters then they are public fields:
class Banana(val gram: Int) {
  def kilo = gram / 1000.0
}
val b2 = new Banana(399)
b2.gram
b2.kilo

//create a companion object with apply factory using :paste
class Banana(val gram: Int) {
  def kilo = gram / 1000.0
  override def toString = s"Banana($kilo) // in kilograms"
}
object Banana { //same name as class in same code file
  def apply(kilo: Double) = new Banana((kilo*1000).toInt)
}
val b3 = Banana(0.333333)

//create a case class:
```

```scala
case class Orange(gram: Int)
//by adding 'case' in front of 'class' you get all these goodies for free:
//  * object with apply factory; no need for new
//  * a nice toString of all class parameters
//  * class parameters become public val fields
//  * an equals method implementing structural equality over class params with ==
//  * a hash code making objects hash well in e.g. HashMap and Set collections
//  * an unapply method to enable pattern matching
val o1 = Orange(123)
Orange(123) == Orange(123) //structural equality
Vector(Orange(123), Orange(234)).map{case Orange(g) => g} //pattern matching on Orange

//operator method
case class Apple(val gram: Int) {
  def +(that: Orange) = Vector(this, that)
}
Apple(111) + Orange(222)

//Scala raw strings
"""This string has "quotes" in it without escape chars."""

//Scala string interpolator s
val it = 42
println(s"This is it: $it")
println(s"This is almost it: ${it-1}")


//########################## Part 2 Some reqT basics ##########################
//run the below statements in the reqT shell line by line

//reqT includes a requirements DSL embedded in scala
//implemented using scala case classes
Feature("x")
Stakeholder("a")
Stakeholder("a").requires(Feature("x"))

//reqT has a special collection called Model
//Model can contain elements of 3 kinds:
//1. Entities each having its own id:
Model(Stakeholder("a"))
//2. Attributes each holding some value:
Model(Prio(42))
//3. Relations:
Moodel(Feature("x") has Prio(42))

//Model is actually a tree-like data structure:
val m = Model(
  Stakeholder("a") has (
    Feature("x") has Prio(41),
    Feature("y") has Prio(42)),
  Stakeholder("b") has (
    Feature("x") has Prio(99),
    Feature("y") has Prio(1)))

//You can access parts of a Model tree with paths:
m/Stakeholder("b").has
```

```scala
m/Stakeholder("b").has/Feature("x").has/Prio

//Models are immutable, each operation result in a new Model
m + Goal("profit")
var m2 = m + Product("cool")
m2 = m2 - Stakeholder("a")
m2.pp //pretty-print m2
m2.p  //print m2 in indented textual format "textified model"

//the reqT metamodel
reqT.metamodel.//press <TAB>
reqT.metamodel.entityTypes

//the reqT DSL is metaprogrammed...
//  the scala-embedded DSL case classes are generated from this Model:
reqT.meta.model.pp
reqT.meta.model.p

//The base classes of the requirements DSL metamodel:
//https://github.com/reqT/reqT/blob/3.0.x/doc/metamodel-simple.pdf
//Models can be converted to a Vector of elements:
m.toVector

//A Vector of elements can be converted to a Model:
Vector(Feature("x") has Prio(1), Stakeholder("a")).toModel
//How is that possible when Vector is part of the Scala libs???
//Use implicit classes to "pimp" existing classes with new methods:
implicit class StringPimper(s: String) {
  def toCoolString = s + " is cool!"
}
"Scala".toCoolString

//reqT has a gui with a tree-viewer and a text-editor
edit
//the editor can run scala scripts and much more:
//syntax coloring
//auto completion
//export and import
//etc.

//Run scripts using reqt:
//Put this text in a file called
//my-reqt-script.scala

val m1 = Model(Req("hello") has Spec("Print hello world"))
val m2 = m1.transform{case Req(id) => Req(id.reverse)}
println(m2)
println("""Model(Req("hejsan"))""".toModel)
sys.exit //exit reqT shell

//run the above script file using the -i option to reqT:
//java -jar /Path/to/the/reqT.jar -i my-reqt-script.scala

//next step: do reqT Lab 1
//https://github.com/reqT/reqT/blob/3.0.x/doc/lab1/lab1.pdf

```