## Working with Scala REPL - basic

1. Quickly review the code under Recipe.scala and Cookbook.scala in the ***cookbook.basic*** package to understand what we are working with.

2. Make sure you include all *compiled classes* (generally located under bin) to make them available to the REPL. You can do this by starting the REPL this way: ***scala -cp bin***

```
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ pwd
/Users/sanjoypinto/Documents/scala-workspace/SparkScala
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ ls
bin     src
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ ls src
com             cookbook
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ ls bin
com             cookbook
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ ls bin/cookbook/
Cookbook.class          Demo.class              README.doc
Demo$.class             Instructions.pages      Recipe.class
Sanjoys-MacBook-Pro:SparkScala sanjoypinto$ scala -cp bin
Welcome to Scala version 2.11.6 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_25).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

You can also build a jar file with your classes and provide it instead if you'd like.

3. Quickly review the various command available via the Scala REPL, by using ***:help***.

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:edit <id>|<line>          edit history
:help [command]            print this summary or command-specific help
:history [num]             show the history (optional num is commands to show)
:h? <string>               search the history
:imports [name name ...]   show import history, identifying sources of names
:implicits [-v]            show the implicits in scope
:javap <path|class>        disassemble a file or class name
:line <id>|<line>          place line(s) at the end of history
:load <path>               interpret lines in a file
:paste [-raw] [path]       enter paste mode or paste a file
:power                     enable power user mode
:quit                      exit the interpreter
:replay [options]          reset the repl and replay all previous commands
:require <path>            add a jar to the classpath
:reset [options]           reset the repl to its initial state, forgetting all session entries
:save <path>               save replayable session to a file
:sh <command line>         run a shell command (result is implicitly => List[String])
:settings <options>        update compiler options, if possible; see reset
:silent                    disable/enable automatic printing of results
:type [-v] <expr>          display the type of an expression without evaluating it
:kind [-v] <expr>          display the kind of expression's type
:warnings                  show the suppressed warnings from the most recent line which had any

scala>
```

4. Now we have 2 classes to import (any one way is fine):

```
scala> import cookbook.{Recipe,Cookbook}
import cookbook.{Recipe, Cookbook}

scala> import cookbook._
import cookbook._

scala> _
```

(screen shot may be off, give full package path to class file to be able to import)

_ is a wilcard import - it imports everything from the package that is specified.

5. Now Let's create a Recipe for peanut butter and jelly sandwiches.

```
scala> val cookbook = new Cookbook
cookbook: cookbook.Cookbook = cookbook.Cookbook@2b05039f

scala> val recipe = new Recipe
recipe: cookbook.Recipe = cookbook.Recipe@5fdef03a
```

6. Let's populate the map.

```
scala> cookbook.recipes = Map("p.b. & j." -> recipe)
cookbook.recipes: Map[String,cookbook.Recipe] = Map(p.b. & j. -> cookbook.Recipe@5fdef03a)

scala> recipe.ingredients = List("peanut butter, jelly, bread")
recipe.ingredients: List[String] = List(peanut butter, jelly, bread)

scala> recipe.directions = List("put peanut butter and jelly on the bread")
recipe.directions: List[String] = List(put peanut butter and jelly on the bread)
```

7. Check if the values have saved properly.

```
scala> cookbook.recipes("p.b. & j.")
res1: cookbook.Recipe = cookbook.Recipe@5fdef03a

scala> cookbook.recipes("p.b. & j.").ingredients
res2: List[String] = List(peanut butter, jelly, bread)

scala> cookbook.recipes("p.b. & j.").directions
res3: List[String] = List(put peanut butter and jelly on the bread)
```

The process of creating and populating these objects is cumbersome - had to instantiate the objects in one line and then populate each field on subsequent lines. Typically what we do with with Java is use constructors, so that's the next thing we'll look at.

**Working with Scala REPL - getting better, following convention**

1. Quickly review Cookbook.scala and Recipe.scala in the ***cookbook.better*** package to understand what we are working with.
2. Start the console again and import the classes. You need to provide full path including packages to the class to import. So, let's define a recipe

```
scala> import cookbook.better._
import cookbook.better._

scala> val recipe = new Recipe(
     | List("peanut butter", "jelly", "bread"),
     | List("put peanut butter and jelly on the bread")
     | )
Ingredients: List(peanut butter, jelly, bread)
Directions: List(put peanut butter and jelly on the bread)
recipe: cookbook.better.Recipe = cookbook.better.Recipe@7dc3712

scala> _
```

3. Without arguments: (we can say new Recipe with no ingredients and no directions)

```
scala> val recipe = new Recipe()
Ingredients: List()
Directions: List()
recipe: cookbook.better.Recipe = cookbook.better.Recipe@294425a7

scala> _
```

4. Only with one or the other argument:

```
scala> new Recipe(List("peanut butter", "jelly", "bread"))
Ingredients: List(peanut butter, jelly, bread)
Directions: List()
res0: cookbook.better.Recipe = cookbook.better.Recipe@488d1cd7

scala> new Recipe(directions = List("make a sandwich"))
Ingredients: List()
Directions: List(make a sandwich)
res1: cookbook.better.Recipe = cookbook.better.Recipe@56276db8

scala> _
```

5. Without defining a variable name

```
scala> new Recipe
Ingredients: List()
Directions: List()
res1: cookbook.better.Recipe = cookbook.better.Recipe@d2de489
```

**Working with Scala REPL - Singleton Object or Companion Object**

1. Scala classes cannot have static variables or methods. Instead a Scala class can have what is called a singleton object, or sometime a companion object.
2. Review Cookbook.scala and Recipe.scala from the ***cookbook.singleton*** package to understand what we are doing here.
3. Now that we have declared Cookbook as a singleton Object, we can refer to it directly.

```
scala> import cookbook.singleton._
import cookbook.singleton._

scala> Cookbook
Ingredients: List(peanut butter, jelly, bread)
Directions: List(put the peanut butter and jelly on the bread)
Ingredients: List(blueberries, panckaes)
Directions: List(take some blueberries and put it in a pancake)
res0: cookbook.singleton.Cookbook.type = cookbook.singleton.Cookbook$@18769467

scala> _
```

Something interesting has happened - when I refer to Cookbook for the 1st time, it's fields have been initialized as defined, and due to println statements in Recipe, we see the output on the screen of those initializations.

4. So what happens when you refer to the object Cookbook for a second time? … and a few more times? :)

```
res0: cookbook.singleton.Cookbook.type = cookbook.singleton.Cookbook$@18769467

scala> Cookbook
res1: cookbook.singleton.Cookbook.type = cookbook.singleton.Cookbook$@18769467

scala> Cookbook
res2: cookbook.singleton.Cookbook.type = cookbook.singleton.Cookbook$@18769467

scala>
```

First time its initialized, Second time onward it is referred, though creating a new result set each time. Note the same memory location the val.

5. To refer to the elements of Cookbook:

```
scala> Cookbook.pbj
res3: cookbook.singleton.Recipe = cookbook.singleton.Recipe@b065c63

scala> Cookbook.pbj.ingredients
res4: List[String] = List(peanut butter, jelly, bread)

scala> Cookbook.bbp
res5: cookbook.singleton.Recipe = cookbook.singleton.Recipe@1a4927d6

scala> Cookbook.bbp.ingredients
res6: List[String] = List(blueberries, panckaes)
```

6. Check what type Cookbook is.

```
scala> :t Cookbook
cookbook.singleton.Cookbook.type

scala> _
```

… ***<object>.type*** which is something called a ***Singleton type***.

So, really when we talk about Singleton Objects - it is an Object you can refer to directly by it's name Cookbook and you can access it's fields