Laboratory: Scala (2)

Summary: We continue our exploration of the Scala programming language, focusing on its collections and some of its functional aspects.

Prerequisites: The first Scala lab.. Tate, Section 5.3.

Contents:

- Preparation
- Exercises
  - Exercise 1: Fun with Sums
  - Exercise 2: Function Composition
  - Exercise 3: Fun with Maps
  - Exercise 4: Ordering in Maps
  - Exercise 5: Fun with Lists and Folds
  - Exercise 6: Expanding TLAs
- For Those with Extra Time

Preparation

a. Create a directory for the lab.

b. Open a browser window on Tate’s examples, in case you want to try any of them.

Exercises

Exercise 1: Fun with Sums

a. Consider the following function that sums a pair of integers (taken as a pair).

```scala
def sump(p: (Int,Int)): Int = p._1 + p._2
```

Verify that the function works as one would expect.

b. Consider the following function that builds a pair.

```scala
def pair(x: Any): (Any,Any) = (x,x)
```

Verify that it works as you might expect.

c. What do you expect the result of the following to be?
d. Check your answer experimentally.

e. As you may have discovered, we get a type error because `pair` returns a pair of “Any” values, rather than a pair in which each element is the same type as the parameter. Fortunately, scala lets us add type parameters to functions by putting them in brackets after the function name.

```scala
def pair[A](x: A): (A, A) = (x, x)
```

Verify that this solves the problem you just encountered.

**Exercise 2: Function Composition**

As you may recall, the composition operator is useful for higher-order programming. Write that operator.

Test it with combinations of the following functions

```scala
def len(x: scala.runtime.RichString) = x.length
def rev(x: scala.runtime.RichString) = x.reverse
def inc(x: Int) = x + 1
def sqr(x: Int) = x * x
def one(x: Any) = 1
```

**Exercise 3: Fun with Maps**

As you may recall, the Map type in scala provides immutable maps (tables, hashes, dictionaries, whatever you want to call the ADT).

```scala
scala> var x = Map("A" -> 1)
```

a. Repeat the definition above.

b. What do you expect the effect of each of the following to be?

```scala
scala> x("A")
scala> x("B")
```

c. Check your answer experimentally.

d. What do you expect the result of the following to be?

```scala
scala> var y = Map("A" -> 1) + ("B" -> 2)
```

e. Check your answer experimentally.

f. What do you expect the result of the following to be?
scala> x += ("B" -> 2)

g. Check your answer experimentally.

h. Explain the difference between the two results.

i. What do you expect the result of the following to be?
scala> x += ("B" -> 3)

j. Check your answer experimentally.

k. What do you expect the result of the following to be?
scala> x += (3 -> "B")

l. Check your answer experimentally.

m. Figure out how to create an empty Map that lets you use any kind of index.

**Exercise 4: Ordering in Maps**

As you know from implementing hash tables, in the most common implementation, when the hash table gains too many elements, we build a new table and rehash all of the elements.

Determine experimentally when (or if) that reordering happens in a Scala Map.

**Exercise 5: Fun with Lists and Folds**

*These problems are based on exercises in Tate.*

a. Using `foldLeft`, find the total length of all the strings in a list of strings.

b. Using `foldLeft`, find a longest string in a list of strings.

c. Using `foldLeft`, find the alphabetically first string in a list of strings.

**Exercise 6: Expanding TLAs**

*This problem is based on an exercise from Tate.*

Write a TLA trait for strings that expands some common three-letter-acronyms. E.g., "TLA" -> "Three Letter Acronym", "DEC" -> "Digital Equipment Corporation", "FOO" -> "Fun Object Orientation".
For Those with Extra Time

If you find yourself with extra time, start working on Assignment 5.