Scala Lab 1: Answer Sheet

CSP344/CS-205 | *by* Keoni D'Souza

Note: type annotations are usually optional.

Part I/ Writing and reading in Scala

I.I/ Writing a simple function in Scala

I.I.a/ The square function

Task 2: Copy the below and run it. Check that what you believe to be the right value is returned:

```
object scalaLab1 extends App {
  def square(n: Int): Int = n * n
  println(square(2))
}
```

Suggested answer: `4` should be returned.

I.I.b/ +3 function

Task 3: Write a function `add3` that takes in a number and adds `3`.

Suggested answer:

```
def add3(x: Int): Int = x + 3
```

I.I.c/ Returning even numbers

Task 4: Write a function `isEven` that takes in an integer and returns a corresponding `Boolean`.

Suggested answer:

```
def isEven(e: Int): Boolean = e % 2 == 0
```

I.I.d/ Combining functions

Task 5: In a new function `squareAnd3`, write something that takes in an integer, squares it, adds `3`, then returns the result.

Suggested answer:

```
def squareAnd3(a: Int): Int = add3(square(a))
```

I.II/ Reading user input

Task 6: Write a function called `greet()` that uses `readLine()` to take in a name and produces a message not unlike the following:

```
Congratulations, [name] – you have been called to learn Scala!
```

Suggested answer:

```
def greet(): Unit = {
  val name = readLine("Please enter your name: ")
  println("Congratulations, " + name + " – you have been called
to learn Scala!")
}
```

Part II/ Lists and higher order functions in Scala

II.I/ Lists in Scala

II.I.a/ Writing a list

Task 7: Create a variable (`var`) called `doorClosers` that creates a new instance of `ListBuffer` that will take in elements of type `String`.

Suggested answer:

```
import scala.collection.mutable.ListBuffer

var doorClosers = new ListBuffer[String]()
```

N.B. Type annotation is required here.

Task 8: Using `++=`, add all the types of door closer above to `doorClosers`.

Suggested answer:

```
doorClosers ++= List("Automatic",
                     "Concealed",
                     "Overhead",
                     "Slide arm",
                     "Transom",
                     "Electromagnetic",
                     "Floor spring",
                     "Emergency")
```

II.I.b/ Printing from a list

Task 9: Have a look at the below function:

```
def questionable(a: List[Any]) = {
  var c = 0
  a.foreach(_ => c += 1)
  if (c == 0) a
  else a(0)
}
```

Which built-in function does it emulate?

Run the third remaining built-in function on `doorClosers` and print the result.

Suggested answer:

It emulates the `head` function. The remaining function is `tail`, evidenced below:

```
println(doorClosers.tail)
```

II.I.c/ Updating a list

Task 10: In a shocking turn of events, I reason that *automatic* is less a type of door closer and more of a classification. To remedy this, use `-=` to get rid of it.

Suggested answer:

```
doorClosers -= "Automatic"
```

Task 11: I realise similar can be said for *emergency*. This time, with postfix notation, use `remove()` to delete it.

Suggested answer:

```
doorClosers remove 6
```

Task 12: Finally, the fragment

```
var doorClosersList = doorClosers.toList
```

provides a copy of `doorClosers` in list-form. Try removing another element. Is it possible? Why/why not? How could the definition be improved?

Suggested answer:

It is not possible to remove another element. You should receive an error message detailing that `-=` or `remove` is `not a member of List[String]`. List is an immutable data structure, so its contents can't be changed, thus it would be better to use `val` instead of `var` as this is made redundant.

II.II/ Higher order functions in Scala

II.II.a/ The `map` operation

Task 13: In a `val` called `lengths`, use `map` to create a list that corresponds to the lengths of the names in `doorClosersList`.

Suggested answer:

```
val lengths: List[Int] = doorClosersList.map(_.length)
```

II.II.b/ The filter operation

Task 14: Write a function `lessThanX` that takes in a list of strings and a length `x`. It should return the items that have a length less than the specified integer.

Suggested answer:

```
def lessThanX(list: List[String], x: Int): List[String] =
  list.filter(_.length < x)
```

Part III/ Interacting Scala with Java

Task 16: In a new file, `BankAccount.scala`, define an `object` called `BankAccount` and write a Scala function `withdrawFunds` that takes in an account and an amount and returns a message that indicates whether the withdrawal has been successful. Indeed, if so, the money should be deducted from the account.

Task 17: Write a Scala function `applyInterest` to apply interest to an account's balance. For the time being, use an integer modifier. You can add a message indicating that interest has been applied, if you so wish.

Suggested answer (for tasks 17 and 18):

```
object BankAccount extends App {

  def withdrawFunds(account: BankAccountJava,
                    amount: Int): String = {
    if (amount > account.balance)
      return "You don't have sufficient funds."
    account.balance -= amount
    "%d withdrawn".format(amount)
  }

  def applyInterest(account: BankAccountJava,
                    interest: Int): Unit = {
    account.balance *= interest
    println("%dx interest rate applied".format(interest))
  }

  // Testing
  var a: BankAccountJava = new BankAccountJava("Jens", 15000)
  println(withdrawFunds(a, 10000))
  a.statement()
  applyInterest(a, 2)
  a.statement()
}
```