Scala Training

eScala Book

Book - Programming Scala - second edition: \\bk-agfil-1001\IT\eBooks\Programming_Scala_Second_Edition.pdf

- Dev: Everything up to (and including) Chapter 12, but not Chapter 6 pages 199-206 and pages 227-242, the whole Chapter 7 pages 243-266, Chapter 8 pages 281-294. (Total of around 308 pages)
- QA: Everything up to page 98

The Scala 10 commandments:

1. Never use "null" (use Option instead)
2. Avoid 'for' loops, just use maps instead
3. Case classes should be used as data containers and only as data containers
4. Avoid "var" as much as possible (use "val" instead)
5. Prefer immutable types over mutable types
6. Avoid explicit casting and type checking, use pattern matching instead
7. Companion objects are should be used if they are beneficial to the class they are companioned with
8. Data dependencies should be resolved with laziness
9. Implicit arguments should be of specialized types
10. It's better to use classes than objects

Testing in Scala

Book - Testing in Scala: \\bk-agfil-1001\IT\eBooks\Testing_in_Scala.pdf

- Read chapters 1-3 and "ScalaMock" under chapter 5

Scala config

read https://github.com/typesafehub/config up until (no included) miscellaneous-notes

Scala Exercises

All exercises should have tests - use FunSuite

Please review all exercises with Rouz or Idan

Q1. Use a trait to define a generic queue of strings with 'put' and 'get' methods, and create a class that implements it using an array. Include tests.

Q2. Starting from the previous exercise, use a trait to modify the behavior of 'put' so it reverses each string before adding it to the queue. Include tests.

Q3. Write the ~= operator for comparing doubles. The operator should return true iff 2 doubles are equal up to a small constant. The constant should be configurable but also have a default. Include tests.

Q4. Implement GCD in Scala (hint: use pattern matching and tail recursion)

Q5. Write a clause that measures the run time of a block of code and prints it (also needs to return the original output of the block):

```
timeit {
        ...
        ...
}
```

Q6. Given a string containing words separated by space, find:

- The longest word

- The most common word
- The most common letter
- Create a map from letter to a set of words it appear in

Q7. Convert a list of strings to a list of all the characters in all the strings

Q8. Given the following code:

```
trait IntSet {
        def incl(x: Int): IntSet
        def contains(x: Int): Boolean
}
class EmptySet extends IntSet {
        def contains(x: Int): Boolean = false
        def incl(x: Int): IntSet = new NonEmptySet(x, new EmptySet, new EmptySet)
}
class NonEmptySet(elem: Int, left: IntSet, right: IntSet) extends IntSet {
        def contains(x: Int): Boolean =
                if (x < elem) left contains x
                else if (x > elem) right contains x
                else true
        def incl(x: Int): IntSet =
                if (x < elem) new NonEmptySet(elem, left incl x, right)
                else if (x > elem) new NonEmptySet(elem, left, right incl x)
                else this
}
```

Write methods union and intersection to form the union and intersection between two sets. Add a method

```
def excl(x: Int)
```

to return the given set without the element x. To accomplish this, it is useful to also implement a test method

```
def isEmpty: Boolean
```

for sets.

Q9. Consider the following definitions representing trees of integers. These definitions can be seen as an alternative representation of IntSet:

```
abstract class IntTree
case object EmptyTree extends IntTree
case class Node(elem: Int, left: IntTree, right: IntTree) extends IntTree
```

Complete the following implementations of function contains and insert for IntTree's.

```
def contains(t: IntTree, v: Int): Boolean = t match { ...
      ...
}
def insert(t: IntTree, v: Int): IntTree = t match { ...
      ...
}
```

Q10. Consider a function which squares all elements of a list and returns a list with the results. Complete the following two equivalent definitions of squareList.

```
def squareList(xs: List[Int]): List[Int] = xs match {
       case List() => ??
       case y :: ys => ??
 }
 def squareMapList(xs: List[Int]): List[Int] =
       xs map ??
```

Q11. Write a function that gets an optional x,y and z and returns the first that is not None

Q12. Given a list:List[Int] and map:Map[Int, Double], multiply all the numbers in the list with their corresponding value in the map, and drop if don't exists

for example: list = [1,2,3,4], map = {1 -> 3, 3-> 5} ==> res = [3.0, 15.0]

Q13. Write a retry method that converts a method to a retry-able method.

the syntax should look like:

```
retry { ... }
```

However, you also need some way to specify how many times to retry and support sleep between retries (hint: implicit arguments)

Q14. Design a class that is given a vector of numbers in the constructor and exposes:

- x: a vector with the square of all elements in the input vector
- y: the sum of x
- z: the square root of y

nothing should be calculated in the constructor of the class assume the calculation of x,y,z can take a lot of time, and should only be done once (at most)

Q15. Add a method "median" to a Seq of integers so that s.median is the median of s for s of type Seq[Int]

- How can you add the same method for a sequence of doubles with minimal code duplication?