Scala Worksheet 1

Please install Scala on your work-machine: You should have Scala up and running after this week, and also have an IDE that allows you to access the Scala REPL. Some installation instructions are given at

https://www.scala-lang.org/download/2.13.10.html

MacOSX

0) (if needed) `brew install java` or `brew reinstall java`

1) `brew install scala@2.13`

Another method to install Scala on MacOSX

1) `curl -s "https://get.sdkman.io" | bash`

2) `sdk install scala 2.13.10`

Windows / Linux Ubuntu

0) (if needed) `sudo apt-get remove scala-library scala`

1) `sudo wget https://downloads.lightbend.com/scala/2.13.10/scala-2.13.10.deb`

2) `sudo dpkg -i scala-2.13.10.deb`

Other Linux distros: `sudo apt-get scala`

In the end you should have something running like in your terminal/shell:

```
$ scala
Welcome to Scala 2.13.10 (OpenJDK 64-Bit Server VM, Java 17.0.1).
Type in expressions for evaluation. Or try :help.

scala>
```

If you want to use VS Code or Codium, you might want to also look into setting up a key-shortcut for "Run-Selected-Text-In-Active-Terminal" and also set up syntax highlighting. Have a look at the Scala handout uploaded to KEATS for information.

Make sure you have checked out the template files for the Scala CWs and registered the Github repository on KEATS. See whether you receive a test report when pushing some code.

Task 1

'Play' with the Scala REPL and try out the following queries. Observe what Scala responds. Does it make sense?

```
scala> 2 + 2
scala> 1 / 2
scala> 1.0 / 2
scala> 1 / 2.0
scala> 1 / 0
scala> 1.0 / 0.0
scala> true == false
scala> true && false
scala> 1 > 1.0
scala> "12345".length
scala> List(1,2,1).size
scala> Set(1,2,1).size
scala> List(1) == List(1)
scala> Set(1,2,3) == Set(3,2,1)
scala> Array(1) == Array(1)
scala> Array(1).sameElements(Array(1))
```

Task 2 (Vals)

Type in the value declarations below and observe Scala's responses. Explain what is printed as the value for `x` in the last line. Is `z` re-assigned in the 4th line?

```
scala> val z = 42
scala> z = z + 1
scala> val x = 2 * z
scala> val z = 466
scala> println(x)
```

Task 3 (Strings)

Get familiar with constructing strings and printing strings (i.e. `println`, `mkString`, `toString`, …)

```
scala> println("Hello " ++ "World")
scala> List(1,2,3,4).mkString("\n")
scala> List(1,2,3,4).mkString("(", "|", ")")
```

Task 4 (Functions)

Write a function `miles2meters` taking an integer as argument and generating an integer. You can assume that one mile is 1609 meters. Remember that functions

in Scala do not use the `return`-keyword.

Task 5 (If-Conditions)

Make sure you understand if-conditions in Scala: They are *expressions*, meaning they need to calculate a result. For example an `if` without an else-branch is nearly always *not* what you intend to write (because you are not meant to change any "state" outside the expression). Also, remember the quirks in Scala with if-conditions needing parentheses and there is no `then`-keyword.

```
scala> val s1 = "foo"
scala> val s2 = "bar"
scala  val s3 = "foobar"
scala> if (s1 == s2) print("equal") else print("unequal")
scala> if (s1 ++ s2 == s3) print("equal") else print("unequal")
```

Task 6 (For-Comprehensions, Harder)

Write `for`-comprehensions that enumerate all triples containing the numbers 1 - 5. That is, print the list

`(1,1,1), (2,1,1), (3,1,1) ... (5,5,5)`

Modify the `for`-comprehensions such that only triples are printed where the sum is divisible by 3. Then sort the list according to the middle element (for this use `sortBy`).

Task 7 (Recursion, Advanced /Hard)

Write a pretty print function for lists of integers which "condenses" elements in the list, meaning if there is a number several times in a row, then print out `n x e`, where `n` is the number of repetitions and `e` is the number. For example

```
List(1,1,1,2,3,3)    =>   List(3 x 1, 2, 2 x 3)
List(1,2,3,4,4,4)    =>   List(1, 2, 3, 3 x 4)
List(1,1,1,1,1,1)    =>   List(6 x 1)
List(1,1,1,2,1,1)    =>   List(3 x 1, 2, 2 x 1)
```

You might like to define a separate function that first counts the occurences for each element and returns a list of (Int, Int)-pairs .