*built on 2016/09/13 at 16:35:13* ***due:*** *tue sep 20 @ 11:59pm*

This short assignment contains only a coding portion. The aim of this assignment is to help familiarize yourself with Scala's programming environment and begin to program in a functional style. We're providing stubs for you; please download starter code from the course website.

**IMPORTANT:** Your implementation must strictly follow the functional style. As a rule of thumb, you cannot use features other than what we have done in class so far. In particular, this means, no loops, no mutable variables (cannot use `var`).

## Non-Task: Scala Environment

In this class, we will be using Scala 2.11.x (the specific minor release doesn't matter), running on Java 1.8. For now, you can use `brew install scala`.

A good IDE can help with your programming a great deal. Many professional programmers are happy with Eclipse and IntelliJ IDEA, all of which are free for academic use. You can't go wrong with any of them. If you're a command-line kind of person, editors such as Sublime, Atom, Vim, and Emacs are good candidates as well.

## Task 1: Factorial (4 points)

*For this task, save your code in* `Factorial.scala`

Remember that $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$ and $0! = 1$. Write a function

```
def factorial(n: Int): Long
```

so that for $0 \leq n \leq 19$, `factorial(n)` $\Rightarrow n!$.

## Task 2: Aggregate: Min, Mean, Median (4 points)

*For this task, save your code in* `Aggregate.scala`

Your goal is to define the following *three* functions

```
def myMin(p: Double, q: Double, r: Double): Double
def myMean(p: Double, q: Double, r: Double): Double
def myMed(p: Double, q: Double, r: Double): Double
```

where

- `myMin(p, q, r)` takes 3 numbers and returns the minimum of the three numbers. For example, `myMin(3.0,1.0,9.0)` should return 1.0.
- `myMean(p, q, r)` takes 3 numbers and returns the average of the three numbers. You should recall that the average of $p$, $q$, and $r$ is simply $\frac{1}{3}(p+q+r)$. Hence, as an example, `myMean(3, 7, 4)` should return $4.6666\cdots$.
- `myMed(p, q, r)` takes 3 numbers and returns the median of the three numbers. Remember that the median of three numbers is the number where one other number is smaller than it and one other number is larger than it. For instance, `myMed(4,1,5)` should return 4. Also, `myMed(13,5.0,12)` should return 12.

## Task 3: Happy Numbers (Again?) (8 points)

*For this task, save your code in* `Happy.scala`

Happy numbers are a nice mathematical concept. Just as only certain numbers are prime, only certain numbers are happy—under the mathematical definition of happiness. We'll tell you exactly what happiness means.

To test whether a number is happy, we can follow a simple step-by-step procedure:

(1) Write that number down
(2) Stop if that number is either 1 or 4.
(3) Cross out the number you have now. Write down instead the sum of the squares of its digits.
(4) Repeat Step (2)

When you stop, if the number you have is 1, the initial number is *happy*. If the number you have is 4, the initial number is *sad*. There are only two possible outcomes, happy or sad.

By this definition, 19 is a happy number because $1^2 + 9^2 = 82$, then $8^2 + 2^2 = 68$, then $6^2 + 8^2 = 100$, and then $1^2 + 0^2 + 0^2 = 1$. However, 145 is sad because $1^2 + 4^2 + 5^2 = 42$, $4^2 + 2^2 = 20$, and $2^2 + 0^2 = 4$.

**Subtask I:** First, you'll implement a function `def sumOfDigitsSquared(n: Int): Int` that takes a positive number `n` and sum the squares of its digits. For example,

- `sumOfDigitsSquared(7)` should return `49`
- `sumOfDigitsSquared(145)` should return `42` (i.e., 1**2 + 4**2 + 5**2 = 1 + 16 + 25)
- `sumOfDigitsSquared(199)` should return `163` (i.e., 1**2 + 9**2 + 9**2 = 1 + 81 + 81)

**Subtask II:** Then, you'll write a function `def isHappy(n: Int): Boolean` that takes as input a positive number `n` and tests if `n` is happy. You may wish to use what you wrote in the previous subtask.

- `isHappy(100)` should return `true`
- `isHappy(111)` should return `false`
- `isHappy(1234)` should return `false`
- `isHappy(989)` should return `true`

**Subtask III:** There are in fact many levels of happiness. The larger the number, the happier we feel about number. The $k$-th happy number is the $k$-th smallest happy number. This means, the 1st happy number is the smallest happy number, which is 1. The 2nd happy number is the second smallest happy number, which is 7. Below is a list of the first few happy numbers:

> 1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, 103, 109, 129, 130, 133, 139, 167, 176, 188, 190, ...

Implement a function `def kThHappy(k: Int): Int` that computes and returns the $k$-th happy number. For example:

- `kThHappy(1)` returns 1
- `kThHappy(3)` returns 10
- `kThHappy(11)` returns 49
- `kThHappy(19)` returns 97

**TIP:** Use what you have already written. Don't repeat yourself.

## Task 4: Roman Numerals (4 points)

*For this task, save your code in* `Roman.scala`

You surely have encountered Roman numerals: I, II, III, XXII, MCMXLVI, etc. They are everywhere. Roman numerals are represented by repeating and combing the following seven characters: `I` = 1, `V` = 5, `X` = 10, `L` = 50, `C` = 100, `D` = 500, `M` = 1000. To understand how they must be composed, we borrow the following excerpt from *Dive Into Python*:

- Characters are additive. `I` is 1, `II` is 2, and `III` is 3. `VI` is 6 ("5 and 1"), `VII` is 7, and `VIII` is 8.
- The tens characters (`I`, `X`, `C`, and `M`) can be repeated up to three times. At 4, you need to subtract from the next highest fives character. You can't represent 4 as `IIII`; instead, it is represented as `IV` ("1 less than 5"). The number 40 is written as `XL` (10 less than 50), 41 as `XLI`, 42 as `XLII`, 43 as `XLIII`, and then 44 as `XLIV` (10 less than 50, then 1 less than 5).
- Similarly, at 9, you need to subtract from the next highest tens character: 8 is `VIII`, but 9 is `IX` (1 less than 10), not `VIIII` (since the `I` character cannot be repeated four times). The number 90 is `XC`, 900 is `CM`.
- The fives characters cannot be repeated. The number 10 is always represented as X, never as `VV`. The number 100 is always `C`, never `LL`.
- Roman numerals are always written highest to lowest, and read left to right, so the order the of characters matters very much. `DC` is 600; `CD` is a completely different number (400, 100 less than 500). `CI` is 101; `IC` is *not* a valid Roman numeral (because you can't subtract 1 directly from 100; you would need to write it as `XCIX`, for 10 less than 100, then 1 less than 10).

**YOUR TASK:** Implement a function `def toRoman(n: Int): String` that takes an integer n ($1 \leq n < 4,000$) and returns a string that represents the number n in Roman numerals.

*Despite the complexity this problem may seem at first, there are nice (and not tedious) ways to implement the logic to convert an integer into a Roman numeral using only features we have learned so far.* You should not write more than 40 lines of code.