Week1 - Scala Language pt.1: Homework

Requirements

We just started to learn Scala language that means we want to practice only a small part of it's features not employing its full power. And, at the same time we are willing to learn how to do things in the Functional Programming way and Scala way. Hence, these requirements:

- No usage of mutable variables (**var**)
- Use pattern matching over lists (pretend you don't know List api yet)
- Use recursion or for-comprehension to iterate over collection
- Use everything else we learned in class

You can breach some of the above of course if you fill it's impossible or way too hard for you to finish the exercise.

Submission

You can program creating new Worksheet in IntelliJ Idea and then submit that worksheet file or multiple worksheets archived if you use a separate worksheet for each exercise.

Grading

This Homework is graded 10 scores from overall 100 for the course.

Penalty

For every day after the deadline maximum score for the homework is reduced by 10%.

Assignments

Some of the assignments were partially covered in class.
Divide '/' operator in Scala returns rounded (to smaller) integer while modulus '%' operator returns remainder. To return a float number, you should turn the initial number into float, e.g. `1.toFloat / 2` should give you 0.5.
Here is how you can put a number to the power in Scala - ` scala.math.pow(2,3)` - meaning 2 in the power of 3 or 2^3.
You can use 'assert(*expected* == *actual*)' to have simple test cases covered.

Exercise 1:

Write a function that takes three numbers as arguments and

returns the sum of the squares of the two larger numbers.

Exercise 2:

Write a function that computes the square root of a given number using Newton's method. Define two functions: 'sqrt' and 'goodEnough'. 'good Enough' should check whether given approximation obtained after some step is good enough given some difference.

Exercise 3:

Write your own 'if' statement that takes a predicate, then-clause and else-clause as parameters (e.g. function *myIf(..)*).
Try to use this custom 'if' statement in the previous function that computes square root. Does it work fine?
If not, try to explain why?

Exercise 4:

Newton’s method for cube roots is based on the fact that if _y_ is an approximation to the cube root of _x_,
then a better approximation is given by the value:

((x/y^2) + 2y)/3

Use this formula to implement a function to estimate the cube root of the given number.

Exercise 5:

Write a function to compute the factorial of the given number using recursive function. Rewrite it to be tail-recursive.

Exercise 6:

Write a function to compute n-th term of the Fibonacci sequence. Write both recursive and tail-recursive functions.

Exercise 7:

The following pattern of numbers is called
Pascal's triangle .

```
      1
    1   1
  1   2   1
1   3   3   1
1   4   6   4   1
```

The numbers at the edge of the triangle are all 1, and each number inside the triangle is the sum of the two numbers above it. Write a function that computes elements of Pascal's triangle.

Exercise 8:

Write a function that tests a given number for primality.

Exercise 9:

Simple Expression parser.
Build an expression evaluator that works on tokens of strings composed of integers and binary operators + and * (see example)
Grammar:
expression -> number {operator number}*
operator -> '+' | '*'
number -> regex([0-9]+)
Hints:
- use pattern matching to match operators and numbers in List
- use .toInt to parse number
"1".toInt == 1

```
def eval(l: List[String]): Int = ???

assert(eval(List("1", "+", "2", "*", "3")) == 7)
assert(eval(List("5", "*", "3", "+", "1")) == 16)
```