Programming Paradigms Tutorials
Tail recursion

Tail recursion

Let's check our last implementations of Factorial function

The implementation of Factorial in "standard" way by for loop:

```
def calculateByForLoop(n: Int): BigInt = {
   require(n>0, "n must be positive")

   var accumulator: Out = 1
   for (i <- 1 to n) {
     accumulator = i * accumulator
   }
   accumulator
}
```

The implementation of Factorial in functional way by recursion:

```
def calculateByRecursion(n: Int): BigInt = {
   require(n>0, "n must be positive")

   n match {
     case _ if n == 1 => return 1
     case _ => return n * calculateByRecursion(n-1)
   }
}
```

As you remember the above calculation by recursion may throw StackOverflow. This is because the space needed to store the variables and information associated with each call is more than can fit on the stack.
The tail recursive functions better than non-tail recursive functions because tail-recursion can be optimized by compiler. A recursive function is said to be tail recursive if the **recursive call is the last thing done by the function**. There is no need to keep record of the previous states on the stack.

Syntax:

```
@tailrec
def FuntionName(Parameter1, Parameter2, ...): type = …
```

The addnotation `@tailrec` do not change the way how the method works, but If the implementation of function is not tail recursive, an error would be issued.

The implementation of Factorial in functional way by tile recursion:

```
def calculateByTailRecursionUpward(n: Int): Out = {
   require(n>0, "n must be positive")

   @tailrec def fac(i: Int, acc: Out): Out = n match {
     case _ if i == n => n * acc
     case _ => fac(i+1, i * acc)
   }

   fac(1, 1)
}
```

Exercise:

1. (1pt)Write a function which will sum the values in a list use tail recursion.
   `sumList: (list: List[Int])Int`
2. (1pt) Create a method reversing a list:
   a) regular recursion,
   b) tail recursion,
3. (2pt) Create a method that merges two lists in such a way that elements of both lists alternate, e.g. for lists [1, 2, 3] and [4, 5, 6] result is [1, 4, 2, 5, 3, 6].
   If one of the lists have too many elements, add rest of them to the end of the result.
4. (2pt) Write a function which return n-th Fibonacci number and use tail recursion. Compare the solution with the code from last tutorials.
5. (2pt) Write function which split list to two lists:
   - first include only odd numbers
   - second include only even numbers
6. (1pt) Write a function which check that numbers in provided list are correctly sorted. Please use tail recursion.
7. (2pt) Define the replaceNth function, which replace the n-th element of the list by given value.
   `replaceNth: [A](xs: List[A], n: Int, x: A)List[A]`
   Remember to do not use standard library functions here.
8. (1pt) Use curried representation and write system which convert pressure units.
   Input: pressure in atm (atmosphere)
   Outputs: PSI, bar, torr
   1 atm = 14.6956 psi = 760 torr = 101325 Pa = 1.01325 bar.

IMPORTANT: For every task, be ready to presents set of tests that verifying correctness of the solution.