Programming Paradigms Tutorials
Functional Programming

Functional (FP) vs Object Oriented (OOP) programming

The rules below should help you to write code in functional way. They are compared with well know rules of Object-Oriented Programming

Definition:
FP:
Functional programing emphases on evaluation of functions.
OOP:
Object oriented programing base on concept of object (data structure with methods available for it)
Data:
FP:
Functional programing uses immutable data.
OOP:
Object oriented programing uses mutable data.
Programming model:
FP:
Functional programing follows declarative programing model. It expresses the logic of a computation without describing its control flow.
OOP:
Object oriented programing follows imperative programing model. It uses statements that change a program's state.
Iteration:
FP:
In functional programing recursion is used for iterative data.
OOP:
In object-oriented programming loops (for, while) are used for iterative data.
Element:
FP:
The basic elements of functional programming are variables and functions.
OOP:
The basic elements of functional programming are objects and their methods.
Use:
FP:
Functional programing is better to use when there are few things with more operations.
OOP:
Object oriented programing is better to use when there are more things with few operations.

Loops vs Recursive

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)
   }
}
```

Note: the recursion solution can throw StackOverlow, which is expected behavior for big value of n. It can be solved by tail recursion which is a topic of next tutorials.

Exercise (2.5 pt. each)

1. Write a function which will recursively sum the odd values in a list.
   `sumList: (list: List[Int])Int`
2. Write a function that takes two parameters: separator and the list of strings. This function should return, a single string (strings connected by a separator).
   `connectStrings: (listOfString: List[String], separator: String)String`
3. Write a function that count number of occurrences of specific element in the list.
4. Write a function which return n-th Fibonacci number, Reminder:
   $$Fib(n) = \begin{cases} 0 \; for \; n = 0, \\ 1 \; for \; n = 1, \\ Fib(n-1) + Fib(n-2) \; for \; n > 1. \end{cases}$$
   The Fibonacci sequence looks like: 0, 1, 1, 2, 3, 5, 8, 13, ….

IMPORTANT: For every task prepare set of tests verifying correctness of the solution.