Programming Paradigms
Week 4

Jacek Wasilewski

Exercises

1. Write a recursive function that computes factorial for a given number `n`. Factorial is defined as follows: $n! = n * (n-1)!$ and $0! = 1$.

2. Check if the function you wrote is tail-recursive or not.

3. Write a tail-recursive version of the `factorial` function.

4. Prove that the new factorial function is tail-recursive.

5. Write a function `series` that calculates a geometric series. Function should be defined by three parameters: the starting term, the ratio and the position of the series that should be returned (check `https://en.wikipedia.org/wiki/Geometric_progression` for more details). Example:

   ```
   series(1, 2, 1) = 1
   series(1, 2, 2) = 2
   series(1, 2, 3) = 4
   series(1, 2, 4) = 8
   ```

   Function should be implemented as a tail-recursive function.

6. Prove that function `series` is a tail-recursive function.

7. Implement three functions: `id`, `square`, `cube`. Functions take one double. `id` should return the parameter, `square` should return the square of the parameter, `cube` should return cube of the parameter.

8. Write anonymous versions of the above functions.

9. Write the function `higher` that takes function like those defined in the previous task, and one additional double. Function `higher` should return the result of the passed function applied on the parameter also passed to the `higher` function. Multiply the result by 2 and subtract 1.

10. Write an anonymous function that takes two doubles and returns the power of those. Use `Math.pow` function.

11. Write a function `powFactory` that does not take any parameters. This function should return the anonymous function you created in the previous task.

12. Write a function `greaterThan` that takes one parameters `n` (can by `Int` or `Double`). Function should return an anonymous function that takes one parameter `m` and checks if `m > n`. Then you should be able to do the following:

```
scala> def greaterThan10 = greaterThan(10)
greaterThan10: Int => Boolean

scala> greaterThan10(20)
res9: Boolean = true
```

13. Function `Math.pow` takes two parameters of type `Double`. Write a wrapper function that allows you to first apply only the first parameter and later the second parameters. The following can be an example of calls:

```
scala> powWrapper(2)(3)
res11: Double = 8.0

scala> def pow2(x) = ???
pow2: Double => Double

scala> pow2(3)
res12: Double = 8.0
```