T-501-FMAL Programming languages, Practice class 1
Spring 2021

1. Consider the function f that, in standard mathematical notation, we would define by

$$f(n) = n * 3 + 8$$

Code this function in F# (i) as an anonymous function bound to the name `f` and (ii) as an expression bound to an application of `f` to a formal parameter `n`.

Apply the function f to 5.

Now code the function g defined in terms of f by

$$g(n) = f(n-1)$$

Apply the function g to 12.

2. Consider the function $pow2$ that, in standard mathematical notation, we would define by

$$pow2\,(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 * pow2\,(n-1) & \text{otherwise} \end{cases}$$

Code this function in F# and bind the name $pow2$ with `let rec` with `if then else` and with pattern matching. (Try out also just `let` without `rec`. What happens, why?)

Apply the function $pow2$ to 7, to 9, to -3. (What happens in the latter case, why?)

3. Code in F# the function mc defined in standard mathematical notation by

$$mc\,(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ mc\,(mc\,(n+11)) & \text{otherwise} \end{cases}$$

Apply the function mc to 73, to 84, to 92, to 95, to 109, to 117, to 125. What do you observe?

4. Code in F# the function *feq* that takes as inputs functions h, k and integers n, m and checks whether $h\,i = k\,i$ for all $i = n..m$.

You will need to know that, in F#, conjunction is `&&` (and disjunction is `||`).

(It should be a good idea to stop the check as soon as a value for i is found for which $h\,i \neq k\,i$.)

Apply the function *feq* (i) to the function mc, the constantly 91 function, 1 and 100, (ii) to the function mc, the function $\lambda n.\ n - 10$, 101 and 200.

($\lambda x.\ e$ is mathematical notation for an anonymous function: the expressions e seen as a function of the variable x. It is the same as `fun x -> e` in F#.)

5. Code in F# a function `groups3 : 'a list -> ('a list) list` that splits a given list into groups of three values (the last group may also be of length 1 or 2, but there should not be any empty groups).

```
> groups3 [1;5;3;4;3;5;5;7];;
val it : (int list) list = [[1;5;3];[4;3;5];[5;7]]
```

6. (i) Code in F# a function `takeDrop : int -> 'a list -> 'a list * 'a list` that splits a given list into two lists, where the first one contains its first n elements (or less, if the list is too short) and the second one all the remaining elements. (The function should traverse the given list only once.)

```
> takeDrop 4 [3;2;5;9;4;6;8;0;1];;
val it : int list * int list = ([3;2;5;9];[4;6;8;0;1])
> takeDrop 6 [3;2;5;9;4];;
val it : int list * int list = ([3;2;5;9;4];[])
```

(ii) Using `takeDrop`, code a function `groupsN : int -> 'a list -> ('a list) list` that splits a given list into groups of n values (the last group may also be of length $1..n-1$, but there should not be any empty groups).

```
> groupsN 4 [3;2;5;9;4;6;8;0;1];;
val it : (int list) list = [[3;2;5;9];[4;6;8;0];[1]]
```