Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

```
#r "nuget:diku.canvas, 1.0.1"
open Canvas

let rec tree (sz: int) : Canvas.turtleCmd list =
   if sz < 5 then
     [Move sz; PenUp; Move (-sz); PenDown]
   else
     [Move (sz/3); Turn -30]
     @ tree (sz*2/3)
     @ [Turn 30; Move (sz/6); Turn 25]
     @ tree (sz/2)
     @ [Turn -25; Move (sz/3); Turn 25]
     @ tree (sz/2)
     @ [Turn -25; Move (sz/6)]
     @ [PenUp; Move (-sz/3); Move (-sz/6); Move (-sz/3)]
     @ [Move (-sz/6); PenDown]

let w = 600
let h = w
let sz = 100
turtleDraw (w,h) "Tree" (tree sz)
```

Figure 1: A turtle graphics program for drawing a fractal tree in Canvas.

0.1 Forest

0.1.1 Teacher's guide

Emne rekursion, grafik og winforms

Sværhedsgrad Middel

0.1.2 Introduction

In the following we are going to work with lists and Canvas. The module Canvas has the ability to perform simple turtle graphics. To draw in turtle graphics, we command a little invisible turtle, which moves on the canvas with a pen. The function `turtleDraw` is given a list of `turtleCmds`, such as `PenUp` and `PenDown` to raise and lower the pen, `Turn 250` and `Move 100` to turn 250 degrees and move 100 pixels, and `SetColor red` to pick a red pen. For example in **??**, the function `tree sz` creates the set of turtle commands for drawing a fractal tree of size `sz` and returns the turtle to the starting point. The command `turtleDraw` executes the list of turtle commands, which in this case draws the tree on a canvas and displays it. In this exercise, you are to work with turtle commands.

0.1.3 Exercise(s)

0.1.3.1: Consider a list of pairs of integers `(dir,dist)` which are to be translated into a list of turtle commands, such that `[(10, 30); (-5, 127); (20, 90)]` is translated into `[Turn 10;`

`Move 30; Turn -5; Move 127; Turn 20; Move 90]`. You are to

(a) Write the following functions:

```
type move = int*int // a pair of turn and move
fromMoveRec: lst: move list -> Canvas.turtleCmd list
fromMoveMap: lst: move list -> Canvas.turtleCmd list
fromMoveFold: lst: move list -> Canvas.turtleCmd list
fromMoveFoldBack: lst: move list -> Canvas.turtleCmd list
```

where

i. `fromMoveRec` is a recursive function, that does not use the List module

ii. `fromMoveMap` is non-recursive function, which uses `List.map`

iii. `fromMoveFold` is non-recursive function, which uses `List.fold`

iv. `fromMoveFoldBack` is non-recursive function, which uses `List.foldBack`

In some of the above functions, you may also find it useful to use `List.concat` and `List.rev`.

(b) Demonstrate that these 4 functions produce the same result given identical input.

0.1.3.2: (a) The following program

```
let rnd = System.Random()
let v = rnd.Next 10
```

makes a random integer between the integer $0 \leq v < 10$. Use this to make a function

```
randomTree: sz: int -> Canvas.turtleCmd list
```

which calls `tree sz` and concatenates further turtle commands to place a tree randomly on a canvas, and return the turtle to the origin. Test your function by calling `turtleDraw` with such a list.

(b) Write a recursive function

```
forest: sz: int -> n: int -> Canvas.turtleCmd list
```

which makes n random trees on the canvas by calling `randomTree` n times. Test your function by calling `turtleDraw` with such a list.