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

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 Bindings

0.1.1 Teacher's guide

Emne bindinger af værdier, funktioner, mutérbare variable, og løkker

Sværhedsgrad Let

0.1.2 Introduction

Being a functional-first programming language, many of its structures are designed to support functional programming style. However, imperative programming constructs are also available. In the following exercises, you will work with lightweight and verbose syntax and simple imperative programming assignments.

0.1.3 Exercise(s)

0.1.3.1: Type the following program in a text file, compile, and execute it:

Listing 1: Value bindings.

```
let a = 3
let b = 4
let x = 5
printfn "%A * %A + %A = %A" a x b (a * x + b)
```

Explain why there must be a paranthesis in the `printfn` statement. Add a binding of the expression $ax + b$ to the name `y`, og modify the `printfn` call to use `y` instead. Are parantheses still necessary?

0.1.3.2: Listing 1 uses Lightweight syntax. Rewrite the program such that Verbose syntax is used instead.

0.1.3.3: The following program:

Listing 2: Strings.

```
let firstName = "Jon"
let lastName = "Sporring" in let name = firstName + " " +
   lastName;;
printfn "Hello %A!" name;;
```

was supposed to write "Hello Jon Sporring!" to the screen. Unfortunately, it contains an error and will not compile. Find and correct the error(s). Rewrite the program into a single line without the use of semicolons.

0.1.3.4: Add a function:

```
f : a:int -> b:int -> x:int -> int
```

to Listing 1, which returns the value of $ax+b$, and modify the call in `printfn` to use the function rather than the expression `(a * x + b)`.

0.1.3.5: Use the function developed in Assignment 4, such that the values for the arguments $a = 3$, $b = 4$, and $x = 0\ldots5$ are written using 6 `printfn` statements. Modify this program to use a `for` loop and a single `printfn` statement. Repeat the modification, but this time by using a `while` loop instead. Which modification is simplest and which is most elegant?

0.1.3.6: Make a program, which writes the multiplication table for the number 10 to the screen formatted as follows:

	1	2	…	10
1	1	2	…	10
2	2	4	…	20
⋮				
10	10	20	…	100

I.e., left row and top columns are headers showing which numbers have been multiplied for an element in the temple. You must use `for` loops for the repeated operations, and the field width of all the positions in the table must be identical.

0.1.3.7: Consider multiplication tables of the form,

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	19	20
3	3	6	9	12	15	18	21	24	27	30
…										

where the elements of the top row and left column are multiplied and the result is written at their intersection.

In this assignment, you are to work with a function

```
mulTable : n:int -> string
```

which takes 1 argument and returns a string containing the first $1 \leq n \leq 10$ lines in the multiplication table including `<newline>` characters. Each field must be 4 characters wide. The resulting string must be printable with a single`printf "%s"`statement. For example, the call `mulTable 3` must return.

Listing 3: An example of the output from `mulTable`.

```
printf "%s" (mulTable 3);;
        1   2   3   4   5   6   7   8   9  10
   1    1   2   3   4   5   6   7   8   9  10
   2    2   4   6   8  10  12  14  16  18  20
   3    3   6   9  12  15  18  21  24  27  30
```

All entries must be padded with spaces such that the rows and columns are right-aligned. Consider the following sub-assignments:

(a) Create a function with type

```
mulTable : n:int -> string
```

such that it has one and only one value binding to a string, which is the resulting string for $n = 10$, and use indexing to return the relevant tabel for $n \leq 10$. Test `mulTable n` for $n = 1,2,3,10$. The function should return the empty string for values $n < 1$ and $n > 10$.

(b) Create a function with type

```
loopMulTable : n:int -> string
```

such that it uses a local string variable, which is built dynamically using 2 nested `for`-loops and the `sprintf`-function. Test `loopMulTable n` for $n = 1,2,3,10$.

(c) Make a program, which uses the comparison operator for strings, "`=`", and write a table to the screen with 2 columns: `n`, and the result of comparing the output of `mulTable n` with `loopMulTable n` as `true` or `false`, depending on whether the output is identical or not.

(d) Use `printf "%s"`and `printf "%A"`to print the result of `mulTable`, and explain the difference.

0.1.3.8: Consider the faculty-function,

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \ldots \cdot n \tag{1}$$

(a) Write a function

```
fac : n:int -> int
```

which uses a `while`-loop, a counter variable, and a local variable to calculate the faculty-function as (1).

(b) Write a program, which asks the user to enter the number `n` using the keyboard, and which writes the result of `fac n`.

(c) Make a new version,

```
fac64 : n:int -> int64
```

which uses `int64` instead of `int` to calculate the faculty-function. What are the largest values n, for which `fac` and `fac64` respectively can calculate the faculty-function for?

0.1.3.9: Consider the following sum of integers,

$$\sum_{i=1}^{n} i. \tag{2}$$

This assignment has the following sub-assignments:

(a) Write a function

```
sum : n:int -> int
```

which uses the counter value, a local variable (mutable value) `s`, and a `while`-loop to compute the sum $1+2+\cdots+n$ also written in (2). If the function is called with any value smaller than 1, then it is to return the value 0.

(b) By induction one can show that

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, n \geq 0 \tag{3}$$

Make a function

```
simpleSum : n:int -> int
```

which uses (3) to calculate $1+2+\cdots+n$ and which includes a comment explaining how the expression implemented is related to the mentioned sum.

(c) Write a program, which asks the user for the number n, reads the number from the keyboard, and write the result of `sum n` and `simpleSum n` to the screen.

(d) Make a program, which writes a table to the screen with 3 columns: `n`, `sum n` and `simpleSum n`. The table should have a row for each of $n = 1,2,3,..,10$, and each field must be 4 characters wide. Verify that the two functions calculate identical results.

(e) What is the largest value n that the two sum-functions can correctly calculate the value of? Can the functions be modified, such that they can correctly calculate the sum for larger values of n?

0.1.3.10: Perform a trace-by-hand of the following program

```
let a = 3.0
let b = 4.0
let f x = a * x + b

let x = 2.0
let y = f x
printfn "%A * %A + %A = %A" a 2.0 b y
```