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 My first Fsharp program

0.1.1 Teacher's guide

Emne De studerende skal

- lave deres første F# program
- komme til at kende forskellen mellem decimal, binær, hexadecimal, og oktal repræsentation af heltal, samt at kunne konvertere imellem dem.
- komme til at kunne beskrive simple typer i F#: `int`, `float`, `char`, `string`, `bool`, samt konvertering imellem dem.
- komme til at kunne bruge F# som en lommeregner.

Sværhedsgrad Let

0.1.2 Introduction

F# is a functional programming first language, i.e., functional programming is very well supported by the language constructions. It uses types, but they do not always need to be specified. It has two modes: Interactive and Compile mode. The following assignments are designed to get you accustommed with F#.

0.1.3 Exercise(s)

0.1.3.1: Start an interactive F# session and type the following ending with a newline:

Listing 1: My first F#.

```
3.14+2.78;;

```

Describe what F# did and if there was an error, find it and repeat.

0.1.3.2: Consider the F#-expression `"hello\nworld\n"`. Explain what the "\n" means, compute the expression using F# and discuss the result.

0.1.3.3: Repeat Exercise 1, but this time, type the code in a text editor and save the result in a file with the suffix `.fsx`. Run through `fsharpi` from the console, and by first compiling it with `fsharpc` and executing the compiled file using `mono`. Consider whether the result was as expected and why.

0.1.3.4: Write an expression which concatenates the strings `"hello"`, `" "`, `"world"` and run it in F#.

0.1.3.5: Use a recursive-loop and pattern recognition to write a program, which prints the numbers 1...10 on the screen.

0.1.3.6: Use a `while`-loop and a mutable value to write a program, which prints the numbers 1...10 on the screen.

0.1.3.7: Using pen and paper:

a) Write the integer 3_{10} on binary form by using the divide-by-2 algorithm.

b) Write the integer 1001_2 on decimal form using the multiply-by-2 algorithm.

c) Write the integer 47_{10} on hexadecimal and octal form.

0.1.3.8: Modify the following program (`quickStartRecursiveInput.fsx`):

```
let rec readNonZeroValue () =
  let a = int (System.Console.ReadLine ())
  match a with
    0 ->
      printfn "Error: zero value entered. Try again"
      readNonZeroValue ()
    | _ ->
      a
printfn "Please enter a non-zero value"
let b = readNonZeroValue ()
printfn "You typed: %A" b
```

such that instead of asking the user for a non-zero value, repatedly asks the user for the name of a programming language. If the user enters `"quit"`, then the program should stop. If the user enters `"fsharp"`, then the program should write `"fsharp is cool"`. In all other cases, program should write `"I don't known <name>"`, where `"<name>"` is the string, the user entered. Example of a user dialogue is:

```
% dotnet fsi fsharpIsCool.fsx
Please enter the name of a programming language:
c
I don't know "c"
Please enter the name of a programming language:
fsharp
Fsharp is cool
Please enter the name of a programming language:
quit
```

0.1.3.9: Start `dotnet fsi` and interactive mode, write a function `subInt a b`, where the arguments are integers, and which returns $a - b$. Write a similar function `subFloat a b` where the arguments are floats. In the type-signature returned by F#, how can you see that these functions have 2 arguments and what their types are?

0.1.3.10: Write a program in an editor, which

(a) Writes `"Hello, what is your name:"` to the screen

(b) Reads the users name from the keyboard

(c) Prints `"Hello <name>"` to the screen where `<name>` is replaced by what the user enters.

Save the file, and run the program using `"dotnet fsi <filename>"` and verify that it does as you expect.

0.1.3.11: Start an editor and write a program which prints “hello world” on the screen. Save the file as a `.fsx` file. Execute the program from the command line using

```
dotnet fsi <filename>
```

where `<filename>` is the name of the file you chose, and verify that it prints as expected.

0.1.3.12: Describe the 3 ways an F# program can be run from the command line (terminal), and discuss the advantages and disadvantages of each method.

0.1.3.13: Enter the integer 47_{10} on hexadecimal, octal, and floating-point form in F# and verify that all represents the same value.

0.1.3.14: Make a program, which opens a canvas and draws a simple shape of your own choosing. Save the image to a `.png` file and verify that the result is as expected.

0.1.3.15: Write an F#-expression which extracts the 3. element and the substring from the 2. to the 4. element in the string “abcdef”.

0.1.3.16: Write the F#-expression, which extracts the first and second word from the string “hello world” using slicing.

0.1.3.17: Use pen and paper to complete the following table

Decimal	Binary	Hexadecimal	Octal
10			
	10101		
		2f	
			73

such that every row represents the same value written on 4 different forms. Include a demonstration of how you converted binary to decimal, decimal to binary, binary to hexadecimal, hexadecimal to binary, binary to octal, and octal to binary.

0.1.3.18: Write the truth table for the boolean expression a **or** b **and** c, where a, b, and c are boolean values.

0.1.3.19: Type the following expression in F# interactive mode,

Listing 2: Problematic F#.

```
3 + 1.0;;
```

and explain the result. Consider whether you can improve the expression.

0.1.3.20: Write an F#-expression for a string that contains the characters “edb” solely by using unicode escape codes.

0.1.3.21: Consider the F#-expression `164uy+230uy`. Explain what `"uy"` means, compute the expression with `fsharpi`, and discuss the result.

0.1.3.22: Write an F#-expression for a string which contains the character sequence “\n”, but where “\n” is not converted to a newline. How many different ways can this be done?