Getting Started

Jon Sporring

July 12, 2019

1 Check the installation, get accustomed to the interactive and the compile mode

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#.

1.1. 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.

1.2. Repeat Exercise 1.1, 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.

1.3. 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.

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

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

2 Get accustomed to F# syntax and precedence

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.

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

Listing 3: 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?

2.2. Listing 3 uses Lightweight syntax. Rewrite the program such that Verbose syntax is used instead.

2.3. The following program:

Listing 4: 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.

2.4. Add a function:

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

to Listing 3, which returns the value of $ax+b$, and modify the call in printfn to use the function rather than the expression (a * x + b).

2.5. Use the function developed in Assignment 2.4, such that the values for the arguments $a = 3$, $b = 4$, and $x = 0\dots5$ 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?

2.6. 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.

3 Advanced exercise using modules

This assignment is about 2-dimensional vectors. A 2-dimensional vector (henceforth just called a vector) is a geometrical object consisting of a length and a direction. Typically, a vector is represented as a pair of numbers, $\vec{v} = (x, y)$, where its length and direction are found as,

$$\text{len}(\vec{v}) = \sqrt{x^2 + y^2} \tag{1}$$
$$\text{ang}(\vec{v}) = \text{atan2}(y, x) \tag{2}$$

Vectors are often drawn as arrows with a head and a tail. In the Cartesian coordinate system, if the tail is placed at $(0, 0)$, then the head will be at (x, y). There exists a number of standard operations for vectors:

$$\vec{v}_1 = (x_1, y_1) \tag{3}$$
$$\vec{v}_2 = (x_2, y_2) \tag{4}$$
$$a\vec{v}_1 = (ax_1, ay_1) \tag{5}$$
$$\vec{v}_1 + \vec{v}_2 = (x_1 + x_2, y_1 + y_2) \tag{6}$$
$$\vec{v}_1 \cdot \vec{v}_2 = x_1x_2 + y_1y_2 \tag{7}$$

Addition can also be drawn, as shown in Figure 1.

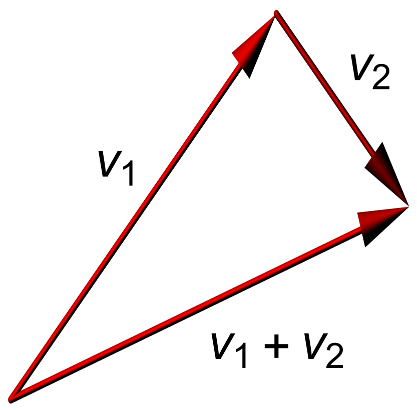

Figure 1: An illustation of vector addition.

3.1. Write a library vec2d.fs, which implements the signature file given in Listing 5.

Listing 5 vec2d.fsi:
A signature file for vector operations.

```
/// A 2 dimensional vector library.
/// Vectors are represented as pairs of floats
module vec2d
/// The length of a vector
val len : float * float -> float
/// The angle of a vector
val ang : float * float -> float
/// Multiplication of a float and a vector
val scale : float -> float * float -> float * float
/// Addition of two vectors
val add : float * float -> float * float -> float * float
/// Dot product of two vectors
val dot : float * float -> float * float -> float
```

3.2. Points on a circle of radius 1 can be calculated as $(\cos\theta, \sin\theta)$, $\theta \in [0, 2\pi)$. Consider the closed polygon consisting of $n > 1$ points on a circle, where $\theta_i = \frac{2\pi i}{n}$, $i = 0..(n-1)$, and where neighbouring points are conected with straight lines.

Write a program with the function,

```
polyLen : n:int -> float
```

which uses the above library to calculate the length of the polygon. The length is calculated as the sum of line pieces connecting neighbouring points. The program should further write a table of lengths for increasing number of points n, and the results should be compared with the circumference of a circle with radius 1. What appears to be the limit, when $n \to \infty$?

3.3. The library vec2d is based on the representations of vectors as pairs (2-tuples). Make a sketch of a signaturfil for a variant of the library, which avoids tuples. Discuss possible challenges and major changes, which the variant will require both for the implementation of the library and the application program.

4 Recursion in F#

Recursion is a central concept in functional programming, and is usually used instead of for and while loops. The following exercise thus train the use of recursion.

4.1. Write a function, fac : n:int -> int, which calculates the faculty function $n! = \prod_{i=1}^{n} i$ using recursion.

4.2. Write a function, sum : n:int -> int, which calculates the sum $\sum_{i=1}^{n} i$ using recursion. Make an implementation using a while loop, compute a table for the values $n = 1..10$ and compare the results. Discuss the difference between the two methods.

4.3. The greatest common demoninator of 2 integers t and n is the greatest number c, which integer-divides both t and n with 0 remainder. Euclid's algorithm[1] finds this greatest common denominator via recursion:

$$\gcd(t,0) = t, \tag{8}$$
$$\gcd(t,n) = \gcd(n, t \% n), \tag{9}$$

where $\%$ is the remainder operator (as in F#).

(4.3.1) Implement Euclid's algorithm as the recursive function:

```
gcd : int -> int -> int
```

(4.3.2) Test your implementation.

(4.3.3) Make a tracing-by-hand on paper for `gcd 8 2` and `gcd 2 8`.

5 Lists and patterns

A list is a very important programming data structure, and functional programming is particularly well suited for processing lists. Hence, F# has constructs that support list operations, and some of these will be worked with in the following assignments.

5.1. Write a function `multiplicity : x:int -> xs:int list -> int`, which counts the number of occurrences of the number `x` in the list `xs`.

5.2. Use `List.filter` to make a function `evens : int list -> int list`, which returns all the even numbers of a list.

5.3. Define a function `reverseApply : 'a -> ('a -> 'b) -> 'b`, such that the call `reverseApply x f` returns the result of `f x`.

5.4. Use `List.map` and `reverseApply` (from Exercise 5.3) to make a function `applylist : ('a -> 'b) list -> 'a -> 'b list`, which applies a list of functions to the same element and returns a list of results.

5.5. Define a function `concat : 'a list list -> 'a list`, which concatenates a list of lists to a single list. For example, the list `concat [[2]; [6; 4]; [1]]` should be flattened into `[2; 6; 4; 1]`.

6 Individual hand-in assigment: Map, Fold, and Filter

The functions `map`, `fold`, and `filter` are very powerful functions for processing lists, and they are often essential parts of high-performing parallel programs. In this assigment, you are to work with implementing some of these yourself.

[1] https://en.wikipedia.org/wiki/Greatest_common_divisor

In the file recursiveMapFoldFilter.fsx there is a fully functioning program, which must be compiled and executed from the console. It takes 2 arguments: a string and a postive integer *n*. The string can be either map, fold, or filter . The output is a random list of length *n* consisting of positive integers less than 10 and a processed list. For map, the random elements have been multiplied by 2, for fold, the random elements have also been multiplied by 2 but their order have been reversed, and for filter , only those elements larger than 4 have been included.

6.1. Replace the library calls in myFold and myFilter with your own recursive implementations of these functions.

You must also write a short report, which

- is no larger than 2 pages;
- contains a brief discussion on how your implementation works, and if there are any possible alternative implementations, and in case, why you chose the one, you did;
- includes output that demonstrates that your program works as intented.

The report is to be handed in as a pdf document together with the single F# source code as an fsx file.

7 Group hand-in assigment: Random Text

H.C. Andersen (1805-1875) is a Danish author who wrote plays, travelogues, novels, poems, but perhaps is best known for his fairy tales. An example is Little Claus and Big Claus (Danish: Lille Claus og store Claus), which is a tale about a poor farmer, who outsmarts a rich farmer. A translation can be found here: http://andersen.sdu.dk/vaerk/hersholt/LittleClausAndBigClaus_e.html. It starts like this:

> "LITTLE CLAUS AND BIG CLAUS a translation of hans christian andersen's 'lille claus og store claus' by jean hersholt.
>
> In a village there lived two men who had the self-same name. Both were named Claus. But one of them owned four horses, and the other owned only one horse; so to distinguish between them people called the man who had four horses Big Claus, and the man who had only one horse Little Claus. Now I'll tell you what happened to these two, for this is a true story."

In this assigment, you are to work with simple text processing, analyse the statistics of the text, and use this to generate a new text with similar statistics.

7.1. The script readFile.fsx reads the content of the text file readFile.fsx. Convert this script into a function which reads the content of any text file and has the following type:

```
readText : filename:string -> string
```

7.2. Write a program that converts a string, such that all letters are converted to lower case, and removes all characters except a. . . z. It should have the following type:

```
convertText : src:string -> string
```

7.3. Write a program that counts occurrences of each lower-case letter of the English alphabet in a string and returns a list. The first element of the list should be the count of 'a's, second the count of 'b's etc. The function must have the type:

```
histogram : src:string -> int list
```

7.4. The script sampleAssignment.fsx contains the function

```
randomString : hist:int list -> len:int -> string
```

which generates a string of a given length, and contains random characters distributed according to a given histogram. Modify the code to use your histogram function. Further, write a program, which reads the text littleClausAndBigClaus.txt using readText, converts it using convertText, and calculates its histogram and generates a new random string using histogram and randomString. Test the quality of your code by comparing the histograms of the two texts.

7.5. Write a program that counts occurrences of each pairs of lower-case letter of the English alphabet in a string and returns a list of lists (a table). The first list should be the count of 'a' followed by 'a's, 'b's, etc., second list should be the count of 'b' followed by 'a's, 'b's, etc. etc.

```
cooccurrence : src:string -> int list list
```

7.6. Write a program that generates a random string of length len, whose character pairs are distributed according to a user specified cooccurrence histogram cooc. The function must have the type:

```
fstOrderMarkovModel : cooc:int list list -> len:int -> string
```

Test your function by generating a random string, whose character pairs are distributed as the converted characters in H.C. Andersen's fairy tale, "Little Claus and Big Claus", calculate the cooccurrence histogram for the random string, and compare this with the original cooccurrence histogram.

You must also write a short report, which

- is no larger than 5 pages;
- contains a brief discussion on how your implementation works, and if there are any possible alternative implementations, and in case, why you chose the one, you did;
- includes output that demonstrates that your program works as intented.

The report is to be handed in as a pdf document together with the single F# source code as an fsx file.