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 Higher-order Functions

0.1.1 Teacher's guide

0.1.2 Introduction

De følgende opgaver omhandler højereordens funktioner. Specifikt handler opgaverne om brugen af pipe-funktionerne (`|>` og `<|`) samt af de højereordens funktioner, der kan benyttes til funktionssammensætning (>> og <<).

0.1.3 Exercise(s)

0.1.3.1: Skriv en funktion `sumsq : float list -> float`, som benytter sig af en `let`-binding til at definere en variabel `squares`. Variablen `squares` skal bindes til resultatet af at konstruere en liste indeholdende kvadratet på hvert element i argumentlisten (benyt `List.map`). Funktionen `sumsq` kan nu returnere resultatet af at udregne udtrykket `List.fold (+) 0.0 squares`. Test funktionen på en række forskellige float-lister. For eksempel skal kaldet `sumsq [3.0;4.0]` resultere i værdien `25.0`.

0.1.3.2: Opskriv typerne for de fire funktioner `|>`, `<|`, << og >>:

```
val |> : ____________________________________   // right-pipe

val <| : ____________________________________   // left-pipe

val << : ____________________________________   // left-compose

val >> : ____________________________________   // right-compose
```

0.1.3.3: Omdefinér funktionen `sumsq : float list -> float` således at den benytter sig af F#'s **right-pipe funktion** `|>` til at pipe resultatet af `List.map` udtrykket ind i `List.fold` udtrykket. Test funktionen på en række forskellige float-lister.

0.1.3.4: Omdefinér funktionen `sumsq : float list -> float` således at den benytter sig af F#'s **left-pipe funktion** `<|` til at pipe resultatet af `List.map` udtrykket ind i `List.fold` udtrykket, men fra højre mod venstre. Test funktionen på en række forskellige float-lister.

0.1.3.5: Omdefinér funktionen `sumsq : float list -> float` således at den benytter sig af F#'s **right-compose funktion** >> til at sammensætte (1) en funktion defineret ved at kalde `List.map` med en funktion samt (2) en funktion defineret ved at kalde `List.fold` med to argumenter. Definitionen skal følge formen:

```
let sumsq : float list -> float =
  ... >> ...
```

Test funktionen på en række forskellige float-lister.

0.1.3.6: Omdefinér funktionen `sumsq : float list -> float` således at den benytter sig af F#'s **left-compose funktion** << til at sammensætte (1) en funktion defineret ved at kalde `List.map`

med en funktion samt (2) en funktion defineret ved at kalde `List.fold` med to argumenter. Definitionen skal følge formen:

```
let sumsq : float list -> float =
  ... << ...
```

Bemærk at F#'s left-compose funktion svarer til matematisk funktionssammensætning givet ved $(f \circ g)(x) = f(g(x))$. Test funktionen på en række forskellige float-lister.

0.1.3.7: Betragt funktionen

```
let sumadd2sq (xs : float list) : float =
  let ys = List.map (fun x -> x*x) xs
  let zs = List.map (fun y -> y+2.0) ys
  in List.fold (+) 0.0 zs
```

Omskriv funktionen ved at benytte ligningen `map f << map g = map (f << g)` til at undgå at listen `ys` konstrueres. Omskriv funktionen yderligere ved enten at benytte en pipe-funktion eller funktionssammensætning til at undgå `let`-bindingen af variablen `zs`. Test funktionen på en række forskellige float-lister. For eksempel skal kaldet `sumadd2sq [2.0;3.0]` resultere i værdien `17.0`.