Assignment1 - Resubmission

Victor Vangkilde Jørgensen
kft410

October 12, 2024

1 Question 1 - Vectors

Consider vectors in the two-dimensional (2D) euclidian plane

1.a What type do you use to represent a 2D vector in F#? Why?

I denne opgave har jeg valgt, at repræsentere 2D vektorer som tupel, bestående af 2 floats som er vektorens koordinater. Dette valg kom nemt til, da jeg fortragte, at regne vektorer repræsenteret som tupels, frem for andre types.

Consider the three following operations on 2D vectors, defined in Danish:

- **Længde af vektorer**
- **Addition af vektorer**
- **Skalering af vektor**

1.b What names do you give to these three functions? Why?

Jeg ville navngive dem noget, som indikerer, at funktionerne har med vektorer at gøre, samt noget, der indikerer, at funktionerne gør noget ens. Eksempelvis:

- lengthVector
- addVector
- extVector

Her kan man se, at alle funktioner har med vektorer at gøre, og man kan få en ide om hvad hver enkel funktion gør. Derudover skriver man den spicifikke del af funktionsnavnet først, som gør det nemmere for autocomplete til at regne ud, hvad man gerne vil skrive for en af de tre funktioner.

1.c Write the specification of the three functions

- lengthVector

Vi vil gerne lave en funktion, der kan beregne længden af en vektor. Til dette vil det være værd at have en vektors koordinater som input i form af floats, der benyttes til at beregne vektorens længde i form af en float (vores output).

- addVector

Denne funktion kræver koordinatsættet af 2 vektorer. Disse koordinater vil jeg gerne definere i types floats for nøjagtighedens skyld (såvel som i de andre funktioner). Som output skal vi have koordinaterne til en vektor, som er resultatet af de 2 tidligere lagt sammen.

- extVector

Her vil vi gerne angive en faktor som argument til funktionen i form af type float, og en vektors koordinater i type floats. Funktionen skal multiplicere faktoren med koordinaterne, som skal give os en ny vektor, ved et output af 2 nye (eller de samme) koordinater.

1.d Give two examples (in F#) for each of the three vector functions

- lengthVector funktion og eksempler i F#

```
1 let lengthVector (a: float, b: float) : float =
2     sqrt (a ** 2 + b ** 2)
3 printfn "Length of vector: %f" (lengthVector (1.0, 1.0)) //prints:
     Length of vector: 1.414214
4 printfn "Length of vector: %f" (lengthVector (2.0, 1.0)) //prints:
     Length of vector: 2.236068
```

- addVector funktion og eksempler i F#

```
1 let addVector (a: float, b: float) (c: float, d: float) : float *
     float =
2     (a + c), (b + d)
3 printfn "Added vector: %A" (addVector (1.0, 2.0) (1.0, 2.0)) //
     prints: Added vector: (2.0, 4.0)
4 printfn "Added vector: %A" (addVector (4.0, 3.0) (-1.0, -2.0)) //
     prints: Added vector: (3.0, 1.0)
```

- extVector funktion og eksempler i F#

```
1 let extVector (factor: float) (a:float, b:float) : float * float =
2     (factor * a, factor * b)
3 printfn "Extended vector: %A" (extVector 2.0 (2.0, 1.0)) //prints:
     Extended vector: (4.0, 2.0)
4 printfn "Extended vector: %A" (extVector -1.0 (2.0, 1.0)) //prints:
     Extended vector: (-2.0, -1.0)
```

1.e Give the F# code for these functions and explain in detail how you design them

Alle mine konstruerede funktioner er opbygget af "let bindings", som er en form for struktur, der binder et navn med eventuelle parametre, samt en "function body", der indeholder hvad funktionen skal gøre, for at danne en output.
Jeg går mere i dybde om let bindings senere.

- lengthVector funktion og design

```
1 let lengthVector (a: float, b: float) : float =
2     sqrt (a ** 2 + b ** 2)
```

Under funktionens designprocess havde jeg overvejet hvordan jeg ville tage kvadraten af vektorens koordinater, som skulle bruges i Pythagoras formel til at bestemme længden af vektoren. Jeg tænkte på at definere en anden funktion til at bestemme kvadraten af en værdi, og jeg overvejede også at gange koordinaterne med sig selv, for at tage deres kvadrat, men så kom jeg i tankte om, at jeg blot kan skrive: "x ** 2", for at tage kvadraten.
Med dette i omtanke valgte jeg, at benytte floats frem for integers som inputs, da man dermed kan benytte decimaltal som vektorkoordinater. Som vist i koden ovenover, er outputten beregnet som: længde af vektor$(a, b) = \sqrt{a^2 + b^2}$, hvor sqrt tager kvadratroden af $a^2 + b^2$ eller $a**2 + b**2$

- addVector funktion og design

```
1 let addVector (a: float, b: float) (c: float, d: float) : float *
      float =
2      (a + c), (b + d)
```

For den næste funktion: "addVector", har jeg valgt, at opskrive fire inputs. De 2 første (a og b), er koordinaterne til den første vektor, som skal lægges til den anden vektor, med koordinaterne c og d. Læg mærke til, at vi igen arbejder med floats, da vi i så fald kan arbejde med flere forskellige mængder af tal.
Jeg har benyttet vektorregnereglen om addition af vektorer til at finde ud af, hvordan outputten bør beregnes. Det endelige design er som vist ovenover: $(a + c), (b + d)$, som angiver de 2 koordinater for den nye vektor.

- extVector funktion og design

```
1 let extVector (factor: float) (a:float, b:float) : float * float =
2      (factor * a, factor * b)
```

For den sidste funktion har vi ikke 2 vektorer, men i stedet en vektor og en faktor, som bliver ganget på. Jeg har derfor valgt at kalde denne faktor for "factor", så vi nemmere kan holde styr på vores inputs. Faktoren bliver ganget på hvert koordinat af vektoren, som giver vores skalerede vektor.

1.f Explain how your code would change if you were to represent 3D vector instead of 2D vectors?

Forklaringsmæssigt ville de største ændringer til koden komme fra måden man beregner de forskellige 3D vektorer. Vektorer i 3D-planen har tre koordinater i stedet for to, og vi bliver derfor nødt til, at forlænge vores definerede input og body i alle disse tilfælde. Eksempelvis bliver "extVector" funktionen nødt til at have tre inputs i form af vektorens koordinater, og body'en for functionen skal derfor også gange faktoren ind i endnu et led.

2 Question 2 - Survey

A survey with 4 questions is taken by students. For each question, a student picks one of 4 possible answers. Each student has a unique id

2.a Describe two possible ways to represent the survey answers in F#. Also, give F# code with two examples that illustrate the representations you have chosen

Ud fra en besvarelse skal vi kunne opbevare følgende: den studerendes id, besvarelse 1, besvarelse 2, besvarelse 3, besvarelse 4.
En simpel måde man kan få alt nødvendigt data nemt tilgængelig, er ved at benytte en liste:

```
1 //Survey structure example 1:
2 //[410; 4; 2; 1; 3]
3 //[411; 2; 1; 1; 1]
```

Her er besvarelsen struktureret, så vi får den studerendes id, og derefter personens besvarelse af survey'et, med svarmulighederne 1-4, alt sammen i form af integers.
Dog skal vi være opmærksomme på, at lister kun kan opbevare elementer af samme type (integers i det viste tilfælde). Hvis vi derfor gerne vil have en studerendes navn, eller hvis den svarende studerendes id indeholder bogstaver, er vi nødt til at gøre noget andet. Vi kan for eksempel bruge en tuple, der kan opbevare flere types på en gang.

```
1 //Survey structure example 2:
2 //("kft410", 4, 2, 1, 3)
3 //("kft411", 2, 1, 1, 1)
```

Som vi kan se, indeholder vores tuples flere forskellige types: (string, int, int, int, int).

2.b Design a function that counts the number of students that have given a specifc answer to a given question. For example: how many students have given answer 2 to question 3? Explain your design using Ken's method, include your F# code and describe it

Ken's method:

1. **Write a brief description of what the function should do**
2. **Find a name for the function**
3. **Write down test examples**
4. **Find out the type of inputs and outputs**
5. **Generate code for the function**

Vores funktion bør kunne indtage en liste med alle de studerendes svar, som er angivet i tuples, og give os antallet af studerende, der har svaret på en specifikt svarmulighed i et specifikt spørgsmål.
Vi kan eksempelvis kigge på, hvor mange studerende, der har svaret med svarmulighed 1 i spørgsmål 3.

```
1  //Random examples of student answers:
2      [
3          ("kft410", 2, 1, 3, 1)
4          ("kfc420", 3, 2, 4, 2)
5          ("rgb110", 4, 3, 1, 3)
6          ("lgb690", 1, 4, 2, 4)
7          ("lwt900", 2, 2, 1, 2)
8          ("kmc069", 4, 4, 4, 2)
9          ("ops710", 4, 1, 1, 4)
10         ("msw910", 4, 4, 4, 2)
11     ]
12
13 //Example for input and output for a function that counts student
       answers for question 3, answer option 1:
14 //Input should be any list of example student answers, like the one
       written above.
15 //Output should be the integer 3, since 3 students answered with
       option 1 in question 3.
```

Vores inputs er altså en liste af studerendes svar, hvilket spørgsmål og hvilken svarmulighed vi leder efter. Jeg vælger at kalde funktionen "questionSpicificAnswers", da funktionen bestemmer antalet af svar der er givet til et bestemt spørgsmål.
Med det i omtanke, kan vi nu udvikle vores funktion:

```
let questionSpicificAnswers (answerOption: int) (question: int)
    studentAnswers: int =
     studentAnswers
     |> List.filter (fun (_, q1, q2, q3, q4) ->
     match question with
     | 1 -> q1 = answerOption
     | 2 -> q2 = answerOption
     | 3 -> q3 = answerOption
     | _ -> q4 = answerOption)
     |> List.length
printfn "%d" (questionSpicificAnswers 1 3 [
         ("kft410", 2, 1, 3, 1)
         ("kfc420", 3, 2, 4, 2)
         ("rgb110", 4, 3, 1, 3)
         ("lgb690", 1, 4, 2, 4)
         ("lwt900", 2, 2, 1, 2)
         ("kmc069", 4, 4, 4, 2)
         ("ops710", 4, 1, 1, 4)
         ("msw910", 4, 4, 4, 2)
     ]) //prints: 3
```

Her har vi benyttet den tidligere liste af spørgeskemasvar i en pipeline, hvor vi for hver tuple, tager svaret til det spørgsmål, som vi leder efter, angivet med argumentet "question". Vi sammenligner nu hvert svar i det spørgsmål med den ønskede svarmulighed, angivet ved "answerOption". Ved brug af List.filter, kan vi gøre, så vi kun beholder de svar, der har samme integer som den ønskede svarmulighed. Vi tager derefter længden af listen med List.length.
Vi printer resultatet seperat for funktionen, da vi ikke vil blande computations og actions sammen.

2.c Design a function that returns the percentage of student answers for all answers to a given question. Explain your design using Ken's method, include your F# code and describe it

Funktionen bør returnere procentdelen af studerende, der har svaret med hhv. svarmulighed 1, 2, 3 og 4, til et givet spørgsmål fra spørgeskemaet. Jeg vælger at kalde funktionen for "questionPercentageAnswers", da funktionen retunerer procentdelen af de forskellige svarmuligheder til et givet spørgsmål, og fordi dette navn er releteret til den tidligere lavet "questionSpicificAnswers" funktion.
Funktionen skal eksempelvis kunne indtage den liste af studerendes svar fra tidligere, og returnere andelen af hver valgt svarmulighed i eksempelvis spørgsmål 4:

```
25.00%
12.50%
50.00%
12.50%
```

Jeg forventer, at have floats som output type i de fleste tilfælde ved brug af sådan en funktion, så jeg har også valgt at bruge floats i eksemplet ovenover.
Til selve funktionen har jeg valgt, at lave en recursive funktion, der indtager mængden af svarmuligheder i spørgsmålet (answerOption), hvilket spørgsmål vi gerne vil have resultaterne fra (question) i type int, samt listen med surveysvar:

```
let rec questionPercentageAnswers (answerOption: int) (question: int
    ) studentAnswers =
```

```
    match answerOption with
    | 0 -> ()
    | _ -> printfn "%.2f%% ansered with option %d in question %d" (
        float (
         studentAnswers
         |> List.filter (fun (_, q1, q2, q3, q4) ->
         match question with
         | 1 -> q1 = answerOption
         | 2 -> q2 = answerOption
         | 3 -> q3 = answerOption
         | _ -> q4 = answerOption)
         |> List.length) / (float (
             studentAnswers
             |> List.length)) * 100.0) answerOption question;
                questionPercentageAnswers (answerOption - 1)
                question studentAnswers
questionPercentageAnswers 4 4 [
         ("kft410", 2, 1, 3, 1)
         ("kfc420", 3, 2, 4, 2)
         ("rgb110", 4, 3, 1, 3)
         ("lgb690", 1, 4, 2, 4)
         ("lwt900", 2, 2, 1, 2)
         ("kmc069", 4, 4, 4, 2)
         ("ops710", 4, 1, 1, 4)
         ("msw910", 4, 4, 4, 2)
    ]
```

I koden ovenover, benytter vi recursion for hver svarmulighed, hvor vi først tester gennem brug af pattern matching, om recursion'en er gået "forbi" sidste svarmulighed. Hvis ikke, så bruger vi en pipeline af listen, som bliver reduceret til kun de svar, der matcher den iterering af recursion'en der findes sted, og angiver længden (antallet), som er vores antal af svar til svarmuligheden. Herefter laver vi en pipeline, hvor vi tager længden af vores originale liste, der gerne skulle matche, hvor mange studerende, der har svaret på spørgeskemaet. Vi dividerer den første værdi med den anden, og ganger med 100, for at få svaret i procent. Dette printes, og vi får:

```
25.00% ansered with option 4 in question 4
12.50% ansered with option 3 in question 4
50.00% ansered with option 2 in question 4
12.50% ansered with option 1 in question 4
```

2.d Design a function that find the ids of two students with the same answers. Explain your design using Ken's method, include your F# code and describe it

Vores funktion bør tjekke listen med hver studerendes svar til et givet spørgsmål, og printe dem, der har svaret med den samme svarmulighed i alle spørgsmål.
Jeg vælger, at kalde denne funktion for matchingStudents, da funktionen matcher id'en af de studerende, der har præcist samme svar.
Med samme spørgeskemasvar fra tidligere, bør vores output være id'en af følgende studerende:

```
kmc069 msw910
```

Da disse har angivet samme svar til alle spørgsmål, og fordi de er de eneste 2 personer, der har svaret det samme.
Med det sagt, kan vi nu lave vores funktion.

```
let rec matchingStudents (studentIndex: int) studentAnswers =
    if (studentIndex >= List.length studentAnswers) then
        ()
    else
        let filteredStudents =
            studentAnswers
                |> List.filter (fun (_, q1, q2, q3, q4) -> (q1, q2,
                    q3, q4) = (
                     studentAnswers[studentIndex]
                     |> fun (_, q1, q2, q3, q4) -> (q1, q2, q3, q4)))
        if List.length filteredStudents = 2 then
            filteredStudents
            |> List.iter (fun (id, _, _, _, _) -> printf "%s " id)
        else
            matchingStudents (studentIndex + 1) studentAnswers
matchingStudents 0 [
        ("kft410", 2, 1, 3, 1)
        ("kfc420", 3, 2, 4, 2)
        ("rgb110", 4, 3, 1, 3)
        ("lgb690", 1, 4, 2, 4)
        ("lwt900", 2, 2, 1, 2)
        ("kmc069", 4, 4, 4, 2)
        ("ops710", 4, 1, 1, 4)
        ("msw910", 4, 4, 4, 2)
    ]
//prints:
//kmc069 msw910
```

Her benytter vi endnu en gang recursion til at iterere gennem hver persons svar i survey'et. Vi benytter pattern matching til at stoppe recursion'en, når vi kommer til svarmulighed 0, som ikke er en gyldig svarmulighed i min spørgeskemastruktur. For alle andre svarmuligheder gælder det, at vi bruger List.filter til at tjekke, om der er præcist 2 studerende, der har svart det samme. Hvis dette er sandt, printer vi første element af deres tupel, som indeholder deres id, i form af en string.

3 Question 3 - Concepts and syntax

3.a What is a function?

En funktion består af en let binding, funktionsnavn, parametre og en function body. Gennem en let binding, er vi i stand til at lave en funktion, der kan kaldes ved dens valgte navn. Når en funktion kaldes, vil "stedet" hvor den bliver kaldt, blive erstattet med funktionens body, hvor parametreværdierne indsættes i de samme steder, som parametrene er skrevet i funktionsdefinationen.
Et eksempel på en funktion kunne være:

```
let plus (x: int) (y: int) = x + y
```

Her benytter vi en let binding. Funktionens navn er "plus", argumenterne er "x" og "y", begge af type integer. Funktionsbody'en er x + y.

3.b What is type inference?

Type inference er hvordan et sprog håndterer værdiers types. Nogle gange, når jeg selv angiver værdier til en parameter, såsom:

```
n = 1
```

Vil F# være i stand til at se, at n er af type integer.
Dette findes ikke sted når jeg selv definerer hvilken type jeg benytter.

```
1  (n: int) = 1
```

Her definerer jeg selv hvilken type n består af. Type inference findes derfor ikke sted, da en relevant type er blevet angivet.

3.c What are the advantages of functional programming based on what you have seen during the exercises?

Set fra synsvinkelen som en, der kun har programmeret i objekt-orienteret programmeringssprog. Er funktionel programmering mere "frit". Med det mener jeg, at man har i større omfang mulighed for, at vælge hvordan ens kode bliver opbygget. F# har eksempelvis stærk understøttelse for pattern matching, som gør det muligt at skrive klar og kompakt kode. Det gør det lettere at håndtere forskellige mulige udfald og reducere behovet for if-else-kæder. Generelt set er F# et meget kompakt sprog, da alt er en expression. Vi kan derfor benytte statements som inputs i funktioner, som giver et nyt output.