# Programmering og Problemløsning
# Datalogisk Institut, Københavns Universitet

Jon Sporring

October 18, 2018

# recursion

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Rekursion

**Sværhedsgrad** Middel

## 2 Introduktion

## 3 Opgave(r)

1. Skriv en funktion, `fac : n:int -> int`, som udregner fakultetsfunktionen $n! = \prod_{i=1}^{n} i$ vha. rekursion.

2. Skriv en funktion, `sum : n:int -> int`, som udregner summen $\sum_{i=1}^{n} i$ vha. rekursion. Lav en tabel som i Opgave 3i.0 og sammenlign denne implementation af `sum` med `while`-implementation og `simpleSum`.

3. Skriv en funktion, `sum : int list -> int`, som tager en liste af heltal og returnerer summen af alle tallene. Funktionen skal traversere listen vha. rekursion.

4. Den største fællesnævner mellem 2 heltal, $t$ og $n$, er det største heltal $c$, som går op i både $t$ og $n$ med 0 til rest. Euclids algoritme[1] finder den største fællesnævner vha. rekursion:

$$\gcd(t,0) = t, \tag{1}$$
$$\gcd(t,n) = \gcd(n, t \,\%\, n), \tag{2}$$

   hvor % er rest operatoreren (som i F#).

   (a) Implementer Euclids algoritm, som den rekursive funktion

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

   (b) lav en white- og black-box test af den implementerede algoritme,

   (c) Lav en håndkøring af algoritmen for `gcd 8 2` og `gcd 2 8`.

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

5. Lav dine egen implementering af `List.fold` og `List.foldback` ved brug af rekursion.

6. Benyt `List.fold` og `List.foldback` og dine egne implementeringer fra Opgave 5 til at udregne summen af listen `[0 .. n]`, hvor `n` er et meget stort tal, og sammenlign tiden, som de fire programmer tager. Diskutér forskellene.

# continuedFractions

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Rekursion

**Sværhedsgrad** Middel

## 2 Introduktion

I denne opgave skal I regne med kædebrøker (continued fractions)[1]. Kædebrøker er lister af heltal, som repræsenterer reelle tal. Listen er endelig for rationelle og uendelig for irrationelle tal.

En kædebrøk skrives som: $x = [q_0; q_1, q_2, \ldots]$, hvilket svarer til tallet,

$$x = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \ldots}}. \tag{1}$$

F.eks. listen $[3; 4, 12, 4]$ evaluerer til

$$\begin{aligned} x &= 3 + \frac{1}{4 + \frac{1}{12 + \frac{1}{4}}} && (2)\\ &= 3 + \frac{1}{4 + \frac{1}{12.25}} && (3)\\ &= 3 + \frac{1}{4.081632653} && (4)\\ &= 3.245. && (5) \end{aligned}$$

Omvendt, for et givet tal $x$ kan kædebrøken $[q_0; q_1, q_2, \ldots]$ udregnes ved følgende algoritme: For $x_0 = x$ og $i \geq 0$ udregn $q_i = \lfloor x_i \rfloor$, $r_i = x_i - q_i$ og $x_{i+1} = 1/r_i$ indtil $r_i = 0$. Nedenfor ses en udregning for tallet $x = 3.245$:

[1] `https://en.wikipedia.org/wiki/Continued_fraction`

| $i$ | $x_i$ | $q_i$ | $r_i$ | $1/r_i$ |
|---|---|---|---|---|
| 0 | 3.245 | 3 | 0.245 | 4.081632653... |
| 1 | 4.081632653... | 4 | 0.081632653 | 12.25 |
| 2 | 12.25 | 12 | 0.25 | 4 |
| 3 | 4 | 4 | 0 | - |

Resultatet aflæses i anden søjle: $[3;4,12,4]$.

Kædebrøker af heltalsbrøkker $t/n$ er særligt effektive at udregne vha. Euclids algoritme for største fællesnævner. Algoritmen i **??** regner rekursivt på relationen mellem heltalsdivision og rest: Hvis $a = t$ div $n$ er heltalsdivision mellem $t$ og $n$, og $b = t$ % $n$ er resten efter heltalsdivision, så er $t = an+b$. Man kan nu forestille sig $t$, $n$ og $b$ som en sekvens af heltal $r_i$, hvor

$$r_{i-2} = q_i r_{i-1} + r_i, \tag{6}$$

$$r_i = r_{i-2} \ \% \ r_{-1} \quad \text{(rest ved heltalsdivision)}, \tag{7}$$

$$q_i = r_{i-2} \text{ div } r_{i-1} \quad \text{(heltalsdivision)}, \tag{8}$$

Hvis man starter sekvensen med $r_{-2} = t$ og $r_{-1} = n$ og beregninger resten iterativt indtil $r_{i-1} = 0$, så vil største fællesnævner mellem $t$ og $n$ være lig $r_{i-2}$, og $[q_0;q_1,\ldots,q_j]$ vil være $t/n$ som kædebrøk. Til beregning af største fællesnævner regner algoritmen i **??** udelukkende på $r_i$ som transformationen $(r_{i-2},r_{i-1}) \to (r_{i-1},r_i) = (r_{i-1}, r_{i-2} \ \% \ r_{-1})$ indtil $(r_{i-2},r_{i-1}) = (r_{i-2},0)$. Tilføjer man beregning af $q_i$ i rekursionen, har man samtidigt beregnet brøkken som kædebrøk. Nedenfor ses en udregning for brøken 649/200:

| $i$ | $r_{i-2}$ | $r_{i-1}$ | $r_i = r_{i-2} \ \% \ r_{-1}$ | $q_i = r_{i-2}$ div $r_{i-1}$ |
|---|---|---|---|---|
| 0 | 649 | 200 | 49 | 3 |
| 1 | 200 | 49 | 4 | 4 |
| 2 | 49 | 4 | 1 | 12 |
| 3 | 4 | 1 | 0 | 4 |
| 4 | 1 | 0 | - | - |

Kædebrøkken aflæses som højre søjle: $[3;4,12,4]$.

Omvendt, approksimationen af en kædebrøk som heltalsbrøken $\frac{t_i}{n_i}, i \geq 0$ fås ved

$$t_i = q_i t_{i-1} + t_{i-2}, \tag{9}$$

$$n_i = q_i n_{i-1} + n_{i-2}, \tag{10}$$

$$t_{-2} = n_{-1} = 0, \tag{11}$$

$$t_{-1} = n_{-2} = 1, \tag{12}$$

Alle approximationerne for $[3;4,12,4]$ er givet ved,

$$\frac{t_0}{n_0} = \frac{q_0 t_{-1} + t_{-2}}{q_0 n_{-1} + n_{-2}} = \frac{3 \cdot 1 + 0}{3 \cdot 0 + 1} = \frac{3}{1} = 3, \tag{13}$$

$$\frac{t_1}{n_1} = \frac{q_1 t_0 + t_{-1}}{q_1 n_0 + n_{-1}} = \frac{4 \cdot 3 + 1}{4 \cdot 1 + 0} = \frac{13}{4} = 3.25, \tag{14}$$

$$\frac{t_2}{n_2} = \frac{q_2 t_1 + t_0}{q_2 n_1 + n_0} = \frac{12 \cdot 13 + 3}{12 \cdot 4 + 1} = \frac{159}{49} = 3.244897959\ldots, \tag{15}$$

$$\frac{t_3}{n_3} = \frac{q_3 t_2 + t_1}{q_3 n_2 + n_1} = \frac{4 \cdot 159 + 13}{4 \cdot 49 + 4} = \frac{649}{200} = 3.245. \tag{16}$$

Bemærk at approximationen nærmer sig 3.245 når $i$ vokser.

# 3 Opgave(r)

1. Skriv en rekursiv funktion

   ```
   cfrac2float : lst:int list -> float
   ```

   som tager en liste af heltal som kædebrøk og udregner det tilsvarende reelle tal.

2. Skriv en rekursiv funktion

   ```
   float2cfrac : x:float -> int list
   ```

   som tager et reelt tal og udregner dens repræsentation som kædebrøk.

3. Skriv en rekursiv funktion

   ```
   frac2cfrac : t:int -> n:int -> int list
   ```

   som tager tæller og nævner i brøken $t/n$ og udregner dens repræsentation som kædebrøk udelukkende ved brug af heltalstyper.

4. Skriv en rekursiv funktion

   ```
   cfrac2frac : lst:int list -> i:int -> int * int
   ```

   som tager en kædebrøk og et index og returnerer $t_i/n_i$ approximationen som tuplen `(ti, ni)`.

5. Skriv en white- og black-box test af de ovenstående funktioner.

# sort

Jon Sporring

October 18, 2018

# 1 Lærervejledningn

**Emne** Mønstergenkendelse og black-box test

**Sværhedsgrad** Let

# 2 Introduktion

# 3 Opgave(r)

1. Omskriv funktionen `insert`, som benyttes i forbindelse med funktionen `isort` (insertion sort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

2. Omskriv funktionen `bsort` (bubble sort) fra forelæsningen således at den benytter sig af pattern matching på lister. Funktionen kan passende benytte sig af "nested pattern matching" i den forstand at den kan implementeres med et match case der udtrækker de to første elementer af listen samt halen efter disse to elementer.

3. Opskriv black-box tests for de to sorteringsfunktioner og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

4. Omskriv funktionen `merge`, som benyttes i forbindelse med funktionen `msort` (mergesort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

5. Opskriv black-box tests for sorteringsfunktionen `msort` og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

# imgutil

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** rekursion, grafik og winforms

**Sværhedsgrad** Middel

## 2 Introduktion

## 3 Opgave(r)

1. Ved at benytte biblioteket `ImgUtil`, som beskrevet i forelæsningen, er det muligt at tegne simpel liniegrafik samt fraktaler, som f.eks. Sierpinski-fraktalen, der kan tegnes ved at tegne små firkanter bestemt af et rekursivt mønster. Koden for Sierpinski-trekanten er givet som følger:

```
open ImgUtil

let rec triangle bmp len (x,y) =
  if len < 25 then setBox blue (x,y) (x+len,y+len) bmp
  else let half = len / 2
       do triangle bmp half (x+half/2,y)
       do triangle bmp half (x,y+half)
       do triangle bmp half (x+half,y+half)

do runSimpleApp "Sierpinski" 600 600 (fun bmp -> triangle bmp 512
  (30,30) |> ignore)
```

   Tilpas funktionen således at trekanten tegnes med røde streger samt således at den kun tegnes ned til dybde 2 (hint: du skal ændre betingelsen `len < 25`).

2. I stedet for at benytte `ImgUtil.runSimpleApp` funktionen skal du nu benytte `ImgUtil.runApp`, som giver mulighed for at din løsning kan styres ved brug af tastaturet. Funktionen `ImgUtil` har følgende type:

```
val runApp : string -> int -> int
          -> (int -> int -> 's -> System.Drawing.Bitmap)
          -> ('s -> System.Windows.Forms.KeyEventArgs
```

```
                    -> 's option)
          -> 's -> unit
```

De tre første argumenter til `runApp` er vinduets titel (en streng) samt vinduets initielle vidde og højde. Funktionen `runApp` er parametrisk over en brugerdefineret type af tilstande (`'s`). Antag at funktionen kaldes som følger:

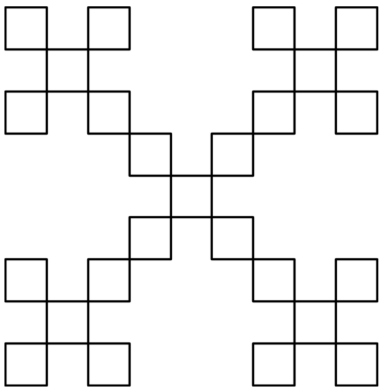
```
runApp title width height draw react init
```

Dette kald vil starte en GUI applikation med titlen `title`, vidden `width` og højden `height`. Funktionen `draw`, som brugeren giver som 4. argument kaldes initielt når applikationen starter og hver gang vinduets størrelse justeres eller ved at funktionen `react` er blevet kaldt efter en tast er trykket på tastaturet. Funktionen `draw` modtager også (udover værdier for den aktuelle vidde og højde) en værdi for den brugerdefinerede tilstand, som initielt er sat til værdien `init`. Funktionen skal returnere et bitmap, som for eksempel kan konstrueres med funktionen `ImgUtil.mk` og ændres med andre funktioner i `ImgUtil` (f.eks. `setPixel`).

Funktionen `react`, som brugeren giver som 5. argument kaldes hver gang brugeren trykker på en tast. Funktionen tager som argument en værdi svarende til den nuværende tilstand for applikationen samt et argument der kan benyttes til at afgøre hvilken tast der blev trykket på.[1] Funktionen kan nu (eventuelt) ændre på dens tilstand ved at returnere en ændret værdi for denne.

Tilpas applikationen således at dybden af fraktalen kan styres ved brug af piletasterne, repræsenteret ved værdierne `System.Windows.Forms.Keys.Up` og `System.Windows.Forms.Keys.Down`.

3. Med udgangspunkt i øvelsesopgave 1 skal du i denne opgave implementere en GUI-applikation der kan tegne en version af X-fractalen som illustreret nedenfor (eventuelt i en dybde større end 2).



Bemærk at det ikke er et krav at dybden på fraktalen skal kunne styres med piletasterne som det er tilfældet med Sierpinski-fraktalen i øvelsesopgave 2.

[1] Hvis `e` har typen `System.Windows.Forms.KeyEventArgs` kan betingelsen `e.KeyCode = System.Windows.Forms.Keys.Up` benyttes til at afgøre om det var tasten "Up" der blev trykket på.

# weekday

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Sumtyper og lister

**Sværhedsgrad** Let

## 2 Introduktion

I de næste to opgaver skal følgende sum-type benyttes til at repræsentere ugedage:

```
type weekday = Monday | Tuesday | Wednesday | Thursday
             | Friday | Saturday | Sunday
```

## 3 Opgave(r)

1. Lav en funktion `dayToNumber : weekday -> int`, der givet en ugedag returnerer et tal, hvor mandag skal give tallet 1, tirsdag tallet 2 osv.

2. Lav en funktion `nextDay : weekday -> weekday`, der givet en ugedag returnerer den næste dag, så mandag skal give tirsdag, tirsdag skal give onsdag, osv, og søndag skal give mandag.

3. Givet typen for ugedage øverst på denne ugeseddel, lav en funktion `numberToDay : int -> weekday option`, sådan at `numberToDay` $n$ returnerer `None`, hvis $n$ ikke ligger i intervallet $1\ldots7$, og returnerer `Some` $d$, hvor $d$ er den til $n$ hørende ugedag, hvis $n$ ligger i intervallet $1\ldots7$.

   Det skulle gerne gælde, at `numberToDay (dayToNumber` $d$`)` $\rightsquigarrow$ `Some` $d$ for alle ugedage $d$.

# sudoku

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Funktionsprogrammering

**Sværhedsgrad** Hård

## 2 Introduktion

Temaet for ugernes opgaver er at programmere et Sudoku-spil. *Sudoku* er et puslespil, som er blevet opfundet uafhængigt flere gange; den tidligste "ægte" version af sudoku synes at kunne spores tilbage til det franske dagblad *Le Siècle* i 1892.

Vi betragter her kun den basale variant, som spilles på en matrix af 81 små felter, arrangeret i 9 rækker og 9 søjler. Matricen er desuden inddelt i 9 "bokse" eller "regioner", hver med 3 gange 3 felter.

Nogle af felterne er udfyldt på forhånd, og puslespillet går ud på at udfylde de resterende felter på en sådan måde, at hver af de 9 rækker, hver af de 9 søjler og hver af de 9 regioner kommer til at indeholde en permutation af symbolerne fra et forelagt alfabet af størrelse 9; vi vælger her (som man plejer at se det) alfabetet bestående af cifrene fra 1 til 9.

Her er en lovlig starttilstand for et spil sudoku:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Og følgende er en vindende tilstand (en "løsning") af ovenstående:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

## Nummerering og filformat

Lad os nummerere rækkerne $r = 0, 1, \ldots, 8$ ovenfra og ned og søjlerne $s = 0, 1, \ldots, 8$ fra venstre mod højre. Også regionerne vil vi nummerere $q = 0, 1, \ldots, 8$, i "normal læseretning" (for vestlige sprog).

Sammenhængen mellem rækkenummer $r$, søjlenummer $s$ og regionsnummer $q$ kunne så udtrykkes i følgende formel ved brug af heltalsoperationer:

- Feltet i række nummer $r$ og søjle nummer $s$ vil ligge i region nummer

$$q = r \texttt{ / 3 * 3 + } s \texttt{ / 3} \tag{1}$$

- Region nummer $q$ består af felterne

$$(r,s) = (q \texttt{ / 3 * 3 + } m, q \texttt{ \% 3 * 3 + } n)\,;\; m,n \in \{0,1,2\}. \tag{2}$$

En spiltilstand kan gemmes i en fil, ved at antage, at indholdet *altid* ser ud som følger:

- Der er mindst 90 tegn i filen (der kan være flere, men vi er kun interesseret i de 90 første)
- De første 90 tegn i filen er delt op i 9 grupper, som repræsenterer indholdet af række 0,..., 8 i nævnte rækkefølge. Hver gruppe består af 10 tegn: Først 9 tegn, som er et blandt `'1'`, ..., `'9'`, `'*'`, og til sidst strengen `"\n"`.

F.eks. er nedenstående indholdet i en fil, som indeholder starttilstanden for ovenstående sudoku:

```
53**7****\n6**195***\n*98****6*\n8***6***3\n4**8*3**1\n7***2***6\n*6****28*\n***419**5\n****8**79\n
```

Hvis vi fortolker strengen `"\n"` som "ny linje", bliver ovenstående lettere at læse:

```
53**7****
6**195***
*98****6*
8***6***3
4**8*3**1
7***2***6
*6****28*
***419**5
****8**79
```

# 3 Opgave(r)

I skal programmere et Sudoku spil og skrive en rapport. Afleveringen skal bestå af en pdf indeholdende rapporten, et katalog med et eller flere fsharp programmer som kan oversættes med Monos fsharpc kommando og derefter køres i mono, og en tekstfil der angiver sekvensen af oversættelseskommandoer nødvendigt for at oversætte jeres program(mer). Kataloget skal zippes og uploades som en enkelt fil. Kravene til programmeringsdelen er:

1. Programmet skal kunne læse en (start-)tilstand fra en fil.

2. Brugeren skal kunne indtaste filnavnet for (start-)tilstanden
3. Brugeren skal kunne indtaste triplen $(r,s,v)$, og hvis feltet er tomt og indtastningen overholder spillets regler, skal matrixen opdateres, og ellers skal der udskrives en fejlmeddelelse på skærmen
4. Programmet skal kunne skrive matricens tilstand på skærmen (på en overskuelig måde)
5. Programmet skal kunne foreslå lovlige tripler $(r,s,v)$.
6. Programmet skal kunne afgøre, om spillet er slut.
7. Brugeren skal have mulighed for at afslutte spillet og gemme tilstanden i en fil.
8. Programmet skal kommenteres ved brug af fsharp kommentarstandarden
9. Programmet skal struktureres ved brug af et eller flere moduler, som I selv har skrevet
10. Programmet skal unit-testes

Kravene til rapporten er:

11. Rapporten skal skrives i LaTeX.
12. I skal bruge `rapport.tex` skabelonen
13. Rapporten skal som minimum i hoveddelen indeholde afsnittene Introduktion, Problemformulering, Problemanalyse og design, Programbeskrivelse, Afprøvning, og Diskussion og Konklusion. Som bilag skal I vedlægge afsnittene Brugervejledning og Programtekst.
14. Alle gruppemedlemmer skal give feedback på et af hovedafsnittene i en anden gruppes rapport. Hvis og hvilke dele I gav feedback og hvem der gav feedback på jeres rapport skal skrives i Forordet i rapporten.
15. Rapporten må maximalt være på 20 sider alt inklusivt.

Bemærk, at Sudoku eksemplerne i denne tekst er sat med LaTeX-pakken `sudoku`.

# mastermind

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Funktionsprogrammering

**Sværhedsgrad** Hård

## 2 Introduktion

Spillet Mastermind er et spil for 2 deltagere, en opgavestiller og en opgaveløser. Opgavestilleren laver en skjult opgave bestående af en kombination af 4 farvede opgavestifter i ordnet rækkefølge, hvor farverne kan være rød, grøn, gul, lilla, hvid og sort. Opgaveløseren skal nu forsøge at gætte opgave. Dette gøres ved, at opgaveløseren foreslår en kombination af farver, og til hvert forslag svarer opgavestilleren med et antal hvide og sorte Svarstifter. Antallet af sorte Svarstifter svarer til hvor mange af opgavestifterne, som havde den rigtige farve og er på den rette plads, og antallet af hvide Svarstifter svarer til antallet af opgavestifter, som findes i opgaven men på den forkerte plads. F.eks. hvis opgaven er (rød, sort, grøn, gul), og gættet er (grøn, sort, hvid, hvid) er svaret 1 sort og 1 hvid.

## 3 Opgave(r)

I skal programmere spillet Mastermind. Minimumskrav til jeres aflevering er:

- Det skal være muligt at spille bruger mod bruger, program mod bruger i valgfrie roller og program mod sig selv.
- Programmet skal kommunikere med brugeren på engelsk.
- Programmet skal bruge følgende typer:

```
type codeColor =
  Red | Green | Yellow | Purple | White | Black
type code = codeColor list
```

```
type answer = int * int
type board = (code * answer) list
type player = Human | Computer
```

hvor `codeColor` er farven på en opgavestift; `code` er en opgave bestående af 4 opgavestifter; `answer` er en tuple hvis første element er antallet af hvide og andet antallet af sorte stifter; og `board` er en tabel af sammenhørende gæt og svar.

- Programmet skal indeholde følgende funktioner:

```
makeCode : player -> code
```

som tage en spillertype og returnerer en opgave enten ved at få input fra brugeren eller ved at beregne en opgave.

```
guess : player -> board -> code
```

som tager en spillertype, et bræt bestående af et spils tidligere gæt og svar og returnerer et nyt gæt enten ved input fra brugeren eller ved at programmet beregner et gæt.

```
validate : code -> code -> answer
```

som tager den skjulte opgave og et gæt og returnerer antallet af hvid og sort svarstifter.

- Programmet skal kunne spilles i tekst-mode dvs. uden en grafisk brugergrænseflade.
- Programmet skal dokumenteres efter fsharp kodestandarden
- Programmet skal afprøves
- Opgaveløsningen skal dokumenteres som en rapport skrevet i LaTeX på maksimalt 20 oversatte sider ekslusiv bilag, der som minimum indeholder
  - En forside med en titel, dato for afleveringen og jeres navne
  - En forord som kort beskriver omstændighederne ved opgaven
  - En analyse af problemet
  - En beskrivelse af de valg, der er foretaget inklusiv en kort gennemgang af alternativerne
  - En beskrivelse af det overordnede design, f.eks. som pseudokode
  - En programbeskrivelse
  - En brugervejledning
  - En beskrivelse af afprøvningens opbygning
  - En konklusion
  - Afprøvningsresultatet som bilag
  - Programtekst som bilag

Gode råd til opgave er:

- Det er ikke noget krav til programmeringsparadigme, dvs. det står jer frit for om I bruger funktional eller imperativ programmeringsparadigme, og I må også gerne blande. Men overvej i hvert tilfælde hvorfor I vælger det ene fremfor det andet.

- For programmet som opgaveløser er det nyttigt at tænke over følgende: Der er ikke noget "intelligens"-krav, så start med at lave en opgaveløser, som trækker et tilfældigt gæt. Hvis I har mod på og tid til en mere advanceret strategi, så kan I overveje, at det totale antal farvekombinationer er $6^4 = 1296$, og hvert afgivne svar begrænser de tilbageværende muligheder.

- Summen af det hvide og sorte svarstifter kan beregnes ved at sammenlinge histogrammerne for de farvede stifter i hhv. opgaven og gættet: F.eks. hvis opgaven består af 2 røde og gættet har 1 rød, så er antallet af svarstifter for den røde farve 1. Ligeledes, hvis opaven består af 1 rød, og gættet består af 2 røde, så er antallet af svarstifter for den røde farve 1. Altså er antallet af svarstifter for en farve lig minimum af antallet af farven for opgaven og gættet for den givne farve, og antallet af svarstifter for et gæt er summen af minima over alle farver.

- Det er godt først at lave et programdesign på papir inden I implementerer en løsning. F.eks. kan papirløsningen bruges til i grove træk at lave en håndkøring af designet. Man behøver ikke at have programmeret noget for at afprøve designet for et antal konkrete situationer, såsom "hvad vil programmet gøre når brugeren er opgaveløser, opgaven er (rød, sort, grøn, gul) og brugeren indtaster (grøn, sort, hvid, hvid)?"

- Det er ofte sundt at programmere i cirkler, altså at man starter med at implementere en skald, hvor alle de væsentlige dele er tilstede, og programmet kan oversættes (kompileres) og køres, men uden at alle delelementer er færdigudviklet. Derefter går man tilbage og tilføjer bedre implementationer af delelementer som legoklodser.

- Det er nyttigt at skrive på rapporten under hele forløbet i modsætning til kun at skrive på den sidste dag.

- I skal overveje detaljeringsgraden i jeres rapport, da I ikke vil have plads til alle detaljer, og I er derfor nødt til at fokusere på de vigtige pointer.

- Husk at rapporten er et produkt i sig selv: hvis vi ikke kan læse og forstå jeres rapport, så er det svært at vurdere dens indhold. Kør stavekontrol, fordel skrive- og læseopgaverne, så en anden, end den der har skrevet et afsnit, læser korrektur på det.

- Det er bedre at aflevere noget end intet.

- Der er afsat 1 undervisningsfri og 3/2 alm. undervisningsuger til opgaven (7/11-30/11 fraregnet 14/11-20/11, som er mellemuge, og kursusaktiveter på parallelkurser). Det svarer til ca. 40 timers arbejde. Brug dem struktureret og målrettet. Lav f.eks. en tidsplan, så I ikke taber overblikket over projektforløbet.

# triangles

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Sumtyper

**Sværhedsgrad** Middel

## 2 Introduktion

Til de følgende opgaver udvider vi typen `figure` med en mulighed for at repræsentere trekanter:

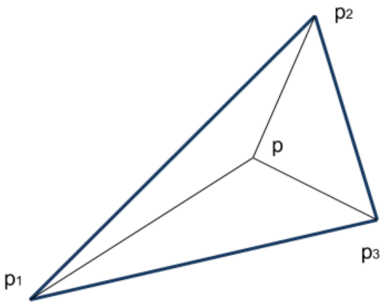
```
type figure =
  | Circle of point * int * colour
     // defined by center, radius, and colour
  | Rectangle of point * point * colour
     // defined by corners bottom-left, top-right, and colour
  | Mix of figure * figure
     // combine figures with mixed colour at overlap
  | Triangle of point * point * point * colour
     // defined by the three points and colour
```

Konstruktøren `Triangle` tager tre punkter og en farve som argument. Der er intet krav til hvordan de tre punkter er placeret i forhold til hinanden.

For at bestemme hvorvidt et punkt er placeret inde i en trekant benytter vi os af et trick der forudsætter at vi kan beregne arealet af en trekant ved at kende dens hjørnepunkter. Det viser sig at hvis en trekant er bestemt af punkterne $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ og $p_3 = (x_3, y_3)$ vil følgende relativt simple formel kunne benyttes til at udregne arealet:

$$area = \left| \frac{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}{2} \right|$$

Tricket som vi nu skal benytte til at afgøre om et punkt $p$ ligger inden i en trekant udspændt af hjørnerne $p_1$, $p_2$ og $p_3$ forklares lettest ved at iagttage figuren til højre. Såfremt arealet af de tre trekanter $(p_1, p_2, p)$, $(p_2, p_3, p)$, og $(p_1, p_3, p)$ tilsammen er større end arealet af trekanten $(p_1, p_2, p_3)$, da ligger punktet $p$ udenfor trekanten $(p_1, p_2, p_3)$; ellers ligger punktet indenfor trekanten.




# 3 Opgave(r)

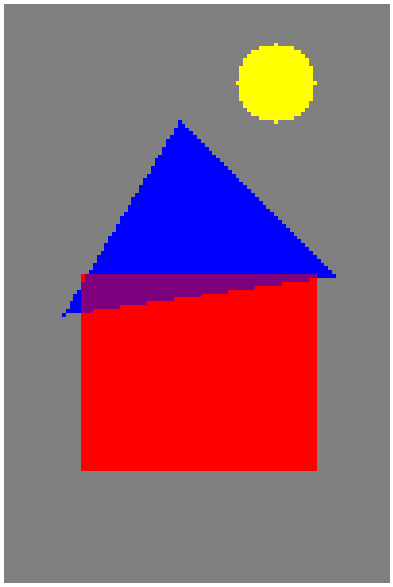
1. Lav en figur `figHouse : figure`, som består af en rød firkant udspændt af punkterne (20,70) og (80,120), en blå trekant udspændt af punkterne (15,80), (45,30) og (85,70), samt en gul cirkel med centrum (70,20) og radius 10.

2. Skriv en F# funktion `triarea2` der kan beregne den dobbelte værdi af arealet af en trekant udfra dens tre hjørnepunkter ved at benytte formlen ovenfor. Funktionen skal tage hjørnepunkterne som argumenter og have typen `point -> point -> point -> int`. Test funktionen på et par simple trekanter.[1]

3. Udvid funktionen `colourAt` til at håndtere trekantsudvidelsen ved at implementere tricket nævnt ovenfor samt ved at benytte den implementerede funktion `triarea2`.

4. Lav en fil `figHouse.png`, der viser figuren `figHouse` i et $100 \times 150$ bitmap.

   Resultatet skulle gerne ligne figuren nedenfor.



5. Udvid funktionerne `checkFigure` og `boundingBox` fra øvelsesopgaverne til at håndtere udvidelsen.

   `boundingBox houseFig` skulle gerne give `((15, 10), (85, 120))`.

[1] Det viser sig at være hensigtsmæssigt at undgå divisionen med 2, som kan forårsage uheldige afrundingsfejl.

# awari

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Typer, lister, mønstergenkendelse, funktionsprogrammering

**Sværhedsgrad** Hård

## 2 Introduktion

I denne opgave skal I programmere spillet Awari, som er en variant af Kalaha. Awari er et gammelt spil fra Afrika, som spilles af 2 spillere, med 7 pinde og 36 bønner. Pindene lægges så der dannes 14 felter ('pits' på engelsk), hvoraf 2 er hjemmefelter. Bønnerne fordeles ved spillet start med 3 i hvert felt pånær i hjemmefelterne. Startopstillingen er illustreret i Figur 1.

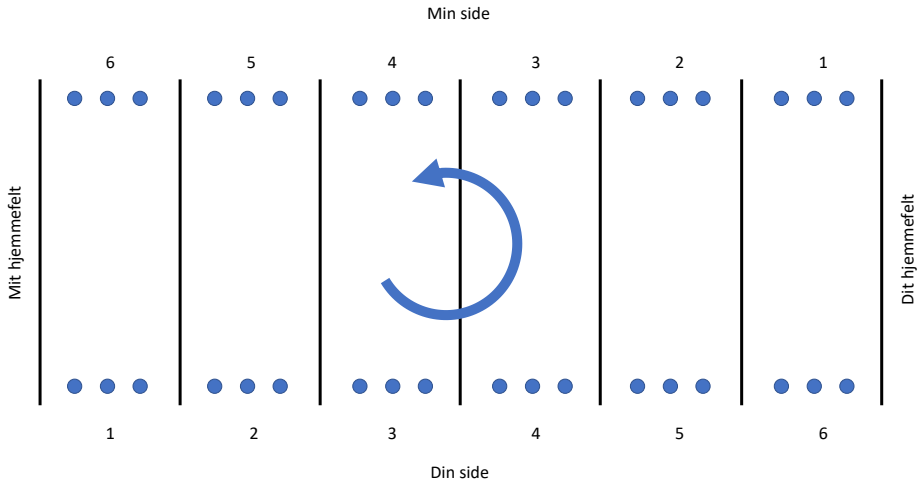



Figure 1: Udgangsopstillingen for spillet Awari.

Spillerne skiftes til at spille en tur efter følgende regler:

- En tur spilles ved at spilleren tager alle bønnerne i et af spillerens felter 1-6 og placerer dem i de efterfølgende felter inkl. hjemmefelterne en ad gangen og mod uret. F.eks., kan første spiller vælge at tage bønnerne fra felt 4, hvorefter spilleren skal placere en bønne i hver af felterne 5, 6 og hjemmefeltet.
- Hvis sidste bønne lægges i spillerens hjemmefelt, får spilleren en tur til.
- Hvis sidste bønne lander i et tom felt som ikke er et hjemmefelt, og feltet overfor indeholder bønner, så flyttes sidste bønne til spillerens hjemmefelt, og alle bønnerne overfor fanges og flyttes ligså til hjemmefeltet.
- Spillet er slut når en af spillerne ingen bønner har i sine felter 1-6, og vinderen er den spiller, som har flest bønner i sit hjemmefelt.

# 3 Opgave(r)

1. I skal implementere spillet Awari, som kan spilles af 2 spillere, og skrive en kort rapport. Kravene til jeres aflevering er:
    - Koden skal organiseres som bibliotek, en applikation og en test-applikation.
    - Biblioteket skal tage udgangspunkt i følgende signatur- og implementationsfiler:

**Listing 1 awariLibIncompleteLowComments.fsi:**
**En ikke færdigskrevet signaturfil.**

```
module Awari
type pit = // intentionally left empty
type board = // intentionally left empty
type player = Player1 | Player2

/// Print the board
val printBoard : b:board -> unit

/// Check whether a pit is the player's home
val isHome : b:board -> p:player -> i:pit -> bool

/// Check whether the game is over
val isGameOver : b:board -> bool

/// Get the pit of next move from the user
val getMove : b:board -> p:player -> q:string -> pit

/// Distributing beans counter clockwise,
/// capturing when relevant
val distribute :
    b:board -> p:player -> i:pit -> board * player * pit

/// Interact with the user through getMove to perform
/// a possibly repeated turn of a player
val turn : b:board -> p:player -> board

/// Play game until one side is empty
val play : b:board -> p:player -> board
```

**Listing 2 awariLibIncomplete.fs:**
**En ikke færdigskrevet implementationsfil.**

```
module Awari
type pit = // intentionally left empty
type board = // intentionally left empty
type player = Player1 | Player2

// intentionally many missing implementations and additions

let turn (b : board) (p : player) : board =
  let rec repeat (b: board) (p: player) (n: int) : board =
    printBoard b
    let str =
      if n = 0 then
        sprintf "Player %A's move? " p
      else
        "Again? "
    let i = getMove b p str
    let (newB, finalPitsPlayer, finalPit)= distribute b p i
    if not (isHome b finalPitsPlayer finalPit)
       || (isGameOver b) then
      newB
    else
      repeat newB p (n + 1)
  repeat b p 0

let rec play (b : board) (p : player) : board =
  if isGameOver b then
    b
  else
    let newB = turn b p
    let nextP =
      if p = Player1 then
        Player2
      else
        Player1
    play newB nextP
```

En version af signaturfilen med yderligere dokumentation og implementationsfilen findes i Absalon i opgaveområdet for denne opgave.

- Jeres løsning skal benytte funktionsparadigmet såvidt muligt.
- Koden skal dokumenteres vha. kommentarstandarden for F#
- Jeres aflevering skal indeholde en afprøvning efter white-box metoden.
- I skal skrive en kort rapport i LaTeX på maks. 10 sider og som indeholder:
  - en beskrivelse af jeres design og implementation
  - en gennemgang af jeres white-box afprøvning
  - kildekoden som appendiks.

# levensthein

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Strenge, rekursion, tests, caching

**Sværhedsgrad** Svær

## 2 Introduktion

Du skal i de følgende to opgaver arbejde med en funktion til at bestemme den såkaldte *Levensthein-distance* mellem to strenge $a$ og $b$. Distancen er defineret som det mindste antal editeringer, på karakter-niveau, det er nødvendigt at foretage på strengen $a$ før den resulterende streng er identisk med strengen $b$. Som editeringer forstås (1) sletninger af karakterer, (2) indsættelser af karakterer, og (3) substitution af karakterer.

Varianter af Levensthein-distancen mellem to strenge kan således benyttes til at identificere om studerende selv har løst deres indleverede opgaver eller om der potentielt set er tale om plagiatkode ;)

Matematisk set kan Levensthein-distancen $leven(a,b)$, mellem to karakterstrenge $a$ og $b$, defineres som $lev_{a,b}(|a|,|b|)$, hvor $|a|$ og $|b|$ henviser til længderne af henholdsvis $a$ og $b$, og hvor funktionen $lev$ er defineret som følger:[1]

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

hvor $1_{(a_i \neq b_j)}$ henviser til *indikatorfunktionen*, som er 1 når $a_i \neq b_j$ og 0 ellers.

## 3 Opgave(r)

1. Implementér funktionen *leven* direkte efter den matematiske definition (ved brug af rekursion) og test korrektheden af funktionen på nogle små strenge, såsom "house" og "horse" (distance

[1] See `https://en.wikipedia.org/wiki/Levenshtein_distance`.

1) samt “hi” og “hej” (distance 2).

2. Den direkte implementerede rekursive funktion er temmelig ineffektiv når strengene $a$ og $b$ er store. F.eks. tager det en del millisekunder at udregne distancen mellem strengene “dangerous house” and “danger horse”. Årsagen til denne ineffektivitet er at en løsning der bygger direkte på den rekursive definition resulterer i en stor mængde genberegninger af resultater der allerede er beregnet.

   For at imødekomme dette problem skal du implementere en såkaldt “caching mekanisme” der har til formål at sørge for at en beregning højst foretages en gang. Løsningen kan passende gøre brug af gensidig rekursion og tage udgangspunkt i løsningen for den direkte rekursive definition (således skal løsningen nu implementeres med to gensidigt rekursive funktioner `leven` og `leven_cache` forbundet med `and`). Som cache skal der benyttes et 2-dimensionelt array af størrelse $|a| \times |b|$ indeholdende heltal (initielt sat til $-1$).

   Funktionen `leven_cache`, der skal tage tilsvarende argumenter som `leven`, skal nu undersøge om der allerede findes en beregnet værdi i cachen, i hvilket tilfælde denne værdi returneres. Ellers skal funktionen `leven` kaldes og cachen opdateres med det beregnede resultat. Endelig er det nødvendigt at funktionen `leven` opdateres til nu at kalde funktionen `leven_cache` i hver af de rekursive kald.

   Test funktionen på de små strenge og vis at funktionen nu virker korrekt også for store input.

   Det skal til slut bemærkes at den implementerede løsning benytter sig af $O(|a| \times |b|)$ plads og at der findes effektive løsninger der benytter sig af mindre plads ($O(\max(|a|, |b|))$). Det er ikke et krav at din løsning implementerer en af disse mere pladsbesparende strategier.

# polynomials

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Højere-ordens funktioner, currying

**Sværhedsgrad** Middel

## 2 Introduktion

Denne opgave omhandler polynomier. Et polynomium af grad $n$ skrives som

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i x^i.$$

## 3 Opgave(r)

1. Skriv en funktion `poly: a:float list -> x:float -> float`, som tager en liste af coefficienter med `a.[i]` $= a_i$ og en $x$-værdi og returnerer polynomiets værdi. Afprøv funktionen ved at lave tabeller for et lille antal polynomier af forskellig grad, med forskellige koefficienter og forskellige værdier for $x$ og validér den beregnede værdi.

2. Afled en ny funktion `line` fra `poly` således at `line : a0:float -> a1:float -> x:float -> float` beregner værdien for et 1. grads polynomium hvor `a0` $= a_0$ og `a1` $= a_1$. Afprøv funktionen ved at tabellere værdier for `line` med det samme sæt af coefficienter $a_0 \neq 0$ og $a_1 \neq 0$ og et passende antal værdier for $x$.

3. Benyt Currying af `line` til at lave en funktion `theLine : x:float -> float`, hvor parametrene `a0` og `a1` er sat til det samme som brugt i Item 2. Afprøv `theLine` som Item 2.

4. Lav en funktion `lineA0 : a0:float -> float` ved brug af `line`, men hvor `a1` og `x` holdes fast. Diskutér om dette kan laves ved Currying uden brug af hjælpefunktioner? Hvis ikke, foreslå en hjælpefunktion, som vil gøre en definition vha. Currying muligt.

# integration

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Højere-ordens funktioner

**Sværhedsgrad** Middel

## 2 Introduktion

Denne opgave omhandler integration. Integralet af næsten alle integrable funktioner kan approximeres som

$$\int_a^b f(x)\,dx \simeq \sum_{i=0}^{n-1} f(x_i)\Delta x,$$

hvor $x_i = a + i\Delta x$ og $\Delta x = \frac{b-a}{n}$.

## 3 Opgave(r)

1. Skriv en funktion `integrate : n:int -> a:float -> b:float -> (f : float -> float) -> float`, hvis argumenter `n`, `a`, `b`, er som i ligningerne, og `f` er en integrabel 1 dimensionel funktion. Afprøv `integrate` på `theLine` fra **??** og på `cos` med $a = 0$ og $b = \pi$. Udregn integralerne analytisk og sammenlign med resultatet af `integrate`.

2. Funktionen `integrate` er en approximation, og præcisionen afhænger af $n$. Undersøg afhængigheden ved at udregne fejlen, dvs. forskellen mellem det analytiske resultat og approximationen for værdier af $n$. Dertil skal du lave to funktioner `IntegrateLine : n:int -> float` og `integrateCos : n:int -> float` vha. `integrate`, `theLine` og `cos`, hvor værdierne for $a$ og $b$ og $f$ er fastlåste. Afprøv disse funktioner for $n = 1, 10, 100, 1000$. Overvej om der er en tendens i fejlen, og hvad den kan skyldes.

# exceptions

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Untagelser og option typen

**Sværhedsgrad** Let

## 2 Introduktion

Denne opgave omhandler untagelser (exceptions), option typer og Stirlings formel. Stirlings formel er en approximation til fakultetsfunktionen via

$$\ln n! \simeq n \ln n - n.$$

## 3 Opgave(r)

1. Implementer fakultetsfunktionen $n! = \prod_{i=1}^{n} i$, $n > 0$ som `fac : n:int -> int` og kast en `System.ArgumentException` undtagelse, hvis funktionen bliver kaldt med $n < 1$. Kald `fac` med værdierne $n = -4, 0, 1, 4$, og fang evt. untagelser.

2. Tilføj en ny og selfdefineret undtagelse `ArgumentTooBig of string` til `fac`, og kast den med argumentet `"calculation would result in an overflow"`, når $n$ er for stor til `int` typen. Fang untagelsen og udskriv beskeden sendt med undtagelsen på skærmen.

3. Lav en ny fakultetsfunktion `facFailwith : n:int -> int`, som `fac`, men hvor de 2 undtagelser bliver erstattet med `failwith` med hhv. argument `"argument must be greater than 0"` og `"calculation would result in an overflow"`. Kald `facFailWith` med $n = -4, 0, 1, 4$, fang evt. untagelser vha. `Failure` mønsteret, og udskriv beskeden sendt med `failwith` undtagelsen.

4. Omskriv fakultetsfunktionen i Opgave 2, som `facOption : n:int -> int option`, således at den returnerer `Some m`, hvis resultatet kan beregnes og `None` ellers. Kald `fac` med værdierne $n = -4, 0, 1, 4$, og skriv resultatet ud vha. en af `printf` funktionerne.

5. Skriv en funktion `logIntOption : n:int -> float option`, som udregner logaritmen af `n`, hvis `n` $> 0$ og `None` ellers. Afprøv `logIntOption` for værdierne `-10`, `0`, `1`, `10`.

6. Skriv en ny funktion `logFac : int -> float option` vha. `Option.bind` 1 eller flere gange til at sammensætte `logIntOption` og `facOption`, og sammenlign `logFac` med Stirlings approximation `n * (log n) - n` for værdierne $n = 1, 2, 4, 8$.

7. Funktionen `logFac : int -> float option` kan defineres som en enkelt sammensætning af funktionerme `Some` og `Option.bind` en eller flere gange og med `logIntOption` og `facOption` som argument til `Option.bind`. Opskriv 3 udtryk, der bruger hhv. `|>` eller `>>` operatorerne eller ingen af dem.

8. Der skal laves følgende implementationer af samme funktion

```
safeIndexIf : arr:'a [] -> i:int -> 'a
safeIndexTry : arr:'a [] -> i:int -> 'a
safeIndexOption : arr:'a [] -> i:int -> 'a option
```

   De skal alle returnere værdien i `arr` på plads `i`, hvis `i` er et gyldigt index, og ellers håndtere fejlsituationen. Fejlsituationerne skal håndteres forskelligt:

   - `safeIndexIf` må ikke gøre brug af `try-with` og må ikke kaste en undtagelse.
   - `safeIndexTry` skal benytte `try-with`, og ved fejltilstand skal `failwith` kaldes.
   - `safeIndexOption` skal returnere `None` i en fejlsituation.

   Der skal laves en kort afprøvning af alle 3 funktioner, der indebefatter at den indicerede værdi udskrives til skærmen som et heltal og ikke som en option type. Afprøvningen skal også afprøve korrekt håndtering af funktionernes evt. kastede undtagelser. Lav en kort beskrivende sammenligning af metodernes evne til at håndtere fejltilstande.

# treeStructure

Jon Sporring

October 18, 2018

## 1 Lærervejledningn

**Emne** Træstrukturer og grafik

**Sværhedsgrad** Middel

## 2 Introduktion

I de følgende opgaver skal vi arbejde med en træstruktur til at beskrive geometriske figurer med farver. For at gøre det muligt at afprøve jeres opgaver skal I gøre brug af det udleverede bibliotek `img_util.dll`, der blandt andet kan omdanne såkaldte bitmap-arrays til png-filer. Biblioteket er beskrevet i forelæsningerne (i uge 6) og koden for biblioteket ligger sammen med forelæsningsplancherne for uge 6. Her bruger vi funktionerne til at konstruere et bitmap-array samt til at gemme arrayet som en png-fil:

```
// colors
type color = System.Drawing.Color
val fromRgb : int * int * int -> color
// bitmaps
type bitmap = System.Drawing.Bitmap
val mk      : int -> int -> bitmap
val setPixel : color -> int * int -> bitmap -> unit

// save a bitmap as a png file
val toPngFile : string -> bitmap -> unit
```

Funktionen `toPngFile` tager som det første argument navnet på den ønskede png-fil (husk extension). Det andet argument er bitmap-arrayet som ønskes konverteret og gemt. Et bitmap-array kan konstrueres med funktionen `ImgUtil.mk`, der tager som argumenter vidden og højden af billedet i antal pixels, samt funktionen `ImgUtil.setPixel`, der kan bruges til at opdatere bitmap-arrayet før det eksporteres til en png-fil. Funktionen `ImgUtil.setPixel` tager tre argumenter. Det første argument repræsenterer en farve og det andet argument repræsenterer et punkt i bitmap-arrayet (dvs. i billedet). Det tredie argument repræsenterer det bitmap-array, der skal opdateres. En farve kan nu konstrueres med funktionen `ImgUtil.fromRgb` der tager en triple af tre tal mellem 0 og 255 (begge inklusive), der beskriver hhv. den røde, grønne og blå del af farven.

Koordinaterne starter med $(0,0)$ i øverste venstre hjørne og $(w-1,h-1)$ i nederste højre hjørne, hvis bredde og højde er hhv. $w$ og $h$. Antag for eksempel at programfilen `testPNG.fsx` indeholder følgende F# kode:

```
let bmp = ImgUtil.mk 256 256
do ImgUtil.setPixel (ImgUtil.fromRgb (255,255,0)) (10,10) bmp
do ImgUtil.toPngFile "test.png" bmp
```

Det er nu muligt at generere en png-fil med navn `test.png` ved at køre følgende kommando:

```
fsharpi -r img_util.dll testPNG.fsx
```

Den genererede billedfil `test.png` vil indeholde et sort billede med et pixel af gul farve i punktet (10,10).

Bemærk, at alle programmer, der bruger `ImgUtil` skal køres eller oversættes med `-r img_util.dll` som en del af kommandoen. Bemærk endvidere, at LaTeX kan inkludere png-filer med kommandoen `includegraphics`.

I det følgende vil vi repræsentere geometriske figurer med følgende datastruktur:

```
type point = int * int // a point (x, y) in the plane
type colour = int * int * int  // (red, green, blue), 0..255

type figure =
  | Circle of point * int * colour
     // defined by center, radius, and colour
  | Rectangle of point * point * colour
     // defined by corners bottom-left, top-right, and colour
  | Mix of figure * figure
     // combine figures with mixed colour at overlap
```

For eksempel kan man lave følgende funktion til at finde farven af en figur i et punkt. Hvis punktet ikke ligger i figuren, returneres `None`, og hvis punktet ligger i figuren, returneres `Some` $c$, hvor $c$ er farven.

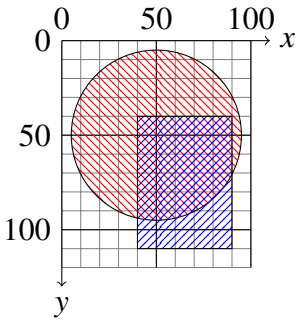
```
// finds colour of figure at point
let rec colourAt (x,y) figure =
  match figure with
  | Circle ((cx,cy), r, col) ->
      if (x-cx)*(x-cx)+(y-cy)*(y-cy) <= r*r
        // uses Pythagoras' formular to determine
        // distance to center
      then Some col else None
  | Rectangle ((x0,y0), (x1,y1), col) ->
     if x0<=x && x <= x1 && y0 <= y && y <= y1
        // within corners
     then Some col else None
  | Mix (f1, f2) ->
      match (colourAt (x,y) f1, colourAt (x,y) f2) with
      | (None, c) -> c  // no overlap
      | (c, None) -> c  // no overlap
      | (Some (r1,g1,b1), Some (r2,g2,b2)) ->
```

```
        // average color
        Some ((r1+r2)/2, (g1+g2)/2, (b1+b2)/2)
```

Bemærk, at punkter på cirklens omkreds og rektanglens kanter er med i figuren. Farver blandes ved at lægge dem sammen og dele med to, altså finde gennemsnitsfarven.

# 3 Opgave(r)

1. Lav en figur `figTest : figure`, der består af en rød cirkel med centrum i (50,50) og radius 45, samt en blå rektangel med hjørnerne (40,40) og (90,110), som illustreret i tegningen nedenfor (hvor vi dog har brugt skravering i stedet for udfyldende farver.)




2. Brug `ImgUtil`-funktionerne og `colourAt` til at lave en funktion

   ```
   makePicture : string -> figure -> int -> int
              -> unit
   ```

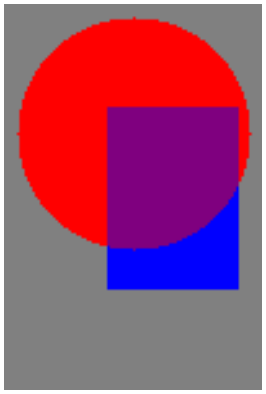
   sådan at kaldet `makePicture` *`filnavn figur b h`* laver en billedfil ved navn *`filnavn`*`.png` med et billede af *`figur`* med bredde *`b`* og højde *`h`*.

   På punkter, der ingen farve har (jvf. `colourAt`), skal farven være grå (som defineres med RGB-værdien (128,128,128)).

   Du kan bruge denne funktion til at afprøve dine opgaver.

3. Lav med `makePicture` en billedfil med navnet `figTest.png` og størrelse $100 \times 150$ (bredde 100, højde 150), der viser figuren `figTest` fra Opgave **??**.

   Resultatet skulle gerne ligne figuren nedenfor.



4. Lav en funktion `checkFigure : figure -> bool`, der undersøger, om en figur er korrekt: At radiusen i cirkler er ikke-negativ, at øverste venstre hjørne i en rektangel faktisk er ovenover og til venstre for det nederste højre hjørne (bredde og højde kan dog godt være 0), og at farvekompenterne ligger mellem 0 og 255.

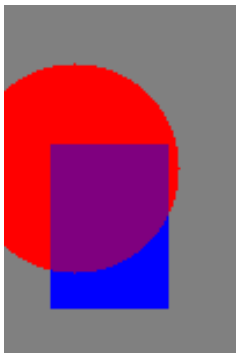
   Vink: Lav en hjælpefunktion `checkColour : colour -> bool`.

5. Lav en funktion `move : figure -> int * int -> figure`, der givet en figur og en vektor flytter figuren langs vektoren.

   Ved at foretage kaldet

   ```
   makePicture "moveTest" (move figTest (-20,20)) 100 150
   ```

   shulle der gerne laves en billedfil `moveTest.png` med indholdet vist nedenfor.



6. Lav en funktion `boundingBox : figure -> point * point`, der givet en figur finder hjørnerne (top-venstre og bund-højre) for den mindste akserette rektangel, der indeholder hele figuren.

   `boundingBox figTest` skulle gerne give `((5, 5), (95, 110))`.

# io

Jon Sporring

October 18, 2018

# 1 Lærervejledningn

**Emne** Input/output

**Sværhedsgrad** Let

# 2 Introduktion

# 3 Opgave(r)

1. Lav en funktion,

   ```
   printFile : unit -> unit
   ```

   som starter en dialog med brugeren. Programmet skal spørge brugeren om navnet på en fil, og derefter skrive filens indhold ud på skærmen.

2. Lav en funktion,

   ```
   printWebPage : url:string -> string
   ```

   som indlæser indholdet af internetsiden på `url` og returnerer resultatet som en streng.

3. Lav en lommeregner,

   ```
   simpleCalc : unit -> unit
   ```

   som starter en uendelig dialog med en bruger. Brugeren skal kunne indtaste simple regnestykker på postive heltal, og hvert regnestykke må kun bestå af en enkelt af følgende binære operatorer: `+`, `-`, `*`, `/`. Resultatet skal kunne genbruges i den efterfølgende beregning med navnet `ans`.

4. Der skal laves et program

   ```
   fileReplace :
     filename:string -> needle:string -> replace:string -> unit
   ```

som erstatter alle forekomster af `needle` argumentet med `replace` argumentet i filen med navn `filename`. Løsningen skal som minimum bruge funktionerne `System.IO.File.OpenText`, `ReadLine` og `WriteLine` til at tilgå filerne. Der skal laves en kort afprøvning, og en kort beskrivelse af løsningen med argumenter for større valg, der er foretaget, for at nå til den givne løsning.

5. I html-standarden angives links med `<a></a>` tags, f.eks. kunne et link til Googles hjemmeside skrives som `<a href="https://google.com">Tryk her for Google</a>`. Der skal laves et program

   ```
   countLinks : url:string -> int
   ```

   som henter internetsiden angivet med argument `url` og som tæller, hvor mange links der er på siden ved at tælle antallet af `<a` delstrenge.

   Bemærk: Langt de fleste internetsider kræver et gyldigt certifikat for at dit program kan læse siden, og som udgangspunkt har mono ingen certifikater installeret. For at installere et nyttigt sæt certifikater kan du bruge `mozroots`, som er en del af Mono pakken. På Linux/MacOS gør følgende fra Konsollen:

   ```
   mozroots --import --sync
   ```

   På Windows gør du følgende (på samme linje)

   ```
   mono "C:\Program Files (x86)\Mono\lib\mono\4.5\mozroots.exe" --import
   --sync
   ```

   Ret evt. stien, hvis din installation af mozroots ligger et andetsted. Derefter kan du læse de fleste sider uden at blive afvist.

   Til besvarelsen skal der laves en kort afprøvning, og en kort beskrivelse af løsningen med argumenter for større valg, der er foretaget, for at nå til den givne løsning.

6. Filen `storeClausLilleClaus.txt` indeholder H.C. Andersens eventyr "Store Claus og Lille Claus" fra 1835. I skal skrive et program, som indlæser filen og udskriver hyppigheden af alle de ord, som bruges i eventyret, til filen `hyppighed.txt`. Hyppigheden skal være sorteret fra mest til mindst hyppige. Med ord skal forstås tegnfølger, som ikke indeholder whitespaces eller tegnsætning, og hvor store bogstaver er konverteret til små.