Programmering og ProblemlÃÿsning, ugeseddel 1

Version 1.5

1. September 2015

Velkommen til kurset "Programmering og Problemløsning".

Dette er den første af i alt 14 *ugesedler*. Ugesedlerne indeholder information om ugens opgaver samt diverse praktisk information og perspektivering af ugens pensum.

Den første undervisningsuge har til formål at gøre dig fortrolig med redigering og afvikling af programmer med Emacs og `F# Interactive` fsi (fsharpi på linux).

Der er ingen obligatoriske opgaver i uge 1, men der er stillet en gruppeopgave og en individuel opgave som træning til de efterfølgende obligatoriske opgaver. Det anbefales stærkt, at aflevere begge opgaver. Der gives feedback til jeres afleveringer, som kan være nyttig til de senere opgaver. Denne uges opgaver kan I aflevere så mange gange I lyster og få ny feedback, så I kan øve jer i at lave en perfekt aflevering. Giv jeres instruktor besked (email) ved nye afleveringer.

Vi bruger følgende lærebøger:

- Hansen & Rischel: *Functional Programming Using F#*, Cambridge University Press 2013. Vi forkorter denne som "HR".

Vi antager at du allerede har anskaffet dig bøgerne ved kursusstart. De kan begge købes i Polyteknisk Boghandel i Biocenteret.

1 Pensum og plan for ugen

Pensum for uge 1: HR kap. 1 og 2, IP-2 kap. 1, 2 og 3 (1.7 dog kun kursorisk).

Foreslået læserækkefølge: IP-2, kap. 1, HR kap. 1 og 2, og dernæst IP-2 kap. 2 og 3.

Forelæsningen mandag vil fokusere på at skrive simple udtryk og funktioner i emacs og køre dem i `F# Interactive` . Hyppigt forekomne fejlmedelelser vil blive forklaret. Øvelserne vil give hjælp til installering af Eduroam, Emacs og `F#` til de studerende, der ikke har gjort det allerede, og der vil derefter laves simple funktioner i `F# Interactive` .

Tirsdag omhandler både forelæsninger og øvelser rekursive funktionserklæringer, par/tupler (sæt), betingede udtryk og simple typer, herunder heltalsaritmetik.

Fredag introducerer tegn (`char`) og tekster (`string`). Øvelserne bruges til at arbejde med den individuelle opgave.

2 Mandagsopgaver

Mål: Fortrolighed med `F# Interactive` systemet, heltal og kommatal, sandhedsværdier, unære og binære operatorer og operatorpræcedens.

1M1 Følg din instruktors anvisninger til installering af `F#` , og Emacs på din bærbare. Hvis du ikke har en med selv, så kig med hos en anden.

1M2 Opsæt (med hjælp fra instruktor) Eduroam på din bærbare.

1M3 Log ind på KUnet og find Absalonsiden for PoP.

1M4 Start `F# Interactive` .

1M5 Evaluer udtrykkene `2+3`, `2.0+3.0` og `2+3.0` i `F# Interactive` husk at udtryk skal afsluttes med `;;` og vigtigs, bemærk forskellene mellem resultaterne. Hvad skyldes de?

1M6 Evaluer udtrykkene `-3 - -5` og `-3 --5` (bemærk forskellen på mellemrumstegn). Hvad sker der?

1M7 Evaluer udtrykket `9*9`, og gentag derefter evaluering af udtrykket `it*it` fire gange. Hvad sker der, og hvorfor?

1M8 Evaluer udtrykket `99.0 * 99.0`, og gentag derefter evaluering af udtrykket `it*it` syv gange. Hvad sker der, og hvorfor?

1M9 Evaluer i `F# Interactive` udtrykkene `2<3`, `2=3`, `3<=3`, `3>=3`, `3>3` og `4<>4`.

1M10 Evaluer udtrykkene `2+3*4` og `(2+3)*4`. Forklar forskellen.

1M11 Evaluer udtrykkene `2+3<4` og `2+(3<4)`. Forklar forskellen.

1M12 Skriv i Emacs en fil `plus5.fsx`, der indeholder en funktionserklæring til en funktion `plus5`, sådan at funktionskaldet `plus5(n)` evalueres til $n + 5$ for alle heltal n. F.eks. skal kaldet `plus5(7)` evaluere til værdien `12`. Hent funktionen ind i `F# Interactive` med kommandoen `#load "plus5.fsx";;`. Hvilken type har funktionen? Hvad sker der, når du laver kaldene `plus5(7)`, `plus5 7`, `plus5(7.0)` og `plus5("7")`?

1M13 Hvis der er tid tilovers, så regn opgave 2.1–2.9 i IP-2.

3 Tirsdagsopgaver

Mål: Læse, håndkøre og skrive simple rekursive funktioner.

Det forventes, at du inden øvelserne tirsdag har forberedt dig på opgaverne ved at løse så mange som muligt på egen hånd.

1T1 Erklær fakultetsfunktionen (n!) i en fil:

```
let rec fact = function
    | 0 -> 1
    | n -> n * fact(n-1)
```

Hent funktionen ind i `F# Interactive` og evaluer kaldene `fact 0`, `fact 7` og `fact 25`. Forklar resultatet af det sidste kald.

Evaluer kaldet `fact -3`. Hvad sker der, og hvorfor?

1T2 Erklær potensfunktionen power i en fil:

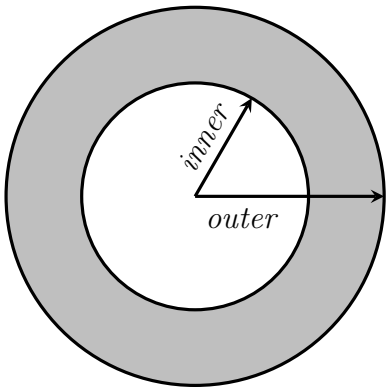
```
let rec power = function
    | (x,0) -> 1
    | (x,n) -> x * power(x,n-1)
```

Evaluer kaldene `power(2,4)` og `power(2.0,4)`. Hvad sker der og hvorfor?

1T3 Overvej, hvorfor funktioner og værdier har typer. Skriv dit svar ned, så diskuterer vi det til øvelserne.

1T4 Håndkør beregningen af `power(3,3)`, dvs. skriv de forskellige skridt i beregningen ned.

1T5 Erklær i en fil en funktion `ringArea : float * float -> float`, sådan at kaldet `ringArea outer inner` er arealet af en ring med radius *outer* og et hul med med radius *inner*. Det vil sige, det grå areal i følgende figur:



Vink: brug funktionen `circleArea` fra HR afsnit 1.2.

1T6 Erklær i en fil en funktion `fooint : int * int -> int`, sådan at kaldet `fooint(`n`,`k`)` evaluerer til værdien $2n-k^2$ for heltal n og k. Beregn `fooint(3,2)`.

Erklær dernæst en funktion `foofloat : float * float -> float`, sådan at kaldet `foofloat(`n`,`k`)` evaluerer til værdien $2n - k^2$ for kommatal n og k. Vær opmærksom på to-tallet. Beregn `foofloat(3.0,2.0)`.

Erklær dernæst en funktion `foomix : float * int -> float`, sådan at kaldet `foomix(`n`,`k`)` evaluerer til værdien $2n - k^2$ for kommatal n og heltal k. Bemærk de forskellige typer af argumenterne til minus. Hvordan håndterer du det? Beregn `foomix(3.0,2)`.

1T7 Omskriv funtionerne `fact` og `power`, så de bruger `if-then-else` i stedet for mønstergenkendelse.

1T8 Hvis der er tid tilovers, løs opgave 1.3, 1.4, 2.1, 2.2 i HR samt 4.1–4.6 i IP-2.

4 Gruppeaflevering

Gruppeafleveringer laves i grupper på op til fire personer (typisk jeres læsegrupper).

I uge 1 er gruppeafleveringen frivillig, men det anbefales at aflevere den og få feedback. Besvarelsen afleveres i Absalon. Der afleveres en fil pr. gruppe, men den skal angive alle deltageres fulde navne i kommentarlinjer øverst i filen. Filens navn skal være af formen `1G-`*`initialer`*`.fsx`, hvor initialer er erstattet af gruppemedlemmernes initialer. Hvis f.eks. Bill Gates, Linus Torvalds, Steve Jobs og Gabe Logan Newell afleverer en opgave sammen, skal filen hedde `1G-BG-LT-SJ-GLN.fsx`. Brug gruppeafleveringsfunktionen i Absalon. Ved genaflevering bruges samme navnekonvention, men med "`-genafl`" tilføjet før `.fsx`, f.eks. `1G-BG-LT-SJ-GLN-genafl.fsx`. Lad filen fra den oprindelige aflevering blive liggende.

I gymnasiet lærte I, at man kan løse andengradsligningen $ax^2 + bx + c = 0$ med formlen

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Som giver 2 reelle løsninger, hvis $b^2 - 4ac \geq 0$ og ingen reelle løsninger, hvis $b^2 - 4ac < 0$.

1G1 Skriv en funtion `solve2 : float * float * float -> float * float`, sådan at
`solve2 a b c` giver de to løsninger for x i ligningen $ax^2 + bx + c = 0$, såfremt $b^2 - 4ac \geq 0$. Du behøver ikke tage stilling til tilfældet $b^2 - 4ac < 0$. Vink: Kvadratrodsfunktionen hedder `sqrt`.

1G2 Hvad giver kaldet `solve2 2.0 3.0 1.0`? Er svaret rigtigt?

1G3 Hvad giver kaldet `solve2 2.0 3.0 4.0`? Hvorfor?

1G4 Den potensfunktion `power`, der er defineret i tirsdagsopgaverne skal bruge 21 multiplikationer til at beregne `power 2 21` –helt generelt bruges n multiplikationer

til at beregne `power a n`. Det kan gøres med væsentligt færre multiplikationer ved at benytte regnereglen $a^{2n} = (a^n)^2$, når man har en lige potens.

Skriv en ny potensfunktion `powerNew : int * int -> int`, som bruger denne regneregel til at reducere antallet af multiplikationer, så f.eks. `powerNew 2 21` kan beregnes med højest 11 multiplikationer. Vink: Brug funktionerne `div` og `mod`.

1G5 Udvid funktionen, så den udover a^n også returnerer antallet af brugte multiplikationer. Lav altså en funktion `powerCount : int * int -> int * int`, sådan at `powerCount a n` returnerer parret `(`a^n`,`m`)`, hvor m er antallet af brugte multiplikationer. Vis resultatet for `powerCount 2 21`.

5 Individuel aflevering

Også den individuelle opgave er frivillig i uge 1, men vi anbefaler igen, at den afleveres. Den afleveres i Absalon som en fil med navnet `1I-`*`navn`*`.fsx`, hvor *`navn`* er erstatttet med dit navn. Hvis du fx hedder Anders A. And, skal filnavnet være `1I-Anders-A-And.fsx`. Skriv også dit fulde navn som en kommentar i starten af filen. Ved genaflevering bruges samme navnekonvention, men med "`-genafl`" tilføjet før `.fsx`, f.eks. `1I-Anders-A-And-genafl.fsx`. Lad filen fra den oprindelige aflevering blive liggende.

1I1 Skriv en funktion `powerFloatInt: float * int -> float`, der givet et kommatal a og et heltal n beregner a^n. Funktionen skal virke både for positive og negative værdier af n. Brug reglen $a^{-n} = \frac{1}{a^n}$. Vis værdien af kaldet `powerFloatInt 0.99 -100`.

1I2 Håndkør funktionen:

```
let rec mystisk m n =
    if (m = 0) then n
    elif (m > n) then (mystisk (m-n) n)
    elif (m < n) then (mystisk m (n-m))
    else m
```

med argumentet `(21,56)` i samme stil som HR afsnit 2.4. Vis håndkøringen i en kommentar i din afleverede .fsx fil. Hvad beregner funktionen?

1I3 Definer en funktion `rightAlign: int * int -> string`, sådan at kaldet `rightAlign(`n`,`w`)` returnerer en tekst af længde w, der indeholder tekstrepræsentationen af tallet n højrestillet i teksten. For eksempel skal `rightAlign(17,4)` returnere teksten `"  17"` (som har længden 4). Hvis der er mere end w cifre i n, skal teksten indeholde alle cifrene (og evt. fortegn), selv om længden dermed overstiger w. For eksempel skal `rightAlign(~17,2)` returnere teksten `"~17"` (som har længden 3).

6 Ugens nød

»[Rework needed]: Der mangle parameter angivelser. Det kan betyde endeløst mange forskellige ting Ugens nød er en frivillig, særligt udfordrende opgave, der kan løses af særligt interesserede studerende. Opgaven kan løses kun ved brug af begreber, der allerede er undervist i, men kræver ekstra omtanke og nogen gange en god ide. Afleveringsfristen er den samme som for den individuelle opgave. Ved fredagsforelæsningen i den efterfølgende uge kåres den bedste løsning til ugens nød, og vinderen får en lille præmie, typisk et stykke chokolade eller andet guf.

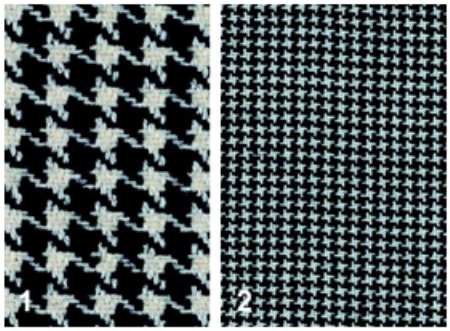
På en gammeldags pedalvæv (som den herover viste) laves mønstre i stoffet ved, at trendtråde og skudtråde er over eller under hinanden. Dette sker ved, at man med pedaler kan løfte eller sænke trendtrådene, så skyttelen (der leder skudtråden igennem) kommer under de løftede tråde men over de sænkede. Ved i forskellige rækker at løfte/sænke forskellige tråde, laves forskellige mønstre. I lærredsvævning løfter man f.eks. på skift de lige og de ulige tråde, så der opstår et "skakbræt" mønster.

Vi antager, at der er fire pedaler nummereret 1 til 4. Hver pedals opførsel beskrives som et m-cifret tal ($2 \leq m \leq 8$) med cifrene 1 og 2, hvor 1 betyder "sænk" og 2 betyder "løft". De m første trendtråde vil blive løftet og sænket efter denne opskrift, hvorefter mønstret gentages for de næste m tråde osv. hele vejen gennem alle trendtråde.

Rækkefølgen af trædning på pedaler ved hvert skyttelskud giver mønstret. Denne rækkefølge beskrives som et n-cifret tal ($2 \leq n \leq 8$), hvor cifrene er 1 til 4, svarene til pedalernes numre.

For eksempel vil pedalerne for lærredsvævning beskrives med tallene 12, 21 og to vilkårlige tal (som vi ikke bruger), og pedalrækkefølgen som 12, idet man skifter mellem de to pedaler.

Pepitatern (nummer 2 herunder):



laves med følgende fire pedaler:

1: 1211
2: 2111
3: 2221
4: 2212

som gentages i rækkefølgen 1234. Det antages at trendtråde er hvide og skudtråde er sorte.

1N1 Skriv en funktion

```
vaev : int * int * int * int * int * int -> string
```

som givet tal for m, de fire pedaler, n og pedalrækkefølgen returnerer en tekst, der viser mønstret ved at vise `+` for hvide tråde og `#` for sorte tråde.

For eksempel skal kaldet `vaev (4, 1211, 2111, 2221, 2212, 4, 1234)` returnere teksten `"+#++\n#+++\n###+\n##+#\n"`, sådan at hvis man skriver

```
print(vaev (4, 1211, 2111, 2221, 2212, 4, 1234))
```

får man vist teksten

```
+#++
#+++
###+
##+#
```

1N2 Det kan være svært at få et indtryk af mønstret uden at se en gentagelse af det. Skriv derfor en funktion

```
vaev2 : int * int * int * int * int * int * int -> string
```

der givet de samme parametre som før samt et heltal k viser mønstret gentaget k gange både vandret og lodret. For eksempel skal kaldet

```
print(vaev2 (4, 1211, 2111, 2221, 2212, 4, 1234, 3))
```

vise teksten

```
+#+++#+++#++
#+++#+++#+++
###+###+###+
##+###+###+#
+#+++#+++#++
#+++#+++#+++
```