Innlevering 1b i IN2040, høst 2021

- Det er mulig å få opptil 10 poeng for denne innleveringen (1b), som altså er andre og siste del av obligatorisk oppgave 1. Du må ha minst 12 poeng tilsammen for 1a pluss 1b for å få den godkjent. For mer informasjon, se obligsiden: https://www.uio.no/studier/emner/matnat/ifi/IN2040/h21/innleveringer.html
- Oppgavene skal løses individuelt og leveres via Devilry innen *fredag, 24. september, kl 23:59*. I første oppgave blir du bedt om å tegne noen diagrammer; disse kan ev. leveres som separate filer ved siden av .scm-filen med kildekode, enten som bildefiler eller enda bedre; samlet i én pdf.
- Trenger du hjelp? Spør i gruppetimene eller på github: https://github.uio.no/IN2040/h21/issues

1 Par og lister

- For å få på plass en solid forståelse av de grunnleggende datatypene *par* og *lister* i Scheme skal vi her tegne 'boks-og-peker'-diagrammer for å vise strukturen til noen enkle eksempler. Tenk at uttrykkene under skrives inn på REPL'et, og tegn så strukturen for returverdien. (Ta bilde av en tegning på papir, bruk et tegneprogram, eller hva enn du foretrekker.) I denne oppgaven kan det være bra å støtte seg på seksjon 2.2 og 2.2.1 i SICP.

  (a) `(cons 42 11)`

  (b) `(cons 42 '())`

  (c) `(list 42 11)`

  (d) `'(42 (11 12))`

  (e)
  ```
  (define foo '(1 2 3))
  (cons foo foo)
  ```

- Vis hvordan man trekker ut elementet `42` fra følgende lister, bare ved bruk av `car` og `cdr`:

  (f) `(0 42 #t bar)`

  (g) `((0 42) (#t bar))`

  (h) `((0) (42 #t) (bar))`

  (i) Vis hvordan listen fra deloppgaven (g) over kan lages bare ved bruk av prosedyren `cons` *og* bare ved bruk av `list`.

2 Rekursjon over lister og høyereordens prosedyrer

(a) Skriv en rekursiv prosedyre `take` som tar et tall `n` og en liste `items` som argumenter og returnerer en ny liste med de `n` første elementene av `items`. (Dersom listen er mindre enn `n` blir alle elementene med.) Les neste del-oppgave også før du begynner. Eksempler på kall:

```
? (take 3 '(a b c d e f)) → (a b c)

? (take 1 '(a b c d e f)) → (a)

? (take 4 '(a b)) → (a b)

? (take 4 '()) → ()
```

(b) Skriv en ny versjon av `take` fra oppgaven over som i stedet benytter *hale-rekursjon*. Den skal ellers oppføre seg likt som i oppgaven over; samme argumenter og returverdier.

(c) Skriv en prosedyre `take-while` som tar et predikat `pred` og en liste `items` som argumenter og returnerer en ny liste med elementene fra `items` som tester sant for `pred`, fra starten av lista og frem til første element som tester usant. Eksempler på kall:

```
? (take-while even? '(2 34 42 75 88 103 250))
→ (2 34 42)

? (take-while odd? '(2 34 42 75 88 103 250))
→ ()

? (take-while (lambda (x) (< x 100)) '(2 34 42 75 88 103 250))
→ (2 34 42 75 88)
```

(d) I foilene fra 3. uke ga vi en definisjon av prosedyren `map` som var noe forenklet (sammenliknet med den innebygde R5RS-versjonen av prosedyren) ved at vår versjon bare kan operere over én liste om gangen. Skriv en variant `map2` som opererer over to lister parallelt, dvs. kaller dens første argument (som må være en prosedyre) på de første elementene i listene, så på elementene på andreplass, osv. Hvis de to listene ikke har samme antall elementer så avslutter `map2` når den har kommet gjennom den korteste listen, f.eks.

```
? (map2 + '(1 2 3 4) '(3 4 5))
→ (4 6 8)
```

(e) I eksempelet over brukte vi den forhåndsdefinerte Scheme-prosedyren `+` som første argument til `map2`. Her skal vi i stedet bruke et `lambda`-uttrykk direkte som en anonym prosedyre. Vis hvordan `map2` kan kombineres med en anonym prosedyre som beregner gjennomsnittet av to tall. F.eks:

```
? (map2 ??? '(1 2 3 4) '(3 4 5))
→ (2 3 4)
```

Lykke til og god koding!