

# Innlevering (1b) i INF2810 (Vår 2016)

- 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 godkjent den. For mer informasjon, se kurssiden: <http://www.uio.no/studier/emner/matnat/ifi/INF2810/v16/innleveringer.html>.
- Oppgavene skal løses individuelt og leveres via Devilry innen *fredag, 26. februar, kl 10:00 (om morgenen)*.
- Trenger du hjelp? Spør i gruppetimene eller forumet: <https://piazza.com/uio.no/spring2016/inf2810/>.

## 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. Bruk gjerne ASCII-grafikk som kan inkluderes som kommentar i samme fil som Scheme-kode. En annen mulighet er å bruke favorittegneprogrammet ditt eller ta bilde av en papirtegning med mobilen og levere dette som en egen fil. I denne oppgaven kan det være bra å støtte seg på seksjon 2.2 og 2.2.1 i SICP.

- (a) `(cons 47 11)`
- (b) `(cons 47 '())`
- (c) `(list 47 11)`
- (d) `'(47 (11 12))`
- (e) `(define foo '(1 2 3))`  
`(cons foo foo)`

- Vis hvordan man trekker ut elementet 3 fra følgende lister, bare ved bruk av `car` og `cdr`:

- (f) `(1 2 3 4)`
- (g) `((1 2) (3 4))`
- (h) `((1) (2) (3) (4))`

- (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) På forelesningen den 2. februar så vi på en rekursiv egendefinert variant av den innebygde prosedyren `length`. Prosedyren tar en liste som argument og teller hvor mange elementer den inneholder. Skriv en *halerekursiv* versjon av denne (kall den f.eks. `length2`).
- (b) Skriv en rekursiv prosedyre `rev-list` som tar en liste som argument og returnerer en ny reversert versjon av lista (uten å bruke den innebygde prosedyren `reverse`). Eksempler på kall:

```
? (rev-list '("towel" "a" "bring" "always"))
→ ("always" "bring" "a" "towel")

? (rev-list '())
→ ()
```

Ta med en kommentar som forklarer hvorvidt du har brukt “vanlig” rekursjon eller spesialtilfellet halerekursjon. Forklar også kort hvorfor du har valgt det ene fremfor det andre.

- (c) Skriv en prosedyre `ditch` som tar to argumenter: et tall og en liste av tall. `ditch` skal returnere en ny liste, der alle forekomster av tallet er fjernet, f.eks.

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

Ta med en kommentar som forklarer hvorvidt du har brukt “vanlig” rekursjon eller halerekursjon. Beskriv også hvordan den tilsvarende prosessens ressursbehov (tid/minne) vokser relativt til størrelsen på input.

- (d) Skriv en prosedyre `nth` som tar to argumenter: et (indeks)tall og en liste. `nth` skal returnere liste-elementet på posisjonen som indekstallet indikerer. Vi ønsker å telle fra 0, dvs. at det første elementet har indeks 0. (Du trenger ikke bry deg om feilsjekking, vi bare antar at prosedyren ikke vil kalles med en ugyldig indeks.) Eksempel på kall:

```
? (nth 2 '(47 11 12 13)) → 12
```

- (e) Skriv en prosedyre `where` som tar to argumenter: et tall og en liste. `where` beregner posisjonsindeks til (første forekomst av) tallet i listen, f.eks.

```
? (where 3 '(1 2 3 3 4 5 3)) → 2
? (where 0 '(1 2 3 3 4 5 3)) → #f
```

- (f) I forelesningen 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 andre plass, 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)
```

- (g) I eksempelet over brukte vi den forhåndsdefinerte Scheme-prosedyren `+` som første argument til `map2`. Vi skal nå i stedet ta i bruk anonyme `lambda`-prosedyrer for å operere over listeelementer. Vis hvordan `map2` kan kombineres med en anonym prosedyre som beregner gjennomsnittet av to tall. Med argumentene `'(1 2 3 4)` og `'(3 4 5)` burde returverdien bli `(2 3 4)`.

På samme måte, vis at en anonym prosedyre kan brukes for å få tilbake en liste som inneholder #t på posisjoner der begge elementene er partall, og ellers #f (dvs. hvis minst én av tallene på denne listeposisjonen er oddetall).

- (h) Skriv en prosedyre `both?` som genererer et predikat som tar to argumenter. `both?` selv tar ett argument som er et predikat som kun tar ett argument. Returverdien til `both?` er en prosedyre som returnerer #t i tilfelle predikatet som var argument til `both?` er gyldig for begge argumentene. Her er noen eksempler som viser hva vi er ute etter:

```
? (map2 (both? even?) '(1 2 3) '(3 4 5)) → (#f #t #f)
```

```
? ((both? even?) 2 4) → #t
```

```
? ((both? even?) 2 5) → #f
```

- (i) Skriv en prosedyre `self` som tar en prosedyre som argument og returnerer en ny prosedyre slik at f.eks.

```
? ((self +) 5) → 10
```

```
? ((self *) 3) → 9
```

```
? (self +) → #<procedure>
```

```
? ((self list) "hello") → ("hello" "hello")
```

Lykke til og god koding!