Oblig 1 - A

1

a) (* (+ 4 2) 5) = 30
Først plusses 4 og 2 til 6, som deretter ganges med 5. Da sitter vi igjen med 30.

b) (* (+ 4 2) (5)) blir feil.
(5) er ikke en gyldig prosedyre, den har ingen argumenter. Det forventes noe mer. Derfor får vi feil ved kjøring.

c) (* (4 + 2) 5) blir feil.
4 + 2 skrives med prefiks notasjon, slik: (+ 4 2). Derfor får vi feil ved kjøring.

d) (define bar (/ 44 2))
bar = 22
Vi gir bar en verdi, og når vi da kaller på bar får vi tilbake den verdien. Her er verdien 44 delt på 2, som er 22.

e) (- bar 11) = 11
Siden vi har definert bar over, kan vi bruke den på nytt. Her tar vi bar, som er 22, minus 11, som blir 11.

f) (/ (* bar 3 4 1) bar) = 12
Først ganger vi listen med tallene 11, 3, 4 og 1. Da får vi 264. Deretter deler vi på bar, som er 11, og får 12.

2

a) or utrykket leter etter den første verdien som kan bli sett på som True/#t. Derfor vil den returnere «paff!». Den vil da ikke fortsette å gå gjennom prosedyren, og derfor ikke legge merke til feilen som ligger i (zero? (1 - 1)) hvor det ikke er brukt prefiks-notasjon.

and utrykket trenger at alle argumenter er True for å ikke returnere en False. Det vil si at hvis den finner en False er den ferdig, fordi da kan den umulig returnere True. Samme som over vil den stoppe før feilen, men her stopper den på første argument som er (= 1 2).

if utrykket sjekker en condition er True, og handler deretter. Her er den det, og «poff!» blir returnert. Da ignorerer den else blokken under, som er udefinert og ville gitt feil hvis den kom dit.

b)

```
(define (sign x)
  (if (positive? x)
      1
      (if (> 0 x)
          -1
          0)))
```

```
(define (sign x)
  (cond ((positive? x) 1)
        ((> 0 x) -1)
        ((= 0 x) 0)))
```

c)

```
(define (sign x)
  (or (and (< 0 x)
           1)
      (and (> 0 x)
           -1)
      (and (= 0 x)
           0)))
```

3

a)

```
(define (add1 x)
  (+ x 1))

(define (sub1 x)
  (- x 1))
```

b)

```
(define (plus x y)
  (if (zero? y)
      x
      (plus (add1 x) (sub1 y))))
```

eller

```
(define (plus-v2 x y)
   (if (< 0 y)
       (plus-v2 (add1 x) (sub1 y))
       x))
```

c)

Prosessen over er rekursiv, siden minnebruken øker eksponensielt. Beregningene bygger seg opp til slutten er nådd.
Prøver derfor å lage en iterativ prossess under:

```
(define (plus-v3 x y)
  (define (iter sum countd)
    (if (zero? countd)
        sum
        (iter (add1 sum)
              (sub1 countd))))
  (iter x y))
```

countd står for countdown

d)

```
 (define (power-close-to b n)
   (define (iter c m e)
     (if(> (expt c e) m)
        e
        (iter c m (+ 1 e))))
   (iter b n 1))
```

Usikker på hvordan jeg kan forenkle definisjonen av hjelpe-prosedyren…

e)
Jeg ser ikke noen måte å gjøre det på, men det er mest fordi jeg er usikker på akkurat hvordan den fungerer…