Structuur van Computerprogramma's I
Tussentijdse Evaluatie

Deze tussentijdse evaluatie is **gesloten boek**. Neem voor **elke vraag** een volledig nieuw blad en schrijf op elk blad je naam, rolnummer en het nummer van de vraag. Antwoorden in potlood worden niet aanvaard!

Merk op dat `do`-lussen en destructieve operaties (`set!`, `set-car!`, `set-cdr!`) **niet** gebruikt mogen worden op deze tussentijdse evaluatie!

Veel succes!

1 Recursie/Iteratie

Als je bij de bank 3% interest krijgt op je spaargeld is elke 100 euro die je inlegt een jaar later 100 + 3% van 100 = 103 euro geworden. Als je de interest niet opneemt maar gewoon herbelegt krijg je volgende opbrengsten:

na 2 jaar	103	+ 3% van 103	= 106.09
na 3 jaar	106.09	+ 3% van 106.09	= 109.2727
na 4 jaar	109.2727	+ 3% van 109.2727	= 112.550881

...

(a) Schrijf een functie `(define (opbrengst x c n) ...)` die uitrekent hoeveel geld er op je spaarrekening staat na `n` jaar als je `x` euro inlegt aan een interestvoet van `c`%.

```
> (opbrengst 100 3 4)
112.550881
> (opbrengst 1000 0.1 20)
1020.1911448605427
```

(b) Levert je oplossing een recursief of een iteratief proces op? Leg uit waarom je dit antwoord geeft. Schrijf dan ook de andere versie.

2 Lijsten en Hogere Orde

(a) Schrijf een functie `(aantal-positief lst)` die van een lijst van getallen berekent hoeveel positieve getallen er in voor komen.

```
> (aantal-positief '(-1 1 2 -2))
2
> (aantal-positief '(0 -1))
1
```

(b) Schrijf een veralgemening van je functie `(aantal test lst)` zodat ze van een lijst berekent hoeveel elementen aan een gegeven predicaat voldoen.

```
> (aantal even? '(1 2 3))
1
> (aantal odd? '(1 2 3))
2
```

(c) Herschrijf **(a)** door je oplossing van **(b)** te gebruiken.

3 Lijsten

(a) Schrijf een functie `(schrap-3 lst)` die van een gegeven lijst elk derde element schrapt.

```
> (schrap-3 '(1 2 3 4 5 6 7 8))
(1 2 4 5 7 8)
> (schrap-3 '(a b c d e f g h j k l m n o p))
(a b d e g h k l n o)
```

(b) Veralgemeen je functie naar een functie die elke **n-de** element uit een lijst schrapt.

```
> (schrap-n 4 '(1 2 3 4 5 6 7 8))
(1 2 3 5 6 7)
> (schrap-n 4 '(a b c d e f g h j k l m n o p))
(a b c e f g j k l n o p)
```

4 Recursie

```
(define (controleer lijst test)
  (if (null? lijst)
      #t
      (and (test (car lijst))
           (controleer (cdr lijst)
                       test))))
```

(a) Wat doet de functie `controleer`?

(b) Is de functie `controleer` een hogere orde functie? Waarom wel/niet?

(c) Schets de output die je ziet bij de evaluatie van onderstaande oproep van controleer.

```
> (trace controleer)
> (controleer '(5 7 1 4 6 4 9) odd?)
```

5 Omgevingsmodel

In een computerspel loopt een spin rond. De snelheid waarmee de spin zich voortbeweegt hangt af van het gewicht van de spin en van haar/zijn energielevel. In de formule wordt een constante `c` gebruikt die een globale variabele is van het computerspel.

```
(define c 5)

(define (snelheid gewicht energie)
      (* c (/ energie gewicht)))
```

(a) Wat is het resultaat van volgende oproep? Teken een omgevingsmodel dat je antwoord uitlegt.

```
(snelheid 10 20)
```

De makers van het spel willen een extra feature invoeren. De spin kan een upgrade verdienen en op dat moment wordt haar snelheid geboost.

Een eerste deel van het team wil de snelheid boosten door de constante `c` tijdelijk te verdubbelen en voorziet volgende code.

```
(define (verdubbel-snelheid-1 gewicht energie)
   (let ((c (* 2 c)))
     (snelheid gewicht energie)))
```

(b) Wat is het resultaat van volgende oproep? Teken een omgevingsmodel dat je antwoord uitlegt.

```
(verdubbel-snelheid-1 10 20)
```

Een ander deel van het team wil de snelheid boosten door de snelheid eerst uit te rekenen en ze dan te verdubbelen. Bijvoorbeeld:

```
(define (verdubbel-snelheid-2 gewicht energie)
   (let ((basisnelheid (snelheid gewicht energie))
         (eindsnelheid (* 2 basissnelheid)))
     eindsnelheid))
```

(c) Wat is het resultaat van volgende oproep? Teken een omgevingsmodel dat je antwoord uitlegt.

```
(verdubbel-snelheid-2 10 20)
```

6 Theorie

Onderstaande code, die we in de cursus bestudeerd hebben, berekent de wortel van een gegeven getal.

```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001))
  (define (improve guess)
    (average guess (/ x guess)))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess))))
  (sqrt-iter 1))
```

(a) Leg in je eigen woorden uit waarom deze methode een *benaderingsmethode* is.

(b) Leg uit waarom het in de `good-enough?` hulpfunctie niet voldoende is om `(< (- (square guess) x) 0.001)` te gebruiken.

(c) Wat zijn in de definitie van de `improve` functie de gebonden en de vrije variabelen? Wat is het taalmechanisme dat maakt dat je in de body van de `improve` functie de parameter `x` van de hoofdfunctie mag gebruiken?

(d) Veronderstel dat (alle andere code identiek) de `improve` functie vernederlandst wordt naar

```
(define (improve gok)
  (average gok (/ x gok)))
```

gaat het geheel nog werken?