Scheme Cheat-Sheet (versie 2/11/2021)

1. Syntaxregels van Scheme-talen (a.k.a. S-Expressions)

a. Primitieve Expressies:

Gehele getallen: 0, 4, -4
Floating-point getallen: 3.4, 3.14e+1, 3.4e-34
Karakters: #\a, #\b, #\newline, #\space
Booleans: #t, #f
Strings: “hallo dit is een string”
Variabelen / identifiers: n, fac, x, my_varke, $123

b. Samengestelde Expressies:

$(\underline{expr_1}\ expr_2\ \ldots.\ expr_n)$

Voorbeelden:

```
(if #t 1 3)
(define n 3)
(+ (* 5 4) 30)
(* (if (= 1 2) 3 4) 5)
```

Betekenis van samengestelde expressies:

- **Indien** $\underline{expr_1}$ een “special form is” ⇒ werking van buiten kennen
  Scheme leren is deels de special forms ervan leren

- **Anders** ⇒ pas procedure $\underline{expr_1}$ toe op argumenten $expr_2$ … $expr_n$

2. Special Forms: De Lego-blokken van Scheme

a. Gewone `define`

Algemene Vorm:

```
(define var expr)
```

Voorbeelden:

```
(define n 10)
(define lettertje #\a)
(define waar #t)
```

b. Procedure `define` (syntactische suiker voor `define` + `lambda`)

Algemene Vorm:

```
(define (var var1 …. varn)
  exp)
```

Voorbeelden:

```
(define (square x) (* x y))
(define (gemiddelde a b)
  (/ (+ a b) 2))
(define (som-der-kwadraten x y)
  (+ (square x) (square y)))
```

c. Normale `if` (met 2 takken)

Algemene Vorm:

```
(if expr1 expr2 expr3)
```

Voorbeelden:

```
(if (= x y) (+ x y) (* x y)))
(if (zero? 0) #t #f)
```

d. `if` met ontbrekende else-tak (slechte stijl meestal)

Algemene Vorm:

```
(if expr1 expr2)
```

Voorbeeld:

```
(if (equal? dag “donderdag”)
  (display “Vandaag is het donderdag”))
```

e. cond

Algemene Vorm:

```
(cond (predicaat1 expr1 … exprk1)
      (predicaat2 expr1 … exprk2)
      (predicaat3 expr1 … exprk3)
      …
      (predicaatN expr1 … exprkN))
```

Waarbij `predicaat`$_N$ meestal `else` is en exp voor expressie staat
De predicaten zijn natuurlijk ook gewoon expressies.

f. Let, let* en letrec

Algemene Vorm:

```
(let ((var1 expr1)
      (var2 expr2)
      . . .
      (varn exprn))
   expr1
   . . .
   exprk))
```

Voorbeelden:

```
(let ((x 1) (y 1)) (+ x y))
(let* ((x 1) (y x)) (+ x y))
(let ((x 1) (y 1)) (display x) (+ y x))
```

g. and en or

Algemene Vorm:

```
(and expr1 . . . exprN)
```

Voorbeelden:

```
(and #t #f #t)
(and #t 0 1)
(or #f #t #f #t)
(or 1 #t 4 5)
```

h. do

Algemene Vorm:

```
(do ((var1 init-expr1 update-expr1)
     (var2 init-expr2 update-expr2)
     . . .
     (varn init-exprn update-exprn))
  (test-expr res1 res2 … resk)
  body-expr1
   . . .
  body-exprm)
```

Voorbeelden:

```
(do ((invoer 5 (- invoer 1))
     (facult 1 (* facult invoer)))
    ((= invoer 0) facult))
```

i. lambda

Algemene Vorm:

(**lambda** (var_1 var_2 … var_n)
  $body\text{-}expr_1$
  . . .
  $body\text{-}expr_m$)

Voorbeelden:

```
(lambda (x) x)
(lambda (x y) (+ x y))
(lambda (x) (display x) (newline) (+ x 1))
```

j. quote

Algemene Vorm:

(**quote** expr)

Voorbeelden:

```
(quote (* x y))
(quote (quote x))
(quote (define x 10))
```

k. set!

Algemene Vorm:

(**set!** var expr)

Voorbeelden:

```
(set! x (+ x 1))
```

3. Procedures en Processen

a. Primitieve (= ingebouwde) procedures

Voorbeelden: `+, *, sin, not`

b. Samengestelde (= zelfgemaakte) procedures

Voorbeelden (zie cursus): `square, sqrt, ...`

c. Procedures roepen soms andere procedures op (⇒ “procedurele abstractie”)

d. Procedures roepen soms zichzelf op (⇒ naam “re occurs” in body ⇒ recursie)

e. Procedures genereren (= “roepen tot leven”) processen

Sommige processen zijn iteratief
(= gebruiken constante hoeveelheid geheugen)
Te herkennen aan staartrecursieve code in de procedure
Sommige processen zijn recursief
(= gebruiken een variabele hoeveelheid geheugen)
Te herkennen aan constructiefrecursieve code in de procedure
Recursieve **procedure** kan zowel recursief als iteratief **proces** genereren

f. De looptijd van processen begrijpen

Tracing = een spoor achterlaten op het scherm
Profiling = een plot maken van looptijden in functie van invoer-args
Performantieschatting = zie algo&data

g. Soorten processen

Lineaire processen (zowel recursief als iteratief geschreven)
Logaritmische processen (zowel recursief als iteratief geschreven)
Exponentiële processen (boomrecursief geschreven)