Minimalbeispiele für Scheme

Die folgenden Beispiele sollen einen Überblick über die in Scheme vorhandenen Kontrollstrukturen geben. Weitergehende Informationen sind bei Wikipedia (http://de.wikipedia.org/wiki/Scheme) und in Bibliotheken zu finden. Leider ist Skript-Fu nicht vollständig zu dem Scheme-Standard R5RS kompatibel, vieles lässt sich jedoch übertragen.

Funktionsaufrufe (auch geschachtelt)

```
(function arg1 arg2 arg3 ... argN)
(function1 arg1 (funktion2 arg2 arg3) arg4)
(+ 2 3 (/ 12 4))
```

Funktionsdefinitionen (typisch)

Der folgende Block definiert eine Funktion mit drei Eingabeparametern. Der Rückgabewert ist der Rückgabewert des letzten Ausdrucks. Innerhalb des let*-Blocks werden drei lokale Variablen definiert (foo, bar, baz), das set! entspricht einer Zuweisung. Variablen, denen man etwas zuweist, sollten innerhalb des let*-Blocks definiert sein.

```
(define (newfunc arg1 arg2 arg3)
   (let* ((foo (+ arg1 5))
          (bar 0)
          (baz (+ arg1 arg2 arg3))
         )
      (set! bar (* arg2 arg3))
      (- foo bar baz)
   )
)
```

Kontrollstrukturen

Der begin-Block gruppiert mehrere Funktionsaufrufe. Damit ist es möglich, in einem Zweig der if-Abfrage mehrere Funktionsaufrufe zu tätigen.

```
(begin
   (func1 arg1 arg2)
   (func2 arg4 arg5)
)

(if (< a b)
   (then-stuff arg1 a b)
   (else-stuff arg2 b a foo)
)

(while (> foo bar)
   (func1 arg foo)
   (func2 bar baz)
)

(set! i 0)
(while (< i 10)
   (func foo i bar)
   (set! i (+ i 1))
)
```

Listen

Listen können mit der list-Funktion erzeugt werden, alternativ kann man '(...) als Abkürzung verwenden. Mittels car greift man auf das erste Element zu, den Rest der Liste (alles bis auf das erste Element) erhält man mit cdr.

```
(car (list 1 2 3))
  --> 1
(cdr (list 1 2 3))
  --> (2 3)
(cdr '(1 2 3))
  --> (2 3)
(car (cdr (list 1 2 3)))
  --> 2
(cadr '(1 2 3))
  --> 2
(nth 2 '(1 2 3))
  --> 3
(length '(1 2 3))
  --> 3
```

Arrays

Einige Prozeduren in Gimp (z.B. gimp-curves-spline) arbeiten mit Arrays. In Skript-Fu kann man folgendermaßen mit Arrays arbeiten:

```
(set! a (cons-array 4 'byte))
  --> #4"00000000"
(aset a 2 42)
  --> 42
(aset a 3 23)
  --> 23
(aref a 1)
  --> 0
(aref a 3)
  --> 23
```