Scheme-Kurzreferenz

Listenfunktionen

Funktion	Funktionsaufruf allgemein	Beispiel	Erläuterung
car	(**car** *liste*)	(**car** '(der hund frisst)) → der **Achtung!** (car '()): **Fehler!**	liefert das erste Element einer Liste
cdr	(**cdr** *liste*)	(**cdr** '(der hund frisst)) → (hund frisst)	liefert die Liste ohne das erste Element
list	(**list** *element-1 ... element-n*)	(**list** 'der 'hund 'frisst) → (der hund frisst)	liefert die Liste mit den Elementen
cons	(**cons** *element liste*)	(**cons** 'der '(hund frisst)) → (der hund frisst)	fügt das Element vorn in die Liste ein
append	(**append** *liste-1 ... liste-n*)	(**append** '(der hund) '(frisst) '(leckeren Pansen)) → (der hund frisst leckeren Pansen)	liefert eine Liste, die der Reihe nach die Elemente der n Listen enthält
null?	(**null?** *liste*)	(**null?** '(der)) → #f (**null?** '()) → #t	prüft, ob die Liste leer ist

list?	(**list?** *objekt*)	(**list?** 'a) → #f (**list?** '(der hund)) → #t	prüft, ob das Objekt eine Liste ist
equal?	(**equal?** *objekt-1 objekt-2*)	(**equal?** 'der 'the) → #f	prüft, ob die Objekte gleich sind

not	(**not** arg1)	(**not** test)	Gibt „wahr“ zurück, wenn das Argument falsch ist
number?	(**number?** arg1)	(**number?** 234)	Gibt „wahr“ zurück, wenn das Argument eine Zahl ist.

Zeichenketten:

symbol->string	(**symbol->string** *symbol*)	(**symbol->string** 'hund) → "hund"	liefert eine Zeichenkette aus den Zeichen des Symbolnamens
string->symbol	(**string->symbol** *string*)	(**string->symbol** "hund") → hund	liefert ein Symbol, dessen Name der Zeichenkette entspricht
string<? **string<=?** **string=?** **string>?** **string>=?**	(**string=?** *string1 string2*)	(**string<?** "hund" "hand") → #f (**string=?** "hund" "hand") → #f (**string>?** "hund" "hand") → #t	Vergleichsfunktionen für Zeichenketten; der Vergleich erfolgt Zeichen für Zeichen von vorn nach hinten gemäß der ASCII-Tabelle
string-ci<? **string-ci<=?** **string-ci=?** **string-ci>?** **string-ci>=?**	(**string-ci=?** *string1 string2*) **ci: case insensitiv**	(**string-ci=?** "hund" "Hund") → #t	Vergleichsfunktionen für Zeichenketten; der Vergleich erfolgt **ohne Berücksichtigung von Groß- und Kleinschreibung** Zeichen für Zeichen von vorn nach hinten gemäß der ASCII-Tabelle
string?	(**string?** *objekt*)	(**string?** 'hund) → #f (**string?** "hund") → #t	prüft, ob das Objekt eine Zeichenkette ist
string-null?	(**string-null?** *string*)	(**string-null?** "hund") → #f (**string-null?** "") → #t (**string-null?** " ") → #f	prüft, ob die Zeichenkette leer ist

string->list	(**string->list** string)	(**string->list** "hallo") -> (#\h #\a #\l #\l #\0)	Ein String wird in eine Liste aus einzelnen Charaktern umgewandelt. **Achtung:** Scheme stelle einen Charakter als #\a oder # \b dar.
list->string	(**list->string** liste)	(**list->string** (#\h #\a #\l #\l #\0)) -> "hallo"	Eine Liste aus Charaktern wird in einen String umgeformt. Beachte die Charakter müssen als #\a usw. vorliegen.
char->integer	**(char->integer** character)	**(char->integer** #\a) -> 97	Liefert zu einem Charakter den entsprechenden ASCII Wert.
integer->char	**(integer->char** zahl)	**(integer->char** 97) -> #\a	Liefert zu einem ASCII Wert den Charakter.
string-ref	**(string-ref** string zahl**)**	**(string-ref** „abcd" 2)-> #\c**)**	Liefert den Character an der Stelle zahl des gegebenen Strings (zahl fängt bei 0 an)

Makros:

if	(**if** *bedingung* *ausdruck1* *ausdruck2)*	(if (null? liste) liste (cdr liste)) → liefert die Restliste, falls die Liste nicht leer	liefert Ausdruck 1 als Funktionswert, wenn die Bedingung zu true evaluiert, sonst Ausdruck 2
cond	(**cond** (*bedingung1 ausdruck2*) ... (*bedingung-n ausdruck-n*))	(**cond** ((>? x 2) (+ x 5)) ((=? x 2) (* x 5)) (else (* x 7)))	Mehrfachfallunterscheidung: liefert den Ausdruck zurück, dessen Bedingung zu true evaluiert
define	(**define** *variable ausdruck*)	(**define** *lex* '((cat katze) (dog hund)))	Definition von globalen Variablen
define	(**define** (*funktionsname parameter-1 ... parameter-n) ausdruck*)	(**define** (sqr x) (* x x))	Definition von Funktionen
let	(**let** ((var-1 ausdr-1) (var-2 ausdr-2) ... (var-n ausdr-n)) Ausdruck)	(**let** ((x 2) (y 3)) (* x y))	lokale Variablenbindungen

and	(**and** (*bedingung1*) (*bedingung2*) … (*bedingungn*))	(**and** (> x 1) (< x 10))	Und-Verknüpfung, liefert #t, wenn alle Bedingungen erfüllt sind.
or	(**or** (*bedingung1*) (*bedingung2*) … (*bedingungn*))	(**or** (> x 1) (< x 10))	Oder-Verknüpfung, liefert #t, wenn eine Bedingungen erfüllt sind.

Prozeduren zur Ausgabe auf dem Bildschirm

Da der Scheme-Interpreter stets eine „Lesen-Auswerten-Schreiben-Schleife durchläuft, ist eine Prozedur zur Ausgabe **nicht** notwendig. Sie kann aber zur Ausgabe von Zwischenergebnissen verwendet werden.

display	(**display** arg1)	(**display** ergebnis)	Der Wert des Argumentes wird auf dem Bildschirm ausgegeben

Funktionen für das Rechnen mit Zahlen

+	(+ arg1 arg2*)*	(**+** 3 4)	Addiert die beiden Argumente (7)
-	(**-** arg1 arg2*)*	(**-** 4 3)	Subtrahiert das 2. vom 1. Argument (1)
*****	(* arg1 arg2*)*	(***** 3 4)	Bildet das Produkt der beiden Argumente (12)
/	(/ arg1 arg2*)*	(**/** 8 4)	Dividiert die beiden Argumente (2)
sqrt	(**sqrt** arg1 *)*	(**sqrt** 4)	Liefert die Quadratwurzel des Argumentes (2)
truncate	(**truncate** arg1*)*	(**truncate** 3,4)	Gibt den ganzen Teil einer reellen Zahl zurück (3)
round	(**round** arg1*)*	(**round** 3,6)	Rundet die reelle Zahl (4)
random	(**random** arg1*)*	(**random** 4)	Liefert Zufallszahl zwischen 0 und arg1
modulo	(**modulo** arg1 arg2*)*	(**modulo** 13 4)	Liefert nur den Rest der Division der Argumente (1)