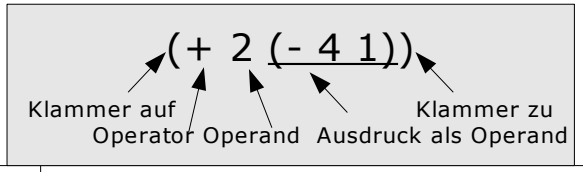
Scheme Sprachübersicht (für Übungen und zur Klausurvorbereitung)



Sprache	*Definitionen*	*Ausdrücke (Expressions)*	*Primitive Operationen*
Beginner	> (define (NAME NAME NAME ...) EXPRESSION) *definiert eine Prozedur mit Parametern* > (define NAME EXPRESSION) *bindet einen Namen an einen Ausdruck („Variablendefinition“)* > (define-struct NAME (NAME ...)) *erstellt eine Struktur mit folgenden Prozeduren:* make-structname (K*onstruktor),* structname? *(Abfrage) und* structname-attributname *(Selektoren)*	> (NAME EXPRESSION EXPRESSION ...) *führt eine Prozedur mit Parametern aus* > (PRIM-OP EXPRESSION ...) *wendet einen primären Operator (+, -, ...) an* > (if CONDITION EXPRESSION EXPRESSION) *falls Bedingung wahr werte ersten Ausdruck aus, ansonsten den zweiten* > (cond [CONDITION EXPRESSION] ... (else EXPRESSION)) *Abfrage mehrerer Fälle, falls Bedingung stimmt werte den Ausdruck aus* > (and EXPRESSION EXPRESSION EXPRESSION ...) *gibt wahr zurück wenn alle Ausdrücke wahr sind* > (or EXPRESSION EXPRESSION EXPRESSION ...) *gibt wahr zurück falls mindestens ein Ausdruck wahr ist* > NUMBER *Zahl*, empty *leere Liste*, true, false *Wahrheitswerte*	> +, *, /, -, = *bekannte Rechenzeichen (preorder beachten, siehe Beispiel oben)* > symbol=? : (symbol symbol -> boolean) *Ist das ein Symbol?* > number? : (any -> boolean) *Ist das eine Zahl?* > list? : (any -> boolean) *Ist das eine Liste?* > empty? : (any -> boolean) *Ist das eine leere Liste?* > first : ((cons y (listof x)) -> y) *Erstes Element der Liste* > rest : ((cons y (listof x)) -> (listof x)) *Rest der Liste* > cons : (x (listof x) -> (listof x)) *setze zwei Objekte (Listen) zusammen*
Beginner with List Abbrev.		> 'QUOTED *Symbol* > ('QUOTED ...) *Liste von Symbolen*	> list : (any ... (listof any) -> (listof any)) *erzeuge eine Liste*
Intermediate		> (local (DEFINITION ...) EXPRESSION) *lokale Umgebung*	
Intermediate w. Lambda		> (lambda (NAME NAME ...) EXPRESSION) *„namenlose“ Prozedur*	
Advanced		> (begin EXPRESSION EXPRESSION ...) *führt Ausdrücke in Sequenz (d.h. Nacheinander) aus und gibt den Wert des letzten zurück* > (set! NAME EXPRESSION) *ändert die Zuweisung einer Variable die mit* define *angelegt wurde*	> build-vector : (nat (nat -> x) -> (vectorof x)) *erzeuge Vektor mit Hilfe einer Prozedur* > make-vector : (number x -> (vectorof x)) *Konstruktor* > vector : (x ... -> (vector x ...)) *Konstruktor* > vector-length : ((vector x) -> nat) *Länge des Vektors* > vector-ref : ((vector x) nat -> x) *Selektor* > vector? : (any -> boolean) *Ist das ein Vektor?*

Mehr Informationen in der Hilfe von Dr.Scheme, auf http://www.plt-scheme.org/docs.html und auf http://www.htdp.org/.

 TU-Darmstadt, Fachbereich Informatik, Fachgebiet Softwaretechnik, Scheme Sprachübersicht (Version: 06.12.05, Zusammenstellung: Lukas Rosenstock)