Grundlegende Schemefunktionen

- `car` bzw. `first`
- `cdr` bzw. `rest`
- `define`
- `cons` und `list`
- `null?`
- `quote` oder `'`
- `reverse`
- `append`
- `length` & `list-ref`
- `if` und `cond`
- `let`
- `assoc`
- `member`

car / first

`car` oder **`first`**

- gibt das erste Element einer Liste zurück
- die Liste darf nicht leer sein 
- Beispiel:

```
(first '(a b c d e))
```

→ a

cdr / rest

`cdr` oder **`rest`**

- gibt die Restliste nach dem ersten Element einer Liste zurück
- die Liste darf nicht leer sein 
- Beispiel:

```
(rest '(a b c d e))
```

→ `'(b c d e)`

define

`define`

- erzeugt in der Auswertungsumgebung eine Bindung eines Ausdrucks (*einer Zahl*, *Variablen*, *Funktion*, ...) an einen Namen
- Beispiel:

```
(define zahl 5)

(define (quadrat a) (* a a))

(quadrat zahl)

→ 25
```


Start

cons und list

`list`

- gibt eine Liste der übergebenen Parameter zurück

`cons`

- gibt eine Liste mit dem neuen Element am Kopf der Liste zurück

```
(list 1 3 5)

→ (1 3 5)

(cons 1 '(3 5))

→ (1 3 5)
```


Start

null?

`null?`

- prüft, ob die übergebene Liste leer ist und gibt #t oder #f zurück
- Beispiel:

```
(null? '(1 3 5))

→ #f

(null? '())

→ #t
```

quote

quote , normalerweise durch das Zeichen ' vor einer Klammer verwendet

- verhindert die Auswertung der übergebenen Liste
- Beispiel:

```
'(1 3 5)    oder   (quote (1 3 5))

→ (1 3 5)

'(sqr 3) → (sqr 3)   aber

(sqr 3) → 9
```


Start

reverse

`reverse`

- gibt die übergebene Liste mit umgekehrter Reihenfolge der Elemente zurück
- Beispiel:

```
(reverse '(1 3 5))
```

```
→ (5 3 1)
```

append

append

- gibt eine Gesamtliste der übergebenen Listen zurück
- Beispiel:

```
(append '(1 3 5) '(a b) '(7))
```

```
→ (1 3 5 a b 7)
```

length und list-ref

`length`

- gibt die Länge der Liste zurück
- Beispiel:

```
(length
 '(1 3 5))
→ 3
```

`list-ref`

- gibt das Element einer Liste an der Position zurück
  [Achtung: 0-basierend!]
- Beispiel:

```
(list-ref
 '(1 3 5 7) 2)
→ 5
```


Start

if und cond

`if`

leitet eine einfache Verzweigung ein

- Beispiel:

```
(if (< a 0)
'negativ
'nicht-negativ)
```

cond

leitet eine Mehrfach-verzweigung ein

- Beispiel:

```
(cond
   ((< a 0) -1)
   ((> a 0) 1)
   (else 0))
```


Start

let

`let`

bindet eine lokale Variable

- Beispiel:

```
(define a 2)
(let ((a 7))
  (* 3 a))
→ 21
(* 3 a)
→ 6
```

let <name>

"named let" bindet eine lokale Funktion [mit Var.]

- Beispiel:

```
(let loop
((n 5))
  (if (= n 1)
  1
  (* n
  (loop (- n 1))))
→ 120
```


Start

Assoziationsliste und assoc

`assoc`

- gibt aus einer Assoziationsliste die erste Teilliste mit dem passenden Kopfelement zurück, wenn es nicht vorhanden ist: **`#f`**.
- Beispiel:

```
(define asli '((a 1)(b 2)(c 3)(d 4)))

(assoc 'b asli)

→ (b 2)
```


Start

Liste und member

member

- gibt aus einer Liste die Restliste ab dem ersten Auftreten des Elements zurück, wenn es nicht vorhanden ist: **#f**.
- Beispiel:

```
(define liste '(a 1 b 2 c 3 d 4))

(member 'b liste)

→ (b 2 c 3 d 4)
```


Start