ADT – Liste

Konstruktoren

In Analogie zu **first** und **rest** dient die Funktion **cons** dazu, durch Aufbau einer neuen Verkettung ein Element zu einer Liste hinzuzufügen.

```
(cons 'a '(b c))  -->  (A B C)
(cons '(a) '(b c))  -->  ((A) B C)
(cons (first '(1 2 3)) (rest '(1 2 3)))
     -->  (1 2 3)
```

Zwischen **first**, **rest** und **cons** steht folgende Beziehung:

```
(cons (first <liste>) (rest <liste>)) ≡ <liste>
```


ADT – liste

Konstruktoren

Neben **cons** stehen noch weitere Funktionen für den Listenkonstruktion zur Verfügung.

(list *arg-1 arg-2* **...** *arg-n***)** Erzeugt eine liste, bestehend aus seinen Argumente

(append *liste-1* **...** *liste-n***)** Erzeugt durch Konkatenation seine Argumente (die Listen sein müssen), eine neue Liste.

Beispiele:

```
(list 1 2 3) --> (1 2 3)
(list (list 1) 2 (list 1 2 3)) --> ((1) 2 (1 2 3))

(append '(1) (list 2)) --> (1 2)
(append (cons 1 (list 2)) nil (list 3)) --> (1 2 3)
```

ADT – Liste

Beispiele:

```
? (setq imperativ-satz '(kommt nach hause))
(kommt nach hause)
? (setq aussage-satz (cons 'ihr imperativ-satz))
(ihr kommt nach hause)
? (setq aussage-satz '(peter faehrt mit dem bus))
(peter faehrt mit dem bus)
? (setq relativ-satz '(der sich beeilen muss))
(der sich beeilen muss)
? (setq aussage-satz (cons (first aussage-satz)
                           (cons relativ-satz
                                 (rest aussage-satz))))
(peter (der sich beeilen muss) faehrt mit dem bus)
```


ADT – Liste

Aufbau von Listen aus Atomen

Im Prinzip genügt **cons** zum Aufbau beliebig komplexer Listen aus Atomen und der leeren Liste:

Beispiele:

```
-> (cons 'a (cons 'b (cons 'c nil)))
(a b c)
-> (cons 'a (cons (cons 'c (cons 'b nil))
                  (cons 'd nil)))
(a (b c) d)
```

ADT – Liste

Noch zwei nützliche Funktionen:

(reverse <*liste*>**)** Erzeugt eine Liste mit den gleichen Elementen wie <*liste*>, aber in umgekehrter Reihenfolge.

(length <*liste*>**)** Liefert die Anzahl der Elemente von <*liste*>.

```
? (reverse '(a b c))
(c b a)
? (first (rest (rest (rest '(a b c d)))))
d
? (first (reverse '(a b c d)))
d
? (length '())
0
? (length '(((a b)) c))
```