**Begriffe und Grundkonzepte**

**Leere Liste**

* Eine Liste ohne Elemente, die durch ein spezifisches Record-Objekt (z. B. empty-list) dargestellt wird.
* Beispiele: (define empty (make-empty-list))

**Cons-Liste**

* Eine nicht-leere Liste, die aus zwei Teilen besteht:
  1. **First**: Das erste Element der Liste.
  2. **Rest**: Eine weitere Liste (kann leer oder eine weitere Cons-Liste sein).
* Wird mit dem Konstruktor cons erzeugt. Beispiel:

racket

Code kopieren

(define list1 (cons 42 empty))

**First**

* Selektor, der das erste Element einer Cons-Liste zurückgibt.
* Beispiel: (first list1) liefert 42, wenn list1 wie oben definiert ist.

**Rest**

* Selektor, der die Liste ohne das erste Element zurückgibt (der Rest der Liste).
* Beispiel: (rest list3) liefert (cons 2 (cons 3 empty)), wenn list3 die Liste (cons 1 (cons 2 (cons 3 empty))) ist.

**Gemischte Daten**

* Daten, die aus unterschiedlichen Arten bestehen können (z. B. leere Listen und Cons-Listen).
* Listen sind ein Beispiel für gemischte Daten, da sie entweder leer oder zusammengesetzt sind.

**Zusammengesetzte Daten**

* Daten, die aus mehreren Bestandteilen bestehen. Bei einer Cons-Liste sind das das erste Element (first) und die restliche Liste (rest).

**Funktionen**

**Funktion: evens**

* Ziel: Gibt eine Liste mit allen geraden Zahlen aus einer gegebenen Liste zurück.
* Rekursive Struktur:
  1. Bei einer leeren Liste wird die leere Liste zurückgegeben.
  2. Bei einer Cons-Liste prüft die Funktion, ob das erste Element gerade ist:
     + Falls ja, wird das Element zur Ergebnisliste hinzugefügt.
     + Falls nein, wird es übersprungen.
* Beispiel:

racket

Code kopieren

(define (evens lst)

(cond

[(empty? lst) empty]

[(even? (first lst)) (cons (first lst) (evens (rest lst)))]

[else (evens (rest lst))]))

**Funktion: positives**

* Ziel: Gibt eine Liste mit allen positiven Zahlen aus einer gegebenen Liste zurück.
* Struktur ist analog zu evens.
* Beispiel:

racket

Code kopieren

(define (positives lst)

(cond

[(empty? lst) empty]

[(> (first lst) 0) (cons (first lst) (positives (rest lst)))]

[else (positives (rest lst))]))

**Erweiterte Konzepte**

**Asymmetrische Struktur**

* Eine Datenstruktur ist asymmetrisch, wenn ihre Bestandteile nicht gleichartig sind. Bei einer Cons-Liste ist das erste Element ein einfacher Wert (z. B. eine Zahl), während der Rest eine Liste ist.

**Refactoring**

* Prozess, bestehende Funktionen oder Datenstrukturen zu verbessern, ohne deren Verhalten zu ändern.
* Im Kontext von Listen kann Refactoring bedeuten:
  + Redundante Konstruktionen zu vermeiden.
  + Allgemeinere Lösungen für spezifische Probleme zu schreiben.

**Funktion-Auswertung (Stepper)**

* Schrittweises Nachvollziehen, wie eine Funktion ausgeführt wird. Besonders nützlich zur Fehlersuche oder zum Verständnis von Rekursion.
* Beispiel:

racket

Code kopieren

(evens (cons 2 (cons 3 (cons 4 empty))))

* 1. (even? 2) → #true, also wird 2 behalten.
  2. (evens (cons 3 (cons 4 empty)))
  3. usw.

**Gerüstbau**

* Ein systematischer Ansatz, um Funktionen über Listen zu schreiben. Dabei wird zuerst die Grundstruktur mit cond oder match angelegt, bevor spezifische Logik implementiert wird.

Beispiel für das Gerüst:

racket

Code kopieren

(define (list-func lst)

(cond

[(empty? lst) ...] ; Fall: leere Liste

[else ; Fall: Cons-Liste

... (first lst) ...

... (list-func (rest lst))]))