Restrekursion (5 Punkte)

Definiere eine Funktion `sum2` als restrekursive Variante der Funktion `meine-summe` aus dem 3. Übungsblatt. Eine Funktion heißt *restrekursiv*, wenn das Ergebnis (d.h. der Rückgabewert) des rekursiven Aufrufs nicht in eine weitere Berechnung eingeht, also nicht als Argument in eine andere Funktion eingesetzt ist.

Auswertungsstrategie (2 Punkte)

Definiere eine Funktion, die zeigt, daß der Rumpf einer Funktionsdefinition zur Definitionszeit (also während der Auswertung des `defun` nicht evaluiert wird.

Bindungsumgebungen (2 + 2 + 2 + 2 Punkte)

Gegeben seien folgende globale Variablen und Funktionsdefinition:

```
(defparameter foo 2)
(defparameter bar 4)
(defun konstig (foo)
  (print foo)
  (print bar)
  (let ((foo 4)
        (bar (+ foo 42)))
    (print foo)
    (print bar)
    (let* ((foo 3)
           (bar (+ foo 17))
           (bar (- bar 1)))
      (print foo)
      (print bar))))
```

Erkläre die Ausgaben, die bei der Auswertung von `(konstig 42)` entstehen.

Rekursion und Funktionsapplikation (5 Punkte)

Schreibe eine Funktion `ntimes`, die 3 Argumente nimmt: Das erste soll ein beliebiges Objekt sein `etwas`, das zweite eine positive Zahl `n`, und das dritte eine Funktion `f`.
`ntimes` soll die Funktion `f` `n`-mal auf `etwas` anwenden: `(f (f (f` .. (f x)))).

Beispiel: `(ntimes 2 3 #'(lambda (x) (* x x))) -> 256`
`(ntimes 'a 3 #'list) -> (((a)))`

Schmökige Lambda-rekursion (3 Schmökpunkte)

Definiere eine zweistellige rekursive Funktion `remove1`, die sich analog zur Common-Lisp Funktion `remove` verhält, e.g. `(remove1 'b '(a b c (a b) b)) -> (A C (A B))`

Erweitere die Funktion `remove1 -> remove42` so, daß das erste Argument eine Funktion ist, die berechnet, ob das Element entfernt werden soll:

```
(remove42 #'(lambda (x) (eq x 'b)) '(a b c (a b) b)) -> (A C (A B))
(remove42 #'(lambda (x) (listp x)) '(a b c (a b) b)) -> (A B C B)
```

Abgabetermin ist der 25. Nov. 1999