Iteration

Die **do**-Anweisung stellt ein allgemeines Iterationskonstrukt dar, bei dem eine beliebige Anzahl lokaler (Lauf-) Variablen genutzt werden kann. Diese Variablen sind innerhalb der Iteration gebunden und können bei jedem Iterationsschritt wie gewünscht verändert werden:

(do ({(*var* [*init* [*step*]])}*) **(***end-test* {*result*}*) {*form*}***)**

Zunächst werden die Laufvariablen *var* initialisiert mit *init* bzw. **nil**. Dann erfolgt der Abruchtest und gegebenenfalls die erste Iteration ({*form*}*).

Bei jedem weiteren Iterationschritt erhalten die Variablen gleichzeitig den jeweils durch *step* bestimmten Wert, bzw. sie bleiben unverändert, falls *step* nicht spezifiziert wurde.

Der Abruchtest erfolgt nach diesen Zuweisungen. Bei Erfolg des Testes werden die Resultatsanweisungen ({*result*}*) evaluiert und der Wert der letzten Anweisung (bzw. **nil**, falls kein *result* angegeben wurde) als Ergebnis des **do** zurückgegeben.

In Analogie zu **let*** steht auch ein **do*** Konstrukt zur Verfügung, bei dem im Gegensatz zu **do** Variablenwerte bei jedem Iterationsschritt rein sequentiell modifiziert werden.

Iterative Version von reverse

```
(defun do-reverse (liste)
   (do ((rest-liste liste (rest rest-liste))
        (result nil result))
       ((null rest-liste) result)
     (setq result
       (cons (first rest-liste) result))))
```

Elegantere iterative Version von reverse

```
(defun do-reverse (liste)
   (do ((rest-liste liste (rest rest-liste))
        (result nil (cons (first rest-liste) result)))
       ((null rest-liste) result)))
```

Verwendung von do*

```
(defun do-reverse (liste)
   (do* ((rest-liste liste (rest rest-liste))
         (result nil result)
         (first (first rest-liste) (first rest-liste)))
       ((null rest-liste) result)
     (setq result
       (cons first result))))
```