

(Keine Angst vor (runden Klammern))

Eine kleine Einführung in Emacs Lisp (Elisp)

22.1.2002

Patrick Gundlach

Was ist Elisp?

... ein Lisp-Interpreter

... die Sprache, mit der ich Emacs erweitern kann

... eine Werkzeugsammlung

... Inhalt dieses Vortrags

[anfang]

[-]

[+]

[ende]

[zurück]

[schließen]

Was ist Elisp?

... ein Lisp-Interpreter

... die Sprache, mit der ich Emacs erweitern kann

... eine Werkzeugsammlung

... Inhalt dieses Vortrags

[anfang]

[-]

[+]

[ende]

[zurück]

[schließen]

Was ist Elisp?

... ein Lisp-Interpreter

... die Sprache, mit der ich Emacs erweitern kann

... eine Werkzeugsammlung

... Inhalt dieses Vortrags

[anfang]

[-]

[+]

[ende]

[zurück]

[schließen]

Was ist Elisp?

... ein Lisp-Interpreter

... die Sprache, mit der ich Emacs erweitern kann

... eine Werkzeugsammlung

... Inhalt dieses Vortrags

[anfang]

[-]

[+]

[ende]

[zurück]

[schließen]

Was ist Elisp?

... ein Lisp-Interpreter

... die Sprache, mit der ich Emacs erweitern kann

... eine Werkzeugsammlung

... Inhalt dieses Vortrags

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

[anfang]

[-]

[+]

[ende]

[zurück]

[schließen]

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

[\[anfang\]](#)

[\[−\]](#)

[\[+\]](#)

[\[ende\]](#)

[\[zurück\]](#)

[\[schließen\]](#)

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

Was ist Lisp?

... List Processing

... die zweitälteste (höhere) Programmiersprache

... entstanden ab 1958

... aber immernoch aktuell

... eine Sprache mit vielen Besonderheiten

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Was ist besonders an Lisp?

- ... erweiterbar (z. B. for-Schleifen)
- ... keine Pointer-Arithmetik
- ... kurze Programme
- ... elegante Konstrukte (anonyme Funktionen, ineinanderschachteln von Funktionen)
- ... einfach zu erlernen
- ... Klammern dürfen den Programmierer nicht abschrecken

Warum ich Lisp mag ...

- Funktionen können übersichtlicher werden

```
(if test
    (other-window (- n))
    (other-window -1))
```

```
(other-window
 (if test (-n) -1))
```

```
(if a
    (if b
        (...)))
```

```
(and a b (...))
```

Warum ich Lisp mag ...

- Funktionen können übersichtlicher werden

```
(if test  
    (other-window (- n))  
    (other-window -1))
```

```
(other-window  
  (if test (-n) -1))
```

```
(if a  
    (if b  
        (...)))
```

```
(and a b (...))
```

Warum ich Lisp mag ...

- Funktionen können übersichtlicher werden

```
(if test
      (other-window (- n))
      (other-window -1))
```

```
(other-window
 (if test (-n) -1))
```

```
(if a
    (if b
        (...)))
```

```
(and a b (...))
```

Warum ich Lisp mag ...

- Funktionen können übersichtlicher werden

```
(if test
      (other-window (- n))
      (other-window -1))
```

```
(other-window
 (if test (-n) -1))
```

```
(if a
    (if b
        (...)))
```

```
(and a b (...))
```

Ausdrücke in Lisp

- ein Lisp-Programm besteht aus einer Ansammlung von (symbolischen) Ausdrücken
- ein Ausdruck ist entweder ein *Atom* oder eine *Liste*
- eine Liste kann Ausdrücke enthalten

Beispiele:

```
rot blau 4 gelb "Linux" 1  
( ) (rot blau) (gelb (gelb))
```

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

[\[anfang\]](#)

[\[−\]](#)

[\[+\]](#)

[\[ende\]](#)

[\[zurück\]](#)

[\[schließen\]](#)

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

[\[anfang\]](#)

[\[−\]](#)

[\[+\]](#)

[\[ende\]](#)

[\[zurück\]](#)

[\[schließen\]](#)

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

[\[anfang\]](#)

[\[−\]](#)

[\[+\]](#)

[\[ende\]](#)

[\[zurück\]](#)

[\[schließen\]](#)

Auswerten von Ausdrücken

Lisp wertet jeden Ausdruck aus, wenn er nicht besonders geschützt ist.

Wie können Ausdrücke geschützt werden?

- `(quote blau)`
- `'blau`

Variablen (Atome) sind erst undefiniert.

Zahlen und Zeichenketten werden zu ‚sich selbst‘ ausgewertet

Das Ergebnis wird zurückgeliefert

Lisp sieht eine Liste

Was passiert?

1. die Liste wird zurückgegeben (schützen!)
'(eins 2 drei)
2. es gibt einen Fehler
('un sinn)
3. das erste Symbol der Liste wird ausgewertet
(message "Hallo Elug")

Lisp sieht eine Liste

Was passiert?

1. die Liste wird zurückgegeben (schützen!)
'(eins 2 drei)
2. es gibt einen Fehler
('un sinn)
3. das erste Symbol der Liste wird ausgewertet
(message "Hallo Elug")

Lisp sieht eine Liste

Was passiert?

1. die Liste wird zurückgegeben (schützen!)
'(eins 2 drei)
2. es gibt einen Fehler
('un sinn)
3. das erste Symbol der Liste wird ausgewertet
(message "Hallo Elug")

Lisp sieht eine Liste

Was passiert?

1. die Liste wird zurückgegeben (schützen!)
'(eins 2 drei)
2. es gibt einen Fehler
('un sinn)
3. das erste Symbol der Liste wird ausgewertet
(message "Hallo Elug")

Wert zuweisen

Was passiert bei ...?

... `(set a 3)`

... `(set 'a 3)`

... `(setq a 3)`

Wert zuweisen

Was passiert bei ...?

... (set a 3)

... (set 'a 3)

... (setq a 3)

Wert zuweisen

Was passiert bei ...?

... `(set a 3)`

... `(set 'a 3)`

... `(setq a 3)`

Wert zuweisen

Was passiert bei ...?

... `(set a 3)`

... `(set 'a 3)`

... `(setq a 3)`

Wert zuweisen

Was passiert bei ...?

... `(set a 3)`

... `(set 'a 3)`

... `(setq a 3)`

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Listen

```
(setq os '(linux bsd emacs))
```

```
(setq hdd '((hda "/usr")  
            (sda "/mnt")  
            (null "/usr/local")))
```

(car os) ergibt linux

(cdr os) ergibt die Liste (bsd emacs)

(cdr (cdr os)) ergibt die Liste (emacs)

(car (cdr hdd)) ergibt die Liste (sda "/mnt")

(nth 2 os) ergibt emacs

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionen

- haben meist keine Seiteneffekte
- funktioniert nicht: `(+ 1 i)`
- `(setq i (+ i 1))`
- manchmal passieren merkwürdige Dinge

```
(setq os '(linux bsd emacs))
```

```
(reverse os), os = (linux bsd emacs)
```

```
(nreverse os), os = (linux)
```

Funktionsaufrufe

- `(message "Hallo Welt")`
- Funktionsaufrufe können geschachtelt werden:
`(message (downcase "Hallo Welt"))`
- Oder mehrfach:

```
(message  
  (if (< erstes zweites)  
      "weniger"  
      "gleichviel oder mehr"))
```

Funktionsaufrufe

- `(message "Hallo Welt")`
- Funktionsaufrufe können geschachtelt werden:
`(message (downcase "Hallo Welt"))`
- Oder mehrfach:

```
(message  
  (if (< erstes zweites)  
      "weniger"  
      "gleichviel oder mehr"))
```

Funktionsaufrufe

- `(message "Hallo Welt")`
- Funktionsaufrufe können geschachtelt werden:
`(message (downcase "Hallo Welt"))`
- Oder mehrfach:

```
(message  
  (if (< erstes zweites)  
      "weniger"  
      "gleichviel oder mehr"))
```

Definieren von Funktionen

```
(defun malzwei (zahl)
```

```
  "Verdoppelt die Zahl ZAHL und liefert das  
  Ergebnis zurueck."
```

```
  (* zahl 2))
```

- Definition der Funktion muss ausgewertet werden
- Änderung durch Neudefinition
- Integrierte Dokumentation (describe-function)

```
malzwei is a Lisp function.
```

```
(malzwei ZAHL)
```

```
Verdoppelt die Zahl ZAHL und liefert das  
Ergebnis zurueck.
```

Echtes Elisp

```
(defun pg-zeile-duplizieren (n)
  "Dupliziert die Zeile, in der der Cursor ist.
  Mit Argument, dupliziere N-Mal."
  (interactive "p")
  (copy-region-as-kill
   (line-beginning-position)
   (progn
    (forward-line 1)
    (point)))
  (save-excursion
   (let ((i 1))
     (while (<= i n)
       (yank)
       (setq i (1+ i))))))
```

[\[anfang\]](#)

[\[−\]](#)

[\[+\]](#)

[\[ende\]](#)

[\[zurück\]](#)

[\[schließen\]](#)

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)
- Taste zuweisen

```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```
- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus

```
(pg-zeile-duplizieren 3)
```

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)
- Taste zuweisen


```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```
- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus

```
(pg-zeile-duplizieren 3)
```

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)

- Taste zuweisen

```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```

- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus
(pg-zeile-duplizieren 3)

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)
- Taste zuweisen

```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```

- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus
(pg-zeile-duplizieren 3)

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)
- Taste zuweisen

```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```

- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus
(pg-zeile-duplizieren 3)

Wie rufe ich die Funktion auf?

- Emacs Kommandozeile (M-x pg-zeile-duplizieren)
- mit Argument (C-u 3 ...)
- Taste zuweisen

```
(global-set-key [(control insert)]  
  'pg-zeile-duplizieren)
```

- (eval-expression) (M-:)
- Aufruf aus einer Funktion heraus
(pg-zeile-duplizieren 3)

Abspeichern von Funktionen

```
;;; duplicate.el --- Dupliziert Zeilen.  
(message "Loading duplicate...")
```

```
(defun pg-zeile-duplizieren (n)  
  ...)  
(provides 'duplicate)  
(message "Loading duplicate...done")
```

- Laden mit (require 'duplicate)
- Datei duplicate.el muss im load-path zu finden sein
- load-path erweitern mit
(add-to-list 'load-path "~/elisp")

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen

Dokumentation

- `(describe-function)`, bzw. `(describe-variable)`
- `(apropos)`
- Elisp-Info-Datei
- (E)Lisp Bücher
- Quelltexte von anderen Programmen
- Newsgruppen