Funktionale Sprachen

- Eigenschaften funktionaler Programmierung:
  - Programm: ausschließlich Funktionen
  - „Main“-Funktion: Funktion mit Eingaben als Argument und die Ausgabe als Ergebnis
  - Funktionen rufen andere Funktionen auf, die wiederum andere Funktionen aufrufen usw. ; auf unterster Ebene vordefinierte Funktionen (Primitive)
  - keine Zuweisungen, d.h. „Variablen“ erhalten sich nicht ändernden Startwert (=Konstanten)
  - keine Seiteneffekte: Funktionen berechnen nur, liefern Ergebnis
  - Programmierer: nicht um Kontrollfluss kümmern

Funktionale Sprachen

- Vorteile funktionaler Programmierung:
  - mathematisch leicht nachvollziehbar, Korrektheitsbeweis in der Regel einfach
  - Modularität existiert automatisch: schnelle Erstellung kleiner Module (Funktionen), können in komplexeren wieder-verwendet werden
  - schnelle Programmentwicklung, sehr zuverlässige Programme

Funktionale Sprachen: LISP

- entstanden in den frühen 60ger Jahren
- **List** **P**rocessor, entwickelt von McCarthy am MIT
- theoretische Grundlage: Lambda-Kalkül
- verschiedene Versionen → manchmal leicht unterschiedliche Funktionsnamen
- extrem einfache Syntax (Präfix-Form) → einfaches Parsen
- viele wichtige PS-Ideen erstmalig in Lisp, z.B
  - Garbage Collection
  - Rekursion
  - Quellcode-Tracing und Debugging

Funktionale Sprachen: LISP

- funktionale Programmiersprache, aber nicht (mehr) völlig funktional: enthält heute z.B.
  - Zuweisungen
  - Kontrollfluss (Schleifen, ...)
- bekanntester Standard: Common Lisp (mit hunderten von vordefinierten Funktionen), 1984 (1994)
- Erweiterungen von Lisp
  - CLOS (Common Lisp Object System)
  - CLIM (Common Lisp Interface Manager)

Funktionale Sprachen: LISP

Literatur zu funktionaler Programmierung mit Lisp:

- David S. Touretzky: *Common Lisp: A Gentle Introduction to Symbolic Computation*

  http://www.cs.cmu.edu/~dst/LispBook
- Guy L. Steele: *Common Lisp: The Language*, 2. Auflage, 1990

  http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/cltl2.html
- Internetseite zu clisp: http://clisp.cons.org
- Diverses: http://clisp.cons.org/resources.html

Funktionale Sprachen: LISP

zwei Arten von „Objekten“: **Atom** oder **Liste**
(früher Cons cell)

1. Atom: elementarer Lisp-Ausdruck
   - Zahl
     - Integer: 3 oder b1101 oder xD5
     - Float: 123.45 oder 3.0E-2
     - Rational: Bruch zweier Integer -17/30
     - Komplexe 2+3i: #C(2 3)

   - Symbol
     - bel. Kombination Zahlen/Zeichen, beginnend mit einem Buchstaben
     - stellen „Variablen“, Funktionsnamen oder Prädikate dar
     - haben Wert (Konstante, Fkt-Ergebnis, Wahrheitswert)
     - Beispiele: X11, ABS, Montag-22-10-12, VariablenSolltenImmerBedeutungsvolleNamenHaben
     - T bedeutet true, NIL false (auch: ())

Funktionale Sprachen: LISP

- Zeichenkette: intern nur Großbuchstaben
- bewusst Kleinbuchstaben: \ voranstellen oder Name in |….| einschließen:

aBcDe	→	ABCDE
|aBcDe|	→	aBcDe
\aB\cD\e	→	aBcDe
a.b.c	→	A.B.C
|Lisp Symbol|	→	Lisp Symbol

Funktionale Sprachen: LISP

2. Liste: Reihe von Elementen (= Atome oder Listen)
   - Listen: Datenobjekte, Funktionsaufrufe, Makros, spezielle Ausdrücke, Programmcode
   - durch ( ) geklammert, Elemente durch Leerstelle getrennt
   - leere Liste: () oder NIL
   - Beispiele: (3 6 2 9) ; einfache Liste
     ((a b) (c d)) ; geschachtelte Liste
     (a (b c) ((1 2) 3)) ; bel. Schachtelung möglich

Funktionale Sprachen: LISP

Funktionen: definieren Operation mit Parametern

- Mengen an vordefinierten Funktionen
- haben immer Präfix-Notation
- allgem. Aufbau: (f_name par_1 … par_n)
- Parameter: Lisp-Ausdrücke, werden von links nach rechts evaluiert
- Beispiele:

(+ 3 5 9)	→	17
(+ 3 (* 4 5))	→	23
(/ 2 3)	→	2/3
(/ 6 2.0)	→	3.0
(- 50 3 5 10)	→	32

- soll Parameter nicht evaluiert werden: ´ voranstellen

´(+ 3 (* 4 5))	→	(+ 3 (* 4 5))
(first ´(a b c))	→	A

Funktionale Sprachen: LISP

Prädikate: Funktionen, die Wahrheitswert T oder NIL liefern

– vordefinierte Prädikate (ausschnittsweise!):

```
(eq  x  y) ; identisches Objekt (gleiche Speicheradresse)?
(eql  x  y)          ; gleiche Zahl / identisches Objekt?
(equal  x  y)        ; gleicher Lisp-Ausdruck?
(=  x  y)            ; gleiche Zahl?
(not x)              ; Negation von x
(atom x)             ; x Atom?
(numberp x)          ;ist x vom Typ number? analog: integerp, floatp,
                     ; realp, rationalp, complexp, characterp,
                     ; stringp,…
(evenp x)            ; x gerade? analog: oddp
```

Funktionale Sprachen: LISP

– Beispiele:

```
(numberp 3)          → T
(symbolp 3)          → NIL
(evenp 3)            → NIL
(evenp t)            → error
(eq  ´(a)  ´(a))     → NIL
(eq  ´a  ´a)         → T
(eql  3  3.0)        → NIL
(equal ´(a)  ´(a))   → T
(= 3 3.0)            → T
(and  (> 3 2) (< 10 9))      → NIL
(or (> 3 2) (< 10 9))        → T
```

Funktionale Sprachen: LISP

Urform der Liste in Lisp: sog. *dotted pair* oder *cons cell*

- (a . d) **a**dress-Part, **d**ekrement-Part
- einfaches Modell des Speichers (IBM 704)
- Basisfunktionen auf cons cells und Atomen:
  - cons: kombiniert zwei Atome oder Listen zu cons cell
  - car: contents of the addres register → a
  - cdr: contents of the dekrement register → d
  - eq: Prädikattest (gleiche Speicheradresse?)
  - atom: Prädikattest (ist Argument Atom?)
- Programmierfunktionen:
  - cond, lambda, define, quote, eval, numerische Funktionen wie +,-,*,…

Funktionale Sprachen: LISP

1. (car (cdr (cdr `(a b c d e)))) ~>

2. (first (rest (rest `(a (b c) (d e))))) ~>
   Kurzform: (caddr `(a (b c) (d e)))

3. (caadr `(a (b c) (d e))) ~>

4. (caadr `(a b c d e)) ~>

5. (cddadr `(a (b c) (d e))) ~>

Funktionale Sprachen: LISP

Programmierfunktionen:

- (cond (test1 ausdruck1)
  (test2 ausdruck2)
  …
)

  bedingter Ausdruck: führt Ausdruck hinter erstem test*i* ungleich NIL aus
- (quote param) param wird nicht evaluiert, bleibt als Atom, heute kürzer: ´param
- (lambda (par*1* … par*n*)
  ausdruck*1*
  …
  ausdruck*n*)

  definiert unbenannte Funktion

Funktionale Sprachen: LISP

- (define name ausdruck) Deklaration

  in Kombination mit (lambda …) als Ausdruck: Funktionsdeklaration

  heute („benannte Funktion“) kürzer:

  (defun name (par*1* …par*n*)

  ausdruck*1*

  …

  ausdruck*n*)
- eval erzwingt Berechnung eines Ausdrucks
- Urform bis hierher: *pure Lisp* (ohne Seiteneffekte)