

Programmieren als Grundkompetenz

Irrtümer, Chancen und Methodik
der Programmierausbildung

Michael Sperber

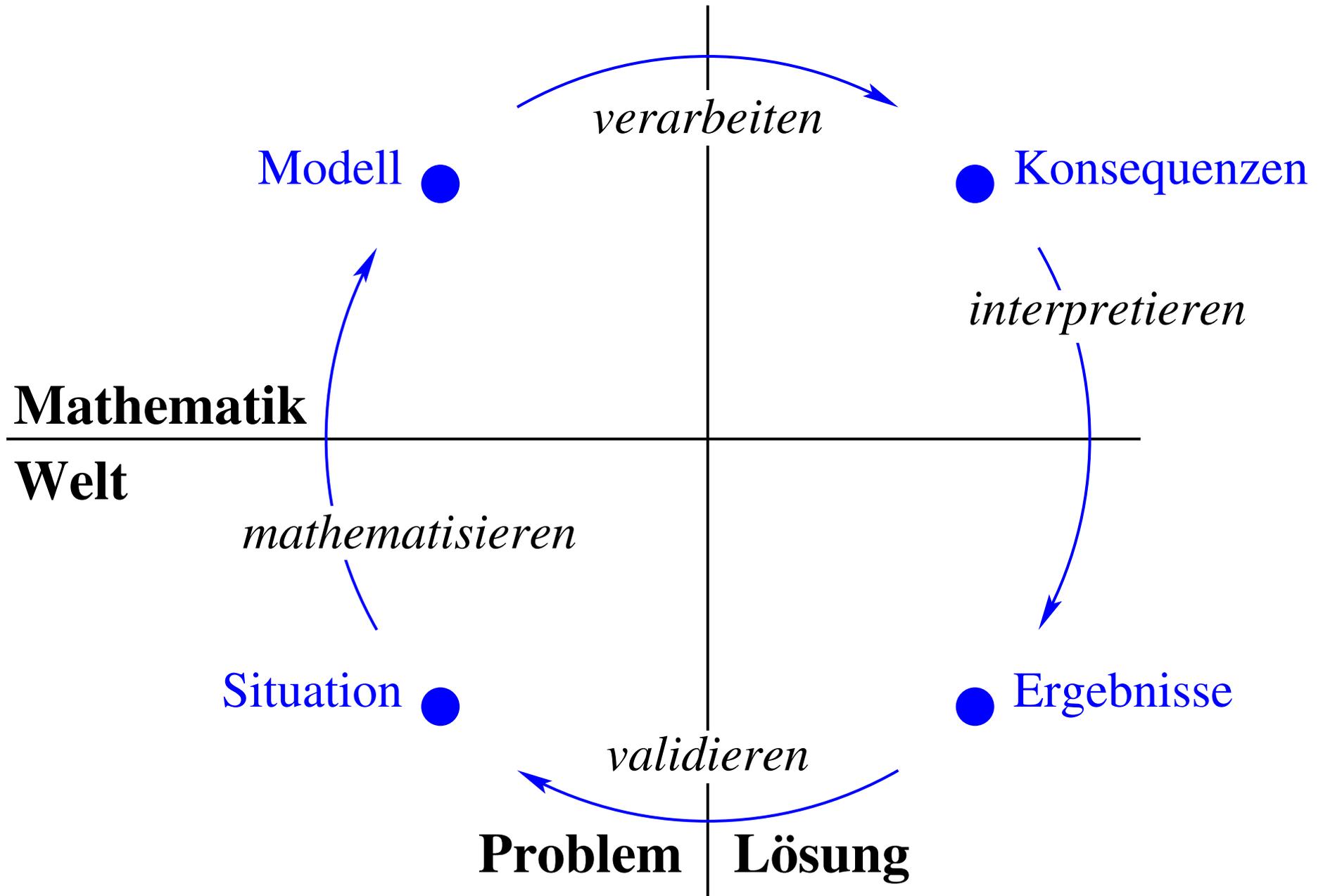
sperber@informatik.uni-tuebingen.de

**dein
programm**

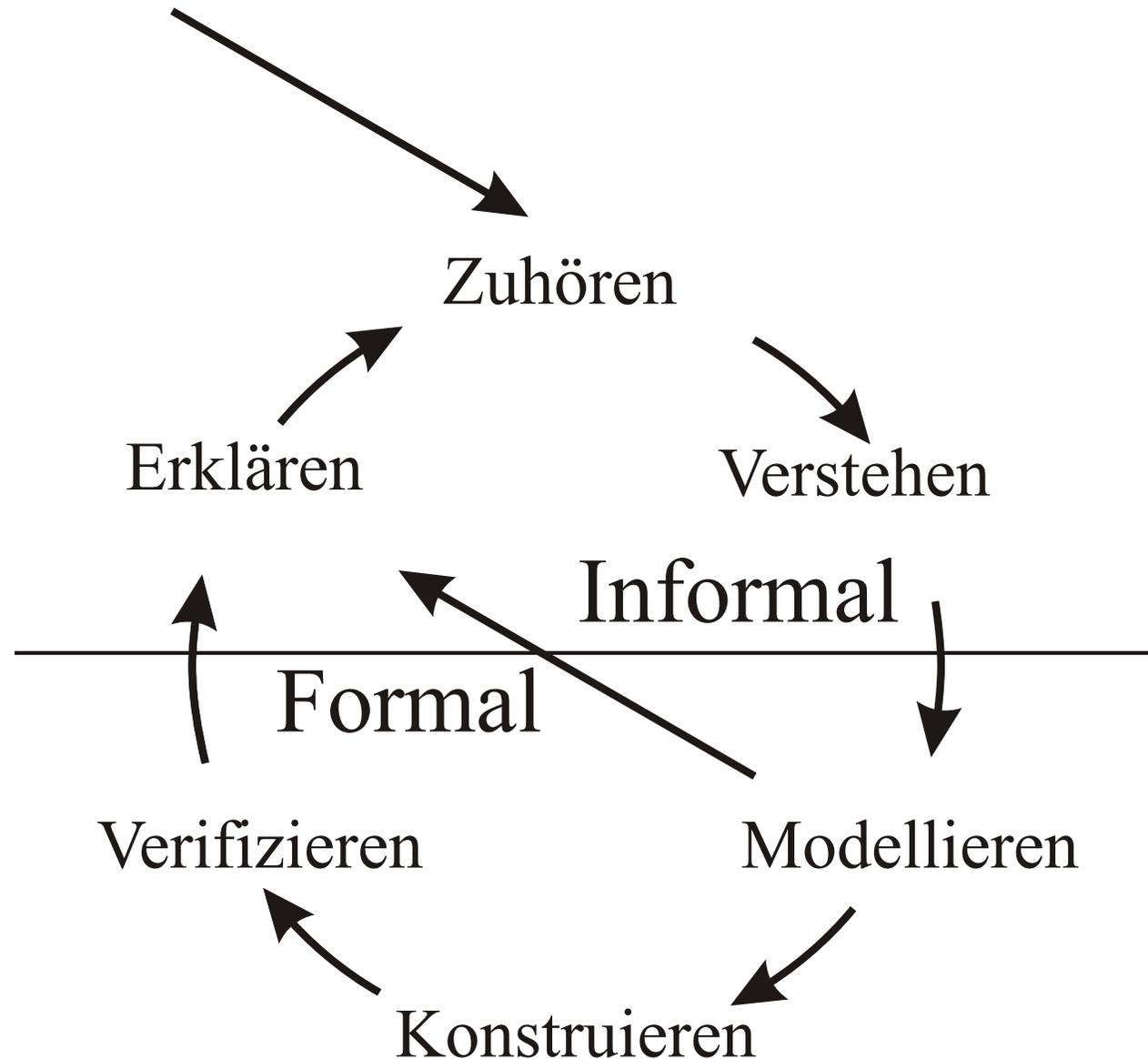
Vorgeschichte

- ...
- „AP Computer Science“ (1987/1998)
- „Informatik I“ (viele Male)
- ...
- „Programmieren für Geisteswissenschaftler“ (1999)
- „Informatik I“ (1999/2000, 2000/2001)
- „Programmieren für Geisteswissenschaftler“ (2001/2002)
- **PISA-Studie** (2002)

Mathematisieren



Programmieren als Form des Problemlösens



Mathematik und Programmieren

PISA-Studie:

Mathematik-Kompetenz

→ *Mathematical Literacy*

→ Problemlösungskompetenz

Techniken:

- Modellierung
- Zerlegung zusammengesetzter Probleme

Programmierung:

- Modellierung
- „Abstraktion, Abstraktion, Abstraktion“
(Grundtechnik zur Zerlegung zusammengesetzter Probleme)

Programmieren

(Realität)

$$f(x) := x + 5$$

```
PROGRAM f(INPUT, OUTPUT);
```

```
FUNCTION f(x : INTEGER) : INTEGER;
```

```
BEGIN
```

```
    f := x + 5;
```

```
END;
```

```
VAR x : INTEGER;
```

```
BEGIN
```

```
    ReadLn(x) ;
```

```
    WriteLn(f(x))
```

```
END.
```

Programmieren

(Realität)

$$f(x) := x + 5$$

```
PROGRAM f(INPUT, OUTPUT);
```

```
FUNCTION f(x : INTEGER) : INTEGER;
```

```
BEGIN
```

```
    f := x + 5;
```

```
END;
```

```
VAR x : INTEGER;
```

```
BEGIN
```

```
    ReadLn(x) ;
```

```
    WriteLn(f(x))
```

```
END.
```



Programmieren

(Realität)

- orientiert an Maschine
- 10% Problemlösung, 90% Syntax/Maschine

Informatik-Curriculum Baden-Württemberg

- Syntax
- Ein- / Ausgabe
- Zahlen, Zeichenketten
- Arithmetik
- Variablen und Zuweisung
- Anweisungsfolge, Verzweigung, Wiederholung
- Prozeduren und Funktionen
- Rekursion in einfachen Fällen
- strukturierte Datentypen

Programmieren und Mathematik

„Ist n Element der Liste l ?“

$$g(n, l) := \begin{cases} false & \text{falls } l \text{ leer} \\ true & \text{falls } n = \text{first}(l) \\ g(n, \text{rest}(l)) & \text{sonst} \end{cases}$$

Realität

TYPE

ListType = ^NodeType;

NodeType = RECORD

 First : String;

 Rest : ListType

END;

FUNCTION IsMember (Name : String; List : ListType)

: BOOLEAN;

BEGIN

 IF List = NIL

 THEN IsMember := FALSE

 ELSE IF Name = List^.First

 THEN IsMember := TRUE

 ELSE

 IsMember := IsMember(Name, List^.Rest)

END;

Realität

```
PROGRAM NameOnList(Input, Output) ;
VAR List : ListType;
    Name : STRING;

PROCEDURE GetList(Var LIST : ListType);
BEGIN
    ...
END;

FUNCTION IsMember ...

BEGIN
    ReadLn(Name);
    GetList(List);
    WriteLn(IsMember(Name, List))
END.
```

Allein am Computer

```
...  
y + 1 := x;  
...
```

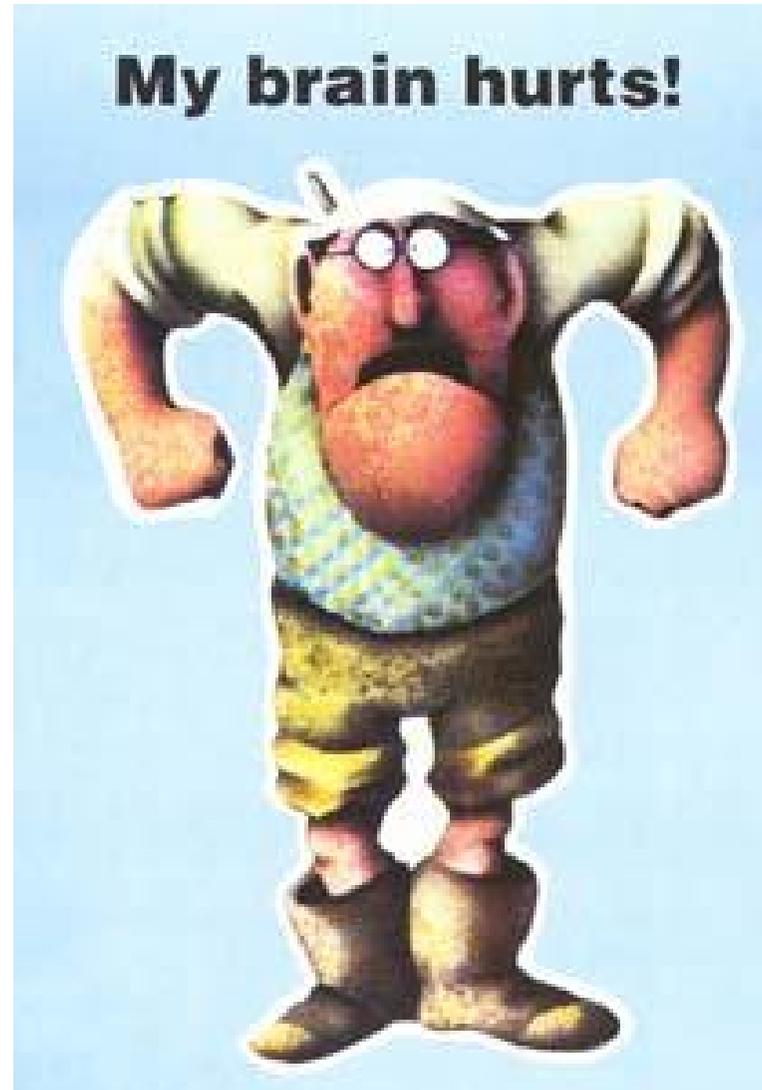
warning: expression used as a statement - value is ignored
parse error before '+'

Realität

- von-Neumann-Modell statt Algebra
- Syntax statt Prinzipien
- Zeitverschwendung mit Ein-/Ausgabe
- Basteln statt systematisches Problemlösen
- Werkzeuge, die für professionelle Programmierer gedacht sind

Leidiges Thema Rekursion

Fakultät, Potenz, Fibonacci ...



Bausteine

- Programmiersprache Scheme
- Programmierumgebung speziell für Anfänger (kostenlos zum Herunterladen)
- systematische Konstruktionsanleitungen

Syntax in Scheme

(Operation Operand₁ ... Operand_n)

$$4 + 5 \implies (+ 4 5)$$

$$(4 + 5) \cdot 6 \implies (* (+ 4 5) 6)$$

Definitionen in Scheme

(define (*Funktionsname* *Parameter*₁ ... *Parameter*_{*n*})
 rechte Seite)

$$f(x) := x + 5$$

```
(define (f x)  
  (+ x 5))
```

Scheme und Mathematik

$$g(n, l) := \begin{cases} false & \text{falls } l \text{ leer} \\ true & \text{falls } n = \text{first}(l) \\ g(n, \text{rest}(l)) & \text{sonst} \end{cases}$$

```
(define (guest name list)
  (cond
    ((empty? list) false)
    ((equal? name (first list)) true)
    (else (guest name (rest list)))))
```

10% Syntax, 90% Problemlösen

Programmierungsumgebung DrScheme

- entwickelt vom TeachScheme!-Projekt
- speziell für Anfänger
- läuft unter Windows, Mac OS, Linux, ...
- kostenlos herunterzuladen

Programmierumgebung für Anfänger

- mehrere Sprachebenen
- präzise Fehlermeldungen
- interaktive Auswertung
- Break-Knopf

Teachpacks

- Grafik
- grafische Benutzerschnittstellen
- interaktive Web-Server
- geplant: Messen/Steuern/Regeln, Robotik, ...

Programmkonstruktion

Eingabe



Problem

Ausgabe



My brain hurts!



Rekursion

„Es ist ganz einfach!“

```
(define (factorial n)
  (cond
    ((= n 1) 1)
    (else (* n
              (factorial (- n 1))))))
```

Rekursion

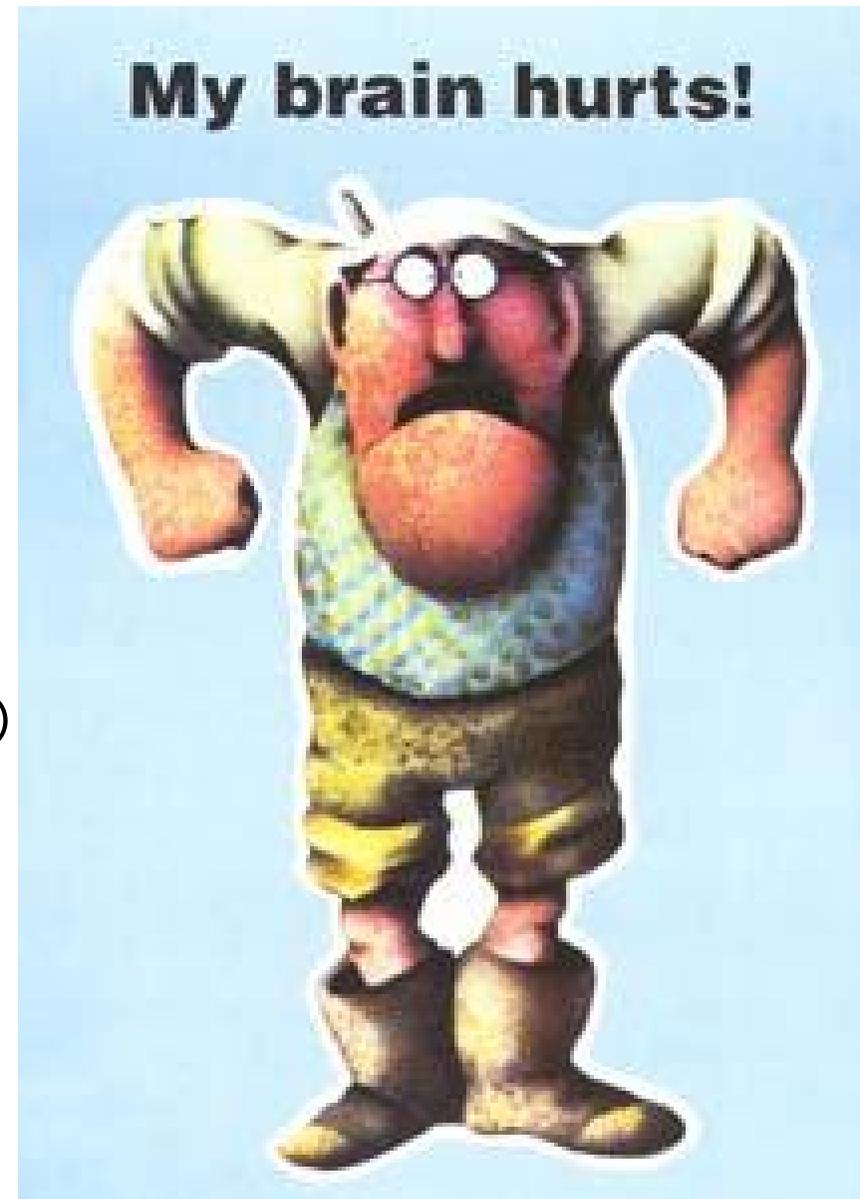
„Es ist ganz einfach!“

```
(define (fibonacci n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (fibonacci (- n 1))
              (fibonacci (- n 2))))))
```



Rekursion

```
(define (gcd a b)
  (cond
    ((= b 0) a)
    (else (gcd b (modulo a b))))
```



Konstruktionsanleitungen

1. Problemanalyse

- Verständnis
- Vertrag
- Beispiele/Testfälle

2. Funktionsdefinition

- Gerüst
- Schablone
- Rumpf

3. Korrektheitsprüfung

- Korrekturlesen
- Syntaxüberprüfung
- Test

Datengesteuerte Programmkonstruktion

„Die Daten bestimmen das Programm.“

zusammengesetzt	⇒	Selektion
gemischt	⇒	Fallunterscheidung
selbstbezüglich	⇒	rekursiver Aufruf
		...

⇒ *Wiederverwendbare Programmschablonen*

Rekursive Daten

Daten \implies Programmschablone

selbstbezügliche Daten \implies rekursive Programmschablonen

„Eine Liste von Symbolen ist ...

- die leere Liste `empty`, oder
- `(cons s l)`, wobei `s` ein Symbol und `l` eine Liste von Symbolen ist.“

Rekursion Nebenprodukt des Datenmodells

Informatik-Curriculum Baden-Württemberg

- Syntax
- Ein- / Ausgabe
- Zahlen, Zeichenketten
- Arithmetik
- Variablen und Zuweisung
- Anweisungsfolge, Verzweigung, Wiederholung
- Prozeduren und Funktionen
- Rekursion in einfachen Fällen
- strukturierte Datentypen

Inhalte

- Algebra, Funktionen
- Fallunterscheidungen
- Konstruktionsanleitungen
- Symbole
- Listen
- Strukturen („Records“)
- Grafik
- selbstbezügliche Strukturen

Ziele

- Programmieren als präzise Kommunikation über Problemlösungen
- Programmieren als Problemlösen
- Programmieren als Training für Mathematisierung
- Programmieren als Zugang zu Inhalten des Mathematik-Unterrichts

Voraussetzungen

- weniger Ketchup
- Werkzeuge für Anfänger, nicht professionelle Programmierer
- systematische Didaktik
mit nachvollziehbaren Anleitungen
- ansprechende Beispiele

dein Programm

Programmieren als Grundkompetenz im Schulunterricht

- Lehrmaterialien
- Weiterentwicklung von DrScheme
- Software für Unterrichtsbeispiele
- Lehrer-Aus- und -Fortbildung

Projektpartner

- Erwin Bauer
(freiberuflicher Systemanalytiker, Berater und Trainer)
- Prof. Herbert Klaeren, Uni Tübingen
- Prof. Helmut Schwichtenberg, LMU München
- TeachScheme!-Projekt in USA:
(Prof. Dr. Matthias Felleisen, Northeastern University)
 - viele Universitäten
 - > 250 ausgebildete Lehrer
- Lehrer in Baden-Württemberg, Bayern, Rheinland-Pfalz

Zusammenfassung

- Programmierausbildung birgt große Chancen
- ... aber:
 - mehr Ketchup als Kaviar
 - anfängeruntaugliche Software
 - Didaktik orientiert sich an Maschine, nicht am Anfänger



Zusammenfassung

- Programmierausbildung für alle ist möglich
- ... erfordert aber Umdenken
- ... Werkzeuge
- ... und systematische Didaktik

<http://www-pu.informatik.uni-tuebingen.de/deinprogramm/>