Modellierung und Programmierung 1

Prof. Dr. Sonja Prohaska

Computational EvoDevo Group
Institut für Informatik
Universität Leipzig

21. Oktober 2015

Automat *versus* Computer

Ein **Automat** ist eine Maschine, die ein eingeschränktes, vorbestimmtest Set von Operationen, üblicherweise Aktionen, selbsttätig ("automatisch") ausführen kann.
Beispiele: Kaugummiautomat, Geldautomat.

Ein **Computer**, auch "Rechner" genannt, ist ein Gerät, das **programmiert** werden kann, um eine Folge arithmetische oder logische Operationen selbsttätig auszuführen. Er kann **verschiedene** Aufgaben/Probelm lösen.

Ein Computer ermöglicht **E**lektronische **D**aten**V**erarbeitung (EDV).

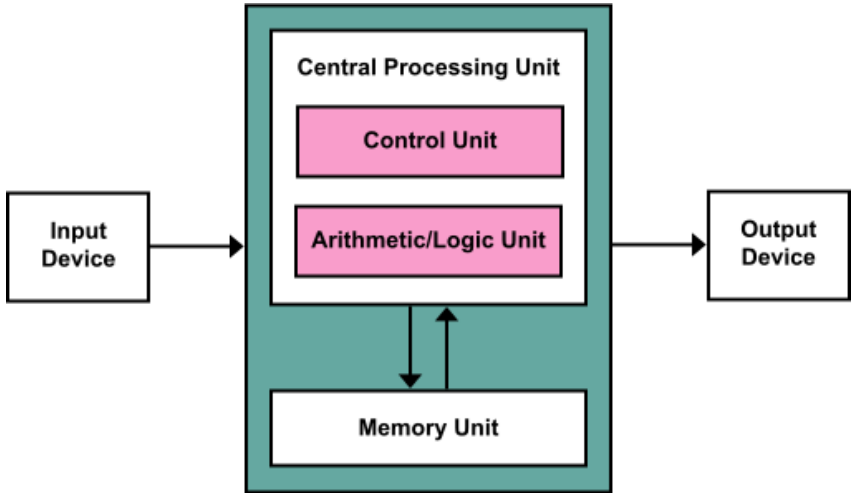
Hardware- und Softwarekomponenten

Hardware ist der Überbegriff für die **materiellen** Bestandteile eines Computers, die mechanische und elektronische Ausrüstung.
zum Beispiele: Motherboard, RAM, Festplatte, Tastatur, Maus, Monitor, Grafikkarte, Soundkarte...

Software ist der Überbegriff für die **immateriellen** Bestandteile eines Computers.
Sie umfasst maschinenlesbare Anweisungen (Programme) und Daten, die zusammen mit der Hardware, den Computer für ein definiertes Aufgabenspektrum nutzbar macht.
Beispiele: Betriebssystem, Texteditor, Bowser, Tetris
Software kann auch die Softwaredokumentation und zogehörige Daten (z.B. Schriftarten, Grafiken, Vorlagen) enthalten.

Hauptkomponenten eines Computer

Von-Neumann-Architektur ("Princton Architecture") (1945)



Die **CPU (central processing unit)** besteht aus

- **arithmetical and logical unit (ALU)**, z.d. *Rechenwerk*
  elektrische Schaltungen realisieren die Basisoperationen des Rechners
- **control unit (CU)**, z.d. *Steuerwerk*
  versorgt die ALU mit Anweisungen, und Daten aus dem Speicher

Ein Computer mit Von-Neumann-Architektur speichert Programm und Daten im selben Speicher.

Wie wird ein Problem in ein Program umgesetzt?

Mensch

Problem

natürliche Sprache

Kommunikation

Sprache

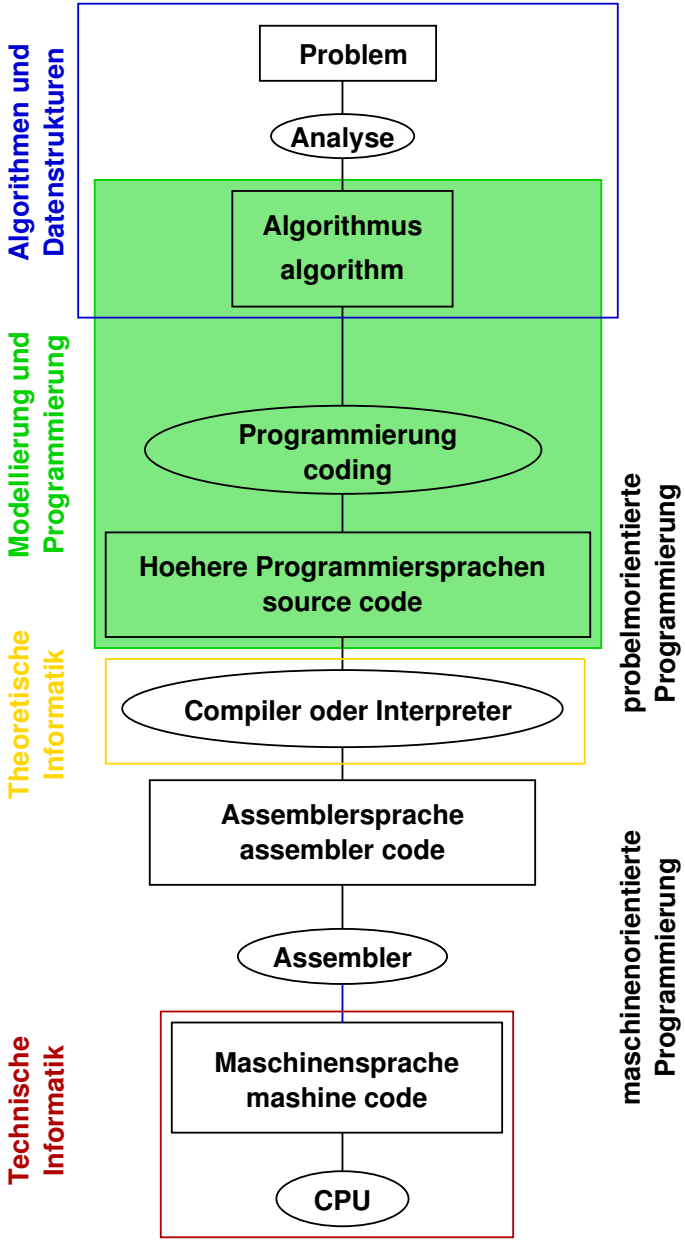
Uebersetzung

Computer

Programm

Maschinensprache

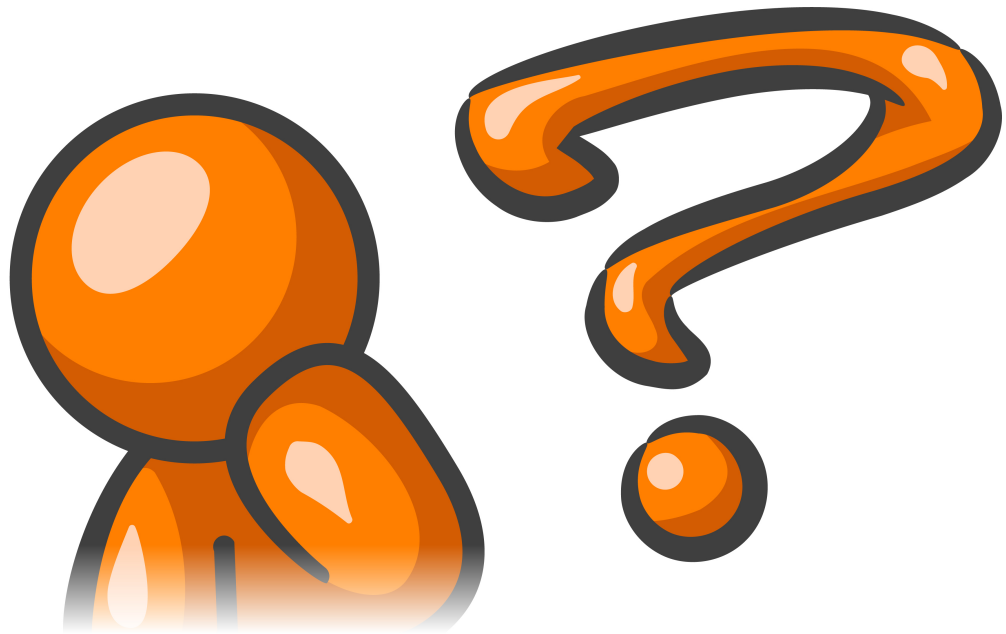
Hierarchie von Programmen



- Ein **Algorithmus** ist eine präzise, das heißt in einer festgelegten Sprache abgefasste, endliche Beschreibung eines **allgemeinen Verfahrens** unter Verwendung ausführbarer Verarbeitungsschritte zur Lösung einer Aufgabe.
- Ein **Programm** legt in einer bestimmten Programmiersprache fest, welche Anweisungen in welcher Reihenfolge zur Lösung der Aufgabe, unter Verwendung der vorhandenen (Basis-)Operationen, durchgeführt werden müssen.

Nehmen wir an, wir hätten das folgende Problem...

Aufgabe: Bestimme den größten gemeinsamen Teiler (ggT) r zweier natürlicher Zahlen m und n (wobei m größer n).



Der größten gemeinsamen Teiler. Hmm, was war denn das, wie ging denn das?

Beispiel: Euklidscher Algorithmus

Aufgabe: Bestimme den größten gemeinsamen Teiler (ggT) r zweier natürlicher Zahlen m und n (wobei m größer n).

Algorithmus in natürlicher Sprache (deutsch):

Im ersten Schritt dividiere m durch n und bestimme Quotient und Rest. In jedem weiteren Schritt wird mit dem Divisor und dem Rest des vorhergehenden Schritts eine erneute Division mit Rest durchgeführt. Und zwar so lange, bis der Rest Null ist. Der Divisor der letzten Division ist der größten gemeinsamen Teiler von m und n.

Dividend Divisor Quotient

$2244 \;/\; 21 = 106$

14

144

18 Rest

$2244 = 106 \times 21 + 18$

Beispiel: Euklidscher Algorithmus

Aufgabe: Bestimme den größten gemeinsamen Teiler (ggT) r zweier natürlicher Zahlen m und n (wobei m größer n).

Algorithmus in formaler Sprache:

$$m; r_0 = n;$$

$$\begin{aligned} m &= q_1 \times r_0 + r_1 \\ r_0 &= q_2 \times r_1 + r_2 \\ r_1 &= q_3 \times r_2 + r_3 \\ &\vdots \\ r_{n-1} &= q_{n+1} \times r_n + 0 \end{aligned}$$

$$ggT(m, n) = r_n$$

Beispiel: Programm

Achtung! Bevor wir mit dem Programmieren loslegen, müssen wir wissen, mit welchen **Basisoperationen** und **Arbeitsanweisungen** wir arbeiten können.

d.h. welche Basisoperationen kann der Computer durchführen und welche Arbeitsanweisungen versteht er?

Als Beispiel sei folgender **Modellrechner** gegeben...

Beschreibung des Modellrechners

Universität Leipzig

Institut für Informatik
Dr. Monika Meiler

Speicher $M \subset N$**, nach oben beschränkt, damit endlich.**

constant	`0, 123`	Konstante
identifier	`a, b, z`	Variable

Basisoperationen

***Rechenoperationen*: Addition +, Subtraktion -, Multiplikation *, ganzzahlige Division /, Rest %**

expression	`a % b`	Ausdruck

***Vergleichsoperationen*: größer >, gleich == und ungleich !=**

condition	`a != 0`	Vergleichsausdruck

Beschreibung des Modellrechners

Universität Leipzig

Institut für Informatik
Dr. Monika Meiler

Arbeitsanweisungen

1. *Wertzuweisungen* als *einfachste Arbeitsanweisungen:*

assignment

```
a = 10;
b = a;
c = a * b;
```

Zuweisung

2. *Zusammengesetzte Arbeitsanweisungen:*

a. Mehrere Arbeitsanweisungen können *hintereinander* ausgeführt werden.

sequence

```
a = m; b = n; c = a % b;
```

Abfolge

b. Arbeitsanweisungen können *wiederholt* ausgeführt werden.

iteration

```
while( n != 0) { s = s + n; n = n - 1;}
```

Schleife
'solange'

c. Eine *Auswahl* von Arbeitsanweisungen soll ausgeführt werden.

selection

```
if( n > m) { c = m; m = n; n = c;}
if( m > n) { x = m;} else { x = n;}
```

Auswahl
'wenn-dann'
'wenn-dann-sonst'

3. Abschluss: Nichts sonst ist erlaubt.

ggT-Program – Version 1

Aufgabe: Bestimme den größten gemeinsamen Teiler r zweier natürlicher Zahlen m und n (wobei m größer n).

Universität Leipzig

Institut für Informatik
Dr. Monika Meiler

Programm $r = ggT(m, n)$

```
a = m;
b = n;
c = a % b;
while( c != 0)
{
  a = b;
  b = c;
  c = a % b;
}
r = b;
```

Protokoll (Speicherbelegungsänderung)

	m	*n*	*r*	*a*	*b*	*c*
Eingabe						
Verarbeitung						
Ausgabe						

ggT-Program – Version 1

Aufgabe: Bestimme den größten gemeinsamen Teiler r zweier natürlicher Zahlen m und n (wobei m größer n).

Universität Leipzig

Institut für Informatik
Dr. Monika Meiler

Programm $r = ggT(m, n)$

```
a = m;
b = n;
c = a % b;
while( c != 0)
{
  a = b;
  b = c;
  c = a % b;
}
r = b;
```

Protokoll (Speicherbelegungsänderung)

	m	*n*	*r*	*a*	*b*	*c*
Eingabe						
Verarbeitung						
Ausgabe						

Institut für Informatik
Dr. Monika Meiler

Protokoll (Speicherbelegungsänderung)

	m	*n*	*r*	*a*	*b*	*c*
Eingabe	130	55				
Verarbeitung				130	55	20
				55	20	15
				20	15	5
				15	5	0
Ausgabe			5			

Beispiel: Programm – ggT Version 2

Rekursion (ein Programm bzw. Teilprogramm ruft sich selber auf)

Universität Leipzig — Institut für Informatik
Dr. Monika Meiler

Programm $r = ggT(m, n)$ mit Rekursion

```
a = m;
b = n;
if( b != 0)
{
  r = ggT( b, a % b);
}
else
{
  r = a;
}
```

Rekursion, problemorientiert

Zu *einem* Algorithmus kann man *verschiedene* Programme formulieren, zum einen in Abhängigkeit von den vorhandenen Basisoperationen, zum anderen aber auch in Abhängigkeit von den verwendeten Arbeitsanweisungen.

Programmsyntax

Die **Syntax eines Programms** definiert die Struktur eines Programs, die Möglichkeiten der Zusammensetzung von Basisoperationen und Anweisungen.

Übliche Formen der Darstellung sind

- die **Backus-Naur-Form (BNF)** und
- die **Syntaxdiagramme** ("railroad presentation of syntax")

Tritt eine Abfolge von Opertationen und Anweisungen auf, die von der Syntax nicht vorgesehen ist, so kommt es zu einem Syntaxfehler (Syntax error.)

Modellrechner – Backus-Naur-Form

Universität Leipzig

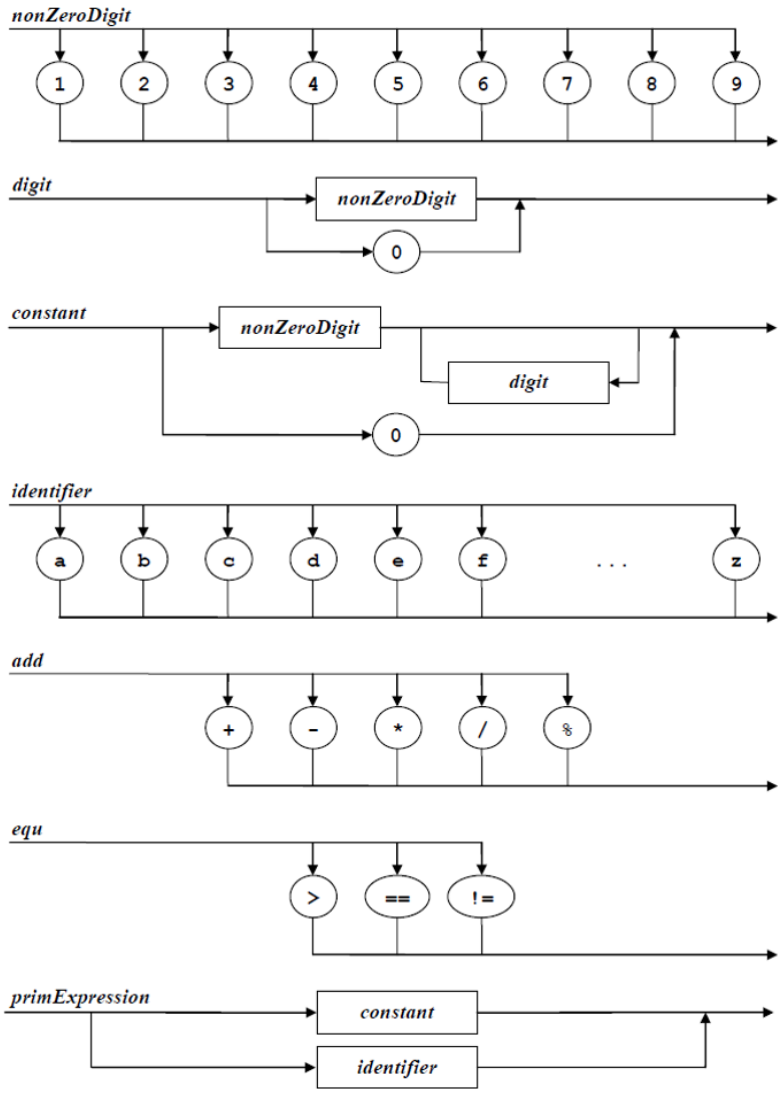
Institut für Informatik
Dr. Monika Meiler

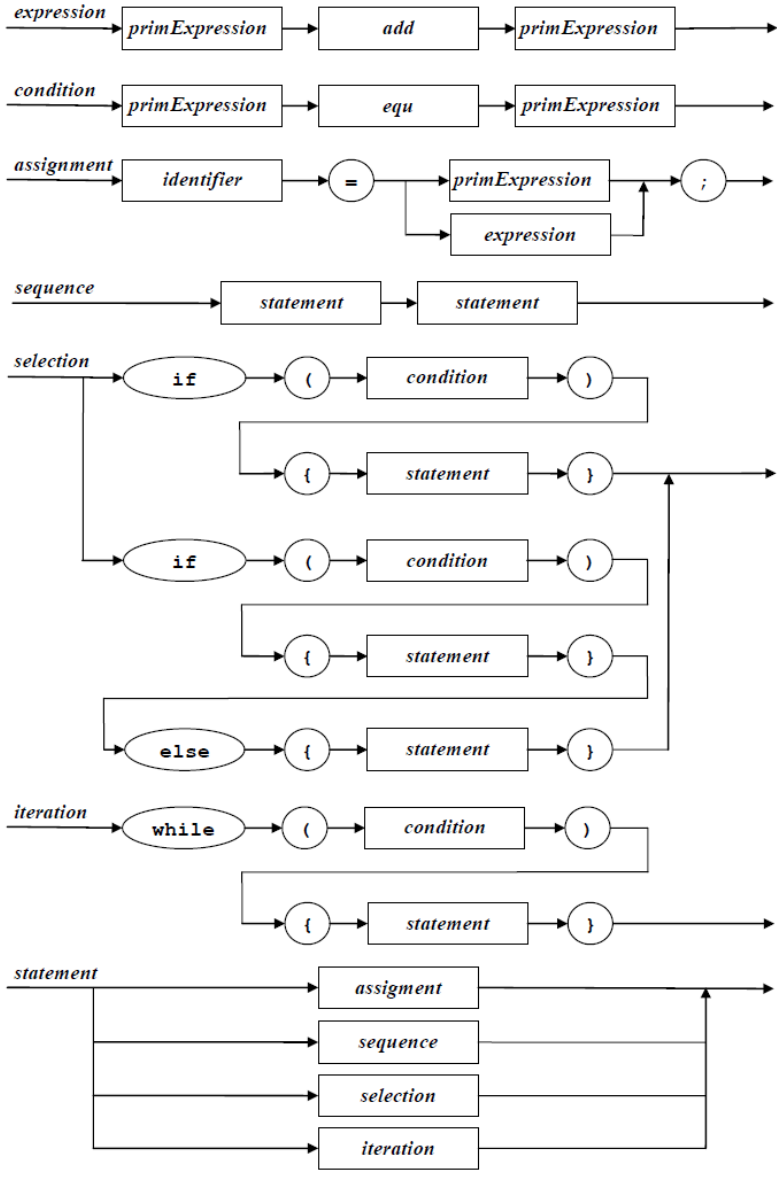
nonZeroDigit	::=	'1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'	
digit	::=	'0' | *nonZeroDigit*	**Ziffern**
constant	::=	'0' | *nonZeroDigit* { *digit* }	**Konstante**
identifier	::=	'a' | 'b' | 'c' | 'd' | 'e' | 'f' | ... | 'z'	**Variable**
add	::=	'+' | '-' | '*' | '/' | '%'	**Rechenoperator**
equ	::=	'>' | '==' | '!='	**Vergleichsoperator**
primExpression	::=	*constant* | *identifier*	**einfache Ausdrücke**
expression	::=	*primExpression add primExpression*	**Rechenausdruck**
condition	::=	*primExpression equ primExpression*	**Vergleichsausdruck**
assignment	::=	*identifier* '=' *primExpression* ';' | *identifier* '=' *expression* ';'	**Wertzuweisung**
sequence	::=	*statement statement*	**Hintereinanderausführung**
selection	::=	'if' '(' *condition* ')' '{' *statement* '}' | 'if' '(' *condition* ')' '{' *statement* '}' 'else' '{' *statement* '}'	**Auswahlanweisung**
iteration	::=	'while' '(' *condition* ')' '{' *statement* '}'	**Schleifenanweisung**
statement	::=	*assignment* | *sequence* | *selection* | *iteration*	**Programm**

Modellrechner – Syntaxdiagramme

Universität Leipzig

Institut für Informatik
Dr. Monika Meiler





Programmierparadigmen

- Imperative Programmierung
- Funktionale Programmierung
- Logische Programmierung
- Objektorientierte Programmierung
- hybride Programmiersprachen

Imperative Programmierung

Imperative von lat. *imperare*, “anordnen”, “befehlen”

Ein Programm besteht aus einer **Folge von Anweisungen**, die vorgeben, in welcher Reihenfolge was vom Computer getan werden soll.
Variablen, Werzuweisungen und Schleifen stellen die wichtigsten Konzepte dar.

Prozedurale Programmierung ist eine Erweiterung der ipmerative Programmierung. Sie erlaubt die Zerlegung eines Algorithmus in Teile, die “Prozeduren”, “Unterprogramme”, “Routinen” oder “Funktionen” genannt werden.

Beispiele: `ALGOL`, `Fortran`, `Pascal`, `Perl`, `C`

Funktionale Programmierung

Das Programm ist eine **mathematischen Funktion**.

Eine funktionale Programmiersprache ist eine Programmiersprache, die Elemente zur Kombination und Transformation von **Funktionen** anbietet.

Sie verwendet Funktionen und Rekursionen, kommt aber ohne Schleifen und Zuweisungen aus.

Beispiele: `Lisp` bzw. `Scheme`, `Haskell`, `Ocaml`

Logische Programmierung

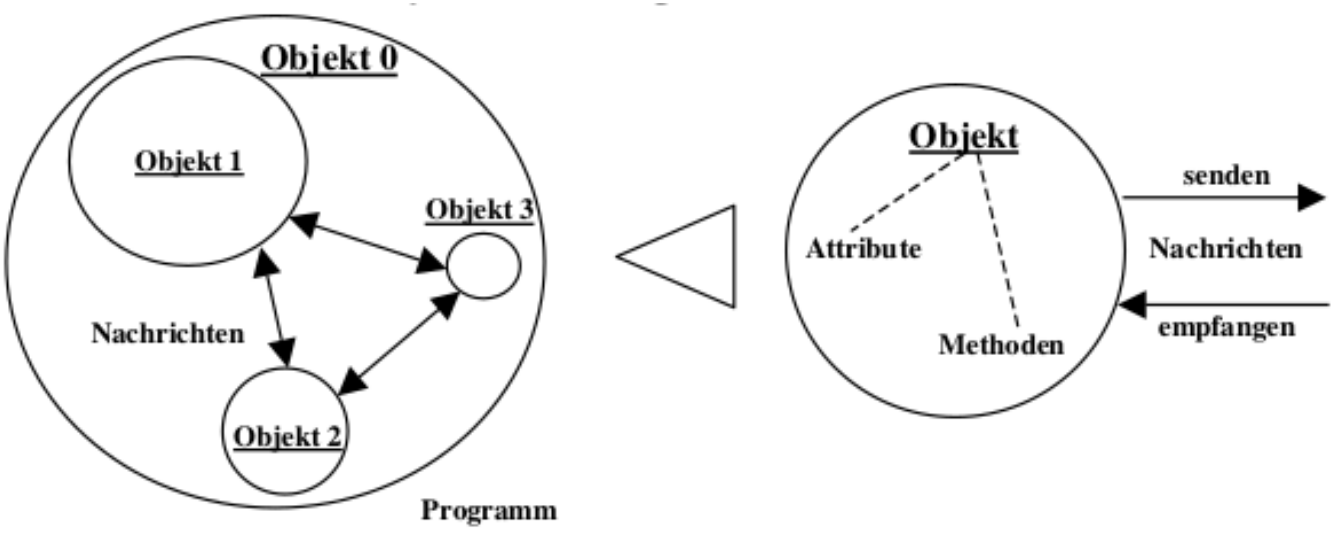
Das Programm ist eine **Frage**.

Logische auch prädikative Programmierung genannt, basiert auf der **mathematischen Logik**. Ein Programm besteht aus einer Menge von Axiomen. Die Lösungsaussage wird alleine aus den Axiomen berechnet und beantwortet eine Anfrage bei Korrektheit mit “ja” und bei Fehlschlag mit “nein”.

Beispiele: `Prolog`

Objektorientierte Programmierung

Ein Programm ist ein **Objekt**, welches selbst aus Objekten aufgebaut ist. Der Datenaustausch zwischen den Objekten erfolgt über Nachrichten.
Es enthält Informationen über die auftretenden Objekte und deren Typen. Wichtige Konzepte, insbesondere Klassen und Vererbung, werden verwendet.



Beispiele: `Smalltalk` (rein objektorientiert)

hybride Programmiersprachen

Hybridsprachen unterstützen sowohl das Programmierparadigma der prozeduralen als auch der objektorientierten Programmierung. Üblicherweise sind dies Sprachen, die zunächst das **imperative Paradigma** verfolgten, und nachträglich eine **objektorientierte Erweiterung** erhielten.

Beispiele: `Objekt Pascal`, `C++`, `Perl`, `Java`