Programmieren

Stichpunkte, quick & dirty …

Was heißt „programmieren“?

Es wird ein Programm erstellt, das eine definierte Aufgabe erfüllt

Eine Programmdatei enthält die Anweisungen, die der Computer ausführen soll

- Anweisungen müssen so formuliert sein, dass die Aufgabe gelöst wird
- Anweisungen werden vom Computer nacheinander („sequenziell“) abgearbeitet
- Was in den Anweisungen nicht formuliert wird, wird auch nicht ausgeführt
- Die Anweisungen müssen so formuliert sein, dass ein Programm zuverlässig beendet wird

EVA

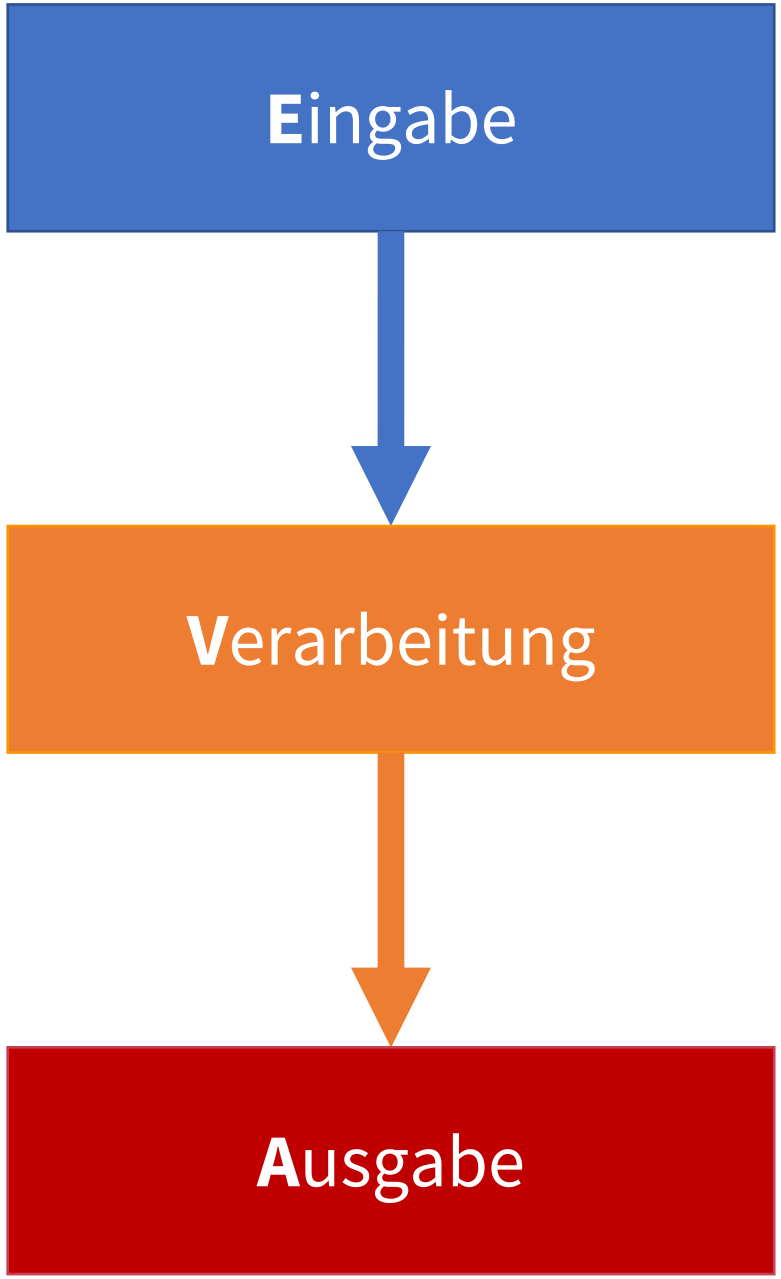
Praktisch alle Programme
arbeiten nach dem „EVA“-Prinzip

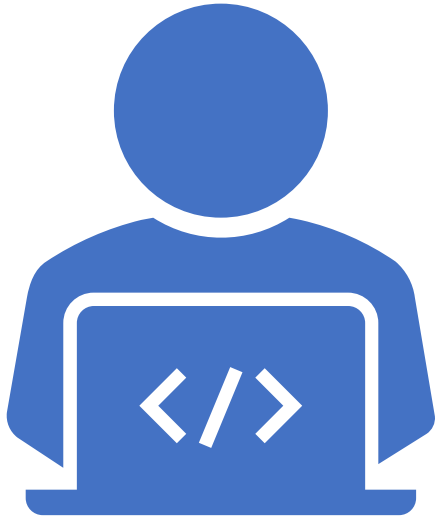
EVA ist die Abkürzung für das
grundlegende Prinzip aller
elektronischen Datenverarbeitung

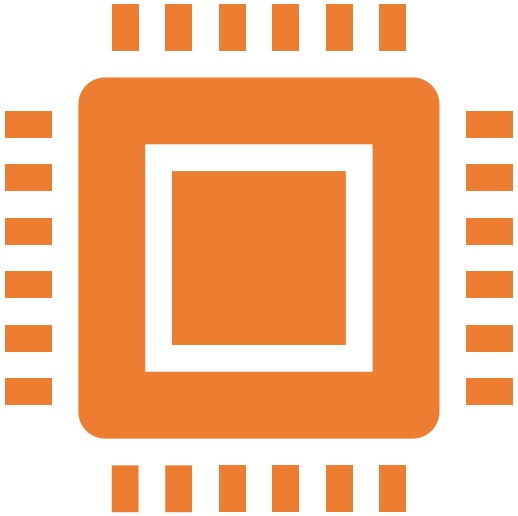
E: Eingabe

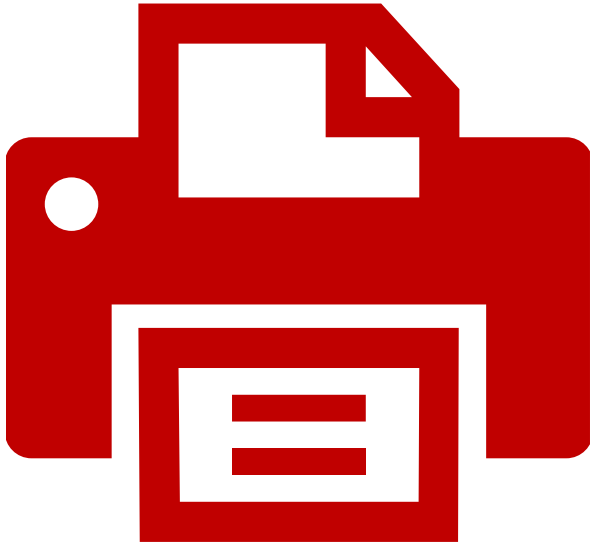
V: Verarbeitung

A: Ausgabe









E

Eingabe

Tastatur, Maus, Mikro, Scanner, Kamera …

V

Verarbeitung

Im Innern des Computers

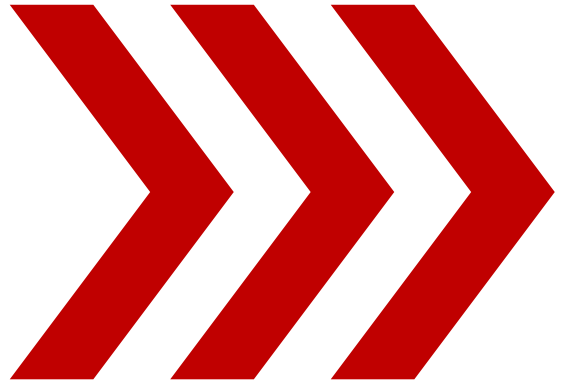
A

Ausgabe

Monitor, Drucker, Lautsprecher …







E

Eingabe

Ein Programm nimmt Daten entgegen

V

Verarbeitung

Die Daten werden im Programm verarbeitet

A

Ausgabe

Weitergabe der Daten an anderes Programm

Der Weg zum Programm

Das Problem / die Anforderungen werden analysiert und detailliert erfasst

Es wird ein Lösungsweg (der „Algorithmus“) formuliert (Anweisungen, die der Reihe nach abgearbeitet werden müssen, um das Problem zu lösen)

Der Lösungsweg wird in einer konkreten Programmiersprache formuliert (wir erzeugen den „Quellcode“)

Der Quellcode wird in die Sprache übersetzt, die der Computer beherrscht und ausführen kann („Maschinencode“)

Algorithmus

Abstrakte, schrittweise Formulierung des Lösungswegs (Rechenweg, Vorgehensweise, Verfahren, „Kochrezept“ …)

- Endlicher Text, der den Lösungsweg vollständig beschreibt
- Jede Anweisung muss zu einem eindeutigen Ergebnis führen
- Gleiche Eingabewerte müssen das gleiche Ergebnis liefern
- Die Anweisungen müssen zum richtigen Ergebnis führen
- Das Verfahren muss allgemeingültig sein
- Das Verfahren muss auf alle zulässigen Daten anwendbar sein
- Das Verfahren muss ein eindeutiges Ende haben

Programmentwurf

Zur Notation der Anforderungen und des Lösungswegs werden verschiedene, streng formalisierte Diagramme / Texte benutzt

Drei Grundstrukturen werden (mindestens) benötigt:

- **Anweisungsfolgen** (Sequenz): Die Anweisungen werden der Reihe nach abgearbeitet
- **Auswahlstrukturen** (Selektion, Bedingungen): Anweisungen werden nur unter bestimmten Bedingungen ausgeführt
- **Wiederholungsstrukturen** (Iteration, „Schleifen“): Anweisungen werden so lange wiederholt, solange eine bestimmte Bedingung erfüllt ist („Abbruchbedingung“)

Pseudocode

Formulierung eines Algorithmus in natürlicher oder in einer an eine Programmiersprache angelehnten Sprache

Keine festen Regeln

Mix aus natürlicher Sprache und mathematischen Formulierungen

Pseudocode soll den Ablauf und die Struktur verdeutlichen, unabhängig von der konkreten Umsetzung

Gewissermaßen das Esperanto der Programmierer …

```
Lies Wert für m
Lies Wert für n
Berechne: r = m % n

Solange r > 0
  m = n
  n = r
  r = m % n
Wiederhole

Gib n aus
```

Programmablaufplan

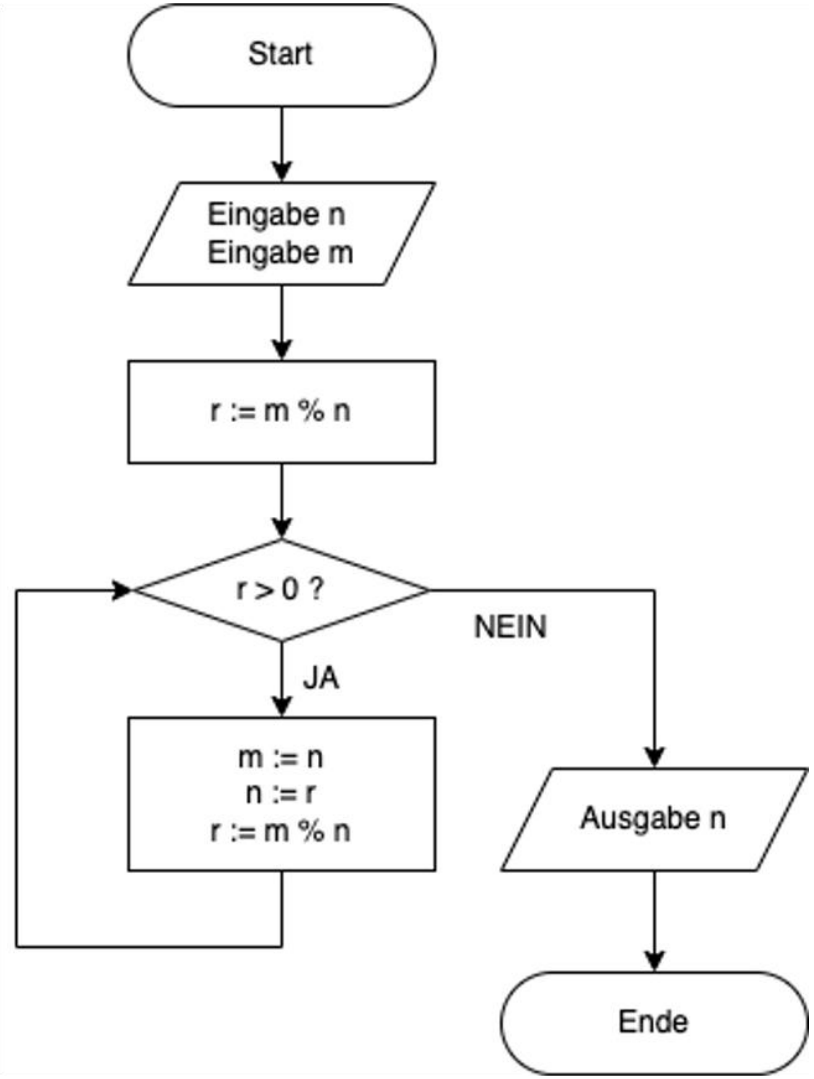
PAP (DIN 66001)

Flussdiagramm (Flow Chart)

Anweisungen stehen in Kästchen, die Pfeile geben die Laufrichtung an

Hier: Berechnung des größten gemeinsamen Teilers zweier Zahlen

Wird heute eher selten benutzt



Struktogramm

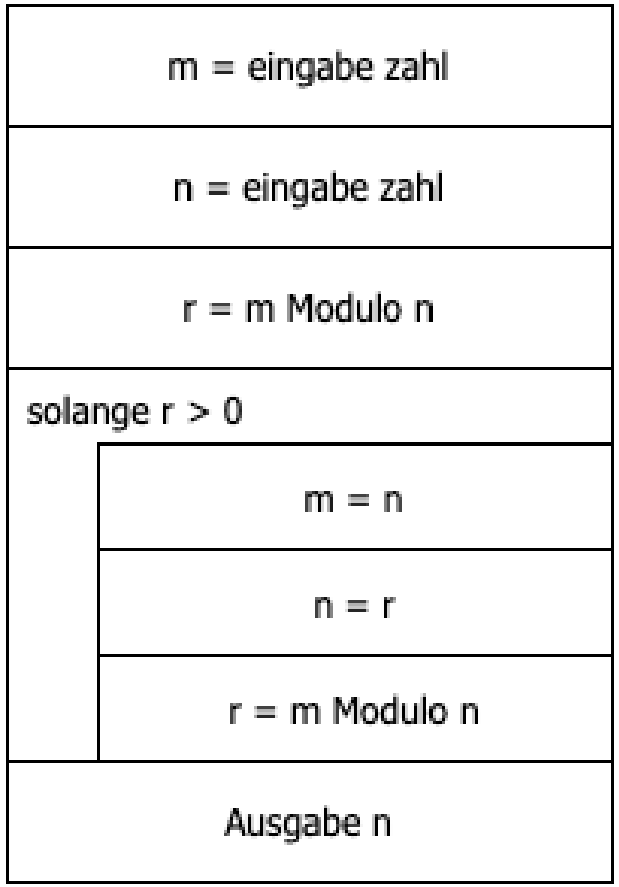
Nassi-Shneiderman-Diagramm

(Isaac Nassi, Ben Shneiderman)

DIN 66261

Jede Anweisung steht in einem Rechteck

Wird heute nur noch zu Übungszwecken eingesetzt, veraltet



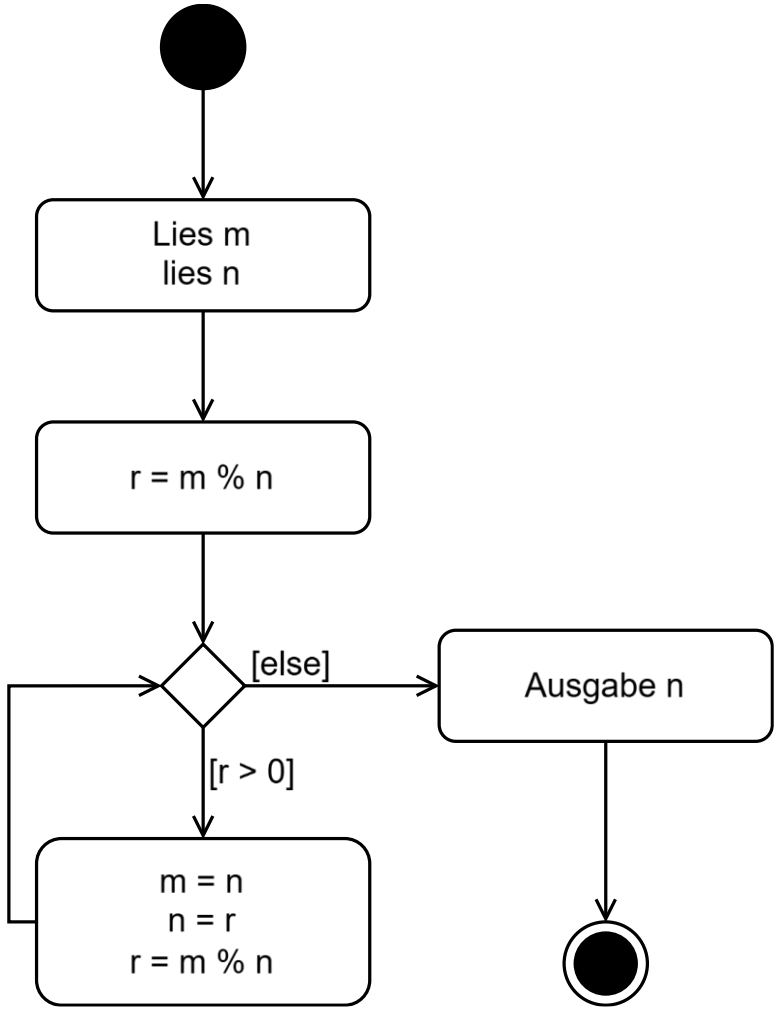
Aktivitätsdiagramm (UML)

Eine moderne Version der Programmablaufpläne

Anweisungen stehen in Kästchen, die Pfeile geben die Laufrichtung an

Nicht im Beispiel:

- Parallele Prozesse lassen sich modellieren
- Es kann *mehrere* Start- und *mehrere* Endpunkte geben



Sprachprobleme

Damit ein Programm vom Computer (genauer: von der CPU) ausgeführt werden kann, muss es in einer Sprache vorliegen, die der Computer „versteht“

Ein Prozessor verarbeitet ausschließlich seinen eigenen „Maschinencode“ (spezifische Bytefolgen)

Problem: Wir verstehen keinen Maschinencode

Lösung: Wir schreiben das Programm in einer Sprache, die wir verstehen, in Form eines „Quellcodes“, der anschließend von einem besonderen Programm (Compiler, Interpreter) in Maschinensprache übersetzt wird

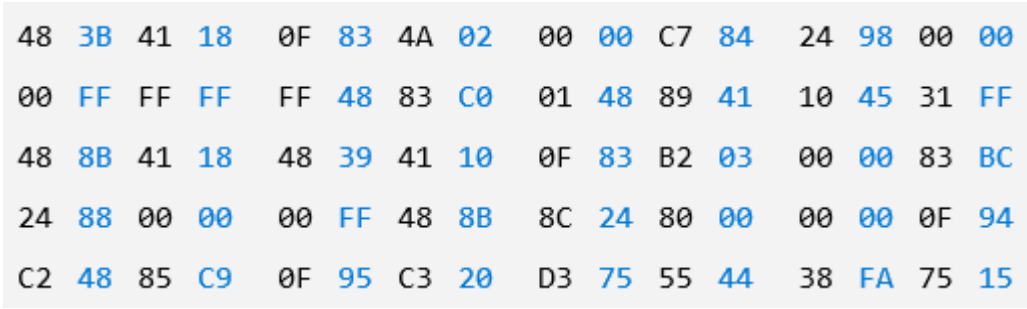
Quellcode & Maschinencode

Maschinencode ist für Menschen nicht lesbar

Programme werden in einem Texteditor als menschenlesbarer „Quellcode“ geschrieben

Quellcode wird von einem speziellen Programm (Compiler, Interpreter) in Maschinencode überführt

Nur Maschinencode wird vom Prozessor verarbeitet



```
48 3B 41 18  0F 83 4A 02  00 00 C7 84  24 98 00 00
00 FF FF FF  FF 48 83 C0  01 48 89 41  10 45 31 FF
48 8B 41 18  48 39 41 10  0F 83 B2 03  00 00 83 BC
24 88 00 00  00 FF 48 8B  8C 24 80 00  00 00 0F 94
C2 48 85 C9  0F 95 C3 20  D3 75 55 44  38 FA 75 15
```

```
let a = 5;
let b = 7;
let c = a + b;
console.log(`${a} + ${b} = ${c}`);
```

Programmiersprachen

Programmiersprachen sind streng formalisierte Sprachen, in denen wir Quellcode schreiben, der anschließend in Maschinencode übersetzt wird

Streng formalisiert: Selbst der kleinste Tippfehler führt zu fehlerhaftem Verhalten des Programms

Es gibt Hunderte von Programmiersprachen (die Wikipedia verzeichnet rund 700) mit jeweils spezifischen Regeln und Besonderheiten

Frühere Programmiersprachen setzten eine spezifische Hardware voraus („maschinenahe Sprachen“, „Assembler“)

- **Assembler:** Sprachen, die ausschließlich auf den Befehlssatz einer bestimmten CPU ausgerichtet sind, schwer lesbar, sehr fehleranfällig, dafür maximale Geschwindigkeit
- **Hochsprachen:** Unabhängig von konkreter Hardware, gut lesbar, erlaubt sehr komplexe Programme, die mit Assembler zwar theoretisch möglich, in der Praxis aber unmöglich sind

Sprachen: Assembler (hier: C64)

Maschinennahe Sprache

Jede CPU besitzt ihre eigene Assemblersprache

Assembler-Code ist nicht von einer CPU zur anderen übertragbar

Der Quellcode wird vom eigentlichen Assembler in ausführbaren Maschinencode übersetzt

```
; Hello World for 6502 CPU (C64)
; Assembler used: win2c64
    LDY #$00     ; Y-Register mit 0 initialisieren
L0
    LDA L1,Y     ; Das Y. Byte aus L1 laden
    BEQ L2       ; Wenn Byte = 0 springe zu L2
    JSR $FFD2    ; Systemroutine zur Zeichenausgabe
    INY          ; Inhalt von Y um 1 erhöhen
    JMP L0       ; Springe zu L0
L1
    .byte "hello world",0
L2
    RTS          ; "Return from Subroutine" = Ende
```

```
LDA, LDY: Load Accumulator, Y-Register
BEQ: Branch on Equal
JSR: Jump to Subroutine
INY: Increment Y
JMP: Jump
```

Sprachen: Hochsprachen

Unabhängig von der konkreten Hardware

Es gibt generell zwei Möglichkeiten, wie aus Quellcode ein Programm wird:

- Quellcode wird über einen **Compiler** in den Maschinencode des Zielsystems übersetzt (= erzeugt eine ausführbare Datei)
- Der Quellcode wird bei jedem Aufruf erneut in Maschinencode übersetzt (Scriptsprachen, **Interpreter**)

Compiler vs. Interpreter

Compiler

Erzeugt direkt ausführbare Datei

Übersetzt kompletten Code

Fehleranzeige während Kompilierung

Schnelle Ausführung

Komplexe Entwicklung

Läuft nur auf dem jeweiligen Zielgerät

C, C++, C#, Pascal …

Interpreter

Übersetzt bei jedem Aufruf neu

Arbeitet zeilenweise

Fehleranzeige bei der Ausführung, Abbruch

Langsame Ausführung

Einfache Entwicklung

Benötigt jeweils eine Laufzeitumgebung

JavaScript, PHP, Perl, BASIC …

Syntax

Als **Syntax** wird der Satzbau (die Grammatik) einer Sprache bezeichnet

Wie alle Sprachen haben auch Programmiersprachen Regeln zur korrekten Kombination von Wörtern (bzw. Anweisung) und Satzzeichen (Syntax-Regeln)

Verstößt man in einer natürlichen Sprache gegen die Syntax-Regeln, sorgt das allenfalls für kurze Irritation – verstanden wird man trotzdem

Das ist bei Programmiersprachen anders: Ein Verstoß gegen die Syntax-Regeln führt zu sofortigem Abbruch

Ein Verstoß gegen die Sprachregeln ist ein **Syntax-Error**, der in der Regel zum Absturz eines Programms führt – dafür reicht z.B. schon ein `,` statt einem `;`

Syntax-Fehler werden in der Regel schon während der Entwicklung erkannt und vom Editor markiert

Semantik

Sprache folgt nicht nur Grammatik-Regeln, sondern überträgt Bedeutung. Die Lehre der Bedeutung von Sprache heißt **Semantik**

In natürlichen Sprachen wird die Bedeutung über eine Vielzahl von Faktoren bestimmt, bei Programmiersprachen bestimmen formale Regeln und Definitionen die Bedeutung

Es wird etwa festgelegt, dass `3 + 8` die Bedeutung hat: „Addiere die Zahl 3 mit der Zahl 8“

Auch bei der Semantik können Fehler auftreten – blöderweise fallen dieser Fehler in der Regel erst während der Laufzeit des Programms auf

Semantik-Fehler liegen vor, wenn der Code zwar syntaktisch korrekt ist, aber nicht zum gewünschten Ergebnis führt