Vom Problem zum Plan: Die Logik hinter dem Code

Bevor wir programmieren, müssen wir denken. Logisch denken. Wir schauen uns an, wie man aus einem alltäglichen Wunsch oder Problem einen strukturierten Plan macht – und diesen dann mit Hilfe von Code umsetzt.

Alles beginnt mit einem Wunsch

Jede App, jede Website, jedes Programm beginnt mit einem Wunsch oder einem Problem. Die Aufgabe von uns als Programmierer:innen ist es, diese Wünsche in konkrete, funktionierende Software zu übersetzen – Schritt für Schritt.

1

Das Bedürfnis

"Ich möchte meine Finanzen jederzeit im Blick behalten und Ausgaben kategorisieren können."

2

Die digitale Lösung

Eine mobile Banking-App entwickeln, die Transaktionen automatisch erfasst und Budget-Tools bietet.

Oft lassen sich komplexe Anliegen in einfache Schritte oder Funktionen zerlegen, die dann programmiert werden können.

Freunde verbinden

Der Wunsch, sich mit Freunden auszutauschen und Erlebnisse zu teilen, führt zur Entwicklung von Social-Media-Plattformen mit Chat- und Foto-Funktionen.

Das Wetter wissen

Die Notwendigkeit, das aktuelle Wetter und Vorhersagen schnell abzurufen, wird durch spezialisierte Wetter-Apps mit Echtzeitdaten erfüllt.

Musik entdecken

Der Wunsch, jederzeit auf eine riesige Bibliothek an Musik zuzugreifen und neue Künstler zu entdecken, wird durch intuitive Streaming-Apps möglich gemacht.

Ohne Plan kein Code

1

Programmieren ist Denken in Lösungen

Bevor eine einzige Zeile Code geschrieben wird, muss das Problem vollständig verstanden und analysiert werden.

2

Planloses Coden führt zu digitalem Chaos

Ohne klare Struktur entstehen fehlerhafte, schwer wartbare Programme, die mehr Probleme schaffen als lösen.

3

Gute Programme starten mit einer klaren Idee

Der erste Schritt ist immer: Problem verstehen, analysieren und in einen strukturierten Plan übersetzen.

Ein Rezept ist ein Algorithmus

Kuchen backen Schritt für Schritt

1. Heize den Ofen auf 200°C vor
2. Mische Mehl, Zucker und Backpulver
3. Gib Eier und Milch dazu
4. Rühre alles zu Teig
5. Backe 45 Minuten

Das ist ein Algorithmus: eine eindeutige, endliche Abfolge von Schritten zur Lösung eines Problems. Der Computer ist wie ein extrem genauer, aber auch extrem wörtlicher Koch – wenn ein Schritt fehlt oder unklar ist, "verkocht" er das Ergebnis.

Was ist ein Algorithmus?

Eindeutig

Jeder Schritt ist klar und unmissverständlich formuliert

Endlich

Er hat ein klares Ende und läuft nicht unendlich weiter

Reproduzierbar

Führt bei gleichen Eingaben immer zum gleichen Ergebnis

Zielgerichtet

Löst ein konkretes, definiertes Problem

Wenn ihr später Programme schreibt, schreibt ihr im Grunde: Algorithmen. Die Kunst besteht darin, Abläufe so präzise und logisch zu formulieren, dass ein Computer sie zuverlässig umsetzen kann.

Was ist ein Programm?

Algorithmus in Programmiersprache

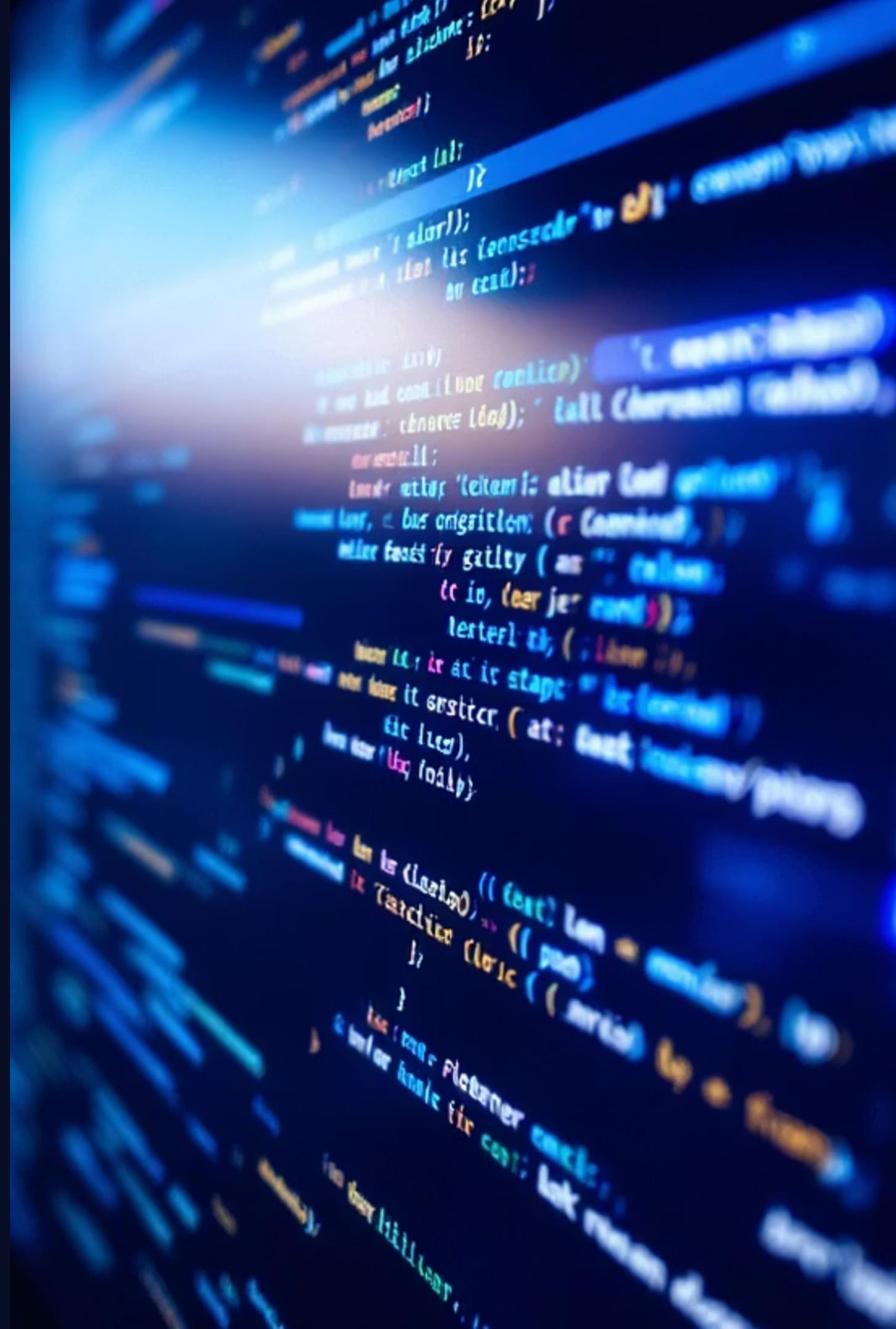
Ein strukturierter Plan wird in eine für Computer verständliche Sprache übersetzt

Für Menschen oft unverständlich

Code folgt strengen Regeln und Syntax, die erst erlernt werden müssen

Für Maschinen klar und direkt ausführbar

Computer können die Anweisungen ohne Interpretation oder Mehrdeutigkeit abarbeiten



Programmiersprachen – Werkzeuge für Ideen

Was sind Programmiersprachen?

Spezielle "Sprachen", um Anweisungen an Computer zu geben.

Sie übersetzen menschliche Absichten in maschinenlesbaren Code.

Jede Sprache hat ihre eigene Syntax (Grammatik), aber teilt ähnliche logische Konzepte.

Betrachtet Programmiersprachen als Werkzeuge: Jedes hat spezifische Stärken und Anwendungsbereiche, um eine Aufgabe zu lösen.

Wichtiger als das spezifische Werkzeug ist das Verständnis des Problems und die Fähigkeit, logisch eine Lösung zu entwickeln.

Abstraktionslevel verstehen

1

Maschinencode (Low Level)

Direkte Anweisungen für den Prozessor, bestehend aus Binärzahlen (0en und 1en). Für Menschen extrem schwer zu lesen und zu schreiben.

2

Assemblersprachen

Eine symbolische Repräsentation von Maschinencode. Nutzt mnemonische Codes (z.B. MOV für 'move') statt reiner Binärzahlen, ist aber immer noch sehr hardwarenah.

3

Kompilierte Sprachen (z.B. C, C++)

Programme werden vor der Ausführung von einem Compiler in Maschinencode übersetzt. Sie bieten hohe Leistung und viel Kontrolle über Hardware-Ressourcen.

4

Höhere Sprachen (z.B. Java, Python, JavaScript)

Diese Sprachen sind menschenlesbarer und abstrahieren viele technische Details der Hardware weg. Sie sind oft leichter zu lernen und schneller zu entwickeln, da sie viele komplexe Operationen vereinfachen.

Je tiefer das Level, desto näher am Prozessor und desto mehr Kontrolle hat man über die Hardware – aber auch desto aufwendiger und fehleranfälliger ist das Schreiben des Codes. Höhere Programmiersprachen erleichtern uns die Umsetzung von Ideen erheblich, da sie viele technische Details für uns übernehmen und uns erlauben, uns auf die Problemlösung zu konzentrieren.

Skript vs. Kompiliert

Merkmal	Skriptsprache	Kompilierte Sprache
Ausführung	Direkt (Interpreter)	Vorab übersetzt (Compiler)
Fehlerprüfung	Während der Laufzeit	Vor dem Start
Flexibilität	Hoch	Niedriger
Geschwindigkeit	Geringer	Oft deutlich höher
Beispiele	JavaScript, Python	C++, Java, Go

Beide Modelle haben ihre Vorteile: Skriptsprachen sind ideal zum schnellen Ausprobieren – kompilierte Sprachen verlangen mehr Disziplin, liefern aber oft schnellere Ergebnisse. In diesem Kurs nutzen wir Python – eine Skriptsprache mit hohem Lernwert.

Eine Idee, viele Sprachen

Beispiel: "Zähle von 1 bis 3"

Python

```
for i in range(1, 4):    print(i)
```

JavaScript

```
for (let i = 1; i <= 3; i++) {
console.log(i);}
```

C++

```
for (int i = 1; i <= 3; i++) {    cout << i;}
```

Drei Sprachen, gleiche Idee. Wenn ihr das Denken hinter der Schleife verstanden habt, ist es nur noch eine Frage der "Vokabeln", sie in einer Sprache umzusetzen.

Beispiel: "Addiere zwei Zahlen"

Python

```
def add(a, b):    return a + bprint(add(5, 3))
```

JavaScript

```
function add(a, b) {    return a +
b;}console.log(add(5, 3));
```

C++

```
int add(int a, int b) {    return a + b;}//
Usage would be in main function:// cout <<
add(5, 3);
```

Im nächsten Kapitel lernen wir erste praktische Strukturen kennen, um einfache Programme aufzubauen. Doch denkt immer daran: Der Code ist nur die Oberfläche. Darunter liegt die Logik.