

# Programmiersprachen – Überblick

*Grundprinzipien verstehen und sicher anwenden*

# Lernziele dieser Einheit

Grundprinzipien

*Übersetzen, Ausführen, Strukturierung*

OOP-Konzepte

*Objektorientierte Programmierung verstehen*

Tooling & Praxis

*Compiler, Debugger, Bibliotheken*



# Praxis in API



Begriffe sicher erklären

*Fachterminologie präzise verwenden  
und definieren*



Beispiele erkennen

*Typische Anwendungsfälle und  
Muster identifizieren*



Fehlerquellen verstehen

*Häufige Probleme ohne Codezwang  
analysieren*

QUIZ

# Verständnisfragen – Teil 1

Fehlerklassifikation

*Woran erkennst du:  
Syntax/Compiler oder  
Logik/Runtime?*

Sprachbausteine

*Nenne 5 typische Bausteine:  
Datentypen,  
Kontrollstrukturen...*

Strukturierte  
Programmierung

*Kernbedeutung und  
Prüfungsrelevanz*

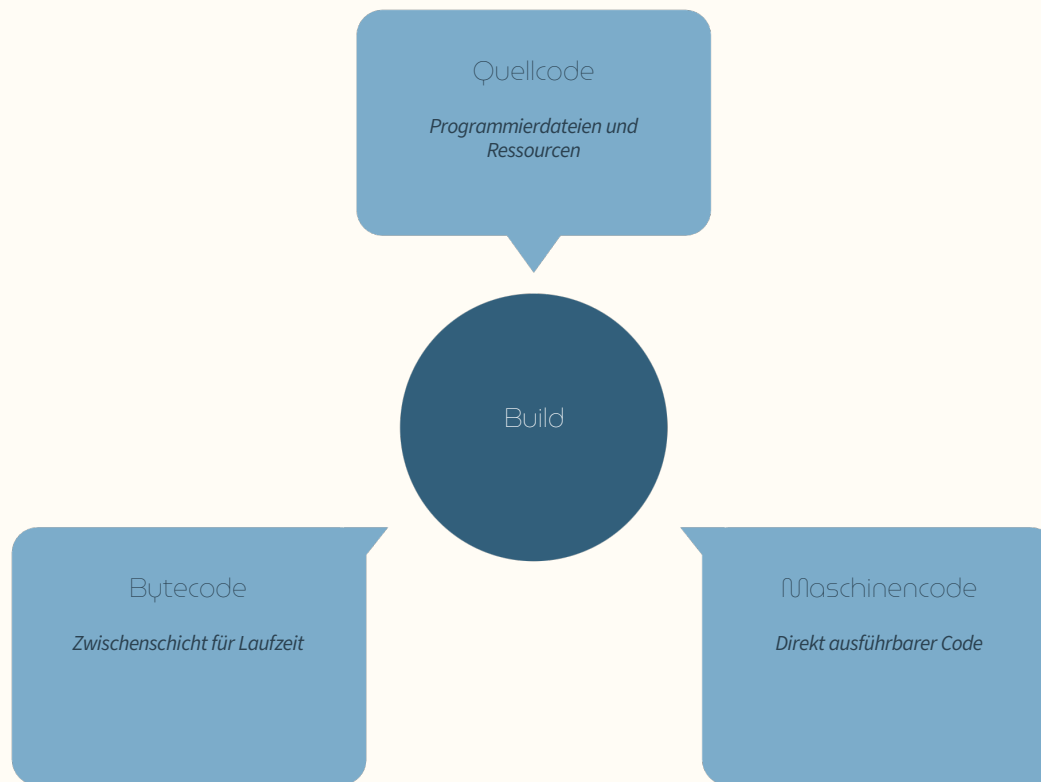
Debugging-  
Strategie

*Welche Infos vor dem  
„Rumprobieren“ sammeln?*

# Compiler, Linker, Interpreter

*Drei zentrale Werkzeuge der Codeausführung*

# Compiler



*Übersetzt Quellcode vorab in ausführbares Format*

## Typische Fehler

- *Syntax-Fehler: ungültige Sprachkonstrukte*
- *Typ-Fehler: inkompatible Datentypen*
- *Fehlende Deklarationen*

☐ *Compile-Time-Fehler werden vor der Ausführung erkannt*

# Linker

## Aufgabe

*Verbindet Übersetzungsartefakte und Bibliotheken zu ausführbarem Programm*

## Typische Fehler

- *Fehlende Symbole oder Bibliotheken*
- *Versionskonflikte*
- *Falsche Architektur (32bit/64bit)*
- *Pfadprobleme bei Dependencies*

# Interpreter

## Funktionsweise

*Führt Code direkt zur Laufzeit aus – schneller Start für  
Entwicklung*

## Runtime-Fehler

*Datei fehlt, Null/None-Zugriff, Index außerhalb, Division durch 0*

QUIZ

## Verständnisfragen – Teil 2

Fehlerszenarien

*Je 1 Beispiel: Compiler-, Linker-, Runtime-Fehler*

Paradox

*Warum kompiliert ein Programm, stürzt aber beim Start ab?*

Linker vs. Code-Bug

*Woran erkennst du Dependency-Probleme statt Fehler im Code?*

Bibliotheken

*Rolle bei Linker-Problemen: Version, Path, Architektur*

# Prozedural vs. Objektorientiert

*Zwei Denkweisen der Programmierung*

# Prozedural

## Kernprinzip

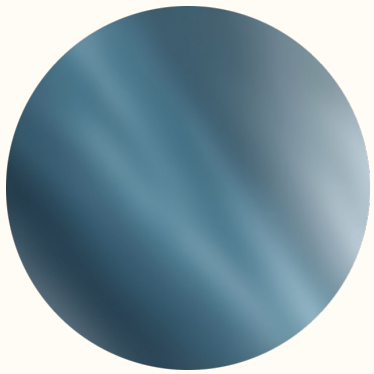
*Ablauf und Funktionen stehen im Mittelpunkt – Daten werden „bearbeitet“*

## Ideal für

- *Klaren Prozessfluss*
- *Kleinere Aufgaben*
- *Skripte und Automatisierung*
- *Mathematische Berechnungen*

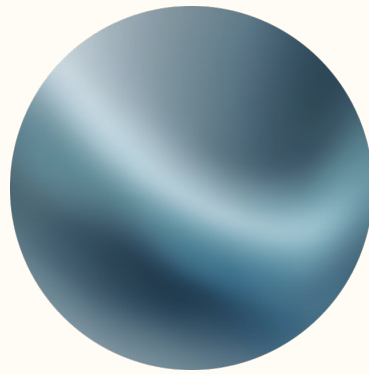


# Objektorientiert (OOP)



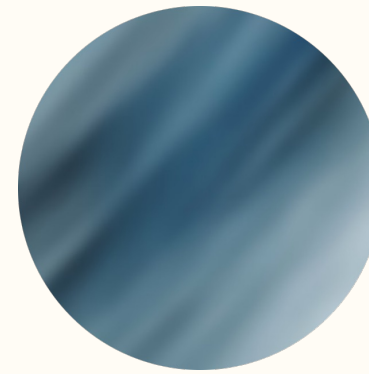
Objekte

*Zustand (Daten) + Verhalten (Methoden)  
kombiniert*



Realweltmodelle

*Kunde, Auftrag, Gerät, Ticket als Objekte*



Große Systeme

*Struktur, Wiederverwendung, Wartbarkeit*

QUIZ

## Verständnisfragen – Teil 3

1

Problemauswahl

*Ein Problem prozedural  
einfacher – eins mit OOP  
besser strukturiert?*

2

Funktion vs.  
Methode

*Was ist der zentrale  
Unterschied?*

3

Overengineering

*Woran erkennst du „zu viel  
OOP“ in kleinem Projekt?*

4

Einfache Erklärung

*Wie erklärst du einem Azubi  
„Objekt“ ohne  
Fachchinesisch?*

# Variablen & Datentypen

Variable

*Benannter Speicherplatz für Werte*

Typische Fehler

- *Typkonflikte bei Operationen*
- *Falsche Annahmen über „leer“*
- *null/None/"/0 verwechselt*
- *Index außerhalb der Grenzen*

# Datentypen im Detail

Integer vs. Float

*Ganzzahl vs. Kommazahl*



*Rundungsprobleme bei Float!*

String

*Text mit  
Encoding/Zeichensatz*

Boolean

*true/false – Quelle von  
Logikfehlern*

Datum/Zeit

*Format, Zeitzone, Parsing als  
Klassiker*

## QUIZ

# Verständnisfragen – Teil 4

## Datentyp-Bugs

*3 typische Bugs durch falschen Datentyp – wie erkennst du sie?*

## Datenstruktur-Wahl

*Wann Liste/Array, Map/Dictionary, Set – mit Begründung?*

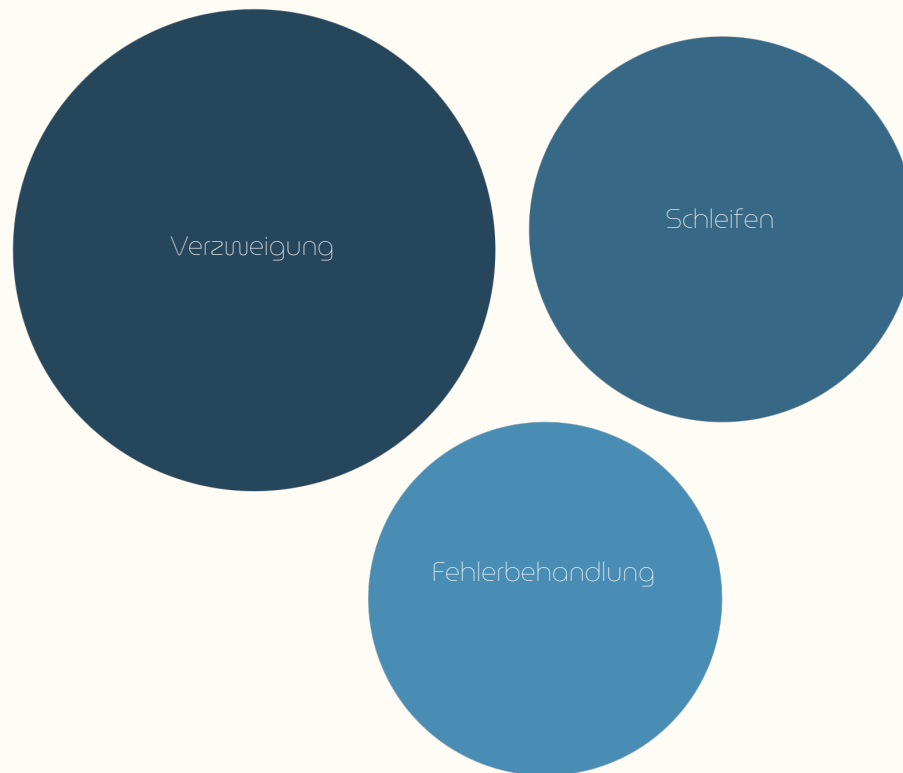
## Leer vs. Nicht-Existent

*Unterschied zwischen „Wert ist leer“ und „existiert nicht“*

## Fehlerabfangen

*Welche Checks gegen Index-/Key-Fehler ohne konkrete Code-Lösung?*

# Kontrollstrukturen



*Steuerung des Programmflusses*

## Verzweigung

*if/else/switch – verschiedene Wege je nach Bedingung*

## Schleifen

*for/while – Wiederholung bis Ende oder Abbruch*

## Häufige Fehler

- *Off-by-one (einmal zu oft/wenig)*
- *Endlosschleife*
- *AND/OR-Logikfehler*

# OOP-Bausteine

1

Klasse

*Bauplan für Objekte mit Attributen +  
Methoden*

2

Methode

*Funktion, die mit Objektzustand arbeitet*

3

Vererbung

*Basisklasse → Spezialisierung in Kindklasse*



*Vorsicht: Vererbung um der Vererbung willen vermeiden – oft ist Komposition besser*

# Bibliotheken vs. Frameworks

## Bibliothek

***Du rufst sie auf*** – du kontrollierst den Ablauf

*Beispiel: JSON-Parser, Datums-Library*

## Framework

***Ruft dich auf*** – Inversion of Control

*Du füllst Bausteine/Callbacks*



# Skriptsprachen & Debugging

## Skriptsprachen

- *Schnell schreiben und ändern*
- *Ideal für Automatisierung*
- *Deploy, Admin, kleine ETL*



*Risiko: Wildwuchs ohne Standards, fehlende Tests*

## Debugging-Strategie

1. *Fehler reproduzieren*
2. *Logs/Fehlermeldung sichern*
3. *Kleinsten Testfall bauen*
4. *Hypothesen einzeln prüfen*
5. *Fix + Regressionstest*