C vs. Python – Kurz-Cheat-Sheet

1. Ausführung: Compiler vs. Interpreter

C (kompilierte Sprache)

- Verwendung eines **Compilers** (z. B. `gcc`).
- Typische Schritte:
  1. Präprozessor (`#include`, Makros, Kommentare entfernen)
  2. Übersetzung in Assembler-Code
  3. Assemblierung zu Objektdateien
  4. Linken zu einem ausführbaren Programm (Maschinencode)
- Eigenschaften:
  - Übersetzung *vor* der Ausführung
  - Hohe Performance, ressourcennah
  - Plattformabhängig (bei Wechsel: neu kompilieren)

Python (interpretierte Sprache)

- Verwendung eines **Interpreters** / Python Virtual Machine (PVM).
- Typische Schritte:
  1. Einlesen des Quellcodes
  2. Lexing und Parsing (Syntaxanalyse)
  3. Übersetzung in Bytecode
  4. Ausführung des Bytecodes durch die PVM
- Eigenschaften:
  - Übersetzung *während* der Ausführung
  - Hohe Portabilität (gleicher Code auf vielen Systemen)
  - Geringere Ausführungsgeschwindigkeit als nativer C-Code

Übersicht: Ausführung

Aspekt	C (Compiler)	Python (Interpreter)
Zeitpunkt Übersetzung	vor Ausführung	während Ausführung
Ausgabe	Maschinencode	Bytecode + PVM
Performance	hoch	niedriger
Portabilität	geringer	höher

2. Typisierung: statisch vs. dynamisch

C: statische Typisierung

- Variablen besitzen einen **festen Typ** bei der Deklaration.
- Typprüfung erfolgt **zur Kompilierzeit**.
- Falsche Typzuweisungen führen zu Compilerfehlern.
- Typumwandlung:
  - implizit (z. B. Promotion in größere Typen)
  - explizit durch Cast (z. B. `(int) ausdruck`)
- Typische Folgen:
  - gute Fehlersuche beim Kompilieren
  - klare Dokumentation der Datentypen
  - bessere Performance, mehr Sicherheit

Python: dynamische Typisierung

- Variablen besitzen keinen fest deklarierten Typ, sondern referenzieren Objekte mit einem Typ.
- Typprüfung erfolgt **zur Laufzeit**.
- Eine Variable kann nacheinander Objekte unterschiedlichen Typs halten.
- Typische Folgen:
  - sehr flexible Verwendung
  - knapper, gut lesbarer Code
  - Typfehler treten erst bei der Ausführung auf

Übersicht: Typisierung

Aspekt	C (statisch)	Python (dynamisch)
Zeitpunkt Typprüfung	Kompilierzeit	Laufzeit
Typbindung	bei Deklaration	beim Zuweisen von Werten
Flexibilität	geringer, dafür klar	hoch, aber weniger strikt
Fehlererkennung	früh (beim Kompilieren)	spät (zur Laufzeit)
Performance	tendenziell höher	tendenziell niedriger