Agenda

1. FAQ zur Programmierung
2. Übersicht der Themen
3. Grundlagen Softwaretheorie
4. **Grundlagen Programmiersprachen**
5. Software-Entwicklungsprozesse
6. Software-Entwurftools

> Danach: Praktisches Programmieren (Python)

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Fallunterscheidung**

Syntax in C# und C++ und JS und PHP:

```
if (condition)
{
  // block of code to be executed if the condition is True
}
else
{
  // block of code to be executed if the condition is False
}
```

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Fallunterscheidung**

  Syntax in Python:

```
if condition:
    # Block of code to be executed if the condition is True
else:
    # Block of code to be executed if the condition is False
```

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Fallunterscheidung**
- Manche Programmiersprachen erlauben per else-if (oder elif) eine Verkettung von Fallunterscheidungen
- Manche Programmiersprachen erlauben eine gekürzte Form
  - Dazu später mehr!
  - Wen es jetzt interessiert: Googlen nach „ternary operator“

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Allgemeine Datentypen** (auch primitive Datentypen genannt)
- Integer: Ganzzahlen
  - In vielen Sprachen beschränkter Wertebereich! (nicht in Python)
  - Üblicher Wertebereich: -32 768 bis 32 767 (16 bits)
  - Manchmal auch Variationen:
    - unsigned int: 0 bis 65 535 (16 bits)
    - (unsigned) long int: -2 147 483 648 bis 2 147 483 647 bzw. 0 bis 4 294 967 295 (32 bits)
    - (unsigned) short int: -128 bis 127 bzw. 0 bis 255 (8 bits)

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Allgemeine Datentypen** (auch primitive Datentypen genannt)
- Boolean: Wahrheitswert
  - Kann entweder Wahr (True/1) oder Falsch (False/0) sein
  - Logische Werte
  - Jede Aussage kann ein Wahrheitswert zugewiesen werden:
    - Draußen scheint die Sonne. => Hoffentlich True :)
    - Jeder Mensch hat grüne Haare. => False, ich hab keine
    - 5 ist größer als 1. => True

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Allgemeine Datentypen** (auch primitive Datentypen genannt)
- Char und String: Zeichen und Zeichenketten („Texte“)
- Char (auch Character genannt) ist ein einzelnes Zeichen:
  - Z.B. "A" oder "5"
- Strings: Mehrere Chars nebeneinander
  - Z.B. "Hallo" oder "Das, ist ein Test"
- Je nach Codierung (ASCII, Unicode etc.) auch Umlaute erlaubt

4. Grundlagen Programmiersprachen

- Wiederkehrende Elemente/Konzepte in verschiedenen Programmiersprachen:
- **Funktionen**
- Einmal definieren, beliebig oft nutzen
- Bauen eigener „Werkzeuge“

4. Grundlagen Programmiersprachen

Programmierparadigmen

- 2 grundsätzliche Unterscheidungen:
  - Imperative Programmierung: Abfolge von Anweisungen, die der Computer abarbeitet (Das „Wie“)
  - Deklarative Programmierung: Beschreibung des gewünschten Endergebnisses (Das „Was“)
- Beide Kategorien bestehen aus Unterkategorien
- Dann gibt’s auch noch OOP…
- Die meisten Sprachen vereinen mehrere Paradigmen

4. Grundlagen Programmiersprachen

Programmierparadigma: Imperativ

- Strukturierte Programmierung:
  - Anstatt Sprunganweisungen (GOTO) sollen Kontrollstrukturen (if-else) oder Schleifen (while/for) verwendet werden
- Prozedurale Programmierung:
  - Programme in kleine Teilprogramme (Prozeduren, Funktionen) aufteilen
- Modulare Programmierung:
  - Weiterentwicklung von Prozeduraler Programmierung
  - Programm in verschiedene Module (z.B. Dateien, Libraries) aufteilen
- Beispiele: Python, C#, C++, Java und viele weitere

4. Grundlagen Programmiersprachen

Programmierparadigma: Deklarativ

- Abfragensprachen:
  - Code besteht aus Beschreibungen des Ergebnisses
  - Das eigentliche Durchsuchen läuft im Hintergrund
- Logische Programmierung:
  - Code besteht aus einer Menge von Regeln & Fakten
  - Interpreter/Compiler schlussfolgert aus diesen Regeln Antworten
- Funktionale Programmierung:
  - Hat prinzipiell nichts mit Funktionen (Prozedurale Programmierung) zu tun!
  - Jedes Programm ist eine Funktion, die zu einer beliebigen Eingabe die dazugehörige Ausgabe liefert (gemäß Aufgabenstellung)
  - Orientiert sich an mathematischen Formeln
  - Meine Empfehlung für Einsteiger: Großen Bogen drumherum machen!

4. Grundlagen Programmiersprachen

Programmierparadigma: Deklarativ

- Beispiele: SQL, Haskell, Prolog

4. Grundlagen Programmiersprachen

- Was ist mit OOP?
- Je nach Quelle: Zuordnung zu imperativ oder eigene Kategorie
- Klassendefinitionen können als Prozeduren und/oder Module verstanden werden

4. Grundlagen Programmiersprachen

Libraries/Frameworks

- Szenario:
  „Ich habe eine App-Idee mit der ich Multimillionär werde!“

- Weg 1: „Ich schreibe alles selbst, weil nur dann kann ich sicher sein, dass ich auch das kriege, was ich will!“
- Weg 2: „Ich nutze bestehendes und bewährtes Wissen, weil mir das Zeit und Mühe spart!“
- Welchen Weg wählt ihr?

4. Grundlagen Programmiersprachen

Libraries/Frameworks

- Definition: Programmcode, der vorgefertigte Funktionalitäten enthält, welche sich eignen um ein bestimmtes Problem oder eine Klasse von Problemen zu lösen
- Werden meist von mehreren Personen entwickelt und gepflegt
- Unterschied Libraries & Frameworks:
  - „Inversion of Control“
  - Library: Funktionalitäten, welche eigenen Code erweitern
  - Framework: Funktionalitäten, welche mit eigenem Code erweitert werden
- Frameworks bringen meist eigene Libraries mit

4. Grundlagen Programmiersprachen

Libraries/Frameworks

Vorteile

- Erleichtern uns Entwicklern das Leben
- Stellen viele verschiedene Funktionalitäten zur Verfügung
- Sind üblicherweise getestet & werden stetig verbessert
- Können angepasst werden

Nachteile

- Können sehr komplex und für Einsteiger überfordernd sein
- Bringen meist mehr mit als man im Endeffekt braucht (Speicherplatzprobleme)

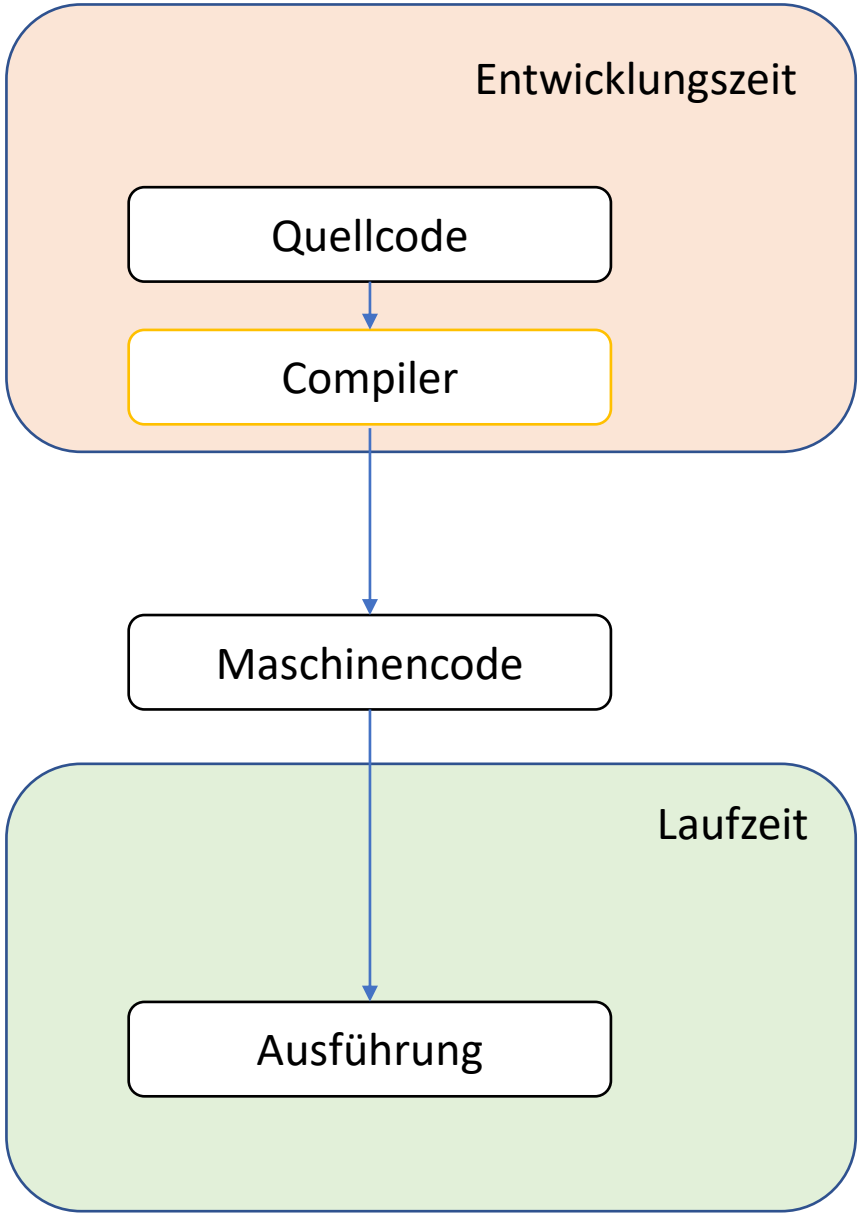
4. Grundlagen Programmiersprachen

Programmierwerkzeuge

- Editor: Programm, welches erlaubt u.a. Quellcode zu schreiben
  - Beispiele: Windows Editor, Notepad++
- IDE: (Integrated Development Environment) Editor, der auch Code ausführen kann
  - Enthält auch üblicherweise eine Reihe von nützlichen Funktionen: Compiler, Interpreter, Debugger, Version Control
  - Beispiele: VS Code, pyCharm, Sublime Text, (Eclipse)

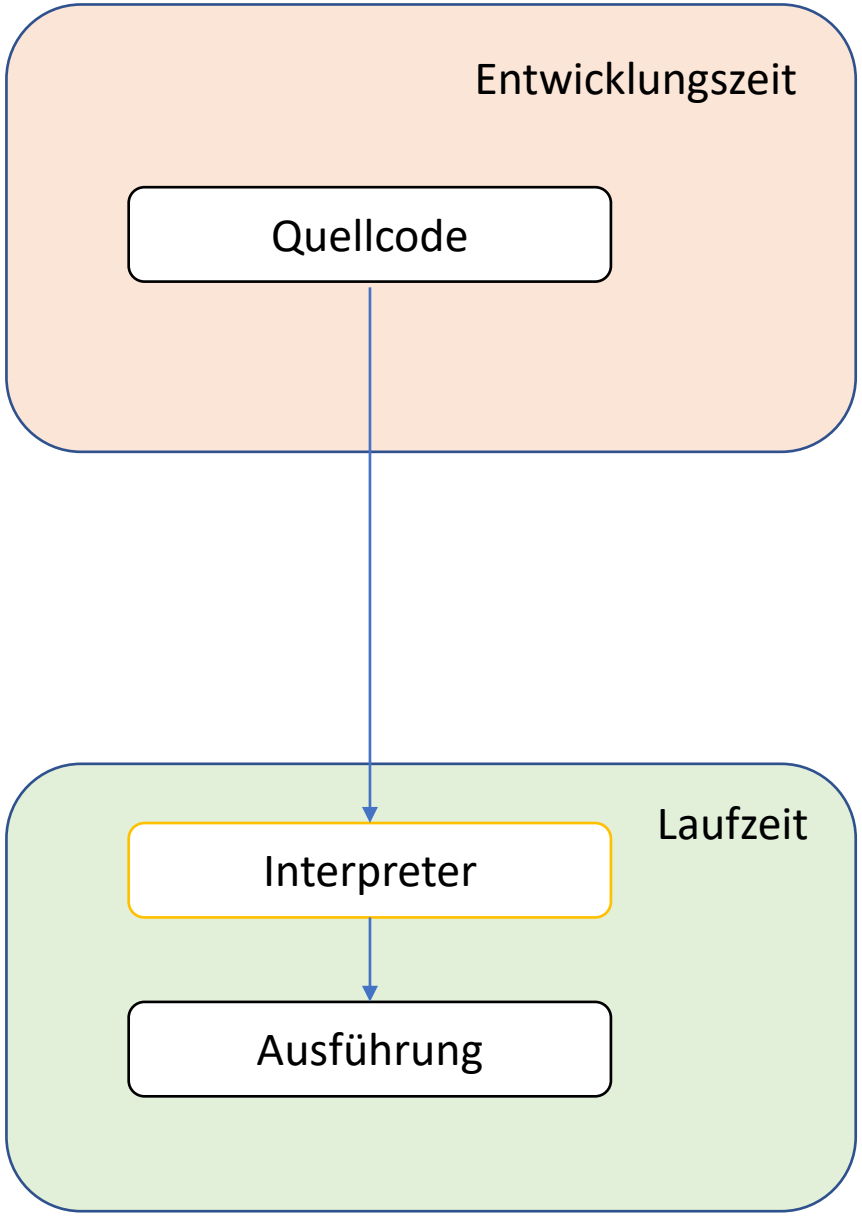
Compiler

- Übersetzt den gesamten Quellcode **vor** Ausführung in Maschinensprache
- Nach Übersetzung folgt die (sehr effiziente) Ausführung
- Der Compiler legt Reihenfolge der Anweisungen fest



Interpreter

- Verarbeitet den Quellcode zur Laufzeit
- Zeile für Zeile:
  - Einlesen
  - Übersetzen
  - Ausführen
- Stopp, wenn Ende des Quellcodes erreicht oder Fehler
  - Fehlerquellen leicht zu finden
- Erzeugt **keine** ausführbare Datei



Vergleich

	Interpreter	Compiler
Wann erfolgt die Quellcode-Übersetzung?	Beim Ausführen des Codes	Vor Ausführung des Codes
Wie erfolgt die Übersetzung?	Zeile für Zeile	Gesamter Code
Wann wird der Entwickler über Fehler informiert?	Direkt nach Ausführung der fehlerhaften Zeile	Nach der kompletten Compilierung
Ausführungseffizienz (im Vergleich)	Gering	Hoch
Entwicklungsaufwand (im Vergleich)	Gering	Hoch
Sprachen	Python, Perl, Ruby etc.	C, C++

4. Grundlagen Programmiersprachen

- Debugger: Hilfestellung zur Fehlersuche im Quellcode
  - Erlaubt, das Programm zur Laufzeit Schritt für Schritt auszuführen
  - Nach jedem Schritt des Programms lassen sich die Werte aller Variablen einsehen
  - Erlauben das Setzen von Haltepunkten (Breakpoints)
- Version Control: Versionierung des Codes
  - Software-Entwicklung dauert seine Zeit
  - Bestimmte „Stufen“ des Software-Projekts setzen (z.B. erfolgreiche Implementierung einer Funktion)
  - Es lässt sich zu vorherigen „Stufen“ zurückkehren
  - SEHR nützlich bei Teamarbeit
  - Beispiele: GIT, SVN
  - Ist ein riesiges Thema, wird in der Praxis besser vermittelt