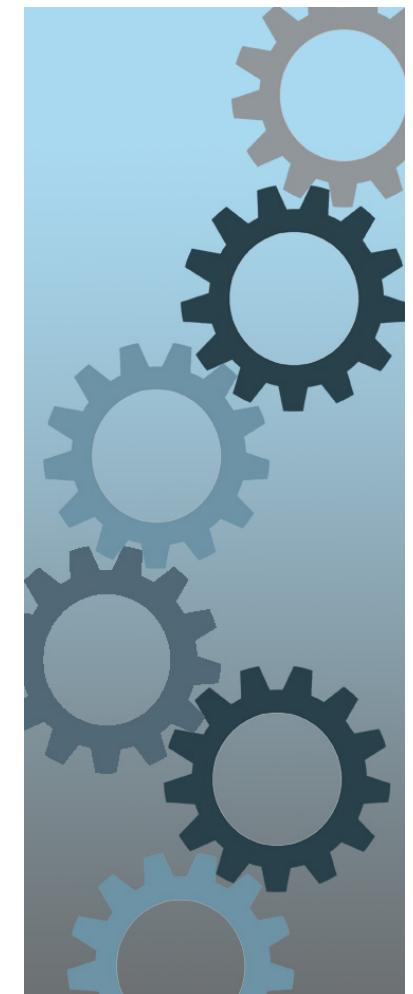


Programming Concepts & Paradigms

Einführung: Programmierparadigmen

Prof. Dr. Ruedi Arnold

ruedi.arnold@hslu.ch



Übersicht Einführung Programmierparadigmen

- Einleitung: Programmiersprachen
- Definition & Übersicht Programmierparadigmen
 - imperativ
 - strukturiert
 - prozedural
 - objektorientiert
 - deklarativ
 - logisch
 - funktional

Java: OO-Programmierung

- Informatikstudium HSLU, Fokus Programmierung: Java
 - Module (PRG), OOP, PLAB, AD,

→ Sie kennen also die Programmiersprache Java

- Kennen Syntax (Form) & Semantik (Bedeutung)

- z.B. Schlüsselwörter, Schleifen, Bedingungen, Deklaration, Methoden, Klassen, Vererbung, Instanziierung, usw.

- Sie können also z.B.
das Java-Code-
Beispiel rechts
lesen und verstehen

```
public class Ackermann extends BaseProblem {  
    private long counter = 0;
```

```
    public long ack(long n, long m) {  
        counter++;  
        System.out.println("ack(" + n + ", " + m + ")");  
        if (n == 0) {  
            for (int i=0; i<n; i++) {
```

...andere Sprachen?

- Zwei Fragen zum Einstieg:
 - Was kennen sie für Programmiersprachen?
 - Inwiefern unterscheiden sich diese Sprachen von Java?



4 http://www.wired.com/images_blogs/threatlevel/2012/03/unclesam.jpeg

Auswahl populärer Sprachen

Auswahl populärer Programmiersprachen, alphabetisch geordnet (ohne Anspruch auf Vollständigkeit!):

- | | | | |
|-------------|--------------|---------------|-------------|
| – Ada | – Delphi | – ML | – Python |
| – Algol | – Eiffel | – Modula-2 | – Ruby |
| – Assembler | – Fortran | – Oberon | – Scala |
| – Bash | – Haskell | – Objective-C | – Scheme |
| – Basic | – Java | – Pascal | – Smalltalk |
| – C | – JavaScript | – Perl | – Swift |
| – C# | – Kotlin | – PL/SQL | |
| – C++ | – Lisp | – PHP | |
| – Cobol | – Lua | – Prolog | |

Programmiersprachen: Wie Sand am Meer

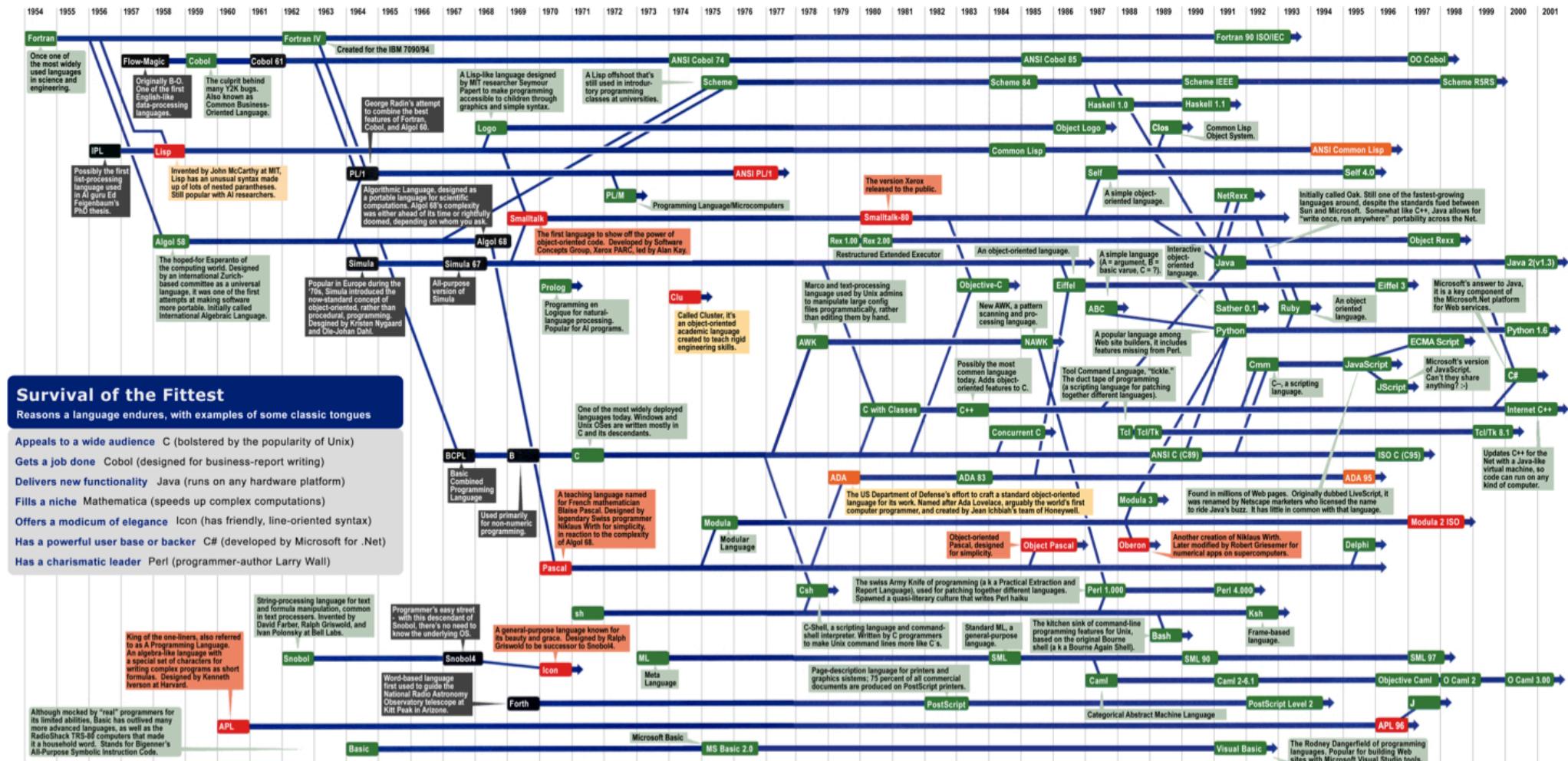
- Es gibt tausende von Programmiersprachen!
 - Wikipedia listet ca. 700 „namhafte“ Sprachen
 - https://en.wikipedia.org/wiki/List_of_programming_languages
 - Es gibt Tausende von weiteren (spezialisierten) Sprachen
- Zusammenstellungen von Programmiersprachen
 - „100 Bottles of Beer“-Programme in 1500 verschiedenen Programmiersprachen und –variationen:
<http://www.99-bottles-of-beer.net/j.html>
 - „Hello World“-Programme in 441 verschiedenen Sprachen:
<http://www.roesler-ac.de/wolfram/hello.htm>

```
10 REM Hello World in BASIC
20 PRINT "Hello World!"
```

„Familienbaum“ wichtiger Programmiersprachen

Mother Tongues

Tracing the roots of computer languages through the ages



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Was ist eine Programmiersprache?

Programmiersprache = formal konstruierte Sprache, entworfen um Befehle an Maschinen (speziell Computer) zu übermitteln

- Programmiersprachen werden eingesetzt, um Programme zu schreiben, welche das Verhalten von Maschinen kontrollieren oder welche Algorithmen beschreiben
 - Algorithmus = schrittweises Verfahren zur Lösung von einem Problem (oder einer Klasse von Problemen)
 - siehe AD-Modul...

Zwei wichtige Begriffe: Syntax & Semantik

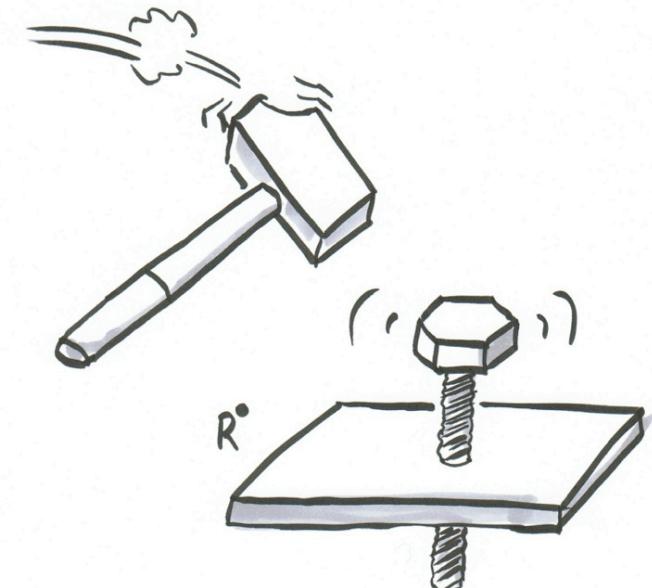
- **Syntax:** Gibt das Muster (die formale Struktur) vor, nach dem Programme einer Sprache aufgebaut sind
 - „Zusammenfügungsregeln“
 - Mathe-Bsp.: $3 * 5$ aber $4 ** 6$ **X**
- **Semantik:** Definiert die Bedeutung von Programmen (Anweisungen, Operatoren, usw.)
 - „Interpretationsregeln“
 - Mathe-Bsp.: $3 * 5$ gibt 15 oder $3 ! = 6$

Mächtigkeit: Turing-Vollständigkeit

- Alle auf den bisherigen Folien aufgelisteten Programmiersprachen sind gleich mächtig, d.h. in diesen können dieselben Algorithmen festgehalten werden
- In der Berechenbarkeitstheorie spricht man von **Turing-Vollständigkeit** (= universelle Programmierbarkeit, d.h. damit lassen sich grundsätzlich alle Funktionen berechnen, welche mit einer [fiktiven] Turing-Maschine berechnet werden können)
 - All diese Sprachen sind also Turing-vollständig
- Turing-Vollständigkeit ist eine notwendige Bedingung für eine Programmiersprache
 - Gemäss dieser Definition ist z.B. HTML keine Programmiersprache
 - Hinweis: SQL Turing-vollständig seit SQL 1999 (rekursive Common-Table-Expressions erlauben allgemeine Rekursion), davor war SQL nicht Turing-vollständig (<http://stackoverflow.com/questions/900055/is-sql-or-even-tsql-turing-complete>)

Kenntnisse & Auswahl von Programmiersprachen

- Praktisch alle Programmiersprachen sind also gleich mächtig!
- Jedoch: Programmiersprachen und –paradigmen haben individuelle Stärken und Schwächen
- Ziel: Sie kennen und verstehen verschiedene Paradigmen und Sprachen...
 - und verstehen unterschiedliche Denkweisen, Berechnungsmodelle und Programmierkonzepte ☺
 - und können so je nach Problem/Aufgabe geeignet auswählen ☺



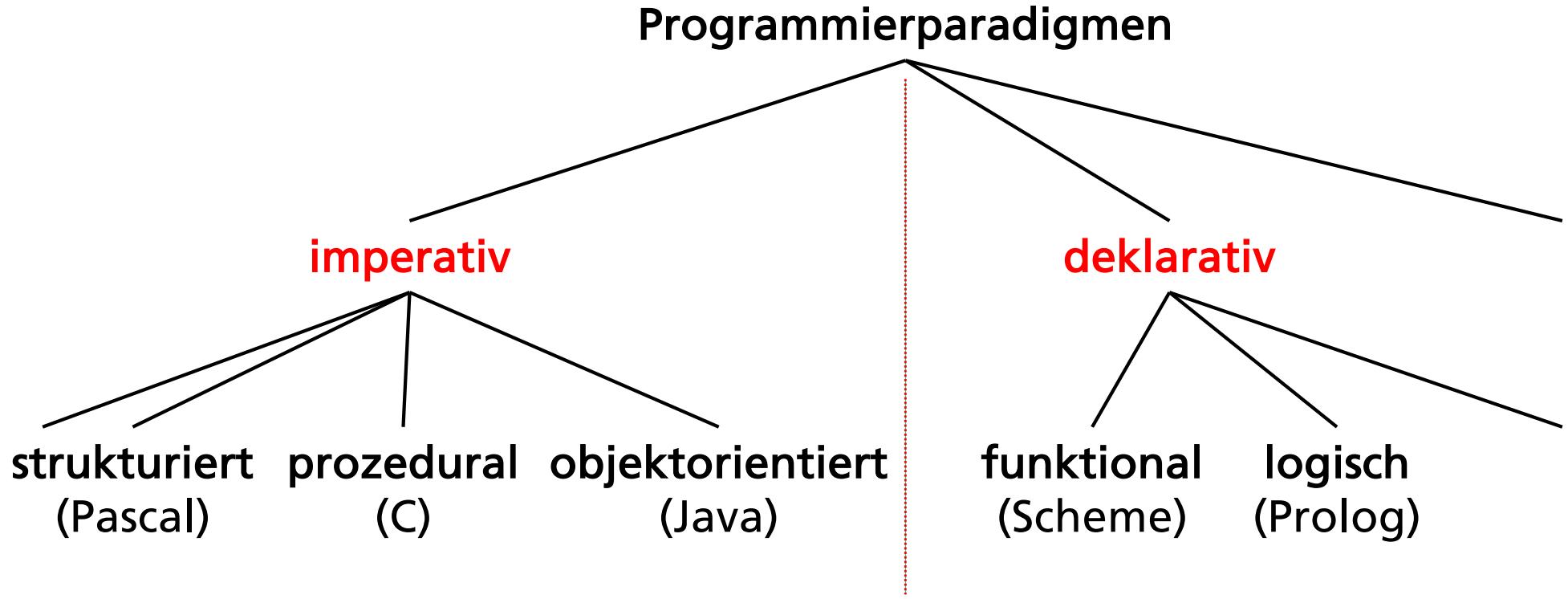
Definition Programmierparadigma

Programmierparadigma = fundamentaler Programmierstil, eine bestimmt Art die Struktur und Elemente von Programmen aufzubauen

Auszug von verschiedenen Definitionen von „Paradigma“:

- Beispiel, Muster (<http://www.duden.de/rechtschreibung/Paradigma>)
- Denkmuster, Schema, eine grundlegende wissenschaftliche Denkweise (<http://de.wiktionary.org/wiki/Paradigma>)
- grundsätzliche Denkweise. [...] eine bestimmte Art der Weltanschauung oder eine Lehrmeinung (<https://de.wikipedia.org/wiki/Paradigma>)

Programmierparadigmen & typische Sprache dazu



→ Fundamental ist die Unterscheidung in imperative und deklarative Programmiersprachen

Weitere Programmierparadigmen

- Neben den eben genannten gibt es weitere Paradigmen, z.B.: Aspekt-orientiert, Automaten-basiert, Constraint-orientiert, Daten-getrieben, Datenfluss-orientiert, Ereignis-orientiert, generisch, generativ, Komponenten-orientiert, nebenläufig, modular, protokoll-orientiert, reaktiv, reflektiv, ...
 - siehe z.B.: https://en.wikipedia.org/wiki/Programming_paradigm
 - Viele dieser Paradigmen lassen sich in verschiedenen Sprachen umsetzen bzw. integrieren...
 - Diese Paradigmen sind daher typischerweise nicht prägend für eine bestimmte Programmiersprache
- Wir konzentrieren uns hier auf die beiden Programmierparadigmen **imperativ** (strukturiert, prozedural, objektorientiert) und **deklarativ** (logisch und funktional)

Hauptparadigma einer Programmiersprache

- Die meisten Programmiersprachen haben ein Hauptparadigma, das sie befolgen, z.B.:
 - Java: objektorientiert
 - Prolog: logisch
 - Scheme: funktional
- Unterscheidung ist nicht messerscharf
 - Objektorientierte Sprachen: meist auch strukturiert und prozedural
 - Die meisten deklarativen Sprachen haben auch imperative Eigenschaften (siehe später)
 - Bei Prolog z.B. zu sehen bei Cut, Assertions oder Endrekursion
 - Bei Scheme z.B. zu sehen bei if- oder let*-Sonderformen

Sprachen & Paradigmen

- Sprachen müssen nicht exklusiv einem Programmierparadigma angehören
- Programmierparadigmen sind eher komplementär (d.h. sich ergänzend) als sich gegenseitig ausschliessend (d.h. entweder-oder) zu sehen
- Viele Sprachen unterstützen mehrere Paradigmen
 - z.B. Java: bekannt als objektorientierte Sprache, das ist die Grund-Charakteristik. Java unterstützt aber z.B. auch prozedurale, modulare und ereignisorientierte Programmierung, seit Java 8 werden auch funktionale Ansätze unterstützt

Multiparadigmen-Programmiersprachen

- Viele „moderne“ Sprachen (z.B. Scala, Kotlin, Swift) unterstützen z.B. imperative und funktionale Programmierung
 - z.B.: Swift (Sprache für macOS und iOS, Screenshot von [https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language)))



Hauptparadigmen populärer Programmiersprachen

strukturiert
objektorientiert

funktional
logisch

- | | | | |
|-------------|--------------|---------------|-------------|
| – Ada | – Delphi | – ML | – Python |
| – Algol | – Eiffel | – Modula-2 | – Ruby |
| – Assembler | – Fortran | – Oberon | – Scala |
| – Bash | – Haskell | – Objective-C | – Scheme |
| – Basic | – Java | – Pascal | – Smalltalk |
| – C | – JavaScript | – Perl | – Swift |
| – C# | – Kotlin | – PL/SQL | |
| – C++ | – Lisp | – PHP | |
| – Cobol | – Lua | – Prolog | |

imperativ vs. deklarativ

- Imperative Programmierung: **WIE**
 - Problemlösung (Berechnungslogik) wird schrittweise in Befehlen beschrieben: Wie ist das Problem zu lösen?
 - Imperativ = unbedingte Anweisung
- Deklarative Programmierung: **WAS**
 - Berechnungslogik wird beschrieben ohne den Kontrollfluss zu definieren: Was sind die Fakten zum Problem?
 - Deklaration = Erklärung

imperativ vs. deklarativ: Beispiel „Kreis“

- Imperative Programmierung: **WIE**
 - Kreis: Resultat einer 360 Grad Rotation um einen festen Mittelpunkt mit dem Zirkel
 - Beschreibung von einer konkreten Lösung, so wird schrittweise vorgegangen!
- Deklarative Programmierung: **WAS**
 - Kreis: Menge aller Punkte, die von einem vorgegebenen Punkt denselben Abstand hat
 - Reine Problembeschreibung ohne konkreten Lösungsweg!

Code-Beispiel: Grösster gemeinsamer Teiler

- Aufgabe: Berechnung des grössten gemeinsamen Teilers (greatest common divisor: GCD) von zwei ganzen Zahlen gemäss dem euklidischen Algorithmus
 - Hinweis: Wir gehen davon aus, dass die beiden Zahlen positive Ganzzahlen sind (ohne dies zu testen)
- Wie sieht GCD-Code aus gemäss Paradigma:
 - imperativ-prozedural? (z.B. in Java oder C)
 - deklarativ-logisch? (z.B. in Prolog)
 - deklarativ-funktional? (z.B. in Scheme)

GCD imperativ/prozedural programmiert in C

```
1. int gcd(int a, int b) {  
2.     while (a != b) {  
3.         if (a > b) a = a - b;  
4.         else b = b - a;  
5.     }  
6.     return a;  
7. }
```

- Charakteristik: Fokus auf Befehlen (Anweisungen)
- Kontrollfluss ist vorgegeben (sequentiell bis auf Kontrollstrukturen wie z.B. Funktionen, Bedingungen oder Schleifen)
- Bemerkung: GCD-Code sieht in Java (fast) gleich aus

GCD deklarativ-logisch in Prolog

```
1. gcd(A, A, A) .  
2. gcd(A, B, G) :- A > B, C is A-B, gcd(C, B, G) .  
3. gcd(A, B, G) :- B > A, C is B-A, gcd(C, A, G) .
```

- Charakteristik: Einzelne Fakten und Regeln sind angegeben
- Kontrollfluss ist nicht (explizit) vorgegeben
- Aufruf in der Prolog-Konsole:

```
?- gcd(42, 35, X) .  
X = 7
```

GCD in Prolog: Interpretation

```
1. gcd(A, A, A) .  
2. gcd(A, B, G) :- A > B, C is A-B, gcd(C, B, G) .  
3. gcd(A, B, G) :- B > A, C is B-A, gcd(C, A, G) .
```

- Die Aussage $\text{gcd}(A, B, G)$ stimmt wenn...
 1. A, B und G alle gleich sind, oder
 2. A ist grösser als B und es gibt eine Zahl C für die gilt C ist $A-B$ und $\text{gcd}(C, B, G)$ stimmt, oder
 3. B ist grösser A und es gibt eine Zahl C für die gilt C ist $B-A$ und $\text{gcd}(C, A, G)$ stimmt
- Um den GCD von zwei Zahlen A und B zu finden, wird eine Zahl G (und verschiedene C's) gesucht, mit deren Hilfe sich $\text{gcd}(A, B, G)$ aus den gegebenen Fakten und Regeln beweisen lässt

→ Ausführliche Prolog-Einführung folgt in SW 2-4

GCD deklarativ-funktional in Scheme

```
1. (define gcd
2.   (lambda (a b)
3.     (cond ((= a b) a)
4.           ((> a b) (gcd (- a b) b))
5.           (else (gcd (- b a) a))))))
```

- Charakteristik: Fokus auf Definition von Ein- und Ausgaben von Funktionen
- Kontrollfluss ist nicht (explizit) vorgegeben
- Aufruf in der Scheme-Konsole:

```
> (gcd 42 35)
```

7

GCD in Scheme: Interpretation

```
1. (define gcd
2.   (lambda (a b)
3.     (cond ((= a b) a)
4.           ((> a b) (gcd (- a b) b))
5.           (else (gcd (- b a) a))))))
```

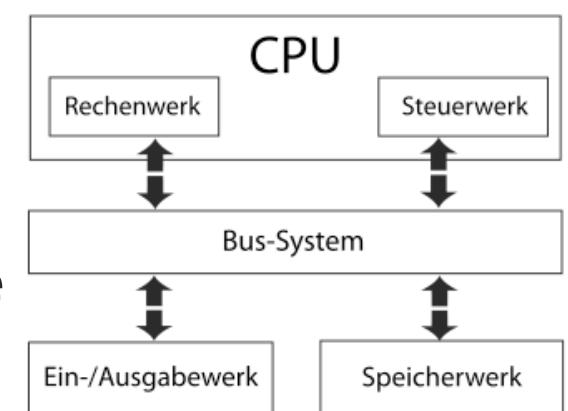
- Der GCD von a und b ist definiert als...
 3. a wenn a und b gleich sind
 4. GCD von a-b und b, wenn a > b
 5. GCD von b-a und a, wenn b > a
 - Um den GCD von zwei Zahlen a und b zu berechnen, wird die gegebene Definition expandiert und vereinfacht bis sie terminiert
- Ausführliche Scheme-Einführung folgt in SW 5-7

Imperative Programmierung: WIE lösen?

- lat.: imperare = befehlen, anordnen
- Problem-Lösungs-Algorithmus wird schrittweise durch Befehle angegeben: Der Quellcode gibt an, was der Computer in welcher Reihenfolge tun soll
- Zur Steuerung der Befehlsausführung stehen Kontrollstrukturen (Sequenz, Schleife [Iteration], Verzweigung [Selektion]) zur Verfügung
- So funktionieren die meisten bekannten und populären Programmiersprachen wie z.B. Java, C#, PHP, usw.
 - Insbesondere funktionieren Hardware-nahe Sprachen wie Assembler (oder C) praktisch immer so

Von Neumann Architektur & imperative Sprachen

- Imperatives Programmierparadigma ist eng angelehnt an die Ausführung von Maschinen-Code (Assembler) auf Computern, die nach der Von-Neumann-Architektur implementiert sind
 - Es existieren z.B. bedingte und unbedingte Sprunganweisungen
 - Zustand vom Programm ergibt sich aus Inhalt von Datenfeldern im Arbeitsspeicher und Systemvariablen (z.B. Register, Befehlszähler)
- Von Neumann-Architektur: wichtigstes Referenzmodell für Computer, realisiert alle Komponenten einer Turing-Maschine
 - Haupteigenschaft: gemeinsamer Speicher enthält sowohl Computerprogrammbefehle als auch Daten



<https://de.wikipedia.org/wiki/Von-Neumann-Architektur>

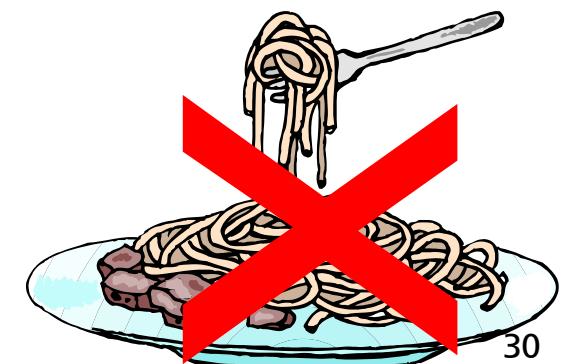
Bekannte imperative Programmiersprachen

Auswahl bekannter imperativer Programmiersprachen,
chronologisch nach Erscheinungsjahr:

- Assembler
- C++ (1983)
- Fortran (1954)
- Objective C (1983)
- Algol (1958)
- Eiffel (1986)
- Cobol (1960)
- Java (1995)
- Basic (1964)
- JavaScript (1995)
- Pascal (1970)
- C# (2000)
- C (1972)

Strukturierte Programmierung

- Spezialisierung des imperativen Paradigmas, verlangt Beschränkung auf drei Kontrollstrukturen:
 1. Sequenzen (Hintereinander auszuführende Programmanw.)
 2. Auswahl (Verzweigung: Bedingung)
 3. Wiederholung (Schleifen)
 - Konsequenz: „*goto*“ darf nicht eingesetzt werden, sonst „Spaghetticode“ (= Code mit verworrenen Kontrollstrukturen)
 - Artikel "Go To Statement Considered Harmful" (1968) von Edsger Dijkstra war Wendepunkt/Auslöser
 - Typische Sprachen: C, Fortran, Pascal



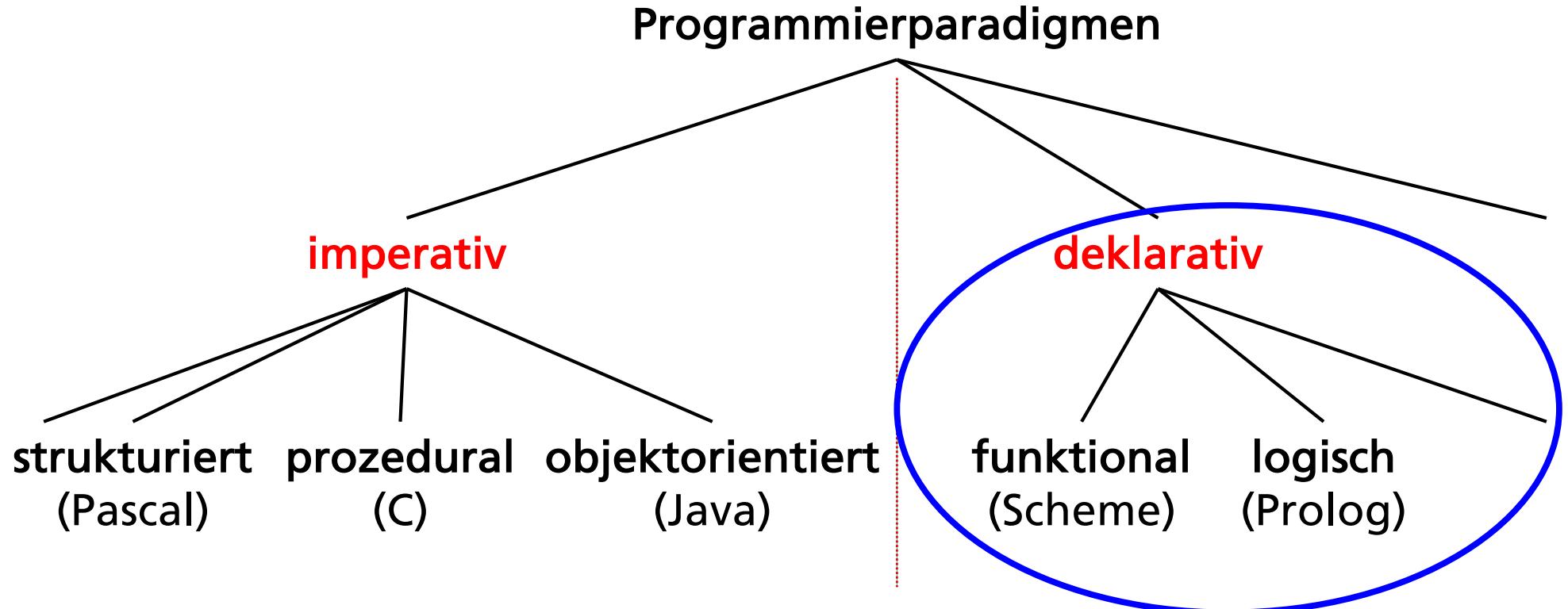
Prozedurale Programmierung

- Spezialisierung des imperativen Paradigmas, bietet Unterteilung von Programmen in Teilprogramme (= Prozedur, oder ja nach Sprache: Routine, Unterprogramm, Funktion)
 - Vermeidet Code-Duplikation
 - Prozeduren können Argumente entgegen nehmen und Ergebnisse zurück geben
 - Unterscheidung in Programm-globale und Prozedur-lokale Variablen
- Oft genannt als „Gegenstück“ zur objektorientierten Programmierung innerhalb der imperativen Sprachen
- Typische Sprachen: C, Fortran, Pascal

Objektorientierte Programmierung

- Spezialisierung des imperativen Paradigmas, laufende Programme bestehen aus einzelnen Objekten, welche miteinander interagieren (Nachrichten austauschen = Methoden aufrufen)
 - So wie Sie objektorientierte Programmierung an der HSLU T&A anhand von Java in PRG1, PRG2, usw. gelernt haben
- Objekte sind typischerweise Instanzen von Klassen (Geht auch anders, z.B. mit Prototypen in JavaScript)
 - Klassen definieren Zustand (Variablen) und Verhalten (Methoden)
 - Vererbung, Polymorphismus, usw.
- Entstanden als Weiterentwicklung von prozeduraler Programmierung und abstrakten Datentypen
- Typische Sprachen: Smalltalk, Objective C, C++, Java, C#

Programmierparadigmen & typische Sprache dazu



→ Fundamental ist die Unterscheidung in imperative und deklarative Programmiersprachen

Deklarative Programmierung: WAS ist gegeben?

- lat.: declaratio = „Erklärung, Kundmachung, Offenbarung“
- Das zu lösende Problem wird beschrieben, der Lösungsweg wird dann automatisch ermittelt: Das Programm enthält genügend Information, sodass das Problem gelöst werden kann
- Der Kontrollfluss von einem Programm ist nicht (explizit durch den Programmierer) geregelt
- Bekannteste Unter-Programmierparadigmen sind logische Programmierung (z.B. Prolog) und funktionale Programmierung (z.B. Scheme)
 - Weitere deklarative Programmierparadigmen sind z.B.: Constraint Programming (Siehe Modul „AI“), Abfragesprachen (z.B. SQL), oder Datenflusssprachen (z.B. Simulink)

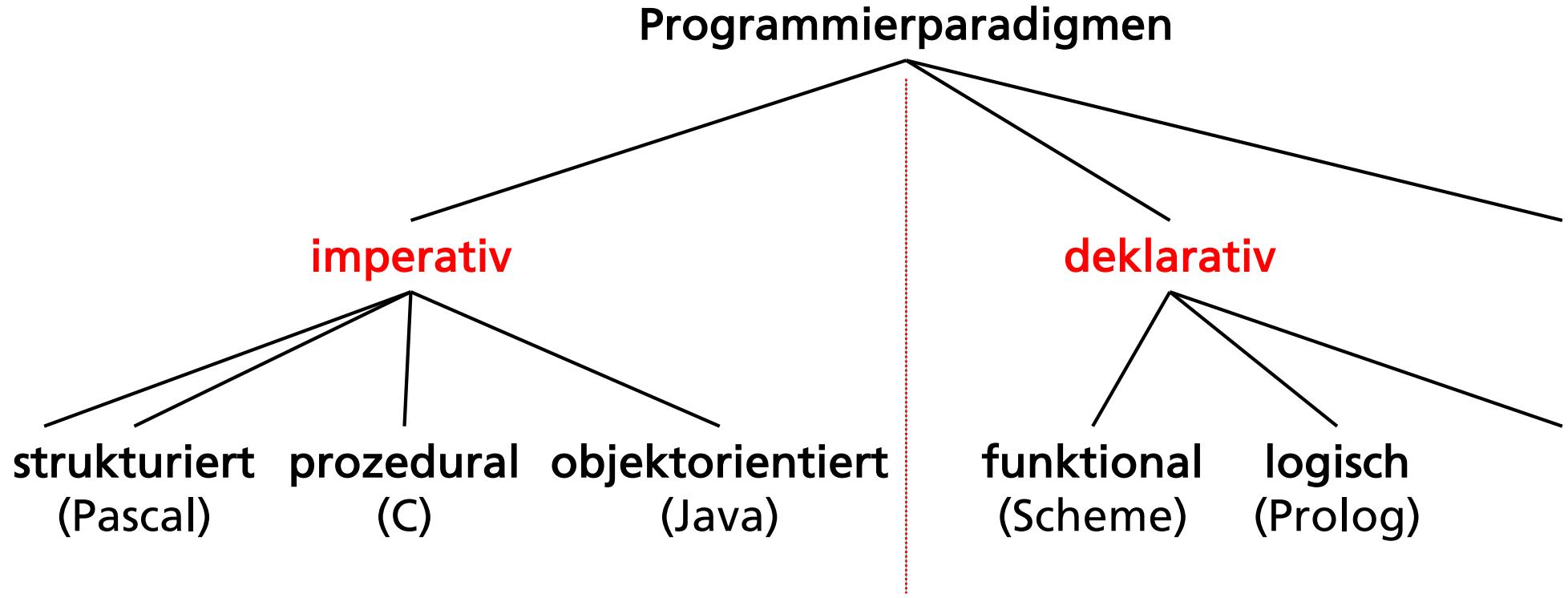
Logische Programmierung

- Spezialisierung vom deklarativen Paradigma:
Programm besteht aus Fakten und Regeln, aus welchen auf Anfrage automatisch versucht wird, eine Lösungsaussage herzuleiten
 - Lösungsweg wird nicht angegeben
- Basierend auf mathematischer Logik (Horn-Klauseln, Resolution)
- Bekannteste Sprache: Prolog (siehe SW2-4)
 - Weitere Sprache: Datalog

Funktionale Programmierung

- Spezialisierung des deklarativen Paradigmas:
Programme bestehen ausschliesslich aus
Definitionen von Funktionen mit Parametern und
Rückgabewerten
 - Der Rückgabewert hängt ausschliesslich von den
Parametern ab (-> Referenzielle Transparenz!)
 - Es gibt keinen veränderbaren (!) Zustand und keine
veränderbaren Daten
- Basiert auf dem formalen System des Lambda-Kalkül
- Bekannte Sprachen: Clojure, Erlang, Haskell, Lisp,
ML, Scheme (siehe SW5-7)

Nochmal: Paradigmen & typische Sprache dazu



→ Mit diesen sieben Programmierparadigmen werden wir uns in diesem Modul vertieft beschäftigen ☺