

Objektorientierte und Funktionale Programmierung

-- Organisation & Übungsbetrieb --

Kristof Van Laerhoven
kvl@eti.uni-siegen.de

Michael Möller
michael.moeller@uni-siegen.de

Andreas Hoffmann
andreas.hoffmann@uni-siegen.de

- Startet um 12:30 Uhr, geht bis ca. 14:00 Uhr, dann 15 min. Kaffeepause und weiter bis ca. 15:45 Uhr
- Wir nutzen Moodle: <https://moodle.uni-siegen.de/course/view.php?id=11376> um alle Inhalte und Folien zur Vorlesung zur Verfügung zu stellen
- Fragen Sie Fragen während der Vorlesung, trauen sie sich, unterbrechen Sie uns
- Für Fragen außerhalb der Vorlesung, nutzen Sie das Moodle Forum – so profitieren alle davon

- **Übungsleitung:** AR Dr.-Ing. Andreas Hoffmann,
Lehrstuhl Betriebssysteme / verteilte Systeme
andreas.hoffmann@uni-siegen.de
- **Tutoren:**
 - Tina Marx <Tina.Marx@student.uni-siegen.de>
 - Jan Morina <Jon.Morina@student.uni-siegen.de>
 - Lea Dielmann <lea3.dielmann@student.uni-siegen.de>
 - Murat Orbay <murat.orbay@student.uni-siegen.de>

Durchführung der Tutorübungen:

- Sie müssen sich in unisono zu einer Übungsgruppe anmelden damit ihnen ein Tutor zugewiesen werden kann
- **Wir setzen die Lehr-/Lernplattform GOALS ein. GOALS ist eine Eigenentwicklung im Rahmen eines Drittmittelprojekts**
- Die Tutorenübungen finden in Präsenz statt
- Die Präsenzübungen starten ab dem 20.04.

Voraussetzungen zur Studienleistung:

- Nur wer zur Studienleistung in unisono angemeldet ist, oder die Studienleistung bereits bestanden hat, erhält Zugang zur GOALS-Plattform!!!
- Sie müssen bis spätestens 24. Juli mindestens 80% der Punkte in der GOALS Plattform erreicht haben
- Bis zum Ende des Python-Teils der Vorlesung müssen Sie bereits mindestens 30% der Punkte in der GOALS Plattform erreicht haben
- Alle weiteren Informationen zum Übungsbetrieb mit der GOALS-Plattform erhalten Sie dann in den Übungen

Klausur: 120 Minuten (eKlausur)

- Prüfungstermin: t.b.a
- **Anmeldefristen beachten**
- **Informationspflicht der Studierenden: Sie müssen sich selbst über die Fristen bei Ihrem zuständigen Prüfungsamt erkundigen, wir können Sie nicht zur Klausur anmelden**

Hilfsmittel: keine erlaubt

Zum Bestehen: sind $\geq 50\%$ der erreichbaren Punkte nötig

Probeklausur: (ebenfalls in elektronischer Form) wird wahrscheinlich in der letzter Vorlesungswoche besprochen

Bekanntgabe der Ergebnisse der Klausur: Unisono

- Einsichtnahme in die Klausur ist möglich (Termin und Regularien werden ihnen per E-Mail nach Fertigstellung der Korrektur mitgeteilt)

Prüfungsvorleistungen:

- ALLE Studierende haben Übungspflicht
- Studierenden die bereits eine Zulassung zur OFP-Klausur haben (aus vorherigen Semestern) haben keine Übungspflicht.
- Durch den Einsatz der GOALS Plattform erfolgt die Bewertung dahingehend das mindestens 80% der Lerninhalte erfolgreich bearbeitet sein müssen. Das wird alles aber noch in der 1. Übung genau erklärt

Sie müssen sich auch in Unisono zur **Studienleistung** OFP anmelden

- Prüfungsnr. “800213-SL“, “4INFBA004-S“, ...
- Sollten Sie Probleme bei der Anmeldung zur Studienleistung haben, wenden Sie sich bitte an Ihr **Prüfungsamt**. Ggf. ist erst während des Semesters eine Anmeldung möglich.

Wichtig: Melden Sie sich **zur Prüfung UND zur Studienleistung** an

Verantwortung der Studierenden

- Informationen *aktiv* beschaffen
 - Vorlesungs-Webseite: Termine, Ankündigungen, Folien, Übungen
 - Prüfungsamt: Anmeldefristen, Klausuranmeldung
 - Bibliothek, ...
- Übungen *aktiv* bearbeiten
- Vorlesung *schritthaltend* nacharbeiten, auch mit weiterführender Literatur
- Aufbau von Lerngruppen
- An der Universität läuft Ihnen niemand nach

Das Geburtstagsparadoxon

Teil 1:

Einführung in Python und Konzepte funktionaler Programmierung

```
1-reduce(lambda x, y: x * y, range(366-n, 366))/365**n
```

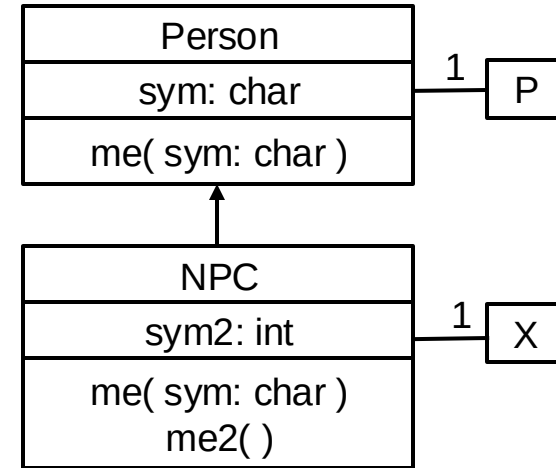
[Hunt19] John Hunt: A Beginners Guide to Python 3 Programming (Undergraduate Topics in Computer Science), 2019.

[Er99] Martin Erwig: Grundlagen funktionaler Programmierung, Oldenbourg, 1999.

Teil 2:

Einführung in Java und Konzepte objektorientierter Programmierung

```
class Person {  
    protected char sym = 'Ä';  
    public boolean me(char sym) { return this.sym == sym; }  
    public Person() { sym = '?'; }  
}  
class NPC extends Person {  
    private int sym2 = 0;  
    public boolean me(char sym) { return sym2 == (int) sym; }  
    public void me2() { this.sym2 = (int) super.sym; }  
    public NPC() { super(); sym2 = (int) '?'; }  
}
```



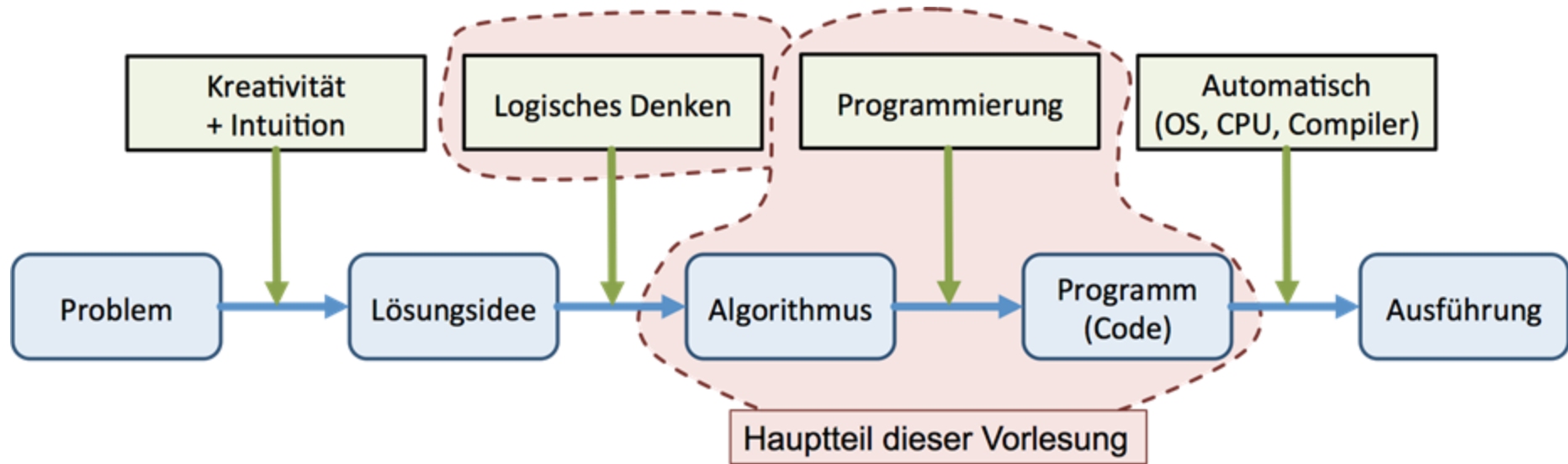
[Ba99] Helmut Balzert: Lehrbuch Grundlagen der Informatik, Spektrum-Verlag, 1999.

[Ba05] Heide Balzert: Lehrbuch der Objektmodellierung, Spektrum, 2005.

Lernplattformen

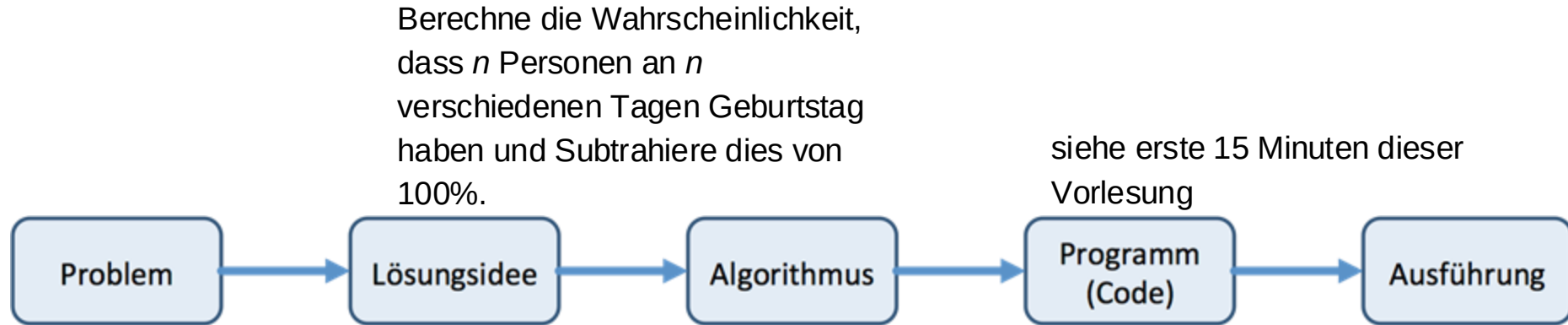
Folgende Lernplattformen ermöglichen eine oftmals spielerische Einführung in die Welt der Programmierung. Hier finden auch bereits geübte Programmierer weitere Herausforderungen:

- <https://www.freecodecamp.org>
- <https://www.codecademy.com>
- <https://www.sololearn.com>
- <https://www.hackerrank.com>



Ein **Algorithmus** ist eine Beschreibung, wie ein Problem mittels individueller kleiner Teilschritte gelöst werden kann

Programmcode ist die Umsetzung eines Algorithmus in einer bestimmten Programmiersprache

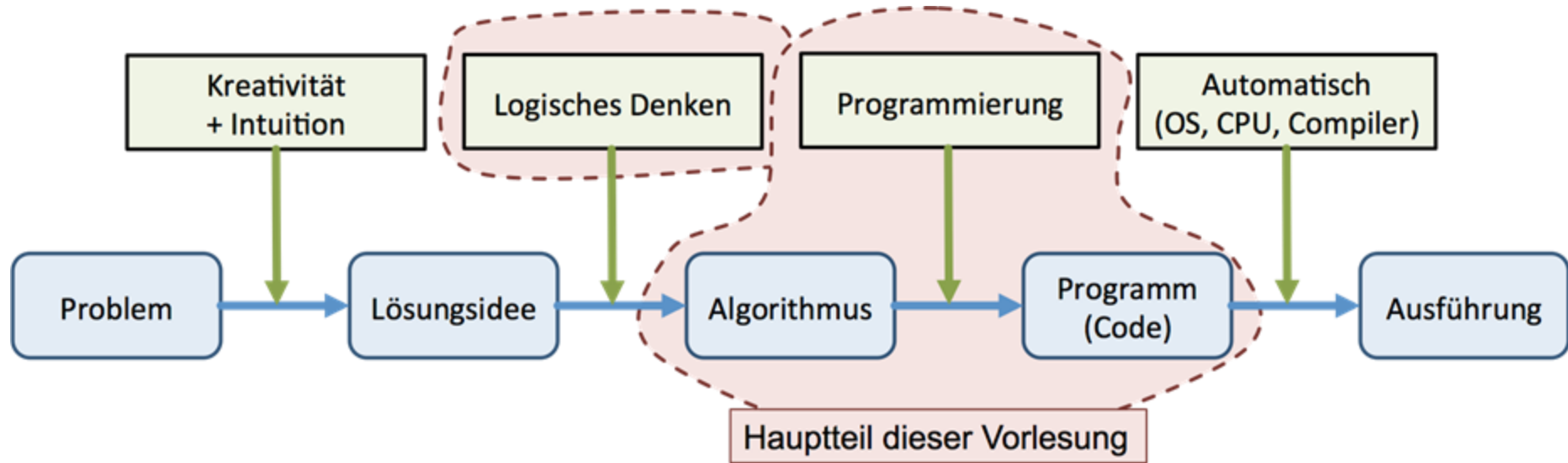


Was ist die Wahrscheinlichkeit, dass mindestens zwei von n Personen am gleichen Tag Geburtstag haben?

$$1 - \frac{365 \cdot 364 \cdot \dots \cdot (365 - n + 1)}{365^n}$$

1. **Hole** die Zutaten (Mehl, Wasser, Salz, Hefe, Tomatensauce, Käse, Gemüse ...)
2. **Zuerst**, siebe 500g Mehl in eine Schüssel
3. **Dann** füge einen halben Würfel Hefe, 10 g Salz und 320ml lauwarmes Wasser hinzu und lass alles gut von einer Küchenmaschine durchkneten. Danach muss der Teig 30 Minuten ruhen.
4. **In der Zwischenzeit**, wasche das Gemüse und rasple den Käse klein.
5. Rolle den Teig aus, belege ihn und streue **solange** Käse hinüber **bis** alles mit Käse bedeckt ist
6. **Falls** sie Ober- und Unterhitze verwenden, backen sie für 20 Minuten bei 230 Grad, **sonst** backen sie für 15 Minuten bei 210 Grad Heißluft

Algorithmen vs. Programme



“One hour of thinking can easily save one week of programming!”
Or: programming is not trial-and-error!

Prozedurale Programmierung – basiert auf einer Sequenz von Befehlen, die dem Computer sagen, was getan werden muss. Diese werden oft von Kontrollstrukturen wie *if statements* oder *Schleifen* kontrolliert.

Beispiele: *C* oder *Pascal*

Deklarative Programmierung – erlaubt es Programmierer*innen zu beschreiben, wie ein Problem gelöst werden soll, während die genaue Umsetzung der Sprache/Umgebung überlassen wird.

Beispiele: *Prolog* oder *SQL*

Prozedurale Programmierung – basiert auf einer Sequenz von Befehlen, die dem Computer sagen, was getan werden muss. Diese werden oft von Kontrollstrukturen wie *if statements* oder *Schleifen* kontrolliert.

Beispiele: *C* oder *Pascal*

Deklarative Programmierung – erlaubt es Programmierer*innen zu beschreiben, wie ein Problem gelöst werden soll, während die genaue Umsetzung der Sprache/Umgebung überlassen wird.

Beispiele: *Prolog* oder *SQL*

```
DROP TABLE Speisen;

CREATE TABLE Speisen
(
  id          INTEGER      NOT NULL,
  Gericht     VARCHAR(255) NOT NULL,
  Preis       FLOAT        NOT NULL,
  Zutaten     VARCHAR(255),
  UNIQUE ( id ),
  UNIQUE ( Gericht ),
  PRIMARY KEY ( id )
);

INSERT INTO Speisen VALUES ( 101, 'Pizza Diabolo', 5.50, 'Teufelsohren' );
```

Prozedurale Programmierung – basiert auf einer Sequenz von Befehlen, die dem Computer sagen, was getan werden muss. Diese werden oft von Kontrollstrukturen wie *if statements* oder *Schleifen* kontrolliert.

Beispiele: *C* oder *Pascal*

Deklarative Programmierung – erlaubt es Programmierer*innen zu beschreiben, wie ein Problem gelöst werden soll, während die genaue Umsetzung der Sprache/Umgebung überlassen wird.

Beispiele: *Prolog* oder *SQL*

Objektorientierte Programmierung – erstellt Lösungen anhand von (als abstrakten Klassen definierten) Objekten, die Daten beinhalten und bestimmte Funktionen ausführen können. Dinge wie eine Spezialisierung (Fahrzeug -> Auto, Fahrzeug -> Fahrrad) und die Vererbung von Eigenschaften spielen hier eine große Rolle.

Beispiele: *Java* oder *C++*

Funktionale Programmierung – beschreibt die Lösung eines Problems strikt durch Auswertung von (*Pure*) Funktionen, deren Verhalten ausschließlich von den Eingaben und nicht von externen Einflüssen abhängt und deren Auswirkung nur das Ausgabeargument ist (= es gibt keine Nebeneffekte).

Beispiele: *Haskell* oder *Meta Language (ML)*

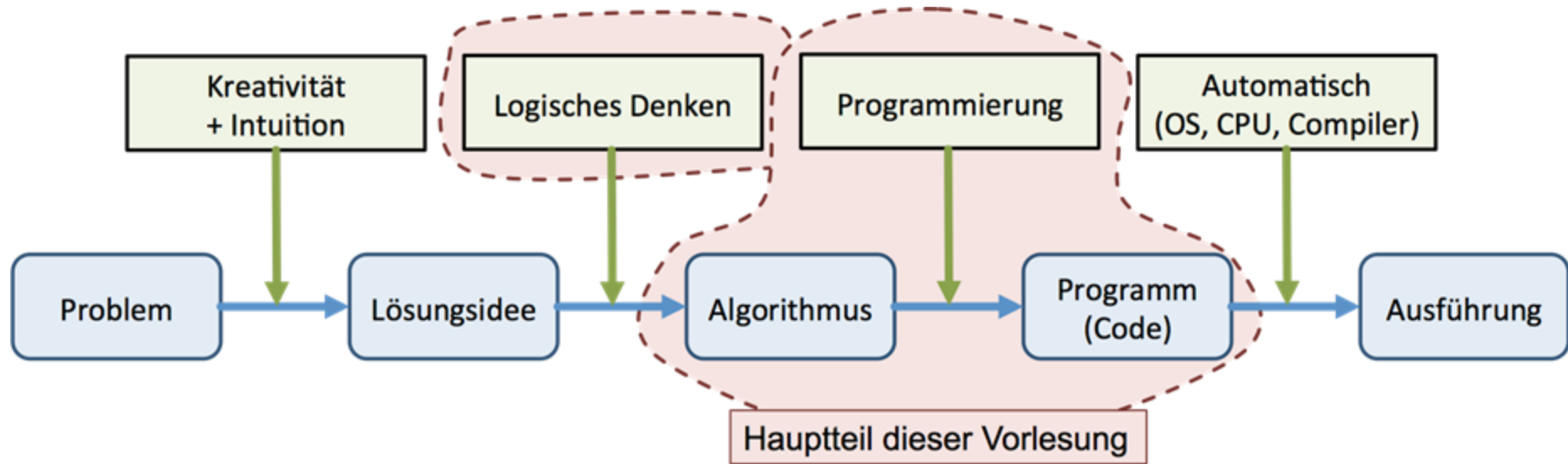
Python ist eine hybride und keine funktionale Sprache. Die Bedeutung von Python in der Praxis geht aber weit über die von Haskell oder ML hinaus, weshalb wir uns für eine Einführung in Python mit einigen funktionalen Konzepten entschieden haben.

Für jede Programmiersprache benötigen wir einen Compiler oder Interpreter, der unseren (text-/buchstabenbasierten) Code für den Computer in etwas ausführbares ‚übersetzt‘ (da der Computer ja nur in Binärdarstellung arbeitet).

Eigentlich nur noch einen Texteditor und eine Kommandozeile. Viel beliebter ist jedoch die Programmierung in einer IDE (integrated development environment).

Diese besteht meist aus

- einem Editor mit Syntax-Highlighting,
- der Möglichkeit Code direkt zu compilieren und auszuführen,
- und evt. Optionen für das Finden von Fehlern (Debugging).



- Programmierung mit Python, Funktionale Programmierung
- Objektorientierte Modellierung mit UML und Java