Computational Thinking

Algorithmisches Denken mithilfe des Jupyter-Ökosystems

Dr. Benedikt Zönnchen
Prof. Dr. Martin Hobelsberger

HM

19. Mai 2022

Überblick

1. **Herausforderung**
2. **Kurskonzeption**
3. **Realisierung**
   - 3.1 **Intuitiver Zugang**
   - 3.2 **Minimale technische Hürden**
   - 3.3 **Kontinuierliches Feedback**
4. **Retrospektive**

Herausforderung

Vielfältige interdisziplinäre CSPlus / PlusCS Studiengänge der Hochschule München:

- Digital Engineering
- Informatik und Design
- Geodata Science
- Data Science & Scientific Computing
- . . . (in der Zukunft)

⇒ Erhöhte Heterogenität der Studierenden

Herausforderung

Studierende jener Studiengänge soll es möglich sein

> *„Aspekte realweltlicher Probleme zu identifizieren, die für eine informatische Modellierung geeignet sind, algorithmische Lösungen für diese (Teil-)Probleme zu bewerten und selbst so zu entwickeln, dass diese Lösungen mit einem Computer operationalisiert werden können.“– Julian Fraillon et al, 2019 [3]*

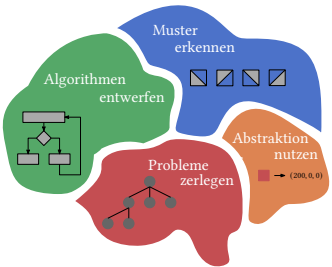
⇒ Studierende sollen **Computational Thinking (CT)** beherrschen.

Überblick

1. Herausforderung
2. **Kurskonzeption**
3. Realisierung
   - 3.1 Intuitiver Zugang
   - 3.2 Minimale technische Hürden
   - 3.3 Kontinuierliches Feedback
4. Retrospektive

Was verstehen wir unter CT?

„It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.“
– Jeannette M. Wing, 2006 [8]



Computational Thinking beschreibt eine Denkweise, bei welcher der Computer als Instrument benutzt wird, um so den Denkprozess zu unterstützen.

Lerninhalte

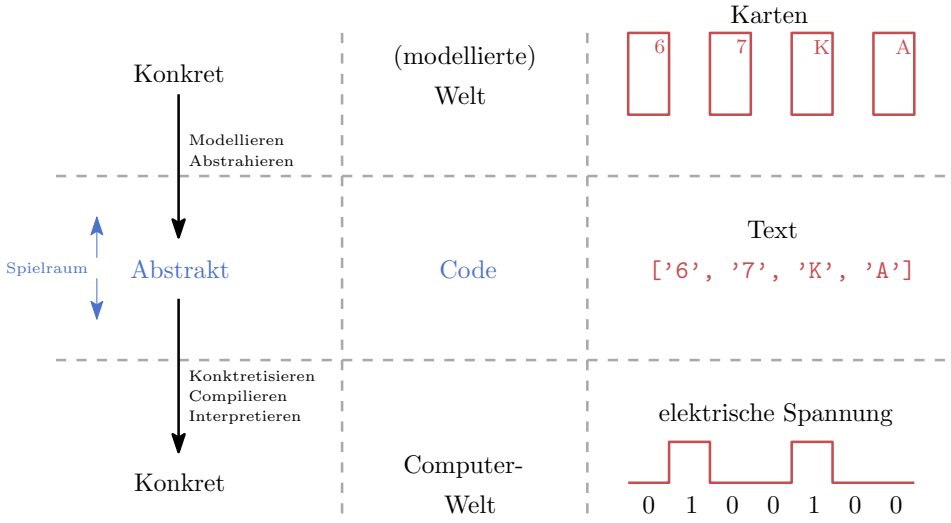
Die Auswahl der Lerninhalte viel uns schwer und steht offen zur Diskussion.

(1) Informationsverarbeitung (des digitalen Computers):
   - Interpretation
   - Repräsentation
   - Manipulation
   - ...

(2) Datenstrukturen und Algorithmen (mit Python)
   - Variablen, Ausdrücke und Funktionsaufrufe
   - Datentypen (Zahlen, Zeichenketten, Listen, Mengen, Wörterbücher, ...)
   - Funktionen
   - Kontrollstrukturen
   - OOP

(3) CT in Aktion

Lerninhalte



Kurskonzept

Rahmen

Bachelor 1. Semester, Vorlesungen (4 SWS) und Praktika (4 bzw. 2 SWS).

Ziel

- Lernen durch selbständiges (Wieder-)Entdecken
- frühzeitig ins “Doing” kommen

⇒ **Praktika greifen voraus!**

Bedingungen

- intuitiver Zugang
- minimale technische Hürden
- kontinuierliches Feedback beim “Doing”
- Entmystifizierung

Überblick

1. Herausforderung
2. Kurskonzeption
3. **Realisierung**
   - **3.1 Intuitiver Zugang**
   - **3.2 Minimale technische Hürden**
   - **3.3 Kontinuierliches Feedback**
4. Retrospektive

Hilfsmittel

- Material zum Anfassen: Karten, Bücherstapel, Lampen, . . .
- Programmiersprachen: Python (+ JavaScript [6] für *Informatik und Design*)
- Aufgabengenerierung: Otter-Grader [7]
- Entwicklungsumgebung:
  - Jupyter-Notebooks (+ P5.js sketch) [4]
  - im JupyterLab oder VSCode [4]
  - gehostet via JupyterHub [5, 4]
- Lehrbuch: Interaktives CT-Buch als JupyterBook [1, 2]
- Roboworld Python Package [9, 10]

Python

Intuitiver Zugang

Vorteile:

- hohe Abstraktion
- Nützlichkeit und Relevanz
- lineare Lernkurve
- schnelle Erfolgserlebnisse
- sehr gutes Ökosystem

Nachteile:

- hohe Abstraktion
- dynamische Typisierung
- keine primitiven Datentypen
- kein echtes Multi-Threading

Notebooks

Intuitiver Zugang

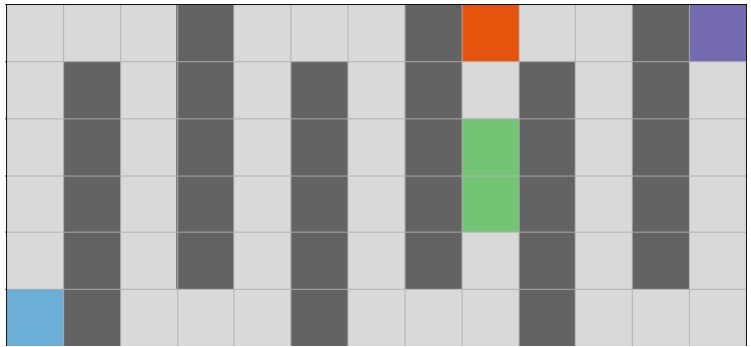
Vorteile:

- einfacher Einstieg
- Synergie zwischen Text und Code
- ermöglicht serverseitige Codeausführung
- zellenweise Auswertung (aka Debugging)
- in sich abgeschlossen

Nachteile:

- schwer zu Warten
- erschweren Zusammenarbeit
- ungeeignet für Anwendungsentwicklung

Roboworld



Roboworld

Roboworld Demo: https://datahub.cs.hm.edu/

JupyterHub

Eigener JupyterHub auf unserem Kubernetes-Cluster:

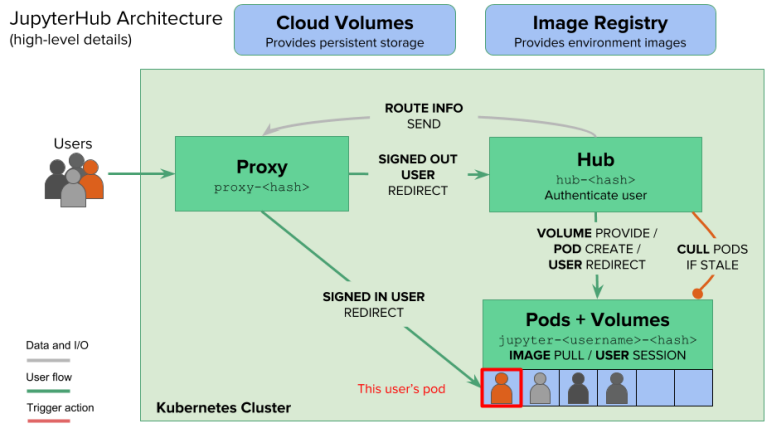


Abbildung: Quelle: The JupyterHub Architecture

JupyterHub

Minimale technische Hürden

Vorteile:

- einheitliche vorkonfigurierte Entwicklungsumgebung
- Aufgabenverteilung durch git (im Hintergrund)
- starke Gemeinschaft
- Rechenleistung auf Serverseite

Nachteile:

- keine individuelle Entwicklungsumgebung
- fehlende Interaktion mit git-Befehlen
- keine Funktionalität für Abgaben (Praktikum, Prüfung)
- Rechenleistung auf Serverseite (ab ca. 100 Benutzer:innen wird Kubernetes benötigt)

Otter-Grader

Otter-Grader Demo

Otter-Grader

Kontinuierliches Feedback

Vorteile:

- realisiert **kontinuierliches Feedback** für Studis
- einfach zu bedienen (für Lehrende als auch für Studis)
- Code, Aufgabenstellung und Tests in einem Dokument

Nachteile:

- funktioniert ausschließlich mit Notebooks
- Auto-Grading ist nicht voll automatisiert
- (sichtbare) Tests werden mitgeliefert

Jupyter-Book

CT-Jupyter-Book Demo

Retrospektive

- Besserer Dialog zwischen Lehrende und Lernende notwendig (insbesondere während der Pandemie)
- Noch mehr “Doing” der Studis
- Zu Beginn kleinteiligere Problemstellungen
- Programmiersprachenspezifika unvermeidbar?
- Tempo musste gedrosselt werden
- Python + Jupyter-Ökosystem eignen sich für CT

Quellen & Material

Vortragsunterlagen: https://github.com/BZoennchen/fdak-sep

[1] Benedikt Zönnchen, Sarah Ottinger, M. H. (2021). Computational Thinking. https://bzoennchen.github.io/ct-book/intro.html.

[2] Community, T. J. B. (2022). Jupyter Book Documentation. https://jupyterbook.org/en/stable/intro.html.

[3] Fraillon, J., Ainley, J., Schulz, W., Duckworth, D., and Friedman, T. (2019). *Computational thinking framework*, pages 25–31. Springer International Publishing, Cham.

[4] Jupyter, P. (2021a). Jupyter. https://jupyter.org/.

[5] Jupyter, P. (2021b). JupyterHub Documentation. https://jupyter.org/hub.

[6] Processing Foundation, N. (2021). P5.js. https://p5js.org/.

[7] Team, U. B. D. S. E. P. I. (2021). Otter-Grader Documentation. https://otter-grader.readthedocs.io/en/latest/.

[8] Wing, J. M. (2006). Computational Thinking. *Commun. ACM*, 49(3):33–35.

[9] Zönnchen, B. (2021a). Roboworld. https://pypi.org/project/roboworld/.

[10] Zönnchen, B. (2021b). Roboworld Documentation. https://robo-world-doc.readthedocs.io/en/latest/index.html.