Computational Thinking

Algorithmisches Denken mithilfe des Jupyter-Ökosystems

Dr. Benedikt Zönnchen
Prof. Dr. Martin Hobelsberger

HM

19. Mai 2022

Überblick

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

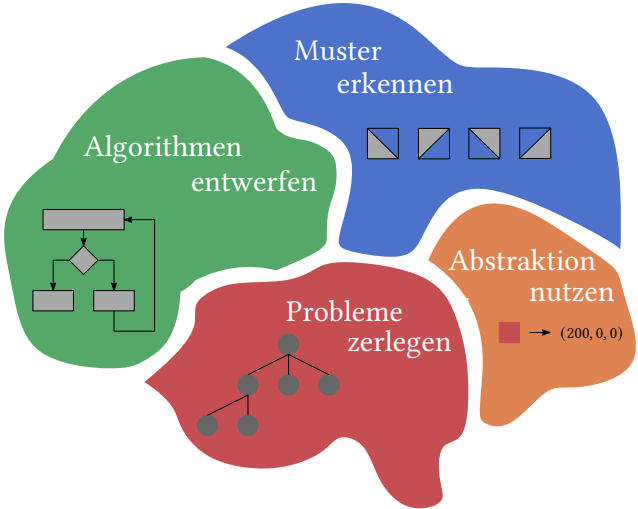
Studierenden jener Studiengänge soll es möglich sein

> *„**Problemstellungen** zu **identifizieren**, **abstrakt** zu **modellieren**, sie dabei in **Teilprobleme** zu **zerlegen**, **Lösungsstrategien** zu **entwerfen** und auszuarbeiten und diese **formalisiert darzustellen**, sodass sie von einem Menschen (oder auch einem Computer) verstanden und ausgeführt werden können.“*

Kurz gesagt: Studierende sollen **Computational Thinking (CT)** beherrschen.

Überblick

Was verstehen wir unter CT?



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

Kurskonzept

Rahmen

Der CT-Kurs besteht aus Vorlesungen (2 SWS) und Praktika (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 für *Informatik und Design*)
- Aufgabengenerierung: Otter-Grader
- Entwicklungsumgebung:
  - Jupyter-Notebooks (+ P5.js sketch)
  - im JupyterLab oder VSCode
  - gehostet via JupyterHub
- Lehrbuch: Interaktives CT-Buch als JupyterBook
- Roboworld Python Package

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

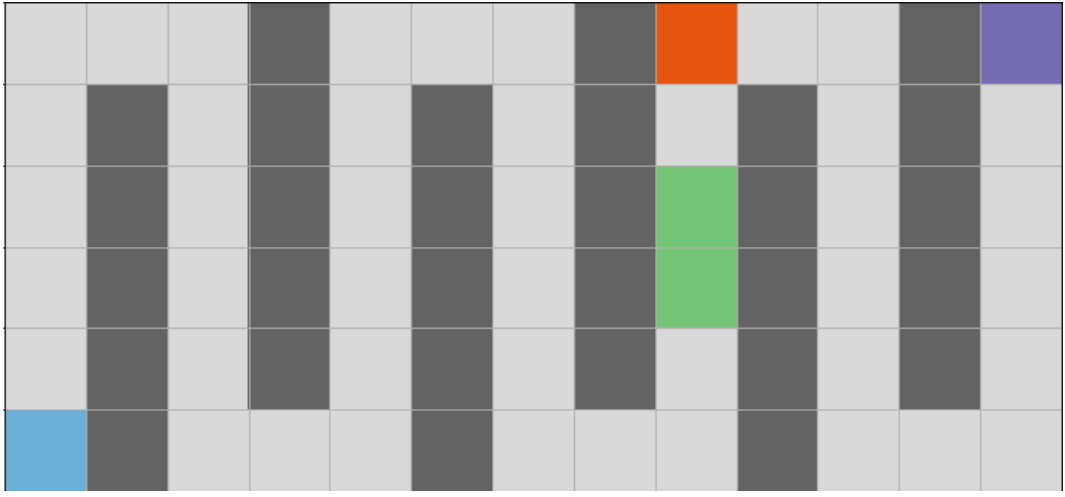
Vorteile:

- 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:

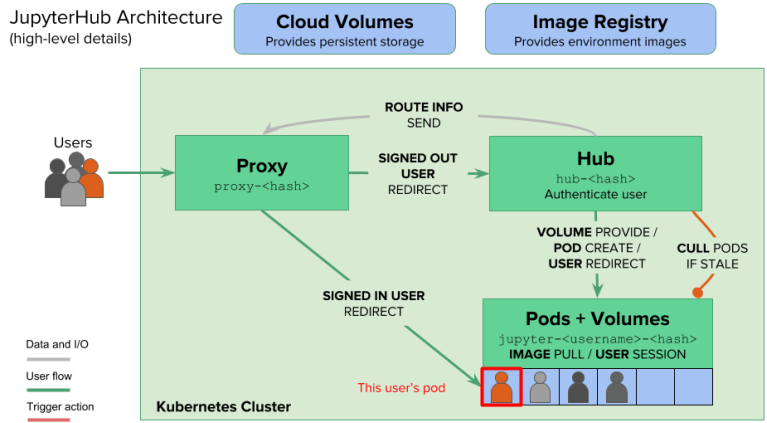


Abbildung: Quelle: The JupyterHub Architecture

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 Demo

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

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

Jupyter-Book: https://bzoennchen.github.io/ct-book/intro.html

Roboworld Doku: https://robo-world-doc.readthedocs.io/en/latest/index.html

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

Otter-Grader: https://otter-grader.readthedocs.io/en/latest/

JupyterHub: https://jupyter.org/hub

Jupyter-Notebooks: https://jupyter.org/