Computational Thinking

Algorithmisches Denken mithilfe von Jupyter-Notebooks

Prof. Martin Hobelsberger
Dr. Benedikt Zönnchen

19. Mai 2022

Herausforderung

Vielfältige interdisziplinäre CSPlus / PlusCS Studiengänge der Hochschule München:

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

$\Rightarrow$ Erhöhte Heterogenität der Studierenden

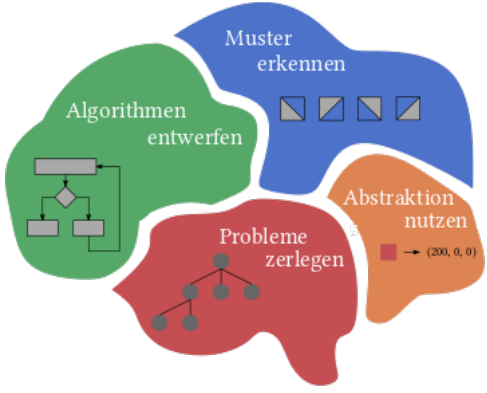
Herausforderung

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.

Was verstehen wir unter CT?



Was verstehen wir unter CT?

Computational Thinking

- ist eine Aktivität
- bedeutet primär konzipieren (Programmierung als Mittel)
- ist eine fundamentale Fähigkeit
- ist eine Denkweise des Menschen (nicht der Maschine)
- kombiniert analytisches, abstraktes und schaffendes Denken

Warum CT?

Drei wesentliche Punkte weshalb wir CT für so wichtig halten sind:

(1) Intrinsischer Wert

(2) (Langfristiger) Nutzen

(3) Selbstbestimmung in der digitalen Ära

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

CT besteht aus Vorlesungen (2 SWS) und Praktika (2 SWS).

Ziel

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

⇒ **Praktika greifen voraus!**

Bedingungen

- Minimale technische Hürden
- “Intuitive” und gut dokumentierte Programmiersprache als Werkzeug
- Kontinuierliches Feedback beim “Doing”
- Entgegenwirken des “Nerdfaktors”

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

Infrastruktur

Eigener JupyterHub auf einem Kubernetes-Cluster:

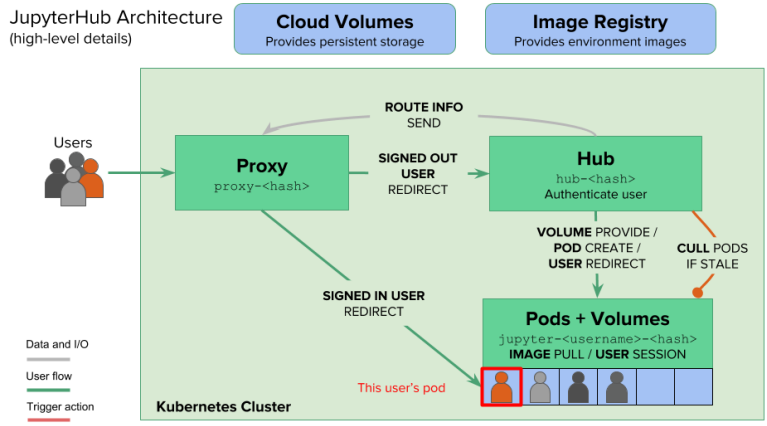


Abbildung: Quelle: The JupyterHub Architecture

Infrastruktur

GitLab gespiegelt auf GitHub