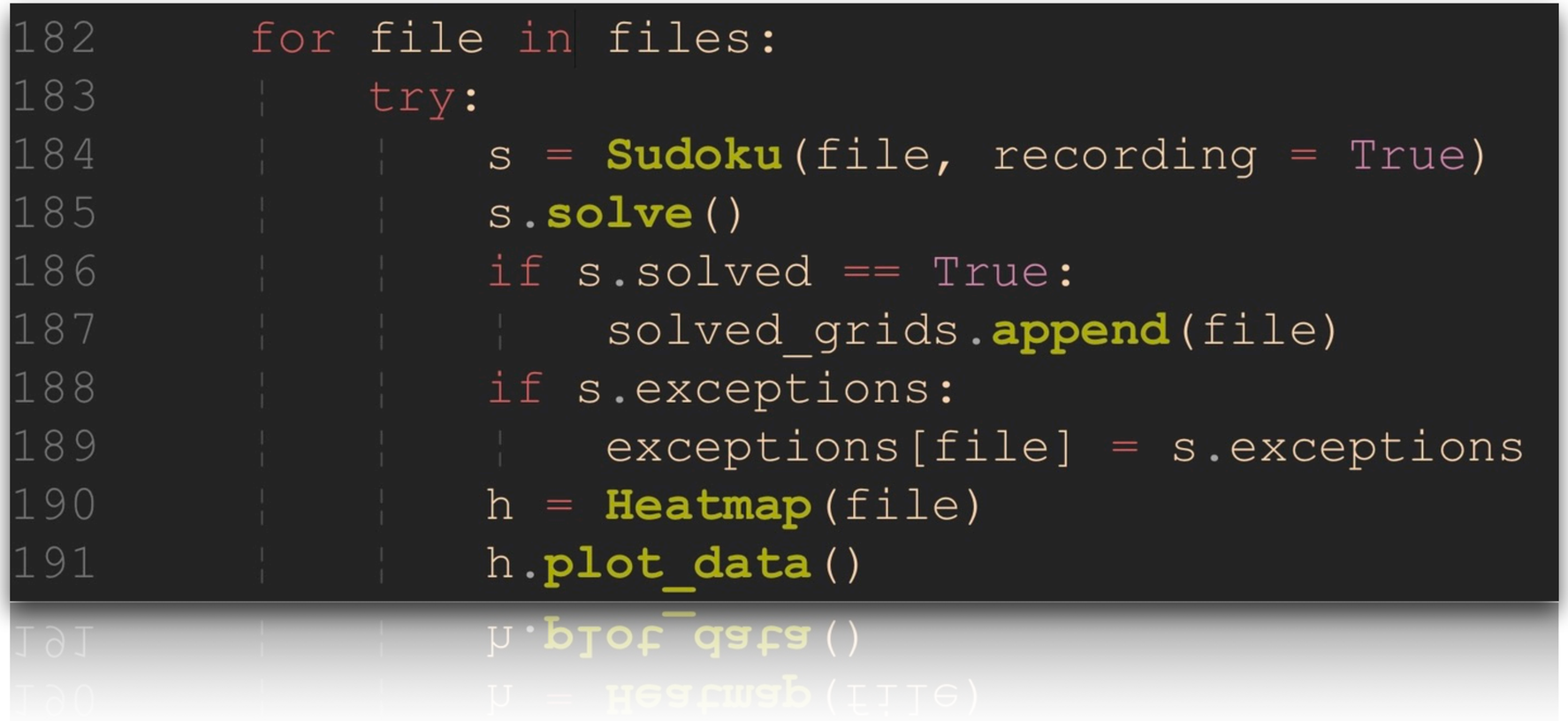
Sudokus lösen mit Python



```
182     for file in files:
183         try:
184             s = Sudoku(file, recording = True)
185             s.solve()
186             if s.solved == True:
187                 solved_grids.append(file)
188             if s.exceptions:
189                 exceptions[file] = s.exceptions
190             h = Heatmap(file)
191             h.plot_data()
```

Abschlusspräsentation - "Programmieren mit Python"
Kurs bei Herrn Dr. Feiler

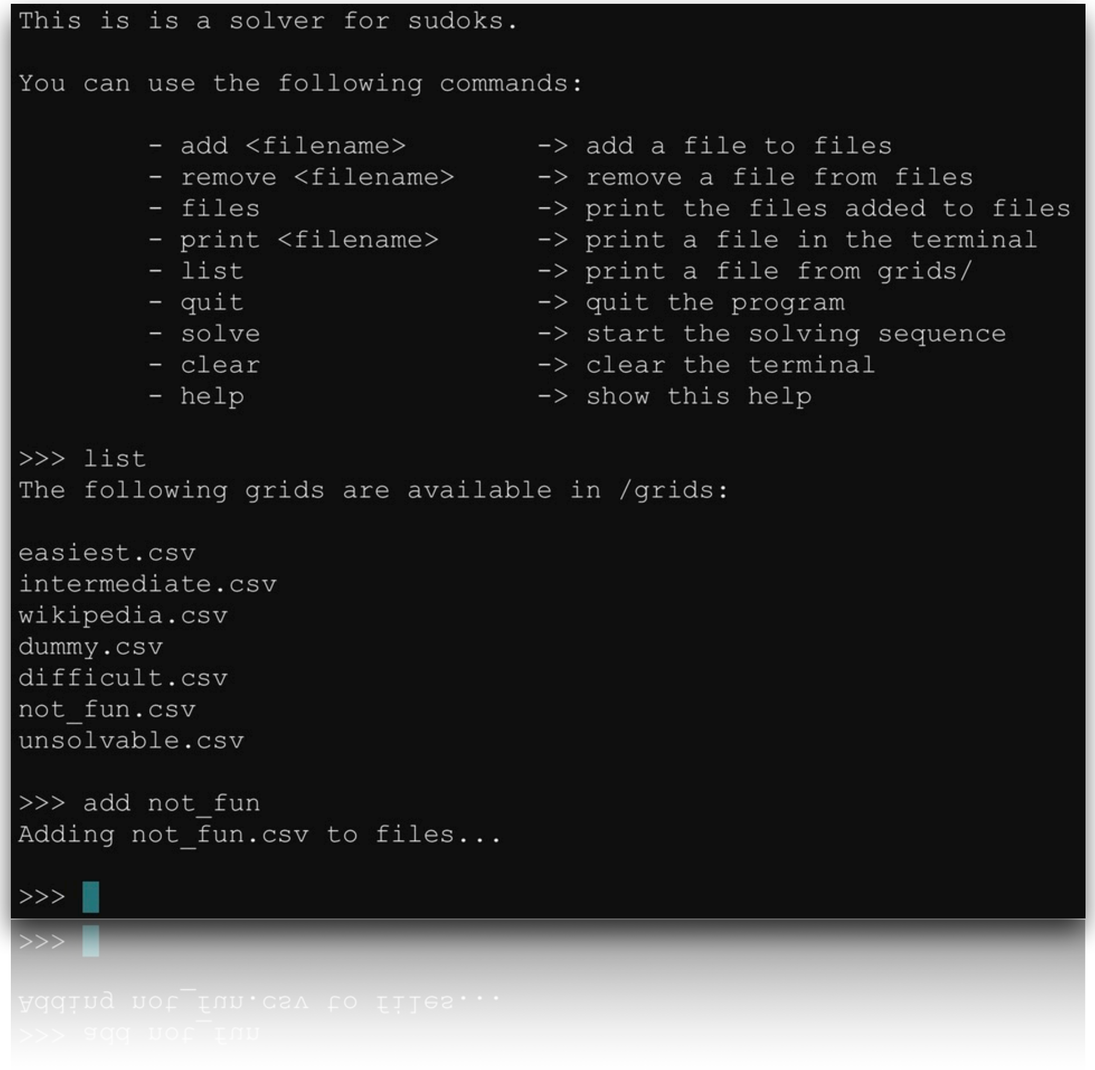
Tobias Hahn, 10. Dezember 2020

Inhalt

- Features des Projekts
- Angewendete Techniken
- Grundlagen von Sudokus
- Vorstellung des Algorithmus
- Ausblick
- Live-Demonstration

Features des Projekts

- Sequenzielles Lösen von Sudokus
- Datenaufzeichnung und -visualisierung
- Klassen zum Lösen und Plotten
- User-Interface für die Kommandozeile
- Tool zum Verifizieren des Solvers



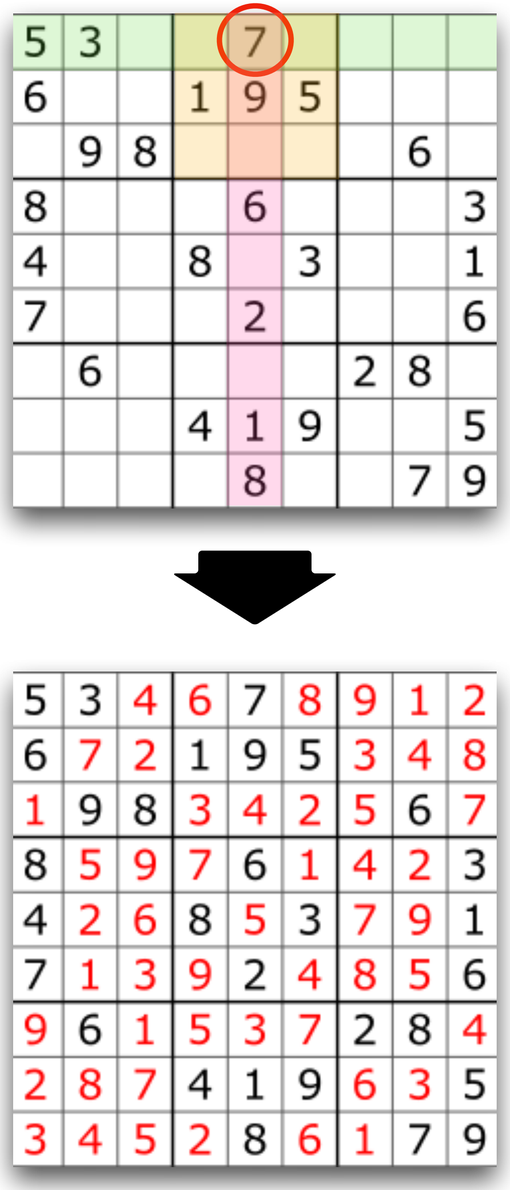
Angewendete Techniken

- Objektorientierte Programmierung
- Interaktion mit dem User per input()
- Daten Input / Output mit csv und json
- Abfangen von Fehlern mit try / except
- Plotten mit matplotlib.pyplot

Sudoku (klassisch)

Logisches Zahlenrätsel

- Quadratisches Zahlengitter mit 9 x 9 Zellen
- Es sind mehr oder weniger Ziffern vorgegeben
- Das Gitter muss mit den Ziffern 1 bis 9 befüllt werden
- Pro Zeile, Spalte und "Einheit" (3 x 3 Sub-Zelle) darf jede Ziffer nur ein Mal vorkommen
- Üblicherweise Lösen durch einen Menschen
  - Logisches Denken
  - Kombinieren



Quelle: https://en.wikipedia.org/wiki/Sudoku (08.12.2020)

Sudoku

Lösen mit Backtracking: Brute Force

```
 1 def step():
 2     if ziffer[cursor] < 9:
 3     ¦   ziffer += 1
 4     ¦   if loesung_moeglich():
 5     ¦   ¦   cursor += 1
 6     else:
 7     ¦   backtrack()
 8
 9 def backtrack():
10     ziffer[cursor] = 0
11     cursor -= 1
12
13 while check_solution() == False:
14     try:
15     ¦   step()
16     except IndexError:
17     ¦   break
```

Grober Ablauf des Backtracking-Algorithmus

5	3	1	2	7	3	1		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Vorwärtsschritte

Sudoku

Lösen mit Backtracking: Brute Force

```
 1 def step():
 2     if ziffer[cursor] < 9:
 3     ¦   ziffer += 1
 4     ¦   if loesung_moeglich():
 5     ¦   ¦   cursor += 1
 6     else:
 7     ¦   backtrack()
 8
 9 def backtrack():
10     ziffer[cursor] = 0
11     cursor -= 1
12
13 while check_solution() == False:
14     try:
15     ¦   step()
16     except IndexError:
17     ¦   break
```

Grober Ablauf des Backtracking-Algorithmus

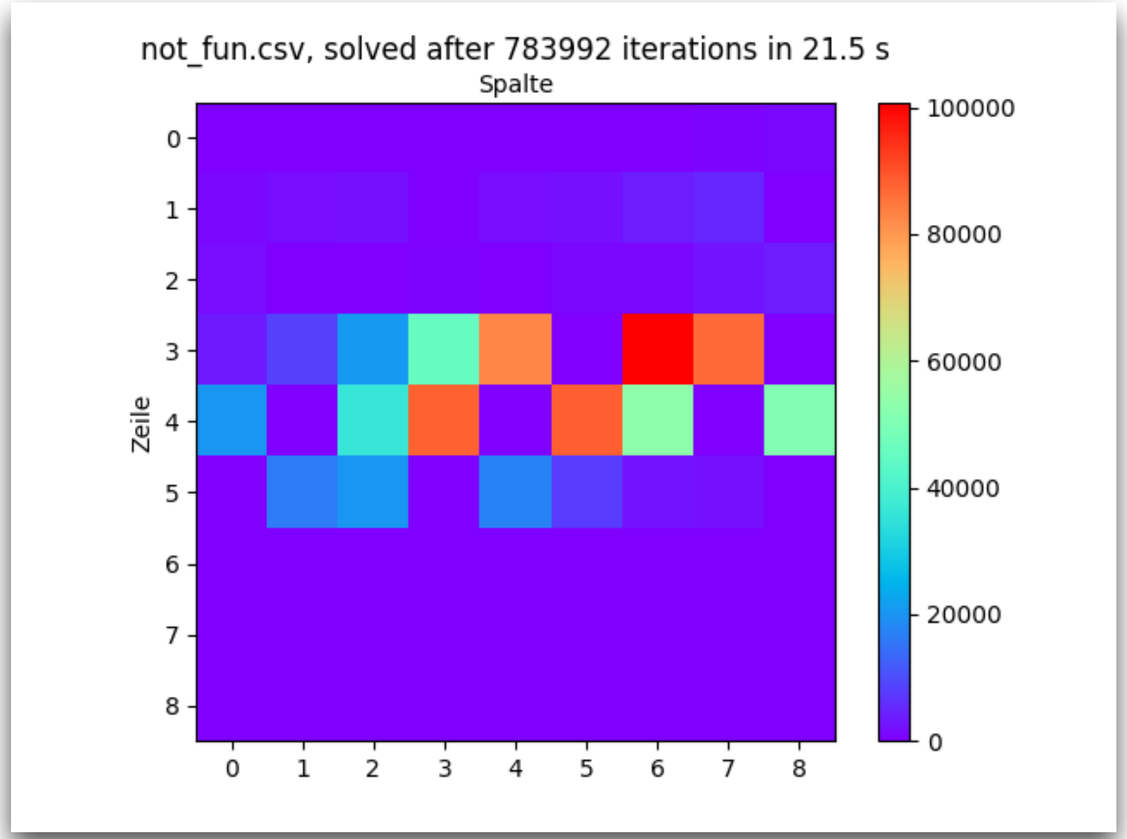
5	3	1	2	7	4	9	9	9
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Backtracking

Visualisierung der Daten

Statstiken und Heatmap

- Daten aus dem Lösungsprozess werden als .csv und .json aufgezeichnet
- In einem Diagramm werden dargestellt:
  - Der Dateiname
  - Die Anzahl der Iterationen
  - Die benötigte Zeitdauer
  - Eine Heatmap zeigt, wie oft welche Zelle iteriert wurde



Exemplarischer Plot

Ausblick

- Refactoring des sehr umfangreichen Codes
- Ausführlichere Kommentare schreiben
- Lösen von Anti-Brute-Force-Rätseln
- Implementierung analytischer Methoden
- Bedienung mit einem GUI
- Schönere Gestaltung der Plots
- Die spielerisch angewendeten Techniken lassen sich auch für "echte" Probleme einsetzen

					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

Keine Lösung nach 10^9 Iterationen

```
1 if fragen:
2         for frage in fragen:
3         |     frage.stellen()
4 else:
5         live_demo()
```