



## Portfolioprüfung - Werkstück A im Modul

### Betriebssysteme und Rechnernetze

Studiengang International Business Information System

### Alternative 5-Sudoku- Dokumentation zum Code

Vorgelegt von

Mustafa Bagci - 1394757

Antonia Katharina Kirsch - 1349935

Fabian Topf - 1398083

Farhat Hossain - 1402063

## Inhalt

Spielablauf.....	3
Sudoku Erstellen.....	4
Spieler 1 und Spieler 2.....	5
Funktion checkSudokuBox() .....	6
Funktion kontrolleFeld().....	6
Print Sudoku Box .....	8
Funktion sudokugewonnen() .....	9
Literatur und Quellen .....	10

**Das Ziel der Portfolioprüfung ist, dass man ein Sudoku-Spiel entwickelt und implementiert, so dass Benutzer gegeneinander in der Shell spielen können.**

## Spielablauf

```
306 #AUSFÜHRUNG DES CODES
307 Spieler1name = input("Name des 1. Spielers eingeben: ")
308 Spieler2name = input("Name des 2. Spielers eingeben: ")
309 auslesenUNDerstellen()
310 zeitmessung_anfang_spieler1 = datetime.datetime.now() #Zeit zum Zeitpunkt des Spielstarts wird gespeichert
311 zeitmessung_anfang_spieler2 = zeitmessung_anfang_spieler1 #ebenso für Spieler 2
312 Spieler1()
```

Zu Anfang des Spiels muss man die Namen der zwei Spieler über eine Abfrage angeben, die während des Spielverlaufes immer wieder in Verwendung kommen. (Zeile 307 & 308)

Danach wird die Funktion `auslesenUNDerstellen()` aufgerufen die ein Sudoku Feld für die Spieler erstellt. (Zeile 309)

Ab dem Zeitpunkt der Erstellung wird auch die Zeit der Spieler gemessen. (Zeile 310 und 311)

`Spieler1()` und `Spieler2()` sind die Hauptbestandteile des Spiels. Sie werden ausgeführt und greifen auf die meisten Funktionen zu. Die Funktion `Spieler1()` und `Spieler2()` unterscheiden sich kaum, jedoch ist jedem Spieler eine eigene Funktion zugewiesen, um zwischen Spieler1 und Spieler2 zu switchen bzw. dann die andere Funktion auszuführen. Es wird sozusagen eine Dauerschleife erzeugt zwischen den beiden Funktionen, bis jemand das Spiel gewonnen hat, da wenn ein Spieler sein Zug beendet die Funktion vom anderen Spieler aufgerufen wird.

Zuallererst wird `Spieler1()` aufgerufen, danach läuft das Spiel in einer „dauerschleife“ zwischen den beiden Spielern. (Zeile 312)

Das Spiel kann entweder durch das Aufgeben des Gegenspielers oder durch das Eintragen des letzten fehlenden Feldes im Sudoku gewonnen werden.

Wenn man gewonnen hat, wird die Funktion `sudokugewonnen()` ausgeführt die dies farbig in der Konsole angezeigt mit der Zeit die man dafür gebraucht hat in Sekunden.

```
Name des 1. Spielers eingeben: a
Name des 2. Spielers eingeben: b
Wieviele Felder sollen entfernt werden? 1-81 3
a ist dran!
1. 4 8 3 | 9 2 1 | 6 0 7
2. 9 6 7 | 3 4 5 | 8 2 1
3. 2 5 1 | 8 7 6 | 4 9 3
-----+-----+-----
4. 5 4 0 | 1 3 2 | 9 7 6
5. 7 2 9 | 5 6 4 | 1 3 8
6. 1 3 6 | 0 9 8 | 2 4 5
-----+-----+-----
7. 3 7 2 | 6 8 9 | 5 1 4
8. 8 1 4 | 2 5 3 | 7 6 9
9. 6 9 5 | 4 1 7 | 3 8 2
Was möchten Sie tun?
1.Zahl eintragen
2.Zahl löschen
3.Passen
4.Aufgeben 4
#####
b HAT GEWONNEN und dafür 3 Sekunden gebraucht!
#####
```

## Sudoku Erstellen

Zu Beginn des Spiels muss man die Namen der zwei Spieler über eine Abfrage geben, die während des Spielverlaufes immer wieder in Verwendung kommen. (Zeile 307 & 308)

```
Spieler1name = input("Name des 1. Spielers eingeben: ")
Spieler2name = input("Name des 2. Spielers eingeben: ")
```

Danach startet die `auslesenUNDerstellen()` Funktion. Diese Funktion liest eine zufällige Textdatei aus der Liste `sudokutxt` (sudoku1.txt, sudoku2.txt – Zeile 33)

```
sudokutxt = ['sudoku1.txt', 'sudoku2.txt']

fp = open(random.choice(sudokutxt))
```

aus dem Ordner aus die einen voll ausgefülltes Sudoku enthält. Wenn man möchte, kann man auch weitere Textdateien in die Liste hinzufügen.

Die Textdateien werden Reihe für Reihe und Spalte für Spalte ausgelesen und in einer Liste `data` abgespeichert. Danach wird diese Liste auf die `s1` Liste übertragen, womit die Spieler dann spielen.

Als nächstes kommt eine Abfrage die fragt wieviele Felder entfernt werden sollen. Je mehr Felder entfernt werden desto schwieriger ist das Sudoku zu lösen. Die Felder werden dann zufällig ausgesucht und ein Feld kann nicht zwei mal zufällig „geleert“ werden da eine Abfrage eingebaut ist die schaut ob das jeweilige Feld bereits 0 ist.

```
if s1[x1][x2] != 0:
    s1[x1][x2]=0
```

Das jetzt erstellte Feld wird dann in einer weiteren `dataX` Liste abgespeichert die es ermöglicht im späteren Spielverlauf abzufragen ob das Feld Anfangs vorgegeben war oder nicht.

```
for i in range(9):
    for z in range(9):
        dataX[i][z] = s1[i][z]
```

Das `dataX` ist dazu notwendig, um später beim Löschen einer Zahl aus einem Feld zu schauen ob dieses Feld selbst eingetragen wurde oder durch das Erstellen selber vorgegeben war (Zeile 206 - 208)

```
if dataX[zeile][spalte]==0:
    s1[zeile][spalte]=0
    Spieler1()
```

Außerdem ist es für die farbige Gestaltung bei `printsudokubox2()` nötig um die vorgegebenen Felder von selbst ausgefüllten zu unterscheiden. (Zeile 124)

```
if dataX[i][z] == 0:
```

## Spieler 1 und Spieler 2

Wenn einer der beiden Funktionen aufgerufen wird, dann hat der Spieler die Möglichkeit zwischen vier Optionen zu wählen. -> 1. Zahl eintragen 2. Zahl löschen 3. Passen 4. Aufgeben. Diese Zahl wird als `Input` in der Variable `Auswahl` gespeichert und in einer if-Schleife je nach Auswahl weitergeführt.

### 1. Zahl eintragen

Der Spieler wird nacheinander nach der Zeile, Spalte und der Zahl abgefragt. Wenn seine Zahl die durch `kontrolleFeld()` gecheckt wird stimmt und somit das Feld konsistent ist, wird seine Zahl in das Feld eingetragen und es wird gecheckt ob das Sudoku Feld voll ist -> falls ja hat er gewonnen. Falls seine Zahl nicht stimmt, wird dies auch durch ein rotfarbiges print ausgegeben und der Zug wird an den anderen Spieler gegeben, somit hat er seine Chance vermasselt. Falls das Feld bereits vorgegeben war, wird dies auch angezeigt und der selbe Spieler kann nochmal zwischen den Optionen entscheiden.

```
if s1[zeile][spalte] == 0: #wenn das Feld keine Zahl ist also noch nicht ausgefüllt wurde
    if kontrolleFeld(zeile,spalte,eingabezahl)==True: #kontrolliert ob die eingegebene zahl im feld konsistent bzw richtig ist
        if eingabezahl > 0 and eingabezahl < 10: #catcht fehlermeldung
            s1[zeile][spalte]=eingabezahl
            printsudokubox2()
            if checkSudokuBox(s1): #checkt ob sudokubox komplett fertig ist
                sudokugewonnen(1)
            else:
                Spieler2() #spieler 2 ist dran
```

### 2. Zahl löschen

Dem Spieler wird wieder durch `input` abgefragt um welche Zeile und Spalte es sich handelt um die dortige Zahl zu löschen. Es wird überprüft, ob das Feld bereits vorgegeben war oder selber eingetragen wurde. Falls er bereits vorgegeben darf es nicht gelöscht werden und der Spieler ist erneut dran.

```
if dataX[zeile][spalte]==0: #guckt ob das Feld Anfangs vorgegeben war oder nicht weil es 0 wäre dann
    s1[zeile][spalte]=0 #setzt das jeweilige Feld dann auf 0
    Spieler1()
```

### 3. Passen

Der Spieler übergibt seinen Zug an den anderen Spieler.

```
elif auswahl == 3: #übergibt den Zug an Spieler 2
    print("Du übergibst dein Zug an",Spieler2name)
    Spieler2()
```

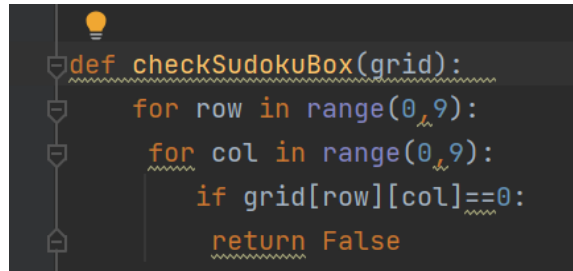
### 4. Aufgeben

Der jeweilige Spieler hat aufgegeben. Der andere Spieler gewinnt und das `sudokugewonnen()` wird ausgeführt für den anderen Spieler mit dem Argument in der Funktion (1 oder 2).

```
elif auswahl == 4:
    sudokugewonnen(2)
```

## Funktion checkSudokuBox()

In der 1. Abbildung ist die Funktion `checkSudokuBox()` abgebildet. Mit dieser Funktion kann überprüft werden, ob das Sudoku bereits vollständig ausgefüllt ist.



```
def checkSudokuBox(grid):  
    for row in range(0,9):  
        for col in range(0,9):  
            if grid[row][col]==0:  
                return False
```

Abbildung 1

Zunächst werden mithilfe der for-Schleife die Zeilen angegeben. Dabei benutzen wir die Funktion `range(0, 9)`, welche eine Liste von den Zahlen 0 bis 9 mit dem Namen `row` zeigt (Zeile 2). Dasselbe gilt dann auch für die Spalten, hier mit dem Namen `col` (Zeile 3). Der Rückgabewert „false“ innerhalb des if-Statements (Zeile 4) sorgt dafür, dass das standardmäßige Verfahren der for-Schleifen nicht stattfindet.

Wenn das Sudoku sich als vollständig ausgefüllt aufdeckt, wird „true“ ausgegeben und es gibt einen Gewinner.

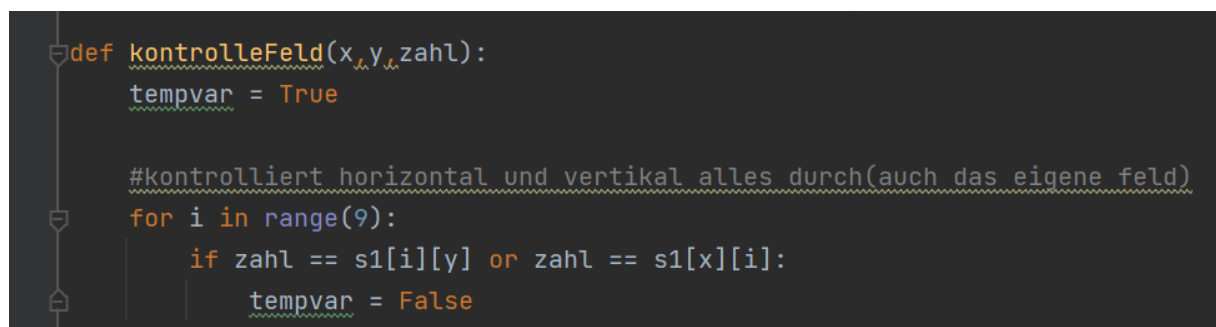
## Funktion kontrolleFeld()

Um ein Sudoku erfolgreich zu lösen, gibt es Bedingungen. Nämlich soll jede Zahl von 1 bis 9 genau ein einziges Mal in jeder Zeile und jeder Spalte auftreten, nicht weniger und auch nicht mehr. Dasselbe gilt dann auch für jedes 3x3 Kästchen. Jede Zahl soll also 1-mal vorhanden sein. Um nun während des Spiels diese Bedingungen zu überprüfen, haben wir die Funktion `kontrolleFeld()` definiert.



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

Abbildung 2



```
def kontrolleFeld(x,y,zahl):  
    tempvar = True  
  
    #kontrolliert horizontal und vertikal alles durch(auch das eigene feld)  
    for i in range(9):  
        if zahl == s1[i][y] or zahl == s1[x][i]:  
            tempvar = False
```

Abbildung 3 kontrolleFeld() I

Ziel des Ganzen ist es, Doppelzählungen zu vermeiden. Das heißt wenn eine Zahl, die bereits im Kästchen existiert, erneut eingegeben wird, ist mit dem Rückgabewert „false“ zu rechnen. Wird

allerdings eine bereits nicht vorhandene Zahl eingegeben, wird das Programm die Funktion weiterführen bis „true“ zurückgegeben wird.

```

81
82     #kontrolliert die blöcke (3x3)
83     if x == 0 or x == 3 or x == 6:
84         if y == 0 or y == 3 or y == 6:
85             if s1[x+1][y+1] == zahl or s1[x+1][y+2] == zahl or s1[x+2][y+1] == zahl or s1[x+2][y+2] == zahl:
86                 tempvar = False

```

Abbildung 4 kontrolleFeld() II

	0	1	2	3	4	5	6	7	8
0	7	1	6	8	0	3	4	2	5
1	5	8	0	6	2	4	3	7	1
2	3	2	4	5	7	1	8	6	0
3	6	5	1	4	8	0	7	3	2
4	8	0	7	1	3	2	6	5	4
5	2	4	3	7	6	5	1	0	8
6	1	3	5	2	4	7	0	8	6
7	4	7	8	0	5	6	2	1	3
8	0	6	2	3	1	8	5	4	7

Um jede der 81 Zahlen kontrollieren zu können, werden jetzt verschachtelte if-Statements eingeführt. Der Kopfzeile hierfür ist in Abbildung 4 zu sehen. Hier werden die vom Benutzer eingegebenen Zahlen für das erste von 9 Kästchen kontrolliert. Angefangen mit den Zahlen 0, 3 und 6 (weil diese Zahlen in der Spalte/Zeile im Raster jeweils als erste Zahl ihres 3x3 Kästchens hervorgehen).

Abbildung 5

Dieser Codeblock wird genau 8 weitere Male wiederholt, allerdings mit anderen Ziffern. Hier ein weiteres Beispiel für das links mittige 3x3 Kästchen:

```

if y == 0 or y == 3 or y == 6: #mitte und links
    if s1[x-1][y+1] == zahl or s1[x-1][y+2] == zahl or s1[x+1][y+1] == zahl or s1[x+1][y+2] == zahl:
        tempvar = False

```

Abbildung 6

## Print Sudoku Box

Die Funktion beginnt damit, dass sie jede Zeile mit einer for-Schleife durchgeht und sie mit den Zahlen 1-9 nummeriert, sowie einen Punkt danach einfügt, hierbei steht i im Code für die Zeilen und z für die Spalten.

```
def printsudokubox2():
    for i in range(9):
        for z in range(9):
            if z == 0: # printet am anfang jeder zeile die Angabe der Zeilennummer
                print(i + 1, end=". ")
```

Im zweiten Abschnitt wird zu Beginn geprüft, ob die eingesetzte Zahl im Sudoku eine Null ist in welchem Fall sie ohne Farbe ausgegeben wird. Anschließend wird mit einem if-Statement kontrolliert, ob die Nummer der Spalte ohne Rest durch 3 teilbar ist. Danach wird durch ein if-Statement kontrolliert, ob die nächste Zahl, die ausgegeben werden soll. Die letzte ihrer Spalte ist und die Nummer der Zeile ohne Rest durch dreigeteilt werden kann, ohne die letzte Zeile zu sein, da wenn dies zutrifft nach der ausgegebenen Zahl eine Linie aus Trennzeichen gezogen wird. Dann wird durch einen elif-Statement kontrolliert, ob man mit z==8 in der letzten Spalte angekommen ist, wenn dies erfüllt wird, wird die letzte Zahl der Zeile ausgegeben und es gibt durch \n einen Zeilenumbruch. Durch die darauffolgenden else Befehle wird nach der Ausgabe jeder dritten Spalte eine Trennwand eingeführt und wenn dies nicht nötig sein sollte die darauffolgende Zahl mit Abstand zur vorherigen ausgegeben.

```
if dataX[i][z] == 0: # wenn die Zahl nicht vorgegeben ist (bei datax mit 0 abgespeichert) dann ohne Farben printen
    if (z + 1) % 3 == 0: # die index von z (spalte) erhöht um 1 um 3. spalten zu filtern mit %3 da ansonten mit 0 1 2 nicht funktionieren würde
        if z == 8 and (i + 1) % 3 == 0 and i != 8: # wenn letzte spalte in der Zeile UND jede 3. zeile UND NICHT allerletzte Zeile dann große Trennwand
            print(s1[i][z], end="\n")
            print("-----+-----+-----")
        elif z == 8: # wenn letzte spalte fängt neue Zeile an (letzte Spalte ist immer index 8)
            print(s1[i][z], end="\n")
        else: # jede 3. Spalte kommt eine kleine Trennwand mit |
            print(s1[i][z], "| ", end="")
    else: # wenn keine Trennwand erforderlich dann Zahl normal mit Abstand zur nächsten printen in der selben Zeile (end="")
        print(s1[i][z], " ", end="")
```

Der letzte Abschnitt der Funktion ist bis auf den Unterschied, dass die Zahlen in der Farbe Grün ausgegeben werden, identisch zum vorherigen Abschnitt. Die Farbe der Zahlen wird hier durch den Befehl \u001b[32m auf grün gesetzt und durch \033[0m wieder entfernt.

```
else: # wenn Zahl vorgegeben ist (bei datax nicht mit 0 abgespeichert) dann mit Farbe printen (HERVORHEBEN)
    if (z + 1) % 3 == 0:
        if z == 8 and (i + 1) % 3 == 0 and i != 8:
            print("\u001b[32m" + str(s1[i][z]) + "\033[0m", end="\n")
            print("-----+-----+-----")
        elif z == 8:
            print("\u001b[32m" + str(s1[i][z]) + "\033[0m", end="\n")
        else:
            print("\u001b[32m" + str(s1[i][z]) + "\033[0m", "| ", end="")
    else:
        print("\u001b[32m" + str(s1[i][z]) + "\033[0m", " ", end="")
```



## Funktion sudokugewonnen()

Wenn ein Spieler durch Lösen des Sudokus oder durch Aufgeben des Gegners gewinnt, wird die Funktion `sudokugewonnen(x)` ausgeführt. Die Funktion führt je nach übergebenem Argument eine print-Meldung über den Gewinner aus.

Wenn das übergebene Argument  $x = 1$  ist dann weiß die Funktion, dass der Spieler 1 gewonnen hat, ansonsten Spieler 2.

```
def sudokugewonnen(x):  
    if x == 1: #wenn Spieler 1 gewonnen hat wird das ausgeführt  
    else: #wenn spieler2 gewonnen hat
```

Abb. 1 & 2

Wenn ein Spieler gewonnen hat, wird sein Spielername mit der vergangenen Zeit in Sekunden in einer Siegermeldung farbig ausgegeben.

```
print("\033[1;96m"+"*****") #Zeit wird nochmal gespeichert und die differenz wird gebildet und in Sekunden ausgegeben sowie Farbig  
print("\033[1;96m"+Spielername,"HAT GEWONNEN und dafür",round(timedelta.total_seconds(zeitmessung_ende_spieler1-zeitmessung_anfang_spieler1)),"Sekunden gebraucht!")  
print("\033[1;96m"+"*****")
```

Abb. 3

Hierbei werden ANSI-Farbcodes benutzt, um die Farbe des Textes im Terminal zu verändern. Man kann mit den Codes Farbe, Dicke, Hintergrundfarbe und Unterstriche hinzufügen. Jedoch muss man beachten, dass man dies wieder zurücksetzen muss wenn man nicht möchte, dass alles was danach in der Konsole steht auch in der Farbeinstellung rauskommt. Hierzu verwendet man den Farbcode „\033[0m“.

Außerdem muss man bevor ein Farbcode benutzt werden kann die Konsole „clearn“ mit dem Befehl `os.system(„cls“)` für Windows oder `os.system(„clear“)` für UNIX.

```
os.system("cls")
```

Abb. 4

## Literatur

<https://de.wikipedia.org/wiki/Sudoku>

<https://github.com/chrischma/ProgrammierenMitChris/blob/master/stopwatch.py>

<https://docs.python.org/3/library/datetime.html>

<https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing>

<https://gist.github.com/vratiu/9780109>

<https://gist.github.com/richardbwest/17674f84961e975d47cf106da9728dd2#file-demo1-py-L30>

<https://de.wikipedia.org/wiki/ANSI-Escapesequenz>

[https://www.youtube.com/watch?v=Srf\\_uSYjKpU](https://www.youtube.com/watch?v=Srf_uSYjKpU)