Aufgabenblatt 5

Das sind die Lösungen zu den Aufgaben der Woche 5 (eigentlich Schulwoche 6 da eine Blockwoche zwischen Woche 4 und 5 liegt)

Aufgabe 21

Implementieren Sie Python-Code, der zwei Listen **liste1** und **liste2** bestehend aus Integer-Zahlen aneinanderfügt und die Ergebnisliste in eine Variable **liste3** speichert. Geben Sie liste3 aus.

Lösung:

Ich habe 3 Lösungen gefunden, es gibt aber noch mehr Lösungen. Alle Lösungen sind auch im Skript „Aufgabe21.py“ im .zip Ordner zu finden.

1. Zusammenfügen mit dem „+“ Operator:

```
20  # Add list1 and list2 together and save the result in list3
21  list3 = list1 + list2
```

Abbildung 1 - Aufgabe 21 a)

2. Die Werte der zweiten Liste mit einer Schleife an die erste Liste hängen

```
25  # Make new list4, because list1 shouldn't be changed
26  # Iterate through list2 and add each value to list4
27  list4 = list1[::]
28  for value in list2:
29      list4.append(value)
```

Abbildung 2 - Aufgabe 21 b)

3. Die Methode „extend()“ verwenden

```
33  # Make new list5, because list1 shouldn't be changed
34  # Use method: "extend" to add Values of list2 to list5 (which has the same values as list1)
35  list5 = list1[::]
36  list5.extend(list2)
```

Abbildung 3 - Aufgabe 21 c)

Info: Ich habe die beiden Listen jeweils zufällig generieren lassen, da ich mich gewundert habe ob dies möglich ist.
Zudem ist mir aufgefallen, dass die Zuweisungen der Listen 4 & 5 nur mit list1[::] funktionieren. Ansonsten wird die Liste 1 verändert.

Aufgabe 22

Liste aus Integer Werten in umgekehrter Reihenfolge in eine neue Variable „listeReversed“ schreiben. Es sollte keine vordefinierte Funktion verwendet werden, sondern eine Schleife.

Lösung:

Meine erste Lösung benutzt keine Schleife, aber auch keine Methode.

1. Die Zuweisung der alten Liste wird mit den eckigen Klammern in den von mir definierten Schritte, also in diesem Fall mit dem Schritt -1, also von hinten nach vorne, was auch dem reversen entspricht (Das ist auch im Skript „Aufgabe22_01.py“ zu sehen):

```
7   import random
8
9   minZahl = 0
10  maxZahl = 100
11  anzZahl = 10
12
13  reverse = -1
14
15  originalList = random.sample(range(minZahl, maxZahl), anzZahl)
16  reversedList = originalList[::reverse]
```

Abbildung 4 - Aufgabe 22 (Semi korrekt)

2. In der zweiten Lösung habe ich die einzelnen Werte aus der Liste gelesen und in die „reversedList“ mit einer „for“ Schleife gespeichert:

```
7   import random
8
9   minZahl = 0
10  maxZahl = 100
11  anzZahl = 10
12
13  originalList = random.sample(range(minZahl, maxZahl), anzZahl)
14  reversedList =  []
15
16  for value in originalList:
17      # Add the Value in front of the list (at the first spot)
18      reversedList = [value] + reversedList
```

Abbildung 5 - Aufgabe 22 mit for Schleife

Info: originale Liste wird zufällig erstellt.

Aufgabe 23

Listen **listeA = [1,3,11,23,4,5]** und **listeB = [4,3,11,23,7,8]** vergleichen:

a) Sind die beiden Listen identisch?
b) An welchen Stellen sind sie identisch? (Liste mit boolean Werten)
c) Sind die Listen gleich lang?
d) Listenelemente in aufsteigender Reihenfolge ausgeben und in eine neue Liste speichern

Lösung:

a) Um die Listen mir ihren Werten zu vergleichen, lässt sich der „=“ Operator verwenden:

```
10  # a) überprüfung der Listen auf Gleichheit
11  if listeA == listeB:
12      print(f"Die Listen: {listeA} und {listeB} sind gleich.")
13  else:
14      print(f"Die Listen: {listeA} und {listeB} sind nicht gleich.")
```

Abbildung 6 - Aufgabe 23 a)

Es werden die Werte und die Positionen (in der Liste) dieser in Betracht gezogen. Deshalb ist kein Loop notwendig.

b) Die Werte in den Listen werden mit einer „for“ Schleife index für index miteinander verglichen:

```
18  # b) Liste mit boolean Werten falls beide Werte in den Listen gleich sind
19  boolList = []
20  if len(listeA) == len(listeB):
21      listLength = len(listeA)
22      for index in range(listLength):
23          boolValue = False
24          if listeA[index] == listeB[index]:
25              boolValue = True
26          boolList.append(boolValue)
27  print(f"Die Werte der Listen: \n{listeA} und \n{listeB} verglichen ergeben: \n{boolList}")
```

```
Anaconda Prompt (anaconda3)
(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_E
lätter\Übung05>python Aufgabe23.py
Die Listen: [1, 3, 11, 23, 4, 5] und [4, 3, 11, 23, 7, 8] sind nicht gleich.
Die Werte der Listen:
[1, 3, 11, 23, 4, 5] und
[4, 3, 11, 23, 7, 8] verglichen ergeben:
[False, True, True, True, False, False]
```

Abbildung 7 - Aufgabe23 b)

Um die Ausgabe zu machen wird ein f String verwendet.

c) Die Länge der beiden Listen wird mit der Methode „len(Liste)“ ausgegeben:

```
30  # c) Länge der beiden Listen
31  if len(listeA) == len(listeB):
32      print(f"Die Länge der Listen ist gleich und beträgt: {len(listeA)}")
33  else:
34      print(f"Die Länge der Listen ist nicht gleich, der Unterschied beträgt: {abs(len(listeA) - len(listeB))}")
```

Abbildung 8 - Aufgabe 23 c)

Wenn die beiden Listen nicht gleichlang sind, wird der Unterschied berechnet und mir der Methode «abs(value)» positiv gemacht, falls der Wert negativ ist.

d) Das Sortieren der Listen und zusammenfügen wurde in zwei Schritten gemacht: Im ersten Schritt werden die Listen geklont, sortiert und ausgegeben. Im zweiten Schritt werden die Listen zusammengefügt und ausgegeben. Ich habe das so gelöst, da ich mir nicht sicher war, wie der Auftrag gemeint war.

```
37  # d) 1) Listen Sortieren
38  listeACopy = listeA[::]
39  listeBCopy = listeB[::]
40  listeACopy.sort()
41  listeBCopy.sort()
42  print(f"Liste A: {listeA} \t\t\t\t----> {listeACopy}")
43  print(f"Liste B: {listeB} \t\t\t\t----> {listeBCopy}")
44
45  # d) 2) listen zusammenfügen und sortieren
46  listeC = listeA + listeB
47  listeC.sort()
48  print(f"Liste C: {listeA + listeB} \t----> {listeC}")
```

Abbildung 9 - Aufgabe 23 d)

Aufgabe 24

Schreiben Sie Python-Code, der die Eingabe einer Integer-Zahl erlaubt, eine Liste der entsprechenden Länge vom Wert der Integer-Zahl generiert und die Liste mit Zufallszahlen füllt, die von Index 0 bis zum grössten Index immer grösser werden. Geben Sie die generierte Liste aus.

Lösung:

Um diese Aufgabe zu lösen habe ich mich entschieden zwei Inputs einzugeben:

1. Länge der Liste als Integer
2. Zufallszahl, damit die sortierte Liste einen definierten Wertebereich erhält (an zufälligen Zahlen)

Gelöst habe ich das mit der „random“ Bibliothek:

```
7   import random
8
9   intInput = int(input("Geben sie eine Ganzzahl ein: "))
10
11  minZahl = 0
12  zufallsZahl = int(input("Wie gross ist der Zufallszahlenbereich? "))
13
14  resultList = random.sample(range(minZahl, zufallsZahl), intInput)
15
16  print(f"Die Zufallsliste unsortiert: \t{resultList}")
17
18  resultList.sort()
19
20  print(f"Die Zufallsliste sortiert: \t{resultList}")
```

```
Geben sie eine Ganzzahl ein: 5
Wie gross ist der Zufallszahlenbereich? 20
Die Zufallsliste unsortiert:    [10, 4, 11, 8, 2]
Die Zufallsliste sortiert:      [2, 4, 8, 10, 11]
```

Abbildung 10 - Aufgabe 24

Um die Zufallsliste zu erstellen habe ich die gleiche Methode verwendet wie bei den vorhergehenden Aufgaben.

Aufgabe 25 – Sudoku

Sudoku ist ein bekanntes Logikrätsel. In der üblichen Version ist es das Ziel, ein 9×9-Gitter mit den Ziffern 1 bis 9 so zu füllen, dass jede Ziffer in jeder Einheit (Spalte, Zeile, Block = 3×3-Unterquadrat) genau einmal vorkommt – und in jedem der 81 Felder exakt eine Ziffer vorkommt. Schreiben Sie Python-Code, der die 81 Felder einer 2D-Liste mit Werten von 1 bis 9 so belegt, dass die oben beschriebene Sudoku-Eigenschaft erreicht wird. (Hinweis: Sie dürfen die Felder zufällig belegen und testen, ob die Eigenschaft erfüllt ist oder Sie belegen Feld für Feld zufällig und nutzen einen Backtracking-Algorithmus.) **(3 P.)** Können Sie das Sudoku im Bildbeispiel unten mit Ihrem Algorithmus lösen?

5	3			7				
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

Abbildung 11 - Bild aus dem Auftrag

Lösung:

Zuerst habe ich die Aufgabenstellung analysiert. Dabei stellt sich heraus, dass es zwei Aufgaben sind:

1. Sudoku erstellen (gegeben ist nichts)
2. Sudoku lösen (Sudoku ist gegeben)

Bei beiden Aufgaben gelten die gleichen Regeln für das Lösen / Erstellen des Sudokus:

1. In der horizontalen dürfen alle Werte von 1 bis 9 jeweils nur einmal erscheinen
2. In der vertikalen dürfen alle Werte von 1 bis 9 jeweils nur einmal erscheinen
3. In den Kacheln von 3x3 dürfen die Werte von 1 bis 9 jeweils nur einmal erscheinen
4. Für Lösen: Es müssen mindestens 17 Werte gegeben sein um eine eindeutige Lösung zu erhalten

Nun wie es im Code gelöst wurde:

1. Alle Funktionen definiert (also die Regeln mit einzelnen Funktionen überprüft)
2. Funktion „Sudoku(grid, startRow, startCol, make)" erstellt
    a. Grid ist das Sudoku Feld
    b. startRow (oder row im Code) ist die Stelle, an welcher gestartet wird
    c. startCol (oder col im Code) ist die Stelle, an welcher gestartet wird
    d. make steht dafür, falls im Sudoku weniger als 17 Werte sind, dann wird nämlich irgendeine Lösung generiert und nicht alle bzw. ein Fehler ausgeworfen, dass nicht 17 Werte gegeben wurden.

CheckNumberOfValues

Diese Methode ist für das Einhalten der 4. Regel zuständig.

Es wird einfach das ganze Board (grid) überprüft und geschaut ob es mehr oder 17 Werte gibt, welche nicht 0 sind:

```
15 def checkNumberOfValues(grid):
16     counter = 0
17     for row in grid:
18         for value in row:
19             if value != 0:
20                 counter += 1
21     if counter >= 17:
22         return True
23     return False
```

Abbildung 12 - Aufgabe 25 Anzahl Werte Check

Um das zu überprüfen wird ein counter verwendet und alle Zahlen, welche über 0 sind, gezählt. Falls dieser Counter am Ende grösser oder gleich 17 ist, dann wird True zurückgegeben, ansonsten False.

CheckValueInRow

Diese Methode überprüft alle Werte in derselben Zeile:

```
27 def checkValueInRow(grid, row, num):
28     for x in range(9):
29         if grid[row][x] == num:
30             return False
31     return True
```

Abbildung 13 - Aufgabe 25 Werte in der Zeile

Dabei wird eine for Schleife verwendet. Ich habe mir überlegt das abhängig vom Sudoku zu erstellen. So könnten auch Sudokus mit mehr als 9 Zeilen und Spalten gelöst und erstellt werden. Schlussendlich habe ich mich dagegen entschieden, da das Testen einer solchen Funktionalität nach Testklasse und Klassen ruft und ich mir nicht zu viel vornehmen wollte.

CheckValueInCol

Diese Methode überprüft die Werte auf Duplikate in derselben Spalte (gleiches Prinzip wie beim Überprüfen der Werte in einer Zeile).

```
35 def checkValueInCol(grid, col, num):
36     for x in range(9):
37         if grid[x][col] == num:
38             return False
39     return True
```

Abbildung 14 - Aufgabe 25 Werte in der Spalte

Hier gilt das gleiche wie bei der Zeile. Der Unterschied zur Zeilenüberprüfung liegt in der Zeile 37. Da werden die Zeilen iteriert, anstatt die Spalten.

CheckValueInBox

Diese Methode überprüft die Werte in einer 3x3 Box.

```python
def checkValueInBox(grid, row, col, num):
    # get the start row of a 3x3 box
    startRow = row - row % 3

    # get the start col of a 3x3 box
    startCol = col - col % 3

    # Checks every Value in the 3x3 box with the input (duplicate check)
    for i in range(3):
        for j in range(3):
            if grid[i + startRow][j + startCol] == num:
                return False
    return True
```

Abbildung 15 - Aufgabe 25 Werte in der Box 3x3

Das Ganze funktioniert indem zuerst ermittelt werden muss, welche Box überprüft werden muss, da es 9 Boxen gibt.

Beispiel:

Wir befinden uns an der Stelle x:

```
1 2 3 | 4 5 6 | 7 8 9
4 5 6 | 7 8 9 | 1 2 3
7 8 9 | 1 2 3 | 4 5 6
------+-------+------
2 1 4 | 3 6 5 | 8 9 7
3 6 5 | 2 1 X | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------+-------+------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
```

Da bei 0 angefangen wird wäre row dann 5 und column wäre 4,

also würde die

- Startzeile: 5 – 5%3 = 5 – 2 = 3
- Startspalte: 4 – 4%3 = 4 – 1 = 3

So wird dann von oben links begonnen und im 3x3 Feld nach unten rechts durchiteriert.

Da wir zweidimensional überprüfen müssen, sind 2 Schleifen notwendig.

CheckRules

Diese Methode führt die 3 Werte-Tester zusammen und gibt True oder False zurück:

```
58  # checks the rules defined at the beginning
59  # returns the methods above (ALL of them have to be true to proceed, since no rules should be broken)
60  def checkRules(grid, row, col, num):
61      return checkValueInRow(grid, row, num) and checkValueInCol(grid, col, num) and checkValueInBox(grid, row, col, num)
```

Abbildung 16 - Aufgabe 25 Regel Check

In dieser Methode werden nur die ersten 3 Regeln überprüft. Warum?

Das geht auf die Performance zurück, denn es kann ganz am Anfang überprüft werden ob das Sudoku wirklich mehr oder 17 Werte besitzt.

Sudoku

In dieser Methode wird das Sudoku gelöst. Wie?

Es werden alle Zahlen von 1 bis 9 probiert bis der check sagt, dass diese Zahl möglich ist und wird dann zur nächsten weiter gehen. Es wird von links nach rechts und von oben nach unten jeder Wert überprüft.

Folgende Schritte (if Bedingungen) werden immer durchgelaufen:

1. Sind es 17 oder mehr Werte bzw. existieren mehrere Lösungen oder nicht?
2. Ist der letzte Wert erreicht? Also row = 8 UND col = 9 (Es ist 8 weil bei 0 begonnen wird und 9 damit die Zahl unten rechts auch überprüft wird)
3. Falls letzte Spalte erreicht wurde, nimm die nächste Zeile (Hier muss nicht auf die letzte Zeile geachtet werden, da dies schon im 2. überprüft wird)
4. Und als letztes wird überprüft ob an jener Stelle schon ein Wert steht (Also kein 0)

```
75  def Suduko(grid, row=0, col=0, make=False):
76
77      # If a Sudoku should be made, then set make, true and the sudoku can be empty (and have multiple solutions)
78      if not make and not checkNumberOfValues(grid):
79          return False
80
81      # Check if last value is reached (bottom right corner)
82      if row == 8 and col == 9:
83          return True
84
85      # Reset at the end of "line" --> next row and reset col to 0
86      if col == 9:
87          row += 1
88          col = 0
89
90      # Search for empty Value (0)
91      if grid[row][col] > 0:
92          return Suduko(grid, row, col + 1, make)
```

Abbildung 17 - Aufgabe 25 Teil 1

Falls nun 17 Werte oder mehr gegeben wurden, keine 0 dasteht und es nicht der letzte Wert oder die letzte Spalte ist, dann beginnt das Spiel wieder von vorne. Warum?

Naja, nicht ganz von vorne, es wird die Spalte um eins erhöht, um den nächsten Wert zu bestimmen.

Fall nun an der Wert 0 ist, werden alle Zahlen von 1 aufsteigend bis 9 ausprobiert.
Das funktioniert etwa so:

1. For Schleife, um von 1 bis 9 zu zählen
2. Überprüfen, ob dieser Wert den Regeln zufolge korrekt ist
3. Falls ja dann den Wert setzen und von vorne beginnen, falls nein nächsten Wert nehmen

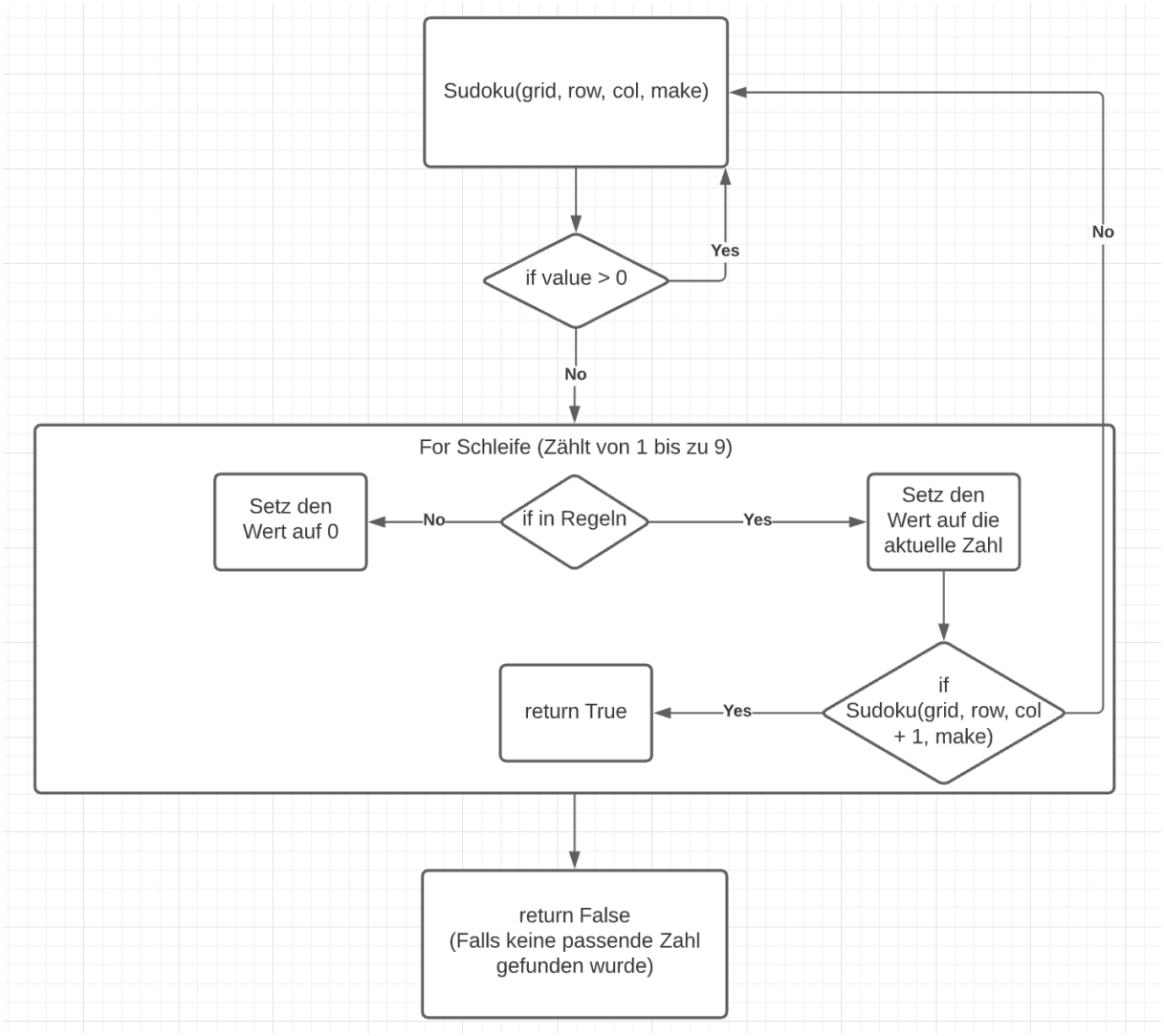

Abbildung 18 - Aufgabe 25 Funktion Sudoku(grid, row, col, make) - Hergestellt mit Lucidchart

Im Bild oben ist der Programmfluss zu sehen. Im Code sieht das so aus:

```
83      # If value is 0 then try all possibilities
84      for num in range(1, 10):
85
86          # Check if the possible Value is in the rules
87          if checkRules(grid, row, col, num):
88
89              # Set at the empty position to possible correct value
90              grid[row][col] = num
91
92              # start process again
93              if Sudoku(grid, row, col + 1, make):
94
95                  # return True --> Value has been found
96                  return True
97
98          # If the set Value is not in the rules then return until value is in the rules
99          grid[row][col] = 0
100
101     # if you come to the last position but didnt return true before, then sudoku is unsolvable
102     return False
```

Abbildung 19 - Aufgabe 25 Teil 2

Wenn man ganz am Ende ankommt ist das Sudoku nicht lösbar.

Verbesserungen:

Es könnten aus dem Lösungsprozess kann vereinfacht werden, indem einmalige Überprüfungen wie das Überprüfen, dass mehr oder 17 Werte im Sudoku enthalten sind oder ob das angegebene Sudoku überhaupt den Regeln entspricht, ausserhalb der Funktion Sudoku(grid, row, col, make) zu überprüfen:

```
118  def solveSudoku(grid=gridDefault, row=0, col=0, make=False):
119      # If a Sudoku should be made, then set make, true and the sudoku can be empty (and have multiple solutions)
120      if not make and not checkNumberOfValues(grid):
121          return False
122
123      # If a Sudoku isn't possible to begin with
124      if not checkRules(grid, row, col, grid[row][col]) and grid[row][col] != 0:
125          return False
126
127      return Sudoku(grid, row, col, make)
```

Abbildung 20 – Verbesserungsvorschlag

Es wäre dann nicht mehr nötig die Variable make mitzunehmen bei jeder Rekursion.

Das ist auch im Skript «Aufgabe25_Verbessert.py» zu sehen.

Ebenfalls habe ich eine Ausgabe der benötigten Zeit in Sekunden, damit der Unterschied zwischen den verschiedenen Varianten des Sudokulösers zu sehen ist:

```
lätter\Übung05>python Aufgabe25.py
Suduko has been solved in 0.63332 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25.py
Suduko has been solved in 0.64569 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25.py
Suduko has been solved in 0.45698 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25.py
Suduko has been solved in 0.88423 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25_Verbessert.py
Suduko has been solved in 0.32008 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25_Verbessert.py
Suduko has been solved in 0.49198 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25_Verbessert.py
Suduko has been solved in 0.42949 seconds

(base) C:\Users\Alex Schneider\Documents\00_FH Chur Graubünden\1. Semester\04_Einführung in die Programmierung\Aufgabenb
lätter\Übung05>python Aufgabe25_Verbessert.py
Suduko has been solved in 0.50353 seconds
```

Abbildung 21 - Geschwindigkeitsunterschied Sudokulöser

Die Verbesserte Variante ist etwa 20% schneller als die alte Lösung.

Bei Fragen oder Unklarheiten BITTE melden ☺ .