Inhaltsverzeichnis

Tipps f. d. Klausur:

- wenn ich weiß, wie oft etw. passieren muss = for-Schleife
- wenn ich ¬ weiß, wie oft etw. passieren muss = while-Schleife
- man könnte solch eine Tabelle machen, damit es nicht so kompliziert wird

Schritt (i)	Key (Element rechts)	Liste vor dem Verschieben	Vergleich im sortierten Teil	Aktion / Ergebnis
1	**1**	[9 | **1**, 2, 7, 3]	9 > 1 ? Ja.	9 schiebt nach rechts, 1 wird vorne eingesetzt: **[1, 9 | 2, 7, 3]**
2	**2**	[1, 9 | **2**, 7, 3]	9 > 2 ? Ja. 1 > 2 ? Nein.	9 schiebt nach rechts, 2 stoppt hinter der 1: **[1, 2, 9 | 7, 3]**
3	**7**	[1, 2, 9 | **7**, 3]	9 > 7 ? Ja. 2 > 7 ? Nein.	9 schiebt nach rechts, 7 stoppt hinter der 2: **[1, 2, 7, 9 | 3]**

Wann // und wann % ?

- `//`:
  → in welcher Kiste bin ich ?
  → Wie viel passt eine gesamte Gruppe irgendwo rein
  → ***Gruppe***
- `%`:
  → Was bleibt übrig
  → ***Rest_Position***

Kommandozeilenargumente

- `import sys` ← immer !!!

Wie ü.gibt man Argumente ?

`<var> = sys.[i]`

```
import sys
```

```
if len(sys.argv) < 3: #guckt, ob ich mind. 3 Elemente habe (Dateiname,
liter, distanz)
    print('Bitte geben Sie Liter und die Distanz an!.')
    sys.exit() #beendet d. Programm sofort

liter = float(sys.argv[1])
distanz = float(sys.argv[2])

verbrauch = (liter *100)/distanz

print(round(verbrauch,2))
```

```
# Terminal
python3 verbracuh.py 45.5 600

# Output
7.58
```

infinet values for max & min

- `float('-inf')`
  → f. max
- `float('inf')`
  → f. min

"Call by Value" vs. "Call by Reference"

- Call by Value (Werte-Kopie):

  → Wie ein Foto v. einem Blatt Papier
  → Wenn ich auf dem Foto herummale, bleibt das Original-Blatt sauber.

  ```
  int, float, str, bool
  ```

- Call by Reference (Adress-Verweis):

  → Wie Google-Doc-Link.
  → Wenn ich ü. den Link etw. im Dokument lösche, dann ist es ∀ anderen, d. den Link haben, auch weg.

  ```
  list, dict, set
  ```

String

Man kann ein Text nicht ändern !!!

```
text = "Was machen Sachen"

text[2] = i

#Augabe: Error
```

- `.upper()` ⟶ *copy*
- `.lower()` ⟶ *copy*
- `.strip()`: *str* ⟶ *str*

  → Entf. Leerzeichen v. Anfang & Ende
- `.split(<elem>)`: *str* ⟶ *list*
- `.replace(<alt>,<neu>)`: *str* ⟶ *str*

```
import string

<var> = string.ascii_lowercase  #abcdefghijklmnopqrstuvwxyz

<var> = string.ascii_uppercase #ABCDEFGHIJKLMNOPQRSTUVWXYZ

<var> = string.ascii_letter
#abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

<var> = string.digits #0123456789

<var> = string.hexdigits #0123456789abcdefABCDEF

<var> = string.whitespace #
```

Arbeiten mir Pfad

```
import os
```

dir = directory (Verzeichnis)

- `<var> = os.getcwd()` ⟶ *str*

  → aktuelles Arbeitsverzeichnis

- `<var> = os.chdir(<pfad>)`

  → wechselt Arbeitsverzeichnis

- `<var> = os.listdir(<pfad>)` $\to *list*

- `<var> = os.isfile(<pfad>)`

  → Ist ü.gebene Pfad eine Datei ?

- `<var> = os.isdir(<pfad>)`

  → Ist ü.gebene Pfad ein Ordner ?

- `<var> = os.path.exists(<pfad>)`

  → exististert Ordner oder Datei ?

<u>Erzeugen löschen & umbenennen</u>:

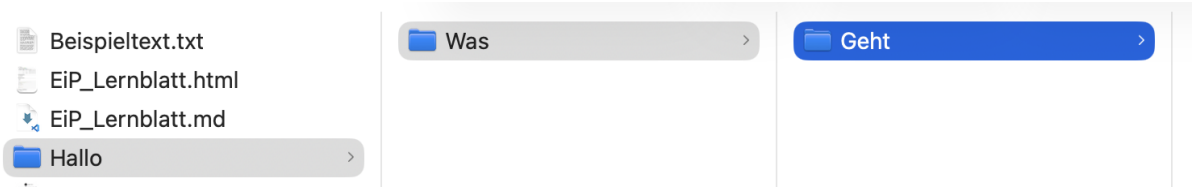
- `os.mkdir(<pfad>/<Ordner_name>)`

  → erzeugt einen Ordner am Pfad
  → `os.mkdir('/Volumes/Efes USB/Uni/Semester 6/EiP/Übung für mich selbst/Hallo')`
  → pfad: /Volumes/Efes USB/Uni/Semester 6/EiP/Übung für mich selbst
  → ordner: /Hallo

- `os.makedirs('/Volumes/Efes USB/Uni/Semester 6/EiP/Übung für mich selbst/Hallo/Was/Geht')`

  → es wird eine Kette an Ordner erstellt

  →



- `os.rmdir(<pfad>)`

  → Ordner löschen

- `os.remove(<pfad>)`

  → Datei löschen

- `os.replace(<alt_name>, <neu_name>)`

  → Name ändern

Listen

- mit Input Liste generieren lassen

```
such_input = input("Zahlen für Suchmuster (getrennt durch Leerzeichen,
z.B. '1 3 1'): ")

mein_muster = [int(x) for x in such_input.split()]
```

Werkzeuge:

- `<liste>.clear()`:

  → in-place

- `<list>.count(<elem>)`:

  → zählen des Elem.

- `<list1>.extend(<liste2>)`:

  → in-place
  → Die Liste wird ausgepackt & hinzugefügt ?

- `<list>.index(<elem>, start, end)`:

  → gibt Index v. Objekt

- `<list>.insert(<elem>)`:

  → in-place

- `<list>.reverse()`:

  → in-place
  → `<list>[::-1]` = Kopie

- `<list>.sort()`:

  → in-place Sortierung
  → sorted() = Kopie

- `.pop(<index>)`:

  → dann wird das Element an dieser stelle gelöscht

- `.remove()`:

  → wenn das erste Element, der dem Element in `.remove(<elem>)` entspricht, wird entf.

- `.copy()`:

  → **wichtig !!!!**
  → man kopirt nur d. ***oberste Ebene***

```
liste_a = [10, 20, 30]
liste_b = liste_a

liste_b[0] = 2
```

```
#Ausgabe:
Originale Listen:
--------------------------------
Lista A:  [10, 20, 30]
Liste B: [10, 20, 30]

V1 Listen:
--------------------------------
Liste_a: [2, 20, 30]
Liste_b: [2, 20, 30]
ID von A: 1834691293696
ID von B: 1834691293696
True
```

→ **Anw. des ersten .copy():**

```
liste_a = [10, 20, 30]
liste_b = liste_a.copy()

liste_b[0] = 2
```

```
Originale Listen:
Lista A:  [10, 20, 30]
Liste B: [10, 20, 30]

V1 Listen:
Liste_a: [10, 20, 30]
Liste_b: [2, 20, 30]
ID von A: 2152350446080
ID von B: 2152352749696

False
```

→ **echte unabhängige copy ?**

```
import copy

liste_a = [10, 20, 30]
liste_b = copy.deepcopy(liste_a)
```

```
liste_b[0] = 2
```

```
#Ausgabe:
Originale Listen:
Lista A:  [[1], [3, 8, 8], [9, 6], 7]
Liste B: [[1], [3, 8, 8], [9, 6], 7]

V1 Listen:
ID "OG-Conteiner": 1667841546688
ID "Copy-Conteiner": 1667839716544

False
```

Wann .append und wann += ?:

- `.append()`:
  → Wenn ich ein Objekt mit den Klammern hinzufügen möchte
    → Bsp.: Ich lege d. gesamte tüte mit den Elementen in den Wagen
  → wenn ich nur den Inhalt hinzufügen möchte
    → Bsp.: Ich lege nur d. Ware in den Wagen & schmeiße die leere Tüte weg

```
haupt_liste = [1, 2]
zusatz = [3, 4]

# Szenario A: append (Das Objekt als Ganzes)
liste_a = [1, 2]
liste_a.append(zusatz)
print(liste_a)
# Ergebnis: [1, 2, [3, 4]]  <-- Liste in der Liste

# Szenario B: += (Nur der Inhalt)
liste_b = [1, 2]
liste_b += zusatz
print(liste_b)
# Ergebnis: [1, 2, 3, 4]    <-- Alles auf einer Ebene
```

Wie werden mehrdimensionale Listen sortiert ?

- Es wird anhand des ersten Elements sortiert

```
[11, 7, 9, 8]
[16, 2, 8, 16]
[4, 5, 12, 20]
[10, 17, 2, 12]
------------------------------------------------------------
```

```
[4, 5, 12, 20] 4
[10, 17, 2, 12] 10
[11, 7, 9, 8] 11
[16, 2, 8, 16] 16

#größere Liste
[16, 0, 19, 7, 3, 18, 2]
[8, 15, 9, 1, 0, 4, 7]
[3, 14, 10, 19, 20, 20, 5]
[11, 1, 17, 11, 1, 7, 8]
[2, 17, 12, 17, 12, 20, 3]
[2, 16, 16, 18, 7, 4, 0]
[14, 12, 1, 6, 13, 8, 20]
------------------------------------------------------------
[2, 16, 16, 18, 7, 4, 0] 2
[2, 17, 12, 17, 12, 20, 3] 2
[3, 14, 10, 19, 20, 20, 5] 3
[8, 15, 9, 1, 0, 4, 7] 8
[11, 1, 17, 11, 1, 7, 8] 11
[14, 12, 1, 6, 13, 8, 20] 14
[16, 0, 19, 7, 3, 18, 2] 16
```

Wie verändert sich meine Liste, wenn ich mit nested-for_schleifen arbeite ?

- Genau die Stelle d. ich verändern möchte, muss ich mit einem Index machen:

```
lst = [[
    [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []]
],
[   [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []]
],
[
    [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []],
    [[], [], [], []]
]]

#Ich möchte d. 4. Ebene ändern = 4. Ebene = Index

for i in lst: #lst = 1.Ebene, i = 2.Ebene
    for j in i: # 3.Ebene
        for k in range(len(j)): #4.Ebene
            j[k] = 2
```

Wie kann ich autom. ein Spielbrett durch gehen & Werte eins. ?

- Def. eine Var. relation aufschrieben, wenn mehrere Var. f. d. Gleiche stehen:
  → y = Zeile = m
  → x = Spalte = n
- Wir müssen d. Spielfeld erst einmal bauen:

```
n = 5
m = 5

spielfeld = [[[]for _ in range(n)]for _ in range(m)]

#spielfeld = [
# [[], [], [], [], []],
# [[], [], [], [], []],
# [[], [], [], [], []],
# [[], [], [], [], []],
# [[], [], [], [], []]
# ]
```

- Zufälliges Verteilen v. Bomben:

```
möglichekeiten = ['', '*']

for zeile in spielfeld:
    for spalte in range(len(zeile)):
        ausgw_elem = r.choices(möglichekeiten, weights=[60,40], k=1)
        zeile[spalte] = ausgw_elem[0]

#spielfeld = [
#   ['*', '', '', '*', '*'],
#   ['*', '', '*', '*', ''],
#   ['*', '', '', '', ''],
#   ['*', '', '', '*', '*'],
#   ['', '*', '*', '*', '']
# ]
```

- Wir zählen d. Nachbarn

```
max_n = n-1
max_m = m-1

for zeile in range(m):
    for spalte in range(n):

        #Ist d. Feld ü.haupt eine Bombe ? Dann brauchen wir d. Gucken
d. N.barn nicht
        if spielfeld[zeile][spalte] == '*':
```

```
            continue

        bomben_count = 0

        #gucken d. Umgebung:
        for y in range(-1,2):
            for x in range(-1,2):

                #Sicherstellen, dass wir nicht auf d. [-1] Element
zugreifen:
                y_now = zeile + y
                x_now = spalte + x


                #Wenn nein, dann gucken, ob es p.haupt in meinem
aufgespannten Brett ist
                if 0 <= y_now <= max_m and 0 <= x_now <= max_n: #wenn
ja, dann zählen wir d. Umgebung
                    if spielfeld[y_now][x_now] == '*':
                        bomben_count += 1
        spielfeld[zeile][spalte] = bomben_count
```

Dictionaries

Wie sortieren ?:

- `sorted_indices = sorted(häufigkeiten.keys())`

```
my_dict = {'a': 2, 'c': 1, 'b': 4, 'd': 3}
sortierte_liste_tupel = sorted(my_dict.keys()) # Ausgabe: {'a': 1,
'b': 2, 'c': 3, 'd': 4}
```

Wie Elemente hinzufügen:

- `dic.upadte()`

```
dic = {}
n = 7
k = 3

dic.update({(n,k) : 45})
print(dic)
```

Mit Text arbeiten

Man muss es sich so vorstellen, dass `.readlines()` unsere Augen sind & wir jede Zeile lesen ! Das heißt, wenn ich etw. hinzufüge, dass ich dann mit den Augen unter d. letzen Zeile bin. Wenn jmd. jzt. fragen würde, was da steht, dann würde ich "nichts" sagen, weil da nichts geschrieben steht. Damit ich lesen kann, muss ich wieder mit den Augen zur ersten Zeile, welches hier mit `.seek({index})` geschieht. ES IST GANZ WICHTIG ZU WISSEN, DASS `.split()`, `.strip()`, usw. EINEN STRING ZURÜCK GEBEN !!!

Also wenn ich in der Klausur eine Aufgabe bekommen sollte, welches mit einer Datei ist, dann muss ich als ersten gucken, ob ich den gesamten Inhalt als Ganzes brauche, oder ob ich es auch zeilenweise angucken kann.

Wenn ich eine „Textanalyse“ machen muss, also den gesamten Text vor mir gedruckt haben möchte, damit ich da zum Bsp Wörter oder Buchstaben „korrigieren“ möchte, dann benutze ich readlines(). Im Computer ist d. gedruckte Blatt dann eine Liste mit den einzelnen Elementen als String mit \n & \t.

Wenn ich aber zeilenweise abschreiben möchte, v. einem Buch in meinen Heft, dann muss ich `.readline()` verwenden. Wichtig hierbei ist, dass `mein Auge immer einen Zeile weiter geht !` + ich muss dem Computer sagen, dass es etwas `n Mal` machen soll, weil er dann etwas z.B. nur 1 mal macht

Allgemein:

- `r` = read
- `w` = write
- `a` = append
- **Wie kombinieren?**

  → bestehende Datei ändern möchte ⟶ `'r+'`
  → neue Datei erstellen & konrollieren möchte ⟶ `'w+'`
  → an bestehende Datei etw. anhängen möchte ⟶ `'a+'`

Wie Datei öffnen ?:

```
with open("Probe.txt", 'r') as data:
    print(data)
```

- somit benötige ich kein `var = close()` am Ende

.readlines():

```
with open("Probe.txt", 'r') as data:
    f_inhalt = data.readlines()
    print(f_inhalt)
```

- das nimmt den Inhalt und packt es in eine Liste, wobei jedes Element des Textest ein Element in der gesamten readline Liste ist
- somit wird mir eine Liste aus den Elementen ausgegeben
- Textdatei

```
    # Ausgabe
    ['1) Papa \n', '2) Mama \n', '3) Efe \n', '4) Sude \n', '5) Simge \n',
'6) Pepee \n', '7) Mayla ']
```

.readline()

.readline() ist so, als ob wir ein Buch haben und ein Heft. Unser Auge guckt im Buch n. d. ersten Zeile und wir schreiben d.as in unser Heft ab. Autom., wenn wir ins Buch gucken, beginnen wir mit d. 2. Zeile & d. schreiben wir ebenso in ins Heft. usw. Da wir dem Computer alles definieren müssen, müssen wir sagen wie viele solcher Prozesse wir durchmachen möchten, welches hier = 66 mal ist. D.h., dass wir 66 × d. Buch lesen & abschreiben

- *Übungsblatt 6, Nr.4*
- hierbei wird nur eine Zeile gelesen

```
with open("Probe.txt", 'r') as data:
    f_inhalt = data.readline()
    print(f_inhalt)
```

```
# Augabe
1) Papa
```

- wenn ich aber `print(f_inhalt, end=' ')` mache, dann wird es ohne den Absatz ausgegeben

```
# Augabe
1) Papa
```

Wie đ Zeilen iterieren ?:

- đ d. Iterieren, wird ein Memory-Problem umgangen !</span

```
with open("Probe.txt", 'r') as data:
    for line in data:
        print(line, end='')
```

```
#Ausgabe
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
```

Wie Inhalt zu d. Datei hinzufügen ?:

- Text2

```
with open("Probe.txt", 'a+') as data:

    mayla = '\n7) Mayla'

    # 1. Schreiben/Anhängen
    data.write(mayla)

    # 2. Zeiger zurücksetzen auf den Anfang (Index 0)
    data.seek(0)

    # 3. Lesen der gesamten Datei ab Index 0
    inhalt = data.read()

    # 4. Ausgabe des gelesenen Inhalts
    print(inhalt)
```

```
# Veränderte Textdatei:
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
7) Mayla
```

- Wie kann man denn direkt einen kompletten langen Text hinzufügen ?

```
with open("Probe.txt", 'a+') as data:

    #String schrieben
    mayla = str(input(f"Was soll hinzugefügt werden: "))
```

```
    #String zur Datei hinzufügen
    data.write('\n' + mayla)

    #Position in der Datei wieder auf die erste zeile setzen
    data.seek(0)

    lines = data.readlines()

    for line in lines:
        print(line, end="")
```

```
# Augabe
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
7) Mayla
```

- w+

```
with open("Probe.txt", 'w+') as data:

    #String schrieben
    mayla = str(input(f"Was soll hinzugefügt werden: "))

    #String zur Datei hinzufügen
    data.write(mayla)

    #Position in der Datei wieder auf die erste zeile setzen
    data.seek(0)

    lines = data.readlines()

    for line in lines:
        print(line, end="")
```

```
# Ausgabe
Hallo was geht ?
# man sieht alos, dass d. gesamte Inhalt gelöscht wurde
```

Wie kann man denn gezielt leere Zeilen entfernen ?:

```
with open("Probe.txt", 'a+') as data:

    data.seek(0)

    lines = data.readlines()

    for line in lines:
        if line.strip() == '':
            continue

        print(line, end="")
```

- Somit sagen wir also, wenn wir an eine leere Zeile stoßen, dann einfach ignorieren & mit d. nächsten Zeile weiter machen zu lesen ?

```
# Text davor

1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
7) Mayla

# Text danach:
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
7) Mayla
```

Wie kann ich denn machen, dass ich jede dritte Zeile oder 4. Zeile lesen möchte ?:

- %

Wie kann man einfach eine neue Datei erstellen ?:

```

```

Wie kontrollieren, ob ene Datei bereits existiert ?:

```

```

```
import os

os.path.exists(<pfad>) -> bool
```

Was ist ein optionaler Parameter ?:

- ein optinaler Parameter ist ein Parameter, dass immer autom, f, eine var, gegesetzt wird, falls hier ichts eingesetzt wird

```
def begruessung(name="Mensch"):
    print("Hallo " + name)

begruessung()
begruessung("Lisa")
```

```
#output
"Hallo Mensch"
"Hallo Lisa"
```

Wie kann man einzelne Zeichen zählen ?:

- wir verw. .count()

```
def letterCount(text, zeichen=string.ascii_letters):
    count_collection = []

    for letter in zeichen:
        count = text.count(letter)
        count_collection.append(count)

    return count_collection

text = str(input('String: ')) #"Was machen Sachen ?"
zeichen = str(input('Zeichen: ')) #'ach'

print(letterCount(text, zeichen))
```

```
#Output
[3, 2, 2]
```

- $3 \times a,\ 2 \times c,\ 2 \times h$

Rekursion

So kann mich sich die Rekursion auch vorstellen.

Bsp.: Fibonacci-Zahlen

Wenn du wissen willst, was der Wert von $F(5)$ ist, gehst du zu Person 5. Da diese d. Antwort ¬ auswendig weiß, ruft sie sofort bei ihren Kollegen in Büro 4 und Büro 3 an und bittet sie um Hilfe. D. Problem bei d. normalen Rekursion ist, dass diese Kollegen wiederum ihre Nachbarn anrufen müssen, was zu unzähligen Telefonaten führt, bei denen dieselben Fragen immer & immer wieder gestellt werden.

Hier kommt unsere Schachtel 📦 ins Spiel, d. im Flur ∀ erreichbar steht: das Memo-Dictionary. Bevor eine Person zum Hörer greift, schaut sie erst in diese Schachtel. Liegt dort schon ein Zettel mit dem Ergebnis f. ihr Büro, nimmt sie den Wert einfach heraus & gibt ihn sofort an den Fragesteller weiter. Ist d. Schachtel leer, müssen die Kollegen angerufen werden. Die Einzigen, d. niemals telefonieren müssen, sind d. Personen in Büro 0 und Büro 1. Sie kennen ihre Werte (0 und 1) sofort, legen sie als Erste in d. Schachtel und starten damit d. Kettenreaktion. Sobald eine Person d. Antw. ihrer beiden Kollegen erhalten hat, addiert sie diese, schreibt das neue Ergebnis auf einen Zettel für die Schachtel und liefert die Antwort ab. So wird jede Zahl im Gebäude nur ein einziges Mal wirklich berechnet. ✨

```
def querSumme(n:str, result:int)->int:
    if len(n) == 0:
        return result
    else:
        b = int(n[0])
        result += b
        return querSumme(n[1:],result)


n = str(input("n: "))
n_lst = []
for i in n:
    n_lst.append(i)
result = 0
print(querSumme(n_lst,result))
```

oder

```
def quersumme(n: int) -> int:
    if n < 10:
        return n

    return (n % 10) + quersumme(n // 10)
```

Dreiecke

1) Fläche deines Dreiecks begrenzen:

- Präsenzblatt
- Was wir berechnen
- Also $\text{len}(\text{Höhe}_{\text{absolut}})$ & je n. dem wo wir uns in dem absoluten intervall befinden, wird d. Breite demetsprechend berechnet.
- $\text{H}_{\text{absolut}} \in [1, 4]$

  → wobei $\text{H}_{\text{relativ}} \in [0, 3]$, weil Länge = $4 - 1 = 3$ (Gesamtlänge)
- Also, wir haben jzt. d. Basis B, aber weil wir aber d. Hälfte d. Basis benötigen = $\frac{\text{B}}{2}$
- d. absolute Höhe H_{abs} & autom. auch H_{rel}
- Die $\text{Gesamthöhe} \in [2.5, 5.5]$ & $\text{H}_{\text{relativ}}$ bewegt sich im Bereich $5.5 - 2.5 = 3 \implies \text{H}_{\text{relativ}} \in [0, 3]$

  1. random Höhe wählen
  2. H_{rel} bestimmen
  3. B bestimmen
  4. x koordinaten Wählen: $\text{r. choice}[\text{x}_{\text{mid}} - \text{w}, \text{x}_{\text{mid}} + \text{w}]$.
- Rampe $\text{B} = 2$ & $\text{H}_{\text{abs}} = 1 \longrightarrow$ Verhätnis zwischen Höhe zur Länge = $\frac{\text{Höhe}}{\text{Länge}}$

  → 100 Menschen ingesamt

  → 34 = krank

  → $34/100 = 0.34 = 34$

  → Das Verhältnis zw. den Menschen & den Kranken beträgt 34

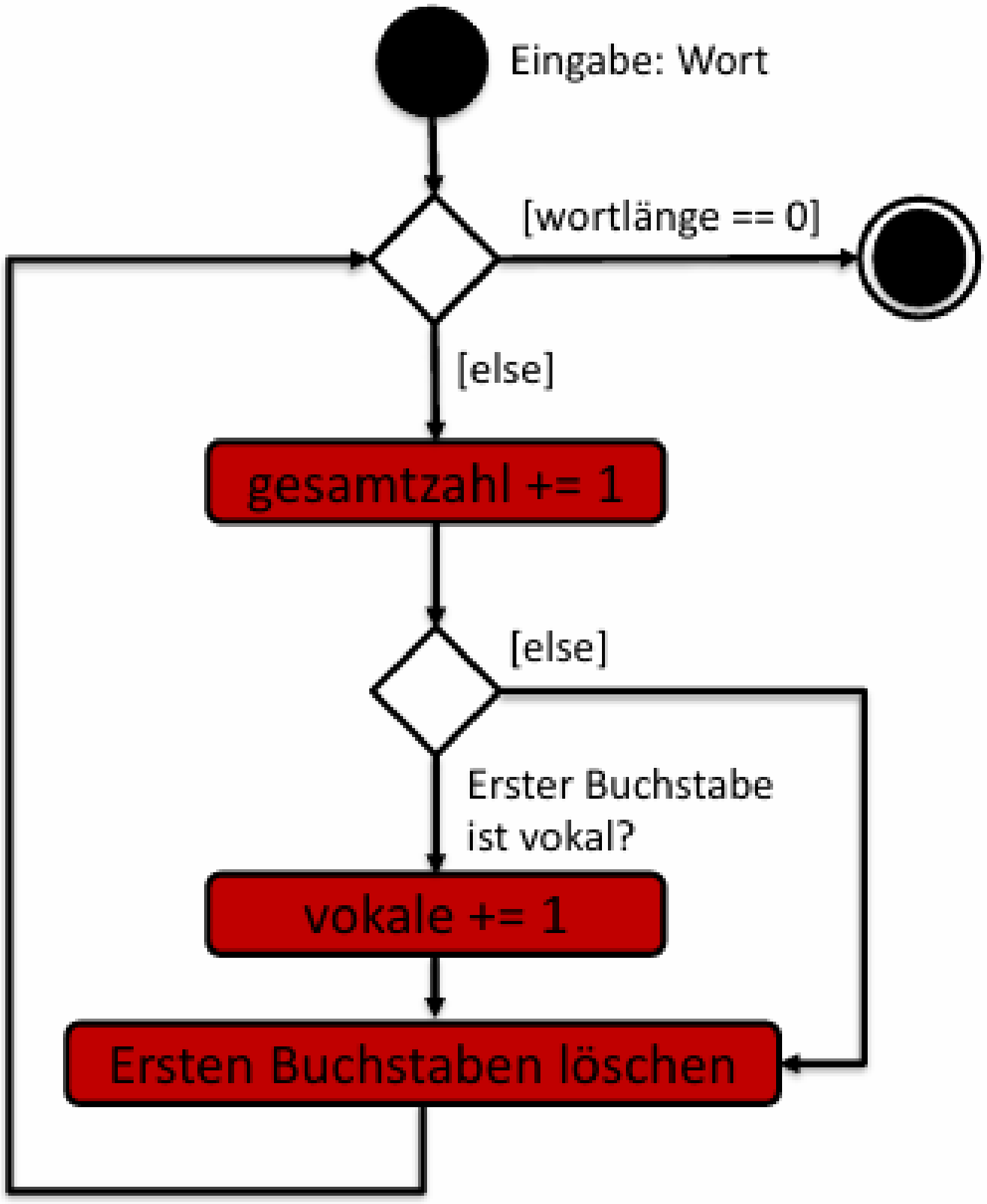
  → wir wollen wissen, wie das Verhältnis zwischen d. Breite & der relativen Höhe ist $= \frac{\frac{\text{B}}{2}}{\text{H}} = \frac{\text{B}}{2\text{H}}$

Tests

Allgemein:

1. Anweisungsü.deckung: C_0
2. Zweigsü.deckung: C_1
3. Pfadü.deckung: $\text{C}_2\text{a}, \text{C}_2\text{b}, \text{C}_2\text{c}$
4. Bedingungsü.deckung: $\text{C}_3\text{a}, \text{C}_3\text{b}, \text{C}_3\text{c}$

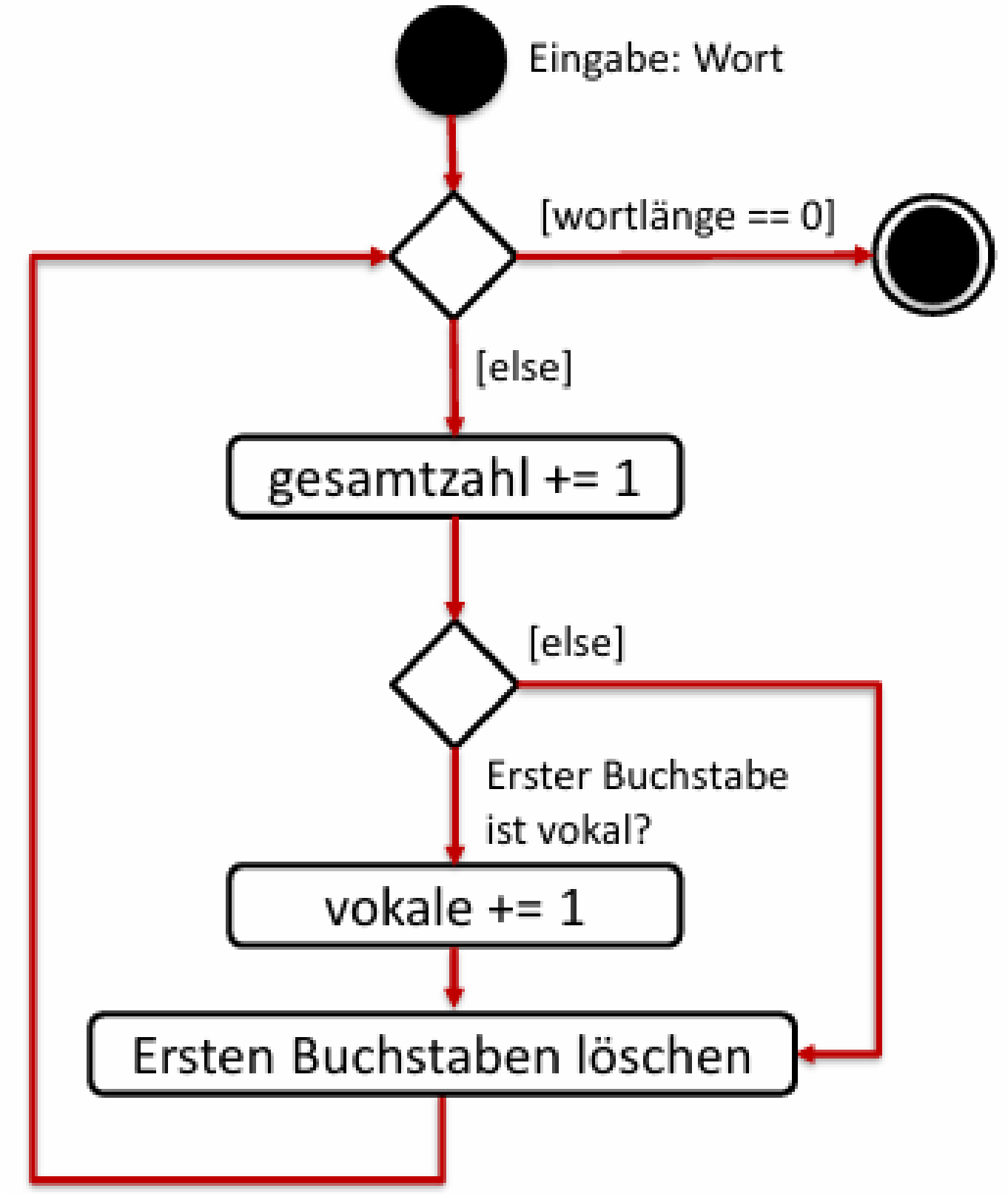
Was ist Anweisungsü.deckung: C_0:




```
def vokal_counter(wort:str)-> tuple:
    gesamtzahl = 0
    vokale = 0

    while len(wort) > 0:
        gesamtzahl += 1

        if wort[0] in "aeiou":
```

```
            vokale += 1

        wort = wort[1:]

    return gesamtzahl, vokale


wort = str(input("Wort: ")).lower().replace(" ","")
print(wort)
print(vokal_counter(wort))
```

Was ist Zweigü.deckung: C_1:




```
def c1_test()->bool:
    return vokal_counter('ab') == (2,1)
```

Was ist Pfadü.deckung: C_2a:

Sortieralgorithmen

2 Insertion Sort (**)

Ein Algorithmus zum Sortieren ist InsertionSort. Hierbei durchläuft eine Markierung das zu sortierende Feld von links nach rechts, wobei der Bereich links der Markierung bereits sortiert ist. In jedem Schritt wird dazu das Element das rechts neben der Markierung steht in die bereits sortierten Elemente links davon an der richtigen Position eingeordnet. Das Verfahren endet wenn die Markierung an der rechten Seite des Feldes angelangt ist.

Beispiel: (Der Strich | stellt die Markierung dar)

```
| 9 1 2 7 3
9 | 1 2 7 3
1 9 | 2 7 3
1 2 9 | 7 3
1 2 7 9 | 3
1 2 3 7 9 |
```

Lösung:

```python
def insertion_sort(lst: list[int])-> list[int]:
    if len(lst) <= 1:
        return lst
    else:
        for i in range(1,len(lst)):
            j = i-1

            key = lst[i]
            while j >= 0 and lst[j] > key:
                lst[j+1] = lst[j]
                j -=1

            lst[j+1] = key

    return lst


lst = [8, 4, 3, 9, 7, 1, 4, 2, 4, 7]
print(lst)
print(insertion_sort(lst))
```

- **Wrm. eine for Schleife ?**
    1. Muster erkennen:
        → es gibt immer einen Trenner |
        → links v. Trenner \underrightarrow{\ \ \ \ \textcolor{#d6b315}{\text{im jeden Schritt}}\ \ \ \ } länger
            → Wir brauchen eine Schleife, d. den Trenner v. vorne bis hinten schiebt.

random

Wie kann ich `r.choice()` verw. und denen eine Wahrscheinlichkeit geben ?

- `r.choices(<lst>, <wahrscheinlichkeit 1>, <wahrscheinlichkeit 2>, ...) -> list`

    → Beispiel 1:

```
seq = [[1, 3, 5]]
wahrscheinlichkeit = r.choices(seq,weights = [20, 60, 20], k = 1)

# Jedem Element wird automatisch d. Wahrscheinlichkeit ü.geb., welches
d. gleiche Index hat:
1 = 20%
3 = 60%
5 = 20%
```

k = die Anzahl der Drehung

→ Wenn k = 3 to 3 × gedreht
→ ***Bsp***: [['', '', '']]
→ Wenn k = 1 to 1 × gedreht
→ ***Bsp***:['*']

Storage

```
#Probe.txt
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
7) Mayla
```

```
#Probe.txt
1) Papa
2) Mama
3) Efe
4) Sude
5) Simge
6) Pepee
```

random Aufgaben

```
#wir müssen uns von innen nach außen arbeiten
def mehrdim_liste_sortieren(lst:list[int]) -> list[int]:
    for zeile in lst:
        for n in range(len(zeile)-1):
            min_elem = min(zeile[n:])
            min_elem_index = zeile.index(min_elem,n)
            zeile[n],zeile[min_elem_index] = zeile[min_elem_index],
zeile[n]

    for i in range(len(lst)-1):
        min_zeile = min(lst[i:])
        index_min_zeile = lst.index(min_zeile,i)
        lst[i], lst[index_min_zeile] = lst[index_min_zeile],lst[i]


    return lst



lst_größe = int(input("Listengröße: "))
lst = [[r.randint(0,20) for _ in range(lst_größe)]for _ in
range(lst_größe)]
print(lst)

#erstellen einer identischen, unabhängigen copy d. unarbeiteten Liste
lst1 = copy.deepcopy(lst)

#Ausführen meiner Liste
i_sorted_lst = mehrdim_liste_sortieren(lst)
print(f'Mein Ergebnis: {i_sorted_lst}')

#test

for zeile in lst1:
    zeile.sort()

lst1.sort()
print(f'Wie es sein sollte: {lst1}')
```

```
if lst1 == i_sorted_lst:
    print(True)
else:
    print(False)
```

Aufgabe 2: Primzahltest (10 Punkte)

Schreiben Sie eine Python-Funktion, die testet, ob eine natürliche Zahl n eine Primzahl ist (also nur durch eins und sich selbst teilbar ist). Eine richtige Lösung erhält 8 Punkte. Lösungen, deren Laufzeit asymptotisch schwächer als linear in n wächst (formal: $\lim_{n\to\infty} \text{Laufzeit}(n)/n = 0$), erhalten zwei zusätzliche Punkte. Dabei soll in 1-2 Sätzen *informell* begründet werden, warum die Laufzeit sich so verhält und warum der Algorithmus (trotzdem) funktioniert (1 Punkt entfällt auf die Begründung).

Lösung:

```
def is_prime_number(n: int) -> bool: # Rückgabe „True" für Primzahlen
```

- **Def. d. Laufzeit**

1. Algorithmus $\underrightarrow{\textcolor{#d6b315}{\text{besteht}}}$ eine Schleife
   - v. 2 bis $\sqrt{n}$
2. Anzahl d. Schrittzahlen: $\sqrt{n} - 2$ weil 1 und 2 ¬ enthalten
3. $\lim_{n\to\infty} = \sqrt{n}$

- **Bew. d. Ü.legenheit**:

$$\lim_{n\to\infty} \frac{\sqrt{n}}{n}$$

$$\lim_{n\to\infty} \frac{n^{0.5}}{n^{1}}$$

$$\lim_{n\to\infty} n^{0.5-1} = \lim_{n\to\infty} n^{-0.5}$$

$$\lim_{n\to\infty} \frac{1}{n^{0.5}} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$$

- **Logische Rechtfertigung**:

1. Jeder Teiler d einer Zahl n hat einen Partner-Teiler d', sodass $d \cdot d' = n$ gilt.
2. Wären beide Teiler $>$ als $\sqrt{n}$, wäre ihr Produkt $d \cdot d'$ automatisch $>$ als n ($\sqrt{n} \cdot \sqrt{n} = n$).
3. Mindestens einer d. beiden Teiler muss $\leq \sqrt{n}$ sein. Finden wir bis $\sqrt{n}$ keine Zahl, d. n ohne Rest teilt, kann es auch danach keine geben.

```
import math

def is_prime_number(n: int) -> bool:


    while n<0:
        print("n muss positiv sein!")
```

```
        n = int(input("n: "))

    if n < 2: return False #Weil 1 keine Primzahl.

    # Wir prüfen nur bis zur Wurzel von n!
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True


n = int(input("n: "))
print(is_prime_number(n))
```

Aufgabe 1: Zum Aufwärmen – ein einfacher inkrementeller Algorithmus (7+8=15 Punkte)

Gegeben Sei ein Array (Python-Liste, Typ **`list`**) mit von ganzen Zahlen (typ **`int`**). In dieser Aufgabe soll eine Python-Funktion implementiert werden, die die Summe der in der Liste enthaltenen Zahlen berechnet. Das Problem soll auf zwei verschiedene Arten gelöst werden (wichtig – andere Ansätze geben jeweils keine Punkte).

Achtung: Die eingebaute Python-Funktion „**`sum()`**" ist in dieser Aufgabe nicht erlaubt! Sie müssen die Iteration auf jeden Fall selbst programmieren, sonst gibt es keine Punkte.

a) Lösen Sie die Aufgabe mit einer Schleife, die die Liste durchläuft, den Inhalt zusammenaddiert, und am Ende mit **`return`** zurückgibt.

```
# Definition der Funktion (Sie können gerne zusätzliche Funktionen definieren).
def sum_iterative(numbers: List[int]) -> int:
```

b) Lösen Sie die gleiche Aufgabe rekursiv, also *ohne Schleifen zu benutzen*. Die Rekursion soll die Elemente der Liste durchlaufen und die Summe bilden.

```
# Definition der Funktion (Sie können gerne zusätzliche Funktionen definieren).
def sum_iterative(numbers: List[int]) -> int:
```

```
import random as r
```

```
def sum_iterativ(numbers:list[int])->int:
    if len(numbers) == 0:
        return 0

    else:
        summe = 0

        for i in numbers:
            summe += i
        return summe


def sum_rekursive(numbers: list[int])->int:

    if len(numbers) == 0:
        return 0
    else:
        if len(numbers) == 1:
            return numbers[0]

        return numbers[0] + sum_rekursive(numbers[1:])


numbers = [r.randint(0,20)for _ in range(3)]
print(numbers)
print(sum_iterativ(numbers))
print(sum_rekursive(numbers))
```

- number = [5,8,3]
- **Schritt 1)** 5 + summe[8,3]

![alt text](image-6.png)

```
import random as r

def berechnen_von_feld_B(n: int, feld_A: list[int])-> list[int]:

    #wir erstellen zunächst das Feld_B
    feld_B = [[[]for _ in range(n-2)] for _ in range(n-2)]

    #Koordinaten für feld_B

    for i in range(1,n-1):
        for j in range(1,n-1):

            summe = 0
            for y in range(-1,2):
                for x in range(-1,2):
                    ni = i+y
                    nj = j+x
```

```
                    summe += feld_A[ni][nj]

            feld_B[i-1][j-1] = summe
    return feld_B



n = int(input('n: '))
feld_A = [[r.randint(1,5)for _ in range(n)] for _ in range(n)]
print(f'Feld A: {feld_A}')
feld_B = berechnen_von_feld_B(n,feld_A)
print(f'Feld B: {feld_B}')

print(20* '-')


for i in feld_A:
    print(*i)

for j in feld_B:
    print(*j)
```

BIld hinzufügen

import random as r

def num_in_bereich(num:int)->int: while 0 > num or num > 3999: print("Die Zahl muss zwischen 1 und 3999 sein!") num = int(input("num: ")) return num

def umwandlung_römische_zahl(num:int)->str: num = num_in_bereich(num) römische_zahl = ""

```
symbole = [['M'], ['C','D', 'M'], ['X','L', 'C'], ['I','V','X']]

einheiten = [1000,100,10,1]
n_num = num
index = 0

for reihe in einheiten:
    rest = n_num % reihe
    ziffer = (n_num - rest) // reihe
    n_num = rest

    if ziffer <= 0:
        index += 1
        continue

    else:
```

```
        if ziffer == 9:
            römische_zahl += symbole[index][0] + symbole[index][2]
        elif 6 <= ziffer <= 8:
            ziffer -= 5
            element = ziffer * symbole[index][0]

            römische_zahl += symbole[index][1] + element
        elif ziffer == 5:
            römische_zahl += symbole[index][1]

        elif ziffer == 4:
            römische_zahl += symbole[index][0] + symbole[index][1]

        elif 1 <= ziffer <= 3:
            element = ziffer * symbole[index][0]
            römische_zahl += element
    index += 1

return römische_zahl
```

num = r.randint(1,3999) print(num) print(umwandlung_römische_zahl(num))

---

4 Text in Dateien (*)

Über Moodle steht Ihnen die Textdatei `Beispieltext.txt` zur Verfügung. Darin ist zufälliger Text enthalten.

Ihre Aufgabe besteht darin, die Textdatei zunächst einzulesen. Nun soll:

- ab dem ersten Wort beginnend, jedes dritte Wort komplett groß geschrieben werden
- ab dem zweiten Wort beginnend, jedes dritte Wort komplett klein geschrieben werden
- ab dem dritten Wort beginnend, jedes dritte Wort ersetzt werden durch **Helau**.

Speichern Sie nun jeden Absatz einzeln in einer Datei `Veraendert1.txt` bis `VeraendertN.txt` ab.

- Datei öffnen

```
import string
import os

#erstellen der Datei
def erstellen_der_dateien(n:int)->None:
    for i in range(1, n+1):
        name = f"Verändert{i}.txt"
        open(name, 'w').close()

def datei_verändern():
```

```
    with open("Beispieltext.txt", "r") as f:
        inhalt = f.read().strip()
        zeilen = inhalt.split("\n\n")
        f.seek(0) #Ich gehe mit den Augen in die erste zeile
        anzahl_der_Absätze = len(zeilen)

        erstellen_der_dateien(anzahl_der_Absätze)


        for i in range(anzahl_der_Absätze): #nimm jeweils eine Zeile:
            splitet_list = zeilen[i].split()
            for j in range(len(splitet_list)):
                if j % 3 == 0:
                    splitet_list[j] = splitet_list[j].upper()
                elif j % 3 == 1:
                    splitet_list[j] = splitet_list[j].lower()
                elif j % 3 == 2:
                    splitet_list[j] = 'Helau'


            datei_i = f"Verändert{i+1}.txt"
            with open(datei_i, 'w') as d:
                d.write(" ".join(splitet_list))


datei_verändern()
```

Rekursion

1. Punktsumme
2. Primzahltest
3. Binärstellen zählen

Übung 9, 1

- n = int, Basis
- p = int, Potenz
- $S(n, p) = \sum_{k=1}^{n} k^p$

  → $S(3, 2) = 1^2 + 2^2 + 3^2 = 1 + 4 + 9 = 14$
- Wenn ich n = 3 habe, dann ist es $3 \rightarrow 2 \rightarrow 1$ & am Ende dann $1 \rightarrow 2 \rightarrow 3$
- p immer gleich !
- **Base case**:

  → Wenn n == 1

```
def punktsumme_berechnen(n:int, p:int):
    wenn n gleich 1 ist:
        dann gib mir die 1 zurück #weil 1**{etwas} immer = 1 ist

    Returne die n**p + punktsumme_berechnen(n-1, p)
```

- 1 Gruppe soll sich das n nehmen und es potenzieren
- sie soll der nächsten Abteilung die n-1 geben und die sollen darauf die p potenzieren
- diese sollen dann das gleiche machen bis eine Abteilung = 1 hat.
- dannach soll die letzte Abteilung das ergbenis an die Abteilung davor geben und diese Addieren
- immer so weiter bis wir am Anfang sind.
- ***Code***:

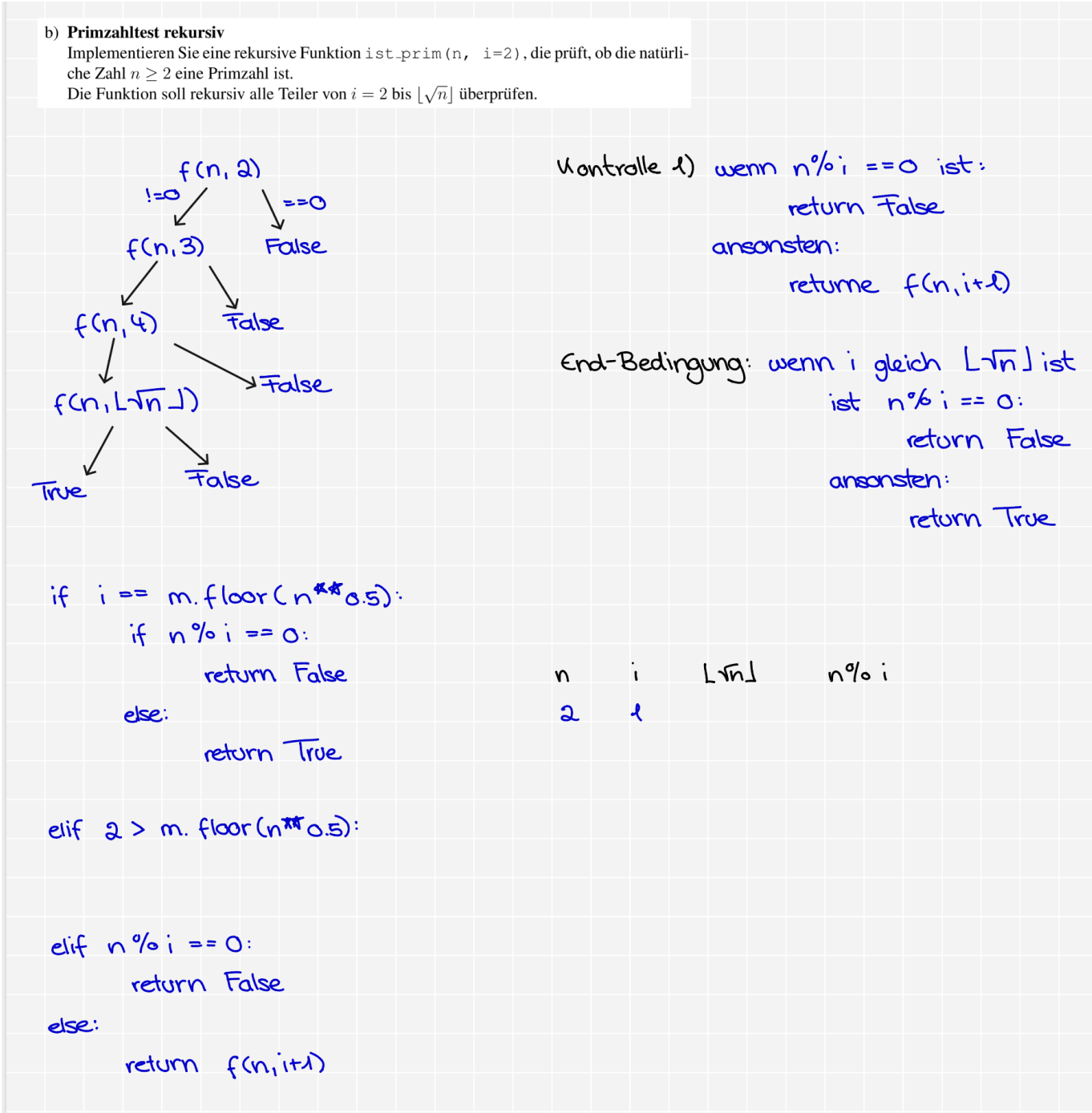
```
def punktsumme_berechnen(n:int, p:int) -> int:
if n == 1:
    return 1

return n**p + punktsumme_berechnen(n-1,p)


n = int(input("n: "))
p = int(input("p: "))
print(punktsumme_berechnen(n,p))
```

Korrektur:

```
def punktsumme_berechnen(n:int, p:int) -> int:
    if n == 0: #Besser, weil man auch n = 0 eingeben kann. Oben würde das
ein error geben !
        return 0

    return n**p + punktsumme_berechnen(n-1,p)


n = int(input("n: "))
p = int(input("p: "))
print(punktsumme_berechnen(n,p))
```

---

b) **Primzahltest rekursiv**
Implementieren Sie eine rekursive Funktion `ist_prim(n, i=2)`, die prüft, ob die natürliche Zahl $n \geq 2$ eine Primzahl ist.
Die Funktion soll rekursiv alle Teiler von $i = 2$ bis $\lfloor \sqrt{n} \rfloor$ überprüfen.



- bei n = 2 & 3 haben wir bei `m.floor(n**0.5)` = 1, welches bedeutet, dass 2 & 3 nur einen Teiler hat und zwar nur den Teiler i = 1, welches bedeutet, dass sie eine Primzahl sind, weil jede Zahl einen Teiler v. 1 hat !

```python
import math as m

def n_größergleich_zwei(n:int)->int:
    while n < 2:
        print("n muss >= 2 sein!")
```

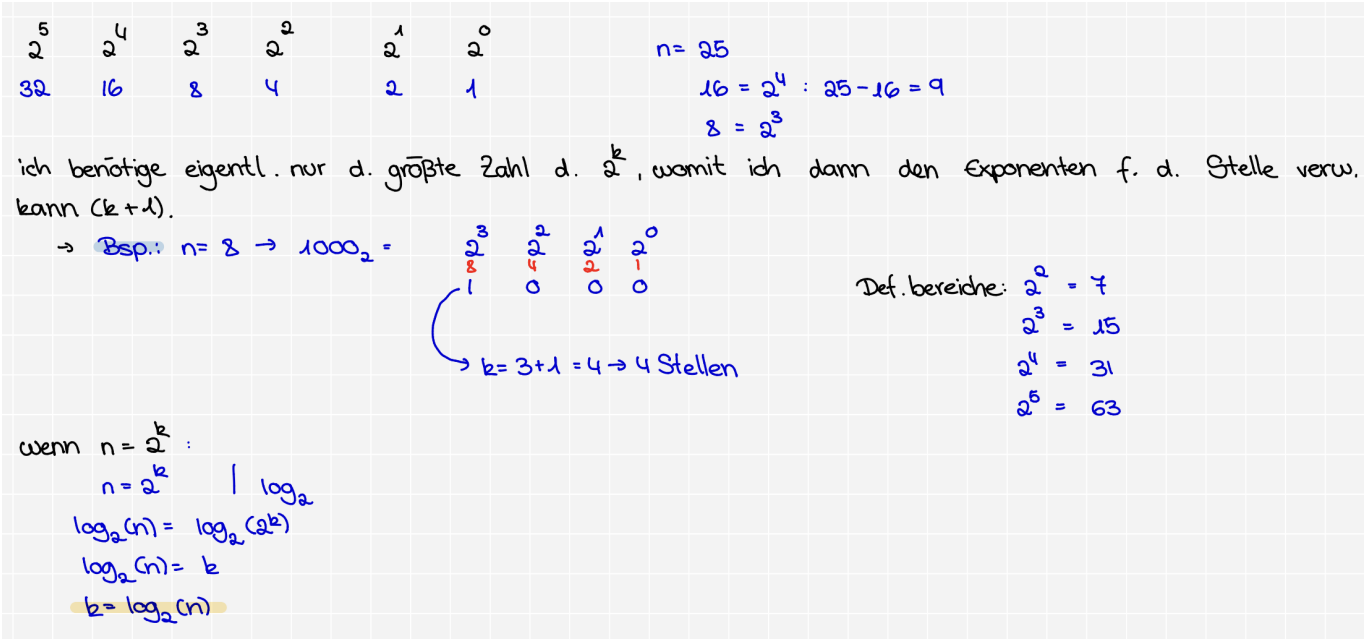
```
        n = int(input("n: "))
    return n

def primzahltest(n:int, i=2)->bool:

    #wenn m.floor(n**0.5), für n=2,3 = 1, dann ist es automatisch eine
Primzahl, weil 1 der Teiler von alles ganzen Zahlen ist. Somit haben 2 und
3 nur die 1 als Teiler == Primzahl !
    if i > m.floor(n**0.5):
        return True

    #wenn n > 3, dann müssen wir ein Stopp setzen, wenn i ==
m.floor(n**0.5) ist
    elif i == m.floor(n**0.5):
        if n %i == 0:
            return False
        else:
            return True

    #das ist die normale Kontrolle zwischen [2,m.floor(n**0.5)]
    elif n % i  == 0:
        return False
    else:
        return primzahltest(n,i+1)


n_test = int(input("n: "))
n = n_größergleich_zwei(n_test)
print(primzahltest(n))
```

---

c) **Binärstellen zählen**
Schreiben Sie eine rekursive Funktion `binaerstellen(n)`, die die Anzahl der Binärstellen (Bits) einer positiven ganzen Zahl n zurückgibt.

Beispiele:
$n = 8 = 1000_2 \rightarrow 4$ Stellen
$n = 1 = 1_2 \rightarrow 1$ Stelle.

- **direkte Weg, ohne Rekurssion**
  → habe d. Formel selbst hergeleitet 😋




```python
import math as m

def kontrolle_n(n:int)->int:
    while n <= 0:
        print('n muss eine positive Zahl & größer als 0 sein!')
        n = int(input('n: '))
    return n

def direkte_audgabe(n:int)->int:
    k = m.log2(n)
    return int(k+1)

n_old = int(input('n: '))
n = kontrolle_n(n_old)
print(direkte_audgabe(n))
```

- **Richtige Vers.**

```python
import math as m

def kontrolle_n(n:int)->int:
    while n <= 0:
        print('n muss eine positive Zahl & größer als 0 sein!')
        n = int(input('n: '))
    return n

def binärstellen(n:int)->int:
    if n <= 1:
        return 1
    else:
        return 1 + binärstellen(n//2)

def direkte_audgabe(n:int)->int:
```

```
    k = m.log2(n)
    return int(k+1)

n_old = int(input('n: '))
n = kontrolle_n(n_old)

print(direkte_audgabe(n))
print(binärstellen(n))
```

- Wrm. funktioniert d. ?
    1. Aufruf: `n=4`: `return 1 + binärstellen(2)`
        → "Ich habe hier eine 1, aber ich weiß noch nicht, was das Ergebnis von binärstellen(2) ist. Ich muss warten." ⏳ (Er schreibt sich "1 + ?" auf einen Zettel und legt ihn beiseite.)
    2. Aufruf: `n=2`: `return 1 + binärstellen(1)`
        → "Ich habe noch eine 1, aber ich muss auf das Ergebnis von binärstellen(1) warten."
    3. Aufruf: `n=1`: `if n <= 1`
        → gibt 1 weiter

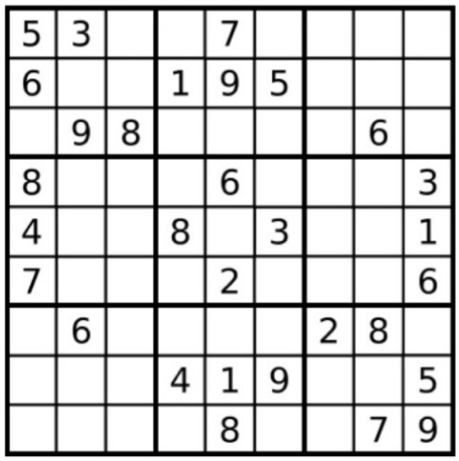
    → 1+1+1 = 3

Backtracking

---

2 Sudoku (**)

Das klassische Sudoku besteht aus 81 Feldern, die mit Zahlen zwischen 1 und 9 belegt werden müssen. Dabei darf in jeder Zeile, jeder Spalte, sowie jedem 3x3 Quadrat (siehe Abbildung) jede Zahl nur einmal vorkommen.



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

Ein Sudoku lässt sich mit Backtracking folgendermaßen lösen:

- Suche das erste freie Feld.
- Trage nacheinander in dieses Feld die Zahlen zwischen 1 und 9 ein.
- Versuche jeweils bei jeder Zahl das restliche Sudoku zu lösen.
- Wenn eine der Zahlenkombinationen zum Erfolg führt, brich die Suche ab und gib die Einträge als Lösung aus.
- Wenn keine Zahl zum Erfolg führt, mache das bearbeitete Feld wieder frei und melde, dass es für dieses Sudoku keine Lösung gibt.

Setzen Sie diese Strategie in einem *rekursiven* Sudoku-Löser um.
Die Funktion `def loese_sudoku(sudoku)` erhält ein eindimensionales Zahlenarray mit 81 Zahlen zwischen 0 und -9 bei dem immer neun aufeinander folgende Zahlen eine Zeile im Sudoku widerspiegeln. Die negativen Zahlen sind natürlich als Betrag zu werten, sie zeigen aber die Stellen an, die fest vorgegeben sind und nicht verändert werden dürfen. Alle Stellen, die zunächst mit 0 belegt sind, sind offene zu besetzende Stellen.

- ∀ geschieht in place !

- $\text{num} \in [0, 9]$, num darf nur $1\times$ in einem 3×3-Block, Spalte & Zeile enthalten sein
  → Allgemien Logik:

```
for num in range(1,10):
    wenn num in Block:
        continue
    wenn num in spalte:
        continue
    wenn num in zeile:
        continue
    ansonsten:
        setzte 0_index = num
```

Soduko Feld geg.:

```
sudoku = [5, 3, 0, 0, 7, 0, 0, 0, 0, 6, 0, 0, 1, 9, 5, 0, 0, 0, 0, 9, 8,
0, 0, 0, 0, 6, 0, 8, 0, 0, 0, 6, 0, 0, 0, 3, 4, 0, 0, 8, 0, 3, 0, 0, 1, 7,
0, 0, 0, 2, 0, 0, 0, 6, 0, 6, 0, 0, 0, 0, 2, 8, 0, 0, 0, 0, 4, 1, 9, 0, 0,
5, 0, 0, 0, 0, 8, 0, 0, 7, 9]
```

Müssen wir weiter gucken oder sind wir fertig ?:

```
wenn es keine 0 gibt:
    return sudoku

mache ansosnten einfach weiter
```

- wir müssen 3 Fälle kontrollieren: Zeile, Spalten & Block:
- wir haben einen Wächter d. sagt, ob es erlaubt oder ¬ erlaubt ist: `ist_erlaubt = True`
  → Wir gehen mal davon aus, dass es erlaubt ist
- wenn ein Fall bereits enthalten ist, dann müssen wir es direkt abbrechen und mit d. nächsten Zahl weiter machen!

Fall 1: Zeile:

```
for i in range(spalte, 81, 9):
    wenn sudoku[i] == num:
        ist_erlaubt = False
        ABBRUCH #wenn es hier schon False ist, dann bracuhen wir nicht
weiter zu gucken
```

```
for i in range(spalte, 81, 9):
    if sudoku[i] == num:
        ist_erlaubt = False
        break #wenn es hier schon False ist, dann bracuhen wir nicht
weiter zu gucken
```

Fall 2: Zeile:

```
start_zeile = zeile * 9
for i in range(start_zeile, start_zeile + 9):
    wenn sudoku[i] == num:
        ist_erlaubt = False
        ABBRUCH #wenn es hier schon False ist, dann bracuhen wir nicht
weiter zu gucken
```

```
start_zeile = zeile * 9
for i in range(start_zeile, start_zeile + 9):
    if sudoku[i] == num:
        ist_erlaubt = False
        break #wenn es hier schon False ist, dann bracuhen wir nicht
weiter zu gucken
```

Fall 3: Block:

- finde d. Position v. meiner 0 innerhalb seiner Blockes ($\text{spalte} \in [0,2]$, $\text{zeile} \in [0,2]$)
  → Berechne d. 1. Pos. in dem jeweiligen Block

```
zeile_rest = zeile % 3
spalte_rest = spalte % 3
block_start = (index - spalte_rest) - (zeile_rest * 9)
block_start1 = block_start + 1
block_start2 = block_start2 + 1

round = 0
for _ in range(3):
    block_start, block_start1, block_start2 = block_start + j,
block_start1 + j, bock_start2 + j
    round = 9
    wenn  sudoku[blockstart] == num oder sudoku[blockstart1] == num oder
sudoku[blockstart2] == num:
        ist_erlaubt = False #Wächter gibt Alerm aus
        break
```

Finale Kontrolle für Zahl:

```
wenn der Wächter True sagt, dann ist d. Zahl erlaubt
    * sudoku[index0] wird dann gleich d. Zahl gesetzt
    * das Ergebnis v. d. loese_sudoku(sudoku) speichern wir in der Var.
ergebnis
        * ist das Ergebnis None ?
            * wenn ja, dann wissen wir, dass diese Zahl nicht die richtige
war und machen mit der nächsten Zahl weiter
            * wenn nein, dann geben wir die veränderte Liste an
loese_sudoku(sudoku) weiter # Sie nimmt ich dann die nächste 0 und macht
wieder das gleiche
```

```
if ist_erlaubt:
    sudoku[index] = num
    ergebnis = loese_sudoku(sudoku)
    if ergebnis != None:
        return Ergebnis
    else:
        sudoku[index] = 0 # wir sind hier immer noch in der Schleife (Es
soll ja noch die restlichen Zahlen gucken!!!
```

- None wird nur einmal ausgegeben

Fertiger Code:

```
def loese_sudoku(sudoku):
    # 1. Basisfall: Wenn keine 0 mehr da ist, sind wir fertig!
    if 0 not in sudoku:
        return sudoku

    # 2. Finde das erste freie Feld
    index = sudoku.index(0)
    zeile = index // 9
    spalte = index % 9

    # 3. Wir probieren die Zahlen 1 bis 9
    for num in range(1, 10):

        # --- DER WÄCHTER-CHECK 👮 ---
        # Wir gehen davon aus, dass die Zahl erlaubt ist, bis wir das
Gegenteil beweisen.
        ist_erlaubt = True

        # Wächter 1: Spalte prüfen ⬇️
        for i in range(spalte, 81, 9):
            if sudoku[i] == num:
                ist_erlaubt = False
                break # Alarm! Zahl gefunden.
```

```
        # Wächter 2: Zeile prüfen ➡
        start_zeile = zeile * 9
        for i in range(start_zeile, start_zeile + 9):
            if sudoku[i] == num:
                ist_erlaubt = False
                break # Alarm! Zahl gefunden.

        # Wächter 3: 3x3 Block prüfen 📦
        # Deine Formel für die Ecke oben links:
        zeile_rest = zeile % 3
        spalte_rest = spalte % 3
        block_start = (index - spalte_rest) - (zeile_rest * 9)
        block_start1 = block_start + 1
        block_start2 = block_start1 + 1

        round = 0
        for _ in range(3):
            block_start, block_start1, block_start2 = block_start + round,
block_start1 + round, block_start2 + round
            round = 9

            if sudoku[block_start] == num or sudoku[block_start1] == num
or sudoku[block_start2] == num:
                ist_erlaubt = False
                break # Alarm! Zahl gefunden.


        # 4. Wenn alle Wächter "Okay" sagen:
        if ist_erlaubt:
            sudoku[index] = num # Zahl setzen
            ergebnis = loese_sudoku(sudoku)
            if ergebnis != None:
                return ergebnis
            else:
                sudoku[index] = 0

    return None


sudoku = [5, 3, 0, 0, 7, 0, 0, 0, 0, 6, 0, 0, 1, 9, 5, 0, 0, 0, 0, 9, 8,
0, 0, 0, 0, 6, 0, 8, 0, 0, 0, 6, 0, 0, 0, 3, 4, 0, 0, 8, 0, 3, 0, 0, 1, 7,
0, 0, 0, 2, 0, 0, 0, 6, 0, 6, 0, 0, 0, 0, 2, 8, 0, 0, 0, 0, 4, 1, 9, 0, 0,
5, 0, 0, 0, 0, 8, 0, 0, 7, 9]
print(loese_sudoku(sudoku))
```