Aufgabe 1: Caesar-Verschlüsselung

Der römische Feldherr Gaius Julius Caesar entwickelte ein Verschlüsselungsverfahren, mit dem er militärische Berichte vor unerwünschten Lesern geschützt hat. Sein Biograph Sueton beschreibt es wie folgt:

> *… wenn etwas Geheimes zu überbringen war, schrieb er in Zeichen, das heißt, er ordnete die Buchstaben so, dass kein Wort gelesen werden konnte: Um diese zu lesen, tauscht man den vierten Buchstaben, also D für A aus und ebenso mit den restlichen.*

Es wird also jeder Buchstabe um drei Zeichen nach rechts verschoben, also ein A durch ein D, ein B durch ein E ersetzt und so weiter. Am „Rand“ des Alphabets wird wieder nach vorne gesprungen: Ein X wird durch ein A, ein Y durch ein B und ein Z durch ein C ersetzt.

Aufgabe a

Hilf Ceasar und schreibe ein Programm, das die Verschlüsselung automatisiert. Es soll aus einer Funktion „`verschluesseln(Text)`“ bestehen, die einen Text mit dem beschriebenen Verfahren verschlüsselt.

Beispiel:

```
>>> verschluesseln("DIESER TEXT IST GEHEIM")
"GLHVHU WHAW LVW JHKHLP"

>>> verschluesseln("HALLO! KANNST DU DAS LESEN?")
"KDOOR! NDQQVW GX GDV OHVHQ?"
```

Aufgabe b

Natürlich sollst Du auch beim entschlüsseln helfen. Schreib dazu eine Funktion “`entschluesseln(Text)`”, die einen Geheimtext entschlüsselt.

Beispiel:

```
>>> entschluesseln("DQJULII DXI JDOOLHQ DE PRUJHQ")
"ANGRIFF AUF GALLIEN AB MORGEN"

>>> entschluesseln("100 ZLOGVFKZHLQH IXHU REHOLA!")
"100 WILDSCHWEINE FUER OBELIX!"
```

Aufgabe c

Der folgende Text wurde mit einem ähnlichen Verfahren verschlüsselt. Durch eine kleine Anpassung Deines Programmes kannst Du ihn entschlüsseln.

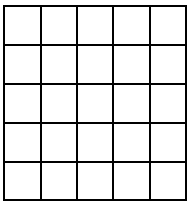
```
FHCRE! QVRFRE GRKG JHEQR ZVG QRZ EBG-QERVMRUA-IRESNUERA
IREFPUYHRFFRYG. QNORV JVEQ WRQRE OHPUFGNOR HZ QERVMRUA
MRVPURA IREFPUBORA. QN QNF NYCUNORG FRPUFHAQMJNAMVT
OHPUFGNORA UNG, XNAA RVA GRKG QHEPU MJRVSNPUR
IREFPUYHRFFRYHAT RAGFPUYHRFFRYG JREQRA. NPU WN: QNF
TRURVZJBEG VFG "XHPURA".
```

Wie lautet das Geheimwort?

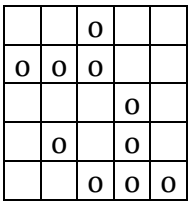
Aufgabe 2: Spiel des Lebens / Game of Life

Beschreibung und Regeln

Der Mathematiker John Horton Conway hat 1970 ein Spiel erschaffen, das mit einfachen Regeln das Leben von Zellen in einer zweidimensionalen Ebene („die **Welt**“) beschreibt. Diese Ebene ist quadratisch und ist in Zeilen und Spalten unterteilt:

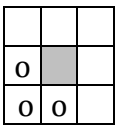


In den einzelnen Feldern können **Zellen** leben, die wir mit Kreise kennzeichnen. Leere Felder entsprechen toten Zellen.



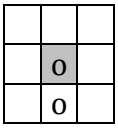
Wir nennen eine solche Verteilung von Zellen auf der Welt eine **Generation**. Für jede Zelle kann nun anhand seiner Nachbarn die **Nachfolgergeneration** berechnet werden. Dabei gibt es vier Regeln:

1. Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Nachfolgegeneration neu geboren.

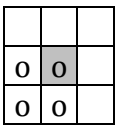
   Hier wird an der grauen Position eine neue Zelle geboren.

2. Eine lebendige Zelle mit weniger als zwei lebenden Nachbarn stirbt in der Nachfolgegeneration an Einsamkeit.

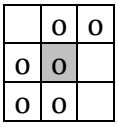
   Die Zelle an der grauen Position stirbt an Einsamkeit.

3. Eine lebendige Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Nachfolgegeneration lebendig.



Die Zelle an der grauen Position überlebt.

4. Eine lebendige Zelle mit mehr als drei lebenden Nachbarn stirbt in der Folgegeneration an Überbevölkerung.



Die Zelle an der grauen Position stirbt an Überbevölkerung.

Dabei hängt das Leben und Sterben einer Zelle nur von seinen direkten Nachbarn ab. Wenn eine Zelle am Rand lebt, hat sie entsprechend weniger Nachbarn. Als Beispiel hier die maximale Anzahl der Nachbarn einer Zelle:

3	5	5	5	3
5	8	8	8	5
5	8	8	8	5
5	8	8	8	5
3	5	5	5	3

Vorbereitung

Wir stellen zwei Funktionen zur Verfügung:

- Die Funktion `generateWorld(size)` gibt eine quadratische Welt der Größe `size` mit einer zufälligen Verteilung von lebendigen Zellen zurück. Diese Welt ist als „Liste von Listen“ repräsentiert. Lebendige Zellen werden durch eine 1, tote Zellen durch eine 0 repräsentiert.
  Auf die dritte Zeile und neunte Spalte einer Welt `world` kann mit `world[2][8]` zugegriffen werden (wir fangen mit 0 an zu Zählen). Die Zelle am oberen linken Rand ist also `world[0][0]` und die am unteren rechten Rand ist `world[size-1][size-1]`. Mit `len(world)` kann die Größe der Welt (d.h. die Anzahl der Zeilen und Spalten) abgefragt werden.

- Die Funktion `printWorld(world)` gibt die Welt graphisch aus.

Beispiel:

```
>>> world = generateWorld(5)
```

```
>>> world
[[0, 0, 1, 0, 0], [1, 1, 1, 0, 0], [0, 0, 0, 1, 0],
[0, 1, 0, 1, 0], [0, 0, 1, 1, 1]]

>>> printWorld(world)
+-----+
|  o  |
|ooo  |
|   o |
| o o |
|  ooo|
+-----+

>>> len(world)
5

>>> world[2][3]
1
```

Außerdem ist die Beispielwelt in der Variable `example` gespeichert.

Aufgabe a

Schreibe eine Funktion `neighbors(world, z, s)`, die die Anzahl der lebendigen Nachbarzellen der Zelle in der Welt `world` in Zeile `z` und Spalte `s` ausgibt! `s` und `z` können dabei Werte zwischen `0` und `len(world)` annehmen.

Für die Beispielwelt gilt:

		o		
o	o	o		
			o	
	o		o	
		o	o	o

```
>>> neighbors(example, 0, 0)
2
>>> neighbors(example, 0, 1)
4
>>> neighbors(example, 0, 2)
2
>>> neighbors(example, 0, 3)
2
>>> neighbors(example, 0, 4)
0
>>> neighbors(example, 1, 0)
1
>>> neighbors(example, 1, 1)
3
>>> neighbors(example, 1, 2)
3
>>> neighbors(example, 1, 3)
3
>>> neighbors(example, 1, 4)
```

Aufgabe b

Schreibe eine Funktion `evolve(world)`, die die Nachfolgegeneration für die Welt `world` berechnet und zurückgibt.

```
>>> # 1. Generation
>>> printWorld(example)
+-----+
|  o  |
|ooo  |
|   o |
| o o |
|  ooo|
+-----+

>>> # 2. Generation
>>> printWorld(evolve(example))
+-----+
|  o  |
| ooo |
|o  o |
|     |
|  ooo|
+-----+

>>> # 3. Generation
>>> printWorld(evolve(evolve(example)))
+-----+
| ooo |
| o o |
| o o |
|  o o|
|   o |
+-----+
```

Aufgabe c

Nach wie vielen Generationen ist die Welt stabil, d.h. ändert sich nicht mehr?

Aufgabe 3: Prüfung einer Kreditkarte

Jede Kreditkarte hat eine 16-stellige Kartennummer. Aus der Kartennummer kann man einige Informationen ablesen. Die ersten sechs Ziffern stehen für das Ausgabeinstitut und die Art der Kreditkarte. Z.B. ist 376211 eine *Singapore Airlines Krisflyer American Express Gold Card*, während 529962 eine Pre *Paid Much Music MasterCard* bezeichnet. Es folgen 9 Ziffern mit der Kontonummer des Karteninhabers. Die letzte Ziffer ist eine Prüfziffer, die es gestattet, „echte" Kartennummern von einfach geratenen zu unterscheiden.

Die Prüfung einer Kartennummer geschieht nach folgendem Verfahren:

- Schreibe unter jede Ziffer an ungerader Position das Doppelte dieser Ziffer (die Ziffer links außen hat Position 1)
- Schreibe unter jede Ziffer an gerader Position die Ziffer selbst
- Addiere die Quersummen der notierten Zahlen
- Die Karte ist gültig, wenn das Ergebnis durch 10 teilbar ist, sonst ungültig.

Beispiel: Wir prüfen die Kartennummer 4417 1234 5678 9113.

4	4	1	7	1	2	3	4	5	6	7	8	9	1	1	3
8	4	2	7	2	2	6	4	10	6	14	8	18	1	2	3

Summe der Quersummen: 8+4+2+7+2+2+6+4+1+0+6+1+4+8+1+8+1+2+3 = 70, also ist die Kartennummer gültig!

Aufgabe a

Schreibe ein Programm, das eine Kartennummer auf Gültigkeit prüft. Es soll aus einer Funktion „`pruefen(Zahl)`" bestehen, die `True` zurückgibt, wenn die Karte gültig ist.

Beispiel:

```
>>> pruefen(4417123456789113)
True

>>> pruefen(4417123456789114)
False

>>> pruefen(11111111111111111111111111)
False
```

```
>>> pruefen(0)
True
```

Aufgabe b

Schreibe ein Programm, das eine Kartennummer zu einem gegebenen 6-stelligen Code für das Ausgabeinstitut und Kartenart sowie einer 9-stelligen Kontonummer erzeugt. Es soll aus einer Funktion „`generieren(Institut,Konto)`“ bestehen, die eine gültige Kartennummer als Zahl zurückgibt.

Beispiel:

```
>>>generieren(441712, 345678911)
4417123456789113

>>>generieren(341712, 345678911)
3417123456789115
```

Aufgabe 4: Schiebepuzzle

Beim Schiebepuzzle können immer neue Kombinationen erzeugt werden, indem eines der Teile in das leere Feld geschoben wird. Wir wollen uns rechteckige Puzzles verschiedener Größe anschauen, wobei in der Startkonfiguration alle Teile geordnet sind und das leere Feld rechts unten ist.

Aufgabe a

Schreibe ein Programm **„`kombinationen(Zeilen,Spalten)`“**, das zu eingegebener Spielfeldgröße ermittelt, wie viele verschiedene Kombinationen aus der Startkonfiguration entstehen können.

Beispiel:

```
>>>kombinationen(2,2)
12

>>>kombinationen(2,3)
360

>>>kombinationen(3,3)
181440
```

Aufgabe b

Schreibe ein Programm `loese(Konfiguration)`, das eine gegebene Konfiguration eines 2x2 Puzzles in die sortierte Anfangskonfiguration überführt. Das Ergebnis soll eine Liste von Schritten aus hoch, runter, links, rechts sein, die jeweils angeben, welche Bewegung ein Steins in das freie Feld hinein ausführt. Die Konfiguration wird als Liste von Listen angegeben, die das Feld zeilenweise beschreiben. 0 repräsentiert das leere Feld.

Beispiel:

```
>>> loese([[2,3],[1,0]])
["runter", "rechts", "hoch", "links"]
```

2	3	→	2		→		2	→	1	2	→	1	2
1			1	3		1	3			3		3	

Zusatzaufgabe

Ermittle den jeweils kürzesten Weg!