INSTITUT FÜR MATHEMATIK LEHRSTUHL IX
Nadja Vater

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Hausaufgabe 3

1 Allgemeines

1.1 Abgabe

Diese Hausaufgabe muss bis 11.03.24 um 9:00 Uhr abgegeben werden. Bitte geben Sie Ihre Lösung bis zu diesem Zeitpunkt über PABS ab.

1.2 Punkte und Dateien

Diese Hausaufgabe besteht aus mehreren Aufgaben, für die insgesamt 20 Punkte erreicht werden können. Zusätzlich können bis zu 2 Bonuspunkte erreicht werden. Insgesamt wird es 3 Hausaufgaben geben. Im `src` Ordner finden Sie Python Dateien mit leeren, zu implentierenden Funktionen. In der Regel gibt es zu jeder Aufgabe eine Python Datei, die Sie durch den Namen erkennen können.

1.3 Integrierte Tests `doctest`

In der Regel sind die Funktionen schon mit *docstrings* mit doctests ausgestattet.

Diese Tests sind für Sie zum Überprüfen Ihrer Funktionen eingefügt, sie haben keinen Einfluss auf die Bewertung der Implementierung. Sie können sie beliebig ändern, entfernen und ergänzen.

2 Aufgabe: Sudoku (10 Punkte)

Für diese Aufgabe sollten Sie mit Generatoren und Backtracking vertraut sein. Sie werden einen Löser für klassische Sudokus implementieren.

2.1 Einleitung: Sudoku

Für diese Aufgabe werden die grundsätzlichen Regeln von Sudoku (Wikipedia) als bekannt vorausgesetzt. Sudokus gibt es in mehreren Formen. Klassischerweise bestehen sie aus einen 9×9 Feld mit 3×3 Blöcken der Größe 3×3. Wir betrachten eine etwas allgemeinere Variante deren quadratische Grundfläche eine *Basis* $n \geq 2$, $n \in \mathbb{N}$, hat. Für $n = 3$ erhalten wir das klassische 9×9 Puzzle, für $n = 2$ besteht das Puzzle aus 2×2 Blöcken der Größe 2×2, sodass das gesamte Feld die Größe 4×4 hat, also aus $4^2 = 16$ Feldern besteht. Genauso besteht ein Puzzle mit Basis $n = 4$ aus 4×4 Blöcken der Größe 4×4 und hat insgesamt $16^2 = 256$ Felder.

2.2 Datenstruktur

Wir repräsentieren ein Sudokufeld als 2-dimensionales `numpy` array. Die Einträge sind Ganzzahlen (`int`) zwischen 0 und n^2, wobei leere Felder durch `0` repräsentiert werden.

```
import numpy as np

# Assumptions: two-dimensional, square shape, side length is a square number,
# entries are nonnegative integers, zero represents a missing value.
Puzzle = np.array
```

Sie können davon ausgehen, dass die zu implementierenden Funktionen nur mit gültigen Argumenten aufgerufen werden, die Arrays also quadratisch mit Größe $n^2 \times n^2$ und Indizes aus dem richtigen Bereich sind. Mit ungültigen Argumenten darf die Funktion beliebig umgehen.

2.3 Zwei Hilfsfunktionen: `base` und `copy_puzzle` (1 Punkt)

Das Ziel der beiden Funktionen `base` und `copy_puzzle` sollte selbsterklärend sein. Die Berechnung der Basis und das Erzeugen einer Kopie werden später hilfreich sein. Mit der Verwendung der richtigen Funktionen aus der Standardbibliothek bzw. dem `numpy` Paket kann die Implementierung sehr einfach gelingen.

```
def base(p: Puzzle) -> int:
    """Determine the base of a puzzle, e.g., 3 for a size 9 x 9 puzzle."""
    # ...

def copy_puzzle(p: Puzzle) -> Puzzle:
    """Return a fully independent copy of a puzzle."""
    # ...
```

2.4 Lesen und Schreiben von Sudokus: `from_file` und `as_string` (2 Punkte)

Implementieren Sie die Funktionen

```
def from_file(path: str) -> Puzzle:
    """Read a puzzle from the file at path.

    Input is assumed to be lines of numbers or underscores separated by
    whitespace. Underscores represent missing values.
    """
    # ...

def as_string(p: Puzzle) -> str:
    """Format a Sudoku puzzle as a string.

    The resulting string has one line per row and entries separated by a
    single space. Empty squares are marked with an underscore and non-empty
    squares are represented as numbers.
    """
    # ...
```

Beide Funktionen sollten Zeichenketten im gleichen Format verarbeiten. Dabei entspricht eine Zeile des Sudokus auch einer Zeile in der Zeichenkette und die Einträge werden durch ein einzelnes Leerzeichen voneinander getrennt, leere Felder werden mit Unterstrich `"_"` markiert. (*Hinweis*: Die Funktion `numpy.genfromtxt` könnte hierfür hilfreich sein, muss aber nicht verwendet werden.) Beispielsweise kann das folgende Puzzle zum Testen verwendet werden, das in der Datei `puzzles/example_9x9_puzzle.txt` abgespeichert ist:

```
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
```

2.5 Einträge sammeln (3 Punkte)

Wir adressieren die Felder des Sudokus durch Paare `i, j`, wobei `i` der Zeilenindex und `j` der Spaltenindex ist. Beide nehmen Werte aus $\{0, 1, \ldots, n^2 - 1\}$ an. Um herauszufinden, welche Werte noch an einer bestimmten Stelle in das Sudoku eingefügt werden können, müssen wir zunächst wissen welche Zahlen schon in der gleichen Zeile, Spalte und im gleichen Block eingetragen sind. Implementieren Sie hierfür die Funktionen

```
def in_row(i: int, p: Puzzle) -> set[int]:
    """Return the entries in row i (0-indexed) as a set."""
    # ...

def in_column(j: int, p: Puzzle) -> set[int]:
    """Return the entries in column j (0-indexed) as a set."""
    # ...

def in_block(i: int, j: int, p: Puzzle) -> set[int]:
    """Return the entries in the block at row i and column j as a set.

    For a Sudoku with base n, the block will have size n x n."""
    # ...
```

Stellen Sie sicher, das keine leeren Felder (also der Wert `0`) in der Rückgabemenge enthalten sind.

2.6 Lösen der Puzzles (4 Punkte)

Implementieren Sie die Generatorfunktion

```
def solve(p: Puzzle):
    """Generate all solutions for the given puzzle."""
    # ...
```

die alle möglichen Lösungen für ein gegebenes Puzzle erzeugt. Die Reihenfolge in der die Lösungen erzeugt werden ist dabei irrelevant, aber es darf weder doppelt, noch falsche oder unvollständige Lösungen geben. Sie können dabei einen beliebigen Ansatz wählen, aber Backtracking wird empfohlen.

2.6.1 Hinweise zum Backtracking

Ihre Lösung wird wahrscheinlich Änderungen der Arguments (Puzzles) beinhalten. Das kann zu logischen Fehlern (Bugs) führen, die schwer zu finden sein können. Daher ist es empfehlenswert bei rekursiven Aufrufen sicherzustellen, dass entweder eine Kopie des Puzzles weitergegeben wird oder die Änderungen am Puzzle gegebenenfalls wieder rückgängig gemacht werden, bevor die Funktion verlassen wird. Außerdem muss für alle Lösungen, die zurückgegeben werden beispielsweise durch sinnvolles Kopieren sichergestellt werden, dass dieses Ergebnisse nicht versehentlich an anderer Stelle verändert werden können.

Achten Sie darauf, Felder in denen Zahlen vorgegeben sind, nicht zu überschreiben. Die Funktion `solve` oder eine beliebig definierbare innere Funktion können rekursiv aufgerufen werden.

2.6.2 Überprüfen der Lösungsfunktion

Sie finden ein Beispielsudoku in der Datei `puzzles/example_9x9_puzzle.txt` und dessen Lösung in der Datei `puzzles/example_9x9_solution.txt`. Sie können dieses oder andere Beispiele nach Belieben verwenden. Alleine die Datei `sudoku.py` ist Teil der Lösung für diese Aufgabe. Wenn dem Programm ein Dateiname als Argument übergeben wird, wird es versuchen mithilfe der von Ihnen implementierten Funktionen ein Sudoku aus dieser Datei zu lesen, alle Lösungen zu erzeugen, und diese auszugeben. Für eine korrekte Implementierung wird also der Aufruf

```
python sodoku.py example_9x9_puzzle.txt
```

die gleiche Lösung erzeugen, die in der Datei `puzzles/example_9x9_solution.txt` hinterlegt ist. Die Lösung ist in diesem Fall eindeutig.

Ihr Programm sollte dazu in der Lage sein alle möglichen Lösungen für Puzzle mit Basis $n = 2$ zu erzeugen. Um sich dieses Ergebnis plausibel zu machen, bestimmen Sie zunächst die Anzahl aller Lösungen. Wie viele der Lösungen haben als erste Zeile `1 2 3 4`? Können Sie den Unterschied begründen? Diese Fragen sind nicht Teil der Aufgabe, aber können Ihnen beim Testen Ihrer Abgabe helfen.

3 Aufgabe: Lindenmayer-Systeme (10 Punkte + 2 Bonuspunkte)

Diese Aufgabe beschäftigt sich mit der Erzeugung von Fraktalen und "künstlichen Pflanzen" durch Lindenmayer-Systeme. Zur Bearbeitung der Aufgabe sollten Sie unter anderem mit Schleifen und dem Arbeiten mit Zeichenketten (Python-Typ `str`) vertraut sein.

3.1 Einführung Lindenmayer-Systeme

Ein Lindenmayer-System (L-System) ist eine Art formale Grammatik mit definierten Ersetzungsregeln mit der sich unter anderem Darstellungen von Fraktalen erzeugen lassen.

Formal können wir ein (kontextfreies) L-System als Tripel (V, ω, P) definieren, wobei V ein Alphabet ist, $\omega \in \bigcup_{k=1}^{\infty} V^k$ ein Wort dieses Alphabets, welches auch Startwort oder Axiom genannt wird, und $P \subseteq V \times \bigcup_{k=0}^{\infty} V^k$ schließlich die Menge der Ersetzungsregeln.

Das zu einem gegebenen L-System gehörige Wort w_n der Tiefe $n \in \mathbb{N}_0$ wird rekursiv erzeugt:

- Für $n = 0$ ist $w_n = \omega$, also das Startwort des L-Systems
- Für $n > 0$ werden die Ersetzungsregeln aus P "gleichzeitig" auf alle Buchstaben des Wortes w_{n-1} angewandt.

Wird eine Ersetzungsregel $(c, v) \in P$ auf ein Wort $w = c_0 c_1 \dots c_n$ angewandt, ergibt sich das neue Wort dadurch, dass jeder Buchstabe c_i mit $c_i = c$ durch v ersetzt wird. Die übrigen Buchstaben von w bleiben unverändert. Die Reihenfolge der Anwendung der Ersetzungsregeln ist für das neue Wort nicht von Belang.

3.1.1 Beispiel

Betrachte das L-System (V, ω, P) mit $V = \{F\}$, $\omega = F$ und $P = \{(F, FF)\}\}$. Dann gilt

$$w_0 = \omega = F, \quad w_1 = FF, \quad w_2 = FFFF, \dots.$$

Um w_1 zu erzeugen wird die Ersetzungsregel einmal angewandt und der Buchstabe F des Axioms durch FF ersetzt. Zur Erzeugung von w_2 wird die Regel zweimal angewandt - einmal für jedes der beiden F aus w_1.

3.1.2 Beispiel

Betrachte das L-System (V, ω, P) mit $V = \{F, G, +, -\}$, $\omega = F$ und $P = \{(F, G - F - G), (G, F + G + F)\}$. Dann gilt

$$w_0 = \omega = F, \quad w_1 = G - F - G, \quad w_2 = F + G + F - G - F - G - F + G + F, \dots.$$

Um w_1 zu erzeugen, wird das F aus dem Axiom ω durch $G - F - G$ ersetzt (erste Regel in P) und um w_2 zu erzeugen werden die beiden G und das F aus w_1 jeweils nach den Ersetzungsregeln aus P ersetzt. Symbole, für die keine Ersetzungsregeln definiert sind, bleiben im Resultat unverändert erhalten, werden also nicht ersetzt.

3.2 Darstellung von Lindenmayer-Systemen

Zur Darstellung eines Wortes aus einem L-System als Zeichnung muss eine geometrische Interpretation der Zeichen des Alphabets vorgegeben werden. Wie im Buch Prusinkiewicz und Hanan (2013) interpretieren wir ein Wort als Beschreibung eines Weges eines Roboters (*turtle*) im Raum.

Den Status unseres Roboters beschreiben wir durch ein Tripel (x, y, α), wobei $(x, y) \in \mathbb{R}^2$ kartesische Koordinaten und $\alpha \in [0, 360)$ die Blickrichtung des Roboters ist, beschrieben durch den Winkel, den die diese mit der x-Achse einschließt. Unser Roboter kann die folgenden Aktionen durchführen, wobei der Winkel δ schon zu Beginn fest vorgegeben ist:

1. einen Schritt der Länge 1 machen und dabei eine Linie zeichnen: Die Position des Roboters mit Status (x, y, α) ändert sich zu $(x + \cos(\alpha), y + \sin(\alpha))$.
2. um den Winkel δ nach links (gegen den Uhrzeigersinn) oder rechts (mit dem Uhrzeigersinn) drehen
3. aktuelle Position und Blickrichtung speichern
4. an den zuletzt gespeicherten Status zurückspringen, ohne dabei eine Linie zu zeichnen

Jedes Zeichen in einem vorgegebenen Wort ruft eine bestimmte Aktion des Roboters hervor. Wir verwenden die folgenden Interpretationen:

Zeichen	Interpretation
Großbuchstabe	1. Mache einen Schritt der Länge 1
+	2. Drehung nach rechts um den Winkel δ
-	2. Drehung nach links um den Winkel δ
[	3. Speichere den aktuellen Status
]	4. Springe zum zuletzt gespeicherten Status zurück
sonstiges	kein Effekt

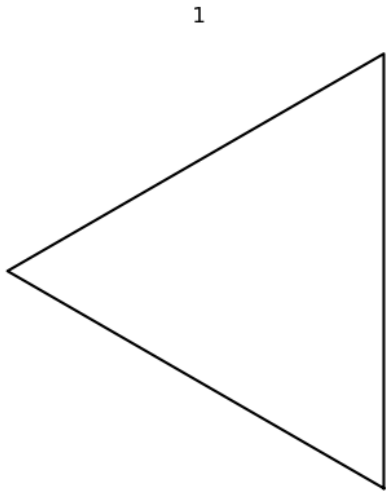
Zu Beginn blickt der Roboter nach oben ($\alpha = 90$) und befinden sich an der Position $(0, 0)$.

- Durch das Wort F zeichnet der Roboter also eine Linie von $(0, 0)$ nach $(0, 1)$.
- Für $\delta = 90$ zeichnet der Roboter durch das Wort $+F$ eine Linie von $(0, 0)$ nach $(1, 0)$, da er sich zunächst um $90°$ nach rechts gedreht hat.

3.2.1 Beispiel

Die Darstellung des Wortes $w_1 = G - F - G$ aus dem L-System (V, ω, P) mit $V = \{F, G, +, -\}$, $\omega = F$ und $P = \{(F, G - F - G), (G, F + G + F)\}$ mit Winkel $\delta = 120°$ ist ein Dreieck:

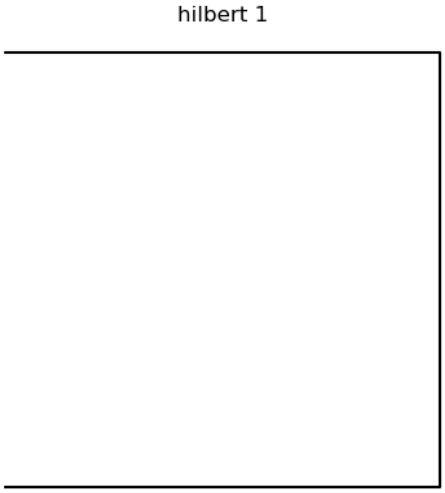
Zeichen	Interpretation
G	Bewegung nach vorn um Länge 1 und Zeichnung
-	Drehung nach links um 120°
F	Bewegung nach vorn um Länge 1 und Zeichnung
-	Drehung nach links um 120°
G	Bewegung nach vorn um Länge 1 und Zeichnung



3.2.2 Beispiel: Kleinbuchstaben werden ignoriert

Die Darstellung des Wortes $w_1 = +yF - xFx - Fy+$ aus dem L-System (V, ω, P) mit $V = \{F, x, y, +, -\}$, $\omega = x$ und $P = \{(x, +yF - xFx - Fy+), (y, -xF + yFy + Fx-)\}$ mit Winkel $\delta = 90°$ ist ein Quadrat mit einer fehlenden Seite:

Zeichen	Interpretation
+	Drehung nach rechts um 90°
y	kein Effekt
F	Bewegung nach vorn um Länge 1 und Zeichnung
-	Drehung nach links um 90°
x	kein Effekt
F	Bewegung nach vorn um Länge 1 und Zeichnung
x	kein Effekt
-	Drehung nach links um 90°
F	Bewegung nach vorn um Länge 1 und Zeichnung
y	kein Effekt
+	Drehung nach rechts um 90°



3.3 Repräsentation in Python

Wir repräsentieren ein L-System mit einer Dataclass `LSystem` in der das Axiom `axiom`, die Ersetzungsregeln `rules` und der Winkel `angle` um den der Roboter sich dreht gespeichert ist.

```
@dataclass
class LSystem:
    axiom: str
    rules: dict[str, str]
    angle: float
```

Ein Objekt vom Typ `LSystem` kann dann durch den Ausdruck

`LSystem(axiom, rules, angle)`

erzeugt werden, wobei für `axiom`, `rules` und `angle` konkrete Werte angegeben werden.

3.3.1 Beispiel

Das L-System (V, ω, P) mit $V = \{F, G, +, -\}$, $\omega = F$ und $P = \{(F, G - F - G), (G, F + G + F)\}$ mit Winkel $\delta = 120°$ erhält man durch

`LSystem('F', {'F': 'G-F-G', 'G': 'F+G+F'}, 120)`

3.3.2 Beispiel

Ein L-System (V, ω, P) mit $V = \{F, x, y, +, -\}$, $\omega = x$ und $P = \{(x, +yF - xFx - Fy+), (y, -xF + yFy + Fx-)\}$ mit Winkel $\delta = 90°$ erhält man durch

`LSystem('x', {'x': '+yF-xFx-Fy+', 'y': '-xF+yFy+Fx-'}, 90)`

3.3.3 Anmerkungen

Weitere Beispiele können Sie in der Datei `main_lindenmayer.py` finden. Beachten Sie, dass das nicht der mathematischen Definition über das Tupel (V, ω, P) entspricht. Die Alphabet V ist implizit durch die Ersetzungsregeln und das Axiom gegeben, daher wird es nicht extra in Python repräsentiert. Zusätzlich wird zu einem gegebenen L-System auch der Winkel δ angegeben, um welchen sich der Roboter bei der Darstellung dreht.

3.3.4 Ersetzungsregeln

Die Menge der Ersetzungsregeln repräsentieren wir wiederuüm durch ein Python `dict`, wobei ein Tupel $(v, w) \in P$ dem Schlüssel-Wert-Paar in einem `dict` entspricht. Beispiel: Die Ersetzungsregeln $P = \{(F, G - F - G), (G, F + G + F)\}$ werden in Python zu

```
{'F': 'G-F-G', 'G': 'F+G+G'}
```

3.3.5 Position

Wir repräsentieren eine Position (x, y) des Roboters im kartesischen Koordinatensystem durch ein Tupel mit zwei Gleitkommazahlen.

```
Position = tuple[float, float]
```

3.3.6 Überprüfen der Ergebnisse

Zustäzlich zu den `doctest` Tests wird eine Datei `main_lindenmayer.py` bereitgestellt in der verschiedene L-Systeme hinterlegt sind. Wird das Modul mit den Argumenten `name` und `iteration` aufgerufen, dann wird das Wort $w_{\texttt{iteration}}$ aus dem L-System mit Namen `name` mithilfe der von Ihnen implementierten Funktionen erzeugt und in Koordinaten übersetzt. Anschließend wird diese Liste von Linien automatisch gezeichnet und grafisch dargestellt. Sie können Ihr Ergebnis mit den Bildern im Ordner `img` vergleichen. Änderungen in der Datei `main_lindenmayer.py` haben keinen Einfluss auf die Bewertung der Aufgabe und sind natürlich auch nicht nötig.

3.4 Aufgaben

3.4.1 Anwenden der Ersetzungsregeln `update_word` (1 Punkt)

Implementieren Sie die Funktion

```
def update_word(word: str, rules: dict[str, str]) -> str:
```

die die Buchstaben in einem gegebenen Wort mit den gegebenen Regeln ersetzt.

3.4.2 Erzeugung eines Wortes der Tiefe n `generate_word` (2 Punkte)

Implementieren Sie die Funktion

```
def generate_word(axiom: str, rules: dict[str, str], iterations: int) -> str:
```

die für ein gegebenes Startwort `axiom` mit den gegebenen Regeln `rules` ein Wort der Tiefe `iterations` erzeugt.

3.4.3 Drehung `turn_right` und `turn_left` (1 Punkt)

Implementieren Sie die Funktionen

```
def turn_right(direction: float, angle: float) -> float:
```

```
def turn_left(direction: float, angle: float) -> float:
```

die für eine gegebene Richtung `direction` $\in [0, 360)$ und einen Winkel `angle` $\in [0, 360)$ den Winkel berechnet, der sich durch eine Drehung um den Winkel `angle` ausgehend vom Winkel `direction` ergibt. Der Wert des Winkels soll in $[0, 360)$ liegen.

3.4.4 Bewegung `move_forward` (1 Punkt)

Implementieren Sie die Funktion

```
def move_forward(pos: Position, direction: float) -> Position:
```

die für eine gegebene Position und Richtung die Position bestimmt, die durch einen Schritt der Länge 1 in Richtung `direction` entsteht. (Erinnern Sie sich an Polarkoordinaten)

3.4.5 Übersetzung eines Wortes in Koordinaten `word_to_lines` (4 Punkte + 2 Bonuspunkte)

Ziel dieser Aufgabe aus einem gegebenen Wort Linien, die der Roboter zeichnet, zu generieren. Die Linien werden dabei durch Listen von (x, y) Koordinaten dargestellt. Wir werden dabei zwei Komplexitätsstufen unterscheiden: L-Systeme ohne Zwischenspeicherung von Stati (ohne `[` und `]`) und L-Systeme mit Speicher (mit `[` und `]`). Implementieren Sie zunächst die Übersetzung von Wörtern aus L-Systemen ohne Speicher und erweitern Sie Ihre Implementierung anschließend für L-Systeme mit Speichern.

Hinweis: Die Docstring Tests für Systeme mit Speicher und die Zusatzaufgabe sind abgetrennt indem mehrere mehrzeilige Zeichenketten am Anfang der Funktion stehen. Um alle Teilaufgaben zu testen müssen alle Zeichenketten zusammengefügt werden. Zur besseren Übersicht ist es empfehlenswert die langen Docstrings in Pycharm einzuklappen.

1. Systeme ohne Speicher (2 Punkte)

   Implementieren Sie die Generatorfunktion

   ```
   def word_to_lines(word: str, angle: float, short: bool = False):
   ```

   für ein Wort `word` aus einem L-System ohne Speicher. Beim Iterieren über einen erzeugten Generator `word_to_lines(word, angle)` soll eine Liste von Koordinaten erzeugt werden, die die Linie beschreibt, die der Roboter für das Wort `word` mit Winkel $\delta =$ `angle` zeichnet. Dabei soll jede Position, die der Roboter besucht, auch in der als Koordinate in der Liste erscheinen.

   Der Parameter `short` hat für diese Teilaufgabe keine Bedeutung. Unabhängig von dessen Wert soll die Funktion einen Generator erzeugen.

2. Systeme mit Speicher (2 Punkte)

   Erweitern Sie die Generatorfunktion

   ```
   def word_to_lines(word: str, angle: float, short: bool = False):
   ```

   für ein Wort `word` aus einem L-System mit Speicher. Beim Iterieren über einen erzeugten Generator `word_to_lines(word, angle)` sollen Listen von Koordinaten erzeugt werden, die die Linien beschreiben, die der Roboter für das Wort `word` mit Winkel $\delta =$ `angle` zeichnet. Dabei soll jede Position, die der Roboter besucht, auch in der als Koordinate in der Liste erscheinen.

   Der Parameter `short` hat für diese Teilaufgabe keine Bedeutung. Unabhängig von dessen Wert soll die Funktion einen Generator erzeugen.

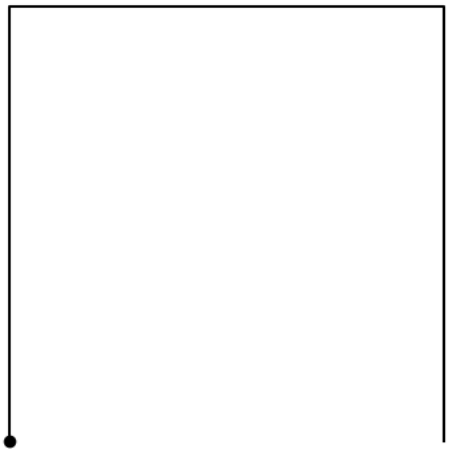
3. Bonusaufgabe: Linien mit wenig Koordinaten (2 Bonuspunkte) Erweitern Sie die Generatorfunktion `word_to_lines`, sodass durch den optionalen Parameter `short` die Liste von Koordinaten verkürzt wird, indem gerade Linien, die durch mehrere Koordinaten beschrieben werden, nur noch durch zwei Koordinaten beschrieben werden.

   Beispiel: FFF wird nicht zu `[(0, 0), (0, 1), (0, 2), (0, 3)]` übersetzt, sondern zu `[(0, 0), (0, 3)]`.

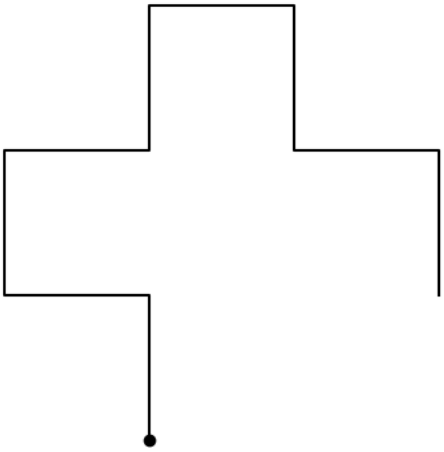
3.4.6 Finden eines L-Systems (1 Punkt)

Bestimmen Sie das L-System, für das für die Tiefen $n \in \{0, 1, 2, 3\}$ die folgenden Linien erzeugt werden. Zur besseren Vergleichbarkeit sind die Startpunkte aller Linien mit einem Punkt markiert. Außerdem haben die einzelnen Liniensegmente die Länge 1 und werden einfach durchlaufen.

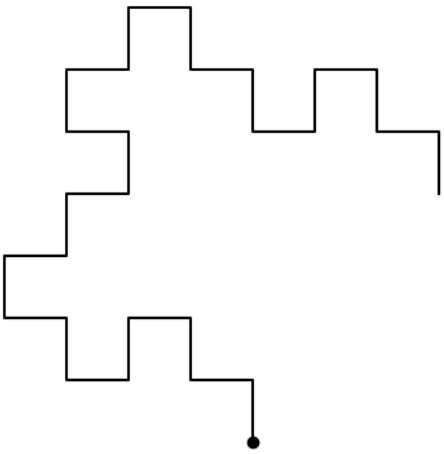
guess 0

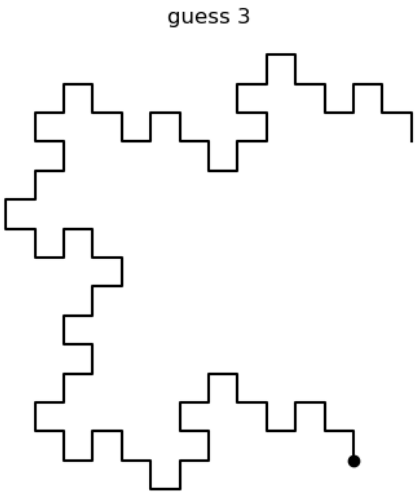


guess 1



guess 2





Implementieren Sie die Funktion

```
def lsystem_guess() -> LSystem:
```

die Ihr gefundenes L-System zurückgibt. Erinnern Sie sich daran, dass ein LSystem durch den folgenden Ausdruck erzeugt wird.

```
LSystem(axiom, rules, angle)
```

Beispiele finden Sie weiter oben in der Anleitung und in der Datei `main_lindenmayer.py`.

Beachten Sie, dass zum Testen eine Referenzimplementierung zur Erzeugung der Koordinaten aus dem zurückgegebenen L-System verwendet wird, sodass die Tests unabhängig von Ihren Implementierungen sind.