

Übung 3

Table of Contents

1. Ermittlung von Rundreisen mit Länge 4	2
2. Ermittlung von einzigartigen Rundreisen	3
3. Berechnung von Durchschnitt und Mittelwert	4
4. Ermittlung von Wegen mit maximaler Distanz (Prozent)	5
5. Gerade Wege.....	6
6. Testen	8

1. Ermittlung von Rundreisen mit Länge 4

Rundreisen können einfach ermittelt werden, indem überprüft wird, ob die Distanz 0 ist und der Schlüssel bzw. die Länge des Dictionaries 4 ist.

```
def roundtrips_with_length_4(walks: Dict):  
    ''' Calculates roundtrips with length of 4 '''  
    print("-- Roundtrips with length 4 --")  
    roundtrips = [walk for walk, distance in walks[4] if distance == 0]  
    print(len(roundtrips))
```

Rundreisen mit der Länge 4:

1379

2. Ermittlung von einzigartigen Rundreisen

Die Comprehensions wurden aufgeteilt, um die Lesbarkeit zu erhöhen. Vorerst werden alle Rundreisen ermittelt, also wo die Distanz 0 beträgt. Dabei sind nur Rundreisen interessant, wo die Länge eine gerade Anzahl ist. Bei N, O, S würde es sich z.B. um keine Rundreise handeln, bei N, O, S, W aber schon.

Mittels `[:10]` können die ersten 10 Rundreisen ermittelt werden.

Für eindeutige Rundreisen wurde das Dictionary in eine einzige Liste umgewandelt. Anschließend wurde `set` aufgerufen, um doppelte Werte zu entfernen. Dazu wurde auch jede Reise in ein Tuple umgewandelt. Zum Schluss wurden diese wieder in eine Liste von Listen konvertiert.

```
def unique_roundtrips(walks: Dict):  
    ''' Calculates all unique round trips '''  
    all_roundtrips = {length: [walk for walk, distance in walks[length] if distance == 0] for  
length in walks if length % 2 == 0}  
  
    print("-- First 10 roundtrips --")  
    first_ten_roundtrips = {length: all_roundtrips[length][:10] for length in all_roundtrips}  
    print(sum(len(first_ten_roundtrips[length]) for length in first_ten_roundtrips))  
  
    print("-- Unique roundtrips --")  
    flattened_roundtrips = [walk for key in all_roundtrips # Iterate over all round trips  
                           for walk in all_roundtrips[key]] # Get walk from each dictionary  
entry  
    unique_roundtrip_set = set(tuple(trip) for trip in flattened_roundtrips) # By creating a set  
all duplicates get removed  
    all_unique_roundtrips = [list(trip) for trip in unique_roundtrip_set] # Convert tuples to  
list  
    print(len(all_unique_roundtrips))
```

Ersten 10 und eindeutige Rundreisen:

```
-- First 10 roundtrips --  
250  
-- Unique roundtrips --  
6451
```

3. Berechnung von Durchschnitt und Mittelwert

Für die Berechnung wurde die Distanz der Reisen extrahiert und `statistics.mean` bzw. `statistics.median` aufgerufen.

```
def average_median(walks: Dict):  
    ''' Calculates average and median for walks of length 5, 10, 15, 20 and 25. '''  
    print("-- Average and median --")  
    for length in [5, 10, 15, 20, 25]:  
        distances = [distance for _, distance in walks[length]]  
        print(f'Average for {length}: {statistics.mean(distances)}')  
        print(f'Median for {length}: {statistics.median(distances)}')
```

Ergebnis für Durchschnitt und Mittelwert:

```
Average for 5: 2.479  
Median for 5: 3.0  
Average for 10: 3.5262  
Median for 10: 4.0  
Average for 15: 4.3734  
Median for 15: 5.0  
Average for 20: 5.0258  
Median for 20: 4.0  
Average for 25: 5.622  
Median for 25: 5.0
```

4. Ermittlung von Wegen mit maximaler Distanz (Prozent)

Dazu wird zu Beginn die Anzahl aller Wege ermittelt, anschließend die Anzahl aller Wege, wo die Distanz gleich der Länge ist. Zum Schluss wurde diese Anzahl durch die aller Wege dividiert und gerundet.

```
def percentage_max_walk_distance(walks: Dict):  
    ''' Calculates the percentage of walks that end at a position with a maximum distance.'''  
    print("-- Percentage of max walks --")  
    walk_count = len([walk for key in walks for walk, _ in walks[key]])  
    max_distance_count = len([walk for key in walks  
                             for walk, distance in walks[key] if distance == key])  
    print(f"{round((max_distance_count / walk_count) * 100, 2)}%")
```

Anteil an maximalen Wegen:

5.37%

5. Gerade Wege

Für die Funktion wird der Iterator durchlaufen und wenn ein Element != dem ersten ist, wird False zurückgegeben, ansonsten True. Bei einem Iterator mit der Länge 0, kann gleich 0 zurückgegeben werden.

```
def check_equal(iterator):  
    ''' Checks if each element in list has the same value'''  
    if len(iterator) == 0:  
        return True  
    for element in iterator:  
        if element != iterator[0]:  
            return False  
    return True
```

Im Wesentlichen wurden die Comprehensions aufgebaut wie bei Aufgabe 3, nur dass zusätzlich check_equal aufgerufen wurde.

```
def straight_walks(walks: Dict):  
    ''' Checks walks for straight walks'''  
    print("-- Check straight walks --")  
    # Check for each walk if it only contains one different element with the check_equal  
    function  
    straight_walks = [walk for key in walks  
                      for walk, _ in walks[key] if check_equal(walk)]  
    straight_unique_walks_set = set(tuple(walk) for walk in straight_walks) # Remove duplicates  
    by creating a set  
    straight_unique_walks = [list(walk) for walk in straight_unique_walks_set] # Change set to  
    list  
    pprint(straight_unique_walks)
```

Gerade Wege:

```
['W', 'W', 'W', 'W', 'W', 'W', 'W'],  
['N', 'N', 'N', 'N', 'N', 'N', 'N'],  
['N', 'N', 'N', 'N', 'N', 'N'],  
['S', 'S', 'S', 'S', 'S'],  
['S', 'S', 'S', 'S', 'S', 'S', 'S', 'S'],  
['E', 'E', 'E'],  
['N', 'N'],  
['W', 'W'],  
['W', 'W', 'W', 'W', 'W', 'W'],  
['N', 'N', 'N'],  
['E'],  
['E', 'E'],  
['S', 'S', 'S'],  
['S', 'S'],  
['E', 'E', 'E', 'E', 'E'],  
['N', 'N', 'N', 'N', 'N', 'N', 'N', 'N'],  
['E', 'E', 'E', 'E'],  
['E', 'E', 'E', 'E', 'E', 'E'],  
['S', 'S', 'S', 'S', 'S', 'S', 'S'],  
['S'],  
['S', 'S', 'S', 'S', 'S', 'S'],  
['W', 'W', 'W', 'W', 'W']]
```

6. Testen

Zum Testen, wurde die Methode zum Erstellen von zufälligen Wegen aufgerufen. Als Parameter wurde die maximale Blocklänge 50 übergeben, für die Wiederholungen muss nichts übergeben werden, da die Standardanzahl sowieso 10000 beträgt.

Anschließend werden die entsprechenden Aufgaben abgearbeitet, wobei für jede eine eigene Funktion erstellt wurde.

```
import Basic_Library

walks = Basic_Library.monte_carlo_walk_analysis(50)
roundtrips_with_length_4(walks)
unique_roundtrips(walks)
average_median(walks)
percentage_max_walk_distance(walks)
straight_walks(walks)
```