Übung 3

Table of Contents

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:

```
1390
```

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 sicht z.B. um keine Rundreise handeln, bei N, O, S, W aber schon.

Für eindeutige Rundreisen wurde jeder Wert in ein Tuple konvertiert und ein Dictionary erstellt.

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

```
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("-- Unique roundtrips --")
    unique_roundtrips = {length: {tuple(trip) for trip in all_roundtrips[length]} for length in
all_roundtrips}
    print(sum(len(unique_roundtrips[length]) for length in unique_roundtrips))

    print("-- First 10 roundtrips --")
    first_ten_roundtrips = {length: [trip for trip in list(unique_roundtrips[length])[:10]] for
length in unique_roundtrips}
    print(sum(len(first_ten_roundtrips[length]) for length in first_ten_roundtrips))
```

Eindeutige Rundreisen:

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

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 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 = {length: [walk for walk, _ in walks[length] if check_equal(walk)] for
length in walks}
    straight_unique_walks = {length: {tuple(walk) for walk in straight_walks[length]} for length
in straight_walks}
    pprint(straight_unique_walks)
```

Gerade Wege:

```
 -- Check straight walks --
{1: {('N',), ('E',), ('S',), ('W',)},
 2: {('N', 'N'), ('E', 'E'), ('W', 'W'), ('S', 'S')},
 3: {('S', 'S', 'S'), ('N', 'N', 'N'), ('W', 'W', 'W'), ('E', 'E', 'E')},
 4: {('E', 'E', 'E', 'E'),
     ('N', 'N', 'N', 'N'),
     ('S', 'S', 'S', 'S'),
     ('W', 'W', 'W', 'W')},
 5: {('E', 'E', 'E', 'E', 'E'),
     ('N', 'N', 'N', 'N', 'N'),
     ('S', 'S', 'S', 'S', 'S'),
     ('W', 'W', 'W', 'W', 'W')},
 6: {('E', 'E', 'E', 'E', 'E', 'E'), ('W', 'W', 'W', 'W', 'W', 'W')},
 7: {('N', 'N', 'N', 'N', 'N', 'N', 'N'), ('E', 'E', 'E', 'E', 'E', 'E', 'E')},
 8: set(),
 9: set(),
 10: set(),
 11: set(),
 12: set(),
 13: set(),
 14: set(),
 15: set(),
 16: set(),
 17: set(),
 18: set(),
 19: set(),
 20: set(),
 21: set(),
 22: set(),
 23: set(),
 24: set(),
 25: set(),
 26: set(),
 27: set(),
 39: set(),
 40: set(),
 41: set(),
 42: set(),
 43: set(),
 44: set(),
 45: set(),
 46: set(),
 47: set(),
 48: set(),
 49: set(),
 50: set()}
```

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)
```