Übung 03

Erstelle 10,000 walks bis zu Länge 50:

```
walks = walk_generator.monte_carlo_walk_analysis(50, 10_000)
```

Roundtrips length 4

Filtere alle Roundtrips mit genau Länge 4:

```
len4Roundtrips = [w for w, dist in walks[4] if dist == 0]
```

Anzahl der Länge 4 Roundtrips: 1451

Unique Roundtrips

Um unique Roundtrips zu bestimmen, werden erst alle Roundtrips in ein Dictionary gespeichert

```
allRoundTrips = {k: [w for w, dist in walks[k] if dist == 0] for k in walks}
```

Um unique Roundtrips zu bestimmen, werden alle Roundtrips einer Länge in ein Set gecastet. Da eine Liste nicht direkt in ein Set gecastet werden kann, muss davor ein Tupel erstellt werden.

```
uniqueRoundTrips = {k: {tuple(w) for w in allRoundTrips[k]}
                                 for k in allRoundTrips}
```

Anzahl unique Roundtrips: 50

Um die ersten 10 Roundtrips zu Filtern, können für jede Roundtriplänge die ersten 10 kopiert werden.

```
first10RoundTrips = {k: [w for w in list(uniqueRoundTrips[k])[:10]] for k in
uniqueRoundTrips}
```

Die ersten 10 unique Roundtrips (Für Übersicht werden nur Daten bis zu Länge 10 angezeigt. Walks mit ungerader Länge werden ausgenommen, da sie keine Roundtrips sein können):

```
{2: [('S', 'N'), ('N', 'S'), ('W', 'E'), ('E', 'W')],
 4: [('N', 'E', 'S', 'W'), ('E', 'N', 'W', 'S'), ('E', 'S', 'N', 'W'), ('S', 'N',
'E', 'W'), ('N', 'S', 'E', 'W'), ('E', 'W', 'S', 'N'), ('S', 'W', 'E', 'N'), ('W',
```

```
'W', 'E', 'E'), ('W', 'S', 'E', 'N'), ('S', 'W', 'N', 'E')],
 6: [('E', 'W', 'N', 'S', 'W', 'E'), ('E', 'W', 'S', 'S', 'N', 'N'), ('S', 'W',
'W', 'N', 'E', 'E'), ('N', 'S', 'N', 'S', 'S', 'N'), ('N', 'S', 'W', 'N', 'E',
'S'), ('N', 'N', 'S', 'E', 'S', 'W'), ('E', 'S', 'S', 'W', 'N', 'N'), ('E', 'S',
'W', 'N', 'E', 'W'), ('S', 'W', 'N', 'E', 'W', 'E'), ('S', 'E', 'W', 'N', 'N',
'S')],
  8: [('S', 'S', 'S', 'N', 'N', 'N', 'N', 'S'), ('N', 'S', 'W', 'E', 'E', 'W',
'E', 'W'), ('N', 'E', 'W', 'S', 'S', 'W', 'N', 'E'), ('S', 'W', 'S', 'E', 'N',
'N', 'N', 'S'), ('E', 'E', 'W', 'S', 'N', 'W', 'S', 'N'), ('W', 'E', 'E', 'E',
'N', 'W', 'S', 'W'), ('N', 'W', 'E', 'E', 'S', 'N', 'S', 'W'), ('N', 'S', 'S',
'W', 'N', 'E', 'E', 'W'), ('S', 'N', 'S', 'E', 'W', 'E', 'W', 'N'), ('E', 'W',
'S', 'S', 'W', 'N', 'E', 'N')],
   10: [('N', 'E', 'N', 'S', 'N', 'S', 'S', 'S', 'N', 'W'), ('E', 'S', 'N', 'W',
'E', 'W', 'S', 'N', 'E', 'W'), ('W', 'E', 'E', 'E', 'S', 'W', 'W', 'S', 'N', 'N'),
('E', 'N', 'N', 'W', 'E', 'S', 'W', 'S', 'S', 'N'), ('N', 'W', 'S', 'S', 'E', 'N',
'W', 'W', 'E', 'E'), ('S', 'E', 'N', 'E', 'S', 'W', 'E', 'W', 'W', 'N'), ('W',
'N', 'E', 'W', 'W', 'E', 'S', 'N', 'S', 'E'), ('N', 'W', 'N', 'E', 'S', 'S', 'N',
'E', 'S', 'W'), ('S', 'N', 'N', 'S', 'E', 'W', 'S', 'S', 'N', 'N'), ('W', 'E',
'S', 'S', 'W', 'E', 'W', 'N', 'N', 'E')]}
```

Zur Berechnung von mean und median werden zwei Funktionen implementiert, die jeweils einen Walk nehmen und die gewünschte Statistik zurückgeben. Für den median muss die Liste an Distanzen zuvor sortiert werden.

```python
def calculate_mean(walk):
    dist = [dist for w, dist in walk]
    return statistics.mean(dist)


def calculate_median(walk):
    dist = [dist for w, dist in walk]
    dist.sort()
    return statistics.median(dist)
```

Ausgabe der Statisiken:

```python
# Gleich für andere Zahlen.
print("List of 5")
print("Mean: " + str(calculate_mean(walks[5])))
print("Median: " + str(calculate_median(walks[5])))
# Ausgabe:
List of 5
Mean: 2.4682
Median: 3.0
List of 10
Mean: 3.5144
Median: 4.0
```

```
List of 15
Mean: 4.3076
Median: 5.0
List of 20
Mean: 4.9994
Median: 4.0
List of 25
Mean: 5.5746
Median: 5.0
```

Max possible distance

Hier werden zuerst alle walks, die die maximale Distanz gehen in ein Dictionary gespeichert. Dieses wird dann verwendet, um den Prozentwert an MaxDistance Walks für jede Blocklänge zu berechnen.

```
maxPossibleDist = {k: [(w, dist) for w, dist in walks[k] if dist == k]
                                  for k in walks}
maxPossibleDistPercentage = {k: (len(maxPossibleDist[k]) / len(walks[k]) * 100)
                             for k in walks}
```

Prozentwert der Walks über maximale Distanz:

```
{1: 100.0,
 2: 75.36,
 3: 43.92,
 4: 23.24,
 5: 12.73,
 6: 5.94,
 7: 2.8,
 8: 1.63,
 9: 0.76,
 10: 0.42,
 11: 0.22,
 12: 0.1,
 13: 0.06,
 14: 0.03,
 15: 0.02,
 16: 0.0,
 17: 0.0,
 ...
 50: 0.0}
```

CheckEquals

Die checkEquals Funktion nimmt ein Iterierbares Objekt. Wenn die Länge > 0 ist und der Wert an einer Stelle ungleich dem Wert an Stelle null ist, wird False zurückgeben.

```
def checkEqual(iterator):
    if len(iterator) == 0:
        return True
    for x in iterator:
        if x != iterator[0]:
            return False
    return True
```

All straight Trips

Um alle geraden Trips zu Filtern, wird die checkEqual Funktion verwendet. Um alle uniqueStraightTrips zu bekommen, werden sie wie bei uniqueRoundTrips in ein Set gecastet.

```
allStraightTrips = {k: [w for w, dist in walks[k] if checkEqual(w)]
                        for k in walks}
uniqueStraightTrips = {k: {tuple(w) for w in allStraightTrips[k]}
                                    for k in allStraightTrips}
```