Übung 1

Table of Contents

1. Lösungsidee

Für die 4 Richtungen muss ein Tupel directions erstellt werden.

Bei der **generate_walk** Methode muss zu Beginn überprüft werden, ob der Eingangsparameter block valide ist. Falls dieser kleiner 1 ist, wird eine Exception geworfen. Danach wird in einer Schleife n mal ein zufälliger Wert von directions mittels yield zurückgegeben, um den Vorteil von Generatoren auszunutzen.
Annahme: Für diese Methode muss ein Generator verwendet werden.

Bei der **decode_walk** werden zwei Variablen für x und y benötigt und in einer Schleife alle Schritte von walk durchlaufen, je nach Richtung wird dann x und y erhöht oder verringert.

Die **distance_manhattan** berechnet die Manhattan-Distance, indem Start und Endpunkt substrahiert werden und davon der Absolutwert berechnet wird.

Die **do_walk** ruft einfach die anderen Methoden auf und gibt walk und distance als Tuple zurück.

Die **monte_carlo_walk_analysis** überprüft zu Beginn die Parameter auf Gültigkeit, ansonsten werden Exceptions geworfen. Anschließend wird in einer Schleife bis zu max_blocks gezählt und die do_walk Methode n Mal ausgeführt. Anschließend wird beim Dictionary die Schrittanzahl als Key gesetzt und für die Werte die ermittelten Wanderungen.

2. Testfälle

Jede Funktion wurde einzeln auf Standardfall und Edge-Case getestet.

2.1. generate_walk

1. Testfall: Aufruf der Methode generate_walk ohne Parameter
2. Testfall: Aufruf der Methode generate_walk mit negativem Wert -1 Exception wird geworfen
3. Testfall: Aufruf der Methode generate_walk mit positivem Wert 1
4. Testfall: Aufruf der Methode generate_walk mit Wert 10

```
def test_generate_walk():

    print('-- test_generate_walk --')

    print('- With default parameter')
    print(list(generate_walk()))

    for value in [-1, 1, 10]:
        print(f'- With value={value}')
        try:
            print(list(generate_walk(value)))
        except ValueError as error:
            print(f'Exception thrown: {error}')
```

2.2. Testfälle zu "decode_walk"

1. Testfall: Aufruf mit leerer Liste
2. Testfall: Aufruf mit befüllter Liste
3. Testfall: Aufruf mit ungültigen Werten Exception wird geworfen

```
def test_decode_walk():

    print('-- test_decode_walk --')

    print('- Empty list')
    print(decode_walk([]))

    print('With list ["N", "N", "E", "E"]')
    print(decode_walk(['N', 'N', 'E', 'E']))

    try:
        print('- With invalid value')
        print(decode_walk(['N','O']))
    except KeyError as error:
         print(f'Exception thrown: {error}')
```

2.3. Testfälle zu "distance_manhattan"

1. Testfall: Mit gleichem Start und Endpunkt Liefert Distanz von 0
2. Testfall: Mit Start Punkt (0, 0) und Endpunkt (2, 3) Liefert Distanz 5

```
def test_distance_manhattan():

    print('-- test_distance_manhattan --')

    print('- With same start and end points')
    print(distance_manhattan((0, 0), (0, 0)))

    print('- With start point(0, 0) end point(2, 3)')
    print(distance_manhattan((0, 0), (2, 3)))
```

2.4. Testfälle zu "do_walk"

1. Testfall: Mit gültigen Werten und Standardmethode distance_manhattan
2. Testfall: Mit gültigen Werten und Lambda-Funktion, welche für Distanz immer 0 liefert Distanz beim Walk ist hier dann logischerweise auch 0

```
def test_do_walk():
    print('-- test_do_walk --')

    print('- With valid parameters')
    print(do_walk(10))

    print('- With fake distance function')
    print(do_walk(5, lambda start, end: 0))
```

2.5. Testfälle zu "monte_carlo_walk_analysis"

1. Testfall: Mit gültigen Paramtern max_blocks=5 & repitions=5
2. Testfall: Mit negativem Parameter für max_blocks Liefert Exception
3. Testfall: Mit negativem Parameter für repetions Liefert Exception

```
def test_monte_carlo_walk_analysis():
    print('-- test_monte_carlo_walk_analysis --')

    print('- With valid parameters')
    print(monte_carlo_walk_analysis(5, 5))

    try:
        print('- Test with invalid parameter for max_blocks')
        print(monte_carlo_walk_analysis(-1))
    except Exception as error:
        print(f'Exception thrown: {error}')

    try:
        print('- Test with invalid parameter for repetition')
        print(monte_carlo_walk_analysis(10, -1))
    except Exception as error:
        print(f'Exception thrown: {error}')
```

2.6. Testergebnisse

```
-- test_generate_walk --
- With default parameter
['N']
- With value=-1
Exception thrown: Block has to be greater zero
- With value=1
['N']
- With value=10
['W', 'N', 'W', 'N', 'N', 'N', 'N', 'N', 'S', 'N']
-- test_decode_walk --
- Empty list
(0, 0)
- With valid parameters
(['N', 'S', 'S', 'N', 'W', 'W', 'W', 'W', 'S', 'N'], 4)
- With fake distance function
(['W', 'E', 'S', 'W', 'W'], 0)
-- test_monte_carlo_walk_analysis --
- With valid parameters
{1: [(['S'], 1), (['S'], 1), (['E'], 1), (['N'], 1), (['S'], 1)], 2: [(['W', 'N'], 2), (['E', 'E'], 2), (['W', 'S'], 2), (['W', 'E'], 0), (['N', 'N'], 2)], 3: [(['S', 'E', 'W'], 1), (['W', 'E', 'N'], 1), (['S', 'E', 'W'], 1), (['S', 'N', 'N'], 1), (['E', 'W', 'S'], 1)], 4: [(['N', 'W', 'W', 'S'], 2), (['N', 'N', 'S', 'E'], 2), (['S', 'N', 'W', 'W'], 2), (['W', 'E', 'E', 'W'], 0), (['E', 'W', 'E', 'E'], 2)], 5: [(['W', 'N', 'S', 'E', 'N'], 1), (['S', 'S', 'W', 'N', 'N'], 1), (['S', 'N', 'E', 'W', 'W'], 1), (['W', 'E', 'W', 'W', 'W'], 3), (['S', 'S', 'N', 'E', 'S'], 3)]}
- Test with invalid parameter for max_blocks
Exception thrown: Max blocks have to be greater zero
- Test with invalid parameter for repetition
Exception thrown: Repetition has to be greater zero
```

3. Quellcode

```
# -*- coding: utf-8 -*-
"""
@author: Andreas Wenzelhuemer
"""

import random

directions = ('N', 'E', 'S', 'W')

def generate_walk(block = 1):
    """
    Generates walk with block count

    Parameters:
        block: Count for walk generation

    Returns:
        Iterable of random directions
    """
    if(block < 1):
        raise ValueError("Block has to be greater zero")

    for _ in range(block):
        yield random.choice(directions)

def decode_walk(walk):
    """
    Calculates end position of given walk

    Parameters:
        walk: List with directions

    Returns:
        Calculated final position after walk
    """
    dx = 0
    dy = 0

    for direction in walk:
        if(direction == 'N'):
            dy += 1
        elif(direction == 'S'):
            dy -= 1
        elif(direction == 'E'):
            dx += 1
        elif(direction == 'W'):
            dx -= 1
        else:
            raise KeyError("Invalid direction")
    return dx, dy


def distance_manhattan(start, end):

    """
```

```
    Calculates manhattan distance from start and end point

    Parameters:
        start: Start point tuple with x and y coordinates
        end: End point tuple with x and y coordinates

    Returns:
        Manhattan distance as an integer value
    """

    return abs(start[0] - end[0]) + abs(start[1] - end[1])


def do_walk(blocks, dist = distance_manhattan):
    """
    Generates walk and calculates distance

    Parameters:
        blocks: Count for walk generation
        dist: Distance calculation method, default is manhattan distance

    Returns:
        Tuple with generated walk and manhattan distance
    """

    walk = list(generate_walk(blocks))
    distance = dist((0, 0), decode_walk(walk))
    return (walk, distance)

def monte_carlo_walk_analysis(max_blocks, repetitions = 10000):

    """
    Generates walks from length 1 to block count with n repetitions

    Parameters:
        max_blocks: Max block count which should be generated
        repetitions: Count how often should each block count generation repeated

    Returns:
        Dictionary with max length as key and generated walks and distances as tuple
    """

    if(max_blocks < 1):
        raise ValueError("Max blocks have to be greater zero")
    if(repetitions < 1):
          raise ValueError("Repetition has to be greater zero")

    walks = {}
    for blocks in range(1, max_blocks + 1):
        walks[blocks] = [do_walk(blocks) for _ in range(repetitions)]
    return walks


def test_generate_walk():

    print('-- test_generate_walk --')

    print('- With default parameter')
    print(list(generate_walk()))
```

```
    for value in [-1, 1, 10]:
        print(f'- With value={value}')
        try:
            print(list(generate_walk(value)))
        except ValueError as error:
            print(f'Exception thrown: {error}')

def test_decode_walk():

    print('-- test_decode_walk --')

    print('- Empty list')
    print(decode_walk([]))

    print('With list ["N", "N", "E", "E"]')
    print(decode_walk(['N', 'N', 'E', 'E']))

    try:
        print('- With invalid value')
        print(decode_walk(['N','O']))
    except KeyError as error:
         print(f'Exception thrown: {error}')

def test_distance_manhattan():

    print('-- test_distance_manhattan --')

    print('- With same start and end points')
    print(distance_manhattan((0, 0), (0, 0)))

    print('- With start point(0, 0) end point(2, 3)')
    print(distance_manhattan((0, 0), (2, 3)))

def test_do_walk():
    print('-- test_do_walk --')

    print('- With valid parameters')
    print(do_walk(10))

    print('- With fake distance function')
    print(do_walk(5, lambda start, end: 0))

def test_monte_carlo_walk_analysis():
    print('-- test_monte_carlo_walk_analysis --')

    print('- With valid parameters')
    print(monte_carlo_walk_analysis(5, 5))

    try:
        print('- Test with invalid parameter for max_blocks')
        print(monte_carlo_walk_analysis(-1))
    except Exception as error:
        print(f'Exception thrown: {error}')

    try:
        print('- Test with invalid parameter for repetition')
        print(monte_carlo_walk_analysis(10, -1))
    except Exception as error:
        print(f'Exception thrown: {error}')
```

```
test_generate_walk()
test_decode_walk()
test_distance_manhattan()
test_do_walk()
test_monte_carlo_walk_analysis()

# print(distance_manhattan((0,0), endPoint))
# print(monte_carlo_walk_analysis(2, 10))
```