I Noten-Tool

Wir wollen ein Programm schreiben, welches eine Datei mit Prüfungsnoten von verschiedenen Fächern einliest. Die Datei enthält pro Fach eine Zeile mit dem Namen des Fachs und den bisher erzielten Noten. Es gibt nur ganze und halbe Noten, also keine 4.6 und die Noten werden jeweils mit einem Leerschlag voneinander getrennt.

Beispiel Datei:

```
Informatik 5 4.5 2
Wirtschaft 6 5.5 5 5.5
```

Es gibt nun pro Fach jeweils noch eine weitere Prüfung bis Semesterende. Wir wollen wissen, welche Note pro Fach mindestens erreicht werden muss, damit das Fach mit einem genügenden Durchschnitt abgeschlossen wird.

Im Beispiel oben wäre das für Informatik eine 4.5 und in Wirtschaft eine 1.

Lösungstipps:

- Jede Zeile separat bearbeiten, also Einlesen und dann die Minimalnote ausgeben
- Methode split verwenden um die Zeile in eine Liste zu verwandeln
- Die Minimalnote kann direkt berechnet oder durch Ausprobieren gefunden werden. Beim Ausprobieren kann mit einer **for**-Schleife für jede mögliche Note von 1 bis 6 der Durchschnitt berechnet werden. Die erste Note, welche zu einem Durchschnitt ≥ 4 führt, ist die Minimalnote, und die **for**-Schleife kann abgebrochen werden z.B. mit **break**.
- Das bestimmen der Minimalnote kann in einer separaten Funktion gemacht werden. So wird der Code übersichtlicher.

```python
def berechne_minimale_note(noten):
    noten_werte = [float(note.replace(',', '.')) for note in noten]  # Korrigiert: Ersetzt Komma durch Punkt für float Konvertierung
    aktueller_durchschnitt = sum(noten_werte) / len(noten_werte)
    for moegliche_note in [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6]:
        neuer_durchschnitt = (sum(noten_werte) + moegliche_note) / (len(noten_werte) + 1)
        if neuer_durchschnitt >= 4:
            return moegliche_note
    return 6  # Falls keine Note gefunden wird, die den Durchschnitt auf >= 4 bringt, wird die höchste Note zurückgegeben

def noten_aus_datei_einlesen(dateipfad):
    with open(dateipfad, 'r', encoding='utf-8') as datei:  # encoding='utf-8' kann nötig sein, abhängig von Ihrem System und der Datei
        for zeile in datei:
            teile = zeile.split()
            fach = teile[0]
            noten = teile[1:]
            minimale_note = berechne_minimale_note(noten)
            print(f"Für das Fach {fach} ist eine minimale Note von {minimale_note} erforderlich, um einen Durchschnitt von mindestens 4.0 zu erreichen.")

# Beispielaufruf mit korrektem Pfad als Argument
noten_aus_datei_einlesen("C:/Users/Marco/Desktop/Übung 6/Noten/noten.txt")
```

II Tic-Tac-Toe

Tic-Tac-Toe ist ein Spiel in dem zwei Spieler in einem 3x3 Spielfeld abwechselnd Kreuze und Kreise setzen. Jemand setzt nur Kreise und jemand nur Kreuze. Ist eine Spalte, eine Zeile oder eine Diagonale mit demselben Symbol gefüllt, hat die Person mit diesem Symbol das Spiel gewonnen.

Das Spielfeld ist im Code eine Liste mit drei Listen mit jeweils drei Zeichen (”x”, ”o”, ”-”) wobei ”-” bedeutet, dass dieses Feld noch nicht ausgefüllt wurde. Eine Liste mit den drei Zeichen ist immer eine Zeile des Spielfeld.
Beispiel:

```
feld = [[”x”, ”x”,”o”],
        [”x”, ”o”,”–”],
        [”o”, ”x”,”–”]]
```

Somit kann das Symbol in Zeile 2 und Spate 1 mit `feld [1][0]` gelesen werden.

Es ist Code gegeben, welcher prüft, ob das Spiel gewonnen wurde und, falls ja, von welchem

Symbol. Der Code hat jedoch Fehler auf vier Zeilen. Die Aufgabe ist es diese zu finden und zu korrigieren. Achtung: Es müssen nur kleine Anpassungen gemacht werden.

```python
def is_winner(field, symbol):
    # count symbol horizontally
    counts = []
    for row in range(3):
        count = 0
        for col in range(3):
            if field[col][row] == symbol:  # Fehler 1
                count = count + 1
        counts.append(count)

    # count symbol vertically
    for col in range(3):
        count = 0
        for row in range(3):
            if field[col][row] == symbol:  # Fehler 2 (identisch zu Fehler 1, aber Kontext ist vertikale Zählung)
                count = count + 1
        counts.append(count)

    # count symbol on first diagonal
    count = 0  # Dies sollte außerhalb der Schleife sein
    for pos in range(3):
        if field[pos][pos] == symbol:
            count = count + 1
    counts.append(count)  # Fehlte

    # count symbol on second diagonal
    count = 0  # Reset für zweite Diagonale
    for pos in range(3):
        if field[2-pos][pos] == symbol:  # Fehler 3
            count = count + 1
    counts.append(count)  # Fehlte

    return max(counts) == 3  # Fehler 4
```

Danach soll in Worten kurz erklärt werden, wieso der **return** Ausdruck am Ende der Funktion `is_winner` das Richtige überprüft.

1. **Fehler in der horizontalen und vertikalen Zählung**: Die Indizes für das Feld müssen getauscht werden, um die korrekte Zeile und Spalte zu referenzieren. Für horizontale Überprüfung sollte es **field[row][col]** sein und für vertikale ebenfalls, aber die Logik bleibt gleich, da der Fehler in der Beschreibung der Indizes lag und nicht in der Verarbeitungslogik selbst.
2. **Diagonale Überprüfungen**: Die Überprüfung der Diagonalen war unvollständig. Nachdem die Diagonale gezählt wurde, muss **count** zu **counts** hinzugefügt werden.
3. **Fehler in der Überprüfung der zweiten Diagonale**: Die Berechnung des Indexes für die zweite Diagonale war fehlerhaft. Es sollte **field[2-pos][pos]** sein.
4. **Endüberprüfung mit max(counts) == 3**: Die Verwendung von **max(counts) == 3** ist korrekt und bedarf keiner Korrektur. Dieser Ausdruck überprüft, ob mindestens eine der Reihen, Spalten oder Diagonalen

genau drei Symbole des gleichen Typs enthält, was bedeutet, dass der Spieler mit diesem Symbol gewonnen hat.

```python
    # count symbol on first diagonal
    count = 0
    for pos in range(3):
        if field[pos][pos] == symbol:
            count += 1
    counts.append(count)

    # count symbol on second diagonal
    count = 0
    for pos in range(3):
        if field[pos][2-pos] == symbol:
            count += 1
    counts.append(count)

    return max(counts) == 3
```

III Namen unkenntlich machen

Wir organisieren einen Event und haben eine Liste mit Namen und weiteren Informationen zu jeder angemeldeten Person. Wir müssen nun jemandem die Tabelle weitergeben, aber die Namen dürfen aus datenschutz-gründen nicht erkenntlich sein. Unsere Aufgabe ist es aus jedem Namen einen eindeutigen String zu erzeugen. Diese Namens-Übersetzung speichern wir als *csv* Datei. Beispiel:

```
Sempach ; as 78 as
Muller ; Name
Dinler ;1111
```

In dieser Aufgabe kannst du selbst entscheiden, wie der User die Namen eingibt (Einlesen einer Datei, Eingabe auf der Kommandzeile, ...). Zudem gibt es keine Vorgaben wie der Name unkenntlich gemacht wird, ausser dass für jeden Namen ein eindeutiger String generiert wird. Das heisst, es gibt keine zwei Namen die in denselben String übersetzt werden.
Also z.B. ”Muster → x7a86” und ”Meier → x7a86” sollte es nicht geben, da beide Namen in x7a86 übersetzt werden. Wenn hier mit Zufallszahlen gearbeitet wird, kann es vorkommen, dass per Zufall gleiche Übersetzungen auftauchen. In diesem Fall wäre das für die Aufgabe ok.

C: > Users > Marco > Desktop > Übung 6 > Hash > hash1.py > ...

```python
import hashlib
import csv

def hash_name(name):
    """Generiert einen eindeutigen Hash für einen Namen."""
    hash_object = hashlib.sha256(name.encode())
    return hash_object.hexdigest()

def main():
    names = []  # Hier könnten die Namen eingelesen werden
    hashed_names = []

    # Pfad für die CSV-Datei
    csv_file_path = 'C:/Users/Marco/Desktop/Übung 6/Hash/hashed_names.csv'  # Beispiel-Pfad, an deine Bedürfnisse anpassen

    # Beispiel: Eingabe der Namen über die Kommandozeile
    while True:
        name = input("Gib einen Namen ein (oder 'ende' zum Beenden): ")
        if name == 'ende':
            break
        names.append(name)

    # Erzeugung der eindeutigen Strings
    for name in names:
        hashed_names.append((name, hash_name(name)))

    # Speichern der Namen und ihrer Hashes in der angegebenen CSV-Datei
    with open(csv_file_path, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['Original Name', 'Hashed Name'])
        writer.writerows(hashed_names)

    print(f"Die Namen wurden erfolgreich gehasht und in '{csv_file_path}' gespeichert.")

if __name__ == "__main__":
    main()
```