Danny Rehl, Programmieren I - Python

Übungsblatt 06

Abgabefrist: Montag, den 02.12.2024, um 23:59 Uhr

Sie finden für dieses Übungsblatt eine Vorlage für Ihre Abgabe zum Herunterladen im Moodle.

Für die Bewertung Ihrer Abgabe zählen die von Ihnen vervollständigten Funktionen im oberen Teil der Vorlage. Schreiben Sie hier nur Ihre Lösungen hin, keine weiteren Befehle außerhalb der Funktionen.

Im unteren Teil finden Sie einen `if __name__ == "__main__"`-Abschnitt, welcher bereits einige Beispielaufrufe zu den Funktionen beinhaltet. Hier können Sie die Befehle beliebig ergänzen oder ersetzen, um Ihre Funktionen zu testen.

Hinweis: Trotz aller Tests muss die hochgeladene `.py`-Datei natürlich immer noch ausführbar bleiben. Beispielsweise wird ein `SyntaxError` im `if __name__ == "__main__"`-Teil in jedem Fall zur Pytest-Bewertung von 0 % führen.

Beginnend mit diesem Übungsblatt soll Ihr Code PEP 8 konform geschrieben sein. Nutzen Sie die Hinweise des Pylint-Scorers, um die verbleibenden Style-Fehler zu finden.

Aufgabe 1 (2 Punkte)

Die Funktion `calculate_digit_sum` enthält einige Verstöße gegen PEP 8, den Style-Guide für Python-Code. Die Befehle werden trotzdem korrekt ausgeführt.

Bereinigen Sie die Style-Fehler, indem Sie PEP 8 anwenden. Ändern Sie also nur die Formatierung des Codes, nicht die Befehle selbst. Sie sollten im Pylint-Scorer keine Meldungen für die Code-Zeilen dieser Aufgabe mehr erhalten, sobald wirklich alle Fehler von Ihnen verbessert wurden.

Bei der Bearbeitung der folgenden Aufgaben sollen zusätzlich keine weiteren neuen Style-Fehler eingebaut werden. Nachdem Sie Ihre Abgabe hochgeladen haben, soll also auf Tutron in der Spalte des Pylint-Scorers "100%" angezeigt werden.

Nutzen Sie diese Aufgabe, um sich nochmal mit den Anforderungen von PEP 8 vertraut zu machen!

Aufgabe 2 (2 Punkte)

Sie haben in Aufgabe 1 die Funktion `calculate_digit_sum` von Style-Fehlern befreit, aber den Code an sich nicht geändert. Nun erweitern wir den Code. Achten Sie darauf, dass Sie nicht neue Style-Fehler einbauen!

Die Funktion berechnet die Quersumme einer natürlichen Zahl, also die Summe der einzelnen Ziffern. Zum Beispiel berechnet `calculate_digit_sum(5237)` die Quersumme folgendermaßen:

```
5 + 2 + 3 + 7 = 17
```

An Fehlerbehandlung fehlt es allerdings: Ruft man die Funktion mit einer negativen Zahl als Parameter auf, erhält man eine `Exception`. Finden Sie heraus, welche `Exception` das ist, welche Zeile im Code dafür verantwortlich ist und ändern Sie die Funktion so, dass explizit diese `Exception` abgefangen wird. (Andere Exceptions sollen also weiterhin auftreten dürfen.)

Fügen Sie dafür einen `try`-`except`-Block ein, sodass die Funktion im Fehlerfall die `Exception` abfängt, abbricht und `None` zurückgibt.

Machen Sie sich hier auch nochmal klar, dass `None` vom Typ `NoneType` ein Singleton ist - also nicht dasselbe wie der String `"None"`.

Aufgabe 3 (2 Punkte)

In der vorherigen Aufgabe haben wir mit einer Funktion gearbeitet, die die Quersumme von Zahlen berechnet. Das heißt, man übergibt eine beliebige Ganzzahl als Funktionsargument und bekommt die entsprechende Quersumme als Rückgabewert der Funktion.

Wir wollen jetzt die Berechnung von Quersummen mehrfach hintereinander ausführen, bis nur noch eine einstellige Zahl übrig ist.

Zum Beispiel bekommt man aus `5237` zunächst:

```
5 + 2 + 3 + 7 = 17
```

Von diesem Zwischenergebnis `17` lässt sich nun erneut die Quersumme bilden:

```
1 + 7 = 8
```

Für dieses Ergebnis `8` (also eine einstellige Zahl) gibt es dann keine sinnvolle Zerlegung mehr - das Endergebnis ist erreicht.

Sie finden in der Vorlage bereits eine Funktion `multiple_digit_sum_iterative`, die diese Berechnung **iterativ** unter Verwendung einer `while`-Schleife ausführt.

Sie sollen nun die Funktion `multiple_digit_sum_recursive` vervollständigen, welche das gleiche Ergebnis liefert - dieses Ergebnis allerdings nicht iterativ sondern **rekursiv** berechnet. Das heißt, die Funktion ruft sich immer wieder selbst auf, bis das gewünschte Ergebnis erreicht ist - und das **ohne** die Verwendung einer `while`- oder `for`-Schleife.

Sie dürfen in Ihrer Implementation von `multiple_digit_sum_recursive` natürlich die bestehende Funktion `calculate_digit_sum` benutzen um die Zwischenergebnisse, also eine (einfache) Quersumme, zu berechnen.

Aufgabe 4 (4 Punkte)

Schreiben Sie eine Funktion `convert_raw_data`, welche die Funktion `calculate_digit_sum` auf die Daten einer `.csv`-Datei anwendet. Die Funktion soll zwei Parameter haben, sodass der Aufruf `convert_raw_data("raw_data.csv", "digit_sums.csv")` die Daten von `raw_data.csv` einliest, die Quersummen aller Einträge berechnet und anschließend das Ergebnis als `digit_sums.csv` speichert.

Die Dateinamen von Eingabe- und Ausgabedatei werden also als Parameter vom Typ String übergeben. `raw_data.csv` und `digit_sums.csv` sind nur beispielhaft gewählte Dateinamen. Benutzen Sie in Ihrer Funktion die Variablen `source` und `destination` für Eingabe- und Ausgabedatei, sodass die Funktion mit beliebigen Werten funktioniert.

Benutzen Sie in Ihrer Implementation die Built-in Function `open` in Kombination mit dem `with` statement. Bei der Auswahl des Modus zum Öffnen der Datei (siehe Dokumentation zum Argument `mode` der Funktion `open`) soll Ihr Code in der Lage sein, vorhandene Dateien zu überschreiben.

Die Eingabedaten liegen CSV-formatiert vor. Jeder Eintrag ist vom nächsten Eintrag durch einen Zeilenumbruch getrennt. Die Werte jeder Zeile sind voneinander mit einem Komma getrennt. Sie finden eine Datei mit Beispieldaten auf der Kursseite zum Download.

Wenn wir beispielsweise annehmen, dass zwei Zeilen in der Input-Datei `raw_data.csv` so aussehen:

```
632444,858
584737196,357728419
```

Dann stehen diese zwei Zeilen also für zwei Einträge mit je zwei Werten `632444` und `858`, beziehungsweise `584737196` und `357728419`. Die passenden Zeilen in `digit_sums.csv` sollen dann entsprechend so aussehen:

```
23,21
50,46
```

Hinweis: Sie müssen die durch Ihre Funktion erzeugte `digit_sums.csv` nicht abgeben oder hochladen, da die CSV-Datei ja durch Ihren Code generiert wurde. Ihre `.py`-Datei genügt also.

Extra: Wenn Sie herausfinden möchten, ob Sie diese Aufgabe richtig bearbeitet haben, wenden Sie Ihre Funktion auf die Beispieldaten der CSV-Datei von der Kursseite an. Schauen Sie sich die Daten der neu erstellten `digit_sums.csv`-Datei einmal genauer an: Öffnen und plotten Sie diese als XY-Punkt- oder Scatterdiagramm zum Beispiel mit Microsoft Excel oder LibreOffice/OpenOffice Calc. Das geht alles auch mit Python, ist aber an dieser Stelle nicht Inhalt der Vorlesung. Wenn es Sie dennoch interessiert, können Sie sich zum Beispiel das Modul `matplotlib.pyplot` aus der SciPy-Sammlung anschauen. Auf der nächsten Seite finden Sie passenden Code dazu.

Exkurs: Ein Beispiel zu Plotten mit Python (keine Punkte)

Python ist eine der beliebtesten Programmiersprachen im Bereich Data Science. Entsprechend gibt es eine Vielzahl von Frameworks und Modulen, welche die Grundinhalte von Python ergänzen. Matplotlib und NumPy aus der SciPy-Sammlung sind zwei besonders relevante Beispiele.

Angenommen, Sie haben bei der vorherigen Aufgabe aus den Beispieldaten der CSV-Datei von der Kursseite `raw_data.csv` mit dem Befehl

```
convert_raw_data("raw_data.csv", "digit_sums.csv")
```

eine Ausgabedatei `digit_sums.csv` erstellt. Dann können Sie sich überzeugen, dass Sie die Aufgabe richtig gelöst haben, indem Sie die neuen Werte einmal plotten:

```
import matplotlib.pyplot as plt
import numpy as np
filename = "digit_sums.csv"
x, y = np.loadtxt(filename, delimiter=",", unpack=True)
plt.plot(x, y, linestyle="", marker=".")
plt.gca().set_aspect("equal")
plt.show()
```

Es ist möglich, dass Sie die Module `matplotlib` und `numpy` vorher erst noch installieren müssen. Das Vorgehen hierzu hängt von Ihrem System ab. Daher hier nochmal der Hinweis, dass dieser Exkurs an dieser Stelle nicht Inhalt der Vorlesung ist, sondern nur als freiwilliger Zusatz gedacht ist.