Computational Thinking WS2023/24 - Abgabe

Autoren:

- Dr. Benedikt Zönnchen
- Prof. Martin Hobelsberger
- Prof. Martin Orehek
- Prof. Benedikt Dietrich

Anmerkungen:

In diesem Notebook werden wir uns Stück für Stück einen Kontext erbauen, um mit Daten besser und bequemer umgehen zu können. Dabei verzichten wir auf Pakete wie

- `numpy`
- `pandas`

und starten stattdessen von Neuem. Für den letzten Teil dieses Notebooks müssen aber beide Pakete installiert sein.

CSV-Dateien

Ein gängiges Format, um Daten zu speichern sind sogenannte CSV-Dateien, wobei CSV für *Comma-Separated Values* steht. In diesen Dateien enthält typischerweise die erste Zeile die Namen der Spalten, durch Kommas getrennt. Alle weiteren Zeilen enthalten die Daten, wiederum jeweils durch Kommas getrennt. Ein Beispiel einer Datei, in welcher Namen, Matrikelnummer, die erzielten Punkte und Noten von Studierenden abgespeichert werden, könnte wie folgt aussehen:

```
Matrikelnummer,Name,Vorname,Punkte,Note
12345678,Müller,Hans,79,2.3
44445555,Musterfrau,Maria,99,1.0
...
```

Der Name der ersten Spalte lautet in diesem Beispiel `Matrikelnummer`, der Name der zweiten Spalte `Name`, usw.. Der Student Hans Müller mit der Matrikelnummer 12345678 hat 79 Punkte erzielt, was der Note 2.3 entspricht.

In den folgenden Aufgaben wollen wir unsere eigene Datenstruktur konstruieren, welche eine CSV-Datei repräsentieren soll. Diese Datenstruktur `data` ist ein Dictionary, welches lauter Listen enthält. Jede Liste repräsentiert eine Spalte der CSV-Datei. Die `keys` des Dictionarys repräsentieren den Spaltennamen.

Die Aufgaben bauen teilweise aufeinander auf. Es ist insbesondere wichtig, dass Sie **Aufgabe 1**, **Aufgabe 2** und **Aufgabe 3** zuerst lösen, da Sie damit die angesprochene

Datenstruktur erzeugen.

In [ ]:
```
import otter
import csv

grader = otter.Notebook('09_abgabe2.ipynb')

path_to_csv_file = './data/test-data.csv'
```

Folgender Code gibt die ersten `n` Zeilen der CSV-Datei `./data/test-data.csv` aus. Dabei verwenden wir aus dem Standardpaket `csv` die Funktion `reader`, welche einen neuen sog. Handle erzeugt, mithilfe dem wir durch die CSV-Datei *iterieren* können.

Hinweis: Debuggen Sie, bevor sie weitermachen, unbedingt folgenden Code Zeile für Zeile und stellen Sie sicher, dass sie verstehen, welche Werte welche Variable bei den einzelnen Durchläufen annehmen.

In [ ]:
```
n = 5
with open(path_to_csv_file, newline='') as csvfile:
    handle = csv.reader(csvfile, delimiter=',')
    count = 1
    for row in handle:
        if count >= n:
            break
        row_as_string = ''
        print(f"Zeile {count}: ")
        for col in row:
            print(col)
        print("")
        count += 1
```

```
Zeile 1:
x
y
name
price

Zeile 2:
1.0
2.0
Toster
5

Zeile 3:
2.3
21
Auto
9
```

Aufgabe 1 (Lesen einer Spalte).

Schreiben Sie eine Funktion `read_column(name, path_to_csv_file)`, welche Ihnen die Werte der Spalte mit dem Namen (Header) `name` als Liste zurückliefert. `path_to_csv_file` ist der Dateipfad zur jeweiligen CSV-Datei.

Angenommen die CSV-Datei sähe wie folgt aus:

```
x,y,name,price
1.0,2.0,Toster,5
2.3,21,Auto,9
```

dann sollte

`read_column('y', './data/test-data.csv')`
folgendes zurückliefern:

`['2.0', '21']`
Gehen Sie bei Ihrer Lösung wie folgt vor:

1. Überlegen Sie sich als erstes, wie sie herausfinden können, welche die gesuchte Spalte ist.
2. Schreiben Sie eine Funktion `read_column(name, path_to_csv_file)`, welche den Index der Spalte mit dem Namen `name` ermittelt und ausgibt. Für einen Aufruf `read_column('y', './data/test-data.csv')` sollte die Funktion also `1`, für `read_column('y', './data/test-data.csv')` die `3` ausgeben.
3. Erweitern Sie nun den Code, sodass Daten der gesuchten Spalte ausgegeben werden, also z.B. `'2.0'` und `'21'` wenn die Funktion mit `read_column('y', './data/test-data.csv')` aufgerufen wird.
4. Erweitern Sie nun Ihren Code und erzeugen zu Beginn der Funktion eine leere Liste `data`. Fügen Sie alle Daten aus der gesuchten Spalte zur Liste hinzu. Die Funktion soll `data` zurückliefern.
5. Erweitern Sie Ihre Funktion nun noch um eine Fehlerbehandlung: Sollte der gesuchte Namen nicht Teil der Daten sein, soll eine Liste zurückgegeben werden.

Hinweis 1: Beachten Sie, dass die eingelesenen Daten alle Strings zurückgeliefert werden. Wir werden uns diesem Problem weiter unten annehmen.

Hinweis 2: Bedenken Sie, dass die erste Zeile, welche Sie einlesen, sich von allen anderen Zeilen unterscheidet, da es sich um die Namen der Spalten handelt!

In [ ]:
```python
def read_column(name, path_to_csv_file):
    data = []

    #Erweiterung 1.5
    try:
        with open(path_to_csv_file, newline='') as csvfile:
            handle = csv.reader(csvfile, delimiter=',')
            headers = next(handle)

            #Erweiterung 1.3
            spalten_index = headers.index(name)

            #Erweiterung 1.4
            data = [row[spalten_index] for row in handle]
            return data

    except ValueError:
```

```
        #print(f'Der gesuchte Name ist nicht vorhanden! \nHier ist eine Liste al
        return data



print(read_column('y', './data/test-data.csv'))
print(read_column('price', './data/test-data.csv'))
```

```
['2.0', '21']
['5', '9']
```

In [ ]:
```
grader.check("q1")
```

Out[ ]:

q1 passed! 🍀

q1 - 1 message: Das sieht gut aus.

Einschub - Funktionen als Referenz

In Python ist es möglich, einer Funktion eine Funktion als Parameter zu übergeben. Hier ein kleiner Beispielcode:

In [ ]:
```
def multiply(a, b):
    return a * b

def add(a, b):
    return a + b

def do_math(function_name, operand1, operand2):
    result = function_name(operand1, operand2)
    return result

result1 = do_math(multiply, 2, 3)   # result1 = funktionsaufruf (Art der Rechnun
result2 = do_math(add, 2, 3)

print(f"{result1=}")
print(f"{result2=}")
```

```
result1=6
result2=5
```

Aufgabe 2 (Erweiterung mit Parser).

Mit der Möglichkeit Funktionen als Parameter zu übergeben erreicht man, dass eine Funktion eine Funktion unserer Wahl aufruft. Das ist für unseren Spaltenleser `read_colum` besonders praktisch, da man so Daten direkt beim Einlesen vorverarbeiten kann.

In der folgenden Aufgabe sollen Sie die oben definierte Funktion `read_column` noch um den optionalen Parameter `parser` erweitern. Übergibt man für `parser` eine Funktion, soll diese Funktion für alle Daten der Spalte angewendet werden.

Beispiel 1:

Der Code

```
def to_int(value):
        return int(value)

    read_column(['price'], './data/test-data.csv', to_int)
```

soll folgendes zurückliefern:

```
[5, 9]
```

Beispiel 2:

Der Code

```
def multiply_by_two(value):
        return int(value) * 2 # Achtung: Beachten Sie die Konvertierung
in int!

    read_column(['price'], './data/test-data.csv', multiply_by_two)
```

soll folgendes zurückliefern:

```
[10, 18]
```

Beispiel 3:

Wird `read_column` nur mit zwei Parametern aufgerufen, soll `parser` den Default-Wert `None` haben. In diesem Fall soll der Wert der Spalte unverändert zurückgegeben werden.

Der Aufruf

```
read_column(['price'], './data/test-data.csv')
```

soll also folgendes zurückliefern:

```
['5', '9']
```

In [ ]:

```
def read_column(name, path_to_csv_file, parser=None):
    data = []

    try:
        with open(path_to_csv_file, newline='') as csvfile:
            handle = csv.reader(csvfile, delimiter=',')
            headers = next(handle)

            spalten_index = headers.index(name)

            if parser is None :                                    # Parser mus
                data = [row[spalten_index] for row in handle]
            else:
                data = [parser(row[spalten_index]) for row in handle]
            return data

    except ValueError:
        #print(f'Der gesuchte Name ist nicht vorhanden! \nHier ist eine Liste al
        return data


# test 1 :
def to_int(value):
```

```
    return int(value)
print(read_column('price', './data/test-data.csv', to_int))


# test 2 :
def multiply_by_two(value):
        return int(value) * 2
print(read_column('price', './data/test-data.csv', multiply_by_two))


# Test 3 :
print(read_column('price', './data/test-data.csv'))     # Geht nur wenn man Pars
```

```
[5, 9]
[10, 18]
['5', '9']
```

Aufgabe 3 (Erzeugen eines Dictionaries aus der CSV-Datei).

Schreiben Sie eine Funktion `read(names, path_to_csv_file, parsers=None)`, welche Ihnen die Spalten, definiert in `names`, in ein Dictionary steckt. Dabei sollen die `keys` des Dictionarys aus den Namen `names` bestehen und die `values` des Dictionarys aus den Spalten (als Listen) bestehen. Die Funktion soll das konstruierte Dictionary zurückliefern.

Mit Hilfe des Parameters `parsers` soll eine Liste an `parser` übergeben werden können. Gibt es für den `i`-ten Namen einen Parser `parsers[i]`, d.h. entspricht der Wert von `parsers[i]` nicht `None`, so sollte jeder Wert `value` der entsprechenden Spalte durch den Wert `parser[i](value)` ersetzt werden.

Beispiele

Für folgende CSV-Datei

```
x,y,name,price
1.0,2.0,Toster,5
2.3,21,Auto,9
```

ergibt

```
data = read(['y', 'x'], path_to_csv_file)
print(data)
```

folgende Ausgabe

```
{'y': ['2.0','21'], 'x': ['1.0','2.3']}
```

und

```
def multiply_by_two(x):
    return 2*x

data = read(['price', 'x'], path_to_csv_file, [multiply_by_two, None])
print(data)
```

führt zu folgender Ausgabe

```
{'price': [10.0,18.0], 'x': [1.0,2.3]}
```

Tipp 1: Sie können (müssen aber nicht) Ihre Funktion `read_column` verwenden.

Tipp 2: Setzen Sie die Funktion erst Mal ohne `parsers` um. Wenn das normale Einlesen der Spalten in das Dictionary funktioniert, erweitern Sie Ihren Code um die Funktionalität `parsers`.

In [ ]:
```
def read(names, path_to_csv_file, parser=None):
    data_dict = {}

    for name in names:
        try:
            parsers = parser[names.index(name)]
        except:
            parsers = None

        data = read_column(name, path_to_csv_file, parsers)
        data_dict[name] = data

    return data_dict


def multiply_by_two(x):
    return 2*int(x)

data = read(['y', 'x'], path_to_csv_file)
print(data)

table = read(['price', 'x'], path_to_csv_file, [multiply_by_two, None])
print(table)
```

```
{'y': ['2.0', '21'], 'x': ['1.0', '2.3']}
{'price': [10, 18], 'x': ['1.0', '2.3']}
```

In [ ]:
```
grader.check("q3")
```

Out[ ]:

q3 passed! 🍀

q3 - 1 message: Das sieht gut aus.

Wir können eine CSV-Datei nun bequem in eine Python-Datenstruktur einlesen. Diese Datenstruktur bezeichnen wir von nun an als `data`.

Aufgabe 4 (Anzahl der Zeilen).

Schreiben Sie eine Funktion `len_data(data)`, welche die Anzahl der Zeilen von `data` zurückliefert.

Der Code

```
print(len_data(read(['y'], './data/test-data.csv')))
```

sollte beispielsweise

```
2
```

zurückliefern.

Existiert der key nicht, oder ist die Liste leer, soll die Funktion `0` zurückliefern.

In [ ]:
```
def len_data(data):
    for x in data:
        if len(data[x]) != 0:
            return len(data[x])
        else:
            return 0
    return 0


print(len_data(read(['y'], path_to_csv_file)))
print(len_data(read(['Fehler'], path_to_csv_file)))
```

```
2
0
```

In [ ]:
```
grader.check("q4")
```

Out[ ]:

q4 passed! ✨

q4 - 1 message: Das sieht gut aus.

Aufgabe 5 (Spaltennamen).

Schreiben Sie eine Funktion `get_names(data)`, welche eine Liste mit allen Spaltennamen unserer Datenstruktur `data` zurückliefert.

Beispielsweise sollte:

`get_names({'x': [1,2], 'name': ['Anna', 'Klaus']})`
`['x', 'name']` oder `['name', 'x']` ausgeben.

In [ ]:
```
def get_names(data):
    names = []
    for x in data:
        names.append(x)
    return names

get_names(read(['y', 'price'], path_to_csv_file))
```

Out[ ]:
```
['y', 'price']
```

In [ ]:
```
grader.check("q5")
```

Out[ ]:

q5 passed! 🍀

q5 - 1 message: Das sieht gut aus.

Aufgabe 6 (Zeilen auswählen).

Schreiben Sie eine Funktion `get_row(data, i)`, welche ein Dictionary zurückliefert, wobei die `keys` die Spaltennamen unserer Datenstruktur `data` sind und die `values` der `i`-te Wert der dazugehörenden Spalte ist.

Beispielsweise sollte

```
my_data = {'x': [1,2,3],
           'name': ['Anna', 'Klaus','Nina']
           }
get_row(my_data, 1)
```

`{'x': 2, 'name': 'Klaus'}` ausgeben.

Ist `i` außerhalb des gültigen Wertebereichs (für das gegebene Beispiel wäre das für `i > 2` der Fall), soll die Funktion ein leeres Dictionary zurückliefern.

In [ ]:
```
def get_row(data, i):
    row_dict = {}
    try:
        for x in data:
            row_dict[x] = data[x][i]
        return row_dict
    except:
        return row_dict


get_row({'x': [1,2,3], 'name': ['Anna', 'Klaus','Nina']}, 1)
```

Out[ ]:
```
{'x': 2, 'name': 'Klaus'}
```

In [ ]:
```
grader.check("q6")
```

Out[ ]:

q6 passed! 🙌

q6 - 1 message: Das sieht gut aus.

Aufgabe 7 (Zeilen filtern).

Beschreiben Sie welche Auswirkungen folgende Funktion `filter_data()` hat.

In [ ]:
```
def keep_all(x):
    return True

def price_is_less_than_nine(x):
    return True if x['price'] < 9 else False

def filter_data(data, predicate = keep_all):
    copy = {name: [] for name in get_names(data)}
    for i in range(len_data(data)):
        row = get_row(data, i)
        if predicate(row):
            for name in row:
                copy[name].append(row[name])
    return copy

data = {'price': [5, 32, 7, 11], 'name': ['Tasse', 'Stuhl', 'Block', 'Koffer']}
filter_data(data, price_is_less_than_nine)
```

Out[ ]:
```
{'price': [5, 7], 'name': ['Tasse', 'Block']}
```

Die funktion "filter_data" kopiert nur die Zeilen aus data, dessen Wert "price" kleiner 9 ist.

Aufgabe 8 (Spalten aggregieren).

Um Daten zu analysieren wollen wir verschiedene Operationen über Spalten durchführen, welche ausschließlich numerische Werte enthalten. Jede dieser Operationen liefert genau einen numerischen Wert zurück.

Schreiben Sie folgende Funktionen:

- `sum_col(data, col_name)` : gibt die Summe aller Spalteneinträge mit dem Namen `name` zurück
- `avg_col(data, col_name)` : gibt den Durchschnittswert der Spalteneinträge mit dem Namen `name` zurück
- `max_col(data, col_name)` : gibt den größten Wert der Spalteneinträge mit dem Namen `name` zurück
- `min_col(data, col_name)` : gibt den kleinsten Wert der Spalteneinträge mit dem Namen `name` zurück

Empfehlung: Verzichten Sie bei Ihrer Implementierung auf die Verwendung der `Python` -Built-in Funktionen `sum` , `max` , `min` . Dadurch lernen Sie bei der Umsetzung der Aufgabe deutlich mehr.

In [ ]:
```python
def sum_col(data, col_name):
    summe = 0
    for i in data[col_name]:
        summe += int(i)
    return summe

def avg_col(data,col_name):
    summe = 0
    for i in data[col_name]:
        summe += int(i)

    avg_summe = summe / len(data[col_name])
    return avg_summe

def max_col(data, col_name):
    max_value = None

    for value in data[col_name]:
        try:                                # geht nur mit Try sonst wird bei der 1.
            value = int(value)
        except ValueError:
            continue

        if max_value is None or value > max_value:
            max_value = value

    return max_value
```

```python
def min_col(data, col_name):
    min_value = None

    for value in data[col_name]:
        try:                                # geht nur mit Try sonst wird bei der 1.
            value = int(value)
        except ValueError:
            continue

        if min_value is None or value < min_value:
            min_value = value

    return min_value


data = {'price': [5, 32, 7, 11], 'name': ['Tasse', 'Stuhl', 'Block', 'Koffer']}
print(sum_col(data, 'price'))
print(avg_col(data, 'price'))
print(max_col(data, 'price'))
print(min_col(data, 'price'))
```

```
55
13.75
32
5
```

In [ ]:
```python
grader.check("q8")
```

Out[ ]:

q8 passed! 🌈

q8 - 1 message: Das sieht gut aus.

q8 - 2 message: Das sieht gut aus.

q8 - 3 message: Das sieht gut aus.

q8 - 4 message: Das sieht gut aus.

Aufgabe 9 (Datenverarbeitung).

Nun wollen wir die geschriebenen Funktionen nutzen, um einen einfachen Datensatz zu verarbeiten. Gegeben ist ein von ChatGPT erzeugter Datensatz `./data/exam_results.csv`. Dieser beinhaltet Name, ID und erreichte Punkte von Studierenden in einer Prüfung.

Werten Sie die Daten mit Hilfe der geschriebenen Funktionen wie folgt aus:

1. Bestimmen Sie die durchschnittlich erreichte Punktezahl und geben diese aus.
2. Bestimmen Sie die minimal erreichte Punktezahl und geben diese aus.
3. Bestimmen Sie die maximal erreichte Punktezahl und geben diese aus.
4. Berechnen Sie für alle Student:innen die Note. Schreiben Sie hierfür eine Funktion `points_to_grade(points)`, welche folgendes Bewertungsschema umsetzt und für eine (!) übergebene Punktezahl die zugehörige Note zurückliefert:

| von einschließlich | bis einschließlich | Note | |---|---|---| | 0 | 49 | 5.0 | | 50 | 62 | 4.0 | | 63 | 75 | 3.0 | | 76 | 89 | 2.0 | | 90 | 100 | 1.0 |

5. Ermitteln Sie die Durchschnittsnote der Prüfung.

6. Ermitteln Sie die Häufigkeit jeder einzelnen Noten, d.h. wie oft gab es eine 1.0, wie oft eine 2.0, usw.

Das Ergebnis Ihrer Auswertesoftware könnte z.B. wie folgt aussehen:

```
Die durchschnittlich erreichte Punktezahl beträgt: 81.05
Die minimal erreichte Punktezahl beträgt: 59
Die maximal erreichte Punktezahl beträgt: 100

Die durchschnittlich erreichte Note beträgt: 2.10

Häufigkeiten der einzelnen Noten:
1.0: 21
2.0: 30
3.0: 23
4.0: 3
5.0: 0
```

In [ ]:

```python
def avg_points(data, col_name="Points"):
    data = read([col_name], data)
    avg_wert = avg_col(data, col_name)
    average = round(avg_wert, 2)
    print(f'Die durchschnittlich erreichte Punktezahl beträgt: {average}')

def min_points(data, col_name="Points"):
    data = read([col_name], data)
    min_wert = min_col(data, col_name)
    print(f'Die minimal erreichte Punktezahl beträgt: {min_wert}')

def max_points(data, col_name="Points"):
    data = read([col_name], data)
    max_wert = max_col(data, col_name)
    print(f'Die maximal erreichte Punktezahl beträgt: {max_wert}')


def points_to_grade(points):
    noten = [1.0, 2.0, 3.0, 4.0, 5.0]
    x = int(points)
    try:
        if x <= 49:
            return noten[4]
        elif 49 < x <= 62:
            return noten[3]
        elif 62 < x <= 75:
            return noten[2]
        elif 75 < x <= 89:
            return noten[1]
        elif 89 < x <= 100:
            return noten[0]
        else:
            print(f'Die Punkte anzahl {x} ist nicht zulässig.')
    except:
```

```
        return


def avg_grade(data, col_name='Points'):
    grades = {}
    grades['Grades'] = read_column(col_name, data, points_to_grade)
    average_grade =  avg_col(grades, 'Grades')
    average_grade =  round(average_grade, 2)
    print(f"Die durchschnittliche Note: {average_grade}")
    return grades


def amount_grades(data, col_name = 'Grades'):
    data = avg_grade('./data/exam_results.csv')
    grade_amount = {}
    print("Häufigkeiten der einzelnen Noten:")

    for n in range (1, 6):
        count = 0
        for value in data[col_name]:
            if int(value) == n:
                count += 1
        grade_amount[float(n)] = count

    for value in grade_amount:
        print(f"{value}: {grade_amount[value]}")




avg_points('./data/exam_results.csv')
min_points('./data/exam_results.csv')
max_points('./data/exam_results.csv')
print(f'Der Schüler hat die Note: {points_to_grade(62)}')
#avg_grade('./data/exam_results.csv')
amount_grades('./data/exam_results.csv')
```

```
Die durchschnittlich erreichte Punktezahl beträgt: 81.05
Die minimal erreichte Punktezahl beträgt: 59
Die maximal erreichte Punktezahl beträgt: 100
Der Schüler hat die Note: 4.0
Die durchschnittliche Note: 2.1
Häufigkeiten der einzelnen Noten:
1.0: 21
2.0: 30
3.0: 23
4.0: 3
5.0: 0
```

Bonusaufgabe

Die Bearbeitung folgender Aufgabe ist freiwillig gedacht. Für eine erfolgreiche Abgabe ist die Lösung dieser Aufgabe nicht notwendig.

Einschub - Lambdas

In den bisherigen Aufgaben haben wir Funktionen als Parameter übergeben. Hierfür haben wir zunächst eine eigene Funktion definiert und dann den Namen der Funktion als Parameter z.B. an `read_column` übergeben.

Eine deutlich kürzere Schreibweise, um Funktionen als Parameter zu übergeben sind sog. Lambdas. Lambdas sind kleine anonyme Funktionen, d.h. Sie haben keinen Funktionsnamen. Die Syntax lautet dabei wie folgt:

`lambda arguments: expression`
Hier ein einfaches Beispiel, wie Lambdas verwendet werden können:

In [ ]:
```python
def add(x, y):
    return x + y

def do_math(function, x, y):
    return function(x, y)

# Mit Hilfe der Funktion
result_function = add(2, 3)

# Mit Hilfe einer Lambda-Funktion
result_lambda = do_math(lambda x, y: x + y, 2, 3)

print(result_function)
print(result_lambda)
```

Aufgabe 10 (Datenvisualisierung).

Mithilfe unserer bisher implementierten Funktionen sind wir in der Lage erste Analysen eines Datensatzes durchzuführen.

Der Datensatz `./data/GlobalTemperature.csv` enthält Informationen über die globale Temperaturentwicklung von 1750 bis 2015. Öffnen Sie den Datensatz (z.B. mit Excel oder einem Texteditor) und sehen Sie sich dessen Struktur an. Wir möchten daraus einen Graphen/Plot erzeugen , der uns die mittlere (Land)-Temperatur pro Jahr darstellt. D.h. auf der x-Achse sollen die Jahre angetragen werden und auf der y-Achse die mittlere Temperatur in jenem Jahr. Diese Informationen stehen in der Spalte `LandAverageTemperature` (Temperaturwerte allerdings pro Monat) und `dt` (allerdings Datum mit Tag, Monat und Jahr).

Wir wollen diese Informationen in unserer Datenstruktur (Dictionary aus Listen) abspeichern, sodass wir diese weiter verarbeiten können. Schreiben Sie hierfür eine Funktion `compute_global_mean_temperature(data)` die Ihnen folgendes Dictionary zurückliefert:

`data_temperature = {'Year': [1750, 1751, ..., 2015], 'Mean Temperature': [...]}`
dabei beinhalte ist `data_temperature['Mean Temperature']` eine Liste aus `float` und beinhaltet die mittleren Jahrestemperaturen. `data_temperature['Year']` ist hingegen eine Liste aus `int` und beinhaltet die dazu passenden Jahre.

Sie können davon ausgehen, dass der Datensatz für jedes Jahr mindestens einen Wert enthält.

Der darauffolgende Code, welcher einen Plot erzeugt, sollte, nachdem Sie die Funktione implementiert haben, funktionieren.

Hinweise: Es gibt Monate in denen kein Wert eingetragen ist, diese filtern wir bereits im folgenden Code. Außerdem wandeln wir bereits das Datum in der Form `'1781-10-01'` in eine ganze Zahl `1781` um. Dies müssen Sie nicht mehr machen. Ganz am Ende des Notebooks plotten wir den Graphen mit `Pandas`. Hier sehen Sie wie der Plot aussehen sollte.

In [ ]:
```
data = read(['dt', 'LandAverageTemperature'], './data/GlobalTemperatures.csv', [
    lambda date: int(date.split('-')[0]),
    lambda temperature: float(temperature) if temperature != '' else 'NaN'])

# filter rows without values
data = filter_data(data, lambda row: type(row['LandAverageTemperature']) == floa
```

In [ ]:
```
def compute_global_mean_temperature(data):
    ...
    return {'Year': years, 'Mean Temperature': temperature_per_year}
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

data_temperature = compute_global_mean_temperature(data)

first_year = min_col(data_temperature, 'Year')
mean_first = 1951-first_year
mean_last = 1980-first_year
corresponding = data_temperature['Mean Temperature'][mean_first:mean_last+1]
corresponding_mean = sum(corresponding) / len(corresponding)


plt.plot(data_temperature['Year'], np.array(data_temperature['Mean Temperature']

plt.title('Globale mittlere Temperaturabweichung im Vergleich zum Mittel zwische
plt.xlabel('Jahr')
plt.ylabel('Temperatur')
```

In [ ]:
```
grader.check("q10")
```

Im folgenden sehen Sie eine Möglichkeit wie wir die gleiche Darstellung mit `Pandas` erzeugen können.

In [ ]:
```
import pandas as pd

df = pd.read_csv('./data/GlobalTemperatures.csv')
df = df.dropna(subset=['LandAverageTemperature'])
df['Year'] = df['dt'].transform(lambda date: int(date.split('-')[0]))

mean = df[(df['Year'] >= 1951) & (df['Year'] <= 1980)]['LandAverageTemperature']
```

```
df = df.groupby(['Year']).mean()

df['Temperaturabweichung'] = df['LandAverageTemperature'].transform(lambda x: x

# plot it
df['Temperaturabweichung'].plot(ylabel='Temperatur', xlabel='Jahr', title= 'Glob
```

Abgabe **Please save before exporting!**

Dieses Notebook ist eine **Abgabe**! Zur erfolgreichen Abgabe wird erwartet, dass Sie die Aufgabe in Moodle hochgeladen und mit Ihrem Praktikumsleiter durchgesprochen haben. Zum Hochladen in Moodle führen Sie alles von oben nach unten aus, speichern Sie Ihr Notebook und laden Sie die generierte .zip-Datei in Moodle hoch.

In [ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False)
```

Your submission has been exported. Click here to download the zip file.