



Dateiverarbeitung

Teil 2

- 1. Dateipositionierung festlegen/abfragen
- 2. CSV-Dateien erstellen/auslesen
- 3. CSV-Dateien schreiben
- 4. CSV-Dateien und Dictionaries
- 5. Hausübungsbeispiel



1.1. Positionierung festlegen

```
f.seek(offset, from_what)
```

Die `seek()` Methode wird verwendet, um die Position des Dateihandles auf eine bestimmte Position zu ändern. Das Dateihandle ist wie ein Cursor, der definiert, von wo aus die Daten in der Datei gelesen oder geschrieben werden müssen, wobei `offset` die Anzahl der Vorwärtsrückungen angibt und `from_what` von welcher Position aus diese Weiterrückung vollzogen wird. `f.seek()` liefert keinen Rückgabewert.

Akzeptierte `from_what` Werte: (Optionen 1 und 2 können nur bei Binärdateien verwendet werden!)

0: Setzt den Referenzpunkt am Anfang der Datei = Standardwert

1: Setzt den Referenzpunkt auf die aktuelle Dateiposition

2: Legt den Referenzpunkt am Ende der Datei fest

Bsp:

```
f.seek(12) # 2. Parameter standardmäßig 0, weitergerückt wird hier  
           # 12 Bytes (12 Zeichen bei txt-Files) vom Anfang der
```

Datei.

1.2. Positionierung abfragen + Bsp.



```
f.tell()
```

Die Methode *tell()* liefert die aktuelle Position in der Datei.

Bsp:

```
fo = open("mitarbeiter.txt", "r+")
str = fo.read(5) # 5 Zeichen werden gelesen
print("Eingelesener String: ", str)
```

```
# Aktuelle Position erfragen
position = fo.tell()
print("Aktuelle Dateiposition: ", position)
```

```
# Repositionierung wieder vom Dateianfang weg
position = fo.seek(0, 0); # 2. Parameter könnte weggelassen werden
str = fo.read(10)
print("Neuer eingelesener String: ", str)
fo.close()
```

```
#Ausgabe:
```

```
Eingelesener String: Archi
```

```
Aktuelle Dateiposition: 5
```

```
Neuer eingelesener String: Architekt:
```

Übung: Probiere diesen Code aus. Die Datei "mitarbeiter.txt" ist vom letzten Mal noch unter Dateien gespeichert.



1.3. Beispiel im Binärmodus

Um die Positionierung nicht nur vom Dateianfang festlegen zu können, müssen wir eine Datei im Binärmodus öffnen. Hier ein entsprechendes Beispiel:

```
# Öffnen der Textdatei im Binärmodus  
f = open("mitarbeiter.txt", "rb")
```

```
# Positionierung 10 Bytes vom Ende der Datei weg  
f.seek(-10, 2)
```

```
# Aktuelle Position ausgeben (wird vom Anfang gezählt)  
print(f.tell())
```

```
# Um Text ausgeben zu können, muss Binärcode in utf-8 decodiert werden.  
print(f.readline().decode('utf-8'))
```

```
# Hinweis: '\n' ebenfalls ein Zeichen von den 10
```

```
f.close()
```

```
#Ausgabe:
```

```
141
```

```
rwendung
```

Übung: Probiere diesen Code wieder aus,

1.4. Bsp. Sortieren (Datei ändern Teil1)



In diesem Beispiel liegt eine Datei `numbers.txt` mit einer Zahl pro Zeile vor.
Diese Zahlen werden gelesen und aufsteigend sortiert am Ende nochmals angehängt.

Die Sortierung soll händisch mit einem Selection-Sort durchgeführt werden.
Hierfür brauchen wir folgende Funktionen:

```
def selection_sort(lst):  
    """ Durcharbeiten aller Zahlen """  
    for i in range(len(lst)-1):  
        k = find_min(lst,i)  
        swap(lst,k,i)
```

```
def swap(lst, x,y):  
    """ Tauschen von zwei Werten """  
    lst[x], lst[y] = lst[y], lst[x]
```

```
# Minimum finden  
def find_min(lst,i):  
    """ Minimum finden """  
    k = i  
    for o in range(i+1,len(lst)):  
        if lst[o] < lst[k]:  
            k = o  
    return k
```

1.4. Bsp. Sortieren (Datei ändern Teil2)



Nach dem Lesen d. Zeilen ist die Dateiposition am Ende der Datei und das Schreiben kann erfolgreich am Ende durchgeführt werden. Um danach alle Dateiinhalte nochmals auszugeben, muss an die Anfangsposition der Datei gesprungen werden.

```
with open("numbers.txt",mode='r+',encoding="utf-8") as f:
    numbers = []
    numbers = f.readlines() # Nummern in Liste lesen
    for i in range(0, len(numbers)):
        numbers[i] = int(numbers[i]) # str in int umwandeln
    selection sort(numbers) # Nummern aufsteigend sortieren
    for i in range(0, len(numbers)):
        numbers[i] = str(numbers[i]) + "\n" # zurückverwandl. in str
    f.write("\nSortierte Nummern:\n") # Überschrift schreiben
    f.writelines(numbers) # neue sort. Nummern zeilenweise schreiben
    print("Datei erfolgreich geschrieben!\nDateiinhalte nun:\n\nOriginalzeilen:")
    f.seek(0) # Dateipos. auf Anfang setzen um alle Inhalte auslesen zu können.
    print(f.read())
```

Übung: Probiere diesen Code aus. Die Datei "numbers.txt" liegt unter Dateien. Hinweis: f.tell kann beim Testen hilfreich sein. Wichtig: Zeilensprung = 2 Bytes \n\n

2.1. CSV-Dateien erstellen



Eine CSV Datei ist eine Textdatei, in welcher die Daten ähnlich wie in einer Tabelle abgelegt werden können, d.h. man besitzt Zeilen und Spalten. Eine Spalte wird dabei durch einen Separator gekennzeichnet. Als Separator kann jedes beliebige Zeichen dienen, jedoch hat sich „eingebürgert“ das ein Semikolon genutzt wird. Sehr häufig liegen auch .csv-Dateien mit Komma als Separationszeichen vor.

Legen wir uns für das nächste Beispiel eine kleine Liste mit Vorname, Alter und Nachname im Excel an. Wir speichern diese Datei als Dateityp “CSV UTF8 (durch Trennzeichen getrennt)” ab.

	A	B	C
1	Christian	35	männlich
2	Victoria	32	weiblich
3	Moritz	43	männlich
4	Sabine	75	weiblich
5	Norbert	61	männlich
6	Gustav	37	männlich
7			



```
datei_personen_utf8 - Editor
Datei Bearbeiten Format Ansicht Hilfe
Christian; 35; männlich
Victoria; 32; weiblich
Moritz; 43; männlich
Sabine; 75; weiblich
Norbert; 61; männlich
Gustav; 37; männlich
Ze 1, S 100% Windows (CRLF) UTF-8 mit BOM
```

Übung: Erstelle nun selber diese Tabelle im Excel und speichere sie mit dem Namen "datei_personen_utf8" als CSV UTF8 Datei.

2.2. CSV-Dateien auslesen



Schleife über Fileobjekt

Vorerst werden wir die Datei im herkömmlichen Sinn auslesen:

```
with open("datei personen utf8.csv", "r", encoding="utf-8") as datei:
    # für jede Zeile in der Datei...
    for name in datei:
        # Lesen der Elemente in Liste - Trennzeichen ;
        # Entfernen der nicht sichtbaren und Leerzeichen am Anfang und Ende
        zeile = name.strip().split(";")
        # Zuweisen der Listenelemente in eigene Variablen
        vorname = zeile[0]
        alter = zeile[1]
        geschlecht = zeile[2]
        # Ausgabe der Werte in einem Satz.
        print(vorname, "ist", geschlecht, "und", alter, "Jahre alt.", sep="
")
```

Übung: Teste nun diesen Code mit deiner selbst erstellten .csv-Datei



2.3. CSV-Dateien auslesen

Auslesen über csv-Modul

Für die Arbeit mit CSV-Dateien gibt es in Python ein eingebautes Modul namens csv.

```
import csv
```

Zum Lesen verwenden wir die `reader()` Methode, welche ein iterierbares Reader-Objekt zurückliefert.

```
csvobj = csv.reader(file)
```

Über dieses kann man dann mittels Schleife die Daten auslesen.

```
for lines in csvFile:  
    print(lines)
```

2.4. Bsp. CSV-Datei auslesen



Die Methode `reader()` geht von einem Komma als Trennungszeichen im csv-File aus, was beim Auslesen unserer CSV-Datei zu Problemen führt.

```
import csv

with open("datei_personen_utf8.csv", "r", encoding="utf-8") as file:
    # reading the CSV file
    csvFile = csv.reader(file)
    # Zeilen einfach ausgeben
    for lines in csvFile:
        print(lines)
```

Übung: In dieser Variante wird das Trennungszeichen ";" nicht erkannt und die gesamte Zeile als Element in die Liste gespeichert.

Damit dieser Code auch mit unserer Datei funktioniert, muss einfach ein zusätzlicher Parameter beim Aufruf der `reader()`-Methode angeführt werden, der das Trennzeichen angibt.

```
csvFile = csv.reader(file, delimiter=";")
```

2.5. CSV Dialektparameter



In CSV-Modulen kann ein optionaler Dialektparameter angegeben werden, mit dem eine Reihe von Parametern definiert werden, die für ein bestimmtes CSV-Format spezifisch sind. Mit der Methode `register_dialect` kann man sich seinen eigenen Dialekt definieren.

Bsp:

```
csv.register_dialect(  
    'mydialect',  
    delimiter = ',',  
    quotechar = '"',  
    skipinitialspace = True,  
    lineterminator = '\r\n',  
    quoting = csv.QUOTE_MINIMAL)
```

Beim Reader-Objekt können wir den Dialekt wie folgt angeben:

```
csvFile = csv.reader(file, dialect="mydialect")
```

2.6 Bsp. CSV-Datei auslesen/anzeigen



Auf der Internetseite <http://blog.wenzlaff.de/?p=5732> kann man sich CSV-Dateien generieren, die für das Testen mit vielen Datenmengen herangezogen werden können. Eine entsprechend exportierte Datei namens "convertcsv.csv" wird nun für das nächste Beispiel herangezogen.

Aufgabenstellung:

- Anzahl Zeilen ermitteln
- auslesen der Daten und
- ausgeben der Elemente
in einzelnen Zeilen

```

1;Rosetta;Graham;Clayton Miles;ivezafa@liotupe.br;-77.84895;-2.45266;sah.uz
2;Louise;Flores;Lester Banks;ladru@ho.ci;6.02553;95.44752;loofhi.zw
3;Travis;Murray;Albert Fox;kowu@co.sy;34.82745;-125.95297;zos.th
4;Landon;Lowe;Danny Farmer;lajjedic@ihwejek.co;62.97625;110.41307;wobtebo.br
5;Blanche;Cook;Millie Terry;docarsu@litam.pe;67.22968;-83.6553;elaegi.edu
6;Mary;Sanders;Nellie Dean;roeciewo@la.tg;6.49575;-45.44036;oz.br
7;Lucinda;Wise;Stanley Briggs;koricz@vizjivge.pn;-46.48371;-49.86802;keb.hu
8;Eula;Sharp;Jason Day;ofcefpd@ikoriw.wf;-8.77384;119.45273;fazerot.ye
9;Coria;Rogers;Addie Perez;cuzwo@fazdo.ae;38.51847;46.34426;tamtas.dz
10;Esther;Daniel;Carl Ferguson;pa@kelropif.br;67.79458;-18.23251;pegaliofe.af
11;Craig;Aguilar;Randy Shaw;etewi@ro.uy;-46.7415;127.71192;mep.zm
12;Harriett;Henry;Howard Myers;ovodaf@rifricbam.ag;-20.35546;107.87631;ozluh.com
13;Nancy;Vargas;Hallie Holmes;jj@ufapud.mm;-47.9689;-105.23138;rubkop.by
14;Donald;Perry;Mamie Parsons;tuvuf@nukicupi.tp;66.04273;81.81376;godwi.li
15;Leo;Garza;Earl Hill;dizva@zejiehu.sa;57.51181;-84.48471;lu.gr

```

2.6 Bsp. CSV-Datei auslesen/anzeigen



Der fertige Python-Code zu der gegebenen Aufgabenstellung 2.6. liegt in der Datei `datei_zahlwerte Ausgabe.py` vor. Diese befindet sich im Register Dateien und kann nun getestet werden. Zum Verständnis des Codes seien hier noch zwei Details erwähnt:

Die `next()` Funktion gibt die aktuelle Zeile in einer Liste zurück und bringt den Iterator in die nächste Zeile.

```
lst= next(csvreaderobj)
```

Das Reader-Objekt `line_num` liefert die Anzahl der durchlaufenen (iterierten) Zeilen.

```
cr.line_num
```

Übung: Teste nun die Python-Datei gemeinsam mit der `convertcsv.csv` Datei. Nimm eigenständig Änderungen/Erweiterungen vor, um den Code gänzlich zu verstehen.



3.1. CSV-Dateien schreiben

Schreiben über csv-Modul

Im csv-Modul gibt es für das Schreiben die Methoden `writerow` und `writerows`.

Um die erste Zeile zu schreiben, die nichts anderes als die Feldnamen beinhaltet, verwendet man `writerow`:

```
csvwriter.writerow(titel)
```

Für alle weiteren Zeilen, um mehrere Zeilen gleichzeitig zu schreiben:

```
csvwriter.writerows(zeilen)
```

3.2. Bsp. CSV-Dateien schreiben



Im folgenden Beispiel werden vordefinierte Listen in ein CSV-File geschrieben:

```
import csv

titel = ["Land", "Hauptstadt", "Einwohner", "Wahrzeichen"]

zeilen=[["Wien", "Wien", 1911728, "Stephansdom"],
        ["Tirol", "Innsbruck", 757852, "Goldenes Dachl"],
        ["Kärnten", "Klagenfurt", 561390, "Lindwurm"],
        ["Steiermark", "Graz", 1246576, "Uhrturm"]]

filename = "bundeslaender.csv"

with open(filename, "w") as csvfile:
    # CSV-Writer Objekt erstellen
    csvwriter = csv.writer(csvfile, delimiter=";", lineterminator = '\r')
    #Felder schreiben
    csvwriter.writerow(titel)
    csvwriter.writerows(zeilen)
```

Übung: Teste dieses Programm. Die Datei `bundeslaender.csv` wird automatisch angelegt, wenn sie nicht vorhanden ist.

4.1. CSV Dateien in Dictionaries lesen



Sehr vorteilhaft kann es sein, wenn man die Daten eines CSV-Files als Dictionary-Objekt einliest. Hierfür benötigt man lediglich die DictReader()-Methode. Die Titelzeile wird hier zu den keywords:

```
import csv

filename = "bundeslaender.csv"

with open(filename, 'r') as data:

    for line in csv.DictReader(data, delimiter=";"):
        print(line) # Dictionary-Objekte ausgeben
                    # umwandeln durch print(dict(line))
```

Ausgabe:

```
OrderedDict([('Land', 'Wien'), ('Hauptstadt', 'Wien'), ('Einwohner', '1911728'), ('Wahrzeichen', 'Stephansdom')])
OrderedDict([('Land', 'Tirol'), ('Hauptstadt', 'Innsbruck'), ('Einwohner', '757852'), ('Wahrzeichen', 'Goldenes Dachl')])
OrderedDict([('Land', 'Kärnten'), ('Hauptstadt', 'Klagenfurt'), ('Einwohner', '561390'), ('Wahrzeichen', 'Lindwurm')])
OrderedDict([('Land', 'Steiermark'), ('Hauptstadt', 'Graz'), ('Einwohner', '1246576'), ('Wahrzeichen', 'Uhrturm')])
```

Übung: Test diesen Code. Verwendet wurde hierfür die bundeslaender.csv-Datei die vorhin erstellt wurde.

4.2. CSV Dateien in Dictionaries schreiben



Zum Schreiben eines Dictionaries in eine CSV-Datei verwenden wir die Methode `DictWriter()`

```
import csv
```

```
mydict = [{"Mo": "Mathematik", "Di": "Deutsch", "Mi": "Englisch", "Stunde": "1"},  
          {"Mo": "Sport", "Di": "Französisch", "Mi": "Geschichte", "Stunde": "2"},  
          {"Mo": "Sport", "Di": "Religion", "Mi": "Kunst", "Stunde": "3"}]
```

```
felder = ["Stunde", "Mo", "Di", "Mi"]  
filename = "stundenplan.csv"
```

```
with open(filename, "w") as csvfile:  
    # CSV-Writer Objekt erstellen  
    csvwriter = csv.DictWriter(csvfile, fieldnames=felder, delimiter=";",  
lineterminator = '\r')  
    #Felder schreiben  
    csvwriter.writeheader() # fieldnames  
    csvwriter.writerows(mydict)
```

Übung: Teste dieses Programm. Die Datei `stundenplan.csv` wird automatisch angelegt, wenn sie nicht vorhanden ist. Aus Platzgründen geht der Stundenplan nur bis Mittwoch ;-)

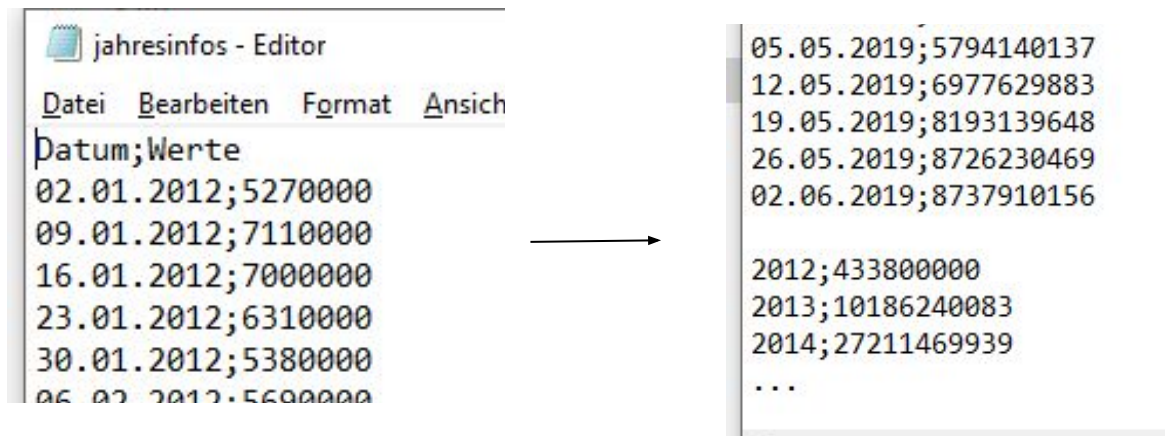
5. Hausübung

bis zur nächsten Theorie-Einheit



Dieses Übungsbeispiel muss bis zur nächsten Theorie-Einheit für die Mitarbeitsnote abgegeben werden:

Schreibe ein Python-Programm, das die Daten der vorliegende csv-Datei "jahresinfo.csv" einliest und am Ende der Datei eine Summenzeile der Werte pro Jahr (für 2012, 13, 14, 15, 16, 17, 18, und 19) hinzufügt.

A diagram illustrating a data transformation process. On the left, a window titled "jahresinfos - Editor" shows a CSV file with a header "Datum;Werte" and several rows of date-value pairs. An arrow points from this window to a second window on the right, which shows the same data but with additional summary rows at the bottom for the years 2012, 2013, and 2014, followed by an ellipsis.

Datum	Werte
02.01.2012	5270000
09.01.2012	7110000
16.01.2012	7000000
23.01.2012	6310000
30.01.2012	5380000
06.02.2012	5600000
05.05.2019	5794140137
12.05.2019	6977629883
19.05.2019	8193139648
26.05.2019	8726230469
02.06.2019	8737910156
2012	433800000
2013	10186240083
2014	27211469939
...	...