File Handling with Python

Nun wollen wir unsere Daten speichern

File-Handling in Python

Dieses Kapitel soll einerseits Wissen über txt, csv und json stärken und aufzeigen, wie einfach die Daten gespeichert werden können.

Einführung

Immer die gleichen 3 Schritte:

- Datei öffnen
- Etwas mit der Datei machen
- Datei schliessen

```
1 file = open('my_file.txt', 'modus')
2 # mach etwas
3 file.close()
```

Modus

Wichtig zu beachten ist, das es auch verschiedene Modi gibt um Dateien zu öffnen:

- 'r': read only
- 'w': write only
- 'r+': read and write
- 'a': append

```
file = open('my_file.txt', 'mode')
```

Schreiben

Die Funktion write() wird verwendet, um etwas in eine Datei zu schreiben.

'\n' wird verwendet, um einen Zeilenumbruch einzufügen.

```
1 # 19-add-lines.py
2
3 file = open('my_file.txt', 'a')
4 file.write('this is a new line')
5 file.write('this is another new line')
6 file.close()
```

Lesen

Eine for-Schleife kann verwendet werden, um eine Datei Zeile für Zeile zu lesen.

line.strip() entfernt das nachfolgende '\n'.

```
1 # 19-read-file-lines
2
3 file = open('my_file.txt', 'r')
4 for line in file:
5     line = line.strip()
6     print(line)
7 file.close()
```

CSV

- Bekanntes Format für strukturierte Daten
- Werte durch Kommas getrennt
- Eine neue Zeile steht für einen neuen Datensatz

```
"41662","Pulp Fiction",1994,168,8000000,8.8,132745,4.5,4.5,4.5,4.5,4.5,4.5,4.5,14.5,24.5,44.5,"R",0,0,0,1,0,0,0
"41663","Pulse",1988,95,NA,4.7,246,4.5,4.5,4.5,14.5,24.5,14.5,14.5,4.5,4.5,4.5,"",0,0,0,0,0,0,0
"41664","Pulse: A Stomp Odyssey",2002,40,3000000,8.5,62,4.5,4.5,4.5,0,4.5,4.5,14.5,14.5,24.5,34.5,"",0,0,0,0,1,0,1
"41665","Pulso, O",1997,21,NA,9.3,27,0,14.5,0,0,0,4.5,0,4.5,34.5,44.5,"",0,0,0,0,0,0,1
"41666","Pump Up the Volume",1990,105,NA,6.8,6054,4.5,4.5,4.5,4.5,4.5,14.5,24.5,14.5,14.5,14.5,"",0,0,1,1,0,0,0
```

CSV

- Bekanntes Format für strukturierte Daten
- Werte durch Kommas getrennt
- Eine neue Zeile steht für einen neuen Datensatz

```
1 # 19-read-csv.py
2
3 import csv
4
5 csvfile = open('cities.csv', 'r')
6 cities = csv.reader(csvfile, delimiter=',', quotechar='|')
7 for row in cities:
8     print(row[8], row[9])
```

CSV

- Möchte man sich das Datei-schliessen sparen, kann man auch with verwenden. Dies raubt aber auch ein wenig Flexibilität (alles eingerückt was passiert):

```
1 # 19-read-csv-with.py
2
3 import csv
4
5 with open("cities.csv", "r") as csvfile:
6     cities = csv.reader(csvfile, delimiter=',', quotechar='|')
7     for row in cities:
8         print(row[8], row[9])
```

CSV

Natürlich, funktioniert **`with`** auch mit normalen Dateien, jedoch nicht so anschaulich.

```
1 # 19-read-csv-with.py
2
3 import csv
4
5 with open("cities.csv", "r") as csvfile:
6     cities = csv.reader(csvfile, delimiter=',', quotechar='|')
7     for row in cities:
8         print(row[8], row[9])
```

CSV - DictReader

Man kann es sich auch noch angenehmer gestalten, indem man den DictReader verwendet und eine Dict-Ausgabe erhält:

```python
# 19-read-csv-with-dict.py

import csv

with open("onboarding.csv", "r") as csvfile:
    csvreader = csv.DictReader(csvfile, delimiter=';', quotechar='|')
    for row in csvreader:
        print(row["Login email"], row["First name"])
```

JSON

- Seit seiner Einführung hat sich JSON schnell zum De-facto-Standard für den Informationsaustausch entwickelt:
- … um Daten von hier nach dort zu transportieren.
- … gesammelte Daten in eine Datenbank zu speichern.
- … eine Möglichkeit für import und export von Daten zu haben.

JSON

- Das Kürzel bedeutet **`JavaScript Object Notation`** und wurde von einer Untergruppe der JavaScript-Programmiersprache inspiriert, die sich mit der Syntax von Objektliteralen beschäftigt.
- Aber keine Sorge:
  - JSON ist seit langem sprachunabhängig und existiert als eigener Standard, sodass JavaScript keine Rolle spielt

JSON

```json
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

JSON

Das Speichern von JSON Dateien ist sehr einfach:

```python
# 19-json-dumps.py

import json
data = {
    'employees' : [
        {
            'name' : 'John Doe',
            'department' : 'Marketing',
            'place' : 'Remote'
        },
        {
            'name' : 'Jane Doe',
            'department' : 'Software Engineering',
            'place' : 'Remote'
        },
        {
            'name' : 'Don Joe',
            'department' : 'Software Engineering',
            'place' : 'Office'
        }
    ]
}
# .dumps() as a string
json_string = json.dumps(data)
print(json_string)
```

JSON

Und wieder laden ist genauso einfach:

```
1 # 19-json-loads.py
2
3 import json
4
5 with open('json_data.json') as json_file:
6     data = json.load(json_file)
7     print(data)
```

JSON load(s) and dump(s)

Wie sie sicher bereits bemerkt haben, haben wir dump(s) und load(s) verwendet.

dump/load ist für geeignet, um eine Datei zu laden (mit JSON).

dumps/loads lädt direkt aus einem JSON-Formatierten String

Ende

Das war alles für dieses Kapitel