

Programmieren mit Python

Teil 10: Persistenz, APIs

Dr. Aaron Kunert

aaron.kunert@salemkolleg.de

9. Dezember 2021

Persistenz

Lesen und Schreiben von Dateien

Grundprinzip

Um mit Dateien zu arbeiten, geht man immer in 3 Schritten vor:

1. Datei öffnen
2. Datei bearbeiten (d.h. z.B. lesen, überschreiben, etwas anhängen)
3. Datei schließen

Das Schließen von Dateien ist relativ wichtig, kann aber schnell mal vergessen werden. Daher bietet Python eine spezielle Syntax mithilfe des Keywords `with` an.

Gesamten Text einer Datei einlesen

```
with open("some_file.txt") as my_file:  
    my_text = my_file.read()  
    print(my_text)
```

Erklärung

- Die Funktion `open` öffnet die angegebene Datei (Python geht per se davon aus, dass die Datei im gleichen Ordner wie das ausgeführte Skript liegt).
- Ein *Dateiobjekt* wird in der Variable `my_file` gespeichert (der Variablenname ist beliebig)
- Die Methode `.read()` liest den Text-Inhalt der Datei, so dass er in einer Variable gespeichert werden kann
- Sobald der eingerückte Block verlassen wird, wird die Datei automatisch geschlossen

Den Text einer Datei zeilenweise einlesen

```
with open("some_file.txt") as my_file:
    my_lines = my_file.readlines()
    for line in my_lines:
        print(f"The line reads: {line}")
```

Erklärung

- Die Methode `.readlines()` gibt eine *Liste* der Zeilen des Inhalts der Datei `"some_file.txt"` zurück.
- Durch diese Liste kann man mittels einer `for`-Schleife durchiterieren.

Text einlesen

Lade Dir aus dem FirstClass die Datei "`tf.txt`" herunter und kopiere sie in Dein Python-Projekt. Gib den Text auf der Konsole aus.

Zeilen zählen

Lade Dir aus dem FirstClass die Datei "`vs.txt`" herunter und kopiere sie in Dein Python-Projekt. Gib auf der Konsole aus, aus wievielen Zeilen der Text besteht.

Zählfunktion

Schreibe eine Funktion, die zu dem übergebenen Dateinamen die Anzahl an Zeilen zurückgibt.

Text einlesen

```
with open("tf.txt") as my_file:  
    my_text = my_file.read()  
    print(my_text)
```

Zeilen zählen

```
with open("vs.txt") as my_file:  
    my_lines = my_file.readlines()  
    length = len(my_lines)  
    print(length)
```

Zählfunktion

```
def count_lines(filename):  
    with open(filename) as my_file:  
        my_lines = my_file.readlines()  
        return len(my_lines)
```

Text in eine Datei schreiben

```
with open("some_file.txt", "w") as my_file:  
    my_file.write("Hello everybody")
```

Erklärung

- Ruft man `open` mit dem zweiten Parameter `"w"` auf, so wird die Datei im Schreibmodus geöffnet.
- Existierte die Datei zuvor noch nicht, so wird sie erzeugt.
- Mit der Methode `.write("Inhalt")` lässt sich Text in eine Datei schreiben.
- Achtung: Öffnet man eine Datei im Schreibmodus, so wird der bisherige Inhalt überschrieben.

Text an eine Datei anhängen

```
with open("some_file.txt", "a") as my_file:  
    my_file.write("Some text to append")
```

Erklärung

- Ruft man `open` mit dem zweiten Parameter `"a"` auf, so wird die Datei im *Append*-Modus geöffnet.
- Existierte die Datei zuvor noch nicht, so wird sie erzeugt.
- Mit der Methode `.write("Inhalt")` lässt sich Text an die Datei anhängen.
- Der bis dahin in der Datei vorhandene Inhalt wird nicht verändert oder gelöscht.
- Der einzige Unterschied zum letzten Punkt ist der Modus (`"a"` statt `"w"`).

Achtung Umlaute

Hat man eine etwas ältere Version von Python und möchte man Dateien, die Umlauten und andere Sonderzeichen enthalten, bearbeiten, so muss man beim Öffnen der Datei noch den Parameter `encoding="utf-8"` übergeben.

Beispiel

```
with open("some_file.txt", "a", encoding="utf-8") as my_file:  
    my_file.write("Hier ein Text mit Umlauten: äöüß")
```

JSON

Ein universelles Datenformat

Definition: JSON

JSON (Java Script Object Notation) ist ein Daten-Format, um Verschachtelungen von Listen und Dictionaries darzustellen, zu speichern und auszutauschen. Die Syntax entspricht (fast) der üblichen Python-Syntax und wird von den meisten Programmiersprachen „verstanden“.

Beispiel: Eine Liste von Ländern

```
[  
  {  
    "name": "Germany",  
    "capital": "Berlin",  
    "population": 83190556,  
    "cities": ["Berlin", "Hamburg", "München", "Köln"]  
  },  
  {  
    "name": "France",  
    "capital": "Paris",  
    "population": 67422000  
    "cities": ["Paris", "Marseilles", "Lyon", "Toulouse"]  
  },  
  ...  
]
```

Eigenschaften

- Dictionaries und Listen dürfen beliebig verschachtelt werden.
- Die äußerste Ebene kann ein Dictionary oder eine Liste sein.
- Es müssen doppelte Anführungsstriche verwendet werden.
- Neben Dictionaries und Listen können folgende Datentypen verwendet werden:
 - Integer
 - String
 - Float
 - Boolean (`true` bzw. `false`)
 - `null` (entspricht `None`)

Python's JSON-Modul

Um in Python Daten im JSON-Format einzulesen und zu speichern, benötigt man das mitgelieferte *JSON-Modul*. Dazu einfach die folgende Zeile am Beginn des Python-Skripts anfügen:

```
import json
```

```
...
```

Daten als JSON-Datei abspeichern

```
import json

my_data = {"a": 1, "b": 2}  # some dummy data

with open("my_data.json", "w") as my_file:
    json.dump(my_data, my_file)
```

Erklärung

- Zunächst wird die Datei `my_data.json` im Schreibmodus geöffnet.
- Die Funktion `json.dump` erwartet die Daten und eine Datei. Die Daten werden im JSON-Format in der Datei abgespeichert.
- Achtung: Der bisherige Inhalt von `my_data.json` wird überschrieben.

Daten aus einer JSON-Datei importieren

```
import json

with open("my_data.json") as my_file:
    data = json.load(my_file)

print(data)
```

Erklärung

- Zunächst wird die Datei `my_data.json` im Lesemodus geöffnet.
- Die Funktion `json.load` erwartet eine JSON-Datei und gibt die eingelesenen Daten als Liste bzw. Dictionary zurück.

Userdaten

Lade Dir aus dem FirstClass die Datei "`player.json`" herunter und kopiere sie in Dein Python-Projekt. Öffne die Datei und gib den Namen, der Spielerin, sowie das Level und den Punktestand auf der Konsole aus.

Lösung

```
import json

with open("player.json") as my_file:
    data = json.load(my_file)

print(f"Name: { data["name"] } ")
print(f"Punktestand: { data["score"] } ")
print(f"Level: { data["level"] } ")
```

Levelfortschritt speichern

Verwende wieder die Datei `"player.json"`. Implementiere die Funktion `levelup()`, die das Userprofil einliest, das Level um 1 und den Punktestand um 100 erhöht und die neuen Daten wieder in der Datei `"player.json"` abspeichert.

Lösung

```
import json

def levelup():
    with open("player.json") as my_file:
        data = json.load(my_file)
    data["score"] += 100
    data["level"] += 1
    with open("player.json", "w") as my_file:
        json.dump(data, my_file)
```

Web APIs

Wie Python im Netz surfen kann

Definition: API

Eine *API* (Application Programming Interface) ist im Allgemeinen eine klar definierte Schnittstelle zwischen Programmen.

Im Kontext des Internets geht es dabei meist um Webanwendungen, die statt einer für menschen lesbaren Seite, ein maschinenlesbares JSON ausliefern. Auf diese Weise ist ein Datenaustausch zwischen Deinem Programm und einer Webanwendung in beide Richtungen möglich.

Eine Web-API besteht meist aus einer (oder mehreren) Webadressen und einer Anleitung, wie sie zu benutzen ist.

Beispiele für APIs

- **Instagram Api:** Erhalte Infos, wer Dir folgt und welche Reaktionen Du auf Deine Bilder erhältst.
- **GoogleMaps Api:** Sende Koordinaten an Google-Maps und erhalte die Adresse.
- **REST Countries:** Erhalte Daten von Ländern dieser Erde.
- **Stock Market Apis:** Erhalte live Daten zu Börsenkursen.

Anwendungsbeispiele

- **Instagram:** Schreibe Dir einen Python-Scheduler, wann welche Bilder von Dir veröffentlicht werden sollen.
- **REST Countries:** Nutze die Daten, um ein Quiz zu schreiben.
- **Stock Market Apis:** Trading Apps, SMS-Warnung bei einbrechenden Preisen.

Wie macht man einen API-Call?

Im Folgenden gehen wir durch die Schritte, die man ausführen muss, um einen lesenden API-Call auszuführen. Wir verwenden dafür die **REST Countries Api**. Die Dokumentation (*Docs*) der Api findet sich unter <https://restcountries.com/>.

Schritt 1: Finde die richtige URL (d.h. Webadresse)

Zunächst benötigt man eine URL (den sogenannte *Endpoint*). Diese kann völlig statisch sein, oftmals aber sind Teile der URL dynamisch bzw. hängen von Deiner genauen Anfrage ab.

Beispiel:

`https://restcountries.com/v3.1/all`

oder

`https://restcountries.com/v3.1/name/france`

Schritt 2: Schicke einen Request an den Endpoint

Verwende folgenden Code, um einen Request zu schicken:

```
import requests
...

response = requests.get("https://restcountries.com/v3.1/name/france")
data = response.json()
# print(data)
```

Schritt 3: Erweitere den Request ggf. um *Query-Parameter*

```
import requests
...

payload = {"fields": ["capital","population","continents"]}
response = requests.get("https://restcountries.com/v3.1/name/france", params=payload)
data = response.json()
# print(data)
```

Schritt 4: Transformiere, filtere und speichere die Daten je nach Bedarf

```
import requests
import json
...

payload = {"fields": ["capital","population","continents"]}
response = requests.get("https://restcountries.com/v3.1/name/france", params=payload)
data = response.json()

with open("france.json","w") as my_file:
    json.dump(data, my_file)
```

Daten eines Landes extrahieren

Stelle einen Request zu Italien an die Countries-API. Bereite die zurückgegebenen Daten so auf, dass folgendes auf der Konsole erscheint:

Hauptstadt: Rome

Bevölkerung: 59554023

Kontinent: Europe

Daten eines Landes extrahieren

```
import requests

response = requests.get("https://restcountries.com/v3.1/name/italy")
data = response.json()

data = data[0]
capital = data["capital"][0]
population = data["population"]
continent = data["continents"][0]

print(f"Hauptstadt: {capital}")
print(f"Bevölkerung: {population}")
print(f"Kontinent: {continent}")
```

Hauptstadt-Orakel

Schreibe eine Funktion, die den (englischsprachigen) Namen eines Landes erwartet und den Namen der Hauptstadt zurückgibt.

Hauptstadt-Orakel

```
import requests

def get_capital(country):
    response = requests.get("https://restcountries.com/v3.1/name/" + country)
    data = response.json()
    data = data[0]
    capital = data["capital"][0]
    return capital
```

Datenaufbereitung

Erstelle eine Liste aller Länder. Jedes Land soll dabei als Dictionary der folgenden Form abgespeichert werden:

```
{  
  "name": "Italy",  
  "capital": "Rome",  
  "continent": "Europe",  
  "population": 59554023  
}
```

Speichere diese Liste als JSON in der Datei `countries.json`.

Datenaufbereitung

```
import requests
import json

response = requests.get("https://restcountries.com/v3.1/all")
data = response.json()

countries = []
for country in data:
    if "capital" in country.keys():
        countries.append({
            "name": country["name"]["common"],
            "capital": country["capital"][0],
            "population": country["population"],
            "continent": country["continents"][0]
        })

with open("countries.json", "w") as my_file:
    json.dump(countries, my_file, indent=4)
```
