Programmieren mit Python

Teil 6: Scope und Datei-Handling

Dr. Aaron Kunert
aaron.kunert@salemkolleg.de

10. Juni 2021

Scope

Wo Variablen gültig sind

Problemstellung

Sei `my_variable` eine Variable mit Wert 1. Schreibe eine Funktion, die bei Aufruf die Variable `my_variable` um 1 erhöht.

Wie macht man das?

Das Problem

```
my_variable = 1

def increment():
  my_variable = my_variable + 1

increment()
print(my_variable)
```

Die offensichtliche Lösung funktioniert nicht. Warum nicht?

Experiment I

```
global_variable = 1

def my_function():
  local_variable = 5

my_function()
print(global_variable)
print(local_variable)
```

Beobachtung

Eine Variable, die innerhalb einer Funktion definiert wurde, ist auch nur innerhalb der Funktion sichtbar.

Experiment II

```
global_variable = 1

def my_function():
  print(global_variable)


my_function()
print(global_variable)
```

Beobachtung

Eine *globale* Variable ist auch innerhalb einer Funktion definiert.

Experiment III

```
global_variable = 1

def my_function():
  global_variable = 5
  print(global_variable)


my_function()
print(global_variable)
```

Beobachtung

Eine Variable innerhalb einer Funktion kann den gleichen Namen wie eine Variable außerhalb haben, allerdings ist die innere Variable nur innerhalb der Funktion sichtbar.

Experiment IV

```
global_variable = 1

def my_function():
  print(global_variable)
  global_variable = 5


my_function()
print(global_variable)
```

Beobachtung/Erklärung

Python entscheidet anhand des Kontexts ob `global_variable` eine globale Variable ist, oder eine lokale Variable, die zufällig den gleichen Namen wie eine globale Variable trägt.

Falls Python denkt, dass es sich um eine globale Variable handelt, so kann diese nur gelesen, nicht aber geschrieben (d.h. neu definiert) werden.

Das Eingangsbeispiel

```
my_variable = 1

def increment():
  my_variable = my_variable + 1

increment()
print(my_variable)
```

Erklärung

Da `my_variable` rechts vom Gleichheitszeichen steht, denkt Python, dass es sich um die globale Variable `my_variable` handelt. Da `my_variable` aber auch links vom Gleichheitszeichen steht, wird auch schreibend auf die Variable zugegriffen. Das ist nicht erlaubt.

Mögliche Lösung

```
my_variable = 1

def increment(var):
  return var + 1

my_variable = increment(my_variable)
print(my_variable)
```

Definition

Der Gültigkeitsbereich einer Variable wird *Scope* genannt.

Scope in Python

In Python unterscheidet man zwischen *global Scope* und *local Scope*. Im local Scope hat man nur Lesezugriff auf den global Scope.

Achtung Ausnahme

```
my_list = [1, 2, 3]

def append(item):
  my_list.append(item)

append(4)
print(my_list)
```

Erklärung

Da die Variable `my_list` nicht überschrieben wird, sondern nur das referenzierte Objekt verändert wird, erkennt Python dies nicht als Schreibzugriff und erlaubt dieses Vorgehen.

Warum ist der Zugriff auf den Global Scope eingeschränkt?

- Funktionen sollen möglichst wenige Nebeneffekte haben. Wenn eine Funktion den global Scope verändern kann, ist dies ein großer Nebeneffekt.
- Wenn man eine Funktion schreibt, muss man sich keine Gedanken machen, ob ein Variablenname schon vergeben ist.
- Wenn man sich innerhalb einer Funktion den Kontakt zum global Scope reduziert, so ist die Funktion besser zu verstehen, zu warten und zu testen.
- ...

Input/Output II

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 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

Will 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.