# Programmieren mit Python

## Teil 6: Listen II, Dictionaries

Dr. Aaron Kunert
*aaron.kunert@salemkolleg.de*

18. November 2021

**Mutability**

Listen sind der erste Datentyp, den wir kennenlernen, der *mutable* (veränderbar) ist. Die bisherigen Datentypen waren *immutable*, d.h. man konnte sie zwar überschreiben, aber nicht verändern.

**Call by Reference vs. Call by Value**

Enthält die Variable `my_list` eine Liste, so speichert Python eigentlich gar nicht die Liste in dieser Variable, sondern nur die Speicheradresse der Liste. Dieses vorgehen nennt man auch *Call by Reference*. Bei den Datentypen `int` und `str` wird stattdessen tatsächlich der Wert der Variable abgespeichert. Dies nennt man *Call by Value*.

# Übung

**Eine Liste kopieren**

Definiere die Variable `my_list` als die Liste `[1,2,3]`. Kopiere die Variable `my_list` in die Variable `my_list_copy`. Füge einen weiteren Eintrag zu `my_list` hinzu. Welchen Wert hat `my_list_copy`?

**Lösung**

```
my_list = [1, 2, 3]
my_list_copy = my_list
my_list.append(4)
print(my_list_copy) # 1 2 3 4
```

### Schleife über Liste

Analog wie über Strings und Ranges kann man Schleifen auch über eine Liste laufen lassen.

**Beispiel**

```
countries = ["Bulgarien", "Griechenland", "Türkei", "Libanon"]

for country in countries:
  print(country)

# Bulgarien
# Griechenland
# Türkei
# Libanon
```

### Schleife über Liste mit Indizes

Möchte man in einer Schleife nicht nur die Listeneinträge, sondern auch die Indizes verwenden, so muss man die Funktion `enumerate()` auf die Liste anwenden.

#### Beispiel

```
countries = ["Guatemala", "Nicaragua", "Honduras", "Belize"]

for (index, country) in enumerate(countries):
  print(f"Das {index + 1}. Land ist {country}")

# Das 1. Land ist Guatemala
# Das 2. Land ist Nicaragua
# Das 3. Land ist Honduras
# Das 4. Land ist Belize
```

# Übung

**Liste durchsuchen**

Prüfe, ob in einer Liste von Ländern das Land `"Italien"` vorkommt. Gib dazu auf der Konsole entweder

```
Italien ist in der Liste
```

oder

```
Italien ist nicht in der Liste
```

aus.

# Lösung

## Liste durchsuchen

```
# Wähle ein Beispiel für countries
countries = ["Finnland", "Norwegen", "Schweden", "Dänemark"]

for country in countries:
  if country == "Italien":
    print("Italien ist in der Liste")
    break
else:
  print("Italien ist nicht in der Liste")
```

**Ist ein Element in einer Liste enthalten?**

Möchte man prüfen, ob ein Element in einer Liste enthalten ist, so kann man auch das Schlüsselwort *in* verwenden.

**Beispiel**

```
countries = ["Finnland", "Norwegen", "Schweden", "Dänemark"]

var_1 = "Finnland" in countries
var_2 = "Deutschland" in countries

print(var_1)  # True
print(var_2)  # False
```

### Eine Liste sortieren

Um eine Liste zu sortieren, verwende die Methode `.sort()`. Dies verändert die Liste dauerhaft.
Um eine sortierte Kopie einer Liste zu erstellen, verwende die Funktion `sorted()`.
Mit Hilfe des Parameters `reverse=True` lässt sich eine Liste absteigend ordnen.

#### Beispiel für `sort`

```
my_list = [1, 5, 2, 7]
my_list.sort()
print(my_list)  # [1, 2, 5, 7]
```

#### Beispiel für `sorted`

```
my_list = [1, 5, 2, 7]
sorted_list = sorted(my_list)
print(my_list)  # [1, 5, 2, 7]
print(sorted_list)  # [1, 2, 5, 7]
```

## Beispiel für absteigende Sortierung

```
my_list = [1, 5, 2, 7]
my_list.sort(reverse=True)
print(my_list)  # [7, 5, 2, 1]

my_list = [7, 12, 5, 18]
sorted_list = sorted(my_list, reverse=True)
print(sorted_list) # [18, 12, 7, 5]
```

# Übung

**Beste/Schlechteste Note**

Sei `grades` eine Liste der Noten deiner letzten Klausuren (z.B. `grades = [12, 9, 14, 11]`). Gib dann auf der Konsole einmal die beste und einmal die schlechteste Note aus.

**Lösung**

```
grades = [12, 9, 14, 11]
grades.sort()
min_grade = grades[0]
max_grade = grades[-1]
print(f"Schlechteste Note: {min_grade}")
print(f"Beste Note: {max_grade}")
```

### Nützliche Funktionen/Methoden

Für Listen stellt Python viele nützliche Methoden bzw. Funktionen bereit. Wenn Du googlest, findest Du für viele „Alltagsfragen“ eine Lösung.

Zum Beispiel hier: `https://docs.python.org/3/tutorial/datastructures.html`

### Beispiele

```
my_list = [2, 4, 8, 1]

len(my_list)    # = 4  (Gibt die Anzahl der Elemente an)
sum(my_list)    # = 15 (Berechnet die Summe der Elemente)
my_list.reverse() # [1, 8, 4, 2] (Dreht die Reihenfolge um)
my_list.insert(2,-1) # [2, 4, -1, 8, 1] (fügt den Wert -1 an Position 2 ein)
my_list.pop() # 1 (Gibt den letzten Eintrag der Liste zurück und entfernt ihn aus der Liste)
```

# Übung

**Durchschnittsnote**

Sei `grades` wieder eine Liste mit deinen letzten Noten. Gib auf der Konsole die Durchschnittsnote aus.

**Lösung**

```
grades = [12, 9, 14, 11]
total_sum = sum(grades)
count = len(grades)
average = total_sum/count
print(f"Die Durchschnittsnote ist {average}")
```

**Slicing**

Wenn man eine Liste hat, ist es oft nötig, einen Teil der Liste „auszuschneiden“.
Dafür hat Python die *Slice-Notation* eingeführt.
Diese funktioniert nach folgendem Schema:

`my_list[start:stop:step]`.

Die Einträge (start, stop, step) sind dabei jeweils optional. Wie immer wird der obere Wert (`stop`) gerade nicht erreicht.

Slicing lässt sich übrigens auch nach dem gleichen Schema auch auf Strings anwenden.

**Wichtig**

Wenn man Slicing anwendet, erhält man eine Kopie der ausgewählten Elemente zurück. Die ursprüngliche Liste wird *nicht* verändert.

## Beispiele

```
my_list = [2, 4, 6, 8, 10]

my_list[1:3]      # [4, 6]
my_list[0:4]      # [2, 4, 6, 8]
my_list[1:1]      # []
my_list[0:4:2]    # [2, 6]
my_list[:3]       # [2, 4, 6]
my_list[2:]       # [6, 8, 10]
my_list[:]        # [2, 4, 6, 8, 10]
my_list[1:-2]     # [6]
my_list[-3:-1]    # [6, 8]
my_list[::-1]     # [10, 8, 6, 4, 2]
```

# Dictionaries

**Problemstellung**

Eine Variable soll nicht nur die Namen von Ländern enthalten, sondern auch noch deren Hauptstadt.

Wie macht man das?

## Lösung

```
capitals = {"Deutschland": "Berlin", "Spanien": "Madrid", "Italien": "Rom"}

country = input("Von welchem Land möchtest Du die Hauptstadt wissen?")

print(f"Die Hauptstadt von { country } ist { capitals[country] } Punkte")
```

**Struktur eines *Dictionaries***

`my_dict = {key_1:value_1, key_2:value_2, ..., key_n:value_n}`

Das *Dictionary* `my_dict` enthält Schlüssel-Wert-Paare (*key-value-pairs*). Die Schlüssel müssen eindeutig und unveränderlich sein (z.B. vom Typ `string` oder `int`). Die Werte dürfen beliebige Datentypen sein.

**Good to know**

- Zur besseren Übersichtlichkeit werden Dictionaries oftmals wie folgt formatiert:

```
capitals = {
  "Deutschland": "Berlin",
  "Spanien": "Madrid",
  "Italien": "Rom"
}
```

- Dictionaries sind mutable, können also verändert werden.
- Dictionaries besitzen keine vernünftige Anordnung und können nicht geordnet werden.
- Ein Dictionary kann leer sein.

- Oftmals bietet es sich an, statt einem Dictionary eine Liste von Dictionaries zu verwenden:

```
countries = [
  {
    "name": "Deutschland",
    "capital": "Berlin",
    "pop": 82000000,
    "is_eu_member": True
  },
  # ...
  {
    "name": "Italien",
    "capital": "Rom",
    "pop": 65000000,
    "is_eu_member": True
  }
]
```

**Auf Dictionary-Elemente zugreifen**

Sei `my_dict = {"a": 5, "b": 8}`.

Mit der Syntax `my_dict["a"]` kann man den Wert an der Stelle `"a"` auslesen.

Mit der Syntax `my_dict["a"] = 12` kann man einzelne Werte des Dictionaries verändern.

Auf diese Weise können auch ganz neue Paare hinzugefügt werden. Zum Beispiel:
`my_dict["c"] = -2`.

# Übung

**Dictionary manipulieren**

Gegeben sei das folgende Dictionary:

```
grades = {"Mathe": 8, "Bio": 11, "Sport": 13}
```

Bestimme die Durchschnittsnote dieser drei Fächer. Verbessere danach Deine Mathenote um einen Punkt und füge noch eine weitere Note für Englisch hinzu (Abfrage über Konsole). Gib danach erneut den Durchschnitt an.

# Lösung

## Dictionary manipulieren

```
grades = {"Mathe": 8, "Bio": 11, "Sport": 13}

grades_sum = grades["Mathe"] + grades["Bio"] + grades["Sport"]
average = grades_sum/len(grades)
print(f"Der Durchschnitt ist {average} Punkte")

grades["Mathe"] += 1

eng_grade = input("Welche Note hast Du in Englisch? ")
eng_grade = int(eng_grade)
grades["Englisch"] = eng_grade

grades_sum += grades["Englisch"]
average = grades_sum/len(grades)
print(f"Der Durchschnitt ist {average} Punkte")
```

**Einen Eintrag aus einem Dictionary entfernen**

Wie bei Listen, kann man mittels `del`-Statement einen Eintrag aus einem Dictionary entfernen:

```
del eu_countries["united_kingdom"]
```

**Was wird hier passieren?**

```
old_capitals = {"Deutschland": "Bonn", "Norwegen": "Oslo"}
new_capitals = old_capitals

new_capitals["Deutschland"] = "Berlin"

print(old_capitals)
print(new_capitals)
```

**Erklärung**

Da Dictionaries mutable sind, findet bei ihnen der Aufruf mittels *Call by Reference* statt. Das heißt, dass in der Variable `old_capitals` bzw. `new_capitals` nicht die Länder gespeichert sind, sondern nur die Speicheradresse, wo die Länder zu finden sind. Ändert man die zugrundeliegenden Daten an einer Stelle, so ändern sie sich daher auch an der anderen Stelle.

**Eine Kopie von einem Dictionary erstellen**

Mit der Funktion `dict()` kann man eine Kopie von einem Dictionary erstellen.

Beispiel: `dict(my_dict)` erstellt eine Kopie von `my_dict`.

## Schleife über Dictionary I

Ähnlich wie bei Listen kann man Schleifen auch über ein Dictionary laufen lassen.

### Beispiel

```
capitals = {"Litauen": "Vilnius", "Lettland": "Riga", "Estland": "Tallin"}

for item in capitals:
  print(item)

# Litauen
# Lettland
# Tallin
```

### Schleife über Dictionary II

Möchte man in der Schleife nicht nur die Schlüssel, sondern auch die Werte des Dictionaries zur Verfügung haben, so muss man die Methode `.items()` auf das Dictionary anwenden.

**Beispiel**

```
capitals = {"Litauen": "Vilnius", "Lettland": "Riga", "Estland": "Tallin"}

for key, value in capitals.items():
  print(f"Hauptstadt von {key}: {value}")

# Hauptstadt von Litauen: Vilnius
# Hauptstadt von Lettland: Riga
# Hauptstadt von Estland: Tallin
```

# Übungen

**Zwei Dictionaries kombinieren**

Gegeben seien zwei Dictionaries, z.B.

```
eu = {"Deutschland": "Berlin", "Frankreich": "Paris" }
```

und

```
non_eu = {"Russland": "Moskau", "China": "Peking" }
```

Füge die Einträge des zweiten Dictionaries zum ersten Dictionary hinzu.

**Ein Dictionary „filtern“**

Sei ein beliebiges Dictionary mit Noten gegeben. Entferne alle Einträge, deren Note schlechter als 5 Punkte ist.

# Lösung

## Zwei Dictionaries kombinieren

```
eu = {"Deutschland": "Berlin", "Frankreich": "Paris" }
non_eu = {"Russland": "Moskau", "China": "Peking" }

for key,value in non_eu.items():
  eu[key] = value
print(eu)
```

## Ein Dictionary „filtern“

```
grades = {"Deutsch": 11, "Mathe": 3, "Sport": 14, "Geschichte": 1}
#  Man darf die Länge eines Dictionaries in einer Schleife nicht verändern, deshalb machen wir eine
result = dict(grades)
for key, value in grades.items():
  if value < 5:
    del result[key]
print(result)
```

### Ein Dictionary zerlegen

Mit der Methode `.keys()` erhält man eine Liste aller Schlüssel eines Dictionaries.

Mit der Methode `.values()` erhält man eine Liste aller Werte eines Dictionaries.

In beiden Fällen, muss das Ergebnis mittels der Funktion `list()` in eine Liste umgewandelt werden.

**Beispiel**

```
my_dictionary = {"China": "Peking", "Japan": "Tokio", "Korea": "Seoul"}

countries = my_dictionary.keys()
countries = list(countries)

capitals = my_dictionary.values()
capitals = list(capitals)

print(countries)   # ["China", "Japan", "Korea"]
print(capitals)    # ["Peking", "Tokio", "Seoul"]
```