

Programmieren mit Python

Teil 4: Datenstrukturen

Dr. Aaron Kunert

aaron.kunert@salemkolleg.de

6. Mai 2021

break, continue **und** else

Den Fluss einer Schleife kontrollieren

Das break-Statement

Taucht innerhalb einer Schleife das Schlüsselwort `break` auf, so wird die weitere Abarbeitung der Schleife abgebrochen. Die Ausführung wird mit dem Code *nach* dem Schleifenblock ausgeführt.

Beispiel

```
for k in range(1,100):  
    print(k)  
    if k > 3:  
        break  
# prints 1 2 3 4
```

Das continue-Statement

Taucht innerhalb einer Schleife das Schlüsselwort `continue` auf, so wird der aktuelle Schleifendurchgang abgebrochen. Die Ausführung wird mit der nächsten Schleifeniteration fortgesetzt.

Beispiel

```
for k in range(1,11):  
    if k % 2 == 0:  
        continue  
    print(k)  
# prints 1 3 5 7 9
```

Der else-Block einer Schleife

Analog zum `if`-Statement, kann auch eine Schleife einen `else`-Block haben. Dieser wird ausgeführt, wenn die Schleife *regulär* (also nicht durch die Verwendung von `break`) beendet wird.

Beispiel

```
name = input("Your name: ")

for letter in name:
    if letter == "a":
        print("Your name contains an a")
        break
else:
    print("Your name contains no a")
```

Zählen bis zur nächsten 10er-Zahl

Lies eine Zahl x ein und gib auf der Konsole die Zahlen von x bis zur nächsten 10er-Zahl aus.
Ist die Eingabe $x = 17$, so soll die Ausgabe wie folgt aussehen:

```
17  
18  
19  
20
```

Zählen mit Lücken

Schreibe ein Skript, dass die Zahlen von 1 bis 99 aufzählt, dabei allerdings die 10er-Zahlen weglässt. Versuche dabei, ein `continue`-Statement zu verwenden.

Zählen bis zur nächsten 10er-Zahl

```
x = input("Gib eine Zahl an: ")  
x = int(x)
```

```
for k in range(x, x + 11):  
    print(k)  
    if k % 10 == 0:  
        break
```

Zählen mit Lücken

```
for k in range(1, 100):  
    if k % 10 == 0:  
        continue  
    print(k)
```

Harte Übung

Primzahltest

Lies eine ganze Zahl x ein und überprüfe, ob diese Zahl eine Primzahl ist. Die Ausgabe des Programms soll etwa wie folgt aussehen:

Die Zahl 28061983 ist eine Primzahl.

Lösung

```
x = input("Gib eine Zahl ein: ")
x = int(x)

for k in range(2, x):
    if x % k == 0:
        print(f"{x} ist keine Primzahl.")
        break
else:
    print(f"{x} ist eine Primzahl.")
```

Listen

Viele Variablen gleichzeitig speichern

Problemstellung

Lies mit Hilfe einer Schleife nach und nach Schulnoten von Dir ein. Alle Noten sollen gespeichert werden. Danach sollst Du die Wahl haben, die soundsovielte Note anzeigen lassen zu können.

Wie macht man das?

Lösung (fast)

```
# ...  
# Um das Eingeben der Noten kümmern wir uns noch  
grades = [12, 10, 7, 14, 13, 13, 6, 4, 15, 14] # Noten in Notenpunkten  
  
index = input("Die wievielte Note möchtest Du nochmal anschauen?")  
index = int(index)  
  
print(f"Deine { index }. Note ist { grades[index] } Punkte")
```

Struktur einer *Liste*

```
my_list = [element_0, element_1, ..., element_n]
```

Die Variable `my_list` trägt nicht nur einen Wert, sondern $n + 1$ Werte. Ansonsten verhält sich `my_list` wie eine ganz „normale“ Variable. Als Einträge einer Liste sind beliebige Werte mit beliebigen Datentypen zugelassen.

Frage: Welchen Datentyp hat die Liste `[2, 2.3, "Hello"]` ?

Auf Listenelemente zugreifen

Auf das n -te Element der Liste `my_list` kann man mittels `my_list[n]` zugreifen.

Mit `my_list[-1]`, `my_list[-2]`, etc. kann man auf das letzte, vorletzte, etc. Element der Liste zugreifen.

Achtung

Python fängt bei 0 an zu zählen. D.h. das erste Element in der Liste hat den Index 0.

Beispiel: `my_list[1]` liefert das **2. Element** der Liste.

Schreibzugriff auf Listenelemente

Nach dem gleichen Prinzip lassen sich einzelne Listeneinträge verändern.

Beispiel: `my_list[3] = -23`.

Achtung

Man kann nur schon existierende Listeneinträge verändern.

Neues Konzept

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.

Listeneinträge hinzufügen

Mit der *Methode* `.append()` kann ein Eintrag zur Liste hinzugefügt werden.

Bsp: `my_list.append(12)` fügt einen weiteren Eintrag mit Wert 12 hinzu.

Listeneinträge entfernen

Mit dem Keyword `del` kann man Einträge an einer bestimmten Position löschen. Dabei verschieben sich die darauffolgenden Einträge um 1 nach vorne.

Beispiel: `del my_list[2]` löscht das dritte Element.

Mit der Methode `.remove()` kann man Einträge mit einem bestimmten Wert löschen.

Beispiel: `my_list.remove(-23)` entfernt den ersten Eintrag mit dem Wert -23. Ist der Wert nicht vorhanden gibt es eine Fehlermeldung.

Eine Liste erstellen

Schreibe ein kleines Programm, dass Dich ca. 4x nach dem Namen einer Freund*in fragt und Dir am Schluss die Liste der eingegebenen Freund*innen ausgibt.

Lösung

```
friends = []  
for k in range(1, 5):  
    friend = input("Wie heißt ein*e Freund*in von Dir? ")  
    friends.append(friend)  
print(friends)
```

Das Eingangsproblem

Schreibe ein kleines Programm, dass solange Deine Noten einliest, bis Du **q** drückst. Danach sollst Du die Möglichkeit haben, eine Zahl k einzugeben, so dass Dir die k -te Note angezeigt wird.

Das Eingangsproblem

```
grades = []
while True:
    grade = input("Gib eine Note an: ")
    if grade == "q":
        break
    grades.append(int(grade))

index = input("Die wievielte Note möchtest Du nochmal anschauen?")
index = int(index)
print(f"Deine { index }. Note ist { grades[index-1] } Punkte")
```

Schleife über Liste

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

Beispiel

```
my_hobbies = ["Segeln", "Tennis", "Schwimmen", "Lesen"]
```

```
for hobby in my_hobbies:  
    print(hobby)
```

```
# prints:  
# Segeln  
# Tennis  
# Schwimmen  
# Lesen
```

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

```
my_hobbies = ["Segeln", "Tennis", "Schwimmen", "Lesen"]
```

```
for index, hobby in enumerate(my_hobbies):  
    print(f"Mein {index + 1}. Hobby ist {hobby}")
```

```
# prints:  
# Mein 1. Hobby ist Segeln  
# Mein 2. Hobby ist Tennis  
# Mein 3. Hobby ist Schwimmen  
# Mein 4. Hobby ist Lesen
```

Liste durchsuchen

Lies wieder eine Liste Deiner Noten ein. Prüfe, ob Du mindestens einmal unterpunktet hast (d.h. 0 Punkte hattest). Auf der Konsole soll dann entweder

Du hast irgendwo unterpunktet

oder

Du hast nirgendwo unterpunktet

ausgegeben werden.

Liste durchsuchen

```
grades = []
while True:
    grade = input("Gib eine Note an: ")
    if grade == "q":
        break
    grades.append(int(grade))

for grade in grades:
    if grade == 0:
        print("Du hast irgendwo unterpunktet.")
        break
else:
    print("Du hast nirgendwo unterpunktet.")
```

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

```
my_hobbies = ["Segeln", "Tennis", "Schwimmen", "Lesen"]
```

```
sailing_in_list = "Segeln" in my_hobbies
```

```
climbing_in_list = "Klettern" in my_hobbies
```

```
print(sailing_in_list)    # True
```

```
print(climbing_in_list)  # 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]
```

Beste/Schlechteste Note

Lies wieder ein paar Noten ein. Gib dann auf der Konsole einmal die beste und einmal die schlechteste Note aus.

Lösung

```
# ...  
# einlesen wie immer  
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)
```

Durchschnittsnote

Lies wieder ein paar Noten ein. Gib auf der Konsole die Durchschnittsnote aus.

Lösung

```
# ...  
# einlesen wie immer  
  
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]
```
