Woche 03: Programmierung - Listen in Jupyter Notebook

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Eine Liste in Python erzeugen und ihr Elemente hinzufügen können
- Auf Elemente einer Liste zugreifen und diese ausgeben können
- Elemente innerhalb einer Liste ersetzen und löschen können
- Zugehörigkeit der Elemente zu einer Liste prüfen können
- Listen verketten können

Projekt „KI für alle"
HeiCAD –Heine Center for Artificial Intelligence and Data Science

Inhalt

Erzeugung von Listen

In Python werden Listen durch eckige Klammern gekennzeichnet. Die Elemente innerhalb einer Liste werden durch Kommas voneinander getrennt. Es gibt mehrere Möglichkeiten eine Liste zu erzeugen. Wenn die Einträge bzw. die Elemente einer Liste bereits bekannt sind, können diese direkt beim Erzeugen angegeben werden.

Quelle [1]

Unsere erste Liste soll *kinderbuecher* heißen, in der wir Titeln einiger Kinderbücher speichern wollen. Zuerst geben wir den Listennamen an, also *kinderbuecher*. Danach kommt der Zuweisungsoperator. Jetzt geben wir mit den eckigen Klammern an, dass es sich um eine Liste handelt.

Jetzt können wir innerhalb der Klammer unsere Elemente angeben, die wir voneinander durch Kommas trennen. Bei Buchtiteln handelt es sich um Texte, die wir als Zeichenketten speichern, also müssen wir diese in Anführungszeichen angeben. Dann schreiben wir „Peter Pan“, „Pippi Langstrumpf“ und „Alice im Wunderland“.

```
kinderbuecher = ["Peter Pan", "Pippi Langstrumpf", "Alice im Wunderland"]
```

Wenn du wissen möchtest, ob du einen Fehler bei der Erzeugung der Liste gemacht hast, kannst du einfach den Code ausführen. Das machen wir jetzt. Es gibt keine syntaktischen Fehler, also wurde unsere Liste erzeugt.

Wir können die gesamte Liste auch mit dem *print()*-Befehl ausgeben. Dazu schreiben wir den Listennamen in der Klammer der *print()*-Anweisung. Die Listen werden in Python genauso ausgegeben, wie sie definiert wurden, vielleicht bis auf die Art der Anführungszeichen.

```
print(kinderbuecher)
```

So, wir haben eine Liste mit drei String-Elementen erzeugt. Jetzt können wir eine weitere Liste mit Zahlen z. B. Erscheinungsjahren der Bücher erstellen. Wir nennen diese Liste *erscheinungsjahr*. Laut Google ist „Peter Pan“ im Jahr 1911, „Pippi Langstrumpf“ 1945 und „Alice im Wunderland“ 1865 zum ersten Mal erschienen.

```
erscheinungsjahr = [1911, 1945, 1865]
```

Wir können auch diese Liste mit der *print()*-Anweisung ausgeben lassen.

```
print(erscheinungsjahr)
```

Im Unterschied zu vielen anderen Programmiersprachen kann man in Python in einer Liste Elemente unterschiedlicher Datentypen speichern. Wir können zum Beispiel eine Liste erstellen, in der wir die Basisinformationen zu einem Buch speichern. Wir nennen diese Liste *kinderbuch_1*. In dieser Liste speichern wir den Titel, den Autor und das Erscheinungsjahr eines Buchs. Also geben wir

```
kinderbuch_1 = ["Pippi Langstrumpf", "Astrid Lindgren", 1945]
```

ein. Auch diese Liste können wir mit der *print()*-Anweisung ausgeben lassen.

```
print(kinderbuch_1)
```

Diese Art von Listen wird oft genutzt, um tabellenartige Strukturen zu erzeugen. Man kann kleine Listen erzeugen, die einzelne Tabellenzeilen darstellen und später diese zu einer verschachtelten Liste zusammenfügen. So erstellen wir eine weitere Liste mit Informationen zu einem Buch und nennen diese *kinderbuch_2*.

```
kinderbuch_2 = ["Alice im Wunderland", "Lewis Carroll", 1865]

print(kinderbuch_2)
```

Jetzt können wir eine übergeordnete Liste erstellen, die diese beiden Listen als Unterlisten enthält. Da wir schon eine Liste namens *kinderbuecher* haben, nennen wir diese Liste einfach *buchliste*.

```
buchliste = [kinderbuch_1, kinderbuch_2]
```

Wir geben auch diese Liste mit der *print()*-Anweisung aus.

```
print(buchliste)
```

In der Ausgabe sieht man, dass *buchliste* zwei Listen als Elemente enthält.

Bis jetzt haben wir Listen mit vorher festgelegten Elementen erstellt. Wenn die Elemente einer Liste am Anfang noch nicht bekannt sind und erst im Laufe des Programms generiert werden, kann zuerst eine leere Liste erstellt werden und dieser nach und nach Elemente hinzugefügt werden.

Wir erstellen eine leere Liste *krimis*, indem wir nach dem Zuweisungsoperator eckige Klammer auf und Klammer zu eingeben.

```
krimis = []
```

Wenn wir diese Liste ausgeben lassen, sehen wir, dass sie keine Elemente enthält.

```
print(krimis)
```

Um ein Element der Liste *krimis* hinzuzufügen, benutzt man die Funktion *append()* und zwar so, dass man nach dem Listennamen einen Punkt schreibt, danach kommt die Funktion *append()* und in der Klammer das Element, welches man einfügen möchte, also „Mord im Orient-Express“.

```
krimis.append("Mord im Orient-Express")
```

Wenn wir unsere Liste jetzt ausgeben, sehen wir, dass sie ein Element enthält.

```
print(krimis)
```

Wir fügen noch den Krimi „Der Pate“ unserer Liste hinzu. Dafür geben wir wieder

```
krimis.append("Der Pate")
```

ein. Wir geben unsere Liste *krimis* mit der *print()*-Anweisung aus und sehen, dass „Der Pate“ am Ende der Liste einzugefügt wurde.

```
print(krimis)
```

Um ein Element an eine bestimmte Stelle in die Liste einzufügen, benutzt man die Funktion *insert().* Wie bei der *append()*-Funktion schreibt man zuerst den Listennamen, danach einen Punkt, dann die Funktion *insert()*. In der Klammer der Funktion gibt man zuerst die Position an, an die das neue Element eingefügt werden soll. Wir wollen den Krimi „Alibi“ an die zweite Position einfügen, also geben wir den Index 1 an. Nach dem Komma geben wir das Element selbst, also „Alibi“ an.

```
krimis.insert(1, "Alibi")
```

Wenn wir jetzt die Liste ausgeben, sehen wir, dass „Alibi“ an der zweiten Stelle in der Liste steht und „Der Pate“ an die dritte Stelle verschoben wurde.

```
print(krimis)
```

Zugriff auf Listenelemente

Um auf die einzelnen Listenelemente zugreifen zu können, muss man hinter dem Listennamen in den eckigen Klammern den Index des Elements angeben. Wenn wir zum Beispiel wissen wollen, welches Kinderbuch in der Liste *kinderbuecher* an der zweiten Stelle steht, müssen wir

```
kinderbuecher[1]
```

eingeben. Wir kombinieren dies direkt mit der *print()*-Anweisung, damit wir die Ausgabe direkt sehen können. An der zweiten Stelle steht also „Pippi Langstrumpf“.

```
print(kinderbuecher[1])
```

Genauso können wir auch negative Indexe angeben. Das ist zum Beispiel dann praktisch, wenn wir nicht wissen, wie lang unsere Liste ist und wir auf das letzte Element der Liste zugreifen wollen. Denn das letzte Element einer Liste hat den negativen Index -1.

Auf diese Weise können wir das letzte Element der Liste *kinderbuecher* ausgeben.

```
print(kinderbuecher[-1])
```

Das ist „Alice im Wunderland“.

Jetzt schauen wir uns noch an, was passiert, wenn wir einen zu hohen Index in der eckigen Klammer angeben. Unsere Liste *kinderbuecher* hat nur drei Elemente, der höchste Index ist also 2. Wir versuchen jetzt das Element mit Index 3 auszugeben. Dafür geben wir Folgendes ein:

```
print(kinderbuecher[3])
```

Da es kein Element an dieser Position in der Liste gibt, bekommen wir eine Fehlermeldung, die besagt, dass unser Index außerhalb des Bereichs liegt. Diese Zeile kommentieren wir direkt wieder aus.

Damit Dir solche Fehler nicht passieren, kannst du mit der Funktion *len(),* die für length steht, die Anzahl der Elemente einer Liste ausgeben. Das machen wir jetzt für die Liste *kinderbuecher*. Also tippen wir folgende Anweisung ein:

```
print(len(kinderbuecher))
```

Wie erwartet enthält unsere Liste *kinderbuecher* nur drei Elemente.

Wenn wir nur eine Teilmenge aufeinanderfolgender Elemente einer Liste ausgeben wollen, zum Beispiel nur die ersten beiden Elemente, können wir in der eckigen Klammer den Start- und den End-Index getrennt durch einen Doppelpunkt angeben. Dabei wird das Element an der Endposition nicht mit ausgegeben. Diese Methode nennt man Slicing. Wir schneiden so gesehen eine Scheibe aus der Liste aus.

Wir wollen jetzt die ersten beiden Elemente der Liste *kinderbuecher* ausgeben. Dazu müssen wir Folgendes eingeben:

```
print(kinderbuecher[0:2])
```

Wenn wir die Elemente ab einer bestimmten Position bis zum Ende der Liste ausgeben wollen, geben wir nur die Anfangsposition und den Doppelpunkt in den eckigen Klammern an. Mit der Anweisung

```
print(kinderbuecher[1:])
```

geben wir alle Kinderbücher ab der zweiten Position in der Liste aus.

Möchten wir alle Elemente einer Liste von Anfang bis zu einer bestimmten Position ausgeben, müssen wir nur die Endposition nach dem Doppelpunkt in den eckigen Klammern angeben. Mit der Anweisung

```
print(kinderbuecher[:2])
```

werden nur die ersten beiden Kinderbücher ausgegeben.

Um auf die einzelnen Elemente einer Unterliste in einer verschachtelten Liste zugreifen zu können, muss man zwei Indexe angeben. Der erste Index gibt die Position der Unterliste an. Der zweite Index gibt das Element innerhalb der Unterliste an.

Wir haben bereits eine verschachtelte Liste erstellt, die *buchliste* heißt. Um das Erscheinungsjahr vom ersten Buch ausgeben zu lassen, müssen wir hinter dem Listennamen zuerst in den eckigen Klammern den Index 0, weil wir auf das erste Buch zugreifen wollen, und in den zweiten eckigen Klammern Index 2 angeben, weil das Erscheinungsjahr an der dritten Stelle steht.

```
print(buchliste[0][2])
```

Wenn wir nur einen Index für eine verschachtelte Liste angeben, dann wird die gesamte Unterliste, die an dieser Position steht, ausgegeben.

```
print(buchliste[0])
```

Elemente innerhalb einer Liste ersetzen und löschen

Wir haben bereits gesehen, dass wir die Listen ändern können, indem wir ihnen neue Elemente hinzufügen. Wir können aber auch Listen verändern, indem wir ihre Elemente ersetzen oder löschen.

Um ein Element der Liste durch ein anderes Element zu ersetzen, müssen wir dieses Element mit Hilfe seines Indexes ansprechen und ihm einen neuen Wert zuweisen. Das schauen wir uns am Beispiel der Liste *krimis* an. Zuerst geben wir die Elemente dieser Liste aus.

```
print(krimis)
```

Unsere Liste hat drei Elemente. Jetzt ersetzen wir den ersten Eintrag der Liste durch einen anderen Krimi. Wir greifen auf dieses Element durch seinen Index zu und ersetzen es durch einen neuen Wert.

Wenn wir jetzt die Liste ausgeben, sehen wir, dass das erste Element ausgetauscht wurde. Die übrigen Elemente sind aber gleich geblieben.

```
krimis[0] = "Der dünne Mann"
print(krimis)
```

Das Löschen der Listenelemente erfolgt mit dem *del*-Befehl. Wenn wir also ein oder mehrere Elemente aus unserer Liste entfernen möchten, müssen wir zuerst den *del*-Befehl eingeben und danach das Element bzw. die Elemente, die gelöscht werden sollen. In unserem Beispiel löschen wir die ersten zwei Elemente. Wenn wir die Liste ausgeben, sehen wir, dass sie nur noch ein Element enthält.

```
del krimis[0:2]
print(krimis)
```

Wir können auch alle Elemente einer Liste löschen, indem wir den Listennamen gefolgt von einem Punkt und der Methode *clear()* eingeben. Die Liste bleibt noch definiert, aber sie ist dann leer.

```
krimis.clear()
print(krimis)
```

Zugehörigkeiten der Elemente zu einer Liste

Um prüfen zu können, ob ein Element in einer Liste enthalten ist, können wir den *in*-Operator verwenden. Dazu geben wir zuerst das Element, gefolgt von *in*-Operator und dem Listennamen an.

```
'Peter Pan' in kinderbuecher
```

Wenn das Element in der Liste enthalten ist, wird ein *True*, also *Wahr* als Ergebnis geliefert. Wenn das Element in der Liste nicht vorkommt, wie zum Beispiel „Der Pate“ in der Liste *kinderbuecher*, dann erhalten wir *False* als Ergebnis.

```
'Der Pate' in kinderbuecher
```

Wir können das Vorkommen eines Elementes in der Liste auch mit der Funktion *count()* implizit prüfen. Diese Funktion berechnet, wie oft ein Element in der Liste vorkommt. Dafür geben wir den Listennamen, gefolgt von einem Punkt und der Funktion *count()* ein. In der Klammer der Funktion geben wir das gesuchte Element an.

```
kinderbuecher.count('Peter Pan')
```

```
kinderbuecher.count('Der Pate')
```

In der Ausgabe sehen wir, dass „Peter Pan“ in der Liste *kinderbuecher* einmal und „Der Pate“ Null mal vorkommt.

Verkettung von Listen

Zum Schluss schauen wir uns noch an, wie man die Listen in Python verkettet. Man spricht von der Verkettung bzw. Konkatenation von Listen, wenn man Elemente einer Liste mit den Elementen einer anderen Liste zusammenfügt. Dabei wird die Reihenfolge der Elemente nicht verändert.

Da wir aus der Liste *krimis* alle Elemente entfernt haben, fügen wir ihr diese noch mal hinzu.

```
krimis = ['Der dünne Mann', 'Alibi', 'Der Pate']
print(krimis)
```

In der Liste *kinderbuecher* haben wir auch drei Elemente.

```
print(kinderbuecher)
```

Man verkettet die Listen in Python mit dem +-Operator, indem man zuerst den Namen der verketteten Liste angibt, dann den Zuweisungsoperator, gefolgt von dem ersten Listennamen, dem +-Operator und dem Namen der zweiten Liste. Eigentlich kann man auch mehr als zwei Listen in einer Anweisung miteinander verketten. Dazu verknüpft man weitere Listennamen mit dem +-Operator.

Wir nennen unsere neue Liste *buecher*, in der wir zuerst die Elemente der Liste *krimis* und dann die Elemente der Liste *kinderbuecher* speichern.

```
buecher = krimis + kinderbuecher
```

Wenn wir jetzt die Liste *buecher* ausgeben, sehen wir, dass zuerst alle Elemente der Liste *krimis* und dann erst alle Elemente der Liste *kinderbuecher* in die neue Liste aufgenommen wurden.

```
print(buecher)
```

Take-Home Message

In diesem Video hast du gelernt, wie man in Python Listen erzeugt und diesen Elemente hinzufügt, wie man auf die Listenelemente zugreift, sie ersetzt oder löscht. Außerdem hast

du gesehen, wie man prüft, ob ein Element in der Liste vorkommt und wie man die Listen verkettet.

Quellen

Quelle [1] Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Weiterführendes Material

Schmitt, S. (2021). *Python Kompendium: Professionell Python Programmieren lernen.* BMU Media Verlag

Barry, P. (2017). *Python von Kopf bis Fuß.* O'Reilly

Disclaimer

Transkript zu dem Video „Woche 03: Programmierung - Listen in Jupyter Notebook", Ludmila Himmelspach.