Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: 01-02 (Was ist Python?), 02-01 (Variablen), 02-02 (Strings)

Einführung in die computerlinguistische Programmierung mit Python

02-03: Listen 📄

Eine Liste in Python kann eine beliebige Mischung von Werten verschiedener Datentypen enthalten. Listen haben eine feste Reihenfolge. In den meisten Anwendungsfällen ist es nützlich, wenn wir Listen als Sinneinheiten begreifen, beispielsweise als sortierte Sammlungen von Wörtern, Zahlen, Wahrheitswerten o.ä. Wir können auch komplexere Datentypen verwenden, oder Listen ineinander verschachteln. Wenn wir dann die Methode `len()` verwenden, um die Anzahl der Elemente zu ermitteln, werden die Elemente auf der obersten Verschachtelungsebene gezählt:

In [ ]:

```
elemente = [1, 2, 3]
print(elemente)
print(len(elemente))
```

In [ ]:

```
komplexe_liste = [[1,2,3], ["a", "b", "c"], True]
print(komplexe_liste)
print(len(komplexe_liste))
```

Anders als Strings können Listen verändert werden, ohne dass man extra eine Kopie erstellen muss. **Achtung:** Diese Operationen haben daher keinen Rückgabewert. Wir dürfen also nicht den Wert der Variable überschreiben!

In [ ]:

```
my_list = [1,2,3]
print(my_list)

my_list.append(4)   # das Element 4 an die Liste anhängen
print(my_list)      # :)
```

In [ ]:

```
my_list = [1,2,3]
print(my_list)

my_list = my_list.append(5)
print(my_list)      # :(
```

Die folgende Tabelle enthält einige Operationen für Listen, die für uns wichtig sind.

Operation	Auswirkung
`l.append(element)`	Das `element` wird ans Ende der Liste angehängt.
`l1.extend(l2)`	Ergänzt die Liste `l1` um alle Elemente von `l2` in der originalen Reihenfolge.
`l.insert(i, element)`	An der Position `i` in der Liste wird das `element` eingefügt.
`l.remove(element)`	Entfernt das erste Vorkommen von `element` aus der Liste (schlägt fehl, falls das Element nicht in der Liste enthalten ist).
`l.reverse()`	Dreht die Liste um.
`l.sort()`	Sortiert die Liste. Schlägt fehl, wenn Sortierung nicht möglich ist (z.B. wenn unterschiedliche Datentypen in der Liste enthalten sind).
`l.pop()`	Entfernt das letzte Element der Liste. Außerdem **Rückgabe des letzten Elements**. Schlägt fehl, wenn die Liste leer ist.
`l.pop(i)`	Entfernt das Element an Position `i` aus der Liste und **gibt es zurück**; schlägt fehl, falls die Position `i` in der Liste nicht existiert.
`element in l`	**Rückgabe: Bool** - `True`, wenn das `element` in der Liste enthalten ist; sonst `False`.
`min(l), max(l)`	**Rückgabe** des kleinsten bzw. größten Elements in der Liste.
`len(l)`	**Rückgabe** der Anzahl von Elementen in der Liste.
`l.count(element)`	**Rückgabe: Integer** - Anzahl der Vorkommen von `element` in der Liste.
`l.index(element)`	**Rückgabe: Integer** - Position des ersten Vorkommens von `element` in der Liste. Schlägt fehl, wenn das Element nicht in der Liste vorkommt.

Zum Schluss noch eine Operation, die als Stringoperation definiert ist, die wir aber meist in Kombination mit Listen verwenden: `join()`. So wird `join` verwendet:

In [ ]:

```
einzelne_woerter = ["Eis", "Schokolade", "Chips", "Kuchen"]
verbindungs_string = " +++ "

print(verbindungs_string.join(einzelne_woerter))
```

Zusammenfassung

- Listen sind sortierte Sequenzen von Elementen beliebiger Typen.
- Listen können ineinander verschachtelt werden.
- Listen können "in place" verändert werden, sodass wir keine Rückgabewerte erhalten. (Mehr dazu in Thema 08-01.)
- Wir können Listen zusammenfügen, einzelne Elemente anhängen oder entfernen, die Länge einer Liste bestimmen, oder prüfen, ob ein einzelnes Element in einer Liste enthalten ist.

Weitere Themen dieser Woche

- 02-01: Variablen
- 02-02: Strings
- 02-04: Integers und Floats (Zahlen)
- 02-05: Booleans (Wahrheitswerte)