Dictionaries

by Woche 2

Der vierte Typ von Sammlungen in Python sind Dictionaries. Auch Dictionaries sind Listen sehr ähnlich: Sie sind nicht homogen, Verschachtelungen sind möglich, sie sind geordnet[1] und mutable. Dictionaries unterscheiden sich aber auch von Listen: Anstelle von Elementen, enthalten sie Key-Value-Paare (Schlüssel-Value-Paare) und werden mit `{}` erstellt.

Arbeiten mit Dictionaries

Erstellen

Dictionaries werden - wie auch Sets - mit geschweiften Klammern `{}` erstellt. Ein Dictionary besteht im Gegensatz zu den drei anderen Sammlungstypen allerdings aus Key-Value-Paaren, wobei der Key eindeutig sein muss. Demnach können also die Values Duplikate enthalten, die Key aber nicht. Die Key-Value-Paare werden durch ein `:` getrennt und die einzelnen Paare durch ein `,`.

Der Key kann dabei ein beliebiges immutable Objekt sein, wie z.B. ein String, Integer oder ein Tupel. Der Value kann ein beliebiges Objekt sein (also immutable **oder** mutable), wie z.B. ein String, Integer, Tupel, Liste, Set oder ein anderes Dictionary.

Wie bei Listen kann mittels eckiger Klammern auf einzelne Elemente zugegriffen werden. Dabei muss im Gegensatz zu Listen nicht der Index, sondern der Key angegeben werden. Ausgegeben wird dann der Value des Keys.

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}



artikel['Burger']
```

```
4.99
```

```
x = {'Name': 'Donald',
     'Jahr': 1934,
     'Freunde': ['Daisy', 'Mickey', 'Goofy']}
```

[1] Seit Python 3.7 sind Dictionaries geordnet. Das bedeutet, dass die Reihenfolge der Elemente im Dictionary erhalten bleibt.

```
x['Jahr']
```

```
1934
```

Darüber hinaus können Dictionarys auch mittels der Funktion `dict()` wie folgt erstellt werden:

```
x = dict(Name='Donald',
         Jahr=1934,
         Freunde=['Daisy', 'Mickey', 'Goofy'])
```

Auch dictionaries können in andere Sammlungstypen umgewandelt werden. Dabei wird die ursprüngliche Sammlung nicht verändert, sondern eine neue Sammlung erstellt. Standardmäßig wird dabei nur der Key umgewandelt. Um stattdessen den Value umzuwandeln, kann die Methode `values()` verwendet werden.

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
list(artikel)
```

```
['Burger', 'Pommes']
```

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
list(artikel.values())
```

```
[4.99, 2.99]
```

Um umgekehrt aus einem anderen Sammlungstyp ein Dictionary zu erstellen, gibt es mehrere Ansätze. Falls vorerst nur die Keys vorhanden sind, kann kann die Methode `fromkeys()` verwendet werden. Dabei wird für jeden Key ein Value erstellt, der standardmäßig `None` ist. Liegen die Keys und Values in zwei getrennten Sammlungen vor, kann die Methode `zip()` verwendet werden. Dabei werden die beiden Sammlungen paarweise zusammengeführt.

```
schluessel_liste = ['a', 'b', 'c']
#
dict.fromkeys(schluessel_liste)
```

```
{'a': None, 'b': None, 'c': None}
```

```
schluessel_tuple = ('a', 'b', 'c')
wert_tuple = (10, 20, 30)
dict(zip(schluessel_tuple, wert_tuple))
```

```
{'a': 10, 'b': 20, 'c': 30}
```

Hinzufügen / Verändern

Anstelle von `append()` oder `insert()` wie bei Listen gibt es bei Dictionaries die Methode `update()`. Diese fügt ein Key-Value-Paar hinzu, falls der Key noch nicht im Dictionary enthalten ist. Falls er bereits enthalten ist, wird der Value des Keys überschrieben. Anstatt die Methode anzuwnden, kann beides aber auch erreicht werden, indem der Key direkt angesprochen wird.

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
artikel.update({'Burger': 5.99})
artikel.update({'Cola': 1.99})
artikel
```

```
{'Burger': 5.99, 'Pommes': 2.99, 'Cola': 1.99}
```

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
artikel['Burger'] = 5.99
artikel['Cola'] = 1.99
artikel
```

```
{'Burger': 5.99, 'Pommes': 2.99, 'Cola': 1.99}
```

Der Unterschied ist vor allem, dass bei `update()` mehrere Key-Value-Paare hinzugefügt werden können. So hätte das erste Beispiel auch mit einer Zeile weniger auskommen können: `artikel.update({'Burger': 5.99, 'Cola': 1.99})`.

Entfernen

Anstelle von `remove()` wie bei Listen gibt es bei Dictionaries die Methode `pop()` aber auch die Funktion `del()`. Die `pop()`-Methode entfernt ein Key-Value-Paar und gibt den Value zurück. Im Unterschied zur selben Methode bei Listen, muss bei Dictionaries der Key angegeben werden, der entfernt werden soll.

```
artikel = {'Burger': 4.99, 'Pommes': 2.99, 'Cola': 1.99}
x = artikel.pop('Burger')
```

```
print(x)
print(artikel)
```

```
4.99
{'Pommes': 2.99, 'Cola': 1.99}
```

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}

del(artikel['Pommes'])
artikel
```

```
{'Burger': 4.99}
```

Prüfen

Wie bei Listen gibt es auch bei Dictionaries z.B. die Möglichkeit via `in` zu prüfen, ob ein Element enthalten ist. Diese prüft allerdings nur die Keys und nicht die Values. Man kann übrigens auch explizit `artikel.keys()` anstatt `artikel` (analog zu `artikel.values()`) schreiben, um zu verdeutlichen, dass die Keys geprüft werden.

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
4.99 in artikel.keys() # oder nur artikel
```

```
False
```

```
artikel = {'Burger': 4.99, 'Pommes': 2.99}
4.99 in artikel.values()
```

```
True
```

Tabellendaten

Dictionaries eignen sich auch, um Tabellendaten zu speichern. Dabei wird jeder Key als Spaltenname und jeder Value als Zeile interpretiert. So kann z.B. eine Tabelle mit den Spalten “Name”, “Alter” und “Geschlecht” als Dictionary dargestellt werden. So lässt sich diese Tabelle

Name	Jahr	Tier
Donald	1934	Ente
Daisy	1937	Ente

Name	Jahr	Tier
Mickey	1928	Maus

wie folgt als Dictionary ausdrücken:

```
dict_tabelle = {'Name': ['Donald', 'Daisy', 'Mickey'],
                'Jahr': [1934, 1937, 1928],
                'Tier': ['Ente', 'Ente', 'Maus']}
```

Allerdings werden wir in diesem Kurs noch eine bessere Möglichkeit kennenlernen, um mit Tabellendaten zu arbeiten: DataFrames. Diese sind speziell für Tabellendaten konzipiert und bieten viele nützliche Funktionen, um diese zu verarbeiten. Dazu aber später mehr.

Weitere Ressourcen

- Python Tutorial #23 (deutsch) - Dictionary **nur bis 8:26**
- ALL 11 Dictionary Methods In Python EXPLAINED