

Tupel und Dictionaries

Richard Müller, Tom Felber

11. November 2021

Python-Kurs

1. Wiederholung

2. Tupel

3. Dictionaries

Wiederholung

Beim letzten Mal

- Booleans mit `and or not`
- Funktionen

```
1 def potenz(x, y):  
2     # x^y wird zurückgegeben  
3     return x**y  
4  
5 p = potenz(2, 3) # p = 8
```

- Selsames Verhalten von Listen im Vergleich zu Zahlen

Tupel

Tupel

Der Tupel (`tuple`) ist ein Datentyp, der ähnlich der Liste ist.

Eigenschaft	Liste	Tuple
enthält mehrere Elemente	✓	✓
geordnet	✓	✓
kann mit <code>[i]</code> indexiert werden (Elemente herausgreifen)	✓	✓
Elemente können angehängt / gelöscht werden	✓	✗
Werte von Elementen können neu zugewiesen werden	✓	✗
Referenzverhalten	mutable	immutable

Tupel - Beispiele

Initialisierung ähnlich der Liste, `()` statt `[]`

```
1 # Initialisierung mit ()
2 tuple_eins = (1, True, 4.5, "hallo")
```

Einzelne Elemente herausgreifen, genau wie bei Listen. Slices funktionieren auch.

```
1 viereinhalb = tuple_eins[2] #4.5
2 hallo = tuple_eins[-1] #'hello'
3 vordere = tuple_eins[:2] #(1, True)
```

Eine Liste zu einem Tuple umwandeln

```
1 # Liste zu Tuple umwandeln
2 liste = [1,2,3]
3 tuple_aus_liste = tuple(liste)
```

Aber! Elemente neu zu setzen ist nicht möglich:

```
1 tupel_eins[0] = "neuer Wert" #TypeError
```

Nur das gesamte Tuple kann neu gesetzt werden:

```
1 tupel_eins = ("neuer Wert", True, 4.5, "hallo")
```


Tuples - Warum ?

Warum Tuples benutzen wenn Listen mehr können ? Es kommt auf den Fall an.

Tuples sind gut für:

- statische Daten: verhindern, dass ausversehen Daten verändert werden, die als Konstanten gemeint sind
- vielen Daten (die keine Listenfunktionalität benötigen): werden schneller verarbeitet als Listen
- als Keys für Dictionaries: da sie immutable sind, können Tuples als Key verwendet werden (Listen nicht)

Dictionaries

Dictionaries - Einleitung



Dictionaries - Erstellen / Lesen / Löschen

In Dictionaries können **Key - Value** Paare abgelegt werden.
Die Keys müssen dabei einzigartig sein. Die Values nicht.
Keys werden links, Values rechts angegeben.

```
1 # ein dictionary erstellen
2 lexikon = dict()
3 lexikon = {}
4 lexikon = {"Haus": "Substantiv", "stehlen": "Verb", "Geld":
    "Substantiv"}
```

Mit dem Key kann der zugehörige Wert gelesen werden.

```
1 # einen wert per key auslesen
2 x = lexikon["Haus"] # "Substantiv"
```

Oder gelesen und gleichzeitig gelöscht werden.

```
1 lexikon.pop("Haus") # "Substantiv"
2 print(lexikon)     #{ "stehlen": "Verb", "Geld": "Substantiv" }
```

Dictionaries - Paare hinzufügen / überschreiben

Die Syntax zum hinzufügen kann genauso benutzt werden, um einen vorhandenen Key zu überschreiben.

```
1 lexikon = {"Haus": "Substantiv", "stehlen": "Verb", "Geld":  
            "Substantiv"}
```

```
1 # einen neues paar einfügen  
2 lexikon["neu"] = "Adjektiv"  
3 # einen wert überschreiben  
4 lexikon["Haus"] = "Verb"
```

Dictionaries - Paare hinzufügen / überschreiben

Die meisten Typen können als Key benutzt werden. Listen und Dictionaries bilden hierzu eine Ausnahme, sie sind als Key nicht zulässig.

```
1 # key Beispiele
2 lexikon[1] = "eins"
3 lexikon[5.0123] = "close to five"
4 # geht nicht:
5 liste = ["Liste"]
6 lexikon[list] = "Subjekt"
```