

Tupel und Dictionaries

Richard Müller, Tom Felber

11. November 2021

Python-Kurs

Gliederung

1. Wiederholung

2. Tupel

3. Dictionaries

Wiederholung

Wiederholung

Beim letzten Mal

- Booleans mit and or not
- Funktionen

```
def potenz(x, y):
    # x^y wird zurückgegeben
    return x**y

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	✓	1
geordnet	✓	1
kann mit [i] indexiert werden (Elemente herausgreifen)	✓	1
Elemente können angehangen / gelöscht werden	1	X
Werte von Elementen können neu zugewiesen werden	✓	×
Referenzverhalten	mutable	immutable

3

Tupel - Beispiele

Initialisierung ähnlich der Liste, () statt []

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

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

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

Eine Liste zu einem Tuple umwandeln

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

Tupel - Beispile

Aber! Elemente neu zu setzen ist nicht möglich:

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

Nur das gesamte Tuple kann neu gesetzt werden:

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

Tuples - Warum ?

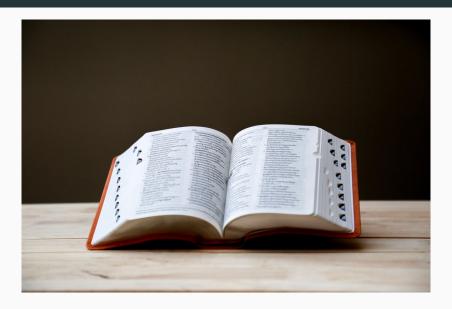
Warum Tuples benutzen wenn Listen mehr können? Es kommt auf den Fall an.

Tuples sind gut für:

- statische Daten: verhindern, dass ausversehen Daten verändert werden, die als Konstanten gemeint sind
- vielen Daten (die keine Listenfunktionialität 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.

Mit dem Key kann der zugehörige Wert gelesen werden.

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

Oder gelesen und gleichzeitig gelöscht werden.

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

Dictionaries - Paare hinzufügen / überschreiben

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

```
# einen neues paar einfügen
lexikon["neu"] = "Adjektiv"
# einen wert überschreiben
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.

```
# key Beispiele
lexikon[1] = "eins"
lexikon[5.0123] = "close to five"

# geht nicht:
liste = ["Liste"]
lexikon[liste] = "Subjekt"
```