Python Kurs 2019/2020

5: Dictionaries

Bernhard Mallinger

`b.mallinger [at] gmx.at`

https://totycro.github.io/python-kurs

Dictionaries

- Dictionaries ordnen Werte andere Werte zu (Key-Value)
- Z.B. Wörterbuch, Telefonbuch:

```
dictionary = {
    "language": "Sprache",
    "value": "Wert",
}
print(dictionary["value"])
```

Output:

```
Wert
```

Dictionaries

Key und Value können beliebige Typen haben

```
natural_logarithm_table = {
    5: 1.6094379124341003,
    6: 1.791759469228055,
    7: 1.9459101490553132,
    8: 2.0794415416798357,
}
prices = {
    'apple': 3,
    'pear': 2,
}
course = {
    "participants": ["Alice", "Bob"],
    "weekday": "Thursday",
    "rating": 4,
}
```

Dictionaries

Keys können aber nur unveränderliche Werte sein:

- Wenn der Key verändert werden würde, wäre das Dictionary verwirrt
- z.B. Zahlen, Strings, Tuple gehen
- Listen oder Dictionaires gehen nicht

```
>>> l = [1, 2]
>>> d = {l: 3}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Einschub: Tuple

- Tuple sind sehr ähnlich wie Listen, können aber nicht verändert werden
  ⇒ kein `append()`, kein `remove()`, kein `tup[1] = 'a'`
  ⇒ `(0, 1) + (2, )` aber möglich (erzeugt neuen Tuple)
- Schreibweise: Wie Listen, aber `()` statt `[]`
  (Sonderfall Tuple mit 1 Element: `('a', )` statt `(a)`)

```
>>> d = {
    (0, 1): 3,
    (3, 1): 2,
    }
>>> d[ (0, 1) ]
3
```

Dictionaries: Zugriff

Zugriff mit `[]` gibt Element oder Error:

```
>>> prices["apple"]
3
>>> prices["banana"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'banana'
```

Zugriff mit `get()` erlaubt Angabe von default-Wert:

```
>>> prices.get("banana", 0)
0
```

Dictionaries: Auflisten

Keys:

```
for product in prices:
    print(product)  # apple, pear
for product in prices.keys():
    print(product)  # apple, pear
```

Items:

```
for product, price in prices.items():
    print(product, price)  # apple 3, pear 2
```

Values:

```
for price in prices.values():
    print(price)  # 3, 2
```

Dictionaries: Beispiele

Nützliche Anwendungen:

```
cheap_products = [
    product
    for product, price in prices.items()
    if price < 3
]

highest_price = max(prices.values())

number_of_products = len(prices)

# "in" arbeitet mit Keys (vgl. Telefonbuch)
have_apples = "apple" in prices
```

Teuerstes Produkt: 3 Möglichkeiten

```
most_expensive_product = [
     product
     for product, price in prices.items()
     if price == highest_price
][0]
```

```
highest_price = 0
most_expensive_product = ""
for product, price in prices.items():
     if price > highest_price:
         highest_price = price
         most_expensive_product = product
```

```
most_expensive_product = max(
    (price, product) for product, price in prices.items()
)[1]
```

Dictionaries: Verändern

Syntax zum Einfügen/Updaten wie bei Listenindices

```
>>> prices
{'apple': 3, 'pear': 2}
>>> prices["banana"] = 5
>>> prices
{'apple': 3, 'pear': 2, 'banana': 5}
>>> prices["apple"] = 2
>>> prices
{'apple': 2, 'pear': 2, 'banana': 5}
```

Löschen auch:

```
>>> del prices['pear']
>>> prices
{'apple': 2, 'banana': 5}
```

Bei neueren Python-Versionen sind die Keys nach der Einfügereihenfolge gereiht.

My courses

AUFGABE

Erstelle eine Liste von Dictionaries, wobei die Dictionaries deine Lehrveranstaltungen aus diesem Semester beschreiben. Als Keys sollen zumindest `name`, `lecturer` und `grade` enthalten sein. Falls du noch keine Note bekommen hast, lass das `grade`-Feld aus.

Schreibe dann Code, der jeweils folgendes berechnet:

1. Alphabetisch sortierte Liste der Namen der Kurse
2. Liste der Namen der Lehrenden (ohne Duplikate)
3. Namen der Kurse, die noch keine Note haben
4. Namen der Kurse mit der Note 1, 2 oder 3
5. Füge einen neuen Kurs hinzu
6. Trag eine Note für einen bestehenden Kurs ein

Dictionary comprehensions

Wie list comprehensions, aber mit `{}` statt `[]` und `key: value` statt nur Werten

```
>>> {product: price * 2 for product, price in prices.items()}
{'apple': 6, 'pear': 4, 'banana': 10}
>>> {
        product + " (new formula)": price
        for product, price in prices.items()
        if product != 'banana'
    }
{
    'apple (new formula)': 3,
    'pear (new formula)': 2,
}
```

Dictionaries: `setdefault`

Nützliche, aber schlecht benannte Operation
`d.setdefault(key, default)` ist äquivalent zu:

```
if key not in d:
    d[key] = default
return d[key]
```

```
course = {"title": "Holzverarbeitung 1"}
[...]
rating = course.setdefault("rating", 3)
# course enthält jetzt "rating" mit dem Wert 3
print("Bewertung:", rating)
```

```
course.setdefault("participants", []).append("Alex")
```

Einschub: Imports

2 wesentliche Möglichkeiten:

```
import collections
collections.Counter()
```

```
from collections import Counter
Counter()
```

Man kann auch alle Namen eines Moduls importieren, das ist aber oft verwirrdend:

```
from math import *
from collections import *
from datetime import *

# jeder Name hier könnte von jedem Modul kommen
```

Pretty Printing

`pprint`-Funktion aus dem `pprint`-Modul macht verschachtelte Strukturen schöner

```
>>> wastewater_measurement
{'ph': {'2014-01-01': 5.5, '2014-04-01': 5.629, '2014-06-01': 5.31,
'2014-09-01': 5.88}, 'toxic': False, 'site': ['Street 12',
'A-1000 Vienna'], 'due_for_inspection': True}
>>> from pprint import pprint
>>> pprint(wastewater_measurement)
{'due_for_inspection': True,
 'ph': {'2014-01-01': 5.5,
        '2014-04-01': 5.629,
        '2014-06-01': 5.31,
        '2014-09-01': 5.88},
 'site': ['Street 12', 'A-1000 Vienna'],
 'toxic': False}
```

Vorkommnisse zählen

AUFGABE

Schreibe eine Funktion, welche die Vorkommnisse von Elementen in einem Iterable zählt:

```
>>> count_occurrences([1, 3, 2, 1, 2, 2])
{1: 2, 3: 1, 2: 3}
>>> count_occurrences("access")
{'a': 1, 'c': 2, 'e': 1, 's': 2}
```

- Die Aufgabe ist mit normalen Dictionaryoperationen gut lösbar
- Die weiterführende Datenstruktur `defaultdict` aus `collections` kann helfen.
- `Counter` aus `collections` löst diese Aufgabe auch. Verwende `Counter` in produktivem Code!

Vorkommnisse zählen 2

AUFGABE

Schreibe eine Funktion, die die Verteilung der Anzahl der KursteilnehmerInnen beschreibt.

Verwende z.B. die Gruppen "0 bis 4 TeilnehmerInnen", "5 bis 10 TeilnehmerInnen", "darüber".

```
>>> courses = [
    {'title': 'Holz 1', participants: ["Noah", "Max", "Anna", "Sara", "Emma"]
    {'title': 'Python', participants: ["Laura", "Leon"]},
    {'title': 'R', participants: ["Mia"]},
]
>>> participant_counts(courses)
{"0-4": 2, "5-10": 1, "10-": 0}
```

Dictionary umkehren

AUFGABE

Schreibe eine Funktion, die ein umgekehrtes Dictionary zurückgibt. Wenn der Input ein z.B. Telefonbuch ist (Name -> Nummer), soll der Output ein Mapping von Nummer auf Name sein.

Achtung: Keys können den gleichen Value haben. Deswegen muss der Value-Typ beim Ergebnis eine Liste mit allen entsprechenden Keys des ursprünglichen Dictionaries sein.

```
>>> invert_dictionary({
"Leon": "555-1234",
"Emilia": "555-1234",
"Max": "555-789",
})
{"555-1234": ["Emilia", "Leon"], "555-789": ["Max"]})
```

Kassensystem (Bonus)

⇒ In der Einheit lösen oder als Hausübung?

AUFGABE

Wir haben eine Preistabelle (Dictionary) für die Produkte und eine Liste von jä-Kunden. Diese Kunden erhalten 20% Rabatt.

Als Lockangebote gibt es auch ein paar Produkte, die gratis sind.

Wenn ein Kunde zur Kassa kommt, erhalten wir von unserem biometrischen Erkennungssystem den Namen des Kunden und vom automatischen Scanner eine Liste der Produkte.

Berechne die Summe, die der Kunde bezahlen muss.

Hinweis: Verwende einfach Strings um die Kundennamen und Produkte zu modellieren.