

# Informatik I: Einführung in die Programmierung

## 14. Dictionaries und Mengen

Albert-Ludwigs-Universität Freiburg



Prof. Dr. Peter Thiemann

16.12.2025

# 1 Dictionaries



- Beispiele
- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Dictionaries

Beispiele  
Operationen  
Geschachtelte  
Dictionaries  
Views  
Zugriff auf REST  
APIs  
Dicts als  
Hashtabellen  
Veränderliche  
Dict-Keys?

## Mengen

- Ein **Dictionary** (Wörterbuch), kurz *Dict*, ist eine Abbildung von **Schlüsseln** (*keys*) auf zugehörige **Werte** (*values*).
- Alternative Bezeichnung: *assoziatives Array*
- **Veränderliche (mutable) Datenstruktur**
- Grundoperationen auf Dictionaries
  - Einfügen einer Assoziation (Schlüssel  $\mapsto$  Wert) (evtl. vorhandene Assoziation mit Schlüssel wird überschrieben),
  - Entfernen einer Assoziation (Schlüssel),
  - Nachschlagen des Werts zu einem Schlüssel,
  - Test auf Anwesenheit eines Schlüssels
- Voraussetzungen
  - Schlüssel müssen auf Gleichheit getestet werden können!
  - Schlüssel müssen unveränderlich (immutable) sein!

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST

APIs

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

- Dictionaries sind so implementiert, dass der Wert zu einem gegebenen Schlüssel unabhängig von der Anzahl der bestehenden Einträge effizient bestimmt werden kann.
- Der Typ eines Dictionaries ist `dict[Key, Value]`, wobei `Key` der Typ der Schlüssel ist und `Value` der Typ der Werte.
- (Ein aktuelles Thema: **key-value stores**; das sind netzweit verteilte Dictionaries.)

## Dictionaries

- Beispiele
- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

# Dictionaries: Ein Beispiel

```
>>> description : dict[str, str] = {  
...     "walk": "silly", "parrot": "dead",  
...     "unladen swallow": "no witchcraft"}  
>>> description["parrot"]  
'dead'  
>>> "walk" in description  
True  
>>> description["parrot"] = "pining for the fjords"  
>>> description["slides"] = "unfinished"  
>>> description  
{'walk': 'silly', 'parrot': 'pining for the fjords', 'unladen swallow': 'no witchcraft'}
```

## Dictionaries

### Beispiele

- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

# Dictionaries erzeugen (Auswahl)

- `{key1: value1, key2: value2, ...}`

Hier sind *key1*, *key2*, ... **unveränderliche Python-Objekte**, d.h. Zahlen, Strings, Tupel, etc.

- `dict(key1=value1, key2=value2, ...):`

Hier sind die Schlüssel *key1*, *key2*, ... **Variablennamen**, die vom `dict`-Konstruktor in Strings konvertiert werden.

- `dict(sop)` wobei `sop: Sequence[tuple[Any,Any]]:`  
`dict([(key1, value1), (key2, value2), ...])`  
entspricht `{key1: value1, key2: value2, ...}`.

Die Werte *value1*, *value2* usw. sind beliebige Objekte.

## Dictionaries

### Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

# Dictionaries erzeugen: Beispiele

```
>>> {"parrot": "dead", "spam": "tasty", 10: "zehn"}
{'parrot': 'dead', 'spam': 'tasty', 10: 'zehn'}
>>> dict(six=6, nine=9, six_times_nine=54)
{'six': 6, 'nine': 9, 'six_times_nine': 54}
>>> english = ["red", "blue", "yellow"]
>>> german = ["rot", "blau", "gelb"]
>>> dict(zip(english, german))
{'red': 'rot', 'blue': 'blau', 'yellow': 'gelb'}
```

## Dictionaries

### Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST

APIs

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

Sei  $d : \text{dict}[\text{Key}, \text{Value}]$

- $key$  in  $d$ :  
True, falls das Dictionary  $d$  den Schlüssel  $key$  enthält.
- $\text{bool}(d)$ :  
True, falls das Dictionary nicht leer ist.
- $\text{len}(d)$ :  
Liefert die Zahl der Elemente (Assoziationen) in  $d$ .

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen



- `d[key]`:  
Liefert den Wert zum Schlüssel *key*.  
Fehler bei nicht vorhandenen Schlüsseln.
- `d.get(key, value)`:  
Wie `d[key]`, aber es ist kein Fehler, wenn *key* nicht vorhanden ist.  
Stattdessen wird in diesem Fall das optionale zweite Argument zurückgegeben (`None`, wenn es weggelassen wurde).

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

## get: Beispiel



```
def get_food_amount(food : str):  
    food_amounts = {"spam": 2, "egg": 1, "cheese": 4}  
    return food_amounts.get(food, 0)  
  
for food in ["egg", "vinegar", "cheese"]:  
    amount = get_food_amount(food)  
    print("We have enough", food, "for", amount, "people.")
```

liefert die Ausgabe:

We have enough egg for 1 people. We have enough vinegar for 0 people. We have enough cheese for 4 people.

Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

Mengen

## ■ `d[key] = value`

Weist dem Schlüssel *key* einen Wert zu. Befindet sich bereits eine Assoziation mit Schlüssel *key* in *d*, wird sie ersetzt.

## ■ `d.setdefault(key, default= None)`

Vom Rückgabewert äquivalent zu `d.get(key, default)`.

Falls *d* den Schlüssel noch nicht enthält, wird `d[key] = default` ausgeführt.

- Auch Dictionaries können selbst Dictionaries enthalten.

```
>>> en_de={'red': 'rot', 'yellow': 'gelb', 'blue': 'blau'}
>>> de_fr={'rot': 'rouge', 'gelb': 'jaune', 'blau': 'bleu'}
>>> dicts = {'en->de': en_de, 'de->fr': de_fr}
>>> dicts['de->fr']['blau']
'bleu'
>>> dicts['de->fr'][dicts['en->de']['blue']]
'bleu'
```

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

Die folgenden Methoden liefern Objekte, die mit **for**-Schleifen durchlaufen werden können.

Der Durchlauf geschieht in der Reihenfolge des Einfügens in das Dictionary.

Achtung: Dabei werden Änderungen am zugrundeliegenden `dict` sichtbar!

- `d.keys()`  
Liefert alle Schlüssel in `d`.
- `d.values()`  
Liefert alle Werte in `d`.
- `d.items()`  
Liefert alle Einträge, d.h. (`key`, `value`)-Assoziationen in `d`.
- Dictionaries können auch direkt in `for`-Schleifen verwendet werden. Dabei wird die Methode `keys` benutzt. `for`-Schleifen über Dictionaries durchlaufen also die *Schlüssel*.

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

- Eine Funktion kann Keyword Parameter der Form `par=wert` akzeptieren.
- Falls der **letzte formale Parameter** der Funktion die Form `**kwargs` hat, so akzeptiert die Funktion beliebige Keyword Parameter.
- Im Funktionsrumpf kann `kwargs` wie ein Dictionary verwendet werden.

```
>>> def echo (**kwargs):  
...     for k, v in kwargs.items():  
...         print(k + " = " + str(v))  
...  
>>> echo (a=42, b='foo')  
a = 42  
b = foo
```

## Dictionaries

Beispiele  
Operationen  
Geschachtelte  
Dictionaries

### Views

Zugriff auf REST  
APIs  
Dicts als  
Hashtabellen  
Veränderliche  
Dict-Keys?

## Mengen

- Im Internet werden Webservices über sogenannte **REST APIs** aufgerufen.
- REST = Representational State Transfer
- Eine solche API wird durch eine Reihe von **URIs** beschrieben.
- Eine URI besteht aus
  - einem Hostnamen, der den anbietenden Rechner benennt,
  - einem Pfad, der die Funktion auswählt und ihre Argumente angibt.
- Beispiel: Zugriff auf Daten der Bitcoin Blockchain  
Dokumentation:  
[https://www.blockchain.com/de/explorer/api/blockchain\\_api](https://www.blockchain.com/de/explorer/api/blockchain_api)

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

Der Rechner `blockchain.info` bietet Informationen über die Bitcoin-Blockchain an.

- Unter `https://blockchain.info/latestblock` ist der letzte erstellte Block verfügbar.
- Das Ergebnis ist ein Objekt, das im **JSON** Format übermittelt wird:

```
{  
  "hash": "0000000000000538200a48202ca6340e983646ca088c7618ae82d68e0c76ef5a",  
  "time": 1325794737,  
  "block_index": 841841,  
  "height": 160778,  
  "txIndexes": [  
    13950369,  
    13950510,  
    13951472  
  ]  
}
```

Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen



- In Python erfolgt der Zugriff auf eine REST API mit Hilfe des Moduls `requests`.
- Die Operation `requests.get()` nimmt als Argument einen String mit einer URI und liefert ein `Response` Objekt.
- Dieses besitzt eine Methode `.json()`, die eine Antwort im JSON Format in ein Dictionary umwandelt.

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

# Zugriff auf REST APIs — Verarbeitung

## Matching auf Dictionaries



```
import requests
def size_of_latest_block(base: str = 'https://blockchain.info/') -> int:
    lb = requests.get(base + 'latestblock')
    match lb.json():
        case {'hash': hash_lb, 'time': time_lb }:
            print("time=", time_lb)
            sb = requests.get(base + 'rawblock/' + hash_lb)
            return len (sb.json()['tx'])
```

Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

Mengen

- Pattern matching auf ein Dictionary wie `lb.json()` listet die Schlüssel auf, die vorhanden sein müssen (im Beispiel `'hash'` und `'time'`) und assoziiert diese jeweils mit einem Pattern (in diesem Fall jeweils nur eine Variable).
- Das Dictionary darf weitere Schlüssel enthalten, die ignoriert werden.

# Wie funktionieren Dictionaries?

Dictionaries sind als **Hashtabellen** implementiert:

- Bei der Erzeugung eines Dictionaries wird eine große Tabelle (die **Hashtabelle**) eingerichtet.
- Eine **Hashfunktion** ordnet jedem Schlüssel einen **Hashwert** zu, der als Tabellenindex dient. (Problem: Mehr Schlüssel als Plätze in der Tabelle.)
- Der zum Schlüssel gehörige Wert wird an dieser Stelle in der Tabelle abgelegt, es sei denn...
- an diesem Index ist bereits ein Eintrag für einen anderen Schlüssel vorhanden: eine Hashfunktion kann unterschiedlichen Schlüsseln den gleichen Hashwert zuordnen (**Kollision**).
- Bei gleichen Hashwerten für verschiedene Schlüssel gibt es eine Spezialbehandlung (z.B. Ablegen des Werts in der nächsten freien Zelle).
- Der Zugriff erfolgt trotzdem in (erwarteter) **konstanter Zeit**.

Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

Mengen

# Eine Hashtabelle bei der Arbeit

Eingabe: ('parrot', 'dead')  
hash('parrot')=4  
Ausgabe: 'dead'

Hashtabelle		
Index	Key	Value
0	'spam'	'tasty'
1		
2		
3		
4	'parrot'	'dead'
5	'zehn'	10
6		

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

- Schlüssel müssen hash-bar sein und auf Gleichheit getestet werden können.
- Objekte, die als Schlüssel in einem Dictionary verwendet werden sollen, dürfen **nicht verändert** werden. Sonst ändert sich der Hashwert und das Objekt wird nicht mehr gefunden.

## Dictionaries

Beispiele

Operationen

Geschachtelte

Dictionaries

Views

Zugriff auf REST  
APIs

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

# Veränderliche Dictionary-Keys (1)

potential\_trouble.py

```
mydict = {}  
mylist = [10, 20, 30]  
mydict[mylist] = "spam"  
del mylist[1]  
print(mydict.get([10, 20, 30]))  
print(mydict.get([10, 30]))
```

```
# Was kann passieren?  
# Was sollte passieren?
```

**Illegal!**

mydict[mylist] liefert schon eine Fehlermeldung!

## Dictionaries

- Beispiele
- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

## Veränderliche Dictionary-Keys (2)

- In Python dürfen nur *unveränderliche* Objekte, die aus Tupeln, Strings und Zahlen konstruiert sind, als Dictionary-Schlüssel verwendet werden.
- Verboten sind also Listen und Dictionaries sowie Objekte, die Listen oder Dictionaries beinhalten bzw. deren Attribute veränderlich sind.
- Selbstdefinierte Klassen, deren Instanzen als Dictionary-Schlüssel verwendet werden sollen, müssen als `frozen` definiert werden, sodass die Attribute nach der Initialisierung nicht verändert werden können:

```
@dataclass(frozen=True)
```

- Für die *Werte* im Dictionary sind beliebige Objekte zulässig; die Einschränkung gilt nur für Schlüssel!

### Dictionaries

- Beispiele
- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

### Mengen

## Python-Interpreter

```
>>> mydict = {"silly", "walk"): [1, 2, 3]}
>>> mydict[[10, 20]] = "spam"
Traceback (most recent call last): ...
TypeError: unhashable type: 'list'
>>> mydict[("silly", [], "walk")] = 1
Traceback (most recent call last): ...
TypeError: unhashable type: 'list'
```

### Dictionaries

- Beispiele
- Operationen
- Geschachtelte Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

### Mengen



## Veränderliche Dictionary-Keys (4)

```
@dataclass(frozen=True)
class Time():
    hours: int
    minutes: int

morning = Time(6, 30)
noon     = Time(12, 00)

d[morning] = "breakfast"
d[noon]     = "lunch"
```

### Dictionaries

- Beispiele
- Operationen
- Geschachtelte
- Dictionaries
- Views
- Zugriff auf REST APIs
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

### Mengen

- Set und Frozenset
- Operationen
- Konstruktion
- Grundlegende Operationen
- Einfügen und Entfernen
- Zusammenfassung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- Der Datentyp Menge ist ein **Container-Datentyp**. Das heißt, ein Element des Datentyps Menge enthält selbst (endlich viele, untergeordnete) Elemente. Die Reihenfolge der Elemente spielt keine Rolle.
- Grundoperationen auf dem Datentyp Menge:
  - Einfügen eines Elements,
  - Entfernen eines Elements,
  - Test ob Element enthalten ist.
- Voraussetzungen
  - Elemente müssen hashbar sein!
  - Elemente müssen auf Gleichheit getestet werden können!
  - Elemente müssen unveränderlich (immutable) sein!
- Einfügen und Entfernen sind **idempotent**; eine Menge kann also nicht dasselbe Element ‚mehrmals‘ enthalten ( $\Rightarrow$  Multimenge).

- Mengen können durch Listen implementiert werden. Dann ist die mittlere Zeit ein Element zu finden, *linear* in der Größe der Menge.
- Mengen können durch (balancierte) binäre Suchbäume implementiert werden. Dann ist die mittlere Zeit ein Element zu finden *logarithmisch* in der Größe der Menge und wir brauchen eine Ordnung auf den Elementen.
- Mengen können durch Dicts implementiert werden, wobei die Elemente die Schlüssel sind und der Wert immer `None` ist (konstante Zugriffszeit).
- Es gibt spezielle Datentypen für veränderliche und unveränderliche Mengen in Python, die alle Mengenoperationen unterstützen.
- Sie sind ebenfalls mit Hilfe von Hashtabellen realisiert.

- Voraussetzung: **Mengenelemente** müssen *hashbar* (unveränderlich) sein (wie die Schlüssel bei Dictionaries).
- Es gibt die Typen `set[Elem]` und `frozenset[Elem]` für Mengen mit Elementen vom Typ `Elem`.
  - Instanzen von `frozenset` sind unveränderlich  $\rightsquigarrow$  `hashbar`,
  - Insbesondere können Instanzen von `frozenset` auch als Elemente von `set` und `frozenset` sowie als Schlüssel von Dictionaries verwendet werden.
  - Instanzen von `set` sind veränderlich.

Wir teilen die Operationen auf Mengen in Gruppen ein:

- Konstruktion
- Grundlegende Operationen
- Einfügen und Entfernen von Elementen
- Mengenvergleiche
- Klassische Mengenoperationen

Dictionaries

Mengen

Set und Frozenset

**Operationen**

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- `{elem1, ..., elemN}`  
Erzeugt die veränderliche Menge `{elem1,...,elemN}`.
- `set()`  
Veränderliche leere Menge.
- `set(iterable)`  
Veränderliche Menge aus den Elementen von `iterable` (ein Tupel, eine Liste, o.ä.).
- `frozenset()`  
Unveränderliche leere Menge.
- `frozenset(iterable)`  
Unveränderliche Menge aus den Elementen von `iterable` (Tupel, Liste, ...).
- Die Elemente von `iterable` müssen *hashbare* Objekte sein!

Dictionaries

Mengen

Set und Frozenset

Operationen

**Konstruktion**

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

# Konstruktion von Mengen: Beispiele (1)



```
>>> s = set("spamspam")
>>> s.remove('p')
>>> print(s)
{'m', 's', 'a'}
>>> f = frozenset("spamspam")
>>> f.difference("sm")
frozenset({'p', 'a'})
>>> print(f)
frozenset({'p', 'm', 's', 'a'})
>>> set(["spam", 1, [2, 3]])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> set(("spam", 1, (2, 3)))
{(2, 3), 1, 'spam'}
>>> set({"spam": 20, "jam": 30})
{'spam', 'jam'}
```

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung



# Konstruktion von Mengen: Beispiele (2)

```
>>> s = set(["jam", "spam"])
>>> set([1, 2, 3, s])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>> set([1, 2, 3, frozenset(s)])
{frozenset({'spam', 'jam'}), 1, 2, 3}
```

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- `element in s`, `element not in s`  
Test auf Mitgliedschaft bzw. Nicht-Mitgliedschaft  
(liefert `True` oder `False`).
- `bool(s)`  
`True`, falls die Menge `s` nicht leer ist.
- `len(s)`  
Liefert die Zahl der Elemente der Menge `s`.
- `for element in s:`  
Iteration über Mengen.
- `s.copy()`  
Liefert eine (flache) Kopie der Menge `s`.

## ■ `s.add(element)`

Fügt das Objekt `element` zur Menge `s` hinzu, falls es noch nicht Element der Menge ist.

## ■ `s.remove(element)`

Entfernt `element` aus der Menge `s`, falls es dort enthalten ist.  
Sonst: `KeyError`.

## ■ `s.discard(element)`

Wie `remove`, aber kein Fehler, wenn `element` nicht in der Menge enthalten ist.

## ■ `s.pop()`

Entfernt ein willkürliches Element aus `s` und liefert es zurück.

## ■ `s.clear()`

Entfernt alle Elemente aus der Menge `s`.

# Viele weitere Operationen



- `union`, `intersection`, `difference`, `symmetric_difference`
- `<=`, `<` (Test auf Teilmenge)
- `==`, `!=` (Test auf Mengengleichheit)

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- dicts sind Abbildungen von Schlüsseln auf Werte.
- Der Zugriff auf Elemente von dicts erfolgt (fast) in konstanter Zeit
- dicts sind veränderlich.
- Die Typen `set` und `frozenset` implementieren Mengen mit allen erwarteten Operationen.
- Die Instanzen von `set` sind veränderliche Strukturen, die Instanzen von `frozenset` sind nicht veränderlich.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung