

# Informatik I: Einführung in die Programmierung

## 13. Dictionaries & Mengen

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

Bernhard Nebel

22. November 2016

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Dictionaries

Beispiele  
Operationen  
Flaches und tiefes  
Kopieren  
Geschachtelte  
Dicts  
Update  
Werte entfernen  
Views  
Dicts als  
Hashtabellen  
Veränderliche  
Dict-Keys?

## Mengen

Zusammen-  
fassung

- **Dictionaries** (Wörterbücher) oder kurz *Dicts* sind assoziative Arrays/Listen.
- Dictionaries speichern Paare von **Schlüsseln** (*keys*) und zugehörigen **Werten** (*values*) und sind so implementiert, dass man sehr **effizient** den Wert zu einem gegebenen Schlüssel bestimmen kann.
- Im Gegensatz zu Sequenzen sind Dictionaries **ungeordnete** Container; es ist nicht sinnvoll, von einem ersten (zweiten, usw.) Element zu sprechen.

## Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

Zusammenfassung

## Python-Interpreter

```
>>> description = {"parrot": "dead", "spam": "tasty",  
...                (1, 2, 3): "no witchcraft"}  
>>> description["parrot"]  
'dead'  
>>> "spam" in description  
True  
>>> description["parrot"] = "pining for the fjords"  
>>> description["slides"] = "unfinished"  
>>> description  
{'slides': 'unfinished', (1, 2, 3): 'no witchcraft',  
 'parrot': 'pining for the fjords', 'spam': 'tasty'}
```

### Dictionaries

#### Beispiele

Operationen

Flaches und tiefes  
Kopieren

Geschachtelte  
Dicts

Update

Werte entfernen

Views

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

### Mengen

Zusammen-  
fassung



Dictionaries können auf verschiedene Weisen **erzeugt** werden:

- `{key1: value1, key2: value2, ...}`:  
Hier sind `key1`, `value1` usw. normale Python-Objekte, z.B. Strings, Zahlen oder Tupel.
- `dict(key1=value1, key2=value2, ...)`:  
Hier sind die Schlüssel `key1` usw. **Variablennamen**, die vom `dict`-Konstruktor in Strings konvertiert werden.  
Die Werte `value1` usw. sind normale Objekte.
- `dict(sequence_of_pairs)`:  
`dict([(key1, value1), (key2, value2), ...])`  
entspricht `{key1: value1, key2: value2, ...}`.
- `dict.fromkeys(seq, value)`:  
Ist `seq` eine Sequenz mit Elementen `key1`, `key2`, ..., erhalten wir `{key1: value, key2: value, ...}`.  
Wird `value` (und das Komma) weggelassen, wird `None` verwendet.

## Dictionaries

### Beispiele

Operationen

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

## Zusammenfassung

- Manche Funktionen auf Dicts (oder auch anderen Python-Typen und -Objekten) werden mit Hilfe der Punkt-Notation angegeben: `dict.fromkeys(seq, value)`.
- Bei einem Typen (wie `dict`) wird dann ein entsprechendes Objekt erzeugt.
- Handelt es sich um ein Objekt, wird eine Operation auf dem Objekt durchgeführt
- Man nennt diese mit einem Punkt angehängten Funktionen **Methoden**.

→ **Objekt-basierte** Notation (führt zu OOP)

## Dictionaries

### Beispiele

Operationen

Flaches und tiefes  
Kopieren

Geschachtelte  
Dicts

Update

Werte entfernen

Views

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

## Zusammenfassung

## Python-Interpreter

```
>>> {"parrot": "dead", "spam": "tasty", 10: "zehn"}
{10: 'zehn', 'parrot': 'dead', 'spam': 'tasty'}
>>> dict(six=6, nine=9, six_times_nine=42)
{'six_times_nine': 42, 'nine': 9, 'six': 6}
>>> english = ["red", "blue", "green"]
>>> german = ["rot", "blau", "grün"]
>>> dict(zip(english, german))
{'red': 'rot', 'green': 'grün', 'blue': 'blau'}
>>> dict.fromkeys("abc")
{'a': None, 'c': None, 'b': None}
>>> dict.fromkeys(range(3), "eine Zahl")
{0: 'eine Zahl', 1: 'eine Zahl', 2: 'eine Zahl'}
```

### Dictionaries

#### Beispiele

Operationen

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

### Zusammenfassung



Sei `d` ein Dict:

- `key in d`:  
True, falls das Dictionary `d` den Schlüssel `key` enthält.
- `bool(d)` (bzw. einfach `d`):  
True, falls das Dictionary nicht leer ist.
- `len(d)`:  
Liefert die Zahl der Elemente (Paare) in `d`.
- `d.copy()`:  
Liefert eine (flache) Kopie von `d` (tiefe Kopie kommt gleich)

## Dictionaries

Beispiele

### Operationen

Flaches und tiefes  
Kopieren

Geschachtelte  
Dicts

Update

Werte entfernen

Views

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

## Zusammenfassung





- `d[key]`:  
Liefert den Wert zum Schlüssel `key`.  
Fehler bei nicht vorhandenen Schlüsseln.
- `d.get(key, default)` (oder `d.get(key)`):  
Wie `d[key]`, aber es ist kein Fehler, wenn `key` nicht vorhanden ist. Stattdessen wird in diesem Fall `default` zurückgeliefert (`None`, wenn kein Default angegeben wurde).

## Dictionaries

Beispiele

### Operationen

Flaches und tiefes  
Kopieren

Geschachtelte  
Dicts

Update

Werte entfernen

Views

Dicts als  
Hashtabellen

Veränderliche  
Dict-Keys?

## Mengen

## Zusammenfassung

## food\_inventory.py

```
def get_food_amount(food):
    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.")

# 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

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

### Zusammenfassung

- `d[key] = value:`  
Weist dem Schlüssel `key` einen Wert zu. Befindet sich bereits ein Paar mit Schlüssel `key` in `d`, wird es ersetzt.
- `d.setdefault(key, default)` (oder `d.setdefault(key)`):  
Vom Rückgabewert äquivalent zu `d.get(key, default)`.  
Falls das Dictionary den Schlüssel noch nicht enthält, wird zusätzlich `d[key] = default` ausgeführt.
- Hier kann man oft besser `defaultdict` aus dem Modul `collections` als Spezialisierung von `dict` einsetzen!
- `collections.defaultdict(Defaultgenerator)` liefert ein `dict`, bei dem immer die Funktion *Defaultgenerator* aufgerufen wird, wenn kein Wert für den `key` vorhanden ist.

## Dictionaries

Beispiele

Operationen

Flaches und tiefes Kopieren

Geschachtelte Dicts

Update

Werte entfernen

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

## Mengen

## Zusammenfassung

## hobbies.py

```
from collections import defaultdict
hobby_dict = defaultdict(list)
def add_hobby(person, hobby):
    hobby_dict[person].append(hobby)

add_hobby("Justus", "Reading")
add_hobby("Peter", "Cycling")
add_hobby("Bob", "Music")
add_hobby("Justus", "Riddles")
add_hobby("Bob", "Girls")
print(hobby_dict)
# Ausgabe: defaultdict(<class 'list'>,
#           {'Bob': ['Music', 'Girls'],
#            'Peter': ['Cycling'],
#            'Justus': ['Reading', 'Riddles']})
```

### Dictionaries

Beispiele

Operationen

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

### Zusammenfassung

# Exkurs: Flaches und tiefes Kopieren (1)



- Wie schon bei Listen, erzeugt eine Zuweisung **keine Kopie!**

## Python-Interpreter

```
>>> en_de={'red': 'rot', 'green': 'grün', 'blue':  
'blau'}  
>>> en_sw = en_de  
>>> en_sw['green'] = 'grää'  
>>> en_de['green']  
'grää'  
>>> en_de={'red': 'rot', 'green': 'grün', 'blue':  
'blau'}  
>>> en_sw = en_de.copy()  
>>> en_sw['green'] = 'grää'  
>>> en_de['green']  
'grün'
```

## Visualisierung

### Dictionaries

Beispiele

Operationen

**Flaches und tiefes  
Kopieren**

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

Zusammen-  
fassung



- Rekursiv enthaltene Strukturen werden beim **flachen Kopieren** nicht kopiert!

## Python-Interpreter

```
>>> snums={'even': [2, 4, 6], 'odd': [1, 3, 5]}
>>> sprimes = snums.copy()
>>> del(sprimes['even'][1:]); del(sprimes['odd'][0])
>>> snums
{'even': [2], 'odd': [3, 5]}
>>> import copy
>>> snums={'even': [2, 4, 6], 'odd': [1, 3, 5]}
>>> sprimes = copy.deepcopy(snums)
>>> del(sprimes['even'][1:]); del(sprimes['odd'][0])
>>> snums
{'even': [2, 4, 6], 'odd': [1, 3, 5]}
```

- Funktioniert bei Listen ebenso!

### Dictionaries

Beispiele

Operationen

**Flaches und tiefes  
Kopieren**

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

Zusammen-  
fassung

- Ebenso wie Listen kann man auch Dicts **rekursiv einbetten**.

## Python-Interpreter

```
>>> en_de={'red': 'rot', 'green': 'grün', 'blue':  
'blau'}  
>>> de_fr = {'rot': 'rouge', 'grün': 'vert', '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

Flaches und tiefes

Kopieren

### Geschachtelte Dicts

Update

Werte entfernen

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

### Mengen

### Zusammen- fassung

# Operationen auf Dictionaries: Werte eintragen mit update



- `d.update(another_dict):`  
Führt `d[key] = value` für alle `(key, value)`-Paare in `another_dict` aus.  
Überträgt also alle Einträge aus `another_dict` nach `d` und überschreibt bestehende Einträge mit dem gleichen Schlüssel.
- `d.update(sequence_of_pairs):`  
Entspricht `d.update(dict(sequence_of_pairs))`.
- `d.update(key1=value1, key2=value2, ...):`  
Entspricht `d.update(dict(key1=value1, key2=value2, ...))`.

## Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update**
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

## Zusammenfassung



# Operationen auf Dictionaries: Einträge entfernen



- `del d[key]`:  
Entfernt das Paar mit dem Schlüssel `key` aus `d`. Fehler, falls kein solches Paar existiert.
- `d.pop(key, default)` (oder `d.pop(key)`):  
Entfernt das Paar mit dem Schlüssel `key` aus `d` und liefert den zugehörigen Wert. Existiert kein solches Paar, wird `default` zurückgeliefert, falls angegeben (sonst Fehler).
- `d.popitem()`:  
Entfernt ein (willkürliches) Paar (`key`, `value`) aus `d` und liefert es zurück. Fehler, falls `d` leer ist.
- `d.clear()`:  
Entfernt alle Elemente aus `d`.
  - Was ist der Unterschied zwischen `d.clear()` und `d = {}`?

## Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen**
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

Zusammenfassung



Die folgenden Methoden liefern iterierbare **views** zurück, die Änderungen an dem zugrundeliegenden `dict` reflektieren!

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

## Dictionaries

Beispiele

Operationen

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

### Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

## Mengen

## Zusammenfassung

Dictionaries sind als **Hashtabellen** implementiert:

- Es wird initial eine große Liste/Tabelle (die **Hashtabelle**) eingerichtet.
- Jeder Schlüssel wird mit Hilfe einer **Hashfunktion** in einen Index (dem **Hashwert**) übersetzt.
- Bei gleichen Hashwerten für verschiedene Schlüssel gibt es eine Spezialbehandlung (z.B. nächste freie Zelle).
- Der Zugriff erfolgt damit in (erwarteter) **konstanter Zeit**.

## Dictionaries

Beispiele

Operationen

Flaches und tiefes

Kopieren

Geschachtelte

Dicts

Update

Werte entfernen

Views

Dicts als

**Hashtabellen**

Veränderliche

Dict-Keys?

## Mengen

## Zusammenfassung

# 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
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views

## Dicts als Hashtabellen

- Veränderliche Dict-Keys?

## Mengen

## Zusammenfassung

- Hashtabellen haben **keine spezielle Ordnung** für die Elemente.
- Daher liefert `keys` die Schlüssel nicht in der Einfügereihenfolge, sondern in einer beliebigen Abfolge.
- Objekte, die als Schlüssel in einem Dictionary verwendet werden, dürfen **nicht verändert** werden. Ansonsten könnte es zu Problemen kommen.

## Dictionaries

Beispiele

Operationen

Flaches und tiefes  
Kopieren

Geschachtelte  
Dicts

Update

Werte entfernen

Views

**Dicts als  
Hashtabellen**

Veränderliche  
Dict-Keys?

## Mengen

## Zusammen- fassung

## 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?
```

### Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?**

### Mengen

Zusammenfassung

- Um solche Problem zu vermeiden, sind in Python nur *unveränderliche* Objekte wie Tupel, Strings und Zahlen als Dictionary-Schlüssel erlaubt.
  - Genauer: Selbst Tupel sind verboten, wenn sie direkt oder indirekt veränderliche Objekte beinhalten.
- Verboten sind also Listen und Dictionaries oder Objekte, die Listen oder Dictionaries beinhalten.
- Für die *Werte* sind beliebige Objekte zulässig; die Einschränkung gilt nur für Schlüssel!

## Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

## Mengen

## Zusammenfassung

## Python-Interpreter

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

### Dictionaries

- Beispiele
- Operationen
- Flaches und tiefes Kopieren
- Geschachtelte Dicts
- Update
- Werte entfernen
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?**

### Mengen

Zusammenfassung



- Set und Frozenset
- Operationen
- Konstruktion
- Grundlegende Operationen
- Einfügen und Entfernen
- Methoden vs. Operationen
- Vergleiche
- Klassische Mengenoperationen

Dictionaries

### Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende  
Operationen

Einfügen und  
Entfernen

Methoden vs.  
Operationen

Vergleiche

Klassische Men-  
genoperationen

Zusammen-  
fassung

- **Mengen** sind Zusammenfassungen von Elementen (in unserem Fall immer endlich),
- Mengenelemente sind einzigartig; eine Menge kann also nicht dasselbe Element ‚mehrmals‘ beinhalten.
- Man könnte Mengen durch Listen implementieren (müsste dann immer die Liste durchsuchen)
- Man könnte Mengen durch Dicts implementieren, wobei die Elemente durch Schlüssel realisiert würden und der Wert immer `None` ist (konstante Zugriffszeit).
- Es gibt allerdings eigene Datentypen für Mengen in Python (auch mit Hilfe von Hashtabellen realisiert), die alle **Mengenoperation** unterstützen.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-

genoperationen

Zusammen-  
fassung

- Mengenelemente müssen *hashbar* sein (wie bei Dictionaries).
- set vs. frozenset:
  - frozensets sind unveränderlich  $\rightsquigarrow$  hashbar,
  - sets sind veränderlich
  - Insbesondere können frozensets also auch als Elemente von sets und frozensets verwendet werden.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-

genoperationen

Zusammen-  
fassung

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

Methoden vs.

Operationen

Vergleiche

Klassische Mengenoperationen

Zusammenfassung



- `{elem1, ..., elemN}`: Erzeugt die veränderliche Menge `{elem1, ..., elemN}`.
- `set()`: Erzeugt eine veränderliche leere Menge.
- `set(iterable)`: Erzeugt eine veränderliche Menge aus Elementen von `iterable`.
- `frozenset()`: Erzeugt eine unveränderliche leere Menge.
- `frozenset(iterable)`: Erzeugt eine unveränderliche Menge aus Elementen von `iterable`.
- `set` und `frozenset` können aus beliebigen iterierbaren Objekten `iterable` erstellt werden, also solchen, die `for` unterstützen (z.B. `str`, `list`, `dict`, `set`, `frozenset`.)
- Jedoch dürfen innerhalb von `iterable` nur *hashbare* Objekte (z.B. keine Listen!) enthalten sein (sonst `TypeError`).

Dictionaries

Mengen

Set und Frozenset

Operationen

**Konstruktion**

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-

genoperationen

Zusammen-  
fassung

## Python-Interpreter

```
>>> set("spamspam")
{'a', 'p', 's', 'm'}
>>> frozenset("spamspam")
frozenset({'a', 'p', 's', 'm'})
>>> set(["spam", 1, [2, 3]])
Traceback (most recent call last): ...
TypeError: unhashable type: 'list'
>>> set(("spam", 1, (2, 3)))
{1, (2, 3), 'spam'}
>>> set({"spam": 20, "jam": 30})
{'jam', 'spam'}
```

Dictionaries

Mengen

Set und Frozenset

Operationen

**Konstruktion**

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-  
genoperationen

Zusammen-  
fassung

## Python-Interpreter

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

Dictionaries

Mengen

Set und Frozenset

Operationen

**Konstruktion**

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-

genoperationen

Zusammen-  
fassung

- `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`:  
Über Mengen kann natürlich iteriert werden.
- `s.copy()`:  
Liefert eine (flache) Kopie der Menge `s`.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

**Grundlegende  
Operationen**

Einfügen und  
Entfernen

Methoden vs.  
Operationen

Vergleiche

Klassische Men-  
genoperationen

Zusammen-  
fassung





- `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`.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

**Einfügen und Entfernen**

Methoden vs. Operationen

Vergleiche

Klassische Mengenoperationen

Zusammenfassung

Viele Operationen auf Mengen sind sowohl als benannte Methoden als auch über Operatoren verfügbar. Beispiel:

- Operator: `s & t`.
- Benannte Methode: `s.intersection(t)`
- Zuweisungsoperator: `s &= t`.
- Benannte Modifikationsmethode:  
`s.intersection_update(t)`
- Im Falle der Methoden wird das *Argument* in eine Menge konvertiert, wenn das *Argument* *iterierbar* ist.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Men-

genoperationen

Zusammen-  
fassung



- `s.issubset(t)`,  $s \leq t$ :  
Testet, ob alle Elemente von  $s$  in  $t$  enthalten sind ( $s \subseteq t$ )
- `s < t`:  
Wie  $s \leq t$ , aber echter Teilmengentest ( $s \subset t$ ).
- `s.issuperset(t)`,  $s \geq t$ ,  $s > t$ :  
Analog für Obermengentests bzw. echte Obermengentests.
- `s == t`:  
Gleichheitstest. Wie  $s \leq t$  and  $t \leq s$ , aber effizienter.
  - Anders als bei den anderen Operatoren ist es *kein* Typfehler, wenn nur eines der Argumente eine Menge ist.
  - In diesem Fall ist `s == t` immer `False`.
  - Ein `set` kann ein `frozenset` sein.
- `s != t`:  
Äquivalent zu `not (s == t)`.

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

**Vergleiche**

Klassische Mengenoperationen

Zusammenfassung

- `s.union(t)`, `s | t`  
`s.intersection(t)`, `s & t`  
`s.difference(t)`, `s - t`  
`s.symmetric_difference(t)`, `s ^ t`

Liefert Vereinigung ( $s \cup t$ ), Schnitt ( $s \cap t$ ), Mengendifferenz ( $s \setminus t$ ) bzw. symmetrische Mengendifferenz ( $s \Delta t$ ) von  $s$  und  $t$ .

Das Resultat hat denselben Typ wie  $s$ .

- `s.update(t)`, `s |= t`  
`s.intersection_update(t)`, `s &= t`  
`s.difference_update(t)`, `s -= t`  
`s.symmetric_difference_update(t)`, `s ^= t`

In-Situ-Varianten der Mengenoperationen.

(Ändern also  $s$ , statt eine neue Menge zu liefern.)

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Mengenoperationen

Zusammenfassung



## Python-Interpreter

```
>>> s1 = frozenset([1, 2, 3])
>>> s2 = set([3, 4, 5])
>>> s1 | s2
frozenset({1, 2, 3, 4, 5})
>>> s2 | s1
{1, 2, 3, 4, 5}
>>> s1 | [3, 4, 5]
Traceback (most recent call last): ...
TypeError: unsupported operand type(s) for |:
'frozenset' and 'list'
>>> s1.union([3, 4, 5])
frozenset({1, 2, 3, 4, 5})
```

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

Klassische Mengenoperationen

Zusammenfassung



## Python-Interpreter

```
>>> s1 = set("dead")
>>> s2 = set("parrot")
>>> s2.update({'a','b','c','d'})
>>> s2
{'t', 'd', 'p', 'r', 'a', 'b', 'c', 'o'}
>>> s1 - s2
{'e'}
>>> s1.symmetric_difference(s2)
{'t', 'p', 'e', 'r', 'b', 'c', 'o'}
>>> (s1 - s2) | (s2 - s1)
{'o', 't', 'b', 'p', 'c', 'e', 'r' }
>>> (s1-s2)|(s2-s1) == s1.symmetric_difference(s2)
True
```

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Methoden vs.

Operationen

Vergleiche

**Klassische Men-  
genoperationen**

Zusammen-  
fassung

# 3 Zusammenfassung



**UNI  
FREIBURG**

Dictionaries

Mengen

Zusammen-  
fassung



- `dicts` können wir als Verallgemeinerung von Listen verstehen, bei denen der Index ein beliebiger (nicht änderbarer) Wert ist.
- Der Zugriff auf Elemente von `dicts` ist fast so effizient wie der Zugriff auf indizierte Listenelemente.
- `dicts` sind änderbare Elemente. Die Kopie eines `dicts` ist erst einmal nur eine Kopie der oberen Struktur!
- Um eine Kopie aller Substrukturen zu erreichen, muss das Modul `deepcopy` benutzt werden (funktioniert genauso bei Listen).
- Mengen in Python (`set`) kann man als `dicts` verstehen, bei denen alle Werte `None` sind.
- Es existieren alle Mengenoperationen.
- Mengen sind veränderliche Strukturen, eingefrorene Mengen (`frozenset`) dagegen nicht.