

Informatik I: Einführung in die Programmierung

13. Objekt-orientierte Programmierung: Aggregation, Invarianten, Datenkapselung, Properties, Operator-Überladung Dictionaries und Mengen

Albert-Ludwigs-Universität Freiburg



Prof. Dr. Peter Thiemann

12.01.2021

1 Aggregation



Aggregation

Properties

Operator-
Überladung

Dictionaries

Mengen

- Oft sind Objekte aus anderen Objekten **zusammengesetzt**.
- Methodenaufrufe auf ein zusammengesetztes Objekt führen meist zu Methodenaufrufen auf eingebetteten Objekten.
- Beispiel: ein zusammengesetztes 2D-Objekt, das andere 2D-Objekte enthält, z.B. einen Kreis und ein Rechteck.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Die Klasse CompositeObject (1)

- Jede Instanz ist ein **2D-Objekt**, aber eine Position macht keinen Sinn.
- Zusätzlich hat jede Instanz als Attribut eine **Liste** von 2D-Objekten.

`newgeoclasses.py` (1)

```
@dataclass
class CompositeObject(TwoDObject):
    contents : list[TwoDObject] = []

    def add(self, obj : TwoDObject):
        self.contents.append(obj)

    def rem(self, obj : TwoDObject):
        self.contents.remove(obj)

    ...
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Die CompositeObject-Klasse (2)



- Die Methoden `size_change`, `move` und `position` werden überschrieben.
- Wir wälzen das Ändern und Verschieben des zusammengesetzten Objektes auf die Einzelobjekte ab: **Delegieren**.

`newgeoclasses.py` (2)

```
def size_change(self, percent : float):
    for obj in self.contents:
        obj.size_change(percent)

def move(self, xchange :float, ychange : float):
    for obj in self.contents:
        obj.move(xchange, ychange)

def position(self):
    return self.contents[0].position() if self.contents else None
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject(contents=[r,c])
>>> a.size_change(200)
>>> r.area()
400.0
>>> a.move(40,40)
>>> a.position()
(40, 40)
>>> c.position()
(41, 42)
>>> b = CompositeObject()
>>> a.add(b)
>>> a.move(-10, -10)
>>> b.position()
```

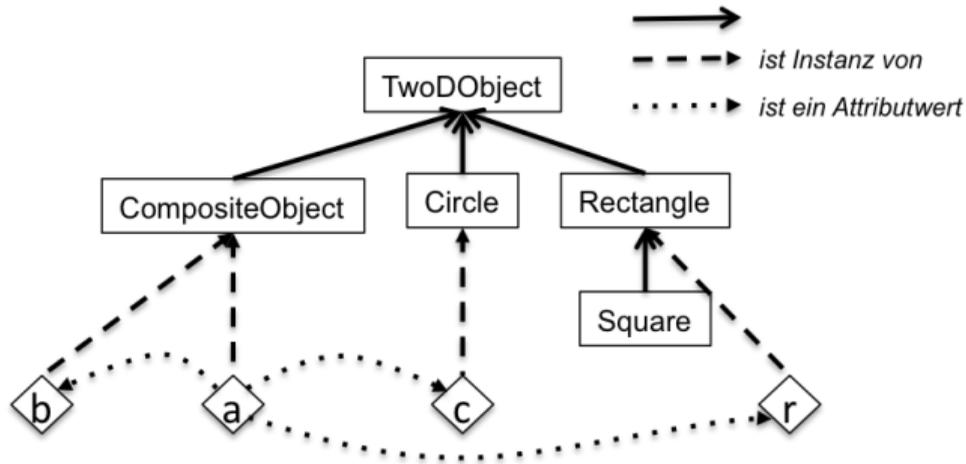
Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen



Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

2 Properties



Aggregie-
rung

Properties

Operator-
Überladung

Dictionaries

Mengen

Zugriff auf Attribute kontrollieren: Getter und Setter



- Ziel ist **Kontrolle** über das Abfragen und Setzen von Attributwerten.
 - Invarianten zwischen Attributwerten sollen respektiert werden.
Es soll nicht möglich sein, unsinnige Attributwerte zu setzen.
 - Der Zustand eines Objekts soll gekapselt werden.
- In anderen Sprachen können Attribute als **privat** deklariert werden.
 - Nur Methoden des zugehörigen Objekts können sie lesen bzw. ändern.
 - Sie sind unsichtbar für Objekte anderer Klassen.

⇒ **Datenkapselung**; **Invarianten** können garantiert werden.
- Für den Zugriff durch andere Objekte werden (häufig) **Getter**- und (seltener) **Setter**-Methoden bereitgestellt.
 - Eine Getter-Methode liest ein privates Attribut.
 - Eine Setter-Methode schreibt ein privates Attribut.
- In Python sind Attribute im wesentlichen *öffentlich*, aber sie können durch Getter und Setter als **Properties** geschützt werden.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Definition: Dateninvariante

Eine Dateninvariante ist eine logische Aussage über die Attribute eines Objekts, die während der gesamten Lebensdauer des Objekts erfüllt sein muss.

- Der Konstruktor muss die Dateninvariante sicherstellen.
- Die Methoden müssen die Dateninvariante erhalten.
- Unbewachtes Ändern eines Attributs kann die Dateninvariante zerstören.

Definition: Datenkapselung

Attribute (Objektzustand) können nicht direkt gelesen oder geändert werden.

- Die Interaktion mit einem Objekt geschieht nur durch Methoden.
- Die Implementierung (Struktur des Objektzustands) kann verändert werden, ohne dass andere Teile des Programms geändert werden müssen.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Beispiel Invariante: Radius eines Kreises



■ Invariante

Das Attribut `radius` der Klasse `Circle` soll immer größer als Null sein.

- **Regel 1:** Jede Invariante **muss** im docstring der Klasse dokumentiert sein!

```
@dataclass
class Circle(TwoDObject):
    '''Represents a circle in the plane.

    Attributes:
        radius: a number indicating the radius of the circle
        x, y: inherited from TwoDObject

    Invariants:
        radius > 0
    '''
    radius : float
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Beispiel Invariante: Radius eines Kreises



- Der docstring kann Verletzungen der Invariante nicht verhindern...
- **Regel 2: Der Konstruktor muss die Einhaltung der Invariante prüfen!**
- Die Prüfung geschieht durch eine Assertion in einer speziellen Methode `__post_init__`. Verletzung führt zu einer **Exception** (Ausnahme).
- `__post_init__` wird automatisch bei Konstruktion einer Instanz einer Datenklasse aufgerufen.

```
@dataclass
class Circle(TwoDObject):
    ...
    radius : float

    def __post_init__(self):
        assert self.radius > 0, "radius should be greater than 0"
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

- Bei falschem Aufruf des Konstruktors wird eine Exception ausgelöst.

Python-Interpreter

```
>>> c = Circle (x=10,y=20, radius=-3)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File ".../properties.py", line 46, in __init__
```

```
    assert radius > 0, "radius should be greater than 0"
```

```
AssertionError: radius should be greater than 0
```

Aggregie-
rung

Properties

Operator-
Überladung

Dictionaries

Mengen

Beispiel: Radius eines Kreises



- Ein böswilliger Mensch kann immer noch folgenden Code schreiben:

```
c = Circle(x=20, y=20, radius=5)
c.radius = -3  ## object invariant broken
```

- **Regel 3:** Das Attribut `radius` muss als Property ohne Setter definiert werden!

```
@dataclass
class Circle(TwoDObject):
    ...
    __radius : float ## Namen von Properties beginnen mit __

    def __post_init__(self):
        assert self.__radius > 0, "radius should be greater than 0"

    @property
    def radius (self) -> float:
        return self.__radius
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Was passiert?



- Der Attributwert für den Radius wird im Feld `__radius` des Objekts gespeichert. **Felder, deren Name mit `__` beginnt, sind von außen nicht ohne weiteres zugreifbar!**
- `radius` ist eine normale **Methode**, der **Getter** für `radius`.
- Die Dekoration mit `@property` bewirkt, dass `radius` wie ein Attribut verwendet werden kann.
- Ein Attributzugriff `c.radius` wird als Methodenaufruf `c.radius()` interpretiert.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Python-Interpreter

```
>>> c = Circle (x=10,y=20, radius=3)
```

```
>>> c.radius
```

```
3
```

```
>>> c.x = -3
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
AttributeError: can't set attribute
```

Eine (Daten-) Invariante ist eine logische Aussage über die Attribute eines Objekts, die während der gesamten Lebensdauer des Objekts erfüllt sein muss.

Regeln zu Dateninvarianten

- 1 Jede Invariante muss im docstring der Klasse dokumentiert sein!
- 2 Der Konstruktor muss die Einhaltung der Invariante prüfen!
- 3 Die Attribute, die in der Invariante erwähnt werden, müssen als Properties ohne Setter definiert werden!

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Aufgabe

Ein Zeichenprogramm verwendet Punkte in der Ebene. Die Hauptoperation auf Punkten ist die Drehung (um den Ursprung) um einen bestimmten Winkel.

Erster Versuch

```
@dataclass
class Point2D:
    x : float
    y : float

    def turn(self, phi : float):
        self.x, self.y = (self.x * cos(phi) - self.y * sin(phi)
                        ,self.x * sin(phi) + self.y * cos(phi))
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Was passiert?



Python-Interpreter

```
>>> pp = Point2D(1,0)
>>> pp.x, pp.y
(1, 0)
>>> pp.turn(pi/2)
>>> pp.x, pp.y
(6.123233995736766e-17, 1.0)
>>> pp.y = -1
>>> pp.turn (pi/2)
>>> pp.x, pp.y
(1.0, 0.0)
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

- Das Interface von `Point2D` Objekten besteht aus den Attributen `x`, `y` und der Methode `turn()`.
- Jeder Aufruf von `turn()` erfordert vier trigonometrische Operationen (naja, mindestens zwei), die vergleichsweise aufwändig sind.
- Möglichkeit zur Vermeidung dieser Operationen: Ändere die Datenrepräsentation von rechtwinkligen Koordinaten (x, y) in **Polarkoordinaten** (r, ϑ) . In Polarkoordinaten entspricht eine Drehung um φ der Addition der Winkel $\vartheta + \varphi$.
- Aber: das Interface soll erhalten bleiben!
- Ein Fall für Datenkapselung mit Gettern **und** Settern!
- (keine Invariante: `x` und `y` sind beliebige Zahlen!)

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Datenkapselung: Änderung der Repräsentation ohne Änderung des Interface



```
@dataclass
class PointPolar:
    x : InitVar[float]
    y : InitVar[float]

    def __post_init__(self, x:float, y:float):
        self.__r = sqrt(x*x + y*y)
        self.__theta = atan2(y, x)

    def turn(self, phi:float):
        self.__theta += phi
    ...
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

- x und y definieren nur die Parameter für den Konstruktor (Effekt von InitVar)
- Repräsentation durch Polarkoordinaten
- Interne Attribute `__r` und `__theta` von außen nicht zugreifbar

Datenkapselung: Interface rechtwinklige Koordinaten



```
@property
def x (self) -> float:
    return self.__r * cos (self.__theta)
@property
def y (self) -> float:
    return self.__r * sin (self.__theta)
@x.setter
def x (self, x : float):
    self.__post_init__ (x, self.y)
@y.setter
def y (self, y : float):
    self.__post_init__ (self.x, y)
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

- Definition der Getter wie gehabt.
- Definition der Setter dekoriert mit `@x.setter`, wobei `x` der Propertyname ist.
- Methodendefinition für den Propertynamen mit einem Parameter (+ `self`).
- Eine Zuweisung `p.x = v` wird interpretiert als Methodenaufruf `p.x(v)`.

Was passiert? Exakt das Gleiche wie mit Point2D!



Python-Interpreter

```
>>> pp = PointPolar(1,0)
>>> pp.x, pp.y
(1, 0)
>>> pp.turn(pi/2)
>>> pp.x, pp.y
(6.123233995736766e-17, 1.0)
>>> pp.y = -1
>>> pp.turn (pi/2)
>>> pp.x, pp.y
(1.0, 0.0)
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

- Intern könnte der Punkt **beide** Repräsentationen vorhalten.
- Nur die jeweils benötigte Repräsentation wird berechnet.
- Transformationen werden immer in der günstigsten Repräsentation ausgeführt:
Rotation in Polarkoordinaten, Translation in rechtwinkligen Koordinaten

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

3 Operator-Überladung



- Überladung von arithmetischen Operatoren
- Überladung von Vergleichsoperatoren
- Zusammenfassung

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen



- Ein **Operator** ist **überladen** (operator overloading), wenn dieser Operator je nach Typ der Argumente (und ggf. dem Kontext) unterschiedlich definiert ist.
- Traditionell sind die arithmetischen Operatoren in vielen Programmiersprachen für alle numerischen Typen überladen.
- In Python sind außerdem die Operatoren „+“ und „*“ für Strings überladen.
- In Python können gewisse Operatoren überladen werden.
- **Überladung ist immer mit Vorsicht zu genießen:**
 - Falls ein Operator wie „+“ überladen ist, ist es im Programmtext nicht offensichtlich, welcher Code ausgeführt wird.
 - Eine Überladung darf nicht “die Intuition” eines Operators verletzen.
 - Beispiel: „+“ (auf Zahlen) hat Eigenschaften wie Kommutativität, Assoziativität, 0 als neutrales Element, etc, die durch Überladung nicht gestört werden sollten.

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

Beispiel: Addition für 2D-Punkte



point2d.py (1)

```
class Point2D:  
    ...  
    def __add__(self, other):  
        return Point2D (self.x + other.x, self.y + other.y)
```

- Die “magische Methode” `__add__` definiert die Überladung des „+“-Operators.
- Wenn `pp = Point2D (...)`, dann wird eine “Addition” `pp + v` als Methodenaufruf `pp.__add__(v)` interpretiert.
- Was fehlt hier?
- Was passiert, wenn `other` keine Instanz von `Point2D` ist?

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren
Zusammenfassung

Dictionaries

Mengen

Beispiel: Addition für 2D-Punkte

point2d.py

```
class Point2D:
    ...
    def __add__(self, other : Point2D):
        if isinstance (other, Point2D):
            return Point2D (self.x + other.x, self.y + other.y)
        else:
            raise TypeError ("Cannot add Point2D and " + str (type (other)))
```

- Der Funktionsaufruf `isinstance (other, Point2D)` testet, ob `other` eine Instanz von `Point2D` ist.
- Hier wird eine Exception erzeugt, aber alles ist möglich (aber nicht unbedingt sinnvoll).

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

Beispiel: Multiplikation für 2D-Punkte

mit den magischen Methoden `__mul__` und `__rmul__`



point2d.py

```
class Point2D:
    ...
    def __mul__ (self, other : Union[Point2D,numbers.Number]):
        if isinstance (other, Point2D):          # scalar product
            return self.x * other.x + self.y * other.y
        elif isinstance (other, numbers.Number): # scalar multiplication
            return Point2D (other * self.x, other * self.y)
        else:
            raise TypeError ("Cannot multiply Point2D and " + str (type (other)))

    def __rmul__ (self, other : numbers.Number):
        if isinstance (other, numbers.Number):
            return Point2D (other * self.x, other * self.y)
        else:
            raise TypeError ("Cannot multiply " + str (type (other)) + " and Point2D")
```

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren
Zusammenfassung

Dictionaries

Mengen

Was passiert?

Python-Interpreter

```
>>> p1 = Point2D (1,0)
>>> p1.x, p1.y
(1, 0)
>>> p2 = p1 * 42 # multiply p1 with a number
>>> p2.x, p2.y # yields a point
(42, 0)
>>> w = p1 * p2 # multiply two points
>>> w # yields a number
42
>>> p3 = 3 * p1 # multiply a number with a point
>>> p3.x, p3.y # yields a point
(3, 0)
```

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen



Was passiert intern?

- `p1 * 42` entspricht `p1.__mul__(42)`; `other` ist eine Zahl
- `p1 * p2` entspricht `p1.__mul__(p2)`; `other` ist eine Instanz von `Point2D`
- `3 * p1` entspricht ...
- `3.__mul__(p1)` ... — im Prinzip; kann so nicht eingegeben werden
- aber der Type `int` kann nicht mit einem `Point2D` multiplizieren und liefert **den Wert `NotImplemented`**.
- Daraufhin versucht Python `p1.__rmul__(3)` ...
- was ein Ergebnis liefert.
- Die arithmetischen Operatoren `+`, `*`, `-`, `/` und `%` können nach dem gleichen Muster überladen werden.

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

Vergleich: `__eq__`, `__ne__`

- `obj.__eq__(other)`:
Wird zur Auswertung von `obj == other` aufgerufen.
Wird zur Auswertung von `other == obj` aufgerufen, falls `other` keine `__eq__` Methode besitzt.
- `obj.__ne__(other)`:
Wird zur Auswertung von `obj != other` (oder `other != obj`) aufgerufen.
- Sind diese Methoden nicht definiert, werden Objekte nur auf Identität verglichen, d.h. `x == y` gdw. `x is y`.
- Der Aufruf von `!=` gibt automatisch das Gegenteil vom Aufruf von `==` zurück, außer wenn `==` das Ergebnis `NotImplemented` liefert. Es reicht also, `obj.__eq__(other)` zu implementieren.

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

Equality

```
@dataclass
class Point2D:
    ...
    def __eq__(self, other):
        return (isinstance(other, Point2D) and
                self.x == other.x and self.y == other.y)
```

- Die Methode `__eq__` wird für Datenklassen automatisch erzeugt, wenn sie nicht explizit definiert wird.

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

Vergleich: `__ge__`, `__gt__`, `__le__`, `__lt__`



- `obj.__ge__(other)`:
Wird zur Auswertung von `obj >= other` aufgerufen.
Wird ebenfalls zur Auswertung von `other <= obj` aufgerufen, falls `other` über keine `__le__`-Methode verfügt.
- `obj.__gt__(other)`, `obj.__le__(other)`, `obj.__lt__(other)`:
Werden analog für die Vergleiche `obj > other` bzw. `obj <= other` bzw. `obj < other` aufgerufen.
- Auch die Vergleichsmethoden können automatisch durch die Datenklasse erzeugt werden, wenn `order=True` angegeben wird:

```
@dataclass(order=True)
class Point2D:
    x : float
    y : float
```

Aggregation

Properties

Operator-Überladung

Überladung von arithmetischen Operatoren

Überladung von Vergleichsoperatoren

Zusammenfassung

Dictionaries

Mengen

- **Aggregation** liegt vor, falls Attribute von Objekten selbst wieder Objekte sind.
- **Properties** erlauben die Realisierung von **Invarianten** und **Datenkapselung**.
Attributzugriffe werden über Getter und Setter (Methoden) abgewickelt.
- **Überladung** liegt vor, wenn ein Operator die anzuwendende Operation anhand des Typs der Operanden bestimmt.
- Python verwendet **magische Methoden** zur Implementierung der Überladung von Operatoren.

Aggregation

Properties

Operator-
Überladung

Überladung von
arithmetischen
Operatoren

Überladung von
Vergleichsoperato-
ren

Zusammenfassung

Dictionaries

Mengen

4 Dictionaries



- Beispiele
- Operationen
- Geschachtelte Dicts
- Views
- Dicts als Hashtabellen
- Veränderliche Dict-Keys?

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

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*
- 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,
 - Anwesenheit eines Schlüssels
- Voraussetzungen
 - Schlüssel müssen auf Gleichheit getestet werden können!
 - Schlüssel müssen unveränderlich (immutable) sein!

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

- Dictionaries sind so implementiert, dass der Wert zu einem gegebenen Schlüssel sehr **effizient** unabhängig von der Anzahl der bestehenden Einträge bestimmt werden kann.
- Dictionaries sind **ungeordnet**; d.h., es ist nicht sinnvoll, von einem ersten (zweiten, usw.) Element zu sprechen.
- (Ein heißes Thema zur Zeit sind **key-value stores**; das sind Netz-weit verteilte Dictionaries.)

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

Python-Interpreter

```
>>> description = {"walk": "silly", "parrot": "dead",
...                (1, 2, 3): "no witchcraft"}
>>> description["parrot"]
'dead'
>>> "walk" 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', 'walk': 'silly'}
```

Aggregation

Properties

Operator-
Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

Dictionaries können auf verschiedene Weisen **erzeugt** werden (Auswahl):

- `{key1: value1, key2: value2, ...}`:
Hier sind `key1`, `key2`, ... **unveränderliche Python-Objekte**, d.h. Zahlen, Strings, Tupel, etc. Für `value1`, `value2`, ... dürfen beliebige Python-Objekte verwendet werden.
- `dict(key1=value1, key2=value2, ...)`:
Hier sind die Schlüssel `key1`, `key2`, ... **Variablennamen**, die vom `dict`-Konstruktor in Strings konvertiert werden.
Die Werte `value1` usw. sind beliebige Objekte.
- `dict(sequence_of_pairs)`:
`dict([(key1, value1), (key2, value2), ...])` entspricht `{key1: value1, key2: value2, ...}`.

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

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'}
```

Aggregation

Properties

Operator-
Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

Sei d ein Dict:

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

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

- `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).

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

get: Beispiel

food_inventory.py

```
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.")

# Ausgabe:
# We have enough egg for 1 people.
# We have enough vinegar for 0 people.
# We have enough cheese for 4 people.
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

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 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.

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

- Auch Dicts können selbst Dicts enthalten.

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'
```

Aggregation

Properties

Operator-
Überladung

Dictionaries

Beispiele

Operationen

**Geschachtelte
Dicts**

Views

Dicts als
Hashtabellen

Veränderliche
Dict-Keys?

Mengen

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)`-Assoziationen 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*.

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

Wie funktionieren Dictionaries?



Dictionaries sind als **Hashtabellen** implementiert:

- Bei der Erzeugung eines Dictionaries wird eine große Tabelle (die **Hashtabelle**) eingerichtet.
- Jedem Schlüssel wird mit Hilfe einer **Hashfunktion** ein Tabellenindex (der **Hashwert**) zugeordnet.
- 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.
- 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**.

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

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		

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

- Schlüssel müssen hash-bar sein und auf Gleichheit getestet werden können
- Hashtabellen haben **keine spezielle Ordnung** für die Elemente.
- Daher liefert `keys` die Schlüssel nicht in der Einfügereihenfolge, sondern in einer unvorhersehbaren Abfolge.
- Objekte, die als Schlüssel in einem Dictionary verwendet werden, dürfen **nicht verändert** werden. Ansonsten könnte es zu Problemen kommen.

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

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?
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als
Hashtabellen

Veränderliche
Dict-Keys?

Mengen

- 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!

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche
Dict-Keys?

Mengen

Veränderliche Dictionary-Keys (3)



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'
```

Aggregation

Properties

Operator-Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

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.
- In der Funktion kann die Variable `kwargs` wie ein Dictionary verwendet werden.

Python-Interpreter

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

Aggregation

Properties

Operator-
Überladung

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als

Hashtabellen

Veränderliche

Dict-Keys?

Mengen

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

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- **Mengen** sind Zusammenfassungen von Elementen (hier immer endlich),
- Grundoperationen auf Mengen:
 - Einfügen eines Elements,
 - Entfernen eines Elements,
 - Test ob Element enthalten ist.
- Voraussetzungen
 - Elemente müssen auf Gleichheit getestet werden können!
 - Elemente müssen unveränderlich (immutable) sein!
- Mengenelemente sind einzigartig; eine Menge kann also nicht dasselbe Element ‚mehrmals‘ beinhalten (\Rightarrow Multimenge).

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



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

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

- 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.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

Wir teilen die Operationen auf Mengen in Gruppen ein:

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

Aggregation

Properties

Operator-
Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

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`).

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

Konstruktion von Mengen: Beispiele (1)



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'}
```

Aggregation

Properties

Operator-
Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung

Konstruktion von Mengen: Beispiele (2)



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'})}
```

Aggregation

Properties

Operator-Überladung

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

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung

- `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`.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und Entfernen

Zusammenfassung

Viele weitere Operationen



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

Aggregation

Properties

Operator-Überladung

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.
- `sets` sind veränderliche Strukturen, `frozensets` sind nicht veränderlich.

Aggregation

Properties

Operator-Überladung

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende

Operationen

Einfügen und

Entfernen

Zusammenfassung