

Informatik I: Einführung in die Programmierung

13. Objekt-orientierte Programmierung: Aggregation, Invarianten, Datenkapselung, Properties,

Albert-Ludwigs-Universität Freiburg



UNI
FREIBURG

Peter Thiemann

15.01.2019



Aggregierung

Aggregierung

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



- Oft sind Objekte aus anderen Objekten **zusammengesetzt**.
- Methodenaufrufe an ein Objekt führen dann 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

Der Zoo der magischen Methoden

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)

```
class CompositeObject(TwoDObject):
    def __init__(self, objs=(), **kwargs):
        self.objects = list(objs)
        super().__init__(None, None)

    def add(self, obj):
        self.objects.append(obj)

    def rem(self, obj):
        self.objects.remove(obj)

    ...
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

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):
    for obj in self.objects:
        obj.size_change(percent)

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

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((r,c))
>>> a.size_change(200)
>>> r.area()
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((r,c))
>>> a.size_change(200)
>>> r.area()
400.0
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((r,c))
>>> a.size_change(200)
>>> r.area()
400.0
>>> a.move(40,40)
>>> a.position()
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((r,c))
>>> a.size_change(200)
>>> r.area()
400.0
>>> a.move(40,40)
>>> a.position()
(40, 40)
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((r,c))
>>> a.size_change(200)
>>> r.area()
400.0
>>> a.move(40,40)
>>> a.position()
(40, 40)
>>> c.position()
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

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

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((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

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Die CompositeObject-Klasse (3)



Python-Interpreter

```
>>> c = Circle(x=1,y=2); r = Rectangle(10,10)
>>> a = CompositeObject((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

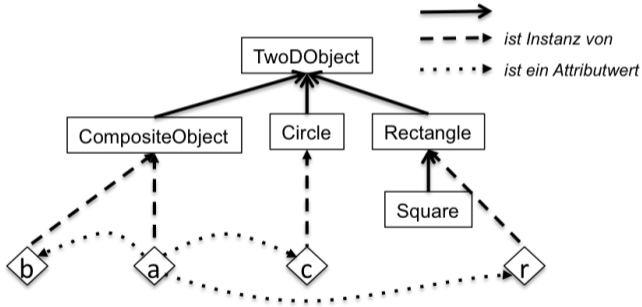
Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



Properties

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Zugriff auf Attribute kontrollieren: Getter und Setter



- Gewünscht: **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

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

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

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



■ Invariante

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

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

```
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
    '''
    def __init__(self, radius=1, **kwargs):
        self.radius = radius
        super().__init__(**kwargs)
```

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



- Der docstring kann Verletzungen der Invariante nicht verhindern...
- **Regel 2: Der Konstruktor muss die Einhaltung der Invariante prüfen!**
- Die Prüfung geschieht durch eine Assertion. Verletzung führt zu einer **Exception** (Ausname).

```
class Circle(TwoDObject):  
    ...  
    def __init__(self, radius=1, **kwargs):  
        assert radius > 0, "radius should be greater than 0"  
        self.radius = radius  
        super().__init__(**kwargs)
```

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

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
```

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Beispiel: Radius eines Kreises



- Ein böswilliger Mensch kann folgenden Code schreiben:

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

- **Regel 3:** Das Attribut `radius` muss privat definiert werden!

```
class Circle(TwoDObject):
    def __init__(self, radius=1, **kwargs):
        assert radius > 0, "radius should be greater than 0"
        self.__radius = radius
        super().__init__(**kwargs)

    def radius(self):
        return self.__radius

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

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

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!**
- Stattdessen kann auf den Radius indirekt zugegriffen werden, über die **Getter-** und **Setter-**Methoden `radius` und `set_radius`.
- Die Dateninvariante `radius > 0` ist gewährleistet.

Python-Interpreter

```
>>> c = Circle(x=10,y=20, radius=3)
>>> c.__radius
[...] AttributeError: Circle instance has no attribute '__radius'
>>> c.set_radius(5)
>>> c.radius()
5
>>> c.set_radius(-3)
[...] AssertionError: radius should be greater than 0
```

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Beispiel: Radius eines Kreises



- Um wie zuvor `c.radius` und `c.radius = 5` schreiben zu können, kann man `radius` als **Property** definieren:

```
class Circle(TwoDObject):
    ...

    @property
    def radius(self):
        return self.__radius

    @radius.setter
    def radius(self, radius):
        assert radius > 0, "radius should be greater than 0"
        self.__radius = radius
```

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



Python-Interpreter

```
>>> c = Circle(x=10,y=20, radius=3)
>>> c.radius = 5
>>> c.radius
5
>>> c.radius = -5
[...] AssertionError: radius should be greater than 0
```

- Lässt man den Setter weg, so kann man den Radius weiterhin lesen, aber das Schreiben führt zu einem Laufzeitfehler:

Python-Interpreter

```
>>> c = Circle(x=10,y=20, radius=3)
>>> c.radius = 5
[...] AttributeError: can't set attribute
```

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



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 Jede Methode, inklusive Konstruktor, muss die Einhaltung der Invariante prüfen!
- 3 Die Attribute, die in der Invariante erwähnt werden, müssen privat definiert werden!
- 4 Mit Properties kann man die gewohnte Schreibweise für Attributzugriffe zurückgewinnen.

Aggregation

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

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

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

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

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



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

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

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

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Datenkapselung: Änderung der Repräsentation ohne Änderung des Interface



```
class PointPolar:
    def __init__(self, x, y):
        self.set_xy(x, y)

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

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

- Repräsentation durch Polarkoordinaten
- Interne Attribute `__r` und `__theta` von außen nicht zugreifbar

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



```
@property
def x(self):
    return self.__r * cos(self.__theta)
@property
def y(self):
    return self.__r * sin(self.__theta)
@x.setter
def x(self, x):
    self.set_xy(x, self.y)
@y.setter
def y(self, y):
    self.set_xy(self.x, y)
```

Aggregation

Properties

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

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

Zusammenfassung

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen



- **Aggregierung** 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.

Aggregierung

Properties

Zusammenfassung

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



Operator-Überladung

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

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, wobei nur der Typ der Argumente berücksichtigt wird.
- **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

Der Zoo der magischen Methoden

Dictionaries

Mengen



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
- Der Zoo der magischen Methoden
- Dictionaries
- Mengen



point2d.py (2)

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

- Der Funktionsaufruf `isinstance(other, Point2D)` testet, ob `other` eine Instanz von `Point2D` ist.

Beispiel: Multiplikation für 2D-Punkte

mit den magischen Methoden `__mul__` und `__rmul__`



point2d.py (3)

```
class Point2D:
    ...
    def __mul__(self, other):
        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:
            return NotImplemented

    def __rmul__(self, other):
        if isinstance(other, numbers.Number):
            return Point2D(other * self.x, other * self.y)
        else:
            return NotImplemented
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen



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

Der Zoo der magischen Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Mengen



Der Zoo der magischen Methoden

Aggregation

Properties

Operator-
Überladung

**Der Zoo der
magischen
Methoden**

Allgemeine
magische
Methoden

Numerische
magische
Methoden

Dictionaries

Mengen



- Methoden wie `__init__`, deren Namen mit zwei Unterstrichen beginnen und enden, heißen *magisch*.
- Es gibt eine Vielzahl an magischen Methoden, die z.B. verwendet werden können, um Operatoren wie `+` und `%` für eigene Klassen zu definieren.
- Magische Methoden wie `__add__` sind nicht prinzipiell anders als andere Methoden, aber wenn sie vorhanden sind, werden sie bei geeigneter Gelegenheit von Python intern aufgerufen.

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Es gibt drei Arten von magischen Methoden:

- Allgemeine Methoden: verantwortlich für Objekterzeugung, Ausgabe und ähnliche grundlegende Dinge.
- Numerische Methoden: verantwortlich für Addition, Bitshift und ähnliches
- Container Methoden: verantwortlich für Indexzugriff, Slicing und ähnliches

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Die allgemeinen magischen Methoden werden weiter unterteilt:

- Konstruktion und Destruktion: `__init__`, `__new__`, `__del__`
- Vergleich und Hashing: `__eq__`, `__ne__`, `__ge__`, `__gt__`, `__le__`, `__lt__`, `__hash__`, `__bool__`
- String-Konversion: `__str__`, `__repr__`, `__format__`
- Verwendung einer Instanz als Funktion: `__call__`
- Attributzugriff: `__getattr__`, `__getattribute__`, `__setattr__`, `__delattr__`
- Magische Attribute: `__dict__` (das dict der Attributnamen) und `__slots__` (um Attribute zu beschränken)

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Die allgemeinen magischen Methoden werden weiter unterteilt:

- Konstruktion und Destruktion: `__init__`, `__new__`, `__del__`
- Vergleich und Hashing: `__eq__`, `__ne__`, `__ge__`, `__gt__`, `__le__`, `__lt__`, `__hash__`, `__bool__`
- String-Konversion: `__str__`, `__repr__`, `__format__`
- Verwendung einer Instanz als Funktion: `__call__`
- Attributzugriff: `__getattr__`, `__getattribute__`, `__setattr__`, `__delattr__`
- Magische Attribute: `__dict__` (das dict der Attributnamen) und `__slots__` (um Attribute zu beschränken)

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

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

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Equality

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Equality

```
class Point2D:  
    ...  
    def __eq__(self, other):  
        return ((type(other) is Point2D) and  
                self.x == other.x and self.y == other.y)
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Allgemeine
magische
Methoden

Numerische
magische
Methoden

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.

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



- Bei Operatoren wie `+`, `*`, `-` oder `/` verhält sich Python wie folgt (am Beispiel `+`):
- Zunächst wird versucht, die Methode `__add__` des linken Operanden mit dem rechten Operanden als Argument aufzurufen.
- Wenn die Methode `__add__` mit dem Typ des rechten Operanden nichts anfangen kann, kann sie die spezielle Konstante `NotImplemented` zurückliefern. Dann wird versucht, die Methode `__radd__` des rechten Operanden mit dem linken Operanden als Argument aufzurufen.
- Wenn dies auch nicht funktioniert, schlägt die Operation fehl.

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Allgemeine magische Methoden

Numerische magische Methoden

Dictionaries

Mengen



Dictionaries

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte
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

Der Zoo der magischen Methoden

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.
- Im Gegensatz zu Sequenzen (also Listen, Tupeln, etc) sind Dictionaries **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 verteilte Dictionaries, die im Netz implementiert sind.)

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte 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

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

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

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

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

Der Zoo der
magischen
Methoden

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

Der Zoo der
magischen
Methoden

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'
>>> x = dicts['en->de']['blue']
>>> dicts['de->fr'][x]
'bleu'
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



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

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Eingabe:

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4		
5		
6		

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

**Dicts als
Hashtabellen**

Veränderliche
Dict-Keys?

Mengen



Eingabe: ('parrot', 'dead')

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4		
5		
6		

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



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

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4		
5		
6		

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Eingabe:

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4	'parrot'	'dead'
5		
6		

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche

Dict-Keys?

Mengen



Eingabe: ('spam', 'tasty')

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4	'parrot'	'dead'
5		
6		

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte
Dicts

Views

**Dicts als
Hashtabellen**

Veränderliche
Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Eingabe: ('spam', 'tasty')
hash('spam')=0

Hashtabelle

Index	Key	Value
0		
1		
2		
3		
4	'parrot'	'dead'
5		
6		

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Eingabe:

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Eingabe: ('zehn', 10)

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Eingabe: ('zehn', 10)
hash('zehn')=4

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Eingabe: ('zehn', 10)
hash('zehn')=4 **Konflikt!**

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Eingabe:

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Anfrage: 'parrot'

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Anfrage: 'parrot'
hash('parrot')=4

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Anfrage: 'parrot'
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

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Anfrage: 'zehn'

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Anfrage: 'zehn'
hash('zehn')=4

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Anfrage: 'zehn'
hash('zehn')=4

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen

Eine Hashtabelle bei der Arbeit



Anfrage: 'zehn'
hash('zehn')=4
Ausgabe:10

Hashtabelle

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte

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

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



```
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

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Python-Interpreter

```
>>> mydict = {"silly", "walk"): [1, 2, 3]}
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

Dicts als Hashtabellen

Veränderliche Dict-Keys?

Mengen



Python-Interpreter

```
>>> mydict = {"silly", "walk"): [1, 2, 3]}  
>>> mydict[[10, 20]] = "spam"
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

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'
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

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
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte Dicts

Views

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'
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

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

Der Zoo der
magischen
Methoden

Dictionaries

Beispiele

Operationen

Geschachtelte
Dicts

Views

Dicts als
Hashtabellen

Veränderliche
Dict-Keys?

Mengen



Mengen

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

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

Der Zoo der magischen Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



Python-Interpreter

```
>>> set("spamspam")
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



Python-Interpreter

```
>>> set("spamspam")  
{'a', 'p', 's', 'm'}
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



Python-Interpreter

```
>>> set("spamspam")
{'a', 'p', 's', 'm'}
>>> frozenset("spamspam")
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



Python-Interpreter

```
>>> set("spamspam")
{'a', 'p', 's', 'm'}
>>> frozenset("spamspam")
frozenset({'a', 'p', 's', 'm'})
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



Python-Interpreter

```
>>> set("spamspam")
{'a', 'p', 's', 'm'}
>>> frozenset("spamspam")
frozenset({'a', 'p', 's', 'm'})
>>> set(["spam", 1, [2, 3]])
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



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'
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



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)))
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



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'}
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

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})
```

Aggregation

Properties

Operator-
Überladung

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



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

Der Zoo der
magischen
Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende
Operationen

Einfügen und
Entfernen

Zusammenfassung



Python-Interpreter

```
>>> s = set(["jam", "spam"])
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



Python-Interpreter

```
>>> s = set(["jam", "spam"])
>>> set([1, 2, 3, s])
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



Python-Interpreter

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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



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)])
```

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



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

Der Zoo der magischen Methoden

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

Der Zoo der
magischen
Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung



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

Aggregation

Properties

Operator-Überladung

Der Zoo der magischen Methoden

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

Der Zoo der magischen Methoden

Dictionaries

Mengen

Set und Frozenset

Operationen

Konstruktion

Grundlegende Operationen

Einfügen und Entfernen

Zusammenfassung