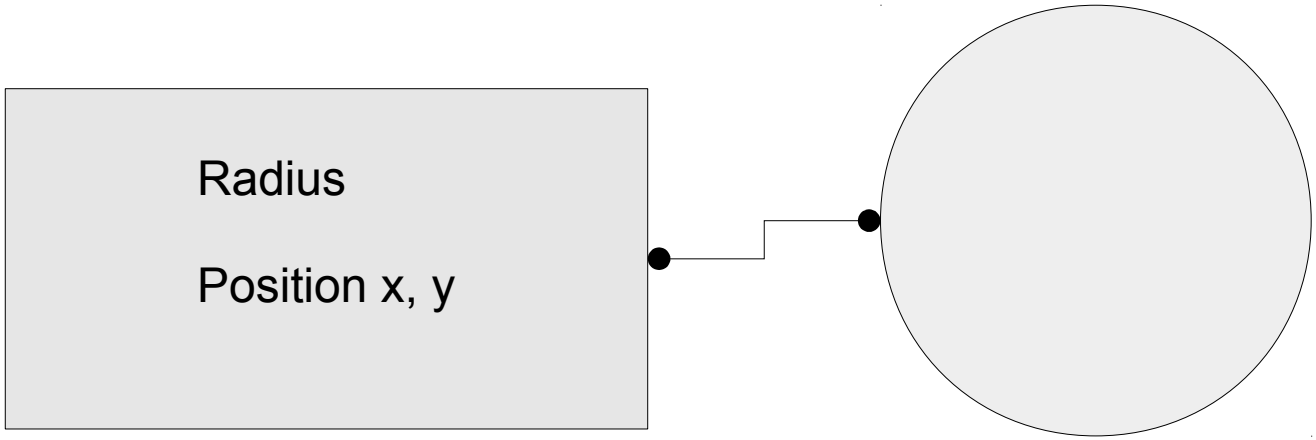
Python
„Gebundene und ungebundene Methoden“



Attribute (Member, Instanzvariablen)

- Beschreibung eines Gegenstandes, einer Person, etc.
- Speicherung und Definition von Eigenschaften für eine Instanz.
- Während der Lebenszeit eines Objekts können sich Attribute verändern.
- Einige Eigenschaften werden von allen Objekten gemeinsam genutzt.

Instanzvariablen

- Nutzung von Variablen in einer Klasse.
- Abbildung von Attributen eines Objekts in einer Programmiersprache.
- Speicherung von Attribut-Werten pro Instanz. Instanzen haben zwar die gleichen Attribute, aber die Ausprägungen können unterschiedlich sein.

… in Python

```
class clsRechteck(object):

    def __init__(self, breite = 1):
        self.breite = breite
        self.hoehe = None

    def getGroesse(self):
        strBreite = str(self.breite)
        strHoehe = str(self.hoehe)
        return (strBreite + " x " + strHoehe)

    def setGroesse(self, breite, hoehe = None):
            self.hoehe = hoehe
            self.breite = breite
```

Klassenvariablen

- Variablen in einer Klasse, die von allen Instanzen gemeinsam genutzt werden.
- Variablen, die in einer Klasse, aber außerhalb einer Methode definiert werden.
- Variablen, die nur einmal pro Klasse existieren.

Nutzung

- Variablen, die das Ergebnis von statischen Berechnungen über alle Instanz hinweg speichern.
- Zählen von Instanzen einer Klasse.
- „Konstante“ Werte einer Klasse.

… in Python

```
class clsArtikel(object):
    max_GelieferteMenge = 0
    max_Warenpreis = 0
    anzahl_ImLager = 0
    wert_WareImArtikel = 0

    def __init__(self, preis, menge = 1):
        self.menge = menge
        self.preis = preis

        clsArtikel.anzahl_ImLager = clsArtikel.anzahl_ImLager + menge
        clsArtikel.wert_WareImArtikel = clsArtikel.wert_WareImArtikel +
                                        (self.preis * self.menge)
```

… definieren

- Innerhalb der Klasse, aber außerhalb jeder Methode.
- Häufig am Anfang oder Ende der Klasse.
- Falls die Klassenvariable vor der Nutzung nicht definiert ist, wird der Laufzeitfehler *AttributeError: ... object has no attribute ...* angezeigt.

Wertzuweisung

max_Warenpreis	=	0
Variable	=	Ausdruck

- Mit Hilfe des Gleichheitszeichen wird der Klassenvariablen ein Wert zugewiesen.
- Die Klassenvariable wird automatisiert durch die Wertzuweisung angelegt.

Name einer Klassenvariablen

max_Warenpreis	=	0
Variable	=	Ausdruck

- Der Bezeichner kommt exakt einmal in der Klasse vor.
- Der Bezeichner der Klassenvariablen ist frei wählbar.
- Der Name sollte aber die Nutzung der Variablen widerspiegeln.

Regeln für Bezeichner

- Der Bezeichner beginnt mit einem Buchstaben oder Unterstrich.
- Ein Bezeichner besteht aus den lateinischen Groß- und Kleinbuchstaben und den Ziffern.
- Ein Bezeichner enthält als Sonderzeichen nur den Unterstrich.
- Die Groß- und Kleinschreibung wird beachtet.
- Schlüsselwörter der Programmiersprache und Bezeichnungen für Elemente aus der Standardbibliothek sind nicht erlaubt.

Anbindung an die Klasse

clsArtikel	.	max_Warenpreis	=	0
Klasse	.	Variable	=	Ausdruck

- Mit Hilfe des Punktoperators wird die Variable an eine Klasse gebunden.
- Die Variable ist immer an die Klasse gebunden, in der sie definiert ist.

Wert einer Klassenvariablen

clsArtikel	.	max_Warenpreis	=	0
Klasse	.	Variable	=	Ausdruck

- Der Startwert der Klassenvariablen wird bei der Definition angegeben.
- In der Initialisierungsmethode einer Klasse wird die Variable entsprechend ihrer Nutzung neu berechnet.
- In Klassenmethoden wird die Klassenvariable in Ausdrücken genutzt oder gelesen.

Initialisierungsmethoden

- Initialisierungsmethoden werden automatisiert durch die Erzeugung einer Instanz aufgerufen.
- Die Methode hat mindestens den Parameter `self` als Platzhalter für die zu erzeugende Instanz.

Beispiel

```
def __init__(self, bezeichnung, preis = None, menge = 1):
    self.warename = bezeichnung
    self.menge = menge
    self.preis = preis

    clsArtikel.anzahl_ImLager = clsArtikel.anzahl_ImLager + menge
    clsArtikel.wert_WareImArtikel = clsArtikel.wert_WareImArtikel +
                                    (self.preis * self.menge)

    if (clsArtikel.max_Warenpreis < preis):
        clsArtikel.max_Warenpreis = preis
```

Instanzvariablen

self	.	menge	=	menge
Instanz	.	Variable	=	Ausdruck

- Instanzvariablen werden in dieser Methode definiert. Jedes Objekt einer Klasse hat die gleichen Attribute.
- Der Attribut-Wert wird mit Hilfe von Parametern oder Literalen gesetzt. In der Ausprägung unterscheiden sich die Objekte einer Klasse.

Klassenvariablen

clsArtikel	.	max_Warenpreis	=	0
Klasse	.	Variable	=	Ausdruck

- Klassenvariablen sind statisch.
- Die Variable hat für alle Instanzen den gleichen Wert. In der Initialisierungsmethode wird dieser Wert entsprechend der zu erstellenden Instanz angepasst.
- Solange eine Instanz von der Klasse existiert, verliert die Variable ihren gespeicherten Wert nicht.

Instanzmethoden

- Methoden, die an ein Objekt gebunden sind.
- Methoden, die über eine Instanz aufgerufen werden.
- Die Methode hat mindestens den Parameter `self` als Platzhalter für Instanz, die die Methode aufruft.

Instanzvariablen

```
def getGesamtpreis(self):
    gesamtpreis = self.menge * self.preis
    print("Gesamtpreis eines konkreten Artikels: ", gesamtpreis)
```

- Schreiben und Lesen von Instanzvariablen.
- Die Attribute eines Objekts werden geändert oder gelesen.
- Aus Attribut-Werten werden neue Werte berechnet.

Klassenvariablen

```
def isGroesserAlsDurchschnitt(self):
    return (clsArtikel.getDurchschnittspreis() < self.preis)
```

- Klassenvariablen werden genutzt, um Aussagen über eine Instanz in Bezug auf alle Objekte einer Klasse zu treffen.
- In diesem Beispiel wird überprüft, ob der Preis der Ware (Instanz) größer als der Durchschnittspreis ist. Der Durchschnittspreis wird in Abhängigkeit aller Instanzen dieser Klasse berechnet.

Klassenmethoden

- Methoden, die einer Klasse zugeordnet sind.
- Ein Objekt wird für den Aufruf von Klassenmethoden nicht benötigt.
- Setzen und Schreiben von Klassenvariablen.
- Erstellung von Fabrikmethoden.

… in Python

```
class clsArtikel(object):
   @classmethod
   def getMaxPreis(cls):
       return cls.max_Warenpreis

   def getLagerbestand(cls):
       return cls.anzahl_ImLager

   lagerbestand = classmethod(getLagerbestand)
```

Aufbau des Methodenkopfs

- Jede Methode beginnt mit dem Schlüsselwort `def`.
- Dem Schlüsselwort folgt der Name der Methode. Der Name ist frei wählbar.
- Dem Methodennamen folgen die runden Klammern. In den runden Klammern wird eine Liste mit mindestens einem Parameter definiert.
- Der Methodenkopf endet mit einem Doppelpunkt.

Beispiel

def	create_Durchschnitt	(	cls	,	name					)
def	methode	(	self	,	par01	,	par02	,	...	)
def	get_Lagerbestand	(	self							)
def	methode	(	self							)

Name der Methode

Parameterliste

Auswahl des Namens

- Der Bezeichner kann sich aus den Buchstaben a...z, A...Z, die Zahlen 0...9 und den Unterstrich zusammensetzen.
- Der Name der Methode beginnt mit einem Kleinbuchstaben.
- Es wird die Groß- und Kleinschreibung beachtet.
- Der Name von Methoden ist in einer Klasse eindeutig.

Parameterliste

def	methode	(	cls	,	par01	,	par02	,	...	)
def	methode	(	self							)

- Die Parameterliste beginnt und endet mit den runden Klammern.
- Die Parameterliste hat als Element mindestens `cls`. Der Parameter verweist auf die Klasse, in der die Methode definiert ist.
- Die Parameter in der Liste werden durch ein Komma getrennt.

Parameter

def	methode	(	cls	,	par01	,	par02	,	...	)
def	methode	(	cls							)

- Parameter sind Variablen, die in der Methode in irgendeiner Form weiterverarbeitet werden.
- Parameter sind Platzhalter von Werten unterschiedlichsten Datentyps. Der Name wird als Label für einen beliebigen Wert genutzt.
- Der Name des Parameters ist in der Liste eindeutig.

Parameter cls

def	methode	(	cls	,	par01	,	par02	,	...	)
def	methode	(	cls							)

- In welcher Klasse ist die Methode definiert?
- An welche Klasse ist diese Methode gebunden?
- Welche Klasse ruft die Methode auf?
- Der Name cls ist eine Konvention. Es kann aber jeder beliebige anderer Name genutzt werden.

Kennzeichnung von Klassenmethoden

```
class clsArtikel(object):

    def getLagerbestand(cls):
        return cls.anzahl_ImLager

    lagerbestand = classmethod(getLagerbestand)
```

1. Schritt

- Die Klassenmethode wird definiert.
- Die Signatur der Klassenmethode wird definiert. Der Methodenrumpf kann leer sein.

Aufruf der Methode

```
clsArtikel.getLagerbestand(objArtikel01)

clsArtikel.getLagerbestand(clsArtikel)
```

- Der Punktoperator verbindet die Klasse mit der gewünschten Klassenmethode.
- Links vom Punktoperator darf nur der Name der Klasse, in der die Methode definiert ist, genutzt werden.
- Die Klassenmethode wird mit Hilfe des Namens aufgerufen.
- Ein Argument für den Parameter `cls` muss übergeben werden. Die gebundene Klasse kann aus einer Instanz ermittelt oder direkt angegeben werden.

2. Schritt: Kennzeichnung

bestand	=	classmethod	(	getLagerbestand	)
variable	=	classmethod	(	Methodenname	)

- Die Funktion classmethod ist eine eingebaute Funktion, die eine Methode in eine Klassenmethode konvertiert.
- Der Funktion wird als Parameter ein Name einer Methode aus der Klasse übergeben. Diese Methode wird konvertiert.
- Der Verweis auf die Klassenmethode wird in einer Variablen gespeichert. Die Variable entspricht dem Namen der Klassenmethode.

Aufruf der Methode

```
objArtikel01.lagerbestand()

clsArtikel.lagerbestand()
```

- Statt dem Methodennamen wird ein Variablenname genutzt. Die Variable verweist auf eine Klassenmethode.
- Die Variable wird wie ein Funktionsname genutzt.
- Die Klassenmethode kann über den Klassennamen oder eine Instanz aufgerufen werden.
- Ein Argument für den Parameter `cls` wird nicht benötigt. Beim Aufruf wird automatisiert die Angabe links vom Punktoperator genutzt.

Nutzung eines Decorators

```
@classmethod
    def getMaxPreis(cls):
        return cls.__max_preis
```

- Mit Hilfe des Decorator @classmethod wird die nachfolgende Methode dekoriert.
- Der Decorator passt die Funktionalität einer Methode entsprechend der Nutzung an.

Decorator

- Ein Decorator arbeitet ähnlich wie ein Hörgerät. Das Hörgerät verbessert das Hören des Ohres. Der Decorator verändert verbessert die Funktionalität einer Methode.
- In die Zeile direkt vor einem Methodenkopf können beliebig viele Decoratoren geschrieben werden.
- Decoratoren beginnen immer mit dem At-Zeichen. Der Name des Decorators spiegelt die Art der Modifizierung wieder.
- Beispiele siehe https://wiki.python.org/moin/PythonDecorators

Aufruf der Methode

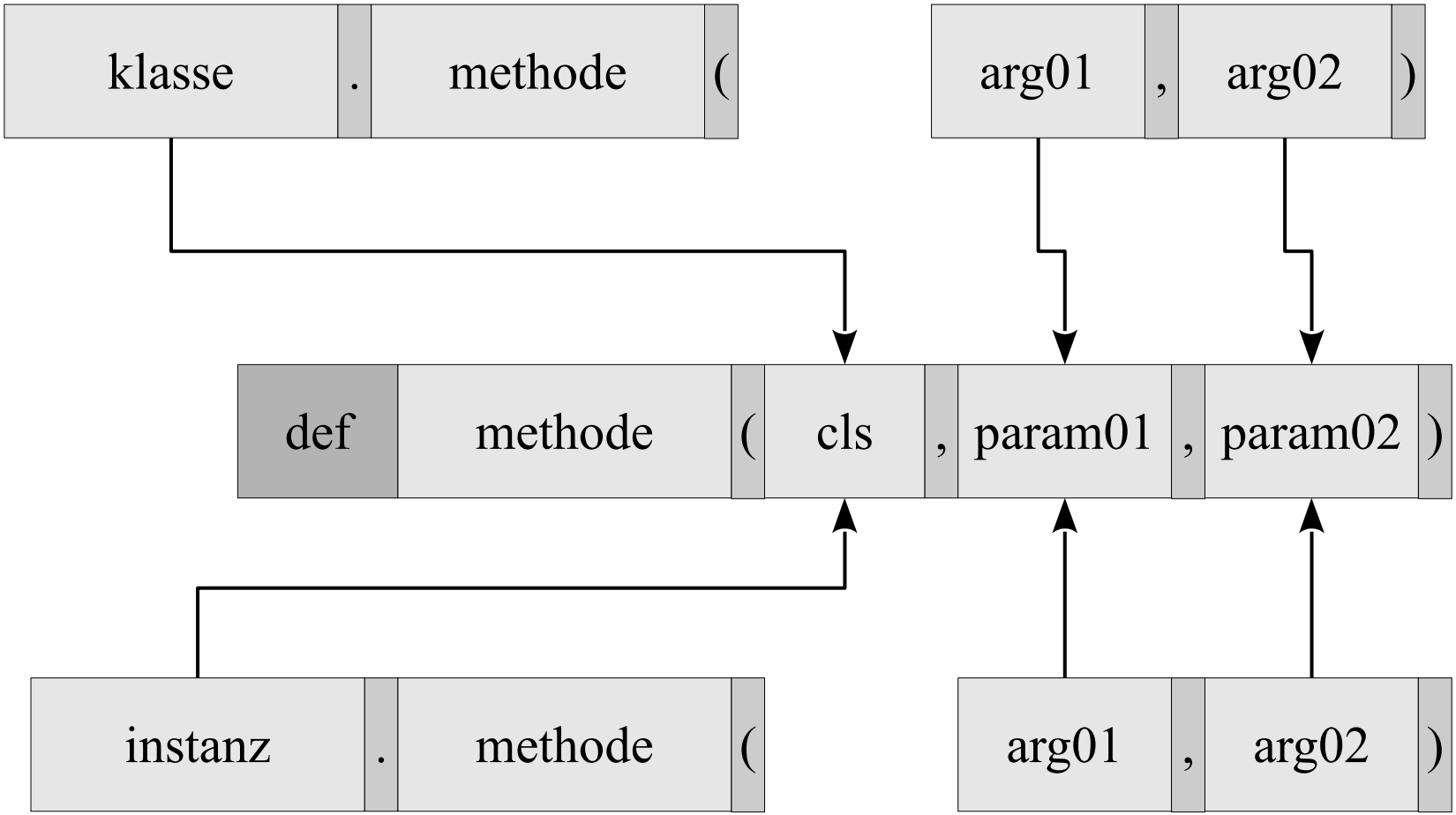
```
clsArtikel.getMaxPreis()

objArtikel02.getMaxPreis()
```

- Der Methodenname wird für den Aufruf genutzt.
- Die Klassenmethode kann über den Klassennamen oder eine Instanz aufgerufen werden.
- Argumente werden entsprechend der Signatur übergeben. Ein Argument für den Parameter `cls` wird nicht benötigt.

Zuordnung der Parameter beim Aufruf

- Die Argumente im Aufruf werden den Parametern der Methode standardmäßig in Abhängigkeit der Position zugeordnet.
- Eine Schlüssel-Wert-Zuweisung unabhängig von der Position der Parameter ist möglich.

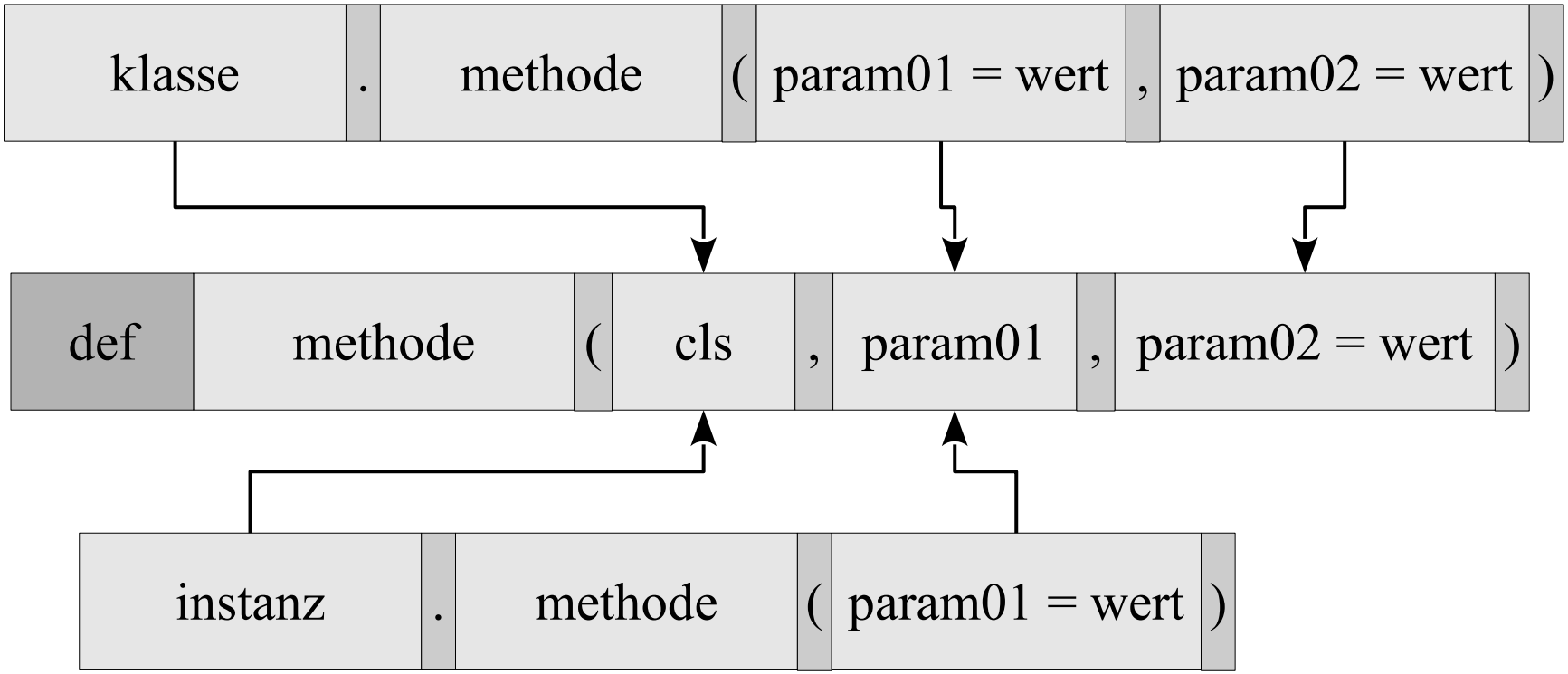
Positionsargumente (positional argument)



Erläuterung

- Die Argumente werden den Parametern entsprechend der Reihenfolge zugeordnet.
- Die Zuordnung der Argumente zu den Parametern erfolgt von links nach rechts.
- Dem ersten Parameter (nach `cls`) wird das erste Argument in der Liste übergeben und so weiter.
- Der Parameter `cls` wird automatisiert durch die Angabe des Klassennamens gesetzt. Andere Möglichkeit: In Abhängigkeit der Instanz wird die Klasse ermittelt.

Schlüsselwortargumente (keyword argument)



Erläuterung

- Als Schlüssel wird ein beliebiger Name eines Parameters aus dem Methodenkopf genutzt.
- Der Zuweisungsoperator weist dem Schlüssel einen beliebigen Wert zu.
- Unabhängig von der Reihenfolge können die Argumente den Parametern zugeordnet werden. Die Reihenfolge der Schlüssel-Wert-Paare spielt keine Rolle.

Schlüsselwörter

kategorie	.	newArtikel	(			artikel = "A"	,	menge = 1	)
	def	newArtikel	(	cls	,	artikel	,	menge	)

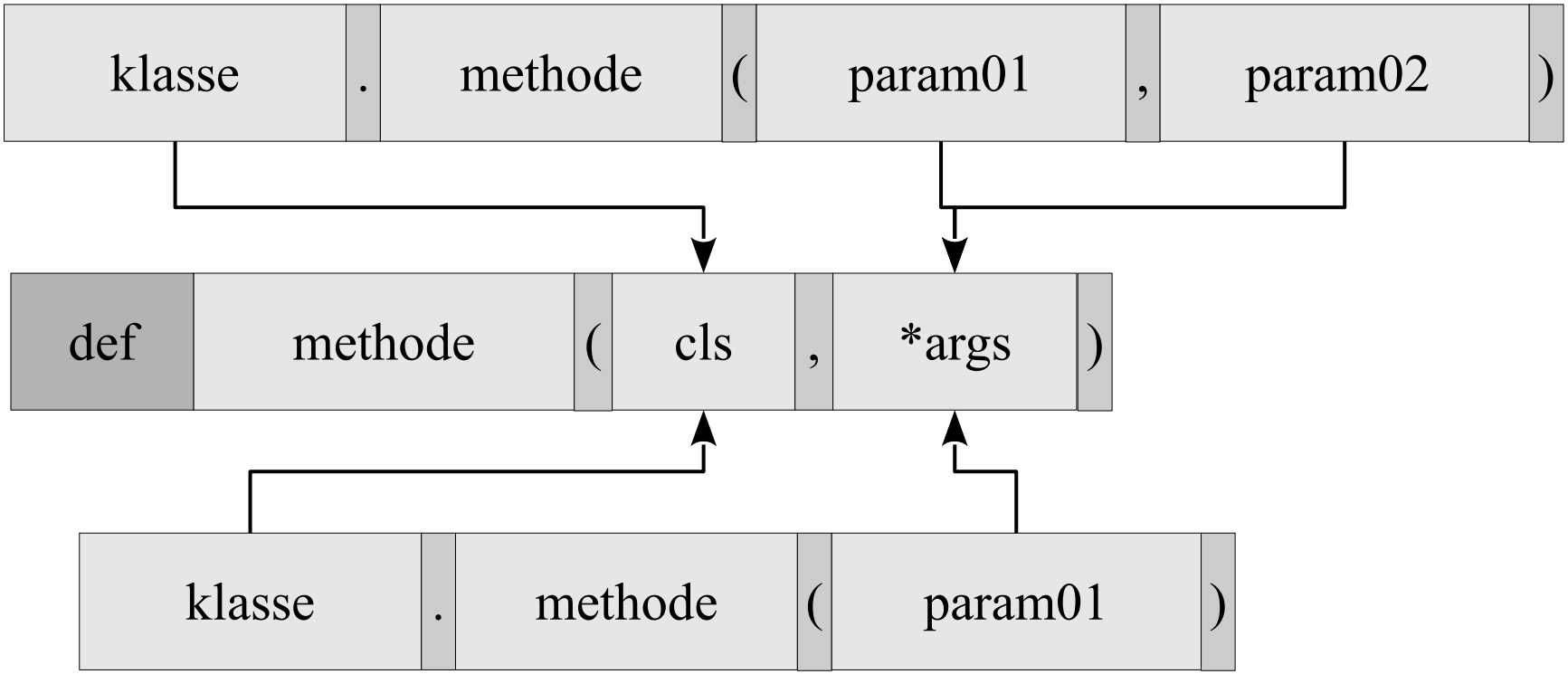
- Die Parameternamen in der Signatur einer Methode werden als Schlüssel genutzt.
- Die Groß- und Kleinschreibung muss bei der Nutzung beachtet werden.

Wert eines Schlüssels

kategorie	.	newArtikel	(			artikel = "A"	,	menge = 1	)
	def	newArtikel	(	cls	,	artikel	,	menge	)

- Mit Hilfe des Zuweisungsoperators wird beim Aufruf dem Schlüssel ein Wert zugewiesen.
- Optionalen Parametern kann ein Wert zugewiesen werden, muss aber nicht.

Übergabe von beliebig vielen Werten



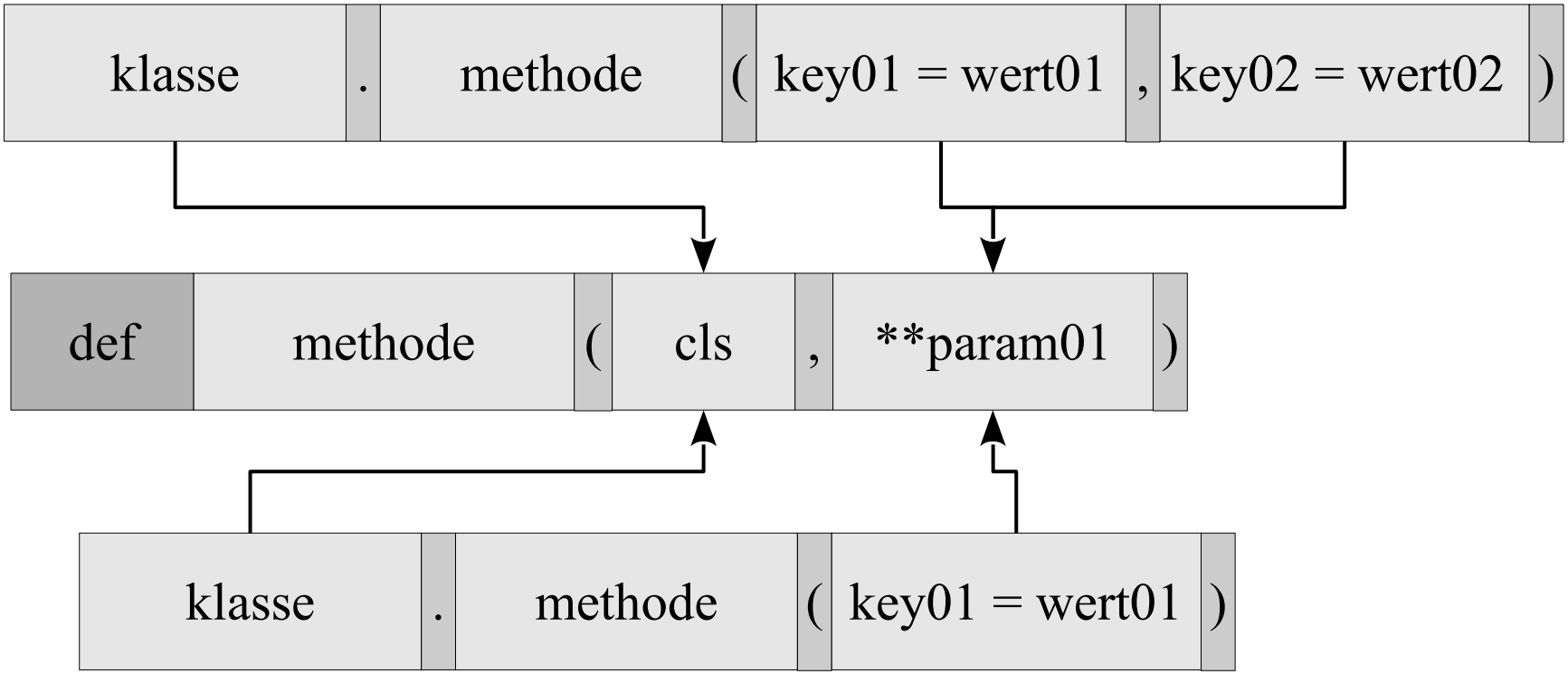
Erläuterung

- Direkt vor dem Namen des Parameters wird ein Sternchen gesetzt. Das Sternchen kennzeichnet eine variable Liste von Werten.
- Der Parameter *args ist ein Platzhalter für Tupel.
- Mit Hilfe einer for-Schleife können die Elemente der Liste in der Funktion gelesen werden.

Beispiel

```
class clsTemperatur(object):
    temperatur = list()

    @classmethod
    def get_ListTemperatur(cls):
        return clsTemperatur.temperatur

    @classmethod
    def set_ListTemperatur(cls, *args):
        for celsius in args:
            clsTemperatur.temperatur.append(celsius)

maxTemp = clsTemperatur()
maxTemp.set_ListTemperatur(4.5, 6.5, 4.3)
```

Beliebig viele Schlüssel-Wert-Paare



Erläuterung

- Direkt vor dem Namen des Parameters werden zwei Sternchen gesetzt. Die Sternchen kennzeichnen eine variable Liste von Schlüssel-Wert-Paaren.
- Der Parameter **kwargs ist ein Platzhalter für Dictonary.
- Mit Hilfe einer for-Schleife können die Elemente des Dictonarys in der Funktion gelesen werden.

Beispiel

```
class clsArtikelKategorie(object):
   anzahl_GesamtArtikel = 0

   def __init__(self, kategorie, **kwargs):
      self.warenkategorie = kategorie
      self.artikel = {}
      self.anzahlArtikel = 0

      for key, value in kwargs.items():
         self.artikel[key]  = value
         self.anzahlArtikel += 1

objKategorie01 =
            clsArtikelKategorie("Kategorie 1", Artikel1 = 1.99, Artikel2 = 3.02)
```

Hinweise

- Falls ein Dictonary als Argument genutzt wird, muss der Platzhalter auch mit zwei Sternchen gekennzeichnet werden.
- Der Parameter für ein beliebig langes Dictonary kann einen beliebigen Namen haben.
- Der Parameter **kwargs muss immer nach dem Parameter *args gesetzt werden.

Fabrikmethoden

```
def createDurchschnittsware(cls, bezeichnung):
    instanz = cls(bezeichnung, cls.getDurchschnittspreis())
    return instanz

durchschnittsware = classmethod(getDurchschnittsware)
```

- Konstruktion von Instanzen aus vorhandenen Informationen.
- Die Methode gibt einen Verweis auf eine Instanz zurück.

Erzeugung der Instanz

```
instanz = cls(bezeichnung, cls.getDurchschnittspreis())
```

- Über den Klassennamen wird der Konstruktor aufgerufen.
- In diesem Beispiel wird die Methode __init__ mit den Parametern „Name der Ware“ und Preis aufgerufen.

Aufruf der Fabrikmethode

```
durchschnitt = clsArtikel.durchschnittsware("Ware 3")
```

- Falls die Klassenmethode mit Hilfe der Funktion classmethod() konvertiert wurde, wird der Name der konvertierten Methode genutzt. Falls die Methode entsprechend dekoriert wurde, kann der Name direkt genutzt werden.
- In der Variablen wird ein Verweis auf das neu erzeugte Objekt gespeichert.

Statische Methoden

- Ungebundene Methoden.
- In der Methode werden keine Instanz- oder Klassenvariablen genutzt.

Beispiele

```
def berechneUmfang(radius):
    pi = 3.1415926
    umfang = 2 * pi * radius
    return umfang

umfang = staticmethod(berechneUmfang)

@staticmethod
def berechneFlaeche(radius):
   pi = 3.1415926
   return(pi * (radius * radius))
```

Kopf einer statischen Methode

```
def berechneUmfang(radius):
    pass
```

- Eine statische Methode wird innerhalb einer Klasse definiert.
- Der erste Parameter in der Liste kann ein beliebiger Parameter sein.
- Die Methode ist nicht an eine Klasse (cls) oder Instanz (self) gebunden.

Kennzeichnung von statischen Methoden

```
umfang = staticmethod(berechneUmfang)
```

- Der Funktion staticmethod wird ein Bezeichner einer statischen Methode übergeben.
- Die genannte Methode wird in eine statische Methode konvertiert.
- Die Funktion gibt einen Verweis auf die konvertierte Methode zurück. Der Name der Variablen wird anstatt des tatsächlichen Methodennamens genutzt.

Aufruf der statischen Methode

```
umfang = Geometrie_Kreis.umfang(6)
umfang = kreis01.umfang(kreis01.kreisradius)
```

- Die Methode kann über den Klassennamen sowohl als auch einer Instanz der Klasse aufgerufen werden.
- Die Argumente werden an die Methode in Abhängigkeit der Signatur übergeben.
- Die Methode wird über den erzeugten Verweis auf die konvertierte Methode aufgerufen.

Statischen Methode „dekorieren“

```
@staticmethod
def berechneFlaeche(radius):
    pi = 3.1415926
    return(pi * (radius * radius))
```

- Mit Hilfe des Decorator @staticmethod eine Zeile vor dem Methodenkopf wird eine Methode als statische Methode gekennzeichnet.
- Die Funktionalität der Methode wird um das Attribut „statisch“ erweitert.

Aufruf der statischen Methode

```
flaeche = Geometrie_Kreis.berechneFlaeche(6)
flaeche = kreis01.berechneFlaeche(kreis01.kreisradius)
```

- Die Methode kann über den Klassennamen sowohl als auch einer Instanz der Klasse aufgerufen werden.
- Die Argumente werden an die Methode in Abhängigkeit der Signatur übergeben.
- Die Methode wird über den, in der Signatur angegebenen Namen aufgerufen.