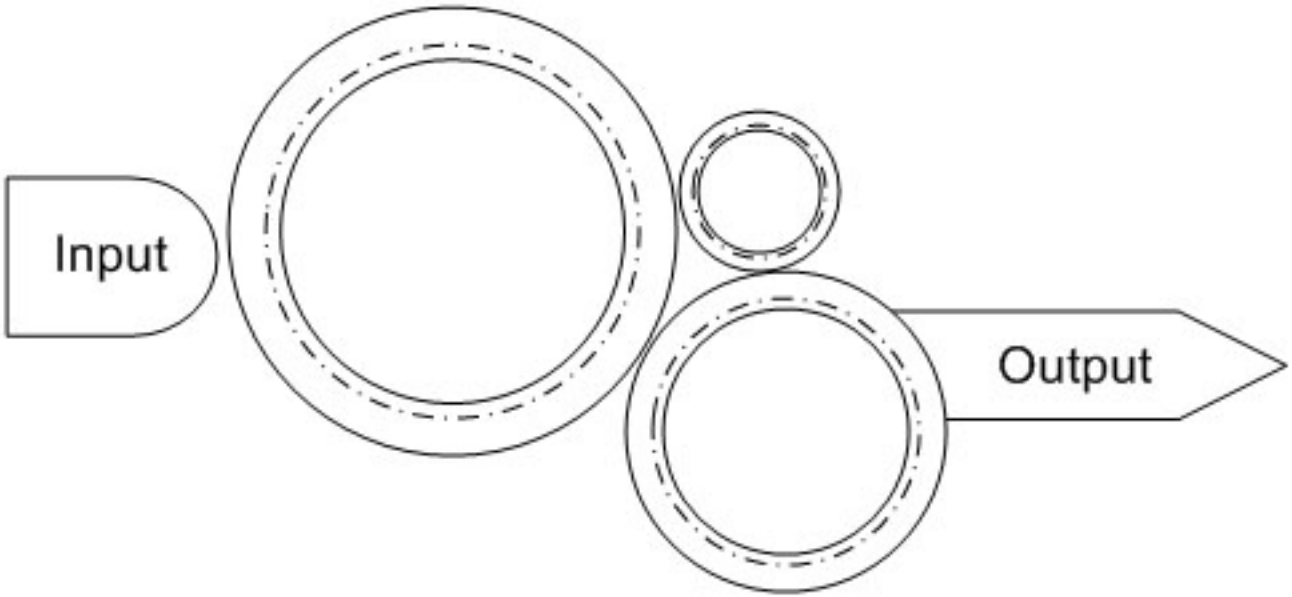
Python
"Objektmethoden“



Methoden

- „Funktionen„ eines Objektes.
- Definition innerhalb einer Klasse.
- Konstruktion eines konkreten Objekts.
- Zugriff auf Attribute eines Objektes.
- Beschreibung des Verhaltens eines Objekts.

… im Klassenrumpf

class clsTemperatur(object):	Klassenkopf
def __init__(self, celsius): self.temperatur = celsius	Initialisierungsmethode

def get_temperatur(self):
return self.temperatur
 	
def celsius_to_fahrenheit(self):
fahrenheit = (1.8 * self.temperatur) + 32
return fahrenheit
	Objekt- und Klassenmethoden

… in einer Projektbeschreibung

- Temperaturen werden in der Einheit „Celsius“ *gespeichert.* Die Temperatur kann in Fahrenheit oder Kelvin *umgerechnet* werden.
- Der Radius eines Kreises wird bei der Erzeugung *gespeichert.* Der Radius kann im Leben des Kreises *vergrößert* oder *verkleinert* werden. Die Fläche und der Umfang des Kreises wird *berechnet*.
- Auf einem Stapel werden Elemente *abgelegt*. Das oberste Elemente kann vom Stapel *entfernt* werden. Die Anzahl der Elemente auf dem Stapel kann *erfragt* werden.

Erläuterung

- Verben in einem Text beschreiben eine Handlung.
- Alle benötigen Handlungen eines Objekts werden in einer Klasse definiert.
- Die Handlung wird in einer Methode beschrieben.
- In einer Methode werden die einzelnen Schritte einer Handlung mit Hilfe von Anweisungen in der Programmiersprache Python beschrieben.

Objektmethoden in Python

```
class clsTemperatur(object):

    def get_temperatur(self):
        return self.temperatur

    def set_temperatur(self, celsius):
        self.temperatur = celsius

    def celsius_to_fahrenheit(self):
        fahrenheit = (1.8  * self.temperatur) + 32
        return fahrenheit
```

Aufbau einer Methode

def celsius_to_fahrenheit(self):	Methodenkopf
fahrenheit = (1.8 * self._temperatur) + 32 return fahrenheit	Methodenrumpf

Methodenrumpf

- Implementierung des Verhaltens eines Objektes.
- Beschreibung der Handlung in einzelnen Schritten
- Zusammenfassung von Anweisungen unter einem Label.
- Die Anweisungen im Rumpf werden entsprechend der Einrückung einem Methodenkopf (einer Methode) zugeordnet.

Leerer Methodenrumpf

```
def celsius_to_fahrenheit(self):
    pass
```

- Das Schlüsselwort pass kennzeichnet einen leeren Methodenrumpf.
- Eine Implementierung der Methode findet später statt.

Methodenkopf

- Signatur einer Methode.
- Beschreibung einer Schnittstelle nach außen.
- Der Nutzer einer Methode kennt nur den Kopf. Wie soll die Methode aufgerufen werden?
- Der Methodenkopf wird entsprechend der Einrückung einer Klasse zugeordnet. Falls der Methodenkopf und die Klasse die selbe Einrücktiefe haben, wird die Methode zu einer Funktion.

Aufbau in Python

- 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	set_temperatur	(	self	,	celsius					)
def	methode	(	self	,	par01	,	par02	,	...	)
def	get_temperatur	(	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 einer Methode ist in einer Klasse eindeutig.

Konventionen

- Falls der Name sich aus mehreren Wörtern zusammensetzt, werden diese durch den Unterstrich getrennt. Andere Möglichkeit: Nutzung der Kamel-Notation.
- Methoden, die mit „get“ beginnen, definieren einen lesenden Zugriff auf ein Attribut.
- Methoden, die mit „set“ beginnen, verändern den Wert eines Attributs.

Parameter in Methoden

```
class ClsRechteck(object)::
    def setGroesse(self, breite, hoehe = None):
        if (breite is not None) and (hoehe is None):
            self.hoehe = breite
            self.breite = breite

        elif (breite is None) and (hoehe is not None):
            self.hoehe = hoehe
            self.breite = hoehe

        elif (breite is not None) and (hoehe is not None):
            self.hoehe = hoehe
            self.breite = breite
```

Zusammenfassung in einer Parameterliste

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

- Die Parameterliste beginnt und endet mit den runden Klammern.
- Die Parameterliste hat als Element mindestens self. Der Parameter verweist auf den Aufrufer der Methode.
- Die Parameter in der Liste werden durch ein Komma getrennt.

Parameter

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

- Parameter sind Variablen, die in der Methode in irgendeiner Form weiterverarbeitet werden.
- Setzen von Attributen eines Objektes.
- 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 self

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

- Der erste Parameter einer Methode muss immer `self` sein.
- Wer hat die Methode aufgerufen?
- Die Anweisung `self.attribut = ausdruck` verändert Attribute des an die Methode gebundenen Objekts.
- Der Parameter verweist auf die aktuelle Instanz.

Instanz „self“

```
def set_temperatur(self, celsius):
    self.temperatur = celsius
```

- Die Instanz self ruft eine Methode auf. Die Methode ist einer Klasse definiert. Die aktuelle Instanz ist von dieser Klasse.
- Der Name self ist eine Konvention für die aktuelle Instanz. Es kann aber jeder beliebige Name genutzt werden.

Nicht optionale Parameter

def	setGroesse	(	self	,	breite	,	hoehe	)

- Parameter, die für den Ablauf der Handlung unbedingt benötigt werden.
- Jedem nicht optionalen Parameter muss beim Aufruf der Methode ein Wert übergeben werden.

Optionale Parameter

def	setGroesse	(	self	,	breite	,	hoehe = None	)

- Parameter, deren Wert für viele Instanzen von der Klasse gleich sind. Parameter, die nicht immer für die Ausführung der Handlung benötigt werden.
- Definition als Schlüssel-Wert-Paare in der Form `key = value`.
- Der Name des Parameters ist in der Liste eindeutig. Der Name wird als Schlüssel für einen beliebigen Wert genutzt.
- Mit Hilfe des Zuweisungsoperator wird einem Schlüssel ein Standardwert zugewiesen.

Hinweise

- Optionalen Parametern kann ein Wert beim Aufruf zugewiesen werden, muss aber nicht.
- Optionale Parameter stehen immer am Ende der Parameterliste.
- Optionalen Parametern folgen keine nicht optionalen Parameter.

Lebenszyklus eines Objekts

… erzeugen
und initialisieren:

```
celsius = clsTemperatur()
```

… arbeiten:
Methoden aufrufen

```
celsius.get_temperatur()
celsius.celsius_to_xxx('k')
celsius.set_temperatur(1.2)
```

… zerstören:

… erzeugen

```
class clsTemperatur(object):
    pass
```

```
>>> celsius = clsTemperatur()
```

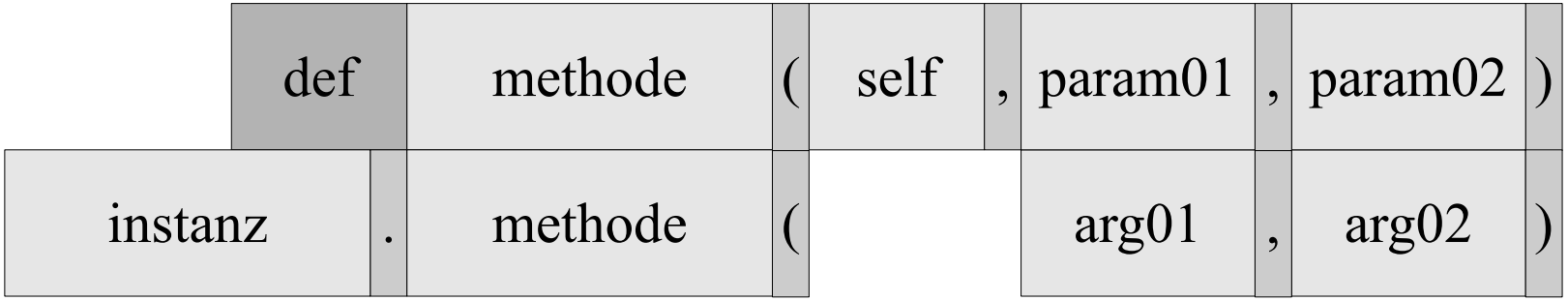
- *Run – Run Module* startet das Modul.
- An der Einfügemarke in der Shell wird folgende Anweisung eingegeben: `celsius = clsTemperatur()`.
- Von dem Bauplan „Temperatur“ wird das konkrete Objekt `celsius` erzeugt. Eine Instanz von der Klasse `clsTemperatur` wird erzeugt.

Beispiele für Methodenaufrufe

	def	set_temperatur	(	self	,	celsius	)
celsius	.	set_temperatur	(			1.2	)

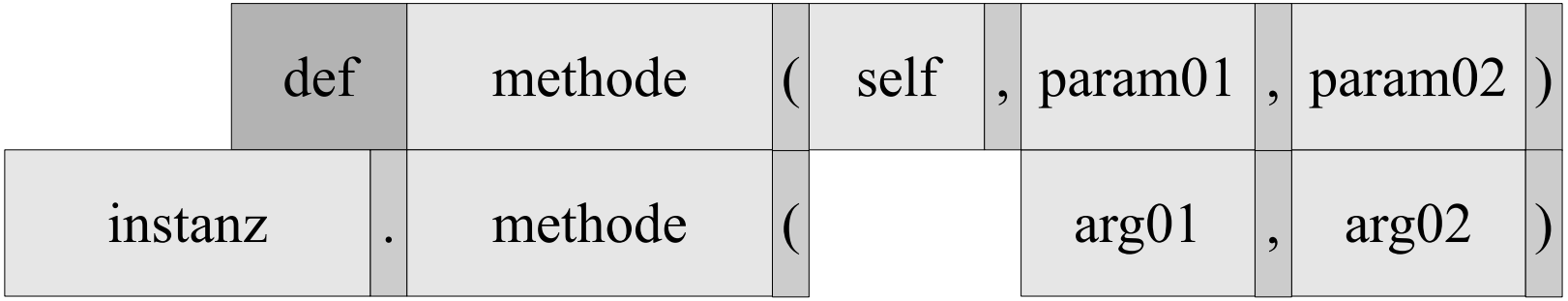
	def	get_temperatur	(	self	)
celsius	.	get_temperatur	(		)

Erläuterung



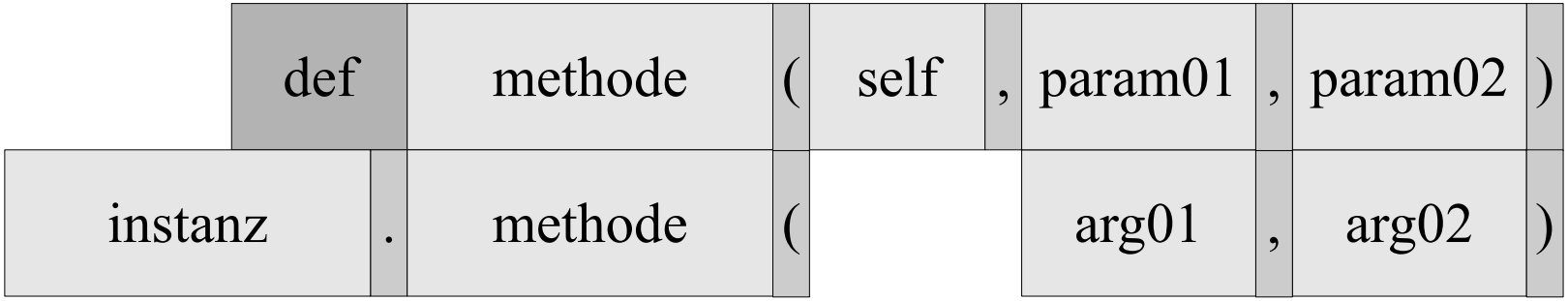
- Mit Hilfe des Namens in der Signatur wird eine Methode aufgerufen.
- Der Aufruf entspricht dem Methodenkopf.
- Der Aufruf der Methode folgt immer über eine Instanz.

Die Instanz



- Vor Aufruf der Methode muss die Instanz deklariert sein.
- Die Instanz, die die Methode aufruft, steht links vom Punkt-Operator.
- Eine Instanz basiert auf einer Klasse. In dieser Klasse ist die gewünschte Methode definiert. Andernfalls wird der Laufzeitfehler *AttributeError: ... object has no attribute ...* oder TypeError angezeigt.

… ruft die Methode auf

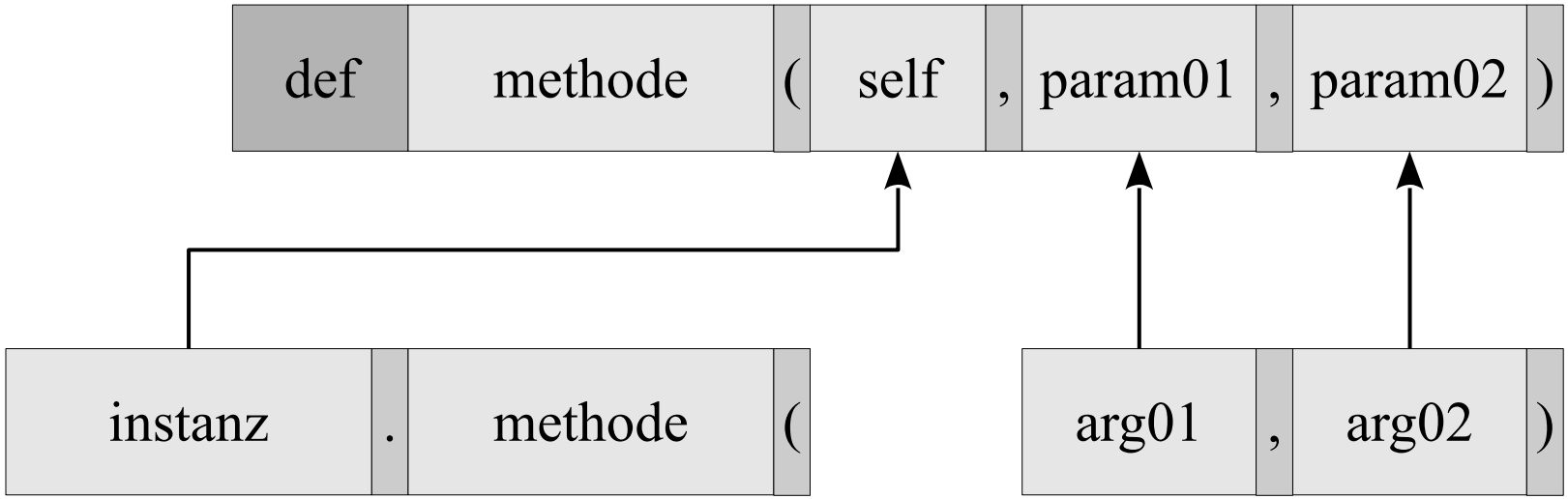


- Der Aufruf der Methode entspricht dem Methodenkopf.
- Die Methode ist in einer Klasse definiert. Die Klasse, in der die Methode definiert ist, wird durch die Instanz festgelegt.
- Der Methodennamen wird rechts vom Punkt-Operator angegeben. Dem Methodennamen folgt die Parameterliste.

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.

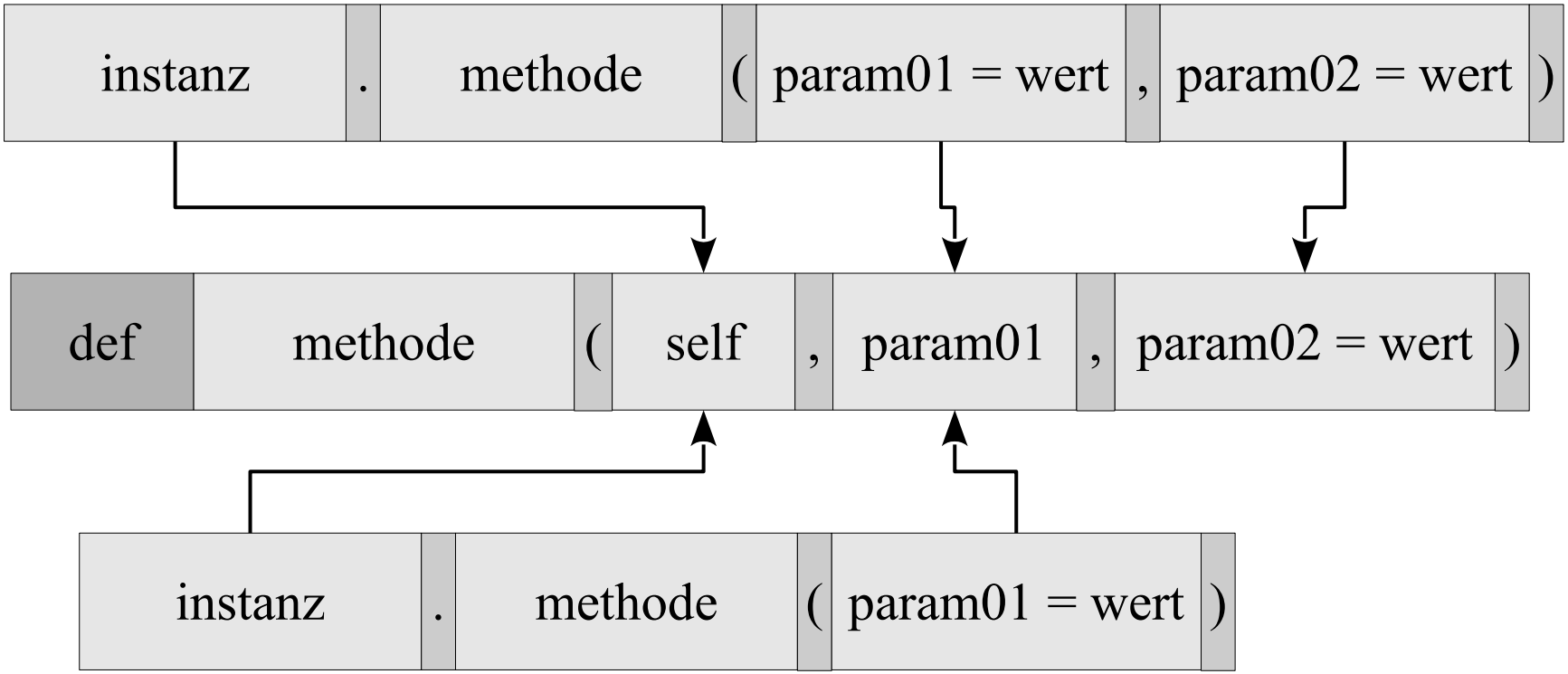
Positionsargumente (positional argument)



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 `self`) wird das erste Argument in der Liste übergeben und so weiter.

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

rechteck	.	setGroesse	(			breite = 3	,	hoehe = 1	)
	def	setGroesse	(	self	,	breite	,	hoehe = None	)

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

Wert eines Schlüssels

rechteck	.	setGroesse	(			breite = 3	,	hoehe = 1	)
	def	setGroesse	(	self	,	breite	,	hoehe = None	)

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

Beispiele

def	setGroesse	(	self	,	breite	,	hoehe = None	)

rechteck	.	setGroesse	(	breite = 3	,	hoehe = 1	)
rechteck	.	setGroesse	(	hoehe = 1	,	breite = 3	)
rechteck	.	setGroesse	(	breite = 3	)		

Rückgabewert einer Methode

```
def get_temperatur(self):
    return self.temperatur

def set_temperatur(self, celsius):
    self.temperatur = celsius
```

Befehl return

- Mit Hilfe des Befehls return kann ein Wert an den Aufrufer zurückgegeben werden.
- Rückgabe von Variablen, Instanzen etc.
- Die Methode endet mit dem Befehl return. Alle nachfolgenden Anweisungen in der Methode werden nicht mehr ausgeführt.

Standardrückgabewert einer Methode

```
return None
```

- Jede Methode in Python gibt den Wert `None` zurück.
- Wenn nichts mit Hilfe von `return` zurückgegeben wird, wird automatisiert `None` zurückgegeben.

Rückgabe eines beliebigen Wertes

```
return(self.breite * self.hoehe)
return (strBreite, strHoehe)
return fahrenheit
```

- Es können Zahlen, boolsche Werte, Strings, Listen, Tupel etc. zurückgegeben werden.
- In Abhängigkeit von Bedingungen können verschiedene Werte an den Aufrufer zurückgegeben werden.

Speicherung des Rückgabewertes

```
>>> flaeche = rechteck.getFlaeche()
```

- Die Methode wird entsprechend der Signatur aufgerufen.
- Der Rückgabewert der Methode wird einer Variablen mit Hilfe des Gleichheitszeichen zugewiesen.
- Der Wert der Variablen kann weiterverarbeitet werden.

Zugriff auf Methoden

In Python sind Methoden immer öffentlich. Auf Methoden in einer Klasse kann von außen her zugegriffen werden.

- Der gewünschte Zugriff auf eine Methode kann nur im Code gekennzeichnet werden. Die Kennzeichnung ist aber keine Einschränkung des Zugriffs für den Interpreter von Python.

Private Methoden

```
def __getTemperatur(self):
        return self._temperatur
```

- Als Präfix werden zwei Unterstriche genutzt.
- Die Methode sollte nur in der Klasse aufgerufen werden.

Geschützte Methoden

```
def _getTemperatur(self):
        return self._temperatur
```

- Als Präfix wird ein Unterstrich genutzt.
- Die Methode kann von einer Klasse, die von dieser erbt genutzt werden.
- Von außen sollte die Methode aber nicht aufgerufen werden.
- Die Methode sollte von einer Kind-Klasse nicht überschrieben werden.

Kennzeichnung von eingebauten Methoden

```
def __init__(self, celsius):
        self._temperatur = celsius
```

- Methoden, die mit zwei Unterstrichen beginnen und enden, sind, von Python vordefinierte Methoden.
- Der Name der Methode darf nicht verändert werden.
- Aber die Methode kann überschrieben werden. Die vordefinierte Methode wird den Bedürfnissen der benutzerdefinierten Klasse angepasst.
- Vordefinierte Methoden sollten möglichst am Anfang einer Klasse überschrieben werden.

Initialisierungsmethode in einer Klasse

```
def __init__(self, celsius):
        self._temperatur = celsius
```

- Initialisierungsmethoden haben immer den Namen `__init__`.
- Durch den Aufruf `instanz = klasse()` wird die Initialisierungsmethode der Klasse aufgerufen. Falls diese nicht vorhanden ist, wird die passende Methode der Eltern `object` aufgerufen.
- Jede Klasse in Python hat exakt eine Initialisierungsmethode.

Beispiele

```
def __init__(self, celsius = 1):
    self.temperatur = celsius
```

```
def __init__(self):
    self.breite = 1
    self.hoehe = 1
```

Bezeichner der Initialisierungsmethode

- Der Name ist immer `__init__`.
- Die Methode beginnt und endet mit zwei Unterstrichen.
- Die Groß- und Kleinschreibung wird beachtet. Der Name `init` beginnt immer mit einem Kleinbuchstaben.

Aufruf der Initialisierungsmethode

- Der Aufruf erfolgt immer automatisch.
- Sobald eine Instanz erzeugt wurde, wird die Methode __init__ aufgerufen.

Kopf der Initialisierungsmethode

def	__init__	(	self	,	par01	,	par02	,	...	)
def	__init__	(	self							)

- Die Parameterliste wird durch die runden Klammern begrenzt.
- Die Initialisierungsmethode hat mindestens den Parameter self.
- Dem Parameter self können beliebig viele Parameter folgen.

Nicht optionale Parameter

def	__init__	(	self	,	par01	,	par02	,	...	)
def	__init__	(	self							)

- Parameter für Attribute, in denen die Instanzen sich sehr stark unterscheiden.
- Setzen von Attributen, die für das Leben der Instanz essentiell notwendig sind.

Optionale Parameter

def	__init__	(	self	,	par01	,	par02	,	...	)
def	__init__	(	self							)

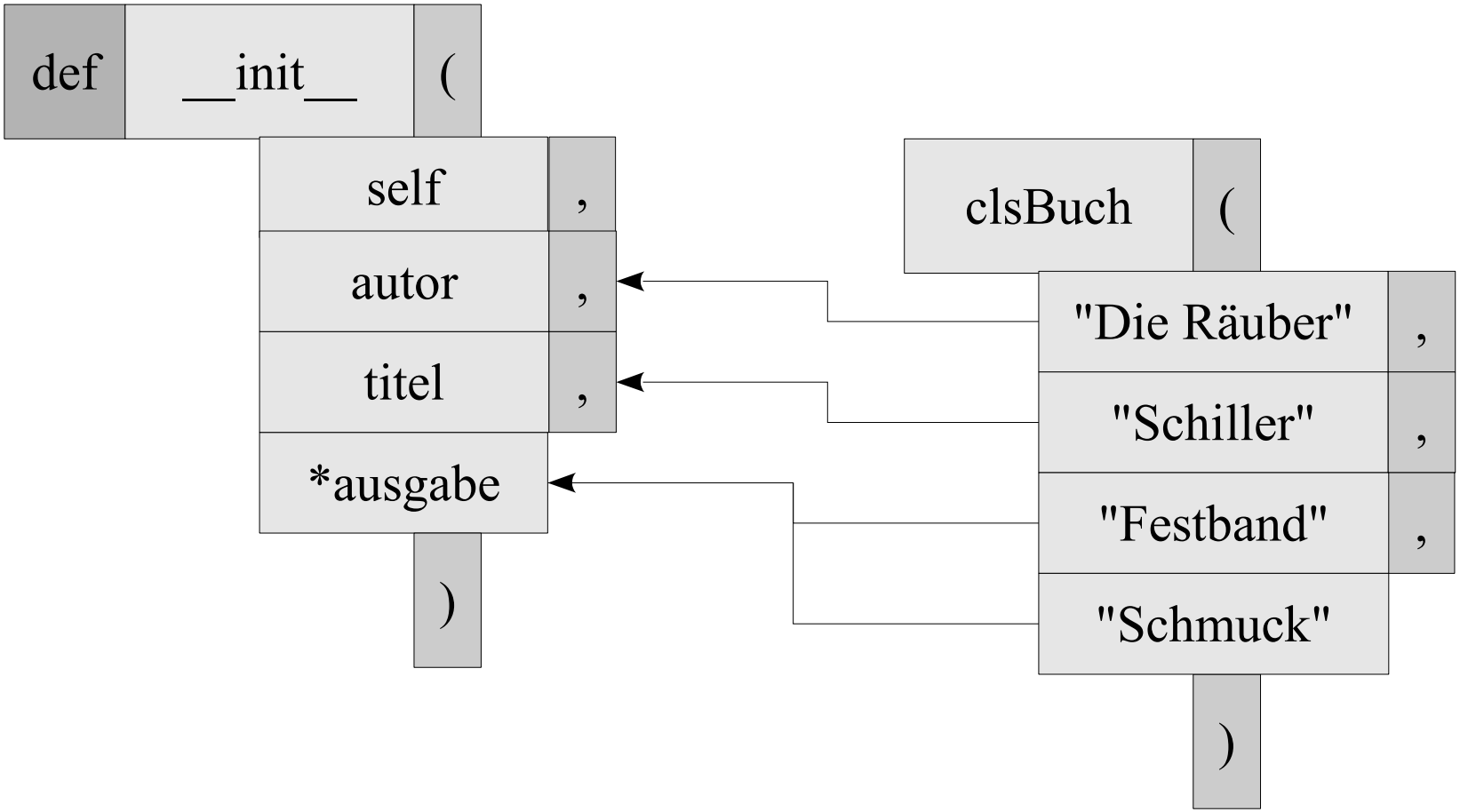
- Setzen von Attributen, die für die späteren Handlung nicht unbedingt notwendig sind.
- Attribute, deren Werte für viele Instanzen gleich sind.

Übergabe von beliebig vielen Werten

```
class clsBuch(object):

    def __init__(self, autor, titel, *ausgabe):
        self.autor  = autor
        self.titel = titel
        self.ausgabe = ausgabe
```

Aufruf


def
__init__
(
self
,
autor
,
titel
,
*ausgabe
)
clsBuch
(
"Die Räuber"
,
"Schiller"
,
"Festband"
,
"Schmuck"
)

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.

Hinweise

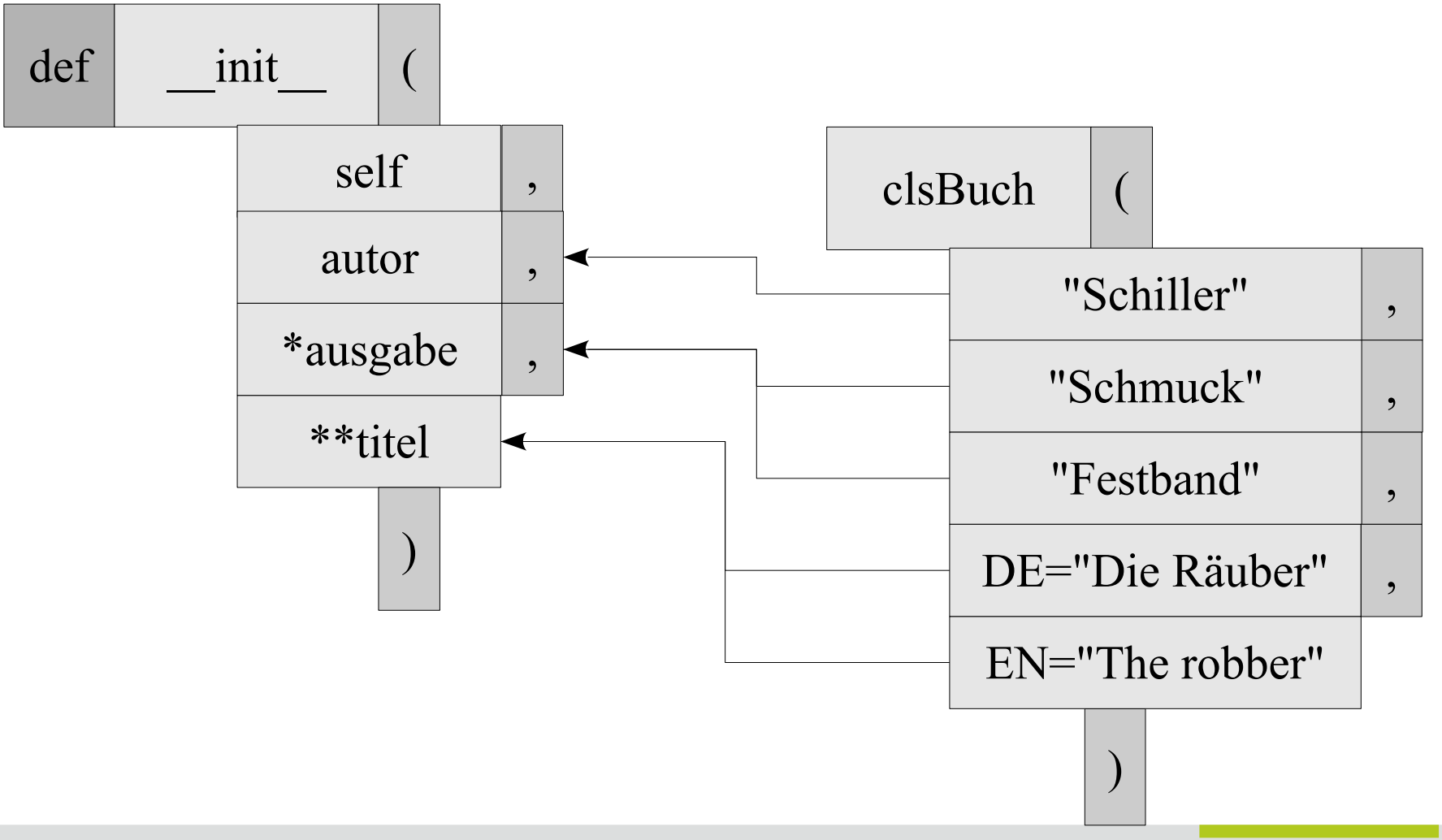
- Falls ein Tupel als Argument genutzt wird, muss der Platzhalter auch mit ein Sternchen gekennzeichnet werden.
- Der Name des Parameters ist frei wählbar.

… von beliebig vielen Schlüssel-Wert-Paaren

```
class clsBuch(object):

    def __init__(self, autor,  *ausgabe, **titel,):
        self.autor  = autor
        self.titel = titel
        self.ausgabe = ausgabe
```

Aufruf


def
__init__
(
self
,
autor
,
*ausgabe
,
**titel
)
clsBuch
(
"Schiller"
,
"Schmuck"
,
"Festband"
,
DE="Die Räuber"
,
EN="The robber"
)

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.

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.

Zerstörung von Objekten

```
def __del__(self):
    pass
```

- Die Methode hat immer den Namen `__del__`.
- Aufruf durch die Anweisung `del instanz`.
- Die Methode enthält Anweisungen, die unbedingt vor der Zerstörung des Objektes ausgeführt werden müssen. In dieser Methode können Aufräumarbeiten ausgeführt werden. Zum Beispiel wird eine geöffnete Datei geschlossen.

Docstrings

```
""" Größe eines Rechtecks lesen

        Parametes:
            self : object

        Returns
            clsrechteck_object.getGroesse() -> string
            breite + ' x ' + hoehe
"""
```

Erläuterung

- Docstrings dokumentieren das Verhalten einer Methode. Der Parameter und der Rückgabewert der Methode werden beschrieben.
- Docstrings sind spezielle Kommentare. Die Kommentare beginnen und enden mit drei Anführungszeichen.
- Mit Hilfe des Befehls help(klassenname) kann die Dokumentation zu einer Klasse angezeigt werden. Falls vorhanden, werden zu den Methoden die Informationen aus den Docstring in den Hilfeseiten angezeigt.