Cheatography

Objektorientierte Programmierung mit Python Cheat Sheet

by ch1phorst (ch1phorst) via cheatography.com/200234/cs/42326/

Beispiel: Klasse

```
# Klassendefinition
class Human:
       # Konstr uktor (Initi ali sie rung)
       def __init __( self, name, age):
                sel f.name = name
                sel f.age = age

       # Methode zum Vorstellen
       def __str_ _(s elf):
               s = "Ich heiße " + self.name +
".\n "
                s += "Ich bin " + str(se lf.age) + "
Jahre alt.\n "
                return s

       # setter -Me thode zum Ändern des Attrib -
utw ertes von age
       def set_ag e(self, new_age):
                sel f.age = new_age
# Deklar ation und Initia lis ierung einer
Human- Instanz
h1 = Human( "Max Müller ", 21)
# Ändern der Wertbe legung des Attributes age vom
Objekt h1
h1.set _ag e(22)
# __str_ _-M ethode wird aufgerufen
print(h1)
```

Exkurs: Python Dataclasses

```
class Robot_traditional:
       def __init __( self, model, serial _nu mber,
manufa ctu rer):
                sel f.model = model
                sel f.s eri al_ number = serial -
_number
                sel f.m anu fac turer = manufa -
cturer
# versus
from datacl asses import dataclass
@dataclass
class Robot:
        model: str
        ser ial _nu mber: str
        man ufa cturer: str
```

Magic Methods einer Klasse

Operator/Aufruf	Magic Method
obj = Klasse()	__new__ und __init__
+	object.__add__(self, other)
-	object.__sub__(self, other)
str(object) oder print(object)	object.__str__(self)
<	object.__lt__(self, other)
==	object.__eq__(self, other)
>=	object.__ge__(self, other)

Die Auflistung zeigt nur eine Auswahl der zur Verfügung stehenden Methoden einer Klasse. Die gesamte Liste (eines Scopes) erhält man beim Aufruf von dir(object).

Hintergrund: Magic Methods

Der Mechanismus: Ein Ausdruck "x + y" mit x und y als Instanzen der Klasse K, wird so ausgewertet: der Interpreter prüft die Klasse-ndefinition von K. Wenn K eine Methode __add__ hat, so wird diese aufgerufen mit x .__ add __ (y).

Type Hints

```
class Position:
       def __init __( self, x: int, y: int) ->
None:
                self.x = x
                self.y = y
       def __add_ _(self, other: Position) ->
Position:
                return Positi on( self.x +
other.x, self.y + other.y)
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Kernkonzepte OOP

Datenkapselung	Attributen und Methoden können unterschiedliche Sichtbarkeitsniveaus zugewiesen werden (+ public, o protected, - private)
Vererbung	aus Oberklassen können Unterklassen abgeleitet werden, die zunächst gleiche Attribute und Methoden besitzen
Polymorphie (Vielgestaltigkeit)	zusätzlich zu vererbten Attributen und Methoden existieren weitere; vererbte Methoden können überladen werden; Objekte können bei Instanziierung unterschiedliche Strukturen annehmen

ch1ph0rst

By **ch1ph0rst** (ch1ph0rst)
cheatography.com/ch1ph0rst/

Not published yet.
Last updated 25th March, 2026.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com