# Einführung in die objektorientierte Programmierung

Grundlagen

# Was ist ein Objekt?

Die Grundidee von OOP ist, dass wir Objekte verwenden, um Dinge aus der realen Welt zu modellieren, die wir in unseren Programmen abbilden.

Beispiel:

Wir können eine Person als Objekt betrachten. Sie hat einen Namen, ein Alter, eine Haarfarbe usw.

Weitere Objekte: Konto, Webshop, Auto, geometrische Figuren, Meßreihen, Möbel...

# Beispiele für objektorientierte Sprachen

Simula (1967)

Smalltalk (1972)

Common Lisp

C++

Java

C#

Python

# Und was ist ein Objekt?

Ein Objekt ist im Grunde eine Sammlung von Daten (Variablen) und Methoden (Funktionen), die auf diesen Daten operieren. Eine Klasse ist eine Blaupause (Blueprint) für dieses Objekt.

Beispiel: Haus

Eine Klasse ist eine Art Vorlage oder Blaupause für ein Haus. Es beinhaltet alle Werte und Methoden, welche für den Bau eines Hauses benötigt werden.

Auf Grundlage dieser Vorlage bauen wir ein Haus. Das erbaute Haus ist ein Objekt der Klasse Haus. Wollen wir ein anderes Haus bauen, bauen wir wieder auf Grundlage dieser Vorlage das Haus.

Ein Haus wird auch Objekt genannt. Dieses Objekt wird auch als Klasseninstanz der Klasse Haus bezeichnet.

# Beispiel: Festlegen einer Klasse

In Python definieren wir eine Klasse mit dem Schlüsselwort class.
Der Name der Klasse wird **in Pascal-Case (Upper Camel Case)** notiert.

```
class MyNewClass:
    '''Das ist der Doc-String der Klasse MyNewClass'''
     pass
```

Unsere Klasse Haus würde so aussehen:

```
class Haus:
    '''das ist die Klasse Haus. Sie kann bisher noch nicht viel'''
    pass
```

# Instanz einer Klasse

Um eine Klasse auch zu nutzen, müssen wir eine Klasseninstanz davon erzeugen. Diese Klasseninstanz nennt man auch Objekt. Eine neue Klasseninstanz wird wie folgt erzeugt:

```
small_house = House()
```

Um zu prüfen, ob eine Klasseninstanz einer speziellen Klasse zugehörig ist, kann man das mit der build-in Funktion isinstance prüfen.

```
isinstance(small_house, House)
```

=> Liefert True zurück, wenn small_house ein Objekt/Klasseninstanz der Klasse House ist.

# Wozu Objektorientiert programmieren?

Früh können wir erkennen, dass Fahrrad, Motorrad, Auto, Bus oder Lastwagen eine Gemeinsamkeit haben: Sie sind alle Fortbewegungsmittel.

Im Alltag ordnen wir Birnen, Äpfel, Orangen etc. ganz selbstverständlich der Gruppe Obst zu und Salat und Kohl sind für uns Gemüse.

Beim Programmieren eines Onlineshops arbeiten wir ebenso mit Objekten. Beispielsweise mit Artikeln, Kunden und Farben.

Unser Leben, die Wirklichkeit, ist voller Objekte: Fahrzeuge, Obst, Gemüse, Artikel, Kunden und viele mehr.

# Attribute

**Eine Klasse kann eigene Variablen besitzen. Diese nennt man Attribute oder auch Klassenvariablen.**

```
class Person:
  name = 'Peter'
```

Über das Objekt kann auf diese Attribute zugreifen und sie auch verändern. Um auf Attribute zuzugreifen, nutzen wir die Dot-Notation (Punkt-Syntax).

```
peter = Person()
print(peter.name)
```

# Klassen-Funktionen

**Eine Klasse kann auch eigene Funktionen besitzen. Diese nennt man Methoden oder auch Klassenfunktionen. Methoden in Python haben die Eigenart, dass der erste Parameter in der Parameterliste zwingend die Referenz auf die Instanz ist, von der sie aufgerufen wird.**

```
class Person:
  name = 'Peter'

  def say_hello(self):
        print('say hello')
```

Das Objekt kann auf diese Attribute zugreifen und sie verändern. Um auf Attribute zuzugreifen, nutzen wir die Dot-Notation (Punkt-Syntax).

```
peter = Person()
print(peter.say_hello())
```

# Zugriff auf Attribute der Klasse

Über das Keyword self können wir auf die Variablen unseres Objektes zugreifen. Über Methoden können wir eigene Werte in die Instanz überführen. Wichtig: der Parametername und der Attribut-Name der Klasse stehen in keinem Zusammenhang.

self bezieht sich auf die jeweilige Objektinstanz

```
class Animal():
    name = None

    def set_name(self, n):
        self.name = n

tier1 = Animal()
tier2 = Animal()
tier1.name = 'Fifi'
tier2.name = 'Herakles'
```

# Initialisieren von Objekten

In der objektorientierten Programmierung gibt es das Konzept der Konstruktoren (constructors). Diese helfen beim Bau der Instanz und ermöglichen die Übergabe von Werten bei der Instanziierung. In Python kann das Objekt über die magische Methode __init__() initialisiert werden.

Die __init__() Methode hat keinen Rückgabewert.

Class Person:

```
  def __init__(self, name: str, age: int, hair_color: str) -> None:
    self.name = name
    self.age = age
    self.hair_color = hair_color
```