

# Einführung in die Objektorientierte Programmierung (OOP)

## Inhalt

<b>Inhalte der Folien</b> .....	1
Grundlagen der Objektorientierung.....	1
Vorteile von OOP in der Softwareentwicklung .....	2
Umsetzung in Python .....	2
Objektorientierter Softwareentwurf.....	2
Weitere Impulse .....	3

## Inhalte der Folien

Diese Abschnitte beziehen sich auf die Inhalte der Folien und enthalten die gebündelten Informationen.

### Grundlagen der Objektorientierung

Die Realität wird durch ein objektorientiertes Modell abgebildet. Hierbei findet eine **Abstraktion** statt. Ein Objekt kann einen Zustand und ein Verhalten haben.

Die folgende Tabelle versucht einen **Erklärungsversuch des Paradigmenwechsels** von klassischer, imperativer Programmierung hin zu objektorientierter Programmierung:

(Imperative) Programmierung	Objektorientierte Programmierung
Strukturierung primär durch Unterprogramme und Aufrufe	Verhalten und Zustand wird zu <b>Objekten</b> zusammengefasst
Trennung von Daten/Funktionen	<b>Kapselung</b> von Daten und Funktionen für Objektspezifika möglich
Der Programmablauf wird zentral gesteuert	Kommunikation zwischen den Objekten erlaubt eine erweiterte Steuerung / Programmfluss

Kurze Erklärungen für **Grundbegriffe** des OOP (*Vorsicht: keine allgemeingültige Definition*):

- **Objekt.** Konzept für eine Repräsentation
- **Klasse.** Bauplan für Objekte

- **Konstruktor.** Methode zur Erzeugung von neuen Objekten
- **Instanz.** Adresse eines Objektes im Arbeitsspeicher
- **Methode.** Verhalten eines Objekts
- **Attribute.** Informationen eines Objekts
- **Vererbung.** Hierarchie zwischen Objekten
- **Komposition.** Beziehung zwischen Objekten

## Vorteile von OOP in der Softwareentwicklung

Die folgenden Aspekte sind Gründe, warum OOP aktuell ein Standard in der Softwareentwicklung ist.

- **Modularität.** Objekte lassen sich im Quellcode gut trennen und oftmals unabhängig voneinander entwickeln.
- **Informationsverdeckung.** Die Implementierung lässt sich nach außen verbergen und nur in Schnittstellen preisgeben.
- **Wiederverwendung.** Existierende Objekte können in neue Programme übernommen werden.
- **Wartbarkeit.** Fehlerhafte Objekte lassen sich einfach ersetzen oder austauschen.

## Umsetzung in Python

Die Grundbegriffe von OOP werden in Python folgendermaßen umgesetzt:

- Eine Klasse wird durch das Schlüsselwort **class** angegeben.
- Der Konstruktor hat die Form **def \_\_init\_\_(self)**
- Eine Instanz wird mit **instanzname = Klassenname(Parameter)** erzeugt
- Vererbung erfolgt mit dem Konstrukt **class Subklasse(Superklasse)**, wobei von der Superklasse Attribute und Methoden geerbt werden.
  - Methoden und der Konstruktor können überschrieben werden. Im Falle des Konstruktors kann der Konstruktor der Superklasse mit **super(Subklasse, self).\_\_init\_\_()** aufgerufen werden, um dessen Initialisierung beizubehalten.

## Objektorientierter Softwareentwurf

Der Objektorientierte Softwareentwurf lässt sich in drei Phasen einteilen:

- **Phase 1:** Objektorientierte Analyse (OOA)  
Analyse der relevanten Eigenschaften und Verhaltensweisen
- **Phase 2:** Objektorientiertes Design (OOD)  
Modellierung von Eigenschaften und Verhalten
- **Phase 3:** Objektorientierte Programmierung (OOP)  
Umsetzung des Designs in einer Programmiersprache

## Weitere Impulse

- Ein **guter Stil** im Bereich OOP ist schwierig einzuschätzen. Es gibt viele unterschiedliche Einschätzungen, z.B. lieber viele kleine Klassen statt wenige große Klassen einsetzen
- In Python ist **alles ein Objekt**, egal ob Funktionen, Listen oder einfache Zahlen.
- Eigentlich ist die Funktion `__init__` **nicht der Konstruktor**. Es gibt davor schon eine Methode `__new__`, die das Objekt erzeugt. Da man hier allerdings so gut wie nie eingreifen muss, wird von `__init__` als Konstruktor geredet.
- Ein großer Unterschied im Vergleich zu Java, C# oder JavaScript besteht in der Verwendung des **self** Parameters. Bei den anderen Programmiersprachen wird der Zugriff auf die Instanzvariable implizit über `this` geregelt. Python wählt hier den expliziten weg.
- In Python gibt es für Instanzvariablen keine **Sichtbarkeiten**. Das Prinzip lautet: Eigene Verantwortung.
- Methoden mit doppeltem Unterstrich heißen auch **Dunder-Methoden** („double-underscore“) oder **Magic Methods**. Letzterer Name basiert auf der Tatsache, dass der Aufruf im Hintergrund geschieht.
- Die Bündelung von Funktionalitäten in einer Klasse bezeichnet man als **Lokalität**. Dies hat den Vorteil, dass ein Entwickler den richtigen Ort zum Platzieren von Code kennt.
- Objekte können andere Objekte verwenden. Dies ist das Prinzip der **Delegation**.
- Objekte können andere Objekte ersetzen. Dieses Prinzip heißt **Substitution**. Wenn die Objektsignatur gleich ist, kann dies auch ohne Codeanpassung geschehen.

## Literatur

- [https://www.python-kurs.eu/python\\_OOP.php](https://www.python-kurs.eu/python_OOP.php)
- Python Grundlagenbücher. Für eine Auswahl siehe: <https://realpython.com/best-python-books/#best-books-for-learning-python>