OOP mit Python — Kurzfassung

Objektorientierte Programmierung

Die objektorientierte Programmierung ist ein Programmiermethode, bei der die zu verarbeitenden Daten als eigenständige Objekte mit Eigenschaften (Werte) und Methoden (Aktionen) aufgefasst werden.

Vorteile:

- Die Objektdarstellung ermöglichst es, die Programmierung gedanklich enger an die zu lösenden Aufgabe zu binden.

  Beispiel: Ein Geldbezug kann dadurch modelliert werden, indem auf ein Konto-Objekt `kto-1234567` mit der Eigenschaft `kontostand` die Methode `auszahlung(betrag)` angwendet wird.

- Der Programmcode lässt sich besser verstehen und warten.
- Der Programmcode lässt sich (bei sorgfältigem Entwurf) besser erweitern.

Klasse

Eine Klasse ist ein Bauplan für Objekte.

In Python wird eine Klasse mit dem Schlüsselwort

```
class <Klassenname>:

    <Klassenrumpf/Klassendefinition>
```

eingeleitet, wobei die vom Anwendungsprogrammierer selber definierte Klassen nach Konvention mit einem Grossbuchstaben beginnen sollten.

Instanz (=Objekt)

Eine Instanz ist ein zur Laufzeit erzeugtes Objekt einer Klasse.

Instanzvariable (=Objektvariable)

Eine Variable, die zu einer Instanz gehört.

Instanzmethode (=Objektmethode)

Eine Funktion, die zu einer Instanz gehört und die Zugriff auf die individuellen Eigenschaften der Instanz hat.

Konstruktor

Eine spezielle Methode, mit der die Eigenschaften eines Objekts initialisiert werden können. Der Konstrukor liefert eine Referenz auf das neu erstellte Objekt zurück.

In Python wird der Konstruktor duch folgenden Code definiert:

```
def __init__(self, x, y, ...):
    self.x = x
    self.y = y
    ...
```

Klassenvariable

Eine Variable die zu einer Klasse gehört und die unabhängig von einer Instanz denselben Wert hat.

Klassenmethode

Eine Funktion, die zu einer Klasse gehört und unabhängig von einer Instanz ist.

Spezalmethoden in Python

Neben dem Konstruktor gibt es verschiedene Spezialmethoden, die das Hantieren mit Objekten vereinfachen:

`__str__(self)`: definiert was bei `print(<obj>)` geschehen soll

`__add__(self, other)`: definiert was bei `<obj1>+<obj2>` geschehen soll

`__sub__(self, other)`: definiert was bei `<obj1>-<obj2>` geschehen soll

`__mul__(self, other)`: definiert was bei `<obj1>*<obj2>` geschehen soll

usw.

Beispiel

```
class Abc:

    # Klassenvariable
    k = 17

    # Klassenmethode
    def hello():
        Abc.k += 1
        print("Hello World")

    # Konstruktor
    def __init__(self, x=1):
        self.x = x

    # Objektmethode
    def add(self, y):
        self.x += y

    # Spezialmethode
    def __str__(self):
        return "x={0.x}".format(self)

# Objekterzeugung (Instanziierung):
a = Abc(7)
# Anwendung einer Objektmethode:
a.add(5)
# Textdarstellung des Objekts:
print(a) # => x=12
# Ausgabe der Klassenvariable k:
print(Abc.k) # => 17
# Aufruf der Klassenmethode:
Abc.hello() # Hello World
# Ausgabe der Klassenvariable k:
print(Abc.k) # => 18
```