ZUSAMMENFASSUNG Objektorientierte Programmierung

EINFÜHRUNG

Objektorientierung ist vor allem relevant, wenn man komplexe Daten darstellen möchte. Mit ihr hat man die Möglichkeit eigene Datentypen nach Belieben zu kreieren.

Vorgehensweise

1. Überlegen, aus welchen unterschiedlichen Objekten das Programm bestehen soll
2. Diese Objekte modellieren indem man für jedes einen eigenen Datentyp baut
3. Mit der Programmlogik die modellierten Objekte in Beziehung setzen

KLASSEN UND OBJEKTE

Einen Datentyp kann man bauen, indem man eine Klasse definiert. Eine Klasse ist eine Art Bauplan, in dem man festlegt wie ein Objekt auszusehen hat.
Eine Klasse definiert man, indem man das Schlüsselwort "class" verwendet, gefolgt von einem Doppelpunkt. Alles folgende wird wieder eingerückt.

Objekte haben dabei Eigenschaften die auch Attribute genannt werden. Diese werden unter "class" definiert, mit der init-Funktion, die wiederum mit dem Schlüsselwort def startet.
Die init-Funktion wird als __init__ benannt (jeweils zwei Unterstriche) und ihr folgt ein Klammerpaar für die Definition der Parameter. Im Funktionskörper werden dann die Attribute definiert.
Zum Beispiel:

```
class test():
  def __init__(self):
   self.car_brand = None
   self.horse_power = None
   self.color = None
```

Nun kann man eine Instanz von einem Objekt mit einem bestimmten Typ erstellen lassen. Das funktioniert, indem man den Objektnamen wählt, gefolgt von einem „=“ und dem Klassennamen.
Zum Beispiel:

```
car1 = Car()
```

Hiermit erzeugen wir ein Objekt car1 von der Klasse Car.

WICHTIG Immer wenn der Name der Klasse in ein Klammerpaar geschrieben wird, wird der Konstruktor der Klasse aufgerufen.

Der Konstruktor sorgt dafür, dass das Objekt systematisch aufgebaut wird. Im gezeigten Beispiel ist das die __init__ Funktion.

WICHTIG Innerhalb einer Klasse spricht man von Methoden, nicht von Funktionen. Richtig ist es also von der init Methode zu sprechen.

Der self-Parameter muss bei der Definition einer Methode innerhalb einer Klasse immer zuerst angegeben werden.
Innerhalb der Methode init, werden dann die Attribute des Objekts erzeugt und diesen wird zu Beginn jeweils der Wert None zugewiesen.

DER SELF PARAMETER

Möchte man den Inhalt einer Variablen, die man von einem bestimmten Klasse erzeugt hat, mit print() ausgeben lassen, erhält man nicht die Auflistung der Attribute des Objekts.

Stattdessen erhält man:
__main__.Car object 0x7fb55a970880

Es gibt an, dass die Klasse im Modul __main__definiert wurde. Die kryptische Zahl am Ende stellt eine Referenz dar.
Diese zeigt den Speicherplatz im Computer an, denn man speichert die Attribute des Objekts nicht direkt in der Variablen. Die Attribute werden stattdessen an irgendeinem Ort im Speicher gespeichert.
Eine Referenz ist somit eine Adresse, die zum reservierten Speicherplatz des jeweiligen Objekts führt.

WICHTIG Die Speicheradresse ist für jede Variable eines Objekts die gleiche! Wenn man bei einer Variablen ein Attribut ändert, ändert es sich auch für alle anderen Variablen.

Der self Parameter ist dafür da die Referenz des ausführenden Objekts zurückzuliefern.
Sobald man ein Objekt instanziiert, wird der Konstruktor aufgerufen, in dem dann die Werte der Attribute gesetzt werden.
Das Programm das man ausführt, muss dann aber genau wissen auf welchem konkreten Objekt es aktuell arbeitet, denn es kann ja mehrere Objekte von einem Typ geben.
Hierfür benötigt man dann self, denn dort wird die Referenz auf das aktuelle Objekt übergeben, sobald man die Konstruktormethode aufruft.
Daher verwenden wir beim Festlegen von Attributen immer self.ATTRIBUT, da es dieses Attribute ja mehrmals geben kann, wenn es mehrere Objekte eines Typs gibt.

METHODEN IN KLASSEN

Eine Methode ist eine Funktion, die innerhalb einer Klasse definiert wird und sich ganz spezifisch auf Objekte dieser Klasse bezieht.
Man beginnt wie gewohnt mit dem Schlüsselwort def und dem Namen der Methode.
Danach folgt der self Parameter, wodurch der Methodenkörper auf das aufzurufende Objekt referenzieren kann.
Zum Beispiel:

```
def drive(self):
    self.xPosition = +x
    self.yPosition = +y
```

Diese Eigenschaften innerhalb der Methode müssen außerdem noch in die init-Methode der Klasse eingefügt werden.
Also:

```
class Car():
    def __init__(self):
        self.car_brand = None
        self.horse_power = None
        self.color = None
        self.xPosition = 5
        self.yPosition = 5
```

Die Methode drive kann somit also direkt in die entsprechende Klasse drive() programmiert werden. Dadurch wird die Klasse zu einer geschlossenen Einheit und alle Funktionen der Klasse sind in der Klassendefinition enthalten.
Das hat den Vorteil, dass man Änderungen der Methode einfach in der Klasse vornehmen kann und nicht im Code im Hauptprogramm die richtige Zeile suchen muss.

MÖGLICHKEITEN DES KONSTRUKTORS

Ein Konstruktor ist dafür verantwortlich, dass ein neu erzeugtes Objekt einen gewünschten Initialzustand aufweist. So kann man z.B. Attribute eines Objekts mit einem gewünschten Wert vorbelegen. Immer wenn man dann ein neues Objekt von einem bestimmten Typ (also mit dem Klassenaufruf) erzeugt, wird die init-Methode aufgerufen, die dafür sorgt, dass die Attribute den jeweiligen Wert haben.

Man kann zum Beispiel festlegen, dass man bei der Erzeugung eines Klassenobjekts dem Konstruktor Argumente mitgibt. Dann muss man diesen auch zwingend Parameter übergeben. Man kann diese Parameter wieder als optional festlegen, indem man ihnen einen default Wert zuweist.

WICHTIG

Die init-Methode wird automatisch aufgerufen sobald ein Objekt erzeugt wird. Sie gibt dann automatisch das erzeugt Objekt zurück. Hierfür benötigt man keine extra return Anweisung!

NAMESKONVENTIONEN

Tips

1. Klassen und Methoden sollten „sprechend“ benannt werden
2. Klassen: Starten mit Großbuchstaben. Besteht der Klassename aus mehreren Wörtern, startet jedes neue Wort ebenfalls mit einem Großbuchstaben (= Pascal Case)
3. Variablen & Funktionen: Generell alles klein, mehrere Wörter werden mit einem _ miteinander verbunden (= Sneak Case)