Vererbung und Module

Richard Müller, Tom Felber

25. November 2021

Python-Kurs

Gliederung

1. Wiederholung

2. Vererbung

3. Module - Einführung

Gesamtübersicht

Gesamtübersicht

Themen der nächsten Stunden

- Referenzen Erklärung
- Klassen
- Imports
- Nützliche Funktionen zur Iteration
- Lambda
- File handeling
- Listcomprehension
- Unpacking
- Dekoratoren

Wiederholung

Wiederholung

Beim letzten Mal:

- Referenzen
- Klassen

```
1 class Rennwagen:
2     def __init__(self):
3         self.speed = "sehr hoch"
```

Vererbung

Vererbung - Konzept

- Vererbung ermöglicht das weitergeben von Eigenschaften und Funktionen einer Klasse an eine andere
- Die Klasse, die von einer anderen erbt, wird auch als Kind-Klasse oder auch Sub-Klasse bezeichnet
- Die Klasse, von der geerbt wird, wird als Eltern-Klasse oder auch Super-Klasse bezeichnet
- Kind-Klassen übernehmen die Funktionalität ihrere Eltern
- Kind-Klassen können wie ihre Eltern behandelt werden, ElternKlassen nicht wie ihre Kinder *

* Alle Kinder sind Menschen, aber nicht alle Menschen Kinder.

Vererbung

Folgende Klasse ist gegeben:

```
1 class Baum:
2     def __init__(self, alter, hoehe):
3         self.alter = alter
4         self.hoehe = hoehe
5 
6     def wachsen(self, hoehe):
7         self.hoehe += hoehe
```

Es kann eine neue Klasse definiert werden, die von `Baum` erbt:

```
1 class NadelBaum(Baum):
2     pass
```

Vererbung

```
1 class NadelBaum(Baum):
2     pass
```

Die neue Klasse `NadelBaum` verhält sich zunächst genau wie `Baum` .

```
1 nadelbaum1 = NadelBaum(15, 20)
2 nadelbaum1.wachsen(5)
3 print(nadelbaum1.hoehe) # 25
```

Dieses Verhalten ermöglicht, dass zukünftig alle Instanzen der NadelBaum Klasse auch als Instanz der Baum Klasse behandelt werden können.

Vererbung - Erweiterung

Um Verhalten zu erweitern, können:

- bestehende Methoden überschrieben werden
- neue Methoden angelegt werden
- bestehende Attribute verändert werden
- neue Attribute angelegt werden

Wenn eine Methode überschrieben wird, kann mit der `super()` Funktion auf die Methoden und Attribute der ElternKlasse zugegriffen werden.

```
1     def wachsen(self, hoehe):
2         super().wachsen(hoehe)
```

Vererbung

Um das Verhalten von Funktionen zu verändern, können sie überschrieben werden.

```
class NadelBaum(Baum):
```

```
    def wachsen(self, hoehe):
        super().wachsen(hoehe)
        print(f"um {hoehe} gewachsen")
```

Um neue Attribute einzufügen, kann der Konstruktor, die `__init__` Methode, überschrieben werden.

```
    def __init__(self, alter, hoehe, laenge_nadeln):
        super().__init__(alter, hoehe)
        self.laenge_nadeln = laenge_nadeln
```

Polymorphismus - Polymorphism

Vererbung erlaubt es, Objekte, die der gleichen Super-Klasse angehören, gleich zu behandeln, auch wenn sie möglicherweise verschiedenen Sub-Klassen angehören.

Beispiel:

```
1  class Tier:
2      def macht(self):
3          print("ein Geräusch")
4
5  class Kuh(Tier):
6      def macht(self):
7          print("muh!")
8
9  class Schwein(Tier):
10     def macht(self):
11         print("oink!")
12
13 tiere = [Kuh(), Schwein(), Schwein(), Kuh(), Kuh()]
14 for tier in tiere:
15     tier.macht()
```

Module - Einführung

Module

Module sind Dateien, die Python-Ausdrücke und Definitionen enthalten.

Sie enden immer mit .py. Diese Dateien kennen wir schon, nämlich als normale Scripte. Das Besondere ist jedoch, dass man diese Module als Objekt in sein eigentliches Script einbinden und nutzen kann.

Import

Über das `import` -Statement kann man Module in sein Script einfügen. Es gibt zwei Möglichkeiten, dieses Statement zu verwenden.

Im Folgenden gehen wir davon aus, dass sich das ausgeführte Script im selben Verzeichnis wie die Datei `mein_modul.py` befindet. In der Datei `mein_modul.py` soll es eine Klasse namens `ImportiereMich` geben.

Möglichkeit 1:

```
1 import mein_modul
2 i = mein_modul.ImportiereMich()
```

Möglichkeit 2:

```
1 from mein_modul import ImportiereMich
2 i = ImportiereMich()
```

Import

Nutzt man bei diesem Beispiel Möglichkeit 1, so verhält sich das Modul wie ein normales Objekt.

```
1 import mein_modul
2 i = mein_modul.ImportiereMich()
```

Mit dem `.` kann man auf Attribute und Methoden zugreifen, was hier nun Funktionen, Klassen und Variablen sind.