

WürGESchlange 3 - Lesson 5

Tobias Maschek, Viktor Reusch

<https://github.com/jemx/wise1920-python>

mit Materialien von Felix Döring, Felix Wittwer <https://github.com/fsr/python-lessons>

Lizenz: CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/>

2. Dezember 2019

Python-Kurs

1. Vererbung

2. Modularität

- Nutzen von Modulen

- Module erstellen

- Kurzüberblick Pakete

- PIP

Folien jetzt auch unter <https://github.com/jemx/wise1920-python>

Vererbung

- englisch „*inheritance*“
- *Basisklasse* beschreibt eine Obergruppe oder generelles Konzept
- *Unterklasse* beschreibt eine Untergruppe oder Spezialisierung
- Unterklasse übernimmt Methoden und Attribute der Basisklasse
- Unterklasse kann neue Attribute und Methoden hinzufügen
- Unterklasse kann Methoden **überschreiben**
- Code für die Basisklasse funktioniert auch mit Unterklasse (*MeistensTM*)

Wer kommt woher?

```
1 class Vehicle:
2     def __init__(self, wheels):
3         self.wheels = wheels
4
5     def status(self):
6         print(f"I have {self.wheels} wheels.")
7
8 class Car(Vehicle):
9     def __init__(self, brand):
10         super().__init__(4)
11         self.brand = brand
12
13     def status(self):
14         print(f"I am a car by {self.brand}.", end=" ")
15         # status() would call this function again
16         super().status()
17
18 car = Car("OOP-Supercar")
19 car.status() # I am a car by OOP-Supercar. I have 4 wheels.
```

Wer von wem? - Quiz

```
1 class Vehicle:
2     pass
3
4 class Car(Vehicle):
5     pass
6
7 class Bike(Vehicle):
8     pass
9
10 car = Car()
11 bike = Bike()
12
13 print(isinstance(car, Car))
14 print(isinstance(car, Vehicle))
15 print(isinstance(bike, Vehicle))
16 print(isinstance(bike, Car))
17 print(isinstance(12.43, Vehicle))
18 print(isinstance(car, object))
```

Wer von wem?

```
1 class Vehicle:
2     pass
3
4 class Car(Vehicle):
5     pass
6
7 class Bike(Vehicle):
8     pass
9
10 car = Car()
11 bike = Bike()
12
13 print(isinstance(car, Car)) # True
14 print(isinstance(car, Vehicle)) # True
15 print(isinstance(bike, Vehicle)) # True
16 print(isinstance(bike, Car)) # False
17 print(isinstance(12.43, Vehicle)) # False
18 print(isinstance(car, object)) # True
```


Aufgabe 5-1

Schreibe ein Python-Programm, dass die Klassen *Rectangle*, *Square*, *Circle* definiert. Dabei soll folgendes gelten:

- ein Quadrat ist Rechteck, bei dem die Seitenlängen gleich sind.
- sowohl *Rectangle* also auch *Circle* besitzen die Methoden *area()* und *circumference()*, die die entsprechenden Eigenschaften berechnen und ausgeben.
- Es soll so viel wie möglich mit Vererbung gearbeitet werden. Nicht jedes Objekt kann von jedem erben.

Hinweis:

```
1 import math
2
3 print(math.pi) # 3.141592653589793
```

Modularität

Warum?

- Große Projekte strukturieren
- Übersichtlichkeit
- Aufgabenteilung
- Nutzen von bereits existierenden Algorithmen

Nutzen von Modulen

Importieren eines Moduls

```
1 import math
2
3 print(math.pi) # 3.141592653589793
```

Importieren einer Funktion

```
1 from math import sin
2 from math import cos as cosinus
3 import sys
4
5 print(sin(1)) # 0.8414709848078965
6 print(cosinus(1)) # 0.5403023058681398
7 print(sys.api_version) # 1013
```

Module erstellen

Ein Modul ist geboren ...

Das Modul, dass wir nutzen wollen:

module.py

```
1 def greet():  
2     print("Hello World")
```

Die Datei, die wir später ausführen und die das Modul importiert:

main.py

```
1 import module  
2  
3 module.greet()
```


... und die Probleme fingen an.

Altes Projekt mit nützlicher Funktion:

old.py

```
1 def usefull():  
2     print("Hello World")  
3  
4 print("really nasty old stuff")
```

Neues Projekt, dass die Funktion nutzt:

new.py

```
1 from old import usefull  
2  
3 usefull()
```

... und die Probleme fingen an.

Altes Projekt mit nützlicher Funktion:

old.py

```
1 def usefull():  
2     print("Hello World")  
3  
4 print("really nasty old stuff")
```

Neues Projekt, dass die Funktion nutzt:

new.py

```
1 from old import usefull  
2  
3 usefull()
```

Ausgabe:

really nasty old stuff

Hello World

Altes, mit Weitsicht erarbeitetes Projekt mit nützlicher Funktion:

old.py

```
1 def usefull():  
2     print("Hello World")  
3  
4 # only true when executed directly not when imported  
5 if __name__ == "__main__":  
6     print("really nasty old stuff")
```

Neues Projekt, dass die Funktion nutzt:

new.py

```
1 from old import usefull  
2  
3 usefull()
```

Aufgabe 5-2

Schreibe ein Python-Programm, das

- ein Modul *fib* enthält. Definiere dort eine Funktion, die die Fibonacci-Folge bis n ausgibt.
Wiederholung: rekursive Funktionen
- in einer *main* Datei dieses Modul einbindet und aufruft.

Zukunftssicher Schreiben

Alte Projekte können besser wiederverwendet werden, wenn der Code, der nicht in einer Funktion steht, durch folgende Kontrollstruktur vor ungewollter Ausführung abgesichert ist.

```
if __name__ == "__main__":
```

Kurzüberblick Pakete

- Ermöglichen Module zu gruppieren
- Effektiv nur Ordner mit einer meist leeren `__init__.py`-Datei
- Enthält dann die Modul-Dateien
- Module des Paketes mit `from package_name import module_name` importieren
- `__main__.py`-Datei wird beim direkten Ausführen des Modules gestartet

Ordnerstruktur:

```
1 coolpackage
2   |- __init__.py
3   |- __main__.py
4   \- crazymodule.py
```

Beispiel - Sourcecode

__init__.py

```
1 # This file can literally be empty.
```

__main__.py

```
1 from coolpackage.crazymodule import crazy
2
3 crazy()
```

crazymodule.py

```
1 def crazy():
2     return "CRAZY"
```


Beispiel - Ausführen

```
1 > python -m coolpackage  
2 CRAZY
```

PIP

DER Packagemanager für Python

Nützliche Packages (Sachen, wovon man mal gehört haben sollte):

- Pillow
- pycryptodome
- matplotlib
- numpy
- request

Syntax: *pip install PACKAGENAME*

Beispiel - Ausführen

```
1 pip list
2 Package                Version
3 -----
4 colour                  0.1.5
5 Django                  2.2.6
6 funcsigns               1.0.2
7 future                  0.17.1
8 latex                   0.7.0
9 manimlib                0.1.5
10 numpy                   1.15.0
11 opencv-python           3.4.2.17
12 pbr                     5.3.0
13 Pillow                  5.2.0
14 pip                     19.1.1
15 progressbar             2.5
16 pytz                    2019.3
17 scipy                   1.1.0
18 WARNING: You are using pip version 19.1.1, however version 19.3.1 is available
19 You should consider upgrading via the 'python -m pip install --upgrade pip'
   command.
```