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

26. November 2019

Python-Kurs

Gliederung

1. Vererbung
2. Modularität
   - Nutzen von Modulen
   - Module erstellen
   - Kurzüberblick Pakete
   - PIP
3. I/O

Jetzt auch im Neuland

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

Vererbung

„Ich bin dein Vater“

- 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 *(Meistens^{TM})*

Wer kommt woher?

```python
class Vehicle:
    def __init__(self, wheels):
        self.wheels = wheels

    def status(self):
        print(f"I have {self.wheels} wheels.")

class Car(Vehicle):
    def __init__(self, brand):
        super().__init__(4)
        self.brand = brand

    def status(self):
        print(f"I am a car by {self.brand}.", end=" ")
        # status() would call this function again
        super().status()

car = Car("OOP-Supercar")
car.status() # I am a car by OOP-Supercar. I have 4 wheels.
```

Wer von wem? - Quiz

```python
class Vehicle:
    pass

class Car(Vehicle):
    pass

class Bike(Vehicle):
    pass

car = Car()
bike = Bike()

print(isinstance(car, Car))
print(isinstance(car, Vehicle))
print(isinstance(bike, Vehicle))
print(isinstance(bike, Car))
print(isinstance(12.43, Vehicle))
print(isinstance(car, object))
```

Wer von wem?

```python
class Vehicle:
    pass

class Car(Vehicle):
    pass

class Bike(Vehicle):
    pass

car = Car()
bike = Bike()

print(isinstance(car, Car)) # True
print(isinstance(car, Vehicle)) # True
print(isinstance(bike, Vehicle)) # True
print(isinstance(bike, Car)) # False
print(isinstance(12.43, Vehicle)) # False
print(isinstance(car, object)) # True
```

Aufgabe 5-1

Schreibe ein Pyhton-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

```
def usefull():
    print("Hello World")

print("really nasty old stuff")
```

Neues Projekt, dass die Funktion nutzt:

new.py

```
from old import usefull

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

Lösung

Altes, mit Weitsicht erarbeitetes Projekt mit nützlicher Funktion:

old.py

```
def usefull():
    print("Hello World")

# only true when executed directly not when imported
if __name__ == "__main__":
    print("really nasty old stuff")
```

Neues Projekt, dass die Funktion nutzt:

new.py

```
from old import usefull

usefull()
```

Aufgabe 5-2

Schreibe ein Pyhton-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

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

Beispiel - Ordnerstruktur

Ordnerstruktur:

```
coolpackage
 |- __init__.py
 |- __main__.py
 \- crazymodule.py
```

Beispiel - Sourcecode

__init__.py

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

__main__.py

```
from coolpackage.crazymodule import crazy

print("crazy")
```

crazymodule.py

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

Beispiel - Ausführen

```
> python -m coolpackage
CRAZY
```

PIP

PIP

DER Packagemanager für Python
Nützliche Packages (Sachen, wovon man mal gehört haben sollte):

- Pillow
- pycryptodome
- mathplotlib
- numpy
- request

Syntax: *pip install PACKAGENAME*

Beispiel - Ausführen

```
pip list
Package             Version
------------------- --------
colour              0.1.5
Django              2.2.6
funcsigs            1.0.2
future              0.17.1
latex               0.7.0
manimlib            0.1.5
numpy               1.15.0
opencv-python       3.4.2.17
pbr                 5.3.0
Pillow              5.2.0
pip                 19.1.1
progressbar         2.5
pytz                2019.3
scipy               1.1.0
WARNING: You are using pip version 19.1.1, however version 19.3.1 is available
You should consider upgrading via the 'python -m pip install --upgrade pip'
    command.
```

I/O

Datei einlesen

```
# hello_world.txt:
#    Hello
#    World
#    !
with open("hello_world.txt") as file:
    for line in file:
        print(line)
# Output:
#    Hello
#
#    World
#
#    !
#
```

with

`with` sorgt dafür, dass die Datei nach dem Ende des Einlese-Blocks wieder geschlossen wird.

Datei schreiben

```
with open("numbers.txt", mode="w") as file:
    for i in range(3):
        file.write(f"{i}\n")
# numbers.txt:
#   0
#   1
#   2
```

mode

Es gibt verschidene Modi, mit denen man eine Datei öffnen kann:

r	einlesen *(Standard)*
w	schreiben
w+	lesen und schreiben
a	an bestehende Datei anfügen

Aufgabe 5-3

Lese und schreibe eine Datei:

- Lese die Zeilen einer Text-Datei in eine Liste.
- Drehe die Reihenfolge der Elemente in der Liste um.
- Schreibe die umgedrehte Liste der Zeilen in eine neue Datei.