Würgeschlange 3 - Lesson 4

Tobias Maschek, Viktor Reusch
https://github.com/jemx/wise1920-python

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

19. November 2019

Python-Kurs

Gliederung

1. builtins

2. Objektorientierte Programmierung

3. I/O

Jetzt auch im Neuland

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

builtins

set

- kann nur *hashbare* Einträge enthalten - **Hash, kann man das rauchen?**
- enthält jedes Element nur einmal
- mit `my_set.add()` Elemente hinzufügen
- schnellere Überprüfung mit `in` (prüft, ob Element enthalten ist)
- Mögliche Operationen: `<=`, `<`, `>`, `|`, `&`, `-`, $\sim$, $\wedge$
- **ungeordnet**

set - Beispiel

```
s1 = {1, 2, 'string', ('ein', 'tuple')}

2 in s1  # ==> True
'ein' in s1  # ==> False
('ein', 'tuple') in s1  # ==> True
set(('ein', 'tuple'))  # ==> {'ein', 'tuple'}
set(['eine', 'liste'])  # ==> {'liste', 'eine'}

{2} < s1  # ==> True
s1 <= {2}  # ==> False

s2 = {'anderes', 'set'}
s1.add('anderes') # s1 == {1, 2, 'string', ('ein', 'tuple'), 'anderes'}
s1 | s2  # ==> {1, 2, 'string', object, ('ein', 'tuple'), 'set'}
s1 & s2  # ==> {'anderes'}
s2 - s1  # ==> {'set'}
```

input()

```
>>> var = input()
>? Hello, this is my input



>>>var
'Hello, this is my input'
```

enumerate()

```
my_list = ['apple', 'banana', 'grapes', 'pear']
for c, value in enumerate(my_list, 1):
    print(c, value)

# Output:
# 1 apple
# 2 banana
# 3 grapes
# 4 pear
```

sum()

```
numbers = [21, 42, 84]

sum(numbers)

# Output:
# 147
```

Achtung

`sum()` funktioniert nur auf Zahlen (int, long, float).

min()

```
numbers = [21, 42, 84]

min(numbers)

# Output:
# 21
```

Achtung

`min()` funktioniert nur auf Zahlen (int, long, float).

max()

```
numbers = [21, 42, 84]

max(numbers)

# Output:
# 84
```

Achtung

`max()` funktioniert nur auf Zahlen (int, long, float).

round()

```
round(5.76543, 2)
# Output
# 5.77

import math
round(math.pi, 0)
# Output
# 3.0

round(math.pi)
# Output
# 3
```

Achtung

`round()` funktioniert nur auf Zahlen (int, long, float).

filter()

```
1 def even(i):
2     return i % 2 == 0
3
4 l = list(range(10)) # ==> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
5 list(filter(even, l)) # ==> [0, 2, 4, 6, 8]
6
7 [e for e in l if even(e)] # ==> [0, 2, 4, 6, 8]
```

all(), any()

```
l1 = [True, False, True]
all(l1) # ==> False
any(l1) # ==> True

l2 = [True, True]
all(l2) # ==> True
any(l2) # ==> True

l3 = [False, False]
all(l3) # ==> False
any(l3) # ==> False
```

map()

```
l = [-1, 4, -5, 3, 6]

a = list(map(abs, l)) # ==> [1, 4, 5, 3, 6]

def even(i):
    return i % 2 == 0

any(map(even, l)) # ==> True
```

sorted()

```
l = [-1, 4, -5, 3, 6]

sorted(l) # ==> [-5, -1, 3, 4, 6]

s = list("Hallo!") # ==> ['!', 'H', 'a', 'l', 'l', 'o']
sorted(s)
```

Aufgabe 4-1

Schreibe ein Pyhton-Programm, dass

- alle geraden Zahlen von 0 bis 100 aufsummiert, benutze dabei *sum()* und *filter()*.
- erst eine Liste der größe nach sortiert und dann jedes Element quadriert.

```
1 unsorted = [3, -2, 4, 0, -16] # input
2
3 squared = [256, 4, 0, 9, 16] # output
```

Objektorientierte Programmierung

OOP - Theorie

- Modellierung von Gegenständen und Konzepten als Klassen
- Ein einzelner konkreter Gegenstand als Objekt
- Diese Objekte besitzen Eigenschaften *(Attribute)*.
- Mit Objekten kann gearbeitet werden *(Methoden)*.
- Verschiedene Gegenstände können in Konzepten gruppiert werden *(Vererbung)*.

Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world.

— Grady Booch (1986)

All OO languages show some tendency to suck programmers into the trap of excessive layering. Object frameworks and object browsers are not a substitute for good design or documentation, but they often get treated as one.

— Eric S. Raymond (2003)

Klassen

Szenario

Auto

Eigenschaften

- Position (x: float)
- Geschwindigkeit (s: float)
- Marke (brand: str)

Aktionen

- Fahren (time: float)
- Status ausgeben

A whole new ~~World~~ Class

```
class Car:
    pass
```

Constructing a constructor

```
class Car:
    def __init__(self, brand):
        self.brand = brand

car = Car("OOP-Speedcar")
print(car.brand) # OOP-Speedcar
car.brand = "Python-master"
print(car.brand) # Python-master
```

Methods

```python
class Car:
    def __init__(self, x, s, brand):
        self.pos = x
        self.speed = s
        self.brand = brand

    def status(self):
        print(f"I am a car by {self.brand} and I am at {self.pos}.")

suv = Car(0, 10, "OOP-SUV")
suv.status() # I am a car by OOP-SUV and I am at 0.

nano = Car(100, 20, "OOP-Nano")
nano.status() # I am a car by OOP-Nano and I am at 100.

suv.pos = 12
suv.status() # I am a car by OOP-SUV and I am at 12.
```

Aufgabe 4-2

Schreibe ein Pyhton-Programm, dass

- ein *Car*-Objekt definiert und instanziiert.
- ein *Car* soll eine Methode *drive(time)* besitzen, welche die aktuelle Position um das Produkt der übergebenen Zeit und der Geschwindigkeit erhöht.

```
car1 = Car(0, 10, "BrandNameHere")
print(car1.pos) # 0
car1.drive(3)
print(car1.pos) # 30
```

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 4-3

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
```

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 4-4

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.