

Grundlagen

Felix Döring, Felix Wittwer

24. April 2017

Python-Kurs

- 1. Scriptcharakter
- 2. Programmierparadigmen
 - Imperatives Programmieren
 - Das Scoping Problem
 - Objektorientiertes Programmieren
- 3. Klassen und Attribute
 - Klassen- und Objektattribute im Detail
 - Super- und Subklassen
- 4. Methoden
 - Spezielle Methoden

Scriptcharakter

- Beim Ausführen oder Importieren wird der Code im obersten Level des Moduls (der .py Datei) ausgeführt
- Funktionen, Klassen und globale Variablen werden üblicherweise auf dem obersten Level definiert
- Imports anderer Python Module werden auch hier ausgeführt

```
1 def main():  
2     pass  
3  
4 if __name__ == '__main__':  
5     main()
```

Soll das entsprechende Modul ausführbar sein und nicht nur als Bibliothek dienen, definiert man üblicherweise eine **main**-Funktion. Zusätzlich fügt man am Ende des Moduls die Boilerplate ein.

- Python Code wird nicht kompiliert, sondern beim importieren in Python Bytecode übersetzt
- Bytecode wird auf einer VM ausgeführt
- Kein Memory Management nötig, alles sind Referenzen
- Syntaxerror wird beim Importieren geworfen
- Andere Fehler findet man erst, wenn die betreffende Zeile ausgeführt wird.

Programmierparadigmen

- Python ist vor allem eine imperative und objektorientierte Sprache
- reine Funktionen und Variablen können auf oberster Ebene definiert werden
- Variablen, Klassen und Funktionen sind ab der Ebene sichtbar, in der sie eingeführt werden

Das Problem mit dem Scope

```
1 var = 12
2
3 def foo():
4     # erwarteter Effekt: var wird auf 9 gesetzt
5     var = 9
6
7 def main():
8     print(var)    # -> gibt 12 zurueck
9     foo()
10    print(var)    # -> Erwartung: gibt 9 aus.
11                  # Realitaet: gibt 12 aus.
12
13 if __name__ == '__main__':
14     main()
```

Das Problem:

Variablen sind zwar nach innen sichtbar, werden aber beim Reassignment innerhalb der Funktion neu angelegt und verschwinden so aus dem Scope.

```
1 var = 12
2
3 def foo():
4     global var
5     # global sagt dem Interpreter, dass er hier auf die
6     # oberhalb definierte Variable zurueckgreifen soll
7     var = 9
8
9 def main():
10    print(var)    # -> gibt 12 zurueck
11    foo()
12    print(var)    # -> gibt jetzt 9 aus
13
14 if __name__ == '__main__':
15     main()
```

- Python ist auch fundamental objektorientiert
- Alles in Python ist ein Objekt
- Selbst die Datentypen `int`, `bool`, `str` und `type` sind Instanzen von `object` und haben folglich Methoden und Attribute
- Der Typ jedes Wertes und jeder Variable lässt sich mit `type()` ermitteln

Klassen und Attribute

- Typen in Python werden ausgedrückt durch Klassen (Keyword `class`)
- Klassen dienen als Vorlage bzw. Schablone → Objekte sind dann Instanzen davon
- Die Besonderheit: alle Variablen und Werte sind Instanzen von `object`

`object` und alle Typen selbst sind wiederum Objekte, genauer gesagt Instanzen vom Typ `type` und `type` wiederum ist eine Subklasse von `object`.

- Klassen und Objekte können selbst auch Variablen tragen (ähnlich wie in Java)
- Man unterscheidet dabei zwischen Klassenattributen und Instanzattributen

Klassenattribute werden für die Klasse definiert und sind für alle Instanzen gleich

Instanzattribute werden außerhalb der Klassendefinition hinzugefügt (normalerweise im Initialisierer) und sind für jede Instanz unterschiedlich.

- Zugriff auf Attribute über Punktnotation (wie in Ruby/Java)
- Attribute werden wie Variablen mithilfe von `=` gesetzt
- Man kann Objekten jederzeit neue Attribute hinzufügen (auch type-Objekten)
- Ist außerhalb des Initialisierers nicht empfehlenswert

Klassen- und Objektattribute im Detail

Klassenattribute sind für jede Instanz eines Objektes gleich.

```
1 class TestClass:
2     # jeder Instanz wird bei Erstellung bereits dieses
3     # Attribut zugewiesen
4     num = 12
5
6 def main():
7     a = TestClass()
8     b = TestClass()
9     # beide Variablen haben fuer 'num' von der Erstellung an
10    # den gleichen Wert
11
12    # das Aendern der Variable ueberschreibt das
13    # Klassenattribut mit einem Instanzattribut
14    a.num = -3
15    print(b.num)    # -> liefert immer noch 12
```

Klassen- und Objektattribute im Detail

Gewöhnlich definiert man Instanzattribute allerdings im **Initialisierer**.

```
1 class Human:
2     def __init__(self, firstname, lastname):
3         # die beiden Parameterwerte werden in
4         # Instanzattributen gespeichert.
5         self.firstname = firstname
6         self.lastname = lastname
7
8 def main():
9     # instanziiert zwei Objekte vom Typ 'Human'
10    matthias = Human("Matthias", "Stuhlbein")
11    john = Human("John", "Doe")
```

- Instanzattribute sollten immer in `__init__` definiert werden, um sicherzustellen, dass alle Instanzen die gleichen Attribute haben

Eine Klasse kann Attribute einer anderen Klasse erben, indem sie mit `class subclass(superclass):` definiert wird.

- Subklassen enthalten von Anfang an alle Attribute der Superklasse.
- Es können neue Variablen und Methoden hinzugefügt, und auch alte überschrieben werden.
- Die Attribute der Superklasse können mit `super()` aufgerufen werden.

Super- und Subklassen

```
1 class Human():
2     def __init__(self, fistname, lastname, dob):
3         self.firstname = fistname
4         self.lastname = lastname
5         self.dob = dob
6
7
8 class Child(Human):
9     # Ein Child ist einfach nur ein Human mit den
10    # zusaetzlichen Attributen father und mother
11    def __init__(self, fistname, lastname, dob, father,
12    mother):
13        super().__init__(fistname, lastname, dob)
14        self.father = father
15        self.mother = mother
```

Methoden

- Methoden sind Funktionen
- Allgemeiner Typ von Methoden ist daher `function`
- Methoden liegen im Namespace der zugehörigen Klasse, müssen daher mit `ClassName.method_name()` angesteuert werden (oder auf dem Objekt aufgerufen werden)
- Methoden haben ein implizites erstes Argument (typischerweise `self` genannt, kann aber variieren)
- Beim Aufruf auf einer Instanz wird das Objekt selbst automatisch übergeben

Dies sind Methoden die auf den meisten Grundlegenden Datenstrukturen implementiert sind, z.B. object.

Die Folgenden beginnen und enden normalerweise mit zwei Unterstrichen.

Initialisierer Oft auch (fälschlicherweise) Konstruktor genannt.

Name: `--init--`

Wird immer aufgerufen wenn eine neue Instanz der Klasse erstellt wird.

Finalisierer Oft auch (fälschlicherweise) Destructor genannt.

Name: `--del--`

Wird immer aufgerufen wenn das Objekt vom Garbage Collector aufgeräumt wird. (selten verwendet)

String Konvertierer Äquivalent zu Java's toString Methode.

Name: `--str--`

String Repräsentation Ähnlich wie `--str--` aber gedacht für eine für Debug verwendbare Repräsentation anstatt für Output wie `--str--`.