

Programmieren mit Python. Komplexere Datenstrukturen.

Frank Hofmann

version 1.1, 16. Juni 2015

Inhaltsverzeichnis

[Vorwort](#)

[1. Datentypen \(Überblick\)](#)

[2. Grundlegende Strukturen](#)

[2.1. Funktionen für Mengen, Listen, Tupel und Disctionaries](#)

[3. Ausgewählte Stringfunktionen für den Alltag](#)

[4. Erweiterte Strukturen](#)

[5. Objekte und Klassen](#)

[5.1. Klassendefinition](#)

[5.2. Öffentliche und private Attribute](#)

[5.3. Instantiierung von Objekten](#)

[5.4. Erzeugung und Benutzung von Methoden](#)

[5.5. Überladen von Methoden](#)

[5.6. Klassen ableiten](#)

[6. Weiterführende Dokumente](#)

Vorwort

Python als Scriptsprache. Komplexere Datenstrukturen bilden das Rückgrat vieler Projekte.

Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

–Tim Peters

1. Datentypen (Überblick)

- Integer

```
betrag1 = 78  
betrag2 = -56
```

- Float

```
umsatzsteuer7 = 0.07  
umsatzsteuer19 = 0.19
```

- String

```
# einzeliger String  
ort1 = "Berlin"  
ort2 = 'Hamburg'  
  
# mehrzeiliger String  
ort3 = """Büro 2.0  
Konferenzraum im 1. Stock  
Weigandufer 45, 12059 Berlin"""
```

- Boolean

```
wahr = True  
falsch = False
```

2. Grundlegende Strukturen

- Listen, Tupel, Dictionaries (Hashtabellen), Mengen
- Umwandlungen zwischen Listen, Dictionaries (Hashtabellen) und Mengen
 - set()
 - list()
 - tuple()

```
menge = set([1,2,3])  
liste = list(menge)  
tupel1 = tuple('abc') # gibt zurück ('a', 'b', 'c')  
tupel2 = tuple([1,2,3]) # gibt zurück [1, 2, 3]
```

- Funktionen für den Zugriff auf Elemente und Schlüssel — Zugriff über den Index

```
liste = [1,2,3,4]
print (liste[1])      # Ausgabe des zweiten Elements: 2
print (liste[-1])     # Ausgabe des letzten Elements: 4
print (liste[1:3])    # ergibt: [2,3]
```

- Ausgabe aller Indizes mit Hilfe von keys()

```
indizes = liste.keys() # ergibt [0,1,2,3]
```

- Ausgabe aller Werte mit Hilfe von values()

```
liste = [15, 34, 5]
werte = liste.values() # ergibt [15, 34, 5]
```

2.1. Funktionen für Mengen, Listen, Tupel und Disctionaries

len()

Anzahl der Elemente bestimmen

map(function, aufzaehlung)

Funktion auf jedes Element der Aufzählung anwenden

```
def quadrat(wert):
    return wert * wert

liste = [1,2,3,4]
print (map(quadrat, liste)) # ergibt [1,4,9,16]
```

min(), max()

Minimum und Maximum bestimmen

range()

Zahlenmenge (Iterator) erstellen

```
print (range(4)) # ergibt Bereich von 0 bis 3
```

sorted()

sortierte Menge oder Liste sum(): Summe der Werte berechnen zip(): zwei Mengen oder Listen kombinieren

```
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
print (zipped) # ergibt [(1, 4), (2, 5), (3, 6)]
```

3. Ausgewählte Stringfunktionen für den Alltag

`str.capitalize()`

String mit großem Anfangsbuchstaben, ansonsten Kleinbuchstaben

`str.upper()`

String nach Großbuchstaben wandeln

`str.lower()`

String Kleinbuchstaben wandeln

`str.endswith(suffix)`

gibt True zurück, falls der String mit dem Suffix endet

`str.startswith(preafix)`

gibt True zurück, falls der String mit dem Präfix beginnt

`str.split(separator)`

einen String anhand des Trennzeichens in Teilzeichenketten zerlegen

`str.find(teilstring)`

eine Teilzeichenkette finden

`str.strip()`

führende und endende Leerzeichen entfernen

```
vorname = " Felix Hauser "  
print (vorname.strip())    # ergibt: "Felix Hauser"
```

`str.join(liste)`

Strings aus der Liste miteinander verketteten

```
a = ("Das", "ist", "ein", "Beispiel")  
b = " ".join(a)  
print(b)          # ergibt 'Das ist ein Beispiel'
```

4. Erweiterte Strukturen

- mehrdimensionale Felder und Matrizen
- einfach und mehrfach miteinander verkettete Listen

5. Objekte und Klassen

5.1. Klassendefinition

- Festlegung über einen Bezeichner und das Schlüsselwort `class`

```
class Point:  
    # alle weiteren Elemente (Methoden und Attribute) gehören zur Klasse
```

5.2. Öffentliche und private Attribute

- keine Unterscheidung zwischen öffentlichen und privaten Attributen
- implizite Unterscheidung durch spezielle Schreibweise mit `_name`

```
class Point:
    pointId = 0      # öffentliches Attribut
    _zusatz = 42     # Attribute mit privatem Bezeichner
```

5.3. Instantiierung von Objekten

- Nutzung einer Konstruktor-Methode namens `__init__`
- wird aufgerufen, wenn des Objekt instantiiert wird

```
class Point:
    def __init__(self, x,y):
        self.xCoord = x
        self.yCoord = y
        return
```

- Erzeugung eines Objekts der Klasse `Point`

```
pointA = Point(2,4)
pointB = Point(17,-3)
```

5.4. Erzeugung und Benutzung von Methoden

- Definition einer Funktion mit dem Schlüsselwort `def` und einem Bezeichner

```
class Point:
    def __init__(self, x,y):
        self.xCoord = x
        self.yCoord = y
        return

    def setX(self, x):
        self.xCoord = x
        return

    def getX(self):
        return self.xCoord
```

5.5. Überladen von Methoden

- Überschreiben und Redefinition bestehender Operatoren

...

```
def __eq__(self, referencePoint):
    return self.getX() == referencePoint.getX() and \
           self.getY() == referencePoint.getY()

...
pointA = Point(3,4)
pointB = Point(4,5)
if pointA == pointB:
    print("beide Punkte sind identisch")
```

5.6. Klassen ableiten

- in der Klassendefinition wird der Bezeichner der Klasse benannt, von der abgeleitet wird

```
class Point3D (Point):
    def __init__(self, x, y, z):
        super().__init__(x,y)
        self.zCoord = z
        return

    def getX(self):
        return super().getX()

    def getZ(self):
        return self.zCoord
```

6. Weiterführende Dokumente

- Dokumentation zu Python 3.4, <https://docs.python.org/3.4/index.html>
- Python Practice Book, <http://anandology.com/python-practice-book/index.html>
- Frank Hofmann: GitHub-Repo mit ausführlichen Beispielen, <https://github.com/hofmannedv/training-python>
- Rance D. Necaise: Data Structures and Algorithms using Python, Wiley, ISBN 9780-47061829-5

Version 1.1

Letzte Änderung 2015-06-16 15:23:07 CEST