Programmieren mit Python. Erste Schritte.

Frank Hofmann
version 1.1, 20. Mai 2015

Inhaltsverzeichnis

Vorwort
1. Entwicklungsumgebung
    1.1. Python Shell
    1.2. iPython Notebook
2. Programmaufbau und -struktur
    2.1. Übersetzung und Ausführung
3. Variablen und Bezeichner
4. Listen, Tupel, Sets und assoziative Arrays
5. Schleifen
6. Nutzung bestehender Funktionen (Module)
7. Entwicklung eigener Funktionen
8. Weiterführende Dokumente

Vorwort

Python als Scriptsprache. Einfach zu schreiben und kompakt, aber sehr mächtig und flexibel für den Alltag.

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. Entwicklungsumgebung

Um Programme mit Python zu entwickeln, bedarf es nicht viel. Ausreichend sind ein Texteditor mit Syntaxhervorhebung und eine Shell. Mehr Möglichkeiten eröffnen die Python Shell, eine IDE oder das iPython Notebook.

1.1. Python Shell

Die Python Shell wird mitgeliefert und ist über den Aufruf `python` (für Python 2.x) bzw. `python3` (ab Python 3.x) erreichbar.

```
user@rechner:~$ python3
Python 3.2.3 (default, Feb 20 2013, 17:02:41)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> wert1 = 15
>>> wert2 = 35
>>> summe = wert1 + wert2
>>> print (summe)
50
>>>
quit()
user@rechner:~$
```

1.2. iPython Notebook

Das iPython Notebook ist eine interaktive Shell im Webbrowser. Für Einsteiger erleichtert es die ersten Schritte bei der Entwicklung erheblich.

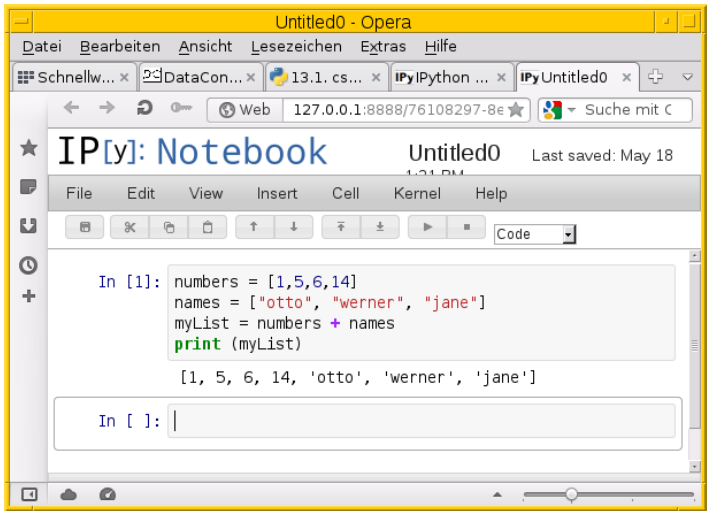

Abbildung 1. Eine Session im iPython Notebook

2. Programmaufbau und -struktur

Python folgt diesen Grundgedanken:

- Strukturierung durch Einrückung mittels Leerzeichen oder Tabulatoren, jedoch

nicht gemischt

- kein Sonderzeichen für das Zeilenende (wie bspw. `;` in C)
- Kommentare beginnen mit einem `#`

Hello World! in Python

```
#!/usr/bin/env/python

# -*- coding: utf-8 -*-

message = "Hello, world!"
print (message)
```

2.1. Übersetzung und Ausführung

- Festlegung des Python-Interpreters über die SheBang-Zeile am Anfang des Programms
- Standardencoding: UTF-8
- Ausführung über die Shell

```
$ python3 programm.py
$
```

- Verwendung als ausführbares Programm

```
$ chmod +x programm.py
$ ./programm.py
$
```

3. Variablen und Bezeichner

- Variablen sind stets im lokalen Kontext gültig, d.h. innerhalb eines Programms, einer Funktion oder Klasse
- als Bezeichner sind Buchstaben, Ziffern und Sonderzeichen wie _ erlaubt
- dynamische Typisierung, d.h. keine besondere Kennung für Datentypen

```
# Zahlenwert (Integer)
anzahl = 15

# Zahlenwert (Gleitkommazahl)
umsatzsteuer_7 = 1.07
umsatzsteuer_19 = 1.19

# Zeichenkette
bezeichnung = "Zahnbürste"
```

- Zuweisung (Einzel- und Mehrfachzuweisung)

```
# Einzelzuweisung
ort = "Berlin"

# Mehrfachzuweisung
ort, postleitzahl = "Berlin", 10247
```

4. Listen, Tupel, Sets und assoziative Arrays

Liste

Menge von Werten beliebigen Inhalts und Typs

```
artikelListe = [1, 4.5, "Berlin"]
```

String

Liste von Zeichen im Encoding UTF-8

```
# einzeilige Zeichenketten
stadt1 = "Berlin"
stadt2 = 'Potsdam'

# mehrzeilige Zeichenketten
mitteilung = """
Text (Zeile 1)
Text (Zeile 2)
Text (Zeile 3)
"""
```

Tupel

wie eine Liste, aber der Inhalt ist unveränderlich

```
herkunft = ("Europa", "USA", "Australien")
```

Set (Menge)

wie eine Liste, jedoch kein Element darf mehrfach vorkommen

```
stadt = {"Oslo", "Berlin", "Lausanne"}
```

```
# Liste
artikelListe = [anzahl, umsatzsteuer_19, bezeichner]

# Menge
farben = {"rot", "grün", "blau"}

# Prüfen auf Enthaltensein
if "blau" in farben:
        print ("blau ist in der Farbmenge enthalten")

if "gelb" not in farben:
        print ("gelb ist nicht in der Farbmenge enthalten")
```

assoziatives Array
: wie eine Liste, jedoch mit String als Index anstatt von Integerwerten. In anderen Programmiersprachen bekannt als Hashtabelle.

```
artikel = {
        "menge": anzahl,
        "ust": umsatzsteuer_19,
        "bezeichner": bezeichner
}

# Ausgabe der Artikelmenge
print (artikel["menge"])
```

5. Schleifen

- `for`-Schleife

```
# Mengen definieren
farben = ("rot", "grün", "blau")
rgb = {
        "rot" : "#F00",
        "grün": "#0F0",
        "blau": "#00F"
}

for eintrag in farben:
        print ("Farbe:%s" % eintrag)
        print ("RGB-Code:", rgb[eintrag])
```

- `while`-Schleife

```
# mit while-Schleife
farben = ("rot", "grün", "blau")

# Anzahl der Mengenelemente ermitteln
anzahlFarben = len(farben)

i = 0
while i < anzahlFarben:
    print ("%i: %s" % (i, farben[i]))
    i = i + 1
```

6. Nutzung bestehender Funktionen (Module)

- Modul vollständig benutzen

```
import Modulname
```

- nur einen Teil davon benutzen

```
from Modulname import Funktion
```

```
import sys

# Funktion aus dem importierten Modul benutzen
# - Kommandozeilenargument 0
print (sys.argv[0])

# - Rückkehr mit Exit-Code 1
sys.exit(1)
```

7. Entwicklung eigener Funktionen

- Bezeichner
  - wie Variablen, d.h. Buchstaben, Ziffern und Sonderzeichen
  - keine Kennzeichnung des Funktionstyps

Funktion ohne Parameter

```
# Funktion definieren
def meineFunktion():
        ort = "Berlin"

        return ort

# Funktion aufrufen
ergebnis = meineFunktion()
```

Funktion mit zwei Parametern und Liste als Rückgabewert

```
def umrechnung(parameter1, parameter2):
        zusatzwert = 15

        return [parameter1, parameter2, zusatzwert]

ergebnisliste = umrechnung(2,4)
```

8. Weiterführende Dokumente

- Dokumentation zu Python 3.4, https://docs.python.org/3.4/index.html
- PEP 0008 — Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008/
- Python Practice Book, http://anandology.com/python-practice-book/index.html
- Frank Hofmann: GitHub-Repo mit ausführlichen Beispielen, https://github.com/hofmannedv/training-python

Version 1.1
Letzte Änderung 2015-05-20 16:56:33 CEST