







And Now For Something Completely Different



Informatik I

12. Erste Schritte in Python

Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

16. Dezember 2010

Informatik I

16. Dezember 2010 — 12. Erste Schritte in Python

12.1 Motivation

12.2 Ausdrücke

12.1 Motivation

- Warum eine zweite Programmiersprache?
- Geschichte
- Warum Python?
- Materialien
- Arbeitsweise

Danksagung/Quellenangabe

- Die Vorlesung bisher baute auf den Materialien von Prof. Dr. Peter Thiemann und seinen Mitarbeitern auf.
- Im Folgenden verwendet die Vorlesung Materialien von Malte Helmert, Robert Mattmüller, Gabi Röger und Felix Steffenhagen, die bei diversen Python-Kursen in Freiburg zum Einsatz kamen.

Warum eine zweite Programmiersprache?

Eine Polemik

Zitate aus *Die Macht der Abstraktion* [KS07]:

- "Zur Illustration und zum Training der Programmierung dient Scheme, eine kleine und leicht erlernbare Programmiersprache, die es erlaubt, die Konzepte der Programmierung zu präsentieren, ohne Zeit mit der Konstruktvielfalt anderer Programmiersprachen zu verlieren."
- "Scheme-Könner sind in der Lage, andere Programmiersprachen in kürzester Zeit zu erlernen."
- "Alle wichtigen Programmiertechniken lassen sich in Scheme demonstrieren."

Positive Antworten

Zitate, gefunden auf der Webseite von Prof. Thiemann:

- "You can never understand one language until you understand at least two." (Ronald Searle, geboren 1920)
- "Eine zweite Sprache zu können, ist wie eine zweite Seele zu haben." (Karl der Große, 747/748(?)-814)

Weitere Gründe

- Früher oder später muss man in jedem Fall mehrere Programmiersprachen lernen, man sollte sich also frühzeitig an Unterschiede in der Syntax gewöhnen.
- Jede Programmiersprache hat ihre Vor- und Nachteile.
- Programmiersprachen gehören verschiedenen Paradigmen an ...

Programmierparadigmen

- Man unterscheidet imperative, funktionale, logische, objektorientierte, prozedurale, Skript-, und weitere Programmiersprachen.
- Es geht hier um grundsätzlich verschiedene Arten zu programmieren. Grob vergleichbar mit dem Unterschied zwischen Alphabetschriften, Silbenschriften und logografischen Schriften.
- Die Einteilung ist keineswegs eindeutig, selbst wenn man dies anstreben würde.
- Im Gegenteil versuchen aber typischerweise die Autoren einer Sprache in einem bestimmten Paradigma, auch Features anderer Paradigmen anzubieten.

Scheme

- Scheme ist, cum grano salis, eine funktionale Programmiersprache.
- In den bisherigen Folien tauchte 22 mal das Wort “Funktion” und 113 mal das Wort “Prozedur” auf, aber nicht ein einziges Mal die Worte “Befehl” oder “Zuweisung”.
- Das Funktionale ist das, was an Scheme so “komisch” ist (abgesehen von den vielen Klammern und der Präfixsyntax).
- Das letzte Kapitel *Prozeduren als Daten* zeigt am deutlichsten, worum es bei der funktionalen Programmierung geht.
- Ich möchte das funktionale Paradigma deutlich herausarbeiten und habe deshalb insbesondere die Kapitel 9 und 12 aus dem Buch [KS07] weggelassen.

Python

- Python ist eine objektorientierte Skriptsprache.
- Ich verkaufe sie Ihnen als imperative Programmiersprache.
- Ich werde hier nicht die Besonderheiten von Python hervorheben, sondern sie im Gegenteil herunterspielen (wie bei Scheme auch, siehe z.B. die Ausführungen zu Sorten).
- Ich möchte das imperative Paradigma deutlich herausarbeiten.
- Allerdings nicht heute! Zunächst wollen wir einige grundlegende Konstrukte aus Scheme einfach nach Python übersetzen.

Zur Geschichte Pythons

- Ursprünglich entwickelt von Guido van Rossum im Rahmen eines Forschungsprojekts am "Centrum voor Wiskunde en Informatica" in Amsterdam.
- Entwickelt seit 1989, erste öffentliche Version 1991.
- Meilensteine: Versionen 1.0.0 (1994), 1.5 (1998), 2.0 (2000), 3.0 (2008)
- Mittlerweile wird Python als Open-Source-Projekt von der Allgemeinheit weiterentwickelt, wobei ein innerer Kern die meiste Arbeit übernimmt. Guido van Rossum hat als "BDFL" (benevolent dictator for life, gütiger Diktator auf Lebenszeit) das letzte Wort.

Zum Namen

Python ist nicht nach einem Reptil benannt, sondern nach Monty Python, einer (hoffentlich!) bekannten englischen Komikertruppe aus den 1970ern.

Daher auch viele Namen von Tools rund um Python:

- IDLE
- Eric
- Bicycle Repair Man
- Grail

Wo andere Programmiersprachen die Variablen foo und bar verwenden, wählt man in Python gerne spam und egg.

Warum Python?

Python hat (z.B. gegenüber der C-Familie) einen hohen Abstraktionsgrad ("weiter weg von der Maschine"), z.B. automatische Speicherverwaltung und unbeschränkte Ganzzahlarithmetik.
Programme sind

- kürzer,
- lesbarer,
- portabler,
- langsamer.

Scheme hat ebenfalls einen hohen Abstraktionsgrad.

Python vs. Scheme

Fundamental unterschiedliche Syntax

Scheme

```
(define
  factorial
  (lambda (n)
    (if (<= n 1)
        1
        (* n (factorial (- n 1))))))
```

Python

```
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n-1)
```

Python vs. Scheme

Viele Gemeinsamkeiten im “Kern”:
→ Python for Lisp Programmers,
`http://www.norvig.com/python-lisp.html`

Materialien

Für die Online-Dokumentation ist der Einstiegspunkt `http://docs.python.org/py3k/`.

Besonders wichtig/interessant:

- am Anfang das Tutorial (`http://docs.python.org/py3k/tutorial/index.html`)
- im Programmieralltag die Library Reference (`http://docs.python.org/py3k/library/index.html`)

Links zu Büchern finden Sie auf unserer Webseite.

Materialien: Bücher

- M. Lutz. Learning Python [Lut09]. (Für Einsteiger)
- A. Martelli. Python in a Nutshell [Mar06].
  (Für Fortgeschrittene. Noch Python 2.)
- A. Martelli, A. Martelli Ravenscroft und D. Ascher. Python Cookbook [MRA05]. (Codebeispiele. Noch Python 2.)
- M. Pilgrim. Dive Into Python 3 [Pil09]. (Für Fortgeschrittene.)
  `http://diveintopython3.org/`
- A. Downey, J. Elkner und C. Meyers. How to Think Like a Computer Scientist: Learning With Python [DEM02].
  (Für Programmieranfänger. Noch Python 2.)
  `http://www.greenteapress.com/thinkpython/thinkCSpy/`
- Peter Kaiser und Johannes Ernesti. Python 3: Das umfassende Handbuch [KE09]. (Deutsch. Für Programmieranfänger.)
  Alte Version (Python 2):
  `http://openbook.galileocomputing.de/python/`

Arbeitsweise

- Auf den BP-Rechnern ist Python selbstverständlich installiert. Zur Installation auf Ihrem eigenen Rechner gibt es Informationen auf unserer Webseite.
- Wie bei Scheme auch gibt es zwei Arten der Interaktion:
  - Schreibe ein Programm (oberes Fenster in DrRacket).
  - Benutze den interaktiven Interpreter (in DrRacket REPL genannt).

  Nur ist die Welt jetzt nicht mehr ganz so komfortabel. Wir müssen ein paar grundlegende Konzepte aus den Betriebssystemen verstehen.

Betriebssysteme

- *Datei*: eine Zeichenkette (wobei es spezielle Zeichen für Zeilenumbruch und Dateiende gibt) versehen mit einem Namen und einigen weiteren Informationen (Datum der letzten Änderung, Leserechte etc.). Insbesondere die Scheme-Programme (`....rkt`) sind Dateien. Die Daten auf der Festplatte eines Computers sind in Dateien organisiert.
- Editor: ein allgemeines Programm zum Erstellen und Bearbeiten von Dateien. Beispiele: vi oder emacs.
  Die meisten Computerlaien kennen dieses Konzept wahrscheinlich nicht, weil sie nur Spezialprogramme (z.B. Word) zum Erstellen von Dateien verwenden. Das obere Fenster in DrRacket ist ein Editor für Scheme-Programme mit moderaten Spezialfunktionen.

Betriebssysteme

- Shell: Ein Programm, das auf Betriebssystemebene Befehle entgegen nimmt, d.h., von dem aus man andere Programme aufruft. Auch dieses Konzept kennen die meisten Computerlaien wahrscheinlich nicht, da in heutigen fensterbasierten Betriebssystemen Programme typischerweise über Menus aufgerufen werden.

Bei weiteren Fragen wenden Sie sich bitte an Herrn Wimmer!

Arbeitsweise

- Python-Programme schreiben Sie mit einem beliebigen Editor.
- Das Programm `python3` wird aus der Shell aufgerufen und kann sowohl verwendet werden, um Python-Programme auszuführen, als auch als interaktiver Interpreter benutzt werden.

Details: siehe Übungen!

Interpreter starten

In der Shell:

Shell

```
# python3
Python 3.1.2 (r312:79147, Apr 15 2010, 12:35:07)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
```

12.2 Ausdrücke

- Ausdrücke
- Auswertung / Ausgabe
- Zahlen

Grundbausteine

Zeichen mit fester Bedeutung

- Konstanten (Literale), z.B. für Zahlen oder Strings:

  ```
  42  -17  3.1415926535  "Gambolputty"
  ```

- Vordefinierte Namen (primitive Operatoren), z.B. für arithmetische Operationen:

  ```
  +   -   *   /
  ```

Zeichen mit frei wählbarer Bedeutung

Namen (Bezeichner, Identifier, Variablen)

```
x  y  egg  spam
```

Aus diesen bilden wir Ausdrücke.

Bildung von Ausdrücken

- Ein Literal ist ein Ausdruck.
- Eine Variable ist ein Ausdruck.
- Die (Funktions-)Anwendung, Applikation eines vordefinierten Namens auf Ausdrücke (Operanden) ist ein Ausdruck:

  `17 + 4` `x * (17 + 4)`

  Unterschied zu Scheme?

Auswertung

- Ausdrücke haben einen Wert, sie können ausgewertet werden.
- Jeder Ausdruck beschreibt einen Berechnungsprozess zur Ermittlung seines Wertes (Auswertung). Start der Auswertung durch Eingabe im Interpreter.

Auswertung: Beispiele

Eingabe eines nackten Ausdrucks im Interpreter:

Python-Interpreter

```
>>> 42
42
>>> 2 * (17 + 4)
42
```

Auswertung: Beispiele II

Man kann auch die `print`-Funktion verwenden, um einen Ausdruck auszugeben:

Python-Interpreter

```
>>> print(42)
42
>>> print(2 * (17 + 4))
42
```

`print` ist der übliche Weg, Ausgaben zu erzeugen und funktioniert daher auch in richtigen Programmen, d.h. außerhalb des Interpreters.

Nackte Ausdrücke vs. print

Es besteht ein kleiner aber feiner Unterschied zwischen nackten Ausdrücken und Ergebnissen der print-Funktion:

Python-Interpreter

```
>>> print(7 * 6)
42
>>> print(2.8 / 7)
0.4
>>> print("oben\nunten")
oben
unten
```

Python-Interpreter

```
>>> 7 * 6
42
>>> 2.8 / 7
0.39999999999999997
>>> "oben\nunten"
'oben\nunten'
```

Zahlen

Python kennt drei verschiedene Typen (Sorten) für Zahlen:

- `int` für ganze Zahlen beliebiger Größe.
- `float` für Fließkommazahlen.
- `complex` für komplexe (Fließkomma-) Zahlen.

int

`int`-Konstanten schreibt man, wie man es erwartet:

Python-Interpreter

```
>>> 10
10
>>> -20
-20
```

Python benutzt für Arithmetik die üblichen Symbole:

- Grundrechenarten: `+`, `-`, `*`, `/`, `//`
- Modulo: `%`
- Potenz: `**`

Rechnen mit `int`: Beispiele

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 13 % 8
5
>>> -2 % 8
6
>>> 11 ** 11
285311670611
```

Integer-Division: Ganzzahlig oder nicht?

Der Divisionsoperator / liefert das genaue Ergebnis. Das Ergebnis der ganzzahligen Division erhält man mit //. Dabei wird immer abgerundet.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>> 20 // 3
6
>>> -20 // 3
-7
```

Fließkommazahlen und komplexe Zahlen

- `float`-Konstanten schreibt man folgendermaßen:
  `2.44`, `1.0`, `5.`, `1e+100`
- `complex`-Konstanten schreibt man als Summe von (optionalem) Realteil und Imaginärteil mit imaginärer Einheit `j`:
  `4+2j`, `2.3+1j`, `2j`, `5.1+0j`

`float` und `complex` unterstützen dieselben arithmetischen Operatoren wie die ganzzahligen Typen (außer Modulo bei komplexen Zahlen).
Wir haben also:

- Grundrechenarten: `+`, `-`, `*`, `/`, `//`
- Potenz: `**`

Rechnen mit `float`

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>> print(10 ** 0.5)
3.16227766017
>>> print(4.23 ** 3.11)
88.6989630228
```

Rechnen mit complex

Python-Interpreter

```
>>> print(2+3j + 4-1j)
(6+2j)
>>> 1+2j * 100
(1+200j) [Achtung, Punkt vor Strich!]
>>> (1+2j) * 100
(100+200j)
>>> print((-1+0j) ** 0.5)
(6.12303176911e-17+1j)
```

Automatische Konversionen zwischen Zahlen

Ausdrücke mit verschiedenen Typen wie `100 * (1+2j)` oder `(-1) ** 0.5` verhalten sich so, wie man es erwarten würde. Die folgenden Bedingungen werden der Reihe nach geprüft, die erste zutreffende Regel gewinnt:

- Ist einer der Operanden ein `complex`, so ist das Ergebnis ein `complex`.
- Ist einer der Operanden ein `float`, so ist das Ergebnis ein `float`.
- Ansonsten ist das Ergebnis ein `int`.

Zusammenfassung

- Es ist gut, mehr als eine Programmiersprache zu lernen.
- Python ist imperativ und relativ einfach.
- Benutzung von Python
- Ausdrücke und Zahlen

Literatur I


Allen Downey, Jeffrey Elkner, and Chris Meyers.
How to Think Like a Computer Scientist: Learning With Python.
Green Tea Press, 2002.


Peter Kaiser and Johannes Ernesti.
Python 3: Das umfassende Handbuch.
Galileo Computing, 2009.


Herbert Klaeren and Michael Sperber.
Die Macht der Abstraktion.
Teubner Verlag, 2007.


Mark Lutz.
Learning Python.
O'Reilly Media, 2009.

Literatur II


Alex Martelli.
Python in a Nutshell.
O'Reilly Media, 2006.


Alex Martelli, Anna Martelli Ravenscroft, and David Ascher.
Python Cookbook.
O'Reilly Media, 2005.


Mark Pilgrim.
Dive Into Python 3.
Apress, 2009.