

Programmieren in Python

2. Ausgaben und Zahlen

Malte Helmert

Albert-Ludwigs-Universität Freiburg

KI-Praktikum, Sommersemester 2009

1 / 22

Ausgaben und Zahlen

In dieser Lektion geht es darum, ein erstes Gefühl für Python zu bekommen. Wir beschränken uns auf zwei einfache Bereiche:

- ▶ Ausgaben: `print`
- ▶ Zahlen: `int`, `long`, `float`, `complex`

2 / 22

Ausgaben und Zahlen

In dieser Lektion geht es darum, ein erstes Gefühl für Python zu bekommen. Wir beschränken uns auf zwei einfache Bereiche:

- ▶ **Ausgaben: `print`**
- ▶ Zahlen: `int`, `long`, `float`, `complex`

3 / 22

Interpreter & Runtime

Das Programm `python` (bzw. `python.exe`) kann sowohl verwendet werden, um Python-Programme auszuführen (so wie es das Programm `java` für Java-Programme tut), als auch als interaktiver Interpreter benutzt werden:

Shell

```
# python beispiel.py
And now for something completely different.
# python
Python 2.6.2 (r262:71600, Apr 29 2009, 21:44:36)
[GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> print 10 * 10
100
>>> exit
```

4 / 22

Ausgaben des Interpreters

Da die Beispiele in diesem Kapitel noch recht elementar sind, können wir sie allesamt im Interpreter behandeln.

Um dem Interpreter eine Ausgabe zu entlocken, gibt es zwei Methoden. Zum einen kann man einfach einen Ausdruck eingeben, woraufhin der Interpreter dann den Ausdruck auswertet und das Ergebnis ausgibt:

Python-Interpreter

```
>>> 7 * 6
42
>>> "hello, " + "world"
'hello, world'
>>> "spam " * 4
'spam spam spam spam '
```

5 / 22

Ausgaben des Interpreters (2)

Zum anderen kann man die `print`-Anweisung verwenden, um einen Ausdruck auszugeben:

Python-Interpreter

```
>>> print 7 * 6
42
>>> print "hello, " + "world"
hello, world
>>> print "spam " * 4
spam spam spam spam
```

`print` ist der übliche Weg, Ausgaben zu erzeugen und funktioniert daher auch in „richtigen“ Programmen, d.h. außerhalb des Interpreters, wo nackte Ausdrücke nur wegen ihrer Seiteneffekte verwendet werden.

6 / 22

Ausgaben des Interpreters (3)

Es besteht ein kleiner aber feiner Unterschied zwischen „nackten“ Ausdrücken und `print`-Anweisungen:

Python-Interpreter

```
>>> print 7 * 6
42
>>> print "Hello, world"
Hello, world
>>> print 2.3 / 4
0.575
>>> print "oben\nunten"
oben
unten
>>> print None
None
```

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello, world"
'Hello, world'
>>> 2.3 / 4
0.57499999999999996
>>> "oben\nunten"
'oben\nunten'
>>> None
>>>
```

Mehr dazu später, wenn es um die Funktionen `str` und `repr` geht.

7 / 22

Etwas mehr zu print

Wir werden die Möglichkeiten von `print` später noch ausführlicher behandeln. Ein Detail soll aber schon jetzt erwähnt werden, da es für die Übungen wichtig ist:

Python-Interpreter

```
>>> print "2 + 2 =", 2 + 2, "(vier)"
2 + 2 = 4 (vier)
```

- ▶ Man kann `print` mehrere Ausdrücke übergeben, indem man sie mit Kommas trennt.
- ▶ Die Ausdrücke werden dann in derselben Zeile ausgegeben, und zwar durch Leerzeichen getrennt.

8 / 22

Ausgaben und Zahlen

In dieser Lektion geht es darum, ein erstes Gefühl für Python zu bekommen. Wir beschränken uns auf zwei einfache Bereiche:

- ▶ Ausgaben: `print`
- ▶ Zahlen: `int`, `long`, `float`, `complex`

9 / 22

Zahlen

Python kennt vier¹ verschiedene Datentypen (bzw. Klassen²) für Zahlen:³

- ▶ `int` für ganze Zahlen im Bereich `-2147483648..2147483647` (mehr auf 64-Bit-Rechnern). `int` entspricht dem gleichnamigen Datentyp in C und Java.
- ▶ `long` für ganze Zahlen beliebiger Größe.
- ▶ `float` für Fließkommazahlen (entspricht `double` in C und Java).
- ▶ `complex` für komplexe (Fließkomma-) Zahlen.

¹Eigentlich mindestens fünf, denn ab Version 2.4 gibt es zusätzlich `Decimal` für finanzmathematische Anwendungen.

²Der Unterschied zwischen Datentypen und Klassen ist in Python vorhanden, aber weniger wichtig als z. B. in Java. Dies ist ein fortgeschrittenes Thema, und in diesem Kurs können wir so tun, als gäbe es keinen Unterschied.

³Weil die vielen Fußnoten nerven, erlaube ich mir, im Folgenden hier und da ein wenig vereinfachend darzustellen...

10 / 22

`int` und `long` (1)

- ▶ `int`-Konstanten schreibt man, wie man es erwartet:
`10`, `-20`, `300`
- ▶ `long`-Konstanten kann man analog zu C und Java schreiben:
`10L`, `-30L`, `9876543210L`
- ▶ Da Python automatisch erkennt, wann eine Konstante zu groß für ein `int` ist, ist das aber nie nötig:

Python-Interpreter

```
>>> 10
10
>>> 9876543210
9876543210L
```

- ▶ Wie in C und Java werden Hexadezimal- und Oktalzahlen notiert:
`0x1ae00`, `0xff`, `0377`

11 / 22

`int` und `long` (2)

- ▶ Man kann den Unterschied zwischen `int` und `long` normalerweise vergessen, da Python automatisch nach `long` umwandelt, wenn ein Rechenergebnis nicht mehr in ein `int` passt:

Python-Interpreter

```
>>> 12345 * 12345
152399025
>>> 12345 * 12345 * 12345
1881365963625L
```

- ▶ `print` gibt das nachgestellte L bei longs nicht mit aus:

Python-Interpreter

```
>>> print 20L
20
>>> print 12345 * 12345 * 12345
1881365963625
```

12 / 22

Rechnen mit int und long

Python benutzt für Arithmetik weitgehend die von C und Java bekannten Symbole:

- ▶ Grundrechenarten: +, -, *, /
- ▶ Modulo: %
- ▶ Potenz (nicht in C/Java): **
- ▶ Boole'sche Bitoperatoren: &, |, ^, ~
- ▶ Bit-Shift: <<, >>

13 / 22

Rechnen mit int und long: Beispiele

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 13 % 8
5
>>> -2 % 8
6 [Unterschied zu C/Java!]
>>> 11 ** 11
285311670611L
>>> 1 << 40
1099511627776L
>>> 1000 & 0xff00ff
232
>>> 21256 ^ (21256 & (21256 - 1))
8
```

14 / 22

Integer-Division: Mit oder ohne Rest? (1)

Traditionell verhält sich die Integer-Division in Python wie in C/Java; der Rest wird ignoriert. Genauer gesagt wird immer abgerundet:

Python-Interpreter

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

Allerdings...

15 / 22

Integer-Division: Mit oder ohne Rest? (2)

... soll sich das Verhalten der Integer-Division in Zukunft ändern. Aus Kompatibilitätsgründen muss man das neue Verhalten zur Zeit noch explizit anfordern:

Python-Interpreter

```
>>> from __future__ import division
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
```

16 / 22

Integer-Division: Mit oder ohne Rest? (3)

In Zukunft (und am besten schon heute) verwendet man für Division mit Abrunden den Operator `//`:

Python-Interpreter

```
>>> 20 // 3
6
>>> -20 // 3
-7
>>> from __future__ import division
>>> 20 // 3
6
>>> -20 // 3
-7
```

17 / 22

Fließkommazahlen und komplexe Zahlen

- ▶ float-Konstanten schreibt man wie in C und Java:
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, aber (sinnvollerweise) keine Bitoperationen.

Wir haben also:

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

18 / 22

Wieviel ist $2 - 2.1$?

Python-Interpreter

```
>>> 2 - 2.1
-0.10000000000000009
```

- ▶ Die meisten Dezimalzahlen können als Fließkommazahlen nicht exakt dargestellt werden.
- ▶ Python-Neulinge finden Ausgaben wie die obige oft verwirrend — dies ist weder eine Schwäche von Python noch die Rückkehr des Pentium-Bugs, sondern völlig normal.
- ▶ Das Ergebnis in C oder Java wäre dasselbe, aber es wird besser vor dem Programmierer versteckt.

`print` rundet das Ergebnis geringfügig und ist daher ansehnlicher:

Python-Interpreter

```
>>> print 2 - 2.1
-0.1
```

19 / 22

Rechnen mit float

Python-Interpreter

```
>>> print 1.23 * 4.56
5.6088
>>> print 17 / 2.0
8.5 [Gemischte Ausdrücke werden nach float konvertiert.]
>>> 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
```

20 / 22

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) ** 0.5
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: negative number cannot be raised to a
fractional power
>>> print (-1+0j) ** 0.5
(6.12303176911e-17+1j)
```

Ausdrücke mit gemischten Typen wie $100L * (1+2j)$ 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`.
- ▶ Bei der Division aus der Zukunft ist das Ergebnis ein `float`.
- ▶ Ist einer der Operanden ein `long` oder passt das Ergebnis der Operation nicht in ein `int`, so ist das Ergebnis ein `long`.
- ▶ Ansonsten ist das Ergebnis ein `int`.