# Programmieren in Python
## 2. Ausgaben und Zahlen

Robert Mattmüller

Albert-Ludwigs-Universität Freiburg

Handlungsplanungs-Praktikum
Wintersemester 2010/2011



## 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`, `float`, `complex`



## Interpreter & Runtime

Das Programm `python3` 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**

```
# python3 beispiel.py
And now for something completely different.
# 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.
>>> print(10 * 10)
100
>>> exit()[Linux: Strg+D]
```



## 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 '
```

## Ausgaben des Interpreters (2)

Zum anderen kann man die `print`-Funktion 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.



## Ausgaben des Interpreters (3)

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

Python-Interpreter

```
>>> print(7 * 6)
42
>>> print("Hello, world")
Hello, world
>>> print(2.8 / 7)
0.4
>>> print("oben\nunten")
oben
unten
>>> print(None)
None
```

Python-Interpreter

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

>>> None
>>>
```

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



## 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.



## 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`, `float`, `complex`

## Zahlen

Python kennt drei verschiedene Datentypen (bzw. Klassen) für Zahlen:

- `int` für ganze Zahlen beliebiger Größe.
- `float` für Fließkommazahlen (entspricht `double` in C und Java).
- `complex` für komplexe (Fließkomma-) Zahlen.



## `int`

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

Python-Interpreter

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

- Hexadezimal-, Oktal- und Binärzahlen werden durch Präfixe `0x`, `0o` bzw. `0b` notiert:

Python-Interpreter

```
>>> 0x1ae00
110080
>>> 0o377
255
>>> 0b101010
42
```



## Rechnen mit `int`

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

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



## Rechnen mit `int`: Beispiele

Python-Interpreter

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

## 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 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 (außer Modulo bei komplexen Zahlen), aber (sinnvollerweise) keine Bitoperationen.
Wir haben also:

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



## 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
```



## 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 gemischten 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`.