Informatik I: Einführung in die Programmierung

2. Erste Schritte in Python

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel

18. & 21. Oktober 2016

1 Allgemeines

Programmiersprachen

Ada, Basic, C, C++, C♯, Cobol, Curry, Fortran, Go, Gödel, HAL, Haskell, Java, Lisp, Lua, Mercury, Miranda, ML, OCaml, Pascal, Perl, Python, Prolog, Ruby, Scheme, Shakespeare, Smalltalk, Visual Basic, u.v.m.

Wir lernen hier Python (genauer Python 3), eine

- objektorientierte,
- dynamisch getypte,
- interpretierte und interaktive
- höhere Programmiersprache.

Die Programmiersprache Python ...

- wurde Anfang der 90er Jahre von Guido van Rossum als Skriptsprache für das verteilte Betriebssystem Amoeba entwickelt;

Foto: Wikipedia

- gilt als einfach zu erlernen, da sie über eine klare und übersichtliche Syntax verfügt;
- wird kontinuierlich von Guido van Rossum bei Google weiter entwickelt.
- bezieht sich auf die Komikertruppe *Monty Python*.

Literatur

Es gibt eine Menge von Lehrbüchern zu Python3. Wir werden im wesentlichen einsetzen

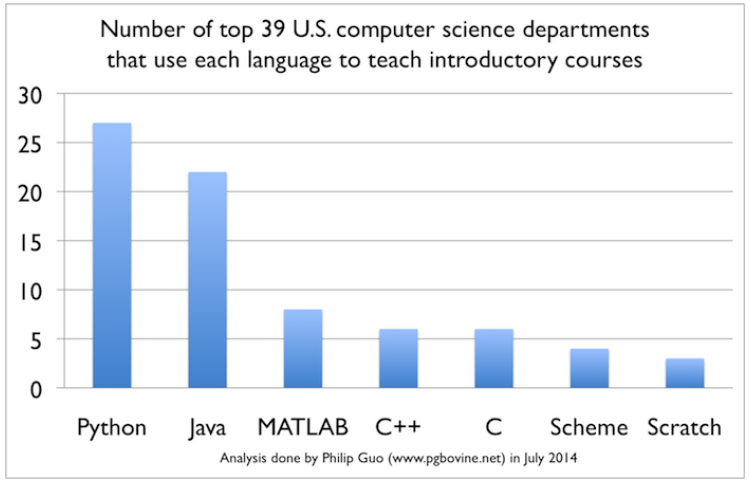
- Allen Downey, *Think Python: How to Think Like a Computer Scientist*, O'Reilly, 2013
- als PDF herunterladbar oder als HTML lesbar (Green Tea Press): `http://greenteapress.com/thinkpython/thinkpython.html`
- als deutsche Version: Programmieren lernen mit Python, O'Reilly, 2013.
- Marc Lutz, *Learning Python*, O'Reilly, 2013 (deutsche Ausgabe ist veraltet!)
- Marc Lutz, *Python kurz & gut*, O'Reilly, 2014 (als Nachschlagwerk)
- Weitere Bücher im Semesterapparat.

2 Warum Python?

Warum Python benutzen?

- Softwarequalität
  - Lesbarkeit
  - Software-Reuse-Mechanismen (wie OOP)
- Programmierer-Produktivität
  - Die Länge von Python-Programmen ist typischerweise weniger als 50% verglichen mit äquivalentem Java oder C++-Programmen.
  - Kein Edit-Compile-Test-Zyklus, sondern direkte Tests
- Portabilität
- Support-Bibliotheken („Batterien sind enthalten“)
- Komponenten-Integrierbarkeit (Java, .Net, COM, Silverlight, SOAP, CORBA, . . .)

Einsteigersprachen in den USA

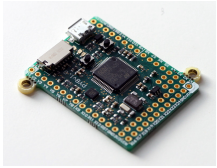


Wer benutzt Python?

- Google: Web search, App engine, YouTube
- Dropbox
- CCP Games: EVE Online
- 2kgames: Civilization IV (SDK)
- Industrial Light & Magic: Workflow-Automatisierung
- ESRI: Für Nutzerprogrammierung des GIS
- Intel, Cisco, HP, Seagate: Hardwaretesting
- NASA, JPL, Alamos: Scientific Computing
- `...http://www.python.org/about/success/`

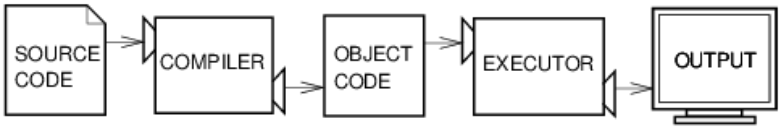
Was geht nicht?

- Python ist langsamer als Java und C++
- Wieviel langsamer? http://benchmarksgame.alioth.debian.org/
- Eignet sich nicht für das Schreiben von Gerätetreibern
- Eignet sich nicht für die Programmierung von eingebetteten Systemen / Mikrocontrollern (*bare metal programming*)
  - Mittlerweile gibt es allerdings MicroPython für einen ARM-Prozessor.



3 Python-Interpreter

Interpreter- versus Compiler-Sprachen



Abbildungen aus Downey 2013

Woher nehmen?

Unter `http://python.org/` findet man aktuelle Dokumentation und Links zum Herunterladen (uns interessiert Python 3.X) für

- *Windows*,
- *MacOSX*,
- *Unixes* (Quellpakete),
- für aktuelle *Linux-Distributionen* gibt es Packages für die jeweilige Distribution, meistens bereits installiert!

Läuft u.a. auch auf dem Raspberry Pi!

Interaktiver und Skript-Modus

Man kann den Python-Interpreter auf folgende Arten starten:

- im interaktiven Modus (ohne Angabe von Programm-Parametern)

→ Man kann interaktiv Ausdrücke und Anweisungen eintippen, der Interpreter wertet diese aus und druckt ggfs. das Ergebnis.

- im Skript-Modus (unter Angabe einer Skript-/Programm-Datei)

→ Ein Programm (auch Skript genannt) wird eingelesen und dann ausgeführt.

4 Interaktives Nutzen der Shell

Erste Schritte: Ausdrücke

Nach Starten des Interpreters erhält man das Prompt-Zeichen, kann Ausdrücke eintippen und erhält ein Ergebnis.

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 '
```

Erste Schritte: „Drucken“

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.

Exkurs: Hello-World-Programme

Hello-World-Programme dienen dazu, eine erste Idee vom Stil einer Programmiersprache zu bekommen.

Python

```
print("Hello World!")
```

Pascal

```
program Hello_World;
begin
  writeln('Hello World!');
end.
```

Brainfuck

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]
>++.>+.+++++++..+++.>++.<<+++++++++++++++.
>.+++.------.--------.>+.>.
```


Ausgaben des Interpreters

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("oben\nunten")
oben
unten
>>> print(None)
None
```

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello world"
'Hello world'
>>> "oben\nunten"
'oben\nunten'

>>> None
>>>
```

Mehr dazu später . . .

Etwas mehr zu `print`

Wir werden die Möglichkeiten von `print` später noch ausführlicher behandeln. Ein Detail soll aber schon jetzt erwähnt werden:

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.

Die Hilfe-Funktion

Wenn Sie etwas zu einem Befehl oder einer Funktion in Python wissen möchten, dann nutzen Sie die help-Funktion:

Python-Interpreter

```
>>> help
Type help() for interactive help, or help(object) for
help about object.
>>> help(print)

Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', ...
```


5 Rechnen

Zahlen

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

- `int` für ganze Zahlen beliebiger Größe (!)
- `float` für Gleitkommazahlen (entspricht in etwa den rationalen Zahlen)
- `complex` für komplexe Gleitkommazahlen.

`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

```
>>> 0x10
16
>>> 0o10
8
```

Rechnen mit `int`

Python benutzt für Arithmetik die folgenden Symbole:

- Grundrechenarten: `+`, `-`, `*`, `/`,
- Ganzzahlige Division: `//`
- Modulo: `%`
- Potenz: `**`
- Bitweise Boolesche Operatoren: `&`, `|`, `^`, $\sim$ (brauchen wir erst einmal nicht)

Rechnen mit `int`: Beispiele

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>> 13 % 8
5
>>> 11 ** 11
285311670611
```


Integer-Division: Ganzzahlig oder nicht?

Der Divisionsoperator `/` liefert das genaue Ergebnis (als `float`). 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
```


Gleitkommazahlen und komplexe Zahlen

- `float`-Konstanten schreibt man mit Dezimalpunkt und optionalem Exponent:
  `2.44`, `1.0`, `5.`, `1.5e+100` (bedeutet $1{,}5 \times 10^{100}$)
- `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.
Wir haben also:

- Grundrechenarten: `+`, `-`, `*`, `/`, `//`
- Potenz: `**`
- Rest bei Division für ganzzahliges Ergebnis: `%`

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
```


Wieviel ist $2 - 2.1$?

Python-Interpreter

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

- Die meisten Dezimalzahlen können als Gleitkommazahlen 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.

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 wird der andere zu `complex` konvertiert (falls er das nicht schon ist).
- Ist einer der Operanden ein `float` (und keiner ein `complex`), so wird der andere zu `float` konvertiert (falls er das nicht schon ist).

Überläufe und Unterläufe

- Im Gegensatz zu anderen Programmiersprachen können in Python ganze Zahlen beliebig groß (und klein) werden.
- Gleikommazahlen haben aber eine beschränkte Darstellung (IEEE 754 Standard) von meist 64 Bit.

Python-Interpreter

```
>>> 1e-999
0.0
>>> 1e+999
inf
>>> 1e+999 - 1e+999
nan
```

`inf` steht für *infinity* und `nan` für *not a number*. Mit beiden kann man weiter rechnen.

Zusammenfassung

- Python ist ein objektorientierte, dynamisch getypte, interpretierte und interaktive höhere Programmiersprache.
- Python wird immer populärer und wird in den USA als die häufgste Anfängersprache genannt.
- Python läuft auf praktisch allen Maschinen und Betriebssystemen.
- Es gibt drei numerische Typen in Python: `int`, `float`, und `complex`.
- Es werden alle bekannten arithmetischen Operationen unterstützt.