Python - eine gute Wahl als Unterrichtssprache

Teil 1: Die Grundlagen

Gregor Lingl

0. Einleitung

Eine ganz wesentliche Bemerkung hat Erich Neuwirth kürzlich in einer Diskussion im Kustodenforum gemacht, als er sagte, dass das erste Werkzeug im Programmierunterricht deswegen von hoher didaktischer Bedeutung ist, weil es den Denkstil prägt. Daran anknüpfend möchte ich nun zeigen, dass die Programmiersprache Python besonders gut als erstes Werkzeug geeignet ist.

Darüber hinaus ist sie als moderne objektorientierte Skriptsprache praktisch für den Unterricht der meisten Programmierkonzepte, die im EDV-Unterricht für 15- bis 19-jährige sinnvoll behandelt werden können, außerordentlich gut geeignet.

Guido van Rossum, der Erfinder von Python schreibt über den Ursprung von Python: "*Als ich Python erschuf habe ich es nicht als Sprache für Lehrer entworfen: ich hatte ein Werkzeug für faule Programmierer wie mich vor Augen. Aber schon bald danach ... erfuhr ich von Programmierern, die ihren Kindern ... erfolgreich Python beibrachten. Das ist eigentlich gar nicht überraschend. Python verwendet viele Ideen von ABC, einer Lehrsprache, die zehn Jahre zuvor entstanden ist ...*"

Ich möchte zunächst die didaktisch relevanten Vorzügen von Python zusammenfassen:

- Python hat eine äußerst klare Syntax. Wesentliches Element ist, dass in Python Blöcke durch Einrückung (also durch *"white space"*) definiert werden. Dies ist ein kontroversieller Zug, der jedoch dazu führt, dass Python-Programme gut strukturiert geschrieben werden müssen, damit sie funktionieren. Und daher auch jedenfalls leicht lesbar sind. Python wurde deshalb auch schon als *"executable pseudocode"* bezeichnet - im Gegensatz zu mancherorts sehr verbreitetem *"executable line-noise"*.
- Python gestattet zwanglos die Anwendung unterschiedlichster Programmierstile - prozedural, funktional, objektorientiert, ... Der Programmierangfänger kann einfache Probleme mit kleinen simplen Programmen lösen, wie der Profi mit ganz fortgeschrittenen Verfahren neue Klassenbibliotheken erstellen kann ...
- Das obwohl Python von Anfang an (und nicht erst in Version 6, 7 oder wer weiß welcher) als objektorientierte Sprache konzipiert wurde: "*In Python ist alles ein Objekt*". Dabei sind solche modernen Konzepte wie Operator-Überladung und Mehrfachvererbung realisiert.
- Darüber hinaus stellt Python mit Listen und Dictionaries (Hash-Tables, assoziativen Feldern) äußerst leistungsfähige Datentypen, d. h. Klassen, bereit (von denen darüber hinaus der Benutzer auch noch eigene Klassen ableiten kann!).
- Python hat einen interaktiven Interpreter, der - ähnlich wie es in Logo oder Lisp funktioniert - hervorragend dazu geeignet ist, das Funktionieren der Sprache experimentell zu untersuchen. Das kommt dem Forschungsdrang der SchülerInnen entgegen, fördert deren Selbsttätigkeit (im Gegensatz zum (beinahe-)Abtippen von Programmen) und führt darüber hinaus zu häufigen Erfolgserlebnissen. Der interaktive Interpreter eignet sich auch hervorragend als "ad-hoc"-Werkzeug für Instant-Lösungen kleiner Probleme.
- Python kommt mit einer einfachen Entwicklungsumgebung namens **IDLE** mit Interpreter im Direktmodus, Editor mit *syntax-colouring* und einem ansehnlichen, aber einfach anzuwendenden Debugger.
- Python kommt mit einem Toolkit zum Bau von GUI-Programmen: **Tkinter**. GUI's können sogar im "Direktmodus" Schritt für Schritt zusammengesetzt werden.
- Python kommt mit einer leistungsfähigen Klassenbibliothek, die die einfache Konstruktion unterschiedlichster Objekte bis hin zu Webservern gestattet. Die Programmierung von Chat-Programmen oder Netzwerkspielen wird mit diesen Klassen zur relativ einfachen Übung. Ebenso gibt es in dieser Bibliothek leistungsfähige Werkzeuge zur CGI-Programmierung. (Apache kann Python als Skriptsprache unterstützen).
- Python kommt aber auch mit einem einfachen Turtle-Grafik-Modul, mit dem besonders die grundlegenden Programm-Konstrukte schön visualisiert werden können. (Diesen Weg habe ich im ersten Teil von "Python für Kids" beschritten.) Sie können auch mehrere Turtles verwenden. Turtles sind zwar keine bunten Bälle, sie lassen sich aber dafür im Direktmodus Schritt für Schritt steuern (und brauchen keine Koordinaten-Geometrie). Scheinbarer Zeitverlust durch die Oberfläche tritt gar nicht erst auf.
- Darüber hinaus gibt es zu Python eine Unzahl von *"third-party"*-Modulen, mit denen einfach fortgeschrittene Anwendungen erstellt werden können. Beispielsweise Spiele mit dem "Pygame"-Modul oder 3D-Animationen mit "VPython".
- Python ist plattform-unabhängig. Python-Programme laufen auf Windows-, Linux-, Mac-Systemen (wenn sie nicht spezielle systemspezifische Module verwenden.)
- Mark Hammond hat eine Win32-Extension für Python geschrieben, mit der beispielsweise auf einfache Weise Microsoft-Office-Komponenten - etwa Spreadsheets - in Python - Programm eingebunden und genutzt werden können. Dieses Paket enthält eine zur IDLE alternative leistungsfähigere Entwicklungsumgebung für Windows.
- All dies ist gratis und jederzeit für jeden in der neuesten Version mit vollständiger Dokumentation übers Internet erhältlich. Denn Python gehört (wie Linux) zum *open-source*-Bereich.
- Mit Python landen Sie in einem nicht-kommerziellen sozialen Environment. Erstaunlich ist die gegenseitige Unterstützung, die Python-Anwender in verschiedenen Mailing-Listen einander leisten. Python-Tutor (`tutor@python.org`) ist ein besonders gutes Beispiel dafür, das speziell für Neulinge, die Python lernen wollen gedacht ist. Die Erfahrung, Hilfe von Gleichgesinnten aus aller Herren Länder zu erhalten und diesen auch zurückzugeben kann meines Erachtens besonders für junge Leute von unschätzbarem Wert sein.
- Die Entwicklung von Python als *open-source*-Produkt ist für jeden, der es wünscht, 100%ig transparent. Ich fand es sehr spannend, den Prozess zu verfolgen, in dem von zahllosen Mitgliedern der Python-Community Vorschläge für die Erweiterung und/oder Änderung der Sprache gemacht, öffentlich zur Diskussion gestellt und dann vom Entwicklerteam begründet angenommen oder abgelehnt wurden. (Jüngst wurde über einen bestimmten Punkt sogar eine öffentliche Abstimmung in der Python-Community durchgeführt.)
- Probieren Sie es einmal aus - Sie werden sehen: *"Have fun!"* bleibt mit Python nicht nur ein Slogan.

Ich möchte Ihnen Python in einer fünfteiligen Serie vorstellen. Ich plane folgende Inhalte für die fünf Teile:

(a) Die Grundlagen
(b) Objekte in Python
(c) GUI-Programmierung
(d) Beispiele für Anwendungen der Klassenbibliothek
(e) Python-Erweiterungen aus der *open source community*: Spiele, 3D-Programmierung, Jython, Python & Win32

1. Getting started (Loslegen!)

In die heutigen Folge möchte ich Ihnen zunächst die Sprachgrundlagen soweit vorstellen, dass Sie Ihre Grundkenntnisse aus anderen Sprachen in Python übertragen können.

Die jeweils aktuellste Version (derzeit Python 2.3) können Sie von der Python-Website `http://www.python.org` herunter laden. Die wichtigsten Bestandteile des Python-Systems, die Sie nach der Installation vorfinden, sind der Python-Interpreter, die Entwicklungsumgebung IDLE

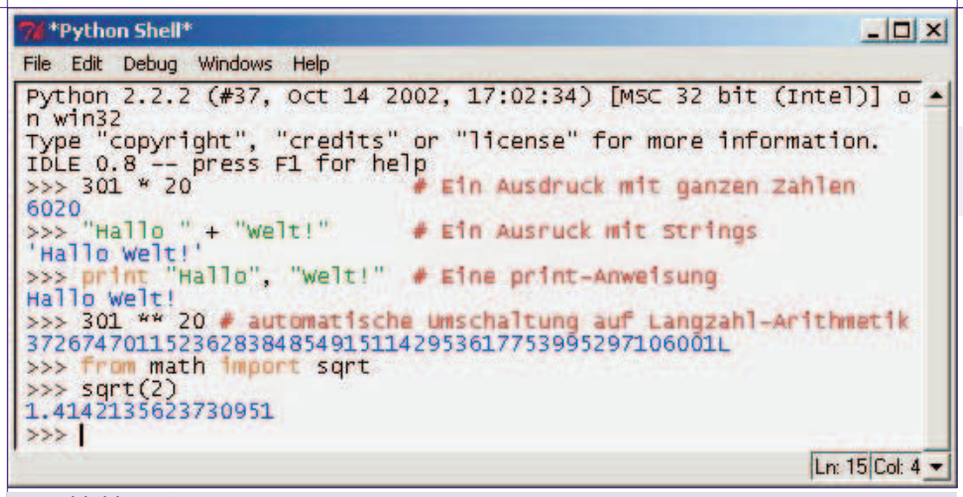

Abbildung 1

und die Dokumentation im HTML-Format.

Um Python kennenzulernen, ist es am bequemsten, aus dem Startmenü ***Python|IDLE*** zu starten. Die IDLE meldet sich mit einem Shell-Fenster, das den interaktiven Python-Interpreter bereitstellt. Dieser meldet sich mit der Eingabeaufforderung

```
>>>
```

Er ist ideal dazu geeignet, Python zu erforschen. Sie können ihm beliebige Python-Ausdrücke oder Python-Anweisungen eingeben. Diese werden vom Interpreter sofort ausgewertet beziehungsweise ausgeführt (***Abbildung 1***).

2. Einfache Datentypen

Wir sehen hier die nummerischen Datentypen `int`, `long` und `float` in Verwendung sowie den Datentyp `string`. Um "höhere mathematische Operationen" auszuführen, müssen diese aus dem Modul `math` importiert werden. Dies geschieht mit der `import`-Anweisung: hier wird die `sqrt`-Funktion importiert. `from math import *` importiert alle Funktionen aus dem Modul `math`.

Um herauszufinden, was ein Modul enthält, ruft man den Menüeintrag ***Help|Python Documentation*** ... auf und klickt zunächst auf den Link ***Global Module Index***, dann auf den Namen des gewünschten Moduls: ***math***. Man erhält eine Liste der enthaltenen Funktionen.

Aufschluss über den Typ von Python-Objekten gibt die eingebaute Funktion `type`:

```
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>> type(301**20)
<type 'long'>
```

Darüber hinaus gibt es in Python auch noch den numerischen Datentyp `complex`.

```
>>> (3+2j)*(1-1j)
(5-1j)
```

Als arithmetische Operatoren gibt es `+`, `-`, `*`, `/`, `//` und `%` (Modulo-Operation). Bezüglich der Ergebnisse von Divisionen ist zu beachten, dass in der aktuellen Version von Python `/` als Ganzzahl-Division arbeitet, wenn beide Operanden ganzzahlig sind:

```
>>> 17 / 3
5
>>> 17.0 / 3
5.666666666666667
>>> 17 // 3
5
>>> 17 % 3
2
```

Variablen müssen (und können) in Python nicht deklariert werden und werden durch Wertzuweisungen mittels des Zuweisungsoperators `=` erzeugt. Mehrfache Zuweisungen in einer Anweisung sind möglich:

```
>>> a = 14
>>> b, c, d = 3.0, 4.0, a / 2.0
>>> d
7.0
>>> a, b, c
(14, 3.0, 4.0)
```

3. Sequenzen

Python kennt drei Datentypen, die als Sequenzen bezeichnet sind, und deren Elemente durch ganze Zahlen indiziert werden können. Strings, Listen und Tupel. Die Indizierung beginnt immer mit 0:

```
>>> wort = "Python"
>>> wort[0]
'P'
>>> len(wort)
6
>>> woerter = ["a", "be", "bu", "und",
"draußt", "bist", "du"]
>>> woerter[2]
'bu'
>>> len(woerter)
7
>>> zahlen = (1, 1.0/2, 9.999)
>>> len(zahlen)
3
>>> zahlen[1]
0.5
```

Nur Listen können, z. B. durch Zuweisung an Elemente verändert werden:

```
>>> woerter[4] = "drin"
>>> woerter
['a', 'be', 'bu', 'und', 'drin', 'bist',
'du']
>>> zahlen[1] = 2.0
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in ?
    zahlen[1] = 2.0
TypeError: object doesn't support item
assignment
```

Spezielle Listen ganzer Zahlen werden durch die eingebaute Funktion **`range`** erzeugt:

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(4,7)
[4, 5, 6]
>>> range(2,13,3)
[2, 5, 8, 11]
```

4. Kontrollstrukturen

Über die Elemente eine Sequenz kann mit der for-Schleife iteriert warden:

```
>>> for zahl in range(4):
        quadrat = zahl ** 2
        print zahl, "zum Quadrat ist",
quadrat
0 zum Quadrat ist 0
1 zum Quadrat ist 1
2 zum Quadrat ist 4
3 zum Quadrat ist 9
```

Diese Beispiel zeigt die allgemeine Struktur zusammengesetzter Anweisungen, also von Anweisungen, die einen Block von zusammengehörigen Anweisungen enthalten.

In Python wird ein Block nicht durch Klammern oder Schlüsselwörter wie begin/end gekennzeichnet, sondern indem die zum Block gehörigen Anweisungen gleich weit eingerückt werden. In Python hat also leerer Raum (*"white space"*) syntaktische Funktion!

Dies führt notwendigerweise zu klar strukturierten Programmen. Die IDLE kennt Pythons Syntax und schlägt daher passende Einrückungen vor.

Die `for`-Anweisung führt den Schleifenblock für jedes Element der Sequenz aus. Der Kopf der Schleife muss, wie jeder Kopf einer zusammengesetzten Anweisung, durch einen Doppelpunkt abgeschlossen werden. Ein weiteres Beispiel, das gleich auch eine Anwendung der `if`-Anweisung zeigt:

```
>>> for wort in woerter:
        if 2<=len(wort)<=3:
            print wort,
be bu und du
```

Die `if`-Anweisung kennt auch optionale `else-if`-Zweige, in Python durch das Schlüsselwort `elif` gekennzeichnet, und einen optionalen `else`-Zweig.

Wir zeigen das, indem wir nun ein erstes Python-Programm, ein Script, schreiben. Dazu öffnen wir von der IDLE aus ein neues Fenster über ***File|New window*** und schreiben folgenden Text hinein:

```
from math import sqrt
print "Loesung quadratischer Gleichung:"
a, b, c = input("Koeffizienten: ")
determinante = b**2 - 4*a*c
if determinante > 0:
    d = sqrt(determinante)
    x1, x2 = (-b+d)/(2.0*a), (-b-d)/(2.0*a)
elif determinante == 0:
    x1 = x2 = -b/(2.0*a)
else:
    x1, x2 = "unloesbar!", ""
print "Loesungen", x1, x2
```

Dann speichern wir diesen Text als Datei unter dem Namen **`quagl.py`** ab und führen im Editor-Fenster den Menübefehl ***Edit|Run script*** aus. Dies führt etwa zu folgendem Dialog im Shell-Fenster:

```
>>>
Loesung quadratischer Gleichung:
Koeffizienten: 4, -8, 3
  # Kursiv:
  # Benutzereingabe
Loesungen 1.5 0.5
>>>
```

Erläuterungen

- die eingebaute Funktion `input()` dient der Eingabe von Python-Ausdrücken. Zahlen, aber auch Tupel wie 4, -8, 3 sind gültige Python-Ausdrücke.

- Zur Einrückung der "Blöcke" in der `if-elif-else`-Anweisung. Zwei davon bestehen nur aus einer einzigen Anweisung.
- der `if`-Block und der `else`-Block enthalten eine Parallel-Zuweisung, der andere Block enthält eine mehrfache Zuweisung.
- Prüfung auf Gleichheit geschieht mit `==`, fortlaufende Vergleiche wie `2<=len(wort)<=3` sind möglich.

5. Funktionsdefinitionen

Funktionsdefinitionen werden in Python mit dem Schlüsselwort `def` eingeleitet. Funktionen können natürlich Parameter haben und daher Argumente übernehmen. Wie probieren wieder ein ganz einfaches Beispiel mit dem Interpreter aus. Achten Sie wieder auf die gleichmäßige Einrückung des Funktionsblocks:

```
>>> def doppelt(x):
        d = 2 * x
        print "Das Doppelte von", x,"ist",d
>>> doppelt(5)
Das Doppelte von 5 ist 10
```

`doppelt` funktioniert wie eine Prozedur. Angemessener ist hier vielleicht ein "echte" Funktion mit Rückgabewert. Dazu benötigen wir die `return`-Anweisung:

```
>>> def doppelt(x):
        return 2 * x
>>> doppelt(5)
10
>>> doppelt(doppelt(5))
20
```

Der letzte Aufruf wäre natürlich mit der ersten "Prozedur"-Version nicht möglich gewesen.

Nützen wir diese Möglichkeit, um unser Script für die Lösung quadratischer Gleichungen zu ändern, indem wir eine Funktion zu definieren. (Im Edit-Fenster hilft dabei ***Edit|Indent region***.)

```
from math import sqrt
def quagl(a,b,c):
    determinante = b**2 - 4*a*c
    if determinante > 0:
        d = sqrt(determinante)
        return (-b+d)/(2.0*a),(-b-d)/(2.0*a)
    elif determinante == 0:
        return -b/(2.0*a)
    else:
        return ()
```

Nun können wir quadratische Gleichungen mittels Funktionsaufrufen lösen:

```
>>> quagl(4,-8,3)
(1.5, 0.5)
>>> quagl(1,2,1)
-1.0
>>> quagl(1,0,4)
()
```

Dabei sehen wir, dass Python-Funktionen mehrere Werte (als Tupel) zurückgeben können.

Neben der `for`-Schleife kennt Python auch die `while`-Schleife, deren Syntax nach dem bisher gesagten eigentlich keiner weiteren Erläuterung bedarf. Sie lässt sich beispielsweise im Euklidischen Algorithmus für die Berechnung des größten gemeinsamen Teilers zweier Zahlen anwenden. Mittels Parallelzuweisung lässt sich der in Python sehr elegant formulieren.

```
>>> def ggt(a,b):
        while b>0:
            a,b = b,a%b
        return a
>>> ggt(105, 42)
21
```

6. Funktionen als Objekte

Zuletzt möchte ich noch zeigen, dass sich Funktionen (in Python sind Funktionen auch Objekte) als Argumente an andere Funktionen übergeben lassen:

```
>>> def tabelle(von, bis, fun):
        for x in range(von,bis+1):
            print x, fun(x)
>>> tabelle(13,16,doppelt)
13 26
14 28
15 30
16 32
>>> def cubus(n):
        return n**3
>>> tabelle(27,29,cubus)
27 19683
28 21952
29 24389
>>>
```

In der nächsten Folge möchte ich zeigen, was die Feststellung bedeutet: "*In Python ist alles ein Objekt*", und etwas OOP treiben: wie man in Python selbst Klassen definieren und Objekte erzeugen kann.

7. Quellen

Wenn Sie sich etwas näher mit Python beschäftigen möchten, können Sie dafür viele Ressourcen in Anspruch nehmen.

(1) Vielfältig und gratis ist alles, worauf auf der Python-Website "for beginners" verwiesen wird: `http://www.python.org/doc/Newbies.html`. Für das meiste davon sind Englischkenntnisse erforderlich.

An deutschsprachigen Büchern bieten sich an:

(2) Mark Lutz und David Ascher: Einführung in Python, OReilly, 2000 (Englische Neuauflage in Vorbereitung.)

(3) Gregor Lingl: Python für Kids, mitp, 2003. Gedacht als Einführung ins Programmieren für Anfänger, unter Verwendung von Python. Siehe dazu auch: `http://python4kids.net`

(4) von Löwis/Fischbeck: Python 2, Addison-Wesley, 2001. Ist keine Einführung, sondern anspruchsvoll und tiefergehend, enthält aber noch nicht die Neuerungen von Python seit Version 2.1.

Für Ambitionierte dürfte derzeit am aktuellsten und ergiebigsten sein:

(5) Alex Martelli: Python in a Nutshell, OReilly, 2003. (Englisch)