Python Tutorial

Geschrieben am 5.5.2002
Von
Sebastian Schäfer
sschaef@cs.uni-frankfurt.de

Inhaltsverzeichnis

0 Einleitung

Das Tutorial entstand im Rahmen eines Praktikums im SS2002. Es stellt eine Einführung in die Programmiersprache Python dar und setzt grundsätzliches Wissen in Bezug auf imperative Programmiersprachen voraus. Vorwissen in anderen Sprachen (Pascal, C, Basic, ...) ist nicht unbedingt erforderlich, allerdings sollte man durchaus wissen, was z.B. eine for-Schleife macht. Das Tutorial orientiert sich im groben an dem Buch „Python Referenz“ von David M. Beazley, erschienen im Markt+Technik Verlag.

Python ist eine relativ junge Sprache. Sie ist eine Mischung aus Pascal, C und Basic, die sehr anfängerfreundlich ist und auch für erfahrene Programmierer noch eine Menge zu bieten hat. Einen Python-Interpreter kann man kostenlos von http://www.python.org/ für alle gängigen Betriebssysteme downloaden. Die Programme können mit einem einfachen Texteditor eingegeben werden. Gerade unter Windows jedoch gibt es eine kostenlose IDE, welche ich sehr empfehlen kann: ActiveStatePython, kostenlos zu beziehen unter http://www.activestate.com/.

Befehle stelle ich grundsätzlich mit einer andern Schrift dar, Ausgaben des Programms sind durch ein vorangestelltes größer Zeichen erkennbar:

```
print „Hello World“
> Hello World
```

1. Einführung

1.1 Python

Wenn man Python startet, gelangt man in den Laufzeitinterpreter. Diesen erkennt man an der Eingabeaufforderung „>>>“. Jeder Befehl, den man eingibt, wird direkt ausgeführt, Variablen werden sogar gespeichert. Möchte man mit Python direkt ein Programm starten, so muß man den Programmnamen als Parameter übergeben:

```
Python programmname.py
```

Man kann ein Programm auch in den interaktiven Modus laden. Wenn man dies macht, wird das Programm nicht sofort ausgeführt, sondern nur die Funktionen, Definitionen etc. in Python geladen und man kann die Funktionen einfacher Testen, indem man z.B. die zu testende Funktion mit unterschiedlichen Parametern aufruft. Das ein Programm in den interaktiven Modus geladen werden soll, erreicht man mit Hilfe des Schalters „-i“:

```
Python –i programmname.py
```

Kommentare werden durch das Gatter `#` eingeleitet.

1.2 Variablen und andere Eigenheiten

Variablen müssen in Gegensatz zu etwa C oder Pascal nicht ausdrücklich deklariert werden. Man darf nur keine reservierten Wörter benutzen, ist aber ansonsten völlig frei in der Wahl des Variablennamens – nur muß er mit einem Buchstaben beginnen. Es versteht sich natürlich von selbst, dass man Variablen immer ihrem Sinn nach benennt (Das ist in wohl allen Programmiersprachen so, bei denen man sich die Variabelennamen aussuchen kann). Dadurch, dass man die Variablen nicht deklarieren muss, gibt es auch kaum Typenfehler (es sei denn, eine Funktion braucht explizit einen Variablentyp, wie z.B. die `mod` Funktion).
Befehlsblöcke werden in Python nicht extra eingeleitet, sie werden eingerückt geschrieben. In C braucht man die geschweiften Klammern {}, in Pascal geschieht dies durch `begin ... end;`, Python benutzt dafür Leerzeichen bzw. Tabstopps. Das dient nicht nur der Leserlichkeit (in anderen Programmiersprachen muss sich der Programmierer dazu „zwingen“), sondern ist auch noch praktisch für ihn, er muss weniger tippen. Ein Beispiel dazu folgt im nächsten Abschnitt.

1.3 Vergleiche

Vergleiche werden, wie man es aus anderen Sprachen kennt mit dem `if ... else` Statement durchgeführt. Allerdings gibt es kein `then`. Ein einfacher Vergleich sieht in Python wie folgt aus:

```
if a == b:
     print "a und b sind gleich"
elif a < b:
     print "a ist kleiner als b"
else:
     print "a ist größer als b"
```

Wie man oben sehen kann, prüft man Gleichheit mit einem doppelten Gleichheitszeichen (==). Sehr interessant ist auch das `elif`, welches für ein herkömmliches `else if` steht.

1.4 Listen

Eine Liste ist eine Sequenz von einzelnen oder mehreren Objekten. Eine Liste kann sogar aus unterschiedlichen Objekten bestehen, wie das folgende Beispiel zeigt:

```
a = [1, 2.321,  „b"]
```

Auf die einzelnen Elemente kann man per Index zugreifen, wobei die 0 für das erste Element steht. Wenn man alle Elemente einer Liste durchlaufen will, bietet es sich an, dies mit einer for-Schleife zu tun:

```
a = [1, 2, 3, 4, 5]
for i in a:
     print i
> 1
> 2
> 3
> 4
> 5
```

Hierbei kann man gleich eine Besonderheit der for-Schleife unter Python erkennen: man iteriert immer über eine Liste. Die Schleifen- oder Indexvariable nimmt dabei immer den Wert des aktuellen Objektes an. Eine andere Möglichkeit, alle Elemente der Liste a auszugeben, wäre folgende Schleifenkonstruktion:

```
a = [1, 2, 3, 4, 5]
for i in range(len(a)):
     print a[i]
> 1
> 2
> 3
> 4
> 5
```

Die Funktion `range(n)` erzeugt eine Liste der Länge n, beginnend mit 0 (im Falle n=5: [0, 1, 2, 3, 4]). `len(x)` gibt die Länge der Liste x zurück. Somit gibt man der Reihe nach folgende Elemente aus: a[0], a[1], ... , a[4].

Python bietet aber noch andere Möglichkeiten als das pure nennen eines einzelnen Indizes (Alle Indexangaben beziehen sich auf den Listenindex (0 → n-1):

`l[i]`	Gibt das i-te Element zurück
`l[n:m]`	Gibt das n-te bis m-te Element zurück
`l[:n]`	Gibt das erste bis n-te Element zurück
`l[n:]`	Gibt die Liste ab dem n-ten Element zurück
`l[n:-m]`	Gibt ab dem n-ten Element alle bis zum m-letzten Element zurück
`l[:-n]`	Gibt die Liste bis zum n-letzten Element zurück

Tabelle 1, Indizierung bei Listen

Listen sind in Python Objekte, die mehrere Methoden besitzen, die die Arbeit mit Listen deutlich vereinfachen. Hier ist eine Übersicht der wichtigsten Attribute:

`l.append(x)`	Fügt x zur Liste l hinzu
`l.count(x)`	Zählt das Auftreten von x in der Liste l
`l.index(x)`	Gibt den Index des ersten Auftretens von x in l zurück
`l.extend(x)`	Erweitert l um die Liste x
`l.insert(i, x)`	Fügt x am Index i in Liste l ein
`l.remove(x)`	Entfernt alle x aus Liste l
`l.pop([i])`	"popt" im Stacksinne das Element i aus Liste l
`l.reverse()`	Dreht die Liste l um
`l.sort([cmpfunc])`	Sortiert Liste l [anhand der Funktion cmpfunc]

Tabelle 2, Methoden von Listen

Interessant ist auch die Tatsache, dass der `'+'` Operator Listen verbinden kann:

```
a = [1, 2, 3]
b = [4, 5]
c = a + b
print c
> [1, 2, 3, 4, 5]
```

In Python gibt es auch sogenannte verschachtelte Listen, d.h. dass ein Listenobjekt selbst eine Liste sein kann:

```
a = [1, 2, [3, 4, [5, 6]]]
print a[2]
print a[2][2]
print a[2][2][1]
> [3, 4, [5, 6]]
> [5, 6]
> 6
```

Anmerkung:
Man kann NICHT wie in anderen Programmiersprachen mit Kommata bei verschachtelten Listen Arbeiten:

```
a = [1, 2, [3, 4, [5, 6]]]
print a[2, 2, 1]
➔ FEHLER!!!!!
```

1.5 Tupel

Tupel unterstützen fast alle Methoden (Indizierung, Verkettung, ...) von Listen. Sie werden nicht mit einer eckigen Klammer, sondern mit einer normalen Klammer erzeugt:

```
a = (1, 2, 3)
b = (4, 5)
c = a + b
print c
> (1, 2, 3, 4, 5)
```

Tupel sind nach ihrer Erzeugung allerdings nicht mehr veränderbar. Es können auch keine Elemente hinzugefügt werden. Sie werden benutzt um z.B. mehr als nur einen einzigen Wert bei einer Funktion zurückzugeben.

1.6 Strings

Ein String ist im Prinzip eine Liste von Chars, auf die man genauso wie auf Listen zugreifen kann. Ein String kann zwischen Hochkommata (') oder zwischen Anführungszeichen (") stehen. Hat man sich jedoch für ein Zeichen entschieden, muß man den String auch mit dem selben Symbol wieder abschließen:

```
a = 'falscher String" # hier gibt's auf jeden Fall eine
                      # Fehlermeldung
a = 'korrekter String'
a = "korrekter String"
a = "korrekte Verknuepfung" + ' von Strings'
print 'das ist manchmal sehr Hilfreich: "test"'
> das ist manchmal sehr Hilfreich: "test"
```

Gerade die letzten zwei Beispiele sind sehr interessant.
Das erste Beispiel zeigt, dass man String-Literale mit dem selben Zeichen beginnen und beenden muss, unterschiedlich „eingeschlossene" Strings werden jedoch völlig gleichwertig behandelt.
Im zweiten Beispiel kann man sehen, dass innerhalb eines „Hochkommata-Strings" das Anführungszeichen wie ein Bestandteil des Strings betrachtet wird. Das kann praktisch sein, wenn man z.B. eine Webseite mit einem Pythonprogramm generieren will (in HTML werden Linkziele in Anführungszeichen eingeschlossen). So erklärt sich auch, warum das erste Beispiel einen Fehler erzeugt: Der String wurde nicht abgeschlossen!

Strings können, wie oben gesehen, mit dem `'+'` Operator verbunden werden. Will man einen Zahlenwert als String mit einem anderen String verknüpfen, muss man diesen erst Umwandeln. Für eingebaute Typen erledigt dies die `str()` Funktion:

```
pi = 3.141592626 #u.s.w. ….
print "Pi ist " + str(pi)
> Pi ist 3.141592626
```

Alternativ zum `str()` Befehl kann auch der `repr()` Befehl benutzt werden. Dies ist eine besondere Eigenschaft von Objekten.

1.7 Dictionaries

Dictionaries sind sogenannte assoziative Felder. Objekte werden mit Hilfe von Schlüsseln indiziert. So kann man z.B. das Auftreten eines Schlüssels elegant verwalten, indem man dem Schlüssel den entsprechenden Wert zuordnet und diesen dann hochzählt:

```
a = {}
a["dummy"] = 0
```

Die geschweifte Klammer erzeugt ein leeres Dictionary. Die zweite Anweisung weist dem Schlüssel `"dummy"` den Wert 0 zu. Existiert dieser Schlüssel noch nicht, so wird er erzeugt und ihm der Wert 0 zugewiesen. Ist in dem Dictionary schon der Schlüssel enthalten, so wird ihm der Wert 0 zugewiesen.
In ein Dictionary können sowohl die Schlüssel, als auch die Werte völlig voneinander unterschiedliche Typen besitzen:

```
a = {}
a["dummy"] = 10
a[123] = "asd"
print a

> {123: 'asd', 'dummy': 10}
```

Jedes unveränderliche Objekt kann ein Schlüssel eines Dictionaries sein.
Auch ein Dictionary hat mehrere eingebaute Methoden, die die Arbeit sehr Erleichtern können:

`len(d)`	Gibt die Anzahl der Schlüssel des Dictionaries d aus
`d[x]`	Gibt das dem Schlüssel x zugeordnete Objekt zurück
`del d[x]`	Entfernt den Schlüssel x aus dem Dictionary d
`d.clear()`	Löscht alle Schlüssel aus d
`d.copy()`	Gibt eine Kopie con d zurück (keine Referenz!)
`d.has_key(x)`	Überprüft, ob der Schlüssel x in d gespeichert ist
`d.keys()`	Gibt eine Liste mit allen Schlüsseln aus d zurück
`d.values()`	Gibt eine Liste mit allen Werten aus d zurück
`d.items()`	Gibt eine (Schlüssel, Wert) Tupel Liste zurück
`d.update(e)`	Überträgt alle Objekte von e nach d

Tabelle 3, Operationen und Methoden auf Dictionaries

1.8 Dateioperationen

Mit Python ist es relativ einfach, Dateien zu öffnen, zu lesen oder zu schreiben. Dies wird am einfachsten an einem kleine Beispielprogramm deutlich:

```
f = open("input.txt")
g = open("output.txt", "w")
lines = f.readlines()
f.close()
for i in lines:
     g.write(i)
g.close()
```

Mit `open(Dateiname, [Zugriffsmodus])` erzeugt man ein neues Dateiobjekt, welches in diesem Beispiel f und g heißt. Dieses Dateiobjekt hat jetzt mehrer Methoden:

`f.read([n])`	Liest maximal n Bytes aus f
`f.readline([n])`	Liest eine einzelne Zeile aus f [wenn n angegeben ist max. n Bytes]
`f.readlines([n])`	Liest n Zeilen aus f [wenn n angegeben ist max. n Bytes]
`f.write(s)`	Schreibt den String s in f
`f.writelines(l)`	Schreibt eine Liste l von Strings in f
`f.tell()`	Gibt die aktuelle Dateiposition zurück
`f.seek(offset[, where])`	Sucht eine neue Dateiposition
`f.flush()`	Schreibt den Ausgabepuffer
`f.close()`	Schließt die Datei
`f.closed`	Gibt an, ob die Datei geschlossen ist
`f.mode`	Gibt an, in welchem Modus die Datei geöffnet wurde
`f.name`	Gibt den Name der Datei an

Tabelle 4, Methoden und Attribute des Dateiobjektes

1.9 Funktionen

Wie in anderen Programmiersprachen auch werden Funktionen in Python mit einem speziellen Schlüsselwort eingeleitet: `def`. Im Gegensatz zu anderen Programmiersprachen gibt es in Python jedoch keine Prozeduren, die entsprechenden Funktionen haben nur keinen Rückgabewert. Eine Funktion gibt mit Hilfe des `return` Befehls einen oder mehrere Werte zurück:

```
def quad(x):
     return x * x

def quadpair(x):
     return (x, x * x)

print quad(2)
a, b = quadpair(2)
print str(a) + " --> " + str(b)
> 4
> 2 --> 4
```

Während die erste Funktion nur das Quadrat zurückgibt, gibt die zweite Funktion ein Tupel zurück. Das Aufrufbeispiel des Hauptprogrammes zeigt, wie man ein solches Tupel auswerten kann.
Auch sehr interessant ist die Möglichkeit, ein Standardwert für ein Parameter zu vergeben. Dieser Parameter muss dann nicht explizit übergeben werden:

```
def connect(hostname, port, timeout = 300)
...

connect("www.python.org", 80)
connect("www.python.org", 80, 20)
```

Im ersten Aufruf der Funktion connect ist timeout nicht angegeben und hat somit innerhalb der Funktion den Wert 300. Bei dem zweiten Beispiel wird timeout übergeben und hat den Wert 20. Eine formale Beschreibung des Funktionskopfes sähe so aus:

```
connect(hostname, port [, timeout=300])
```