Python, eine moderne Programmiersprache

Inhaltsverzeichnis

1 Was ist Python?

Python ist eine moderne Programmiersprache, die sich bei vielen Entwicklerinnen und Entwicklern groSSer Beliebtheit erfreut. Sie wurde Anfang der 90er Jahre von Guido van Rossum, einem niederländischen Programmierer, entworfen und wird seitdem von einer groSSem Team Freiwilliger als freies Open-Source-Projekt gepflegt.

Python ist eine so genannte höhere Programmiersprache, die die Zeit der Programmiererin oder des Programmierers über die Zeit des Computers stellt. Gelegentlich ist Python-Code also etwas langsamer als zum Beispiel mühsam handoptimierter C-Code – dafür lässt es sich in Python viel schneller und angenehmer entwickeln.

Zu den Anwendungsbereichen von Python gehören unter anderen die Web-, System- und Spiele-Entwicklung. AuSSerdem wird Python für wissenschaftliche Zwecke eingesetzt – das ist der Aspekt, den wir beleuchten werden.

2 Installation von Python

Damit ein Computer Python-Programme ausführen kann, muss man zunächst einen *Python-Interpreter* installieren. Das ist die erste Aufgabe an euch!

Unter Ubuntu Linux und anderen Debian-Derivaten. Öffne eine Konsole und setz den Befehl `sudo apt-get install python-numpy python-scipy python-matplotlib` ab. Das war's schon.

Unter Mac OS X und Windows. Lade auf `http://continuum.io/downloads` das Komplettpaket Anaconda herunter und klicke dich durch die Installation. Wähle die Python-Version 2.7 und unter Windows die 32-Bit-Version (auch, wenn dein Computer ein 64-Bit-Prozessor haben sollte).

3 Erste Schritte

3.1 Hallo, Welt!

Das erste Programm, dass man schreibt, wenn man eine neue Programmiersprache lernt, ist *Hallo Welt!*: Ein Programm, dass eine kurze Meldung ausgibt und sich dann beendet. In Python sieht das so aus:

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 print("Hallo Welt!")
```

Tippe das Programm ab (**XXX:** oder kopiere den Quellcode von), speichere es unter einem Namen wie `hallo-welt.py` und führe es aus (**XXX:** wie unter Windows?). Wenn es nicht klappt, dann melde dich bei uns (**XXX:** Forum?).

Die Farben dienen nur der Übersichtlichkeit; es ist üblich, verschiedene Arten von Codepassagen in jeweils anderen Farben zu setzen. Die Färbung muss nicht und kann nicht abgetippt werden; stattdessen wird jeder Quelltext-Editor selbstständig den Code einfärben.

Die ersten beiden Zeilen finden sich in jedem Python-Programm. Die erste ist vor allem unter Linux und OS X wichtig; sie ist der Indikator für das Betriebssystem, dass es sich um ein Python-Programm handelt. Es ist guter Stil, sie auch unter Windows beizubehalten. Zeile 2 hat technische Gründe.[1]

Zeile 3 ist eine Leerzeile. Leerzeilen haben für Python selbst keine Bedeutung, können also nach Belieben hinzugefügt oder entfernt werden. Es ist aber guter Stil, einzelne Sinneinheiten durch Leerzeilen zu trennen, um den Code für den Menschen übersichtlicher zu gestalten. Es gibt ja auch einen Grund, wieso es im Deutschen und vielen anderen natürlichen Sprachen Absätze gibt.

Die eigentliche Arbeit wird durch Zeile 4 angestoSSen. Dort wird die in Python vordefinierte Prozedur `print` mit dem *Argument* `"Hallo, Welt!"` aufgerufen. Die

[1] Zeile 2 setzt fest, dass der Programmcode in der Zeichenkodierung UTF-8 (und nicht etwa in dem älteren Standard ISO-8859-1) interpretiert werden soll. Eine Zeichenkodierung gibt an, wie Umlaute und andere Zeichen, die über den Sprachschatz des amerikanischen ASCII-Standards aus den 60er Jahren hinausgehen, als Bytes gespeichert werden sollen.

Schreibweise soll an die in der Mathematik übliche Notation für Funktionen erinnern – dort schreibt man zum Beispiel „$\sin(5)$“. Im Programmierumfeld meint *print* nur *ausgeben*, nicht *drucken*. Der Begriff stammt aus einer Zeit, als es Bildschirme noch nicht gab und Computerausgaben tatsächlich ausgedruckt werden mussten.

3.2 Die interaktive Python-Shell

Python-Programme speichert man, wie im vorherigen Abschnitt beschrieben, in Dateien. Zum schnellen Experimentieren eignet sich aber auch die *interaktive Python-Shell*. In ihr kann man einzelne Python-Kommandos absetzen und erhält sofort Rückmeldung.

XXX: Beschreibung, wie man die Shell aufruft

Auf diese Weise kann man Python zum Beispiel als Taschenrechner verwenden:

```
$ python
Python 2.7.3 (default, Dec 18 2014, 19:03:52)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
55
>>>
```

3.3 Programmierfehler und wie man mit ihnen umgeht

Syntaxfehler Beim Programmieren macht man Fehler. Von denen gibt es vor allem zwei Arten: *Syntaxfehler* und *inhaltliche Fehler*. Ein Syntaxfehler tritt auf, wenn man sich nicht an die Schreibregeln von Python hält. Zum Beispiel wird der Code

```
1 print("Hallo, Welt!"
```

nicht funktionieren, da die schlieSSende Klammer fehlt. Syntaxfehler werden vom Python-Interpreter direkt nach dem Start, noch bevor er mit der Ausführung des Codes beginnt, erkannt und gemeldet:

```
  File "test.py", line 2

                      ^
SyntaxError: invalid syntax
```

Achtung: Gelegentlich befindet sich ein Syntaxfehler an einer früheren Stelle als der Interpreter angibt – hier etwa befindet sich der Fehler in Zeile 1, Python berichtet jedoch einen Fehler in der (eigentlich gar nicht vorhandenen) Zeile 2. Das liegt daran, dass die Auswirkungen eines Fehlers manchmal erst später eine nicht auflösbare Inkonsistenz verursachen.

Programmiersprachen wie Python sind in ihrer Syntax sehr streng. Schon scheinbare Kleinigkeiten wie Klammerfehler oder Vertauschung ähnlich aussehender Sonderzeichen (zum Beispiel " und ’) führen dazu, dass der Interpreter den Code nicht mehr

versteht. Anfangs macht man viele solche Syntaxfehler, was frustrierend sein kann. Mit der Zeit wird das aber schnell besser!

Eine Anmerkung zu Umlauten: Wenn der Interpreter sich über diese beschwert, liegt das meistens an technischen Kodierungsproblemen. Am einfachsten ist es, in solchen Fällen dem Problem aus dem Weg zu gehen und Umlaute zu umschreiben.

Inhaltliche Fehler Neben Syntaxfehlern kann man inhaltliche Fehler begehen. Diese gehören leider zur schlimmeren Sorte, da der Interpreter sich über diese nicht beschwert – die Schwierigkeit liegt bei diesen Fehlern darin, dass der Code zwar genau das tut, was er sagt; dass das aber nicht das ist, was man meinte. Ein einfaches Beispiel könnte folgender Code sein, der die Inhalte der Variablen `a` und `b` vertauschen soll:

```
a = b
b = a
```

Aber was macht dieser Ausschnitt wirklich? Zu Beginn haben die Variablen `a` und `b` irgendwelche Werte, zum Beispiel 3 und 7. Nach Ausführung der ersten Zeile haben beide Variablen denselben Wert, nämlich 7. Das ändert sich auch nach Ausführung der letzten Zeile nicht mehr.

Eine neue Idee muss her! Der alte Wert der Variablen `a` muss in eine Hilfsvariable gesichert werden, bevor `a` mit dem Inhalt von `b` überschrieben wird:

```
x = a
a = b
b = x
```

Dieser Code funktioniert.[2]

Grundtechniken im Debugging Unter *Debugging* versteht man den Prozess, Syntaxfehler und inhaltliche Fehler im Programmcode zu beheben. Syntaxfehler werden vom Interpreter gemeldet; man behebt sie, indem man sich die betreffende Zeile genau anschaut und die Unstimmigkeit sucht. Manchmal möchte man einen Fehler partout nicht erkennen, dann hilft es, eine Pause zu machen oder den Code jemand anderem zu zeigen.

Eine grundlegende Strategie, um inhaltliche Fehler zu beheben, besteht darin, den Inhalt von Variablen durch `print`-Anweisungen zu verfolgen, zum Beispiel so:

```
print("VORHER", a, b)
a = b
print("DANACH", a, b)
b = a
print("AM ENDE", a, b)
```

So kann man Widersprüche zwischen dem erwarteten und tatsächlichen Verhalten aufdecken.

[2] Wenn man sich in Python besser auskennt, weiSS man, dass es sogar noch eine idiomatischere Lösung gibt: Zur Vertauschung kann man einfach die Anweisung `a, b = b, a` verwenden.

4 Noch auszuarbeiten

- While-Schleifen anhand von „Summe der ersten n Zahlen“ erklären. Vollständigen Programmcode liefern, dann in Aufgaben Variationen fordern. Auch $\sum_n 1/n$ berechnen lassen, Problem erklären.
- Plotten. Dabei deutlich machen, dass man nicht Funktionen, sondern Vektoren von x- und y-Werten gegeneinander plottet. Sowohl anonyme als auch benannte Funktionen einführen. Als Beispiel vollständigen Programmcode liefern, der eine Sinuskurve plottet. Auch auf dreidimensionale Plots eingehen.
- Ableitungen numerisch berechnen.
- Online-Python-Lösungen prüfen.
- Irgendwo auch noch if-Abfragen unterbringen.
- Deadline für diese Punkte: Ende Februar
- Für später, auch in Abhängigkeit der Mathe-Blätter: Newton-Verfahren.