programmierBAR

from __future__ import python

Plädoyer für eine zu Unrecht (noch) kaum bekannte Sprache

Programmiersprachen - inklusive zugehöriger „toolchains“ - gibt es viele. Welche man bevorzugt, hängt oft vom persönlichen Geschmack ab. Mit diesem (zugegebenermaßen subjektiven) Text möchte ich einen Beitrag zur Herausbildung dieses Geschmacks leisten.

Was erwartet man als Ingenieur im 21. Jahrhundert von einer Programmiersprache?
Sie soll ein Medium sein, um Gedanken in Wirklichkeit zu verwandeln. Sie soll einen Rahmen bilden, der Halt bietet, aber kein Korsett darstellt, das einengt. Sie soll Arbeit abnehmen, anstatt zur ständigen Neuerfindung des Rades zu zwingen. Natürlich soll sie in Programme münden, die zügig laufen, aber noch wichtiger ist, dass man diese Programme zügig schreiben und später zügig wieder verstehen kann.

Zunächst stellt sich vielleicht die Frage, warum man als Ingenieur überhaupt „Programmieren“ können sollte - man will ja schließlich kein Softwareentwickler werden. Dem entgegen steht, dass die Grenzen zwischen einerseits „Rechnen“, „Simulieren“, „Messwerte aufnehmen“, „Daten auswerten“ etc. und andererseits „Programmieren“ fließend sind. Je besser man programmieren kann, umso mehr Möglichkeiten hat man in der eigentlichen inhaltlichen Arbeit (natürlich vorausgesetzt, man weiß einigermaßen, was man da tut).

Im Grundstudium bekommt man erstmal Java, C(++) und Assembler verordnet. Im Hauptstudium, spätestens bei Studien- und Diplomarbeit, sind dann oft Kenntnisse in MATLAB oder LabVIEW gefragt. Keine dieser Sprachen erfüllt die oben erwähnten Anforderungen. Dabei wäre es doch super, wenn es eine gäbe, mit der man sowohl wie in C klassisch programmieren, als auch wie in MATLAB oder Maple rechnen kann. Die es wie Java ermöglicht, basierend auf einem objektorientierten Framework eine Webanwendung zu stricken und die gleichzeitig wie LabVIEW in der Lage ist, einfach mit externer Hardware zu kommunizieren.

Eine solche Sprache gibt es und sie heißt python. Wobei die Sprache selbst nur einen Teil des überzeugenden Konzeptes darstellt, noch wichtiger ist die enorme Vielfalt und die Qualität der frei verfügbaren Zusatzpakete. Komischerweise ist python bisher nur in bestimmten Kreisen bekannt, typische Studierende unserer Fakultät gehören eher nicht dazu. Bei den Physikern sieht das schon anders aus.

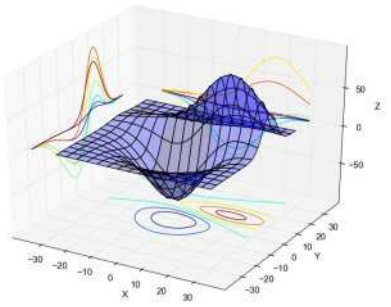
Die Sprache wurde mit dem Ziel entwickelt, verständlichen Programmcode hervorzubringen. Deutlichstes Merkmal dafür ist, dass Einrückungen von Blöcken (Schleifen oder Bedingungen) syntaktische Bedeutung haben. Programme, die funktionieren sind daher schon mal zwangsläufig richtig eingerückt, was der Lesbarkeit sehr zugute kommt. Außerdem erlauben die Syntaxregeln bestimmte Kurz-Schreibweisen nicht, z. B. Zuweisungen innerhalb von if-Abfragen, die in anderen Programmiersprachen oft zu „Missverständnissen“ zwischen Programmierer und Programm führen. Selbstverständlich unterstützt Python objektorientiertes Programmieren, erzwingt es aber nicht. Durch ein einfaches, aber durchdachtes Konzept von Modulen und Namensräumen kommt man im Prinzip auch ganz gut ohne aus bzw. braucht die Objektorientierung nur dort einsetzen, wo sie Sinn macht.

programmierBAR

Python verfügt über mächtige eingebaute Datentypen, z. B. Zeichenketten, komplexe Zahlen, Listen, Zuordnungen, oder Mengen (im mathematischen Sinn). Mächtig heißt hier, dass man mit den Daten ohne viel Aufwand schon eine ganze Menge anstellen kann. Zeichenketten beispielsweise kommen von Haus aus u.a. mit Methoden zum Suchen und zum Ersetzen von Substrings daher, Listen lassen sich in einer Zeile umkehren, sortieren oder in zufällige Reihenfolge bringen und für Mengen gibt es Schnitt, Vereinigung und Differenz ebenfalls schon vorgefertigt. „Batterien inklusive“ ist ein wichtiger Bestandteil der python-Philosophie.

Aus der Vielzahl sehr nützlicher Pakete sollen hier nur ein paar hervorgehoben werden. *numpy* und *scipy* stellen gemeinsam die Basis für sehr ausgefeilte numerische Berechnungen dar: Matrizen multiplizieren, Eigenwerte bestimmen, Funktionswerte interpolieren, Gleichungssysteme lösen, Differentialgleichungen simulieren oder Zielfunktionen minimieren - alles kein Problem. *sympy* ist so ähnlich, aber für symbolisches Rechnen: Integrieren, Differenzieren, Grenzwerte bilden oder schlicht (umfangreiche) Terme ausmultiplizieren - mit *sympy* wird der python-Interpreter zu einem Computer-Algebra-System. Dadurch hat man mehr Zeit, sich dem eigentlichen Problem zu widmen, anstatt die Fehler in den eigenen Rechnungen zu suchen. Schließlich noch *matplotlib*, das Standard-Paket für python um mit wenig Aufwand schöne und professionelle Grafiken zu zaubern. Bei Bedarf in 3D mit LaTeX-Achsbeschriftung, Transparenzeffekten und sehr vielen Anpassungsmöglichkeiten. Von dieser Unterstützung für Rechnen und Plotten sind Sprachen wie C und Java sehr weit entfernt.

Ein weiterer Pluspunkt ist das Lizenzmodell: Während man für so manche MATLAB-Lizenz außerhalb der Uni schon mal eine fünfstellige Summe in die Hand nehmen muss, bekommt man python und die darauf basierenden Pakete für lau, Quellcode inklusive. Vor diesem Hintergrund ist es schon überlegenswert, ob an der öffentlich finanzierten Uni die Ausbildung wirklich zum Teil obligatorisch auf die Beherrschung eines kommerziellen Produktes hinauslaufen muss - „Industriestandard“ hin oder her.



Eine mit matplotlib erstellte Grafik

Es gäbe noch viel mehr über python zu erzählen bzw. zu schwärmen, aber andererseits sollte die Kernbotschaft rübergekommen sein: python ist ultra-cool und, weil es kostenlos und quelloffen ist, erst recht! Natürlich ist auch in der python-Welt nicht alles in fünf Zeilen erledigt und bestimmte Dinge könnten gerne noch besser, schneller, einfacher oder überhaupt gehen aber das Problem hat man quasi überall.
Fazit: die eierlegende Wollmilchsau unter den Programmiersprachen gibt es noch nicht, aber python hat auf dem Weg dahin eine ganze Menge Vorsprung.

Carsten Knoll
http://wwwpub.zih.tu-dresden.de/~knoll/python.html