

Die Skriptsprache Python

E.R. Sexauer

April-2010

Zusammenfassung



Skriptsprachen erlauben schnelle und relativ einfache Erstellung von Programmen und Skripten. Die meisten von ihnen sind freie Software und plattform-übergreifend und erfreuen sich schon deshalb grosser Beliebtheit. Am bekanntesten sind wohl Perl und PHP, die im Systembereich und bei Web-Anwendungen verbreitet sind. Python hat die Zielsetzung sowohl für einfaches Skripting als auch für komplexe Anwendungen ein einfaches und sicheres Werkzeug zu bieten.

Ein besonderer Vorzug von Python ist die leichte Erlernbarkeit und gute Lesbarkeit.

Inhaltsverzeichnis

1	Namensgebung	4
2	Geschichte	4
3	Warum noch eine Skript-Sprache?	5
4	Spracheigenschaften	6
4.1	Allgemeines	6
4.2	Datentypen	6
4.3	Unterprogramme	7
4.4	Module	7
4.4.1	Import von Modulen	7
4.5	Klassen	8
5	Python benutzen	8
5.1	Editoren und IDE's	8
5.2	Indentation (Einrückung)	8
5.3	Programm-Dokumentation	9
5.4	Tutorials	9
6	Python und Datenbanken	10
7	Python und Web-Anwendungen	10
7.1	Moin-Wiki	10
7.2	Zope	10
8	Python und TCP/IP	10
9	Python und graphische Oberflächen	11
10	Python und Mathematik	11
11	Python und andere Programmiersprachen	11
12	Python, Klassen und Objekte	12
12.1	Operator-overloading	12
12.2	Abkapselung	12
12.3	Vererbung und Mehrfachvererbung:	12
12.4	Anwendung von OOP	13
13	Geschwindigkeit	14
13.1	Vergleich mit Perl und Ruby	14
13.2	Optimierung	14
14	Ladezeit und Speichern als Binärdaten	14
15	Python und Projektmanagement	15



16	Aus Kindern werden Leute	15
17	Praxiserfahrungen mit Python, Thomas Waldmann	16
17.1	Vorteile für Entwickler	16
17.2	Pluspunkte für Python:	16
17.2.1	Zusammenfassung	16
17.3	Vorteile für Admins	17
17.4	Vorteile für Anwender	17

1 Namensgebung



Der Name kommt von der englischen Comedy-Gruppe Monty-Python, warum diese den Namen der Riesenschlange mit einbauten, weiß man nicht so genau.

2 Geschichte



Python wurde 1991 veröffentlicht; die Vorarbeiten begannen Ende 1989. Die Version 2.0 wurde im Jahr 2000 freigegeben. Die Sprache wird von der 'Python Software Foundation', python.org, gepflegt. Ihr Vater und 'benevolent dictator' ist der Holländer Guido von Rossum, geb. 31.Jan.1956. Rossum war stark von der Programmiersprache 'ABC' beeinflusst, von der er einige Konzepte übernahm.

Lizenz: Python wird unter der Python Softwarelicence verteilt, einer freien Softwarelizenz. In Python geschriebene Programme unterliegen keinen Einschränkungen.

3 Warum noch eine Skript-Sprache?

Um sich gegen die Vielzahl anderer Programmiersprachen - Wikipedia zählt einige Hundert auf - zu behaupten, muß eine Sprache Eigenschaften bieten, die in dieser Zusammensetzung bei anderen Sprachen fehlen oder nur schwach entwickelt sind. Einige bekannte Konkurrenten sind:

1. Bash: Die klassische Unix-Shell. Bash gibt es für Unix und unter Cygwin für Windows.
2. Visual Basic, Windows Scripting-Host. Diese Sprache ist nur unter Windows verfügbar.
3. PHP. PHP ist frei verfügbar und erfreut sich im Bereich der Web-Programmierung großer Verbreitung.
4. Perl. Im Bereich Systemverwaltung ist Perl die am meisten verbreitete Skriptsprache. In der CPAN-Bibliothek finden sich für nahezu alle denkbaren Anwendungen frei verfügbare Module.
5. Ruby. Ruby ist eine japanische Entwicklung und ist in diesem Land weit verbreitet.

Man sieht, die Konkurrenz ist beachtlich. Welche Zielsetzung verfolgt also Python?

- Vom Leistungsumfang bietet Python (mindestens) alles, was PHP und Perl auch können.
- Python ist einfach zu lernen und gut zu lesen.
- Programmieren mit Python macht Spaß - schon die Namensgebung deutet dies an.

Ein schöner Artikel zu diesem Thema findet sich unter:

- <http://www.linuxjournal.com/article/3882>

Der Verfasser, Eric Raymond, ist in der Linuxwelt durch seine Schrift 'Die Kathedrale und der Bazar' bekannt geworden. Er hat nach seinen Angaben Erfahrungen mit über 20 Programmiersprachen, natürlich auch mit Perl. Drei (sinngemäß übersetzte) Zitate aus diesem Artikel:

- Schon am ersten Tag war ich überrascht, wie schnell ich in Python funktionsfähigen Code schreiben konnte.
- Ich hatte ein kompliziertes Problem mit Klassen zu lösen. Nach 6 Tagen Erfahrung mit Python und zwei Stunden Arbeit hatte ich den ersten Ansatz - und es funktionierte. Es war ein Schock.
- Nach Monaten konnte ich den Code immer noch lesen und leicht verstehen. Das war die größte Überraschung.

4 Spracheigenschaften

4.1 Allgemeines

- Python (genauer gesagt, CPython) ist ein Compiler/Interpreter: Der Quelltext wird in Code für eine virtuelle Maschine übersetzt. Dieser Code wird vom Laufzeitsystem interpretiert.
- Python ist - von außen gesehen - schwach typisiert: Jede Variable kann jeden Datentyp enthalten und den Typ auch zur Laufzeit wechseln; der aktuelle Typ kann per Programm abgefragt werden. Intern ist jedes Objekt streng typisiert. Etwaige Unverträglichkeiten werden zur Laufzeit erkannt und behandelt, daher spricht man auch von dynamischer Typisierung.
- Variable sind per Default lokal. Globale Variable sind möglich, müssen aber explizit deklariert werden.
- Python erlaubt prozedurale und objektorientierte Programmierung. Wer prozedural programmiert, braucht über Objekte nichts zu wissen.
- Python unterstützt Module mit eigenen Namensräumen.
- Die Speicherverwaltung ist für die Anwendung transparent. Nicht mehr verwendeter Speicher wird automatisch freigegeben (garbage collection).

4.2 Datentypen

- Integer beliebiger Genauigkeit, Gleitkomma (Float), Dezimalarithmetik (BCD) und komplexe Zahlen. Als Erweiterung steht die Gmpy zur Verfügung, die Integers, Brüche (rationals) und Floats mit beliebiger Genauigkeit ermöglicht.
- Zeichenketten (Strings) können beliebig lang sein. Ascii, Unicode und UTF-8 werden unterstützt. Zur Bearbeitung stehen u.a. reguläre Ausdrücke wie bei Perl zur Verfügung. Strings sind unveränderliche Objekte, Modifikationsoperationen erzeugen ein neues Stringobjekt.
- Listen (arrays) aus beliebigen Objekten mit beliebiger Schachteltiefe. Die Indizierung beginnt mit '0' oder mit '-1' bei Zugriff von rechts. Teile (slices) werden mit 'array[n:m]' bezeichnet. Tupel sind unveränderliche Listen; sie werden hauptsächlich als Schlüsselbegriff verwendet.
- Mengen (sets). Mengen ähneln Listen, enthalten jedoch jedes Element nur einmal und sind von der Reihenfolge der Elemente unabhängig, d.h. $\text{set}(1,2)=\text{set}(2,1)$.
- Hash-Tabellen (Dictionaries, assoziative Arrays). Die Schlüssel können beliebige unveränderliche Objekte sein, die zugehörigen Werte sind beliebige Objekte.

Hinweis für Perl-Anwender:

Python verwendet grundsätzlich Referenzen für Listen und Hashs; explizite Referenzen und Dereferenzen werden *nicht* benötigt.

Hinweis für PHP-Anwender:

PHP unterscheidet nicht zwischen Array und Hash. Ein Python-Hash ist ein PHP-Array mit Schlüsselwerten.

4.3 Unterprogramme

Python erlaubt positionelle und Schlüsselwort-Parameter, z.B. 'function(a, b, option=False)'. Defaultwerte werden in der Deklarationszeile direkt angegeben. Variable sind grundsätzlich lokal, können aber als global deklariert werden.

Listen und Hashes werden als Referenz übergeben.

Rückgabewerte können beliebige Objekte sein.

4.4 Module

Module werden in eine separate Datei geschrieben. Sie werden mit 'import' geladen und bilden einen eigenen Namensraum. Sie können als vorkompilierte Binärdatei ausgeliefert werden. Ähnlich wie bei Perl gibt es frei verfügbare Module für eine Vielzahl von Anwendungen.

Ein Liste häufig gebrauchter Module findet man unter:

- <http://wiki.python.de/Module>.

Die vollständige Liste von Python.org findet sich unter:

- <http://pypi.python.org/pypi?%3Aaction=browse>

4.4.1 Import von Modulen

Module werden mit 'import module' geladen. Danach können die Komponenten mit 'module.name' angesprochen werden, also z.B. 'math.sin(x)'. Mit 'from module import namelist' können einzelne Komponenten importiert und dann mit ihrem einfachen Namen angesprochen werden. Ist namelist=*, werden (fast) alle Komponenten importiert.

4.5 Klassen

Technisch gesehen ist fast alles (außer Zahlen) in Python ein Objekt, das von eingebauten Klassen erzeugt wird. Wer prozedural programmiert, braucht sich darum nicht zu kümmern. Er wird höchstens merken, dass er nicht `'sort(array)'` schreibt, sondern `'array.sort()'` und damit eine Methode der Klasse `'array'` anwendet.

Selbst definierte Klassen werden ähnlich wie Module behandelt, haben aber weitergehende Eigenschaften wie z.B. Vererbung.

5 Python benutzen

5.1 Editoren und IDE's

Python kann interaktiv über eine eigene Shell benutzt werden. Diesen Modus verwendet man hauptsächlich zum Testen. In der Regel wird man Programme in eine Datei mit der Endung `'.py'` schreiben und von dort ausführen. Die meisten Editoren unterstützen Syntaxhighlighting für Python - natürlich auch Vim und Emacs.

Als IDE (integrierte Entwicklungsumgebungen) empfehlen sich

- Geany,
- Eric,
- Eclipse,
- Komodo und
- PIDA.

Diese IDE's erlauben es, Projekte zu definieren und als Ganzes zu bearbeiten. Mit `'mercurial'` steht ein in Python geschriebenes System zur Revisionskontrolle zur Verfügung.

Bei der Windows-Version ist die simple IDE `'Idle'` im Installationsumfang enthalten.

5.2 Indentation (Einrückung)

Einrückung ist in Textgestaltung und Programmierung allgemein üblich, um untergeordnete Strukturen optisch sichtbar zu machen. In Python wird Einrückung auch syntaktisch verwendet; das spart Klammern und verbessert die Lesbarkeit.

Tücke:

Nicht alle Editoren setzen 'Tabs' gleichartig in Leerzeichen um. Man sollte daher durchgängig Tabs oder Leerzeichen verwenden, nicht aber beides gemischt. Das Python-Styleguide empfiehlt 4 Leerzeichen pro Einrückungsebene. Manche Editoren, z.B. Geany und Kate, erlauben es, Whitespaces anzuzeigen und etwaige Probleme optisch zu erkennen.

5.3 Programm-Dokumentation

Ein gutes Programm sollte Kommentare und Dokumentation enthalten. Zahllose an sich gute Projekte sind schon durch mangelnde Dokumentation zugrunde gegangen.

In Python ist es üblich, der Deklaration einer Funktion oder Klasse einen String folgen zu lassen, in dem der Verwendungszweck erklärt ist; Strings über mehrere Zeilen werden in dreifache Anführungszeichen eingeschlossen. Dies ist gegenüber dem sonst (hoffentlich) verwendeten Kommentar *keine* Mehrarbeit. Der String wird

- automatisch,
- *unverwechselbar* und
- *unverlierbar*

zum Bestandteil des Objekts und läßt sich über die Methode '`__doc__`' abfragen. Die Funktion '`help(Modulname)`' listet alle Definitionen eines Moduls einschließlich ihrer Dokumentationsstrings auf. Bei Klassen werden auch ererbte Methoden angezeigt.

Da diese Doc-Strings Bestandteil des Objekts sind, können sie auch per Programm abgefragt werden und z.B. in Onlinehilfe oder Projektkontrolle eingebaut werden - ein Luxus, den kaum eine andere Programmiersprache bietet (Perldoc bietet etwas ähnliches).

Das Programm 'pydoc' zeigt - auf Wunsch auch im Webbrowser - die im System verfügbare Dokumentation an.

5.4 Tutorials

Es gibt viele Tutorials für Python. Eine kleine Auswahl:

- <http://python.net/~gherman/publications/tut-de/online/tut/> (Übersetzung, G.v. Rossum)
- <http://tutorial.pocoo.org/> (Version-3)
- <http://showmedo.com/videotutorials/series?name=HRCwU3mb5> (Video)
- <http://wiki.python.org/moin/> (Wiki, englisch)
- <http://wiki.python.de/> (enthält auch einen Chat)

6 Python und Datenbanken

Python kommt mit SQLite, einer eingebauten, einfachen Datenbank, die mit Standard-SQL angesprochen wird und für Einzelplatzsysteme in manchen Anwendungen ausreichen mag. In der Regel wird man allerdings Schnittstellen zu 'richtigen' Datenbanken einsetzen; es gibt sie für alle marktüblichen Datenbanken. Das Toolkit SQLAlchemy realisiert - soweit möglich - ein einheitliches Interface zu unterschiedlichen Datenbanken.

Die Programmierung ist ähnlich wie bei PHP und Perl: Man definiert die SQL-Statements als String und schickt diese Strings an die Schnittstelle.

Eine typische Anwendung im Systembereich ist der Datenaustausch zwischen 2 unterschiedlichen Datenbanken. Man baut mit Python eine Verbindung zu beiden Datenbanken auf, und kann dann nach Belieben Daten austauschen.

In Fällen, in denen man eigentlich gar keine Datenbank braucht, oder nicht voraussetzen kann, dass auf dem Zielsystem eine Datenbank existiert, ist es möglich, Datenstrukturen von Python auf die Platte auszulagern.

Als NoSQL-Datenbank wird die Berkely-DB unterstützt.

7 Python und Web-Anwendungen

Python kann wie PHP oder Perl zur Entwicklung von Webseiten verwendet werden. Ausführliche Referenzen finden sich unter:

- <http://wiki.python.org/moin/WebFrameworks>.

Zwei bekannte Anwendungen sind:

7.1 Moin-Wiki

Dieses Wiki wird von vielen Projekten und Communities eingesetzt, unter anderen von Linuxwiki, Apachewiki und Ubuntuwiki. Es ist modular aufgebaut, enthält eine Schnittstelle für Plugins und läßt sich leicht anpassen/erweitern.

7.2 Zope

Zope ist ein objektorientierter Server für Webanwendungen. Er wird z.B. als Grundlage des CMS Plone oder für Wikis und Blogs genutzt.

8 Python und TCP/IP

Das Paket 'Twisted' ist ein Baukasten, aus dem beliebige Netzwerkdienste mit geringem Programmieraufwand zusammengebaut werden können.

9 Python und graphische Oberflächen

Python enthält Module, mit denen GUI's leicht erzeugt werden können. Bekannt sind Schnittstellen zu TK, GTK und QT.

Für Liebhaber von Spielen gibt es 'Pygames', ein Baukasten zum Design von Spielen. Hier wird die Struktur des Spieles in Python implementiert, während aufwändige Graphiken in C-Modulen geschrieben werden können.

Beispiele: solarwolf, monsterz.

10 Python und Mathematik

Für Python gibt es eine Vielzahl von Modulen, die mathematische Probleme lösen. Bekannt sind 'Gmpy' für Rechnungen mit beliebiger Genauigkeit und 'NumPy' zur Bearbeitung numerischer Probleme.

Das Projekt 'Sage', sagemath.org, bringt mathematische Programme unterschiedlicher Herkunft unter den Hut von Python und ermöglicht so eine einheitliche Bedienung. Es ist eine freie Alternative zu teuren Mathematikprogrammen.

11 Python und andere Programmiersprachen

Mit PyInline lassen sich Module, die in anderen Programmiersprachen geschrieben sind, in Python-Programme einbinden. Unterstützt werden u.a. C, Java und Octave. (Octave ist ein frei verfügbares Algebra-Programm.)

JPython (Jython) übersetzt Python in Java-Bytecode.

12 Python, Klassen und Objekte

In OOP (Objekt-orientierter Programmierung) werden grob gesprochen Daten und zugehörige Routinen zu deren Bearbeitung zu einem Ganzen, zu einem Objekt, zusammengefaßt. Dadurch wird sicher gestellt, dass Daten nur von den für sie bestimmten Funktionen bearbeitet werden können. Ein Objekt wird durch ein Schema, seine Klasse, definiert. Fundamentale Konzepte von Oop sind:

- Abkapslung: Interne Datenstrukturen und Funktionen können (und sollen) nach außen unsichtbar gemacht werden.
- Vererbung: Aus einer (oder auch aus mehreren) Klasse(n) können neue Klassen hergeleitet werden, die zusätzliche oder modifizierte Methoden enthalten.
- Polymorphie: Gleiche Methoden können je nach Kontext unterschiedliches Verhalten zeigen. Z.B. erzeugt der Operator '+' für Zahlen, Strings und Arrays unterschiedlich Ergebnisse.

12.1 Operator-overloading

Python erlaubt es, die Standard-Operatoren '+, -, *, /, =, <, >' für eine Klasse zu redefinieren. Dadurch kann die Programmierung vereinfacht und die Lesbarkeit verbessert werden.

12.2 Abkapselung

Die Sichtbarkeit nach außen wird in Python typographisch realisiert:

- Methoden, die mit einem Buchstaben beginnen, sind sichtbar und werden mit 'import' geladen - *public*.
- Methoden die mit einem einfachen '_' beginnen, werden mit 'import' nicht geladen, können aber mit ihrem vollen (qualifizierten) Namen angesprochen werden, z.B. 'Myclass._a()' - *protected*.
- Methoden, die mit einem doppelten '__' beginnen, sind nach außen unsichtbar - *private*.

12.3 Vererbung und Mehrfachvererbung:

'class Z(X):' erzeugt eine Klasse Z, die von X alle Methoden erbt. In Z definierte Methoden überschreiben oder ergänzen die Methoden von X. Bei der Definition von Methoden können die Methoden der übergeordneten Klasse explizit aufgerufen werden. Der Name von X kann mit der Methode '__module__' abgefragt werden.

Python unterstützt auch Mehrfachvererbung, z.B. erzeugt 'class Z(X, Y):' eine Klasse Z, welche die Methoden von X und Y erbt. Namenskonflikte werden von

links nach rechts aufgelöst, d.h. wenn X und Y eine Methode 'M' haben, gilt die Methode X.M.

Manche OOP-Puristen lehnen Mehrfachvererbung ab.

12.4 Anwendung von OOP

Für einfache Skripte braucht man im allgemeinen keine Objekte. Wirklich nützlich werden Objekte bei größeren Projekten - allerdings nur, wenn man die Klassenstruktur *vorher* wirklich gründlich durchdacht hat. Unkritische 'Geschwind-Programmierung' erzeugt mehr Probleme als Lösungen - auch wenn man ihr das Etikett OOP anhängt.

13 Geschwindigkeit

13.1 Vergleich mit Perl und Ruby

Für ein Programm, das hauptsächlich Strings verarbeitet und in einem Hash abspeichert, ergaben sich folgende Vergleichswerte:

- Python ist etwa 10% langsamer als Perl
- Python ist doppelt so schnell wie Ruby

13.2 Optimierung

Die übliche (und gute) Methode besteht darin, *zuerst* die Logik eines Programms zu testen und sich *erst anschließend* mit der Laufzeit zu befassen. Wenn man diese Methode nicht befolgt, kann man viel Zeit mit Optimierung von irrelevanten Programmteilen verschwenden. Dieses Vorgehen ist von der Programmiersprache unabhängig.

Für Python steht ein Modul 'psyco' zur Verfügung, das

- Profiling gestattet und
- Teile eines Programms automatisch optimieren kann. Je nach Programm können Verbesserungen bis zum Faktor drei erreicht werden.

Erst wenn das nichts hilft, sollte man kritische Passagen in C schreiben.

Eine hübsche Geschichte mit einer Moral zum Thema Optimierung findet sich unter:

- <http://www.python.org/doc/essays/list2str.html> (eilige Leser brauchen bloß die Zusammenfassung (conclusion) zu lesen).

14 Ladezeit und Speichern als Binärdaten

Module und Klassen werden beim Laden einmalig übersetzt und in Dateien mit Endung '.pyc' abgelegt. Bei erneutem Aufruf wird geprüft, ob der Quelltext neuer ist. Wenn nicht, wird die vorhandene Pyc-Datei benützt. (Das entspricht in etwa einem PHP-Accelerator.)

Wenn keine Quelltextdatei vorhanden ist, wird die Pyc-Datei ungeprüft geladen.

Durch Strukturierung eines Programmes in Module und Klassen erreicht man:

- Kurze Ladezeiten
- Auslieferungsmöglichkeit eines Programms im Binärformat.

Das Modul 'Pyinstall' erlaubt es, aus einem Programm eine Binärdatei zu erzeugen, die auf dem Zielsystem kein Python benötigt. Natürlich sind solche Dateien entsprechend groß, da sie ja das Laufzeitsystem und alle benötigten Module enthalten müssen.

15 Python und Projektmanagement

Programmierung ist eine Sache, ein erfolgreiches Projekt ist eine andere Sache.

- Ein einzelnes Programm kann durchaus von *einem* genialen Programmierer leben - solange wie dieser greifbar ist...
- Ein Projekt kann nur (über)leben, wenn seine Komponenten auch für durchschnittliche Programmierer verständlich und wartbar sind.

Diese Einsicht ist nicht neu, wird aber in der Praxis selten beherzigt.

Python kann von sich aus keinen Projekterfolg garantieren - kein Werkzeug der Welt kann das - bietet aber Hilfsmittel, deren Anwendung wenig zusätzliche Arbeit verlangt und zum Erfolg beitragen kann.

Anwenden muß man sie allerdings selbst...



16

Aus Kindern werden Leute

Wie alles im Leben, hat auch Python seinen Wachstumszyklus. Was bei Perl und PHP der Sprung auf Version 6 *ist, war* bei Python der Übergang zur Version-3. Der Preis für die Neustrukturierung ist der Verzicht auf Rückwärtskompatibilität.

Python-3.x gibt es seit 2 Jahren; die Version 2.6 bietet einen Übergang. Die neuen Features von 3.x sind schon da und die alten Funktionen von 2.x sind noch da, können aber als Warn[un]g identifiziert werden. Für die Umformung des Quellcodes gibt es automatisierte Hilfsmittel. Ohne manuelle Arbeit geht es allerdings nicht.

17 Praxiserfahrungen mit Python, Thomas Waldmann

17.1 Vorteile für Entwickler

- Ich programmiere seit ca. 8 Jahren in Python (primär MoinMoin, aber auch div. kleinere Projekte und Skripts) und habe Spass dabei.
- Ich habe früher (Schule/Informatik-Studium) in vielen anderen Sprachen programmiert, dann längere Zeit nicht mehr, weil es einfach zu zeitaufwändig war und selbst einfache Dinge kompliziert waren (schlechtes Kosten/Nutzen-Verhältnis). Manche andere "Highlevel-Sprachen" sind zu akademisch oder spezialisiert und daher nur in speziellen Bereichen einsetzbar, aber nicht universell.

17.2 Pluspunkte für Python:

- ist einfach
- ist kurz
- ist mächtig
- ist elegant
- ist gut lesbar
- ist wirklich portabel
- ist wiederverwendbar
- ist fast universell gut verwendbar
- hat "batteries included", d.h. oft benötigten Code findet man oft fertig in der Standardbibliothek
- ... oder in frei verfügbaren Zusatzbibliotheken / Frameworks
- hat eine gute Community mit relativ vielen guten Entwicklern
- durch Interpreter kurze Turnaround-Zeiten
- einfaches Debugging
- einfach zu verwendende Test-Frameworks (z.B. py.test)

17.2.1 Zusammenfassung

Diese Spracheigenschaften führen auch dazu, dass Entwickler dann auch bei eigenem Code versuchen, ähnliche Eigenschaften zu implementieren. Eine Sprache, die kompliziert, unleserlich, ... ist führt hingegen auch oft zu kompliziertem, unleserlichem, ... Code.

17.3 Vorteile für Admins

Shellskripts werden gerne zur Automatisierung verwendet - wenn es aber nicht gerade mehr triviale/kurze Skripts sind, ist es oft einfacher, anstatt dessen Python zu verwenden:

- einfachere Syntax, weniger rumprobieren
- besser erweiterbar / weiterentwickelbar
- einfachere Lösung durch mächtigere Sprache (Shellskripts werden zu komplex, wenn man versucht mit simplen Tools und simplen Datenstrukturen komplexe Probleme zu lösen)
- nur Python, keine wilde Mischung aus Shell, grep, sed, awk, ...
- man kann aber auch Shell-Kommandos aufrufen
- portabel
- Python ist mittlerweile weit verbreitet:
- wird auf Linux/BSD/... viel benutzt und vorinstalliert / leicht installierbar
- standardmäßig auch auf Mac OS X vorhanden
- unter Windows leicht installierbar

17.4 Vorteile für Anwender

Ich berücksichtige inzwischen auch bei der Anwendungsauswahl mit die Implementierungssprache - wenn es ähnliche Software in C, Java, Perl, PHP und Python gibt, bevorzuge ich die Python-Implementierung.

Einfacherer, besser lesbarer Code, geringere Größe der Code-Basis, Spass am Programmieren ist gut für ein Projekt: weniger Fehler, besser wartbar, mehr Contributors, bessere Weiterentwicklung.

Zum Beispiel verwendete MoinMoin anfangs CVS als Versionskontrollsystem. CVS = alt, kompliziert, wenig mächtig, zentralistisch, in C implementiert, keine Weiterentwicklung.

Dann auf GNU Arch (tla) umgestellt, wegen besserem Support für Branching / Merging, Distributed Version Control, etc. - für den damaligen Zeitpunkt sehr gut und deutlich besser als CVS, aber auch in C implementiert. Relativ langsam in der Praxis (vermutlich durch suboptimale Algorithmen, Datenstrukturen und Protokolle), langsame Weiterentwicklung, Probleme auf Windows, relativ große Codebasis.

Später auf Mercurial (hg) umgestellt - ein in Python implementiertes DVCS. hg = klein, schnell (viel schneller als tla!), einfach, mächtig, distributed, schnelle Weiterentwicklung, wenig Bugs.

Mercurial und Git sind übrigens fast gleichzeitig entstanden und haben ähnliche Grundeigenschaften, teilweise auch ähnliche Kommandos, aber: Git = C, Bourne Shell, Perl Mercurial = Python, etwas C an performancekritischen Stellen (aber auch "pure Python" ist möglich, weil es auch entsprechende .py-Module gibt)

Anhang, Programm-Beispiele

Doppelter Loop in Python:

```
#!/usr/bin/python

a=[1,2,3]
b=[3,4,5,6]

def amul(a,b):
    "multiply each element of a and b, return sum of products"
    sum=0
    for ax in a:
        for bx in b:
            sum += ax*bx
    return sum

print "sum=", amul(a,b)
```

Doppelter Loop in Perl

```
#!/usr/bin/perl -w

my @a=(1,2,3);          # my declares a variable as local
my @b=(3,4,5,6);

sub amul {

    (my $a, my $b)= @_;          # fetch arguments, $a, $b are reference
    my $sum=0;
    foreach my $ax (@$a) {      # @$a is dereference
        foreach my $bx (@$b) {
            $sum += $ax*$bx;
        }
    }
    return $sum
}

print "sum=", amul(\@a, \@b), "\n";
```

Doppelter Loop in PHP

```
#!/usr/bin/php
<?php

$a=array(1,2,3);
$b=array(3,4,5,6);

function amul($a, $b) {
    $sum=0;
    foreach ($a as $ax) {
        foreach ($b as $bx) {
```

```

        $sum += $ax*$bx;
    }

    }
    return $sum;
}

echo "sum=", amul($a,$b);

?>
Klasse Demo in Python, implementiert Datentyp Menge
#!/usr/bin/demo

# simple class definition
# implements sets

from types import *

class DemoError:
    "error handling for class demo"
    pass

class Demo():
    """ simple implementation of sets
        overloaded operators: +, -, ==, <, >
    """

    _icount=0 # number of instances , class-variable , hidden

    def __init__(self, a):
        """constructor, hide argument a in internal variable __a, encapsulate
        remove duplicate elements and sort for convenience and comparison
        call with a=list or tuple
        """
        if not type(a) in (ListType, TupleType):
            # check argument type
            print "arg must be List or Tuple", type(a)
            raise DemoError

        result=[]
        for x in a:
            if not x in result:
                result.append(x)
        # sort elements for human readability and comparison
        result.sort()
        # store result in hidden variable
        self.__a=result

```

```

        # count no of instances in class variable
        self.__no=Demo._icount
        Demo._icount += 1

def __delete__(self):
    "destructor , close files , close connections eg."
    pass

def getlist(self):
    "return data of object as list"
    return self.__a

def isitem(self ,a):
    "returns true if a is element of self"
    return a in self.__a

def additem(self ,a):
    "add item to set"
    if not self.isitem(a):
        self.__a.append(a)
        self.__a.sort()

def delitem(self ,a):
    "delete item from set"
    if self.isitem(a):
        x=self.__a.index(a)
        self.__a.pop(x)

def __len__(self):
    "returns number of elements , call as function len(a)"
    return len(self.__a)

def len(self):
    "returns number of elements , call as method a.len()"
    return len(self.__a)

def show(self):
    "simple print method"
    print "id=", id(self.__a), "no=", self.__no, "value:",
    print self.__a

def __iter__(self):
    "iterator , allow construct: for x in self"
    return iter(self.getlist())

def __eq__(self ,other):
    "overload == operator , 2 objects are equal if they contain the
    return self.getlist()==other.getlist() # both lists are sorted

```

```

def __lt__(self, other):
    "true if self is subset of other, overload < operator"
    s=self.getlist()
    o=other.getlist()
    result=True
    for es in s:
        if not es in o:
            result=False
            break
    return result

def __gt__(self, other):
    "return true if other is subset of self, overload > operator"
    return other<self

def __add__(self, other):
    """
        overload + operator, a+b -> union(a,b)
        causes runtime-error if other not instance of class Demo
    """
    result=self.getlist() + other.getlist() # this is + for arrays
    return Demo(result)

def __sub__(self, other):
    """overload - operator, a-b -> array of x in a but not in b
        raise error if other not instance of class demo"""

    tother=repr(other) # type of other
    try:
        tother.index("Demo.Demo") # find string in type
    except:
        print tother, "not instance of class Demo"
        raise DemoError

    result=[]
    for a in self.__a:
        if (not a in result) and (not a in other.__a):
            result.append(a)
    return Demo(result)

class xDemo(Demo):

    "inherited class of Demo, has all properties of Demo plus multiply-meth

def __mul__(self, other):
    "overload * operator"
    result=0
    for xs in self.getlist():
        for xo in other.getlist():
            result += xs*xo

```

```
return result
```

Beispiel zur Anwendung von Demo

```
#!/usr/bin/python
```

```
from Demo import *
```

```
# declaration
```

```
a=Demo([1,2,2,3,3,4])
```

```
b=Demo([16,17,3,4,5,5,16,16,17])
```

```
aa=Demo([4,3,2,1]) # same as a
```

```
# comparisation
```

```
print "aa ==? a", a==aa
```

```
# is x element of a?
```

```
x=22
```

```
print x, "?in?", a.getlist(), a.isitem(x)
```

```
print "add/delete item"
```

```
b.additem(8)
```

```
b.show()
```

```
b.delitem(8)
```

```
b.show()
```

```
# iterate over a
```

```
print "elements of a",
```

```
for x in a:
```

```
    print x,
```

```
print
```

```
print "display set and length of set"
```

```
a.show()
```

```
print "len=", a.len(), len(a)
```

```
b.show()
```

```
print "addition, subtraction"
```

```
s=a+b
```

```
d=a-b
```

```
s.show()
```

```
d.show()
```

```
print "inherited class"
```

```
x=xDemo([3,2,1,3,2,1])
```

```
y=xDemo([5,5,3,3,4,4])
```

```
z=x+y
```

```
z.show()
```

```
print "mul:", x*y
```