

PYTHON BEISST NICHT

oder

Was ist Python?

Andreas Gärtner - a.j.g@gmx.net
Roland Berg - bergroland@gmx.de

13. Juni 2005

Abstract:

Eine Übersicht über die Geschichte und Anwendungsgebiete von Python, unter Berücksichtigung der Lizenzierung, mit einem kurzen Einblick in die (objektorientierte) Syntax im Vergleich mit anderen Skript- und OO-Sprachen.

▷ Version 0.96	24.05.2005	Korrekturen
▷ Version 0.95	19.05.2005	RFC

Inhaltsverzeichnis

I	Python	4
1	Einleitung	4
1.1	Was ist Python? (Teil I)	4
1.2	Der Autor	4
1.3	Die Geschichte	5
1.3.1	Gründe für eine neue Sprache	5
1.3.2	Die Evolution	6
1.4	Die Lizenz	7
2	Die Sprache Python	8
2.1	Allgemeines	8
2.2	Was ist Python? (Teil II)	8
2.2.1	Download	9
2.2.2	Installation	9
2.3	Einführung in Python	12
2.3.1	Tutorials	12
2.3.2	Anwendungsgebiete	12
2.3.3	Die Sprachsyntax	14
2.3.4	Objektorientierung	16
2.3.5	Besonderheiten in Python	19
2.4	Vergleich mit anderen Sprachen	21
2.4.1	Skript- vs. Compilersprachen	21
2.4.2	Python vs. PHP	22
2.4.3	Python vs. C++ / Java	23
2.4.4	Python vs. Perl	24
3	Anwendungsbeispiele	25
3.1	Web-Applikationen	26
3.1.1	Zope	26
3.1.2	Plone	27
3.1.3	Twisted	27
3.1.4	Lizenzen	27
3.2	Anwendungssoftware	28
3.2.1	Wing IDE	28
3.2.2	Eclipse	29
3.2.3	Bittorrent	30
3.2.4	Weitere Anwendungsgebiete	31
II	Anhang	32

Ein Vorwort

Diese Arbeit entstand als eine Studienarbeit zum Fach 'Open Source Software' (OSS) im Sommersemester 2005 an der Fachhochschule Augsburg. Um dem Thema gerecht zu werden, wurde die vorliegende Ausarbeitung zudem in \LaTeX erstellt. Erleichtert haben wir uns die Arbeit gelegentlich mit lyX, einem Open-Source¹ Pseudo-WYSIWYG-Editor für \LaTeX -Dokumente.

Da sich die Veranstaltung OSS intensiv mit der Geschichte der Open Source Bewegung und wichtiger Softwareprodukte dieses Lizenzierungsmodells auseinandersetzt, haben wir uns als Thema für eine genauere Betrachtung Python herausgegriffen.

Um es gleich vorweg zu nehmen, diese Arbeit versteht sich als grundlegende Einführung in das, was Python darstellt. Es soll kein Tutorial in oder über Python sein, da es hier bereits gute und umfassende Werke gibt². Unser Ziel war es, diese Arbeit für jedermann und -frau verständlich zu halten. Voraussetzung sind nach unserem Ermessen höchstens grundlegende Kenntnisse in Windows oder Linux, um den Abschnitt 2.2.2-Installation nachvollziehen zu können. Selbst wenn diese nicht vorhanden sein sollten, hoffen wir dennoch mit diesem Essay einen interessanten und verständlichen Überblick über Python geben zu können. An Stellen, an denen eine tiefergehende Betrachtung den Sinn dieser Übersicht durch den Umfang ad absurdum geführt hätte, haben wir uns daher auf Verweise und Links auf weiterführende Themen beschränkt.

¹lizenziert unter einer leicht modifizierten GNU Public License

²Verwiesen sei hier unter anderem auf [Pilgrim] und auch das Skript zur Vorlesung OOSW - Objektorientierte Software- und Systementwicklung[Hoegl1].

Teil I

Python

1 Einleitung

1.1 Was ist Python? (Teil I)

Wenn man Python im Lexikon nachschlägt, findet man unter anderem folgendes:

- Einen Drachen in der griechischen Mythologie,
- eine Schlangenart,

und mit der, um das Medium Internet erweiterten Suche

- eine Gruppe von englischen Komikern namens „Monty Python“ um Graham Chapman, John Cleese und einige andere.

Einen weiteren, nicht ganz unwichtigen Fund haben wir allerdings noch unterschlagen:

- Die Programmiersprache Python.

Wenn man jetzt noch weiss, dass der Autor dieser Programmiersprache ein begeisterter Fan von Monty Python und der BBC-Fernsehserie 'Monty Python's Flying Circus' ist, erklärt sich der Name dieser Sprache. Bekannt ist auch, dass aufgrund der Affinität zu Monty Python in Arbeiten zu diesem Thema häufiger Anspielungen auf die Serie zu finden sind. Dieser Tatsache wollen wir uns auch nicht ganz verschließen.

Dass sich Van Rossum mit der Benennung nicht auf die oben genannte Schlange bezieht, lässt sich dann auch aus den Worten des Autors von 'A Byte of Python', C.H. Swaroop schließen:

„He [Guido] doesn't particularly like snakes that kill animals for food by winding their long bodies around them and crushing them.”

1.2 Der Autor

Die Entwicklung von Python geht auf Guido van Rossum zurück. Van Rossum ist gebürtiger Holländer und studierte an der Amsterdamer Universität, an der er auch 1982 seinen Master-Abschluss machte.

Nach seinem Abschluss blieb Van Rossum (der übrigens Wert darauf legt, dass das 'van' nur bei der alleinigen Nennung des Nachnamens groß, ansonsten aber generell klein geschrieben wird) am CWI³ der Amsterdamer Universität. Er war dort an mehreren Projekten beteiligt, unter anderem beschäftigte er sich

³Centrum voor Wiskunde en Informatica (Fachbereich für Informatik)

- von 1982 bis 1986 mit dem Design und der Implementierung von ABC, die als eine einfache Sprache, besonders für Programmieranfänger, konzipiert war und BASIC ersetzen sollte, sowie
- von 1986 bis 1991 mit der Entwicklung von Amoeba⁴.

Bis 1995 war Van Rossum noch weiter am CWI tätig, bevor er 1994 eine Stelle als Gastforscher am NIST⁵ in den USA antrat. Dort arbeitete er für das CN-RI⁶, bei dem er 1998 angestellt wurde. Von 2000 bis 2003 war er Leiter der PythonLabs, vorübergehend bei BeOpen.com, die aber später in der Zope Corporation aufgegangen sind. Seit Juli 2003 arbeitet er bei Elemental Security⁷ als 'Language Architect'.

Im Jahr 2002 erhielt er für seine Leistungen den Free Software Award der von Richard Stallmann gegründeten Free Software Foundation (FSF).

1.3 Die Geschichte

Viele Entwicklungsschritte von Python korrespondieren mit Guido van Rossums Lebenslauf. Der eigentlichen Entwicklung ab 1989 gehen allerdings wesentliche Vorarbeiten bzw. die Mitarbeit an mehreren Projekten voraus. Dies ist insofern relevant, da einige Hauptmerkmale von Python durch Van Rossums vorherige Projekte beeinflusst sind.

1.3.1 Gründe für eine neue Sprache

Einer der Anlässe für die Entwicklung von Python war Van Rossums Arbeit am Betriebssystemprojekt Amoeba des schon bekannten CWI. Für dieses wurden bis dahin hauptsächlich in C, oder als Shell-Skripte, kleinere Tools entwickelt. Hier zeigte sich einerseits, dass die Programmierung in C zeitaufwändig war, während andererseits Shell-Skripte zwar schneller programmierbar waren, allerdings meist nicht auf die von Amoeba implementierten Schnittstellen zugreifen konnten.

Da Guido van Rossum bereits aus dem vorangegangenen ABC⁸-Projekt am CWI fundierte Kenntnisse über die Entwicklung interpretierbarer Sprachen hatte, eine Weiterentwicklung von ABC aber aufgrund der begrenzten Erweiterungsmöglichkeit ausser Frage stand, beschloss Van Rossum an einer eigenen Sprache zu arbeiten. Diese sollte für Unix-Shell- und C-Programmierer gleichermaßen attraktiv sein, eine ABC-ähnliche Syntax haben und als eines der wesentlichen Kriterien den Zugang zu den Systemaufrufen von Amoeba bieten. Zwei der für die spätere Entwicklung der Sprache wichtigsten Eigenschaften waren mit Sicherheit die Erweiterbarkeit und Plattformunabhängigkeit, die zur heutigen Verbreitung der Sprache mit Sicherheit beigetragen haben. Unter anderem existieren mittlerweile Portierungen für folgende Plattformen:

⁴ein verteiltes Betriebssystem der Computer Systems Group / Uni Amsterdam

⁵National Institute of Standards and Technology, Gaithersburg, Maryland

⁶Corporation for National Research Initiatives, Reston, Virginia

⁷siehe auch: <http://www.elementalsecurity.com>

⁸Weitere Informationen über ABC: (<http://homepages.cwi.nl/~steven/abc/>)

- Windows (und DOS),
- Macintosh,
- Linux,
- und viele mehr: Sparc Solaris, OS/2, AIX, AROS, AS/400 (OS/400), BeOS, OS/390 und z/OS, Palm OS, QNX, VMS, Psion, RISC OS, VxWorks, Windows CE und Pocket PC

und als vielleicht ungewöhnlichste Anpassung

- für die PlayStation.

Nicht zuletzt sind in die Entwicklung von Python auch Aspekte der Sprachen Modula-2+ und Modula-3 eingeflossen, die nach Angabe von Guido van Rossum unter anderem in der Syntax der Ausnahmebehandlung auftauchen.

1.3.2 Die Evolution

Die Geburt von Python fand im Jahr 1989 statt und begann mit einem Projekt über die Weihnachtsfeiertage. Python war in dieser und der folgenden Zeit für Guido van Rossum hauptsächlich ein Freizeitprojekt mit dem Namen 'Python' als Arbeitstitel, das allerdings schon während seiner Entwicklung im Amoeba-Projekt mit zunehmender Intensität eingesetzt wurde. Viele der ersten Verbesserungen stammten dabei aus den Vorschlägen der Kollegen am Institut.

Python wurde daraufhin das erste Mal 1991, nach mehr als einem Jahr Vorarbeit, der Weltöffentlichkeit vorgestellt. Seit der ersten Veröffentlichung durch Guido haben aber auch eine große Zahl unabhängiger Programmierer zur weiteren Verbesserung und Erweiterung beigetragen. Dennoch ist Guido weiterhin der Hauptautor und wird laut [Wikil] in der Community als BDFL - Benevolent Dictator for Life⁹ bezeichnet.

Die weitere Entwicklung fand dann um 1995 am CNRI statt, aus der auch die Releases¹⁰ 1.3 bis 1.5.2 stammen. Im Jahr 2000, mit dem Wechsel von Guido und dem größten Teil des Entwicklerteams zu BeOpen.com entstand das Release 2.0. Im selben Jahr noch wurden die PythonLabs der BeOpen.com von Digital Creations übernommen, die heute unter dem Namen der Zope Corporation bekannt sind.

Im Jahr 2001 wurde schliesslich die PSF¹¹ gegründet, die seitdem als gemeinnützige Organisation die Rechte an Python hält und die Weiterentwicklung beginnend mit der Python Version 2.1 steuert. Die Zope Corporation unterstützt die PSF bei dieser Arbeit in finanzieller Hinsicht.

⁹engl.: wohlwollender Diktator auf Lebenszeit

¹⁰eng: Veröffentlichung, auch: Softwareversion

¹¹Python Software Foundation

1.4 Die Lizenz

Alle heutigen Versionen von Python sind Open Source Veröffentlichungen und unterliegen einer speziellen Python Lizenz, der PSL¹². Diese entspricht im Wesentlichen der GPL¹³ und wird allgemein auch als GPL kompatibel anerkannt.¹⁴ Einzige Ausnahme bilden einige Versionen um die Version 2.0 bei BeOpen.com. Hier finden sich einige nicht GPL-konforme Lizenzen, die aber heutzutage nicht mehr von großer Bedeutung sind.

Verwiesen sei hier auf den Schriftwechsel über die Lizenz für Python 2.1 zwischen Guido van Rossum und Even Moglen, Rechtsanwalt der Free Software Foundation, nachzulesen in [Python4]. Moglen bemängelte damals zwei Passagen der PSF-Lizenz, genauer gesagt die Punkte 7 und 8, in denen es um das bei Streitigkeiten anwendbare Recht und den Zeitpunkt für eine zwingende Annahme der Lizenzbestimmungen ging. Diese wurden für die späteren Veröffentlichungen nachweislich modifiziert.

Der gravierendste Unterschied der aktuellen PSF-Lizenzen zur GPL ist der Verzicht auf das 'Copyleft', den viralen Effekt der GPL, der eine Offenlegung von Code fordert, sobald Teile eines GPL-lizensierten Codes verwendet werden. Somit dürfen Module und selbst veränderte Versionen von Python auch als sog. Closed-Source verbreitet werden. Das heisst, eine Veröffentlichung des Source-Codes wird *nicht* gefordert. Kurz gesagt und etwas vereinfacht:

Python ist kostenfrei und selbst für eine kommerzielle Nutzung in Closed Source Projekten nutzbar.

Doch nun zu etwas völlig anderem...

¹²Python Software License

¹³Gnu Public License

¹⁴vgl. <http://www.python.org/psf/license.html>

2 Die Sprache Python

2.1 Allgemeines

Wie schon im Vorwort erwähnt, kann und soll diese Arbeit kein Tutorial sein. Sie soll vielmehr eine grundlegende Hinführung zum Thema Python darstellen und erste Anhaltspunkte für die weitere Beschäftigung mit dem Thema Open Source Programmierung in Python sein. Ziel ist also die Schaffung einer Grundlage, um nach dem Lesen dieses Artikels in der Lage zu sein, Python in das Umfeld der Programmiersprachen einzuordnen, Python zu installieren und erste, rudimentäre Kenntnisse der Sprache anwenden zu können.

2.2 Was ist Python? (Teil II)

Auf den Punkt gebracht ist Python eine sogenannte Interpretersprache für interaktive und objektorientierte Programmierung. Im Gegensatz zu den klassischen Sprachen wie C, C++ und Java wird der geschriebene Code folglich nicht kompiliert, sondern erst zur Laufzeit übersetzt. Die Interaktivität ist wahrscheinlich das ungewöhnlichste, was im Vergleich mit anderen Sprachen auffällt. Man könnte es vergleichen mit der Arbeit in einem permanenten Debug-Modus, d.h. alle Befehle die im Interpreter eingegeben werden, werden direkt ausgeführt. Dies ähnelt der Funktionalität einer Shell und geht auf den Wunsch Van Rossums zurück, einen Ersatz für die Sprache ABC zu schaffen. Die gleichzeitige Realisierung der Objektorientierung in einer Skriptsprache ist ebenfalls ungewöhnlich, näheres hierzu aber im Abschnitt 2.3.4.

Die Ziele Van Rossums bei der Entwicklung von Python waren vielfältig. Python sollte primär einfach und intuitiv sein und damit dem hauptsächlich in seiner Funktionalität unzulänglichen ABC folgen. Wie schon beschrieben, wollte Van Rossum auch C- und Unix-Programmierer überzeugen und die Mächtigkeit der Sprache möglichst nicht beschneiden, so dass es auch möglich ist, Python durch die Einbindung von C oder C++ Modulen zu erweitern. Um eine möglichst schnelle Etablierung der Sprache zu erreichen, wurde Python als Open Source-Lösung konzipiert, so dass prinzipiell jeder Interessierte mit Programmierkenntnissen bei der Weiterentwicklung der Sprache helfen kann. Weiterhin sollten mit Python auch besonders kurze Entwicklungszeiten erreichbar sein und Prototyp-Implementierungen unterstützt werden. Aus diesem Grund entstand wohl auch der interaktive Interpreter, mit dem es möglich ist, in kürzester Zeit kleine Lösungen im Dialog mit dem System zu erstellen.

Nicht zuletzt wollte Rossum mit Python aber auch eine Veränderung in der Code-Lesbarkeit erreichen. Während der Vorgänger ABC Prozeduren zum Beispiel noch mit "HOW TO RETURN <funktionsname>" eingeleitet hat, startet Python hier mit einem kurzen "def <funktionsname>". Nicht nur die Zeichenanzahl wurde damit geringer, Python verzichtet gleichzeitig auch auf Klammern zur Gruppierung einer Blockanweisung. Dies erhöht einerseits die Programmiergeschwindigkeit und verkürzt den Code weiter. Nach Meinung Guido van Rossums lässt es andererseits den Code auch lesbarer werden und erfordert daher

auch einen geringeren Dokumentationsaufwand¹⁵. Mit dieser und weiteren Besonderheiten beschäftigt sich der Abschnitt 2.3.5. Bevor wir aber in die Tiefe gehen, wollen wir zu allererst die Basis schaffen und eine Installation für die beiden Plattformen Windows und Linux (bzw. Mac OS X) beschreiben.

2.2.1 Download

Um sich wirklich mit der Sprache Python beschäftigen zu können, benötigt man zumindest eine lokale Installation der Software. Der Vorteil des Open Source Konzepts ist die freie Verfügbarkeit. Es ist folglich möglich, sich die jeweils aktuelle Version kostenlos von den Seiten der Python Software Foundation herunter zu laden. Der direkte Link zur aktuellen Version (derzeit 2.4.1) ist folgender:

<http://www.python.org/download>

Schon auf dieser Seite muss man sich entscheiden, welche Art der Installation für den eigenen Zweck am geeignetsten ist. Grundsätzlich liegt Python in Form des Quelltextes und binärer Distributionen¹⁶ vor. Wer sich die Mühe ersparen will Python selbst zu kompilieren und zu konfigurieren, bedient sich der binären Version. Hier zeigt sich schon etwas für die Plattform Windows sehr typisches: Hierfür liegen keine selbst kompilierbaren Sources¹⁷ vor, vielleicht um den Otto-Normal-Windowsanwender nicht zu sehr zu irritieren...

2.2.2 Installation

Installation unter Windows Windows-Nutzer installieren sich also die angebotene Version in .msi-Form, die sich bei bereits vorhandenem Microsoft-Installer ohne weitere Interaktion problemlos starten lässt. Von den Machern von Python wird hier übrigens auch eine weitere Variante empfohlen, nämlich die von Mark Hammond auf Sourceforge angebotene win32all-Version, derzeit unter

- http://sourceforge.net/project/showfiles.php?group_id=78018

zu finden. Sie kann auch direkt von den Starship-Seiten heruntergeladen werden:

- <http://starship.python.net/crew/mhammond/win32/Downloads.html>

Installation unter Linux Wer ganz im Sinne der Open Source nicht auf Windows arbeiten möchte, hat meist sogar noch weniger Arbeit. Viele der bekannten Linux-Distributionen, z.B. von SuSE, RedHat oder Gentoo liefern Python frei Haus auf einer der Installations-CDs bzw. via Internet-Installation. Viel mehr als ein Klick ist dann nicht mehr nötig und das entsprechende RPM¹⁸ wird installiert.

¹⁵vgl. Guido van Rossum in 'Foreword for "Programming Python", 1st edition, O'Reilly

¹⁶engl: binary distributions

¹⁷dt.: Quelltexte

¹⁸eine Installationsdatei des RPM-Paket Managers, näheres hierzu auf <http://www.rpm.org>

Wer sich nicht auf eine Distribution einlassen will, dem bleibt noch das Paket als 'gezippter tarball'¹⁹, beispielsweise per Download mit *wget*²⁰:

```
mephisto:~ # wget www.python.org/ftp/python/2.4.1/Python-2.4.1.tgz
```

Extrahiert sind die Dateien schnell, z.B. mit dem Befehl:

```
mephisto:~ # tar -zxf Python-2.4.1.tgz21
```

Nach einem Lauf des Konfigurations-Skripts 'configure':

```
mephisto:~/Python-2.4.1 # ./configure
```

in dem alle nötigen Libraries²² zur Kompilierung geprüft werden (für Python durchaus umfangreich und kann auf einem langsamen System durchaus mehrere Minuten dauern), liegt der Code (hoffentlich) fertig zur Kompilierung vor. Sollte dies nicht der Fall sein, müssen je nach Angabe des Skripts, die noch fehlenden Pakete bereitgestellt werden. Hier hilft je nach Distribution das zugehörige Installationstool. Alternativ müssen die Pakete ebenfalls analog kopiert und eventuell kompiliert werden. Wenn schließlich alle Pakete vorliegen und der Konfigurationsdurchlauf erfolgreich war, müssten die folgenden Dateien im angegebenen Verzeichnis zu finden sein:

```
./config.status  
./Modules/Setup.config  
./pyconfig.h  
./Makefile
```

Das Vorliegen der Dateien kann man auch aus der Ausgabe des Configure-Skripts ablesen. Wenn in den letzten Zeilen keine Fehler gemeldet werden, ist alles gut gegangen. Also geht es direkt weiter mit der Kompilierung. Diese startet sich ganz leicht mit

```
mephisto:~/Python-2.4.1 # make
```

Jetzt folgen im Idealfall viele Zeilen mit Compileranweisungen und Informationen, die für den durchschnittlichen Nutzer vermutlich wenig Aussagekraft haben. Nach mehreren Minuten (oder Stunden, je nach System-Leistung) endet der Durchlauf dann in einer binären Fassung der Python- Distribution. Auf der hier verwendeten Test-Maschine *mephisto*, einem alten Pentium II-266, lief die Kompilierung übrigens während der gesamten Arbeit an diesem Absatz.

Sollte es bei einem der vorangegangenen Schritte Probleme gegeben haben, ist die Datei README im Hauptverzeichnis der Distribution die erste und beste Anlaufstelle - hier werden viele der bekannten Probleme zumeist auch mit

¹⁹Endung .tgz, also eine Datei des tar-Archivers mit gzip gepackt.

²⁰GNU-Wget: Download z.B. unter <http://www.gnu.org/software/wget/wget.html>

²¹GNU-tar: <http://www.gnu.org/software/tar/tar.html>

²²dt.: Bibliotheken

einem Workaround²³ genannt. Beispielsweise scheint es in älteren Versionen in der virtuellen Linux-Umgebung Cygwin²⁴ (übrigens auch ein Produkt unter der GNU-Public License) Probleme mit dem Build²⁵ gegeben zu haben. Diese konnten dann mit einer der unzähligen Configure-Optionen, die im allgemeinen mit '--' starten, gelöst werden. Dies aber nur als ein Beispiel des möglichen Troubleshooting²⁶ bei Fehlern im Kompilierungslauf, schliesslich soll die Installation nur einer der Punkte sein, auf die hier eingegangen wird.

Auf unserer Testmaschine, einem SuSE-Linux 9.3-PC, lief der Build übrigens ohne gravierende Probleme. Einzig eine Warnung über die unsichere Verwendung des Posix-Befehls *tempnam* war zu sehen, was uns hier aber nicht weiter stören soll und die Kompilierung auch nicht weiter behindert. Mit den Meldungen

```
running build_scripts
creating build/scripts-2.4
[...]
changing mode of build/scripts-2.4/pydoc from 644 to 755
changing mode of build/scripts-2.4/idle from 644 to 755
changing mode of build/scripts-2.4/smtplib.py from 644 to 755
mephisto:~/Python-2.4.1 #
```

ging der Lauf zu Ende, die Kompilierung war also erfolgreich und python kann gestartet und verwendet werden:

```
mephisto:~/Python-2.4.1 # ./python
Python 2.4.1 (#1, May 13 2005, 18:38:09)
[GCC 3.3.5 20050117 (prerelease) (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Wie man aus obiger Startmeldung sehen kann, handelt es sich hier um Python 2.4.1 mit Build-Nummer 1, vom 31. Mai 2005, auf einem SuSE Linux System.

Der Aufruf von `python` im selben Verzeichnis und ohne das führende `./` startet im übrigen die von der Distribution mitgelieferte Python-Variante, die ein wenig älter und auf dem Stand des Python-Release 2.4 ist:

```
mephisto:~/Python-2.4.1 # python
Python 2.4 (#1, Mar 22 2005, 21:42:42)
[GCC 3.3.5 20050117 (prerelease) (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Diese Ausführungen sollen dem geneigten Leser beweisen, dass es mittlerweile ohne großen Aufwand und Spezialwissen möglich ist, von "null auf Python" zu kommen und wir hier auch nicht getrickst haben.

²³engl. für: Möglichkeit das Problem zu umgehen

²⁴<http://www.cygwin.com>

²⁵dt.: Bau - meint den Vorgang des Kompilierens und Linkens zur ausführbaren Datei

²⁶dt.: Fehlerbehandlung

2.3 Einführung in Python

2.3.1 Tutorials

Nachdem uns nun eine lauf- und verwendungsfähige Installation von Python vorliegt, bleibt die Frage 'Was nun?'.

Da wir annehmen, dass unsere Leser keine Profis der Python-Programmierung sind, die vermutlich nur im Kapitel über die Geschichte von Python etwas neues finden würden, möchten wir auf einige gute Ausgangspunkte für das weitere Lernen hinweisen.

Intuitiv startet man selbstverständlich auf den Web-Seiten der Python Foundation mit dem Tutorial des Python-Erfinders Guido van Rossum unter

<http://www.python.org/doc/2.4.1/tut/tut.html>.

Mit Sicherheit ist dieses Tutorial auch ein umfassender Überblick über die Leistungsfähigkeit von Python. Besser gefallen hat uns zum Einstieg aber das Buch "Dive into Python" von Mark Pilgrim, welches insbesondere in der Online-Fassung durch die klare Gliederung, eine Vielzahl an Beispielen mit farbllichem Syntax-Highlighting und eine Suchfunktion auch als Nachschlagewerk überzeugt. Wer einen Blick riskieren möchte, kann dies auf

<http://www.diveintopython.org/toc/>

tun. Das Buch gibt es zudem in einer gedruckten Fassung zum Kauf z.B. über amazon.de.

Für alle darüber hinausgehenden Ansprüche möchten wir auf die zahlreichen und umfangreichen Dokumentationen auf

<http://www.python.org/doc>

und den FH-Seiten zur Veranstaltung von Herrn Högl

<http://www.fh-augsburg.de/~hhoegl/oosw/oosw.html>

verweisen, die ausreichend Material bieten, um mehr als nur Anfängerkwissen zu erwerben.

2.3.2 Anwendungsgebiete

Python kann prinzipiell aufgrund der breiten Modulunterstützung für viele Anwendungsarten eingesetzt werden. Populär geworden ist Python vor allem aber mit der Entwicklung und Realisierung von Zope²⁷. Dennoch eignet sich Python nicht nur für die Internetprogrammierung wie in Abschnitt 3.1 beschrieben, sondern auch für den Entwurf graphischer Oberflächen oder selbst zur Datenbankprogrammierung. In der Datenbank PostgreSQL können beispielsweise Prozeduren außer in der von Oracle bekannten Sprache pl/SQL, auch in PL/Python geschrieben werden.

²⁷vgl. Abschnitt 3.1.1 zum Thema Zope.

Ein weiteres Anwendungsgebiet ist die Programmierung von Anwendersoftware auf grafischen Oberflächen. Hierzu steht eine große Zahl an Bibliotheken zur Verfügung, mit deren Hilfe verschiedene Frameworks²⁸ verwendet werden können. Als Framework wird hierbei häufig eine Zusammenstellung von Klassen und Modulen verstanden, die einen festen Anwendungsrahmen bzw. das Grundgerüst eines Programms darstellen. Häufig findet man diese Frameworks in den Bereichen umfangreicher Nutzerschnittstellen. Der Begriff selbst stammt v.a. aus der objektorientierten Programmierung. Hier einige Beispiele:

- gtk
- Java Swing
- Qt
- wxWindows
- uvm.

Hinweise zu den benötigten Modulen bietet wiederum der Server der Python Foundation unter

<http://wiki.python.org/moin/GuiProgrammingpython.org>.

Ein Beispiel für eine grafische Anwendung, die sich zudem noch mit der Bildbearbeitung beschäftigt, ist das unter der LPGL²⁹ lizenzierte 'Skencil'³⁰ für Vektorgrafiken, mit dem auch die für diesen Artikel gezeichneten Grafiken erstellt wurden, wohlwissend, dass mit Gimp durchaus mächtige Konkurrenz aus dem nicht-Python Lager existiert.

Mit einigen weiteren Applikationen, unter anderem auch zum Thema Netzwerkprogrammierung und Systemwerkzeuge, beschäftigt sich später noch das Kapitel 3.

Neben den umfangreichen Möglichkeiten zum Einsatz in Python geschriebener Software existieren auch noch python-eigene Charakteristika, die den Einsatz befürworten. Python zeichnet sich beispielsweise durch die Möglichkeit aus, umfangreiche Anwendungen³¹ oder auch Prototypen³² schnell zu entwickeln. Mit den Details zu diesen Besonderheiten beschäftigt sich im Anschluß an die folgende kurze Einführung in die Sprache Python das Kapitel 2.3.5 bzw. der Vergleich von Python mit anderen Hochsprachen im Kapitel 2.4.

²⁸Weiteres hierzu auch auf Wikipedia: <http://www.wikipedia.org>

²⁹GNU - Library (bzw. Lesser) General Public License

³⁰siehe: <http://www.skencil.org>

³¹engl.: rapid application development

³²engl.: rapid prototyping

2.3.3 Die Sprachsyntax

In diesem Kapitel wollen wir uns der Sprache Python selbst nähern. Nachdem wir in Abschnitt 2.2.2 eine Installation durchgeführt und mit dem Kommando

```
./python
```

eine Instanz des Interpreters gestartet haben, können wir nun mit einigen Beispielen die Grundlagen der Python-Programmierung testen.

Zuweisung / Ausgabe einer Zeichenkette

Eine Zuweisung ist neben der Ausgabe von 'Hello world' die wohl einfachste Operation. Nachdem allerdings kein Tutorial ohne die obligatorische Ausgabe einer Zeichenkette auskommt, hier unser Versuch:

```
print "spam" // Ausgabe: spam
```

Mit dem Befehl *print* kennen wir nun schon den einfachsten Weg Variablenwerte gleich welcher Art auf dem Bildschirm auszugeben. Dieses Wissen ist hilfreich, um in den folgenden Beispielen die jeweils veränderten Variablenwerte ausgeben zu können.

Nachdem der Pflicht nun genüge getan ist, folgt gleich die erste Zuweisung:

```
i = -1
```

Wer diesen Befehl im laufenden Interpreter eingegeben hat, wird bemerkt haben, dass sich Python hier nicht gewehrt hat. Hier zeigt sich bereits der Vorteil Variablen nicht deklarieren zu müssen. Die Variable *i* wurde mit der Eingabe erzeugt und mit dem Wert '-1' als Integer initialisiert. Dies lässt sich mit dem Befehl

```
type(i) // Ausgabe: <type 'int'>
```

sofort überprüfen. Kommen wir nun zu den ersten Operatoren:

Vergleichsoperationen

Kontrollstrukturen unterscheiden sich in Python nicht viel von denen anderer Sprachen und lassen sich somit analog verwenden. Auch hier gibt es die Vergleichsoperatoren *<*, *>*, *<=*, *>=*, *==*, *!=*, *<>*, *is*, *is not*. In Python können zudem Objekte unterschiedlicher Datentypen (z.B. `0 == 0.0`)³³ und auch Strings (z.B. `"actor" == "troop"`) miteinander verglichen werden - hier unterscheidet sich Python bereits von einigen anderen Sprachen.

Diese Vergleiche machen allerdings nur dann Sinn, wenn sie nicht zu unvereinbaren Datentypen gehören. Der Vergleich von ungleichen Datentypen (z.B. `"spam" == 3`) liefert daher *FALSE*, was allerdings wenig überraschen dürfte.

Ein kleiner Hinweis noch - Vergleiche können auch beliebig oft hintereinander verkettet werden:

³³also ein Integer mit einem Floating-Wert

```
4 < 5 < 56 < 200 // Ausgabe: True
```

Eine Übersicht über die wichtigsten Datentypen liefert die folgende Graphik:

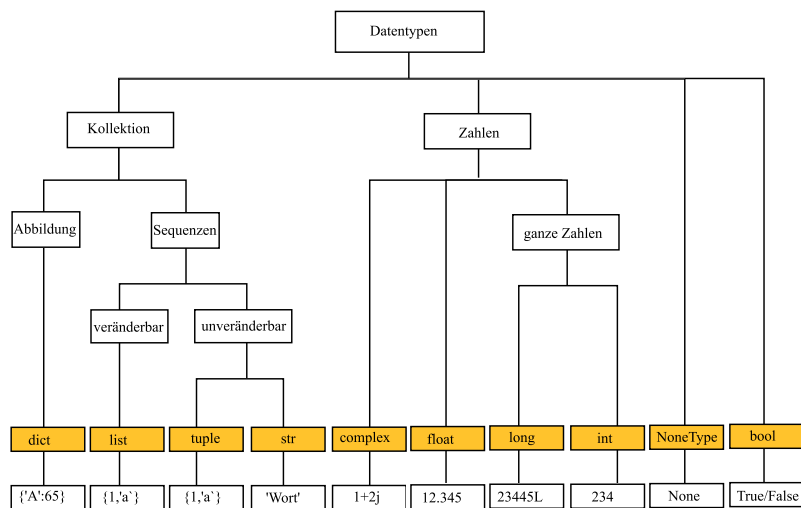


Abbildung 1: Datentypen von Python, modifiziert nach [Weigend]

Ein Beispiel zum jeweiligen Datentyp zeigt die unterste Zeile der Darstellung.

Verzweigung

Eines der wichtigsten Konstrukte einer Programmiersprache ist die Verzweigung, die in Python mit *if* eingeleitet wird. Aus anderen Programmiersprachen kennt man noch zusätzlich die Case-Anweisung, die für eine mehrfache Verzweigung verwendet wird, in Python allerdings nicht implementiert ist, da der Zweck der Anweisung auch auf anderem Wege erreicht werden kann. Hier zeigt sich das Ziel, keine redundanten Implementierungswege zuzulassen. Die unten dargestellten Beispiele demonstrieren die Funktionsweise einer einfachen, zweifachen und mehrfachen Verzweigung in Python (die jeweiligen Werte können wie gehabt mit `print` ausgegeben werden):

<pre>a) einfach: i = -1 if i < 0: i = -i // Ergebnis: i == 1</pre>	<pre>b) 2-fach: i = -1 if i > 0: i = -i else: i = 3 // Ergebnis: i == 3</pre>	<pre>c) mehrfach: actor = "Cleese" if actor == "Chapman": preis = 40 elif actor == "Cleese": preis = 30 elif actor == "Jones": preis = 35 // Ergebnis: preis == 30</pre>
---	--	--

While und For

Eine Wiederholungsanweisung kann auf zweifache Art und Weise realisiert werden. Zum einen gibt es in Python eine bedingte Wiederholung mit *while*, die solange ausgeführt wird, bis diese Bedingung nicht mehr zutrifft:

```
preis = 11
if preis > 10:
    preis = -preis
print preis // Ergebnis: preis = -11
```

Zum anderen kann die Iteration auf eine Menge von Elementen (Listen, Tupel, String) angewendet werden. Die For-Schleife führt dann ihre Anweisungen solange aus, bis alle Elemente durchlaufen sind. Die Syntax der For-Schleife ist:

```
for actor in actorlist:
    print actor
```

Besonders hilfreich im Zusammenhang mit der For-Schleife ist die Funktion *range(n)*. Der Aufruf von *range* erzeugt nämlich eine Liste von ganzen Zahlen von 0 bis n-1, so dass eine Anweisung n mal ausgeführt wird.

Def

Mit der Anweisung *def* wird die Definition einer Funktion eingeleitet, die aus dem Funktionskopf und dem Anweisungsblock besteht:

```
def print_name (name)
    print name
```

Der Anweisungsblock enthält dann (engerückt) die durch die Funktion auszuführenden Operationen. Sie wird aufgerufen, indem der Name der Funktion angegeben wird, gefolgt von einer Liste der nötigen Parameter in runden Klammern:

```
print_name (name)
```

Funktionen akzeptieren in der Regel nur die in der Funktionsdefinition festgelegte Anzahl an Argumenten, die in der durch die Definition vorgegebenen Reihenfolge übergeben werden müssen.

Mit diesem grundlegenden Wissen lassen sich bereits kleinere Programme schreiben. Wer sich von diesem wirklich nur rudimentären Tutorial und von der Sprache Python hat fesseln lassen, kann sein Wissen auf vielfältigem Weg erweitern. Ein Option bietet noch der nächste kleine Abschnitt über die Objektorientierung, der aber sicher auch nur absolute Grundlagen schaffen kann und soll. Danach empfehlen wir die schon in Kapitel 2.3.1 genannten Tutorials zum weiteren Studium.

2.3.4 Objektorientierung

Die objektorientierte Programmierung selbst ist ein sehr umfangreiches Thema, über das schon viele und vor allem auch umfangreiche Bücher geschrieben wurden. Die folgenden Begriffe und Erläuterungen sollen diese Thematik daher ebenfalls lediglich in Ihrer Essenz wiedergeben, dies allerdings soweit möglich aus dem besonderen Blickwinkel eines Python-Programmierers.

Die wesentlichen und oft gehörten Schlagworte im Kontext der objektorientierten Programmierung sind:

- Klasse,
- Objekt,
- Polymorphie,
- Kapselung und
- Vererbung.

Deshalb hier nun eine Begriffserläuterung mit einigen Python-Beispielen aus unserer Feder, die ihre Attributbezeichnungen³⁴ nicht von ungefähr haben...

Klasse: Die Klasse wird in Python durch den Kopf beginnend mit dem Wort *class* und durch den Körper definiert. Optional befindet sich nach dem Klassennamen eine Liste von Oberklassen, die ihre Methoden und Attribute an diese Klasse “vererben”, d.h. weitergeben. Der Körper selbst besteht aus den Klassenattributen, einem Konstruktor und einer Anzahl von Methoden. Konstruktor ist eine besondere Methode der Klasse, die bei der Instanzierung (Definition eines Objekts) aufgerufen wird, um die Objektattribute mit Anfangswerten zu belegen. Als Methode werden üblicherweise die aus der strukturierten Programmierung bekannten Funktionen und als Attribute die entsprechenden Variablen bezeichnet.

Eine Klasse kann öffentliche und private Attribute haben. Sie unterscheiden sich dadurch, dass auf die privaten Attribute außerhalb einer Klasse nicht direkt zugegriffen werden kann, während öffentliche Attribute (oder auch Methoden) die Schnittstelle der Klasse nach aussen bilden und nutzbar sind. Private Attribute sind in Python typischerweise an den Unterstrichen vor dem Attributnamen zu erkennen.

```
class comedian(troop):
    def __init__(self, name):
        self.name = name
    def do_sketch(self, story):
        print story, ' makes people laugh. '
        print self.name, ' is good.'
```

³⁴Variablen werden typischerweise in der objektorientierten Programmierung als Attribute bezeichnet und in Bezug auf eine Klasse oder Methode definiert. Näheres hierzu im folgenden Kapitel.

Mit dieser Definition wurde eine Klasse *comedian* erzeugt, die zwei Methoden besitzt. Die erste Methode ist ein Konstruktor, der die Variable *name* mit dem Anfangswert belegt. Die zweite Methode gibt einen Text aus: *<story> makes people laugh. <name> is good.* Das erste Argument *self* darf bei keiner Methodendefinition fehlen und bezeichnet das aktuelle Objekt. Diese Konvention, Argumentlisten mit *self* zu übergeben, stellt eine der Eigenheiten von Python dar. Den Grund dafür findet man mit etwas Recherche auf python.org oder im späteren Kapitel 2.3.5.

Objekt: Zur Erzeugung von Objekten muss eine Klasse aufgerufen³⁵ und eine Referenz auf diese Instanz erzeugt werden. Als Argumente werden hierbei die Parameterliste des Konstruktors “ *__init__* ” übergeben. Eine Instanz (Objekt) der oben definierten Klasse stellt sich wie folgt dar:

```
leading_actor = comedian('Graham Chapman')
```

Leading_actor enthält danach eine Referenz auf den *comedian* 'Graham Chapman'. Der Zugriff auf die Methoden und die öffentlichen Attribute des Objekts erfolgt dann durch die Angabe des Objektnamens, eines Punktes und dem Namen des Attributs oder der Methode:

```
leading_actor.do_sketch('The Meaning of Life')
// The Meaning of Life makes people laugh. Graham Chapman is good.
```

Polymorphismus: Polymorphismus ist ebenfalls ein Konzept der OO- Programmierung und bedeutet, dass derselbe Name für Methoden in unterschiedlichen Klassen verwendet werden kann, sich dahinter aber durchaus objektspezifische Operationen verbergen können. In diesen Zusammenhang fällt auch das Überladen von Operatoren. Ein Beispiel hierfür ist bereits das Pluszeichen (+). Dieses Zeichen kann üblicherweise gleichermaßen für arithmetische³⁶, als auch für String-Operationen³⁷ verwendet werden. In Python werden Standardfunktionen überladen, indem der Name der Methode mit einem doppelten Unterstrich beginnt und endet z.B. *__add__* .

```
class comedian(troop):
    def __init__(self, name):
        self.name = name
    def do_sketch(self, story):
        print story, ' makes people laugh. '
        print self.name, ' is good.'
    def __add__(self, comedian):
        return self.name 'and' comedian.name
```

³⁵auch: instanziiert

³⁶hier eine Addition

³⁷oft auch als Konkatenation bzw. Verkettung bezeichnet

Werden nun zwei Instanzen der Klasse `comedian` mit einem Pluszeichen verknüpft, so werden die Namen der beiden Objekte mit einem `'and'` verknüpft und das Ergebnis (ein String) zurückgegeben. Besser wäre in diesem Beispiel noch der Aufruf einer Methode, z.B. `get_name()`, so dass das Attribut `'name'` nicht mehr öffentlich ist. Der Grund hierfür liegt in der folgenden Anforderung, dem Geheimnisprinzip.

Geheimnisprinzip: Das Geheimnisprinzip³⁸ sagt aus, dass der Zustand eines Objekts und seine Methoden nach außen nicht sichtbar sind. Der Zustand kann folglich im allgemeinen nur durch die Methoden der Klasse verändert werden. Ausnahmen stellen in Python die Attribute dar, die nicht mit führendem Tiefstrich beginnen und daher öffentlich, d.h. von aussen sichtbar sind.

Kapselung: Das Besondere der OO-Programmierung ist, dass sie Daten und Methoden, die logisch zusammenhängen, in einer Klasse einschließt. Dieses Vorgehen nennt man *Kapselung*³⁹. Die Klasse `actor` enthält hier also das Attribut `name` und die Methode `do_sketch`.

Vererbung: Vererbung schließlich beschreibt die Beziehungen von Klassen untereinander. Eine Unterklasse kann beispielweise die Eigenschaften (Daten und Methoden) der Oberklasse erben und weiter verfeinern. Sie kann auch die Methoden der Oberklasse überschreiben. Dies geschieht dann, wenn eine Unterklasse eine Methode mit dem selben Namen wie die Oberklasse definiert. Für die Instanz der Unterklasse gilt dann alleine die Methodendefinition der Unterklasse. Dies soll nun am folgenden, etwas abgewandelten Beispiel verdeutlicht werden:

```
class troop():
    def __init__(self, name)
        self.name = name
    def do_sketch(self, story)
        print story, ' is good.'
class comedian(troop):
    def __init__(self, name):
        self.name = name
    def do_sketch(self, story):
        print self.name, ' makes people laughing. '
        print story, ' is good.'
```

In diesem Beispiel ist `troop` die Oberklasse. Die Klasse `comedian` erbt die Methode `do_sketch` von `troop` und überschreibt sie. Während folglich alle Mitglieder der Comedy-Truppe ganz offensichtlich 'gut' sind, schaffen es nur die Objekte der Klasse `comedian` wirklich, die Zuhörer zum lachen zu bringen.

³⁸im englischen auch als 'information hiding' bezeichnet

³⁹engl.: encapsulation

2.3.5 Besonderheiten in Python

Einrückungen Bei der Entwicklung von Python war ein Ziel von Guido van Rossum, den Quelltext so übersichtlich wie möglich zu halten. Die Einrückungen tragen dazu enorm bei. Python ist damit im Gegensatz zu Sprachen wie C, C++, oder Java eine formatbehaftete Sprache.

Ein in C geschriebenes Beispiel verdeutlicht die Problematik einer Sprache ohne Formatierungszwang:

```
for (i=0; i<10)
    i++;
    printf("%i, i);
```

Man würde hier erwarten, dass `i` 10 mal ausgegeben wird. Tatsächlich wird aber `i` nur 10 mal erhöht und erst dann ausgegeben, da nur das `++` logisch zur Schleife gehört, das eingerückte `printf` dagegen nicht. In Python beginnt im Gegensatz dazu jeder zusammenhängende Block mit einer Einrückung und kann auf den ersten Blick als solcher identifiziert werden.

```
for i in range(10):
    print i
```

In diesem Beispiel wird `i` folglich 10 mal ausgegeben.

Man könnte die Problematik im ersten Beispiel zwar durch die Verwendung der geschweiften Klammer umgehen, dadurch würde der Code aber auch unruhiger werden und wäre schwerer zu lesen. Die Einrückung erspart dies, bezahlt wird es allerdings mit dem Nachteil, dass ein fehlendes Leerzeichen im Python-Code einen Block beenden kann. Hier ist also besondere Vorsicht geboten!

Self in der Methodendefinition `self` ist ein Parameter, der die Instanz eines Objekts enthält. Es gibt eigentlich drei Gründe, warum `self` verwendet wird. Zuerst signalisiert `self`, dass ein Attribut oder eine Methode der aktuellen Klasse verwendet wird:

```
class boa(reptiles):
    self.name
```

Zweitens ist `self` notwendig, wenn eine überschriebene Methode der Oberklasse aufgerufen werden soll.

```
boa.methode(self,argument_1, argument_n)
```

Im oberen Beispiel ist eindeutig definiert, dass die Methode der Unterklasse `boa` und nicht der Oberklasse `reptiles` aufgerufen wird.

Schließlich werden Variablen in Python nicht deklariert, so dass der Interpreter nach den Variablen im Vererbungsbaum suchen müsste. Die Anweisung `self` zeigt dem Interpreter, dass es sich um eine Variable in der Klasse selbst handelt.

Neben- oder Seiteneffekte Folgende Ausdrücke sind in Python nicht zulässig:

```
while (x[a] = a--){
    ...
};
```

Warum? Der Grund ist nicht so leicht zu erkennen, liegt aber in der schwierigen Fehlersuche und dem Ziel, möglichst verständlichen Code zu erzeugen. Im aktuellen, etwas konstruierten Beispiel wird innerhalb eines Ausdrucks eine Zuweisung und eine Operation auf eine Variable ausgeführt. An dieser Stelle kann es zu sog. Nebeneffekten kommen: Es ist nämlich nicht klar, ob zuerst inkrementiert und dann zugewiesen wird oder umgekehrt. Dies bedeutet, es ist nicht klar erkennbar, ob erst `x[a]` bestimmt, oder `a--`⁴⁰ ausgeführt wird. Man weiss also nicht, welchem Element in `x` der Wert von `a--` zugewiesen wird. Das Ergebnis ist abhängig vom verwendeten Compiler. Diese Unklarheit verhindert Python. Obiges Problem liesse sich übrigens auch in C vermeiden, indem die Dekrementierung innerhalb des Anweisungsblocks der Schleife durchgeführt würde.

Switch- und Case-Anweisungen In Python gibt es keine Switch- oder Case-Anweisungen, da diese Anweisungen mit

```
if ... elif ... elif ... else
```

umgangen bzw. nachgebildet werden können. Zusätzlich können für viele Möglichkeiten 'dictionaries'⁴¹ definiert werden:

```
funktions_sammlung = {'a': funktion1, 'b': funktion2, ...}
```

Der Aufruf eines Dictionaries unter Übergabe des Keys⁴² (hier also z.B. des Wertes `a`) resultiert im Aufruf der im Dictionary hinterlegten Funktion (`funktion1`).

Tupel und Listen Generell sind Tupel und Listen sehr ähnlich und werden in vielen Programmiersprachen synonym verwendet. Python nimmt hier eine Unterscheidung vor, aus der zwei unterschiedliche Typen resultieren:

Ein Tupel ist eine Ansammlung von unterschiedlichen Datentypen, die in einer Beziehung zueinander stehen. Die Anschrift einer Firma mit dem Straßennamen, der Hausnummer, dem Ort und der Postleitzahl würde daher in einem Tupel abgelegt werden. Listen dagegen beinhalten definitionsgemäß eine Menge gleichartiger Datentypen, z.B. eine lange Liste von Städtenamen in Europa.

Mit einer Aufweichung dieser Regel können Tupel und Listen identisch abgebildet werden, was in Sprachen wie beispielsweise PHP so auch realisiert wird. Meist werden diese Listen dann auch einfach als Array⁴³ bezeichnet.

⁴⁰`a--` entspricht in C / C++ der Zuweisung `a = a - 1`

⁴¹dt.: Wörterbücher - eine Form der Listen in Python.

⁴²dt.: Schlüssel

⁴³dt.: Anordnung / Datenfeld

2.4 Vergleich mit anderen Sprachen

2.4.1 Skript- vs. Compilersprachen

Der Unterschied zwischen Compiler- und Skriptsprachen ist die Art und Weise wie Programme in ausführbaren Code übersetzt werden. Bei kompilierten Sprachen wie C, C++, Pascal oder Java übernimmt der Compiler diese Aufgabe. Bei den Skriptsprachen wie Perl, PHP oder Python muss dagegen ein Interpreter vorhanden sein.

Ein Compiler übersetzt vor der Ausführung den kompletten Programmtext und erzeugt ein direkt ausführbares Programm, das vom Betriebssystem geladen und gestartet wird. Hierbei werden die Besonderheiten des Rechners, auf dem das Programm läuft, berücksichtigt. Auf diese Weise entstehen für jede Plattform unterschiedliche Versionen des Programms, die nur auf der Zielplattform (z.B. Windows oder Unix) ausgeführt werden können.

Ein Interpreter dagegen liest einen Programmtext Zeile für Zeile und führt jede Anweisung direkt aus. Zum Start des Programms muss hierzu zuerst der Interpreter aufgerufen werden. Für jedes Betriebssystem gibt es zur Programmiersprache demnach auch einen eigenen Interpreter.

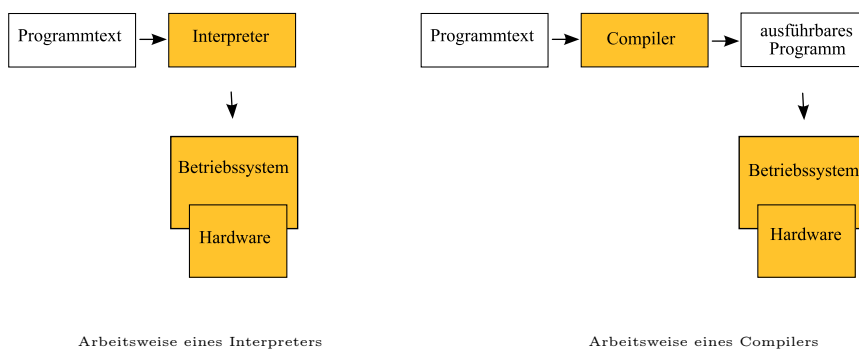


Abbildung 2: Übersetzungszeitpunkt Compiler und Interpreter

Der Vorteil der Skriptsprachen ist, dass ein einmal geschriebenes Programm so auf allen, vom Interpreter unterstützten Rechnerplattformen läuft. Einer der wesentlichen Unterschiede ist die in der Art der Ausführung bzw. Kompilierung begründete Tatsache, dass der Quelltext bei Interpreter-Sprachen automatisch offen und damit jederzeit einsehbar und änderbar ist. Eine Weitergabe des Programms bedingt dadurch *immer* auch die Weitergabe des Quelltexts.

2.4.2 Python vs. PHP

Wie bereits oben erwähnt, gehören Python und PHP zu den Skriptsprachen, weshalb sich ein Vergleich hier besonders anbietet. Für beide Sprachen gilt,

dass sie leicht zu lernen sind, auf den unterschiedlichsten Plattformen laufen und umfangreiche Funktionsbibliotheken besitzen. Zusätzlich können in beiden Sprachen C-, C++- und Java-Skripte leicht eingebunden werden. Sowohl PHP als auch Python haben mittlerweile eine große Entwickler- und Fan-Gemeinde. “Religionskriege” sollten deshalb aber nicht geführt werden, schließlich ist die beste Programmiersprache zumeist die, in der man sich auskennt⁴⁴.

Nichtsdestotrotz gibt es zwischen diesen Sprachen Unterschiede, beginnend bei der Sprachsyntax. Die Syntax von PHP ähnelt der von C oder Perl, mit vielen Kommata, Klammern und Dollarzeichen, auf die Python zum Zwecke der Lesbarkeit fast gänzlich verzichtet. Einzig der Beginn eines Anweisungsblocks, z.B. nach einem IF, WHILE, usw. beginnt nach einem Doppelpunkt. Auf Konstrukte wie “do ... while” oder die Switch-Anweisung wurde in Python ebenfalls verzichtet, da sie durch andere Anweisungen ersetzt werden können.

Eine Erweiterung im Vergleich zu PHP hat Python in der Nutzung von Namensräumen⁴⁵ erfahren, die in PHP komplett fehlen. Bibliotheksfunktionen können in PHP folglich aus allen Methoden und Klassen aufgerufen werden. Oft wird die Zugehörigkeit einer Funktion lediglich durch einen Präfix im Funktionsnamen symbolisiert.

Beispielsweise wird die Verbindung auf die Datenbank *MySQL* mit dem Befehl “*mysql_connect()*” und auf die Datenbank *MSQL* mit “*mysql_connect()*” durchgeführt. Beide Funktionen sind also ohne zusätzliches “Include”⁴⁶ eines Moduls bzw. Namensraums aufrufbar. In Python muss dagegen der entsprechende Namensraum vor der Verwendung mittels des Befehls `import` eingebunden werden.

Auch im Hinblick auf die Datentypen gibt es Unterschiede. PHP besitzt hier, wie schon erwähnt, nur einen Vektortypen, der sowohl eine Liste als auch ein Dictionary sein kann.

Hier ein Beispiel in PHP:

<p><i>Eine einfache Liste:</i></p> <pre><?php \$liste = array(1, 2, 3, 4, 5); ?></pre>	<p><i>Ein Dictionary:</i></p> <pre><?php \$liste = array("monty"=>"python", 12=>"spam"); echo \$arr["monty"]; // python echo \$arr[12]; // spam ?></pre>
--	--

Analog die verschiedenen Vektortypen in Python:

<p><i>Ein Tupel:</i></p> <pre>tupel_monty = ('Monty', 'Python', 4)</pre>	<p><i>Eine Liste:</i></p> <pre>liste_snake = ('Python', 'Boa', 'Anakonda')</pre>
--	--

⁴⁴lt. Dressler, in Vorlesung ‘Software Engineering’ der FH-Augsburg, Sommersemester 2004.

⁴⁵engl.: namespaces

⁴⁶dt.: Einbindung

Ein Dictionary :

```
dict_python = ('Python' : 'Prog.Language', 'Boa' : 'Snake')
```

Python besitzt hier im Gegensatz zu PHP unterschiedliche und strukturierte Datentypen (str, tuple, list, dict).

Auch der Einsatzschwerpunkt und damit der Fokus der weiteren Entwicklung unterscheiden PHP und Python. PHP wird entsprechend des zur Verfügung stehenden Funktionsumfangs hauptsächlich für Web-Anwendungen verwendet, ein Einsatz außerhalb dieses Bereichs ist zwar auch möglich aber umständlich. In dieser Hinsicht ist Python weit eher ein “Allrounder”, der für fast alle Anwendungen geeignet ist. Diese Unterschiede sind teilweise in der Entstehungsgeschichte der beiden Programmiersprachen begründet:

Während in Python viele Themen, wie z.B. die Objektorientierung und strukturiertes Exception-Handling, von Anfang an berücksichtigt und geplant wurden, sind diese in PHP erst deutlich später hinzugefügt worden. Beispielsweise ist das Exception-Handling erst seit der aktuellen Version 5 implementiert.

2.4.3 Python vs. C++ / Java

C++ und Java sind typische Vertreter der objektorientierten Compiler-Sprachen. Der Hauptunterschied zu den Skriptsprachen stellt dabei der Zeitpunkt der Kompilierung dar. Dies wurde bereits im Abschnitt 2.4.1 erläutert. Dadurch, dass der Kompilierungsoverhead bei C++ und Java nur einmal anfällt, bei Skriptsprachen allerdings während jeder Ausführung, läuft ein in C++ geschriebener Code im Durchschnitt fünf- bis zehnfach und ein in Java geschriebener Code drei bis fünffach schneller als ein Python-Code⁴⁷.

Begründet ist dies unter anderem auch im Aufwand der Verwaltung von High-Level-Datentypen (Kollektionen), die jeweils aus mehreren Einzelobjekten zusammengesetzt sind, sowie den dynamischen Variablentypen, deren Datentyp-Analyse und Cast⁴⁸ immer wieder während der Laufzeit auftreten. Der Vorteil ist jedoch, dass in einem Pythonprogramm der Datentyp einer Variablen beim Programmieren nicht deklariert werden muss, sondern je nach Notwendigkeit erst im Ablauf des Programms eingefügt werden kann. Statische Typen sind in Python folglich die Ausnahme. Aus genau diesem Grund muss allerdings der Interpreter während der Laufzeit den Datentyp einer Variablen bei allen Operationen jeweils erst feststellen. Diese Analyse und gegebenenfalls automatische Konvertierung von Datentypen resultiert in den beobachteten Laufzeitnachteilen. Ein ausführlicher Test zu den Laufzeitunterschieden von Python und C++ ist auf der Webseite von Mike Connell unter

<http://www.flat222.org/mac/bench/index.html>

⁴⁷vgl. [Connell]

⁴⁸Umwandlung von einem Typ in einen anderen

veröffentlicht.

Einer der signifikanten Vorteile von Python gegenüber C++ ist allerdings die Geschwindigkeit, mit der Python-Applikationen entwickelt werden können. Grund hierfür ist unter anderem der Umfang des Python-Codes, der meist deutlich kürzer ist, als der von C++.

Auch hier ein Beispiel:

Code in C++:

```
#include <iostream.h>
int main(int argc, char *argv[]) {
  for(int z=0; z<1000000; z++) {
    std::cout<<z<<std::endl; //Ausgabe
  }
}
```

Code in Python:

```
for z in xrange(1000000):
  print z
```

Dieses Beispiel zeigt deutlich den unterschiedlichen Codeumfang, der für das selbe Ergebnis erzeugt werden muss. Auch diese Tatsache macht Python zu einer High-Level- Programmiersprache.

Python ermöglicht es zudem, C++- oder Java-Skripte einzubinden, so dass sich diese Sprachen gegebenenfalls ergänzen können. Es ist also möglich, die Laufzeitvorteile von C++ zu nutzen, indem zeitkritische Vorgänge, falls nötig, ausserhalb von Python implementiert werden.

2.4.4 Python vs. Perl

Perl und Python möchten wir zum Abschluss lediglich noch im Aspekt der Syntax ansprechen, da ein ausführlicher Vergleich der beiden Sprachen auf den Webseiten von Stefan Schwarzer nachzulesen ist:

http://www.sschwarzer.net/python/perlpy_vortrag.html

Einer der in diesem Zusammenhang häufiger zu lesenden Kommentare über Perl und seine Syntax findet sich auch in [Welsh et al.]:

“Eine der Eigenschaften von Perl (manche würden sagen, eines seiner »Probleme«) ist die Fähigkeit, den Code extrem kurz und knapp - bis zur Unlesbarkeit - zu formulieren:

```
while ($_ = <STDIN>) { print; }
while (<STDIN>) { print; }
for (;<STDIN>;) { print; }
print while $_ = <STDIN>;
print while <STDIN>; “a
```

^a<http://www.oreilly.de/german/freebooks/rlinux3ger/ch135.html>

Alle Anweisungen erfüllen hier den selben Zweck und zwar die Ausgabe von Daten, die vom Standard-Input des Programms gelesen werden. Allerdings ist für das ungeübte Auge wohl nur die letzte Anweisung halbwegs verständlich. Interessanterweise ähnelt aber auch gerade diese am meisten dem, was man

in Python schreiben würde. Resultat ist daher, dass Python im Vergleich zu Perl meist nicht nur einfacher zu erlernen, sondern der Code auch selbst nach Wochen noch leichter zu warten ist, weil es generell leichter fällt, den Sinn einer Anweisung nachzuvollziehen.

Etwas provokant formuliert: Ein in Python geschriebenes Programm ist noch nach Wochen lesbar, während ein Perl-Programm nach einigen Tagen auch vom eigenen Entwickler oft nur noch schwer nachvollzogen werden kann.

Der wesentliche Unterschied zwischen Python und Perl ist also die Einfachheit der Syntax.

3 Anwendungsbeispiele

Obwohl Python aus der Open Source Gemeinde stammt und zumindest in der allgemeinen Wahrnehmung (noch) nicht den Stellenwert hat wie PHP⁴⁹, haben sich Anwendungen auf Python-Basis schon längst auf allen Plattformen etabliert und werden oft auch ohne das Wissen, dass es sich um eine in Python geschriebene Software handelt, eingesetzt. Deshalb wollen wir hier einige Beispiele von Python-Software in verschiedenen Anwendungsbereichen aufzeigen, in der Annahme, dass sich kaum jemand finden wird, der nicht mit zumindest ein oder zwei der genannten Produkte bzw. Anwendungsgebiete in Kontakt gekommen ist.

3.1 Web-Applikationen

3.1.1 Zope

Bei Zope handelt es sich um ein zu großen Teilen in Python geschriebenes Content Management Framework (CMF)⁵⁰ der bereits zuvor erwähnten Zope Corporation, mit integrierter Nutzer- und Zugriffsverwaltung, eigenem Webserver u.v.m.. Zope basiert auf einer eigenen Datenstruktur, der sogenannten ZoDB⁵¹, in der alle Anwendungsdaten gehalten werden. Zope kann daher ohne zusätzliche Datenbank auskommen und ist nur auf das Vorhandensein von Python angewiesen. Zope bietet zudem mehrere Möglichkeiten, Inhalte dynamisch zu erzeugen, unter anderem durch Einbindung externer Datenbanken wie MySQL, PostgreSQL, oder Oracle, wobei die Installation und Konfiguration dieser Konnektoren gelegentlich etwas Suche nach den Quelltexten⁵² und deren Kompilierung erfordert.

Die Darstellung selbst basierte lange Zeit auf der mittlerweile nicht mehr zeitgemäßen und Zope-eigenen DTML-Formatierung. Diese wird aufgrund der

⁴⁹Mittlerweile werben beispielsweise viele Web-Hoster mit PHP als Sprache für dynamische Webseiten...

⁵⁰'Content Management Framework' meint hier ein rudimentäres und erweiterbares System zur dynamischen Erzeugung von HTML-Webseiten.

⁵¹Zope Object Database

⁵²engl.: Sources

fehlenden XHTML-Konformität zunehmend durch die 'Zope Pagetemplates' (ZPT) abgelöst. Dennoch sind beide Formatierungen weiterhin möglich.⁵³

Ein gewisser Nachteil von Zope, der allerdings auch einen wichtigen Sicherheitsaspekt darstellt, ist die Blockierung systemnaher Python-Schnittstellen, z.B. aus dem Modul `sys`, oder Zugriffe ins Dateisystem, d.h. dass Python-Skripte innerhalb von Zope nicht auf Funktionen dieser Module zugreifen können. Wer also auf Hardwarenähe in seinen Web-Applikationen angewiesen ist, ist mit Zope in der Standardausführung nicht adäquat bedient bzw. muss seine eigenen Produkte entwickeln und in den Funktionsumfang von Zope einbinden. Eine entsprechende Schnittstelle steht in Zope zur Verfügung.

Dennoch lassen sich typische Funktionen für dynamische Web-Seiten über die mitgelieferten Produkte jederzeit und ohne Probleme nutzen, beispielsweise der Versand von EMail, die Speicherung von Daten innerhalb der ZoDB, Zugriffe auf Datenbanken und externe Webseiten, etc.

Näheres zu Zope, inklusive Download und Dokumentation bietet die Zope Corporation auf ihren Seiten unter

<http://www.zope.org>.

3.1.2 Plone

Bei Plone handelt es sich um eine Erweiterung des oben beschriebenen Zope-Frameworks zu einem einfach zu nutzenden Content Management System. Während ein Nutzer in einer Basis-Installation von Zope alle dynamischen Elemente selbst programmieren bzw. erstellen muss und sich dabei zwangsläufig sowohl mit der Darstellung, als auch mit der Implementierung beschäftigt, übernimmt Plone diese Arbeit zu großen Teilen. Einer der Ansprüche von Plone ist es also, dem Anwender die Notwendigkeit zu nehmen, sich über die Applikationsentwicklung in Zope Gedanken machen zu müssen. Einzig um Inhaltsdarstellungen zu erzeugen, die noch nicht in Plone realisiert sind, muss man sich mit Zope noch auseinandersetzen.⁵⁴

Zu bemerken ist noch, dass die Seiten von <http://plone.org> mit ihrem eigenen Produkt realisiert sind, so dass sie einen direkten Eindruck der Fähigkeiten von Plone vermitteln.

3.1.3 Twisted

Twisted unterscheidet sich von den beiden oben genannten Applikationen grundlegend. Twisted ist weniger ein Content Management Framework als ein Appli-

⁵³Ein Exkurs für den interessierten Leser: DTML kodiert seine (Formatierungs-) Anweisungen direkt in den Tags, während der XHTML-Standard eine Kodierung als Attribut eines Tags vorsieht. Auf diese Art und Weise können Editoren, die ein Attribut nicht implementieren oder darstellen können, dieses einfach übergehen, ohne dass die Formatierung des Dokuments darunter leidet. Im Fall der DTML müsste der Editor die gesamte Anweisung verwerfen, so dass eine Darstellung entweder unformatiert oder nur unvollständig möglich ist. XHTML bietet somit den portableren Quelltext.

⁵⁴vgl. 'What is Plone - A technical overview', auf <http://plone.org/about/plone/>

kationsframework für Netzanwendungen. Es fokussiert also nicht auf der reinen Darstellung dynamischer Web-Inhalte, sondern soll vielmehr eine Sammlung nützlicher Funktionen sein, die es dem Anwender ermöglicht eigene (Web-)dienste zu erstellen. Ein gutes Beispiel für die Mächtigkeit und die Möglichkeiten von Twisted finden sich im Tutorial. Hier wird auf wenigen Seiten beschrieben, wie sich mit Twisted in kurzer Zeit beispielsweise ein eigener Finger-Server schreiben lässt.

Enthalten ist in Twisted aber auch die Funktionalität für Mail-Dienste, Webserver, Authentifikationsdienste und einige Netzprotokolle (unter anderem IRC, ICQ, MSN, etc.). Es ist folglich mit relativ geringem Aufwand möglich, mit Hilfe von Python und Twisted einen eigenen MSN-Client zu entwerfen.

Twisted bringt zudem seinen eigenen Daemon `twistd` mit, so dass eigene Applikationen als Dienst ins System eingebunden werden und als Hintergrund-Anwendung laufen können.

Weitergehende Informationen hierzu unter

<http://twistedmatrix.com>

3.1.4 Lizenzen

Zope wird zur Zeit unter der ZPL⁵⁵ lizenziert, die lt. Angabe der Zope Corporation zur GPL⁵⁶ kompatibel ist. Wesentlicher Unterschied der ZPL zur GPL bestehen in den Verpflichtungen zur Beibehaltung der Copyright-Hinweise der Zope Corp. in den Quelltexten und die Verweigerung des Rechts zur Nutzung der Service - und Trademarks (sm/tm), also der von Zope geschützten Markennamen.

Plone besitzt diese Einschränkung dagegen nicht und ist direkt unter die GPL gestellt. Der Vollständigkeit halber sei darauf hingewiesen, dass Twisted im Gegensatz zu den oben genannten Produkten unter der MIT-Lizenz steht, damit der freiesten Lizenz untersteht und unbeschränkt nutz- und veränderbar ist.

3.2 Anwendungssoftware

Nachdem sich das vorangegangene Kapitel mit einigen Webapplikationen und damit Serverdiensten beschäftigt hat, kommen wir in diesem Abschnitt zu den eher systeminternen Anwendungen von Python-Software und so zum Beispiel der Wing IDE.

3.2.1 Wing IDE

Auch wenn es möglicherweise ein Sakrileg im Themenkreis offener/ freier Software darstellt, wollen wir dennoch Wing IDE erwähnen. Bei der Wing IDE handelt es sich, zum ersten mal und leider, nicht um eine kostenlose Software.

⁵⁵Zope Public License

⁵⁶GNU Public License

Im Gegenteil, sowohl für die Personal- als auch für die Professional-Variante, sowohl auf den Plattformen Windows wie auch Linux, fallen spürbare Lizenzgebühren an. Dennoch ist Wing IDE erwähnenswert, da es sich um eine sehr umfangreiche und professionelle Entwicklungsumgebung (IDE) für Python handelt, die zudem noch in Python selbst geschrieben ist. Ein kurzer, knapp zehnmütiger Blick, der uns ohne Registrierung und Kosten möglich war, hat uns eindrucksvoll gezeigt, was in Python machbar ist. Die Software verfügt über Syntax-Highlighting, Debugging-Funktionalität z.B. mit Breakpoints und ein separates Fenster zur Anzeige der Laufzeitfehler, eine direkte Anbindung an Python und einiges mehr. Ein Screenshot der Entwicklungsumgebung folgt auf der nächsten Seite.

Hier ein kleiner Eindruck von Wing IDE:

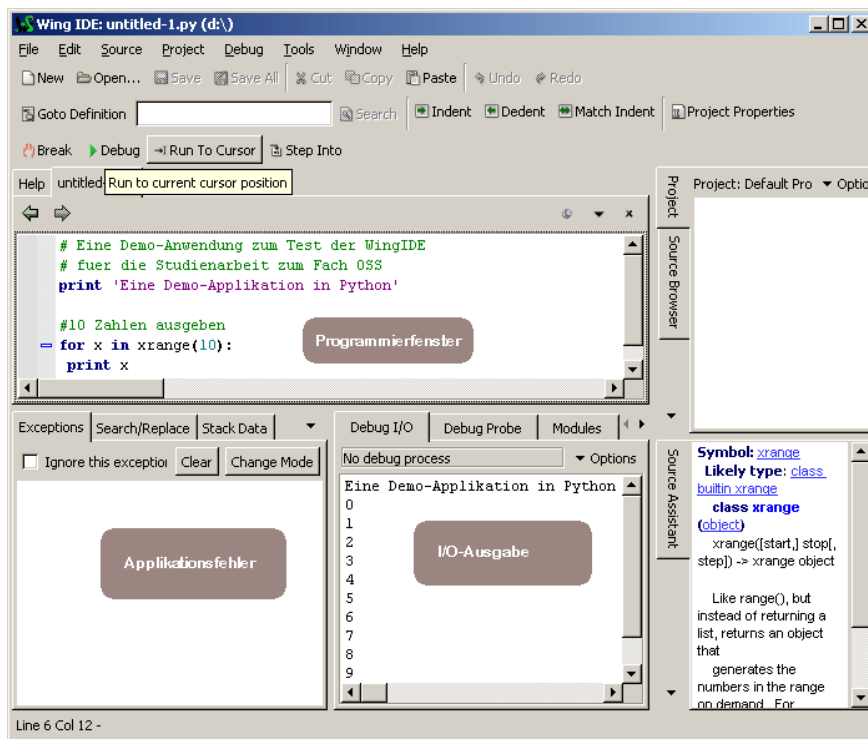


Abbildung 3: Screenshot der Wing IDE 2.0.3

3.2.2 Eclipse

Um nicht den Eindruck zu erwecken, Wing IDE wäre die einzig wahre IDE, möchten wir ausserdem Eclipse⁵⁷ als eine Open Source-Lösung erwähnen, die

⁵⁷siehe: <http://www.eclipse.org>

nur leider eben nicht auf Python basiert. Diese IDE bietet Unterstützung für eine Vielzahl an Sprachen, beispielsweise auch Java und C. Da die Entwicklung der Python-Module aber noch keinen stabilen Stand erreicht hat, soll ein Vergleich der beiden IDEs einem künftigen Artikel vorbehalten bleiben. Für einen Test bzw. Evaluierung der aktuellen Python-Module für Eclipse verweisen wir auf die Wiki-Seiten von Python.org zum Thema Python-Integration:

<http://wiki.python.org/moin/EclipsePythonIntegration>

3.2.3 Bittorrent

Mit Sicherheit weniger bekannt im allgemeinen Gebrauch, aber in der (P2P)⁵⁸-Szene schon seit mehreren Jahren in Verwendung, ist ein Protokoll namens Bittorrent. An sich wäre dies kein Grund für eine Nennung in diesem Zusammenhang, wäre nicht die Referenzimplementierung des Autors Bram Cohen, der sich für die Client-Software der Sprache Python bedient hat. Nachdem lange Zeit Filesharing-Protokolle wie Napster, Edonkey und andere für einen gravierenden Teil des Filesharingaufkommens verantwortlich waren, etabliert sich Bittorrent zunehmend.

Ohne zu tief ins Detail gehen zu wollen, liegt einer der Vorteile dieses Protokolls in der Art, wie die Peer to peer Kommunikation abläuft: Alle Datenpakete werden alleine zwischen den Clients ausgetauscht, ein zentraler sog. Tracker dient im Wesentlichen nur dazu, weitere Mitglieder des Netzes zu finden. Alles in allem hat sich das Protokoll als äusserst effizient bei der Übertragung großer Datenmengen (Dateien mit mehreren hundert Megabyte) in großen Peer-to-peer Verbänden (weit jenseits 1000 gleichzeitiger Nutzer) erwiesen. Häufig steigt in diesem Verbund sogar die Effizienz des Netzes durch eine steigende Anzahl an Mitgliedern⁵⁹. Sicher auch aus diesem Grund wurde das Protokoll bereits in einem bekannten Spiel zur Verbreitung von Updates⁶⁰ implementiert⁶¹.

Die Leistung von Python zeigt sich hier in der Möglichkeit, diese Datenmengen (es kann sich dabei um mehrere Mbit/s handeln) bearbeiten zu können. Da das Protokoll auch Hash-Prüfungen einzelner Datenblöcke, der sogenannten 'Chunks' vorsieht, ist dies ein wesentlicher Aspekt.

Grundlage der Verbreitung eines in Python implementierten Projekts waren unserer Meinung nach auch hier einige der wesentlichen Vorteile von Python:

- die Plattformunabhängigkeit und
- die einfache GUI-Programmierung,

durch die es dem Autor Bram Cohen gelungen ist, seine Referenzimplementierung innerhalb weniger Monate stabil und für jedermann frei und auf mehreren

⁵⁸P2P ist ein gebräuchliches Kürzel für peer-to-peer Filesharing, also Netze zum Datenaustausch unter den angeschlossenen Nutzern.

⁵⁹vgl. [Cohen]

⁶⁰dt.: Aktualisierungen

⁶¹World of Warcraft von Blizzard Entertainment

Plattformen einfach nutzbar anzubieten. Aufgrund der offenen Lizenz⁶² existieren mittlerweile eine ganze Reihe alternativer bzw. auf Bram's Basis aufbauender Implementierungen, mit Namen wie 'Exeem', 'Bittornado' oder 'Azureus'.

Der Bittornado-Client läuft beispielsweise als ein mehrfach und von mehreren Personen modifiziertes Projekt unter Windows, mit Python 2.3.4 und der GUI-API wxWindows 2.5.1.5, unter einer MIT-ähnlichen Lizenz. Dies zeigt plastisch die Informationsseite des Clients:

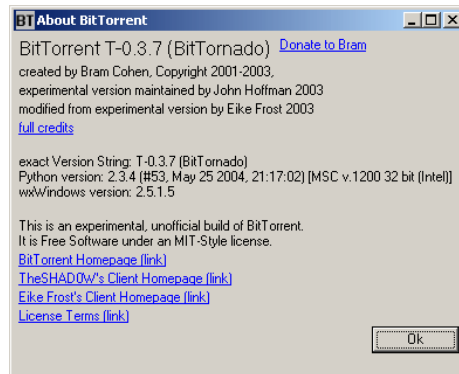


Abbildung 4: Screenshot BitTornado 0.3.7

3.2.4 Weitere Anwendungsgebiete

Neben den oben genannten Anwendungsfeldern findet sich Python auch in einigen weiteren Produkten und Dienstleistungen. Darunter fallen Internet-Suchmaschinen wie google und Infoseek, die Python im Hintergrund für einige ihrer Hilfsprogramme nutzen, aber auch der Handyhersteller Nokia, der ein Python-SDK unter anderem für die Mobiltelefone der 66xx-Serie und das Nokia N-Gage mit dem Betriebssystem Symbian OS anbietet⁶³.

Mit den praktischen Anwendungen endet schließlich der große Bogen von den Anfängen zu den aktuellen Implementierungen mit der Open-Source-Sprache Python. Ganz im Sinne der Open-Source möchten auch wir diese Arbeit unter eine offene Lizenz stellen und damit Änderungen, Verbesserungen und gegebenenfalls die Einarbeitung in eigene Werke ausdrücklich erlauben. Dies ermöglicht uns die folgende Creative Commons Lizenz, mit der wir diese Arbeit daher abschließen.

⁶²Bittorrent Open Source License

⁶³http://www.forum.nokia.com/main/1,6566,1_49,00.html

Teil II

Anhang

Creative Commons License


Dieser Inhalt ist unter einem
Creative Commons Lizenzvertrag lizenziert.
Um die Lizenz einzusehen, gehen Sie bitte auf

<http://creativecommons.org/licenses/by-sa/2.0/de/>

oder schicken Sie einen Brief an

Creative Commons, 5
59 Nathan Abbott Way,
Stanford, California 94305,
USA.

Creative Commons License Deed




creative commons
COMMONS DEED


Namensnennung-Weitergabe unter gleichen Bedingungen 2.0 Deutschland

Sie dürfen:

- den Inhalt vervielfältigen, verbreiten und öffentlich aufführen
- Bearbeitungen anfertigen
- den Inhalt kommerziell nutzen

Zu den folgenden Bedingungen:


 **Namensnennung.** Sie müssen den Namen des Autors/Rechtsinhabers nennen.

 **Weitergabe unter gleichen Bedingungen.** Wenn Sie diesen Inhalt bearbeiten oder in anderer Weise umgestalten, verändern oder als Grundlage für einen anderen Inhalt verwenden, dann dürfen Sie den neu entstandenen Inhalt nur unter Verwendung identischer Lizenzbedingungen weitergeben.

- Im Falle einer Verbreitung müssen Sie anderen die Lizenzbedingungen, unter die dieser Inhalt fällt, mitteilen.
- Jede dieser Bedingungen kann nach schriftlicher Einwilligung des Rechtsinhabers aufgehoben werden.

Die gesetzlichen Schranken des Urheberrechts bleiben hiervon unberührt.

Hier ist eine Zusammenfassung des [Lizenzvertrags](#) in allgemeinverständlicher Sprache.

[Haftungsausschluss](#) 

Literatur

- [Wiki1] http://en.wikipedia.org/wiki/Guido_van_Rossum
- [Python1] <http://www.python.org/psf/license.html>
- [Python2] <http://www.python.org/~guido/>
- [Python3] <http://www.python.org/doc/essays/foreword.html>
- [Python4] <http://www.python.org/2.1/fsf.html>
- [Kubi1] http://www.kubieziel.de/pythonwiki/index.php/Warum_ist_Python_%FCberhaupt_erschaffen_worden%3F
- [ABC] <http://homepages.cwi.nl/~stevan/abc/>
- [Swaroop] 'A Byte of Python', C.H. Swaroop, 2005
- [Pilgrim] 'Dive into Python', Mark Pilgrim, Mai 2004, Apress *Besondere Empfehlung*
- [Hoegl1] 'Vorlesungsskript', Hoegl, <http://www.fh-augsburg.de/~hhoegl/oosw/skript/main.pdf>
- [Connell] 'Python vs. Perl vs. Java vs. C++ Runtimes', Mike Connell, <http://www.flat222.org/mac/bench/index.html>
- [Welsh et al.] 'Linux - Wegweiser zur Installation & Konfiguration', Matt Welsh, Matthias Kalle Dalheimer & Lar Kaufmann, O'Reilly Verlag, 3.Auflage, 2000, online unter <http://www.oreilly.de/german/freebooks/rlinux3ger/ch135.html>
- [Cohen] 'Incentives Build Robustness in BitTorrent', Bram Cohen, 2003
- [Weigend] 'Objektorientierte Programmierung mit Python', Michael Weigend, 1 Auflage 2004