

---

# Python und XML

Python und XML sind zwei sehr verschiedene Dinge, jeweils mit einer eigenen, reichen Geschichte. Python ist eine universale Programmiersprache, die durch die Vision und unter der Leitung ihres Erfinders Guido van Rossum sehr organisch aus ihren Scripting-Wurzeln entstanden ist. Guido berücksichtigt auch weiterhin die Bedürfnisse von Python-Entwicklern bei der weiteren Entwicklung der Sprache. XML dagegen ist – obwohl stark von den Ideen einer kleinen Gruppe von Visionären geprägt – innerhalb von Standardisierungsgremien groß geworden. XML hat dabei stille Phasen gesehen, in denen die Technologie einfach übernommen wurde, aber auch solche, in denen bittere Kämpfe über seine Zukunft geführt wurden. Warum sollte man also die beiden Technologien zusammenlegen?

Vor der Kombination von Python mit XML schien es keinen einfachen oder effektiven Weg zu geben, mit XML in einer verteilten Umgebung zu arbeiten. Entwickler waren gezwungen, sich auf eine Vielzahl von Werkzeugen zu verlassen, die nur in sehr umständlichen Kombinationen miteinander benutzt werden konnten. Wir haben Shell-Skripten und Perl benutzt, um Text zu bearbeiten und mit dem Betriebssystem zu interagieren, und haben dann Java XML-APIs verwendet, um XML zu verarbeiten und Netzwerkprogrammierung zu betreiben. Die Shell war ein hervorragendes Mittel zur Bearbeitung von Dateien und zur Interaktion mit dem Unix-System, und Perl erwies sich als eine gute Wahl für die einfache Textbearbeitung, die Zugriff auf die Unix-APIs bot. Leider fehlte beiden jedoch ein fortschrittliches Objektmodell. Andererseits hatte Java eine objektorientierte Umgebung, eine robuste Plattform-API für die Netzwerkprogrammierung, Threads sowie eine Entwicklungsumgebung für graphische Benutzerschnittstellen (GUIs). Bei Java fanden wir jedoch sofort einen Mangel an Möglichkeiten zur Textbearbeitung; Skriptsprachen boten normalerweise mächtige Möglichkeiten zur Textbearbeitung. Python stellte eine perfekte Lösung dar, da es die Stärken all dieser verschiedenen Möglichkeiten kombiniert.

Wie die meisten Skript-Sprachen bietet auch Python hervorragende Möglichkeiten zur Text- und Dateibearbeitung. Im Unterschied zu den meisten anderen Skript-Sprachen bietet Python jedoch eine mächtige objektorientierte Umgebung mit einer robusten Plattform-API zur Netzwerkprogrammierung, für Threads und zur Entwicklung von graphischen Benutzerschnittstellen. Es kann leicht um Komponenten erweitert werden, die in C oder C++ geschrieben sind, wodurch es mit den meisten existierenden Bibliotheken verbunden werden kann. Um noch eins draufzusetzen: Es wurde nachgewiesen, daß Python portabler ist als andere verbreitete interpretierte Sprachen, da es problemlos auf so unterschiedlichen Plattformen wie massiv parallelen Connection Machines bis hin zu PDAs (Personal Digital Assistants) und eingebetteten Systemen läuft. Als direkte Folge davon eignet sich Python exzellent zur XML-Programmierung und zur Entwicklung verteilter Anwendungen.

Man könnte sagen, daß Python im selben Maß Ordnung und Robustheit in die Skriptwelt bringt, wie Java dies einst in der C++-Welt tat. Wie immer gibt es auch hier Abstriche. Beim Wechsel von C++ nach Java findet man eine einfachere Sprache mit stärkerem objektorientiertem Unterbau. Der Wechsel zu einer einfacheren Sprache bringt weniger Details auf den unteren Ebenen der Speicherverwaltung und Hardware mit sich, wodurch man mehr Robustheit und bessere Möglichkeiten gewinnt, Codierungsfehler zu finden. Ebenso erhält man eine reiche API mit einfacher Thread-Verwaltung, Netzwerkprogrammierung und Unterstützung für Internet-Technologien und -Schnittstellen. Wie Sie es sich denken können, hat diese Flexibilität ihren Preis: eine etwas *verminderte Leistung* verglichen mit Sprachen wie C und C++.

Ebenso gilt: Wenn man eine Skriptsprache wie Python an Stelle von C, C++ oder selbst Java einsetzt, muß man bereit sein, einige Abstriche zu machen. Sie geben einiges an Leistung auf und erhalten im Gegenzug eine höhere Robustheit und die Möglichkeit zur schnelleren Entwicklung. Im Bereich der Entwicklung von Firmen- und Internet-Systemen sind eine verlässliche Software, ein flexibler Entwurf sowie schnelles Wachstum und Auslieferung Faktoren, die solche Leistungsgewinne ausgleichen, die sich durch den Einsatz von Sprachen wie C++ ergeben können. Wenn Sie Teile dieser Leistung zurückbekommen wollen, können Sie immer noch leistungskritische Komponenten Ihrer Anwendung in C oder C++ implementieren, aber Sie können dies so lange vermeiden, bis Sie über Meßdaten verfügen, mit denen Sie wirklich bestimmen können, was tatsächlich ein Problem ist und was nur eines sein könnte. (Wie man diese Analyse vornimmt und Erweiterungen in C/C++ schreibt, ist Gegenstand anderer Bücher.)

Unabhängig von Ihrer Einstellung zu Skriptsprachen, Java oder C++, konzentriert sich dieses Buch auf XML und die Sprache Python. Für jene, die mit XML noch wenig Erfahrung haben, beginnen wir mit einem Überblick, warum es von Interesse ist. Dann werden wir XML von Python aus benutzen und sehen, wie wir damit XML-Anwendungen einfacher erstellen können.



```

date-967302179\pnrnot1\adjustright\rin0\lin0\itap0 {\b0\fs48 Python un
d XML}{
\b0\deleted\fs48\revauthdel1\revdttmdel-2041034726 Grundlagen}{\b0\fs4
8\revised\revauth1\revdttm-2041034726 ?}{\b0\fs48
\par }\pard\plain \qj

```

Das ist schon besser. Der Titel des Kapitels ist sichtbar, so daß wir versuchen können, die Struktur von diesem Punkt an zu dechiffrieren. Die Auszeichnung scheint kompliziert zu sein, und es gibt einen Hinweis auf eine ältere Version des Kapiteltitels. Um den Text zu extrahieren, den wir eigentlich wollen, müssen wir Words Versionskontrollmodell verstehen, was immer noch viele Herausforderungen birgt.

XML dagegen ist anwendungsneutral. Mit anderen Worten: Ein XML-Dokument wird normalerweise von einem XML-Parser oder -Prozessor verarbeitet, kann aber, wenn keiner vorhanden ist, leicht gelesen und geparkt werden. Daten in XML unterliegen nicht den Beschränkungen einer bestimmten Software-Anwendung. Die Möglichkeit, reichhaltige Dateien zu lesen, kann sehr wertvoll werden, etwa wenn Sie in 20 Jahren eine CD-ROM mit alten Geschäftsformularen ausgraben, die Sie plötzlich wieder brauchen. Wird die Software QuickBooks es Ihnen im Jahr 2021 immer noch ermöglichen, diese Daten zu extrahieren? Mit XML können Sie die Daten mit jedem Texteditor lesen.

Sehen wir uns dieses Kapitel in XML an: Da es eine Auszeichnung verwendet, die aus einem Dokumenttyp für Software-Handbücher und -Dokumentation kommt (DocBook), erscheint es ein wenig ausführlich und enthält auch keine Versionsinformationen, aber wir können den Text nun recht einfach identifizieren:

```

<chapter>
  <title>Python und XML</title>
  <para>Python und XML sind zwei sehr verschiedene Dinge, jeweils mit
    einer eigenen, reichen Geschichte. Python ist eine universale
    Programmiersprache, die durch die Vision und unter der Leitung
    ihres Erfinders Guido van Rossum sehr organisch aus ihren

```

Man beachte, daß im Dokument zusätzliche Zeichen (verschieden vom Dokumentinhalt) auftauchen. Diese werden *Auszeichnungen* genannt (oder *Markup*, *Tags*). Das haben wir auch in der RTF-Version des Dokuments gesehen, aber es gab viel mehr Textstücke, die schwer zu verstehen waren, und wir können einigermaßen davon ausgehen, daß die seltsamen Daten im MS Word-Dokument diesen in irgendeiner Form entsprechen. Wäre dies ein Buch über RTF, würden Sie schnell zwei Dinge mutmaßen: RTF ist viel mehr so etwas wie eine Drucker-Protokollsprache als das XML-Beispiel, das wir gerade gesehen haben, und ein Programm zu schreiben, das RTF versteht, wäre recht schwierig. Wir werden Ihnen in diesem Buch zeigen, daß XML benutzt werden kann, um Sprachen zu definieren, die zu Ihrer Anwendung passen, und daß Programme zu schreiben, die XML dechiffrieren können, keine schwierige Angelegenheit ist, besonders nicht mit Hilfe von Python.

## Hierarchische Struktur

XML ist hierarchisch aufgebaut und ermöglicht es Ihnen, eigene Tag-Namen zu wählen. Das ist ziemlich verschieden von HTML. In XML haben Sie alle Freiheiten, Elemente jeden Typs zu erzeugen und andere Elemente darin zu stapeln. Betrachten Sie z.B. einen Adresseneintrag:

```
<?xml version="1.0"?>
<address>
  <name>Bubba McBubba</name>
  <street>123 Happy Go Lucky Ln.</street>
  <city>Seattle</city><state>WA</state><zip>98056</zip>
</address>
```

Im obigen, wohlgeformten XML-Code habe ich einige Datensatznamen erfunden und sie dann mit Daten in einen Topf geworfen. Software, die XML verarbeitet, wie z.B. ein *Parser* (den Sie benutzen können, um die syntaktischen Konstrukte eines XML-Dokuments zu interpretieren), wäre in der Lage, diese Daten auf verschiedene Weise darzustellen, da ihre Struktur erkannt wird. Wenn wir uns z.B. anschauen sollten, was ein Anwendungsprogrammierer als Quellcode schreiben würde, könnten wir diesen Datensatz in ein Objekt verwandeln, das wie folgt initialisiert wird:

```
addr = Address()
addr.name = "Bubba McBubba"
addr.street = "123 Happy Go Lucky Ln."
addr.city = "Seattle"
addr.state = "WA"
addr.zip = "98056"
```

Dadurch wird XML für viele serialisierte Objekte zu einem sehr geeigneten Format. (Es gibt einige Konstrukte, für die XML nicht ganz so geeignet ist, z.B. viele Formate für große numerische Datenmengen, wie sie bei wissenschaftlichen Berechnungen anfallen.) Die hierarchische Struktur von XML macht es einfach, das Konzept von Objektschnittstellen auf Dokumente anzuwenden – es ist relativ einfach, anwendungsspezifische Objekte direkt aus dem Informationsstrom zu bauen, wenn man Abbildungen von Elementnamen auf Objekttypen hat. Wir werden später sehen, daß wir mehr als einfache hierarchische Strukturen mit XML modellieren können.

## Plattformneutralität

Vergessen Sie nicht, daß XML plattformübergreifend ist! Während dies zum größten Teil an seinem textbasierten Format liegt, ist es doch eine Tatsache. Die Verwendung von gewissen Textcodierungen stellt sicher, daß es keine Mißverständnisse bezüglich der Anordnung eines XML-Dokuments gibt. Daher ist es einfach, eine Bestellung in XML von einem Unix-Rechner auf einen drahtlosen PDA weiterzugeben. XML wurde für die Benutzung mit einer existierenden Internet-Infrastruktur basierend auf HTTP, SSL und anderen sich weiterentwickelnden Nachrichtenprotokollen entworfen. Diese Qualitäten

führen dazu, daß sich XML für verteilte Anwendungen anbietet. Es wurde erfolgreich als Grundlage in Message Queuing-Systemen, Instant Messaging-Anwendungen und Remote Procedure Call-Rahmenwerken verwendet. Wir untersuchen diese Anwendungen ausführlicher in Kapitel 9 und Kapitel 10. Es bedeutet auch, daß das zuvor angegebene Beispieldokument mehr als nur anwendungsneutral ist und direkt und ohne Informationsverlust von einer Rechnersorte auf eine andere übertragen werden kann. So kann z.B. ein Kapitel eines technischen Buches von einem Programmierer auf seiner Lieblingsversion von Unix geschrieben und dann an einen Verlag geschickt werden, der eine Buchsatz-Software auf einem Macintosh einsetzt. Die vielen schwierigen Formatumwandlungen können dadurch vermieden werden.

## Unterstützung internationaler Sprachen

Mit der zunehmenden Durchdringung unseres Alltags durch das Internet werden wir der Welt um uns herum mehr gewahr und erkennen sie als einen kulturell reichen und vielschichtigen Raum. Als Techniker arbeiten wir jedoch noch immer daran, unsere Software derart benutzbar zu machen, daß sie mehr als eine Sprache gleichzeitig unterstützt. Unsere Routinen zur Textbearbeitung »8-Bit-sicher« zu machen ist nicht nur nicht länger ausreichend, es ist nicht einmal mehr nah dran.

Standardisierungsgremien weltweit haben Wege gefunden, mit denen Rechner in ihren Nationalsprachen geschriebene Texte austauschen können, manchmal mehrere Wege, von denen jeder unterschiedlich weit akzeptiert ist. Leider enthalten die meisten Anwendungen keine Informationen darüber, in welcher Sprache oder in welchem Austauschformat ihre Daten geschrieben sind, so daß es schwer ist, Informationen über kulturelle und linguistische Grenzen hinweg auszutauschen, die die verschiedenen Standards darstellen. Manchmal ist es auch schwierig, Informationen innerhalb solcher Grenzen auszutauschen, wenn mehrere prominente Standards existieren.

Die Probleme entstehen aus tiefgreifenden kulturellen Unterschieden bezüglich der Handhabung von Text. Zusätzlich zum westeuropäischen Stil (links-nach-rechts, oben-nach-unten), in dem dieses Buch geschrieben ist, gibt es viele andere Schreibsysteme. Rechts-nach-links ist nichts Ungewöhnliches, und »Zeilen« von oben-nach-unten, angeordnet auf einer Seite von rechts-nach-links, werden in China verwendet. Hebräisch verwendet ein Rechts-nach-links-System, aber Zahlen werden mit arabischen Ziffern links-nach-rechts geschrieben. Andere Systeme erlauben textuelle Anmerkungen parallel zum Text. Man stelle sich vor, was passiert, wenn ein Dokument Text aus verschiedenen Schreibsystemen enthält!

Standardisierungsgremien kennen diese Probleme und arbeiten seit Jahren an Lösungen. Die Herausgeber der XML-Spezifikation haben es bei den meisten dieser Probleme klugerweise vermieden, neue Lösungen vorzuschlagen, und haben es vorgezogen, auf die Arbeit von Experten und auf existierende Standards aufzubauen.

Die Internationale Standardisierungsorganisation (ISO) und das Unicode-Konsortium (<http://www.unicode.org/>) haben einen Standard erarbeitet, der zwar nicht perfekt, aber dennoch vielleicht der bestgeeignete Standard ist, indem er versucht, die Textdarstellungen der Welt mit der Absicht zu vereinen, alle Sprachen und Alphabete (inklusive ideographischen und hieroglyphischen Zeichensätzen) darstellbar zu machen. Der Standard ist bekannt als ISO/IEC 10646 oder geläufiger als Unicode. Nicht alle nationalen Standardisierungsgremien haben zugestimmt, daß Unicode der Standard für alle zukünftigen Textaustausch-Anwendungen ist (besonders in Asien), aber der Glaube ist weit verbreitet, daß Unicode das Beste ist, was verfügbar ist, um alle zufriedenzustellen. Der Standard behandelt Probleme mit multidirektionalem Text, Regeln für die Groß- und Kleinschreibung und Codierungsalgorithmen, mit denen verschiedene Eigenschaften von Datenströmen garantiert werden können. Der Standard klammert insbesondere jene Sprachprobleme aus, die nicht eng mit der Problematik einzelner Zeichen verbunden sind. Software, die empfindlich auf natürliche Sprache ist, muß eventuell immer noch eine Menge tun, was über Unicode hinausgeht, um den richtigen Vergleich von Namen in einer speziellen Sprache (oder mehreren Sprachen!) zu garantieren. Einige Sprachen werden umfangreiche zusätzliche Unterstützung für die richtige Textdarstellung benötigen (z.B. Arabisch, das verschiedene Zeichenformen für Buchstaben benötigt, je nach Position in einem Wort und benachbarten Zeichenformen).

Das World Wide Web Consortium (W3C) hat einen simplen und cleveren Schritt unternommen, um es einfacher zu machen, sowohl ältere Austauschstandards wie auch Unicode zu verwenden. Es verfügte, daß alle XML-Dokumente in Unicode sein müssen, und bestimmte, daß sie ihre eigene Codierung derart beschreiben müssen, daß alle XML-Prozessoren in der Lage seien zu bestimmen, in welcher Codierung das Dokument geschrieben ist. Einige spezielle Codierungen müssen von allen Prozessoren erkannt werden, so daß es immer möglich ist, XML zu generieren, das überall gelesen werden kann und alle Zeichen der Welt darstellen kann. Es gibt auch eine Eigenschaft, die es ermöglicht, den Inhalt von XML-Dokumenten mit der Sprache zu markieren, in der er geschrieben ist, aber zur Zeit wird dies nicht so oft benutzt, wie es der Fall sein könnte.

Da XML-Dokumente auch Unicode-Dokumente sind, werden die Sprachen der Welt unterstützt. Die Verwendung von Unicode und Codierungen in XML wird etwas detaillierter in Kapitel 2 erläutert. Unicode-Strings sind seit Version 2.0 Teil von Python, und dessen Standardbibliothek unterstützt eine große Anzahl von Codierungen.

## Die XML-Spezifikationen

In der Presse sieht man oft Hinweise darauf, daß XML eine bestimmte industriespezifische Anwendung »jetzt unterstützt«. Der dann folgende Artikel ist oft verwirrend und bietet einen kleinen Happen Information über ein Industriekonsortium, das eine neue Spezifikation für eine XML-basierte Sprache herausgegeben hat, mit der die Interoperabilität von Daten innerhalb der zum Konsortium gehörenden Industrie unterstützt wird.

Mit unserem technischen Sachverstand bemerken wir normalerweise, daß es nicht in den Bereichen anwendbar ist, mit denen wir uns beschäftigen, oder aber die Spezifikation ist noch zu sehr in der Entwurfsphase, um von Nutzen zu sein. Tatsächlich werden unsere Vorgesetzten wahrscheinlich meistens mit uns darin übereinstimmen, oder sie werden ihre geheimen Gründe haben, anderer Meinung zu sein. Wenn wir jedoch die Firmenleiter ein paar Sprossen hinaufsteigen, finden wir oft einen zunehmend höheren Verwirrungsgrad bezüglich XML vor. Dies wird manchmal von einem Aufruf begleitet, XML zu übernehmen (allzuoft mit einer Liste von besonderen Zusatzspezifikationen, die nicht dafür bestimmt sind, zusammen benutzt zu werden), oder der Reaktion, daß XML noch viel zu unreif sei, um ernsthaft benutzt zu werden.

Also müssen wir darüber nachdenken, womit wir gerade noch arbeiten können, damit dabei folgende Kriterien erfüllt werden:

- Es muß in unserer Anwendung technisch Sinn machen.
- Es sollte hinreichend gut definiert sein, daß eine Implementierung möglich ist.
- Es muß (zumindest) unseren direkten Vorgesetzten erklärt und begründet werden können.
- Das höhere Management darf nicht ausflippen.

Okay, da wir technisch veranlagte Leute sind, müssen wir vielleicht den letzten Punkt ignorieren; er wird definitiv nicht in diesem Buch behandelt. Tatsächlich kann das meiste hiervon nicht wirklich in technischem Material abgedeckt werden. Es gibt viele Spezifikationen in unterschiedlichen Reifephasen, und die meisten sind spezifisch für die eine oder andere Industrie. Wir können jedoch hervorheben, was die grundlegenden Spezifikationen sind, weil es jene sind, die Sie brauchen werden, unabhängig von Ihrer Industrie oder anderen Anforderungen.

## Die XML 1.0-Empfehlung

Die XML-Spezifikation selbst ist ein Dokument, das vom W3C erstellt wurde und auch dort gewartet wird. Beim Schreiben dieser Zeilen lautet die aktuelle Version *Extensible Markup Language (XML) 1.0* (zweite Auflage) und ist auf der W3C-Website unter <http://www.w3.org/TR/REC-xml> verfügbar. (Die zweite Auflage unterscheidet sich von der ersten nur durch einige Textkorrekturen und -klarstellungen; die Spezifikation als solche ist stabil.)

XML selbst ist keine Auszeichnungs- bzw. Markup-Sprache, sondern eine *Metasprache*, mit deren Hilfe spezielle Auszeichnungssprachen definiert werden können. In diesem Punkt hat sie sehr viel von SGML geerbt. Die Spezifikation deckt fünf Aspekte von Auszeichnungssprachen ab:

- die Menge struktureller Formen, die ausgezeichnet werden können
- die spezifische Syntax von Auszeichnungskomponenten

- eine Schemasprache, mit der spezifische Sprachen definiert werden können
- eine Definition von Gültigkeitseinschränkungen
- die minimalen Anforderungen an verarbeitende Werkzeuge

Im Unterschied zu SGML kann man XML als solches benutzen, ohne eine spezielle Auszeichnungssprache in irgendeiner Art formal zu definieren. Ob das in Ihren Anwendungen nützlich ist oder nicht, sei dahingestellt, aber in einigen Entwicklerkreisen hat dies die Akzeptanz von XML-basierten Technologien deutlich beschleunigt. Das kann daran liegen, daß die Eintrittskarte zum XML-Universum recht preiswert ist. Man kann mit XML anfangen, ohne einige der esoterischen Teile der Spezifikation zu kennen, und die Entwicklung von Prototypen mit XML-Technologien kann ohne Unmengen von Vorplanungen beginnen.

Kapitel 2 stellt die meistverwendeten Teile der Spezifikation vor und behandelt jene Teile detaillierter, die für die meisten Leser dieses Buches besonders wichtig sind. Wenn irgendwelche Details von besonderem Interesse für Sie sind, sollten Sie die Zeit investieren und die relevanten Teile der Spezifikation lesen. Auch wenn sie bisweilen ein wenig umständlich erscheint, so ist sie im allgemeinen doch keine besonders schwer zu lesende Spezifikation.

## Namensräume in XML

Obwohl die XML 1.0-Empfehlung zwar spezifische syntaktische Aspekte von XML sowie einen Weg definiert, um Dokumenttypen zu erzeugen, erläutert sie jedoch nicht, wie man Komponenten mehrerer Dokumenttypen kombiniert. Die Empfehlung *Namespaces in XML*, die unter <http://www.w3.org/TR/REC-xml-names> verfügbar ist (von nun an *Namensräume* genannt), behandelt die syntaktische und strukturelle Mechanik der Kombination von strukturellen Komponenten aus verschiedenen Spezifikationen, hält sich jedoch weitgehend bedeckt, was die Bedeutung der resultierenden Kombinationen angeht. Dazu verweist sie auf Spezifikationen, die noch nicht geschrieben waren, als die Empfehlung veröffentlicht wurde.

Diese Empfehlung schränkt zusätzlich die syntaktische Konstruktion konformer Dokumente ein. Sie ermöglicht es einem Dokument, die Quelle eines jeden Elements oder Attributs anzugeben, indem es in einen *Namensraum* plaziert wird. Jeder Namensraum umfaßt Definitionen für Elemente und Attribute. Wie die Elemente und Attribute definiert werden, wird nicht von dieser Spezifikation behandelt, d. h., das Konzept der Validierung eines beliebigen Dokuments, das Namensräume verwendet, ist nicht gänzlich geklärt. Es ist möglich, einen Dokumenttyp mit XML 1.0 zu erzeugen, der eine gewisse Unterstützung für Namensräume bietet, aber ein solches Schema verliert viel von der Flexibilität, die die Namensraum-Spezifikation bietet. So müßte der Dokumenttyp die spezifischen Präfixe angeben, an die jeder Namensraum gebunden ist, während die Namensraum-Spezifikation Präfixe erlaubt, die vom Dokument statt vom Schema

bestimmt werden. Alternative Schemasprachen, die eine bessere Unterstützung von Namensräumen haben, wurden bereits definiert; sie werden kurz in Kapitel 2 diskutiert.

## XML als Grundlage

Wie dessen Vorgänger SGML bietet XML die Möglichkeit, Sprachen zu definieren, die den Anforderungen Ihrer Anwendung entsprechen. Indem die exakte Syntax der grammatikalischen Elemente angegeben wird (z.B. die Zeichen, mit denen bezeichnet wird, wo ein Element beginnt), hat es den Aufwand für die Erstellung konformer Software reduziert – die notwendigen Komponenten für die Gewinnung der Daten einer Anwendung aus XML sind wesentlich kleiner und einfacher zu benutzen als die entsprechenden Komponenten für SGML.

Die zusätzlichen Spezifikationen, die von der Fachpresse so gern diskutiert werden, sobald eine neue Version herauskommt, entstehen im allgemeinen, indem neue Sprachen aus den grundlegenden Empfehlungen für XML und Namensräume definiert werden. Diese werden oft anhand von Schemadefinitionen dokumentiert (die dabei angenommenen Formen werden in Kapitel 2 beschrieben) oder anhand von von Komitees erzeugten Dokumenten, die zu beschreiben versuchen, wie die Sprache benutzt werden soll. Da es in jeder Industrie mindestens ein Konsortium gibt, das sich z.T. mit dem Datenaustausch zwischen verschiedenen Industriekomponenten beschäftigt (man denke an Ärzte, Apotheken und Krankenhäuser im Gesundheitswesen), nehmen viele Standards diese Form an. Viele der XML-Standards leiten sich aus früheren Versuchen ab, ältere industriespezifische SGML-Sprachen zu benutzen, und viele andere sind neu.

Informationen über industriespezifische Sprachen zu finden kann einfach, aber auch schwer sein. Es gibt viele Quellen, die Sie benutzen können, um relevante Spezifikationen zu finden:

*<http://xml.schema.net/>*

Diese Website enthält Informationen über einen weiten Bereich XML-basierter Standards, darunter allgemeine geschäftsorientierte Spezifikationen, industriespezifische Standards, interoperable Sprachen in der akademischen Forschung und allgemeine Internet-bezogene Spezifikationen.

*<http://www.biztalk.com/>*

Auf dieser Website finden Sie Informationen über den von Microsoft gesponserten, unter »BizTalk« bekannten Bereich von Spezifikationen über Business-Interoperabilität.

*<http://www.ebxml.org/>*

Die »e-business XML«-Initiative – oder auch ebXML – entstand aus der EDI-Gemeinde und konkurriert im allgemeinen mit BizTalk.

*<http://www.w3.org/>*

Für allgemeine Internet-bezogene Spezifikationen ist das World Wide Web Consortium der vielleicht beste Platz zum Suchen; die dortigen Arbeitsgruppen setzen sich

aus breiten Kreisen zusammen, und die Ergebnisse ihrer Arbeit werden überall dort schnell aufgenommen, wo sie Anwendung finden.

<http://www.google.com/>

Wenn alles andere scheitert, versuchen Sie hier nach »XML« und verschiedenen industriespezifischen Schlüsselwörtern zu suchen (besonders die Namen von wichtigen Industriekonsortien).

## Die Stärke von Python und XML

Nachdem wir Sie in die XML-Welt eingeführt haben, werden wir sehen, was Python beizusteuern hat. Wir werden uns die Eigenschaften von Python anschauen, die bei XML Anwendung finden, und wir werden ein paar spezifische Beispiele von Python mit XML vorstellen. Als ausgesprochene Hochsprache enthält Python viele mächtige Datenstrukturen bereits im Kern der Sprache und in seinen Bibliotheken. Die neueren Python-Versionen, ab 2.0 und darüber, verfügen über besonders gute Unicode-Unterstützung und eine beeindruckende Vielfalt an Codierungen sowie einen hervorragenden (und schnellen) XML-Parser, der Zeichendaten aus XML als Unicode-Strings zur Verfügung stellt. Pythons Standardbibliothek enthält auch Implementierungen der Industriestandard-schnittstellen DOM und SAX, um mit XML-Daten zu arbeiten, und zusätzliche Unterstützung für alternative Parser und Schnittstellen ist auch verfügbar.

Natürlich könnte man das gleiche auch von anderen modernen Hochsprachen sagen. Java hat ganz gewiß eine beeindruckende Bibliothek von sehr nützlichen Datenstrukturen, und Perl bietet ebenfalls gleichwertige Datenstrukturen. Warum sollte man also Python gegenüber diesen Sprachen und ihren Bibliotheken vorziehen? Es gibt einige Eigenschaften, von denen wir kurz die wichtigsten erläutern:

- Python-Quellcode ist einfach zu lesen und zu warten.
- Der interaktive Interpreter macht es einfach, Codefragmente auszuprobieren.
- Python ist unglaublich portabel, schränkt aber nicht den Zugriff auf plattformspezifische Eigenschaften ein.
- Die objektorientierten Eigenschaften sind mächtig, ohne unverständlich zu sein.

Es gibt viele Sprachen, mit denen man das tun kann, was man auch mit Python tun kann, aber man findet selten alle »Randqualitäten« von Python in einer einzigen Sprache vereint. Diese Qualitäten machen Python nicht unbedingt mächtiger, aber einfacher anzuwenden, was die Anzahl von Programmierstunden verringert. Dadurch hat man mehr Zeit, um bessere Wege zu finden, reale Probleme zu lösen, oder man kann einfach zum nächsten Problem übergehen. Nun erläutern wir diese Eigenschaften im Detail.

### *Einfach zu lesen und zu warten*

Als Programmiersprache zeigt Python eine bemerkenswerte Klarheit im Ausdruck. Obwohl einige Programmierer, die mit anderen Sprachen vertraut sind, erstaunt sind

über die Bedeutung von Leerraum in Python, so scheint doch jeder zu denken, daß er Python-Quellcode wesentlich lesbarer macht als Sprachen, die spezielle Zeichen einführen müssen, um Strukturen im Quellcode anzugeben. Die Strukturen in Python sind nicht einfacher als jene in anderen Sprachen, aber durch die unterschiedliche Syntax sieht Quellcode in Python viel sauberer aus.

Die Verwendung von Leerraum hilft auch, kleinere stilistische Unterschiede zu vermeiden, wie das Setzen von strukturwichtigen geschweiften Klammern. Daher gibt es ein höheres Maß an visueller Konsistenz über Code von verschiedenen Programmierern hinweg. Obwohl dies vielen Programmierern als untergeordneter Punkt erscheinen mag, ist der Effekt der, daß die Wartung von Code, der von anderen Programmierern geschrieben wurde, deswegen viel einfacher wird, weil es einem leichter fällt, sich auf die tatsächliche Struktur und die Algorithmen im Code zu konzentrieren. Für den einzelnen Programmierer ist dies ein netter Nebeneffekt, aber für eine Firma bedeutet dies geringere Kosten bei der Wartung von Code.

#### *Exploratives Programmieren in einem interaktiven Interpreter*

Viele moderne, hochentwickelte Programmiersprachen stellen einen Interpreter zur Verfügung, aber wenige tun dies so erfolgreich wie Python. Andere wie z.B. Java bieten im allgemeinen überhaupt keinen Interpreter. Wenn wir Perl betrachten – eine Sprache, von der man mit Recht behauptet, daß sie von der Kommandozeile aus benutzt sehr mächtig ist –, sehen wir, daß es nicht mit einem reichhaltigen Interpreter ausgestattet ist. Wenn wir den Perl-Interpreter starten, ohne ein Skript anzugeben, wartet er einfach darauf, daß wir ein komplettes Skript auf der Konsole eingeben, und führt es aus, wenn wir damit fertig sind. Er erlaubt uns, ein paar Befehle direkt in der Kommandozeile einzugeben, aber es gibt keine Möglichkeit, jeweils eine Anweisung allein auszuführen und die Ergebnisse Schritt für Schritt zu inspizieren, um zu sehen, ob jedes Codestück genau das tut, was wir erwarten. Der interaktive Python-Interpreter bietet eine reichhaltige Umgebung zum Ausführen individueller Anweisungen und zum Testen der Ergebnisse.

#### *Portabilität ohne Einschränkungen*

Der Python-Interpreter ist einer der portabelsten verfügbaren Sprach-Interpreter. Man weiß, daß er auf den verschiedensten Plattformen läuft – von PDAs und anderen eingebetteten Systemen bis hin zu einigen der mächtigsten jemals gebauten Multiprozessormaschinen. Er läuft auf mehr verschiedenen Betriebssystemen als vermutlich irgendein anderer Interpreter. Darüber hinaus kann sorgsam geschriebener Anwendungscode vieles von dieser Portabilität übernehmen. Python bietet eine Menge Abstraktionen, die gerade soviel tun, um Plattformunterschiede zu verbergen, während es dem Programmierer erlaubt ist, plattformspezifische Dienste zu benutzen, wenn nötig.

Wenn eine Anwendung Zugriff auf Einrichtungen oder Bibliotheken benötigt, die Python nicht bietet, macht Python es einfach, Erweiterungen hinzuzufügen, um solche zusätzlichen Möglichkeiten zu nutzen. Es können Zusatzmodule (normalerweise

in C oder C++, aber andere Sprachen können auch verwendet werden) erstellt werden, die es Python ermöglichen, effizient auf externe Einrichtungen zuzugreifen.

### *Mächtige, aber zugängliche Objektorientierung*

Zu einer gewissen Zeit hörte man oft, wie objektorientiertes Programmieren (OOP) die meisten technischen Probleme lösen würde, mit denen sich Programmierer in ihrem Code auseinandersetzen. Natürlich wußten es die Programmierer besser, nahmen Abstand und verwandelten die Konzepte in nützliche Werkzeuge, die man anwenden konnte, wenn es gerade paßte (obwohl das *Wie* und *Wann* sie angewendet werden sollten immer eine Diskussion wert sein dürfte). Leider sind viele Sprachen mit starker Unterstützung von OOP entweder sehr umständlich im Gebrauch (wie C++ oder in geringerem Maße Java), oder sie sind in weiten Bereichen nicht als Universalsprache akzeptiert worden (z. B. Eiffel).

Python ist anders. Die Sprache unterstützt Objektorientierung ohne einen Großteil des syntaktischen Overheads, den man in vielen weitverbreiteten objektorientierten Sprachen findet, was es sehr einfach macht, neue Objekttypen zu definieren. Im Unterschied zu vielen anderen Sprachen ist Python hochgradig polymorph; Schnittstellen werden wesentlich weniger streng definiert als in Sprachen wie C++ und Java. Das vereinfacht es, nützliche Objekte zu definieren, ohne Code zu schreiben, der nur deswegen existiert, um die Anforderungen einer Schnittstelle zu erfüllen, aber in keiner Anwendung wirklich benutzt wird. Wenn man dies mit den Vorteilen verknüpft, die aus Pythons Standardbibliothek mit einer großen Vielfalt von Schnittstellen entstehen, kann man leicht den Wert erkennen, der in wiederverwendbaren Objekten steckt. Und all dies, während die Leichtigkeit erhalten bleibt, mit der nützliche Schnittstellen implementiert werden.

## **Python-Werkzeuge für XML**

Die Python-Werkzeuge zum Arbeiten mit XML stammen im wesentlichen aus drei Paketen. Dies sind, vom meistbenutzten zum größten:

1. Die Python-Standardbibliothek
2. PyXML, hergestellt von der Python XML Special Interest Group
3. 4Suite von Fourthought, Inc.

Die Python-Standardbibliothek bietet eine minimale, aber dennoch nützliche Menge von Schnittstellen für die Arbeit mit XML, inklusive einer Schnittstelle für den beliebten XML-Parser Expat, einer Implementierung des leichtgewichtigen Simple API for XML (SAX) und einer einfachen Implementierung des Kerns vom Document Object Model (DOM). Die DOM-Implementierung unterstützt den Level 1 und einen Großteil von Level 2 der DOM-Spezifikation des W3C, implementiert jedoch die meisten optionalen Eigenschaften nicht. Das Material in der Standardbibliothek entstand ursprünglich im PyXML-Paket, und weiteres Material wurde von führenden Python-XML-Entwicklern beigesteuert.

PyXML ist ein Paket mit deutlich mehr Funktionalität. Es erweitert die Standardbibliothek um zusätzliche XML-Parser, hat eine wesentlich umfangreichere DOM-Implementierung (inklusive mehr optionaler Eigenschaften), hat Adapter, um weiteren Parsern die Unterstützung der SAX-Schnittstelle zu ermöglichen, kann XPath-Ausdrücke parsen und auswerten, kennt XSLT-Transformationen und eine Menge weiterer Hilfsmodule. Das Paket wird in Gemeinschaftsarbeit von vielen der aktivsten Python/XML-Programmierer gepflegt.

4Suite ist keine Obermenge der anderen Pakete, sondern sollte zusätzlich zu PyXML benutzt werden. Es enthält zusätzliche, auf verschiedene Anwendungen zugeschnittene DOM-Implementierungen, Unterstützung der XLink- und XPointer-Spezifikationen und Werkzeuge zum Arbeiten mit Daten des Resource Description Frameworks (RDF).

Dies sind die Pakete, die durch das gesamte Buch hindurch benutzt werden. Anhang A bietet Ihnen Informationen, wie Sie sie bekommen und installieren. Es gibt jedoch noch weitere Pakete; siehe Anhang F für kurze Beschreibungen einiger davon sowie für weitere, online verfügbare Informationen.

## Die SAX- und DOM-APIs

Die beiden grundlegendsten und meistverwendeten APIs für XML-Daten sind die SAX- und DOM-Schnittstellen. Sie unterscheiden sich grundlegend voneinander, und daher ist es wichtig, unterscheiden zu können, welche davon für Ihre Anwendung die richtige ist.

SAX definiert eine relativ primitive Schnittstelle, die von XML-Parsern einfach zu unterstützen ist, aber dem Anwendungsprogrammierer weitere Detailkenntnisse hinsichtlich der Verarbeitung von Informationen in XML-Dokumenten sowie der Operationen darauf abverlangt. Es bietet jedoch den Vorteil eines geringen Overheads: Es werden keine großen Datenstrukturen aufgebaut, bevor die Anwendung selbst diese wirklich braucht. Dadurch können viele Arten der Verarbeitung wesentlich schneller als mit höherem Overhead durchgeführt werden, und es können wesentlich größere Dokumente effizient verarbeitet werden. Das liegt daran, daß es eine *ereignisorientierte* Schnittstelle ist. Die Verwendung von SAX ähnelt mehr der Verarbeitung von Benutzereingabe-Ereignissen einer graphischen Benutzerschnittstelle als der Bearbeitung einer vorher aufgebauten Datenstruktur. Wie erhalten Sie also »Ereignisse« von einem XML-Parser, und was könnten das für Ereignisse sein?

SAX definiert eine Anzahl von Handler-Schnittstellen, die Ihre Anwendung implementieren kann, um von Ereignissen benachrichtigt zu werden. Die Methoden dieser Objekte werden aufgerufen, wenn die entsprechenden Ereignisse im geparsen XML-Dokument angetroffen werden. Jede Methode kann als das eigentliche Ereignis betrachtet werden, was gut zu objektorientierten Parsing-Ansätzen paßt. Ereignisse werden als Inhalts-, Dokumenttyp-, lexikalische und Fehlerereignisse kategorisiert. Jede Kategorie wird von einer eigenen Schnittstelle behandelt. Die Anwendung kann exakt angeben, an welchen Kategorien von Ereignissen sie interessiert ist, indem sie den Parser mit den passenden

Behandlungsroutinen versorgt und all jene weglässt, die sie nicht braucht. Pythons XML-Unterstützung bietet Basisklassen, die es Ihnen erlauben, nur die Methoden zu implementieren, an denen Sie interessiert sind, während die anderen, die Sie geerbt haben, aber nicht brauchen, einfach nichts tun.

Die meistverwendeten Ereignisse sind inhaltsbezogene Ereignisse, von denen `startElement`, `characters` und `endElement` die wichtigsten sind. In Kapitel 3 betrachten wir SAX gründlicher, aber sehen wir uns vorher schnell an, wie wir mit SAX nützliche Informationen aus einem Dokument extrahieren können. Wir werden dabei ein einfaches Dokument verwenden, damit man leicht sieht, wie man daraus etwas Komplexeres aufbauen kann. So sieht das Dokument aus:

```
<catalog>
  <book isbn="1-56592-724-9">
    <title>The Cathedral & the Bazaar</title>
    <author>Eric S. Raymond</author>
  </book>
  <book isbn="1-56592-051-1">
    <title>Making TeX Work</title>
    <author>Norman Walsh</author>
  </book>
  <!-- Stellen Sie sich hier weitere Einträge vor... -->
</catalog>
```

Wenn wir ein Dictionary erzeugen wollen, das die ISBN-Nummern im `isbn`-Attribut des `book`-Elements auf die Buchtitel abbildet (den Inhalt der `title`-Elemente), würden wir einen Content-Handler erzeugen (wie in Beispiel 1-1 gezeigt), der die drei zuvor genannten Ereignisse untersucht.

*Beispiel 1-1: `bookhandler.py`*

```
import xml.sax.handler

class BookHandler(xml.sax.handler.ContentHandler):
    def __init__(self):
        self.inTitle = 0
        self.mapping = {}

    def startElement(self, name, attributes):
        if name == "book":
            self.buffer = ""
            self.isbn = attributes["isbn"]
        elif name == "title":
            self.inTitle = 1

    def characters(self, data):
        if self.inTitle:
            self.buffer += data

    def endElement(self, name):
        if name == "title":
```

### Beispiel 1-1: *bookhandler.py* (Fortsetzung)

```
self.inTitle = 0
self.mapping[self.isbn] = self.buffer
```

Nun die gesuchte Information zu extrahieren ist trivial. Falls sich der obige Code in *bookhandler.py* befindet und unser Beispieldokument in *books.xml*, könnten wir dies in einer interaktiven Sitzung tun:

```
>>> import xml.sax
>>> import bookhandler
>>> import pprint
>>>
>>> parser = xml.sax.make_parser()
>>> handler = bookhandler.BookHandler()
>>> parser.setContentHandler(handler)
>>> parser.parse("books.xml")
>>> pprint.pprint(handler.mapping)
{u'1-56592-051-1': u'Making TeX Work',
 u'1-56592-724-9': u'The Cathedral & the Bazaar'}
```

Für Referenzmaterial zu den Methoden von Handler-Objekten siehe Anhang C.

DOM ist so ziemlich das genaue Gegenteil von SAX. SAX bietet ein sehr kleines, bewegtes Fenster, durch das man auf das Eingabedokument blicken kann, wobei es sich auf die Anwendung verläßt, um daraus das Ganze abzuleiten. Von DOM erhält die Anwendung das gesamte Dokument, aus dem sie dann die Details selbst herausfinden muß. Anstatt der Anwendung von individuellen Ereignissen zu berichten, sobald der Parser die entsprechende Syntax im Dokument bearbeitet, erzeugt die Anwendung ein Objekt, das das gesamte Dokument als hierarchische Struktur darstellt. Obwohl es keine Anforderung gibt, daß das gesamte Dokument vollständig geparkt und im Hauptspeicher vorliegt, wenn das Objekt der Anwendung übergeben wird, arbeiten die meisten Implementierungen der Einfachheit halber genau so. Einige Implementierungen vermeiden das. Aber es ist durchaus möglich, eine DOM-Implementierung zu schreiben, die das Dokument verzögert parst oder eine Art persistenten Speicher benutzt, um das geparkte Dokument dort statt im Hauptspeicher aufzubewahren.

Das DOM bietet Objekte, die *Knoten* genannt werden, die einer Anwendung gegenüber Teile eines Dokuments darstellen. Es gibt verschiedene Knotentypen, von denen jeder für ein anderes Konstrukt benutzt wird. Es ist wichtig zu verstehen, daß die DOM-Knoten nicht direkt den Ereignissen in SAX entsprechen, obwohl viele ähnlich sind. Am leichtesten sieht man den Unterschied, indem man sich anschaut, wie Elemente und deren Inhalte mit beiden APIs dargestellt werden. In SAX wird ein Element mit Start- und Endereignissen dargestellt, und sein Inhalt wird von allen Ereignissen dargestellt, die dazwischen liegen. Das DOM bietet ein einzelnes Objekt, das das Element darstellt, und Methoden, mit denen die Anwendung an Kindknoten gelangen kann, die den Inhalt des Elements darstellen. Es existieren verschiedene Knotentypen für Elemente, Text und so ziemlich alles andere, was in XML-Dokumenten vorkommt.

Wir werden in Kapitel 4 mehr ins Detail gehen und einige erweiterte Beispiele zu DOM sehen, eine detaillierte Referenz zur DOM-API finden Sie in Anhang D. Um einen ersten Eindruck von DOM zu bekommen, schreiben wir schnell etwas Code, der das gleiche tut, was wir in Beispiel 1-1 mit SAX gemacht haben, aber diesmal verwenden wir die einfache DOM-Implementierung aus der Python-Standardbibliothek, wie in Beispiel 1-2 gezeigt.

*Beispiel 1-2: dombook.py*

```
import pprint

import xml.dom.minidom
from xml.dom.minidom import Node

doc = xml.dom.minidom.parse("books.xml")

mapping = {}

for node in doc.getElementsByTagName("book"):
    isbn = node.getAttribute("isbn")
    L = node.getElementsByTagName("title")
    for node2 in L:
        title = ""
        for node3 in node2.childNodes:
            if node3.nodeType == Node.TEXT_NODE:
                title += node3.data
        mapping[isbn] = title

# Abbildung hat nun den gleichen Wert wie im SAX-Beispiel:
pprint.pprint(mapping)
```

Es sollte klar sein, daß wir es hier mit etwas völlig anderem zu tun haben! Obwohl es im DOM-Beispiel ungefähr gleich viel Code gibt, kann es sehr schwer sein, wiederverwendbare Komponenten zu entwickeln, während etwas Erfahrung mit SAX oft die Richtung für wiederverwendbare Komponenten mit nur wenig Refactoring anzeigt. Es ist möglich, DOM-Code wiederzuverwenden, aber die dazu nötige Einstellung ist eine ganz andere. Zum Ausgleich bietet DOM die Möglichkeit, ein Dokument an beliebigen Stellen bearbeiten zu können, wenn man es vollständig kennt, und der Dokumentinhalt kann auf verschiedene Weise von verschiedenen Teilen der Anwendung extrahiert werden, ohne das Dokument mehr als einmal parsen zu müssen. Bei einigen Anwendungen ist dies ein ausreichender Grund, DOM statt SAX zu gebrauchen.

## Weitere Möglichkeiten, um Informationen zu extrahieren

Mit SAX und DOM haben wir mächtige Werkzeuge, um mit XML zu arbeiten, aber sie verlangen eine Menge Code und Aufmerksamkeit im Detail, will man sie effektiv in großen Anwendungen einsetzen. In beiden Fällen braucht es für die Arbeit mit komplexen Daten eine Menge Aufwand, nur um jene interessanten Teile von XML-Dokumenten zu

extrahieren, die die Daten enthalten. Aber welche Art von Werkzeugen würden wir normalerweise benutzen, wenn wir es mit komplexen Datenmengen zu tun haben? Zwei, die einem einfallen, sind höhere Abstraktionen (wie z.B. APIs, die mehr Arbeit verrichten, und spezialisierte, aufgabenorientierte Sprachen) sowie Techniken zur Vorverarbeitung (Datentransformation von einer Form in eine andere, die für die gegebene Aufgabe besser geeignet ist). Glücklicherweise stehen uns beide zur Verfügung, wenn wir in Python mit XML arbeiten.

Wenn ein XML-Benutzer einen Teil eines Dokuments mit einer Menge von möglicherweise komplexen Kriterien spezifizieren möchte, benutzt er eine Sprache, die es ihm ermöglicht, die Spezifikation kurz und präzise zu schreiben. Diese Sprache heißt *XML Path Language* oder kurz *XPath*. Unterstützung für XPath existiert im 4Suite-Paket und wurde vor kurzem auch dem PyXML-Paket hinzugefügt. Mit XPath kann eine Abfrage formuliert werden, die Knoten aus einem DOM-Baum selektiert, basierend auf den Elementnamen, Attributwerten, dem Textinhalt und den Beziehungen zwischen den Knoten. In Kapitel 5 behandeln wir einige Details von XPath inklusive der Frage, wie man es mit einem DOM-Baum in Python benutzt.

Möglicherweise möchten Sie aber auch gerne ein neues Dokument, das entweder weniger Informationen enthält oder diese auf eine andere Weise anordnet. Dafür brauchen wir eine Möglichkeit, eine Transformation eines Dokuments anzugeben, die ein anderes Dokument erzeugt. Das leisten die XML Stylesheet Language Transformations (XSLT). Ursprünglich als Teil einer neuen Spezifikation für Stylesheets entwickelt, ist XSLT eine XML-basierte Sprache, mit der Transformationen von XML in andere Formate definiert werden. XSLT wird meistens mit XML oder HTML als Ausgabeformat benutzt. Kapitel 6 beschreibt diese Sprache und zeigt, wie man sie in Python benutzt.

## Was fangen wir damit an?

Nachdem wir gesehen haben, wie man XML in Python benutzen kann, müssen wir nun sehen, wie wir unser XML- und Python-Wissen bei echten Anwendungen einsetzen können. Im Zeitalter des Internets bedeutet das weitverteilte Systeme, die über das Internet arbeiten.

Es gibt eine Menge mehr zum Thema Internet, was über die XML- und CGI-Programmierung in den vielen Beispielen dieses Buches hinausgeht. Falls Sie mit diesem Thema noch nicht vertraut sind, geben wir in Kapitel 8 eine Einführung in die Möglichkeiten von Pythons Standardbibliothek, mit denen Internet-Clients und -Server erstellt werden können. Wir zeigen, wie man Daten von entfernten Servern abrufen kann und wie man formularbasierte Anfragen programmatisch abschicken und das Ergebnis lesen kann. Dann lernen wir, wie man spezifische Webserver baut, die auf HTTP-Anfragen antworten, was es uns erlaubt, Server zu erstellen, die genau das tun, was wir von ihnen verlangen.

Mit diesen Fähigkeiten ausgestattet, fahren wir damit fort, uns die entstehende Welt der »Webdienste« anzuschauen. Kapitel 9 beschreibt, was wir unter Webdiensten verstehen, und gibt eine Einführung in die in diesem Bereich entstehenden Spezifikationen. Wir betrachten zwei Pakete, die es uns ermöglichen, Webdienste mit SOAP abzurufen, und zeigen, wie man einen Webdienst in Python realisiert.

In Kapitel 10 kombinieren wir vieles von dem, was wir gelernt haben, in einem erweiterten Beispiel, das demonstriert, wie alles zusammen arbeitet. Indem wir XML als Kommunikationsmedium benutzen, sind wir in der Lage, eine Anwendung zu erstellen, die sich einer ganzen Reihe von Technologien bedient und in unterschiedlichen Umgebungen arbeitet.

