Text Processing in Python

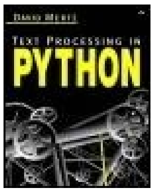
By David Mertz

Start Reading

Publisher : Addison Wesley
Pub Date : June 06, 2003
ISBN : 0-321-11254-7
Pages : 544

Text Processing in Python is an example-driven, hands-on tutorial that carefully teaches programmers how to accomplish numerous text processing tasks using the Python language. Filled with concrete examples, this book provides efficient and effective solutions to specific text processing problems and practical strategies for dealing with all types of text processing challenges.

Text Processing in Python begins with an introduction to text processing and contains a quick Python tutorial to get you up to speed. It then delves into essential text processing subject

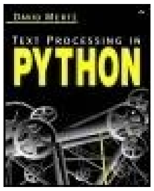


- Table of Contents

areas, including string operations, regular expressions, parsers and state machines, and Internet tools and techniques. Appendixes cover such important topics as data compression and Unicode. A comprehensive index and plentiful cross-referencing offer easy access to available information. In addition, exercises throughout the book provide readers with further opportunity to hone their skills either on their own or in the classroom. A companion Web site (http://gnosis.cx/TPiP) contains source code and examples from the book.

Here is some of what you will find in thie book:

- When do I use formal parsers to process structured and semi-structured data? Page 257
- How do I work with full text indexing? Page 199
- What patterns in text can be expressed using regular expressions? Page 204



Text Processing in Python

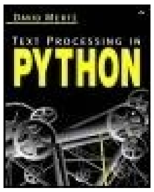
By David Mertz

Start Reading

Publisher : Addison Wesley
Pub Date : June 06, 2003
ISBN : 0-321-11254-7
Pages : 544

- Table of Contents

Top



Text Processing in Python By David Mertz

Copyright

The publisher offers discounts on this book whe
purchases and special sales. For more informat

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales
(317) 581-3793
international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofe

Library of Congress Cataloging-in-Publication D

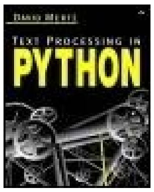
Mertz, David.
Text processing in Python / David Mertz.
p. cm.
Includes bibliographical references and index.
ISBN 0-321-11254-7 (alk. Paper)
1. Text processing (Computer science) 2. Pytl
Title.

QA76.9.T48M47 2003
005.13'-dc21

Boston, MA 02116
Fax: (617) 848-7047

1 2 3 4 5 6 7 8 9 10-CRS-0706050403

First printing, June 2003

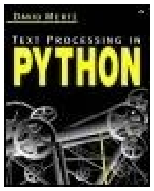


Text Processing in PythonBy David Mertz

Preface

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break t
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptatio
There should be oneand preferably only oneol
Although that way may not be obvious at first
Now is better than never.
Although never is often better than **right** now
If the implementation is hard to explain, it's a
If the implementation is easy to explain, it ma
Namespaces are one honking great idealet's c

Tim Peters, "The Zen of Python"



Preface

0.1 What Is Text Processing?

At the broadest level text processing is simply taking textual information and *doing something* with it. This doing might be restructuring or reformatting it, extracting smaller bits of information from it, algorithmically modifying the content of the information, or performing calculations that depend on the textual information. The lines between "text" and the even more general term "data" are extremely fuzzy; at an approximation, "text" is just data that lives in forms that people can themselves readat least in principle, and maybe with a bit of effort. Most typically computer "text" is composed of sequences of bits that have a "natural" representation as letters, numerals, and symbols; most often such text is delimited (if delimited at all) by symbols and formatting that can be easily pronounced as "next datum."

The lines are fuzzy, but the data that seems least like textand that, therefore, this particular book is least concerned withis the data that makes up "multimedia" (pictures, sounds, video, animation, etc.) and data that makes up UI "events" (draw a window, move the mouse, open an application, etc.). Like I said, the lines are fuzzy, and some representations of the most nontextual data are themselves pretty textual. But in general, the subject of this book is all the stuff on the near side of that fuzzy line.

Text processing is arguably what most programmers spend most of their time doing. The information that lives in business software systems mostly comes down to collections of words about the application domainmaybe with a few special symbols mixed in. Internet communications protocols consist mostly of a few special words used as headers, a little bit of constrained formatting, and message bodies consisting of additional wordish texts. Configuration files, log files, CSV and fixed-length data files, error files, documentation, and source code itself are all just sequences of words with bits of constraint and formatting applied.

Programmers and developers spend so much time with text processing that it is easy to forget that that is what we are doing. The most common text processing application is probably your favorite text editor. Beyond simple entry of new characters, text editors perform such text processing tasks as search/replace and copy/paste, whichgiven guided interaction with the useraccomplish sophisticated manipulation of textual sources. Many text editors go farther than these simple capabilities and include their own complete programming systems (usually called "macro processing"); in those cases where editors include "Turing-complete" macro languages, text editors suffice, in principle, to accomplish anything that the examples in this book can.

After text editors, a variety of text

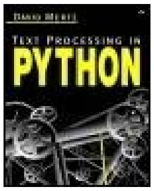
processing tools are widely used by developers. Tools like "File Find" under Windows, or "grep" on Unix (and other platforms), perform the basic chore of *locating* text patterns. "Little languages" like sed and awk perform basic text manipulation (or even nonbasic). A large number of utilitiesespecially in Unix-like environmentsperform small custom text processing tasks: wc, sort, tr, md5sum, uniq, split, strings, and many others.

At the top of the text processing food chain are general-purpose programming languages, such as Python. I wrote this book on Python in large part because Python is such a clear, expressive, and general-purpose language. But for all Python's virtues, text editors and "little" utilities will always have an important place for developers "getting the job done." As simple as Python is, it is still more complicated than you need to achieve many basic tasks. But once you get past the very simple, Python is a perfect language for making the difficult things possible (and it is also good at making the easy things simple).

Team-Fly

Previous Next

Top



0.2 The Philosophy of Text Processing

Hang around any Python discussion groups for a little while, and you will certainly be dazzled by the contributions of the Python developer, Tim Peters (and by a number of other Pythonistas). His "Zen of Python" captures much of the reason that I choose Python as the language in which to solve most programming tasks that are presented to me. But to understand what is most special about *text processing* as a programming task, it is worth turning to Perl creator Larry Wall's cardinal virtues of programming: laziness, impatience, hubris.

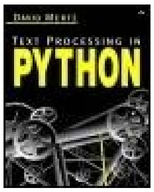
What sets text processing most clearly

apart from other tasks computer programmers accomplish is the frequency with which we perform text processing on an ad hoc or "one-shot" basis. One rarely bothers to create a one-shot GUI interface for a program. You even less frequently perform a one-shot normalization of a relational database. But every programmer with a little experience has had numerous occasions where she has received a trickle of textual information (or maybe a deluge of it) from another department, from a client, from a developer working on a different project, or from data dumped out of a DBMS; the problem in such cases is always to "process" the text so that it is usable for your own project, program, database, or work unit. Text processing to the rescue. This is where the virtue of impatience first appearswe just want the stuff processed, right now!

But text processing tasks that were obviously one-shot tasks that we knew we would never need again have a habit of coming back like restless ghosts. It turns out that that client needs to update the one-time data they sent last month. Or the boss decides that she would really like a feature of that text summarized in a slightly different way. The virtue of laziness is our friend herewith our foresight not to actually delete those one-shot scripts, we have them available for easy reuse and/or modification when the need arises.

Enough is not enough, however. That script you reluctantly used a second time turns out to be quite similar to a more general task you will need to perform frequently, perhaps even automatically. You imagine that with only a slight amount of extra work you can generalize and expand the script, maybe add a little error checking and some runtime options while you are at it; and do it all in time and under budget (or even as a side project, off the budget). Obviously, this is the voice of that greatest of programmers' virtues: hubris.

The goal of this book is to make its readers a little lazier, a smidgeon more impatient, and a whole bunch more hubristic. Python just happens to be the language best suited to the study of virtue.



0.3 What You'll Need to Use This Book

This book is ideally suited for programmers who are a little bit familiar with Python, and whose daily tasks involve a fair amount of text processing chores. Programmers who have some background in other programming languagesespecially with other "scripting" languagesshould be able to pick up enough Python to get going by reading Appendix A.

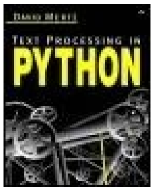
While Python is a rather simple language at heart, this book is not intended as a tutorial on Python for nonprogrammers. Instead, this book is about two other things: getting the job done, pragmatically and efficiently; and understanding why what works works and what doesn't work doesn't work, theoretically and conceptually. As such, we hope this book can be useful both to working programmers and to students of programming at a level just past the introductory.

Many sections of this book are accompanied by problems and exercises, and these in turn often pose questions for users. In most cases, the answers to the listed questions are somewhat open-endedthere are no simple right answers. I believe that working through the provided questions will help both self-directed and instructor-guided learners; the questions can typically be answered at several levels and often have an underlying subtlety. Instructors who wish to use this text are encouraged to contact the author for assistance in structuring a curriculum involving it. All readers are encouraged to consult the book's Web site to see possible answers provided by both the author and other readers; additional related questions will be added to the Web site over time, along with other resources.

The Python language itself is conservative. Almost every Python script written ten years ago for Python 1.0 will run fine in Python 2.3+. However, as versions improve, a certain number of new features have been added. The most significant changes have matched the version number changesPython 2.0 introduced list comprehensions, augmented assignments, Unicode support, and a standard XML package. Many scripts written in the most natural and efficient manner using Python 2.0+ will not run without changes in earlier versions of Python.

The general target of this book will be users of Python 2.1+, but some 2.2+ specific features will be utilized in examples. Maybe half the examples in this book will run fine on Python 1.5.1+ (and slightly fewer on older versions), but examples will not necessarily indicate their requirement for Python 2.0+ (where it exists). On the other hand, new features introduced with Python 2.1 and above will only be utilized where they make a task significantly easier, or where the feature itself is being illustrated. In any case, examples requiring versions past Python 2.0 will usually indicate this explicitly.

In the case of modules and packageswhether in the standard library or third-partywe will explicitly indicate what Python version is required and, where relevant, which version added the module or package to the standard library. In some cases, it will be possible to use later standard library modules with earlier Python versions. In important cases, this possibility will be noted.



0.4 Conventions Used in This Book

Several typographic conventions are used in ma text to guide the readers eye. Both block and ir literals are presented in a fixed font, including names of utilities, URLs, variable names, and co samples. Names of objects in the standard libra however, are presented in italics. Names of modules and packages are printed in a sans se typeface. Heading come in several different fon depending on their level and purpose.

All constants, functions, and classes in discussi and cross-references will be explicitly prepende with their namespace (module). Methods will additionally be prepended with their class. In so cases, code examples will use the local namespace, but a preference for explicit namespace identification will be present in sam code also. For example, a reference might read

SEE ALSO: email.Generator.DecodedGenerator.flatten() *351;* raw_input() *446;* tempfile.mktemp() *71,*

The first is a class method in the *email.Generat* module; the second, a built-in function; the las function in the *tempfile* module.

In the special case of built-in methods on types the expression for an empty type object will be used in the style of a namespace modifier. For example:

Methods of built-in types include *[].sort(),* "".*islower(), {}.keys(),* and *(lambda:1).func_code.*

The file object type will be indicated by the nan FILE in capitals. A reference to a file object met will appear as, for example:

SEE ALSO: FILE.flush() *16;*

Brief inline illustrations of Python concepts and usage will be taken from the Python interactive shell. This approach allows readers to see the immediate evaluation of constructs, much as th might explore Python themselves. Moreover, examples presented in this manner will be self-sufficient (not requiring external data), and ma enteredwith variationsby readers trying to get a grasp on a concept. For example:

```
>>> 13/7 # integer division
1
>>> 13/7. # float division
1.8571428571428572
```

In documentation of module functions, where named arguments are available, they are listed with their default value. Optional arguments ar listed in square brackets. These conventions ar also used in the *Python Library Reference.* For example:

`foobar.spam(s, val=23 [,taste="spicy`

The function *foobar.spam()* uses the argumer to ...

If a named argument does not have a specifiab default value, the argument is listed followed b equal sign and ellipsis. For example:

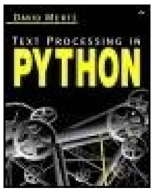
`foobar.baz(string=..., maxlen=... )`

The *foobar.baz()* function ...

With the introduction of Unicode support to Pyt
an equivalence between a character and a byte
longer holds in all cases. Where an operation ta
a numeric argument affecting a string-like obje
the documentation will specify whether charact
or bytes are being counted. For example:

Operation A reads num bytes from the buffer.
Operation B reads num characters from the buffer.

The first operation indicates a number of actual
bit bytes affected. The second operation indicat
an indefinite number of bytes are affected, but they compose a number of (maybe multibyte) characters.



0.5 A Word on Source Code Examples

First things first. All the source code in this book is hereby released to the public domain. You can use it however you like, without restriction. You can include it in free software, or in commercial/proprietary projects. Change it to your heart's content, and in any manner you want. If you feel like giving credit to the author (or sending him large checks) for code you find useful, that is finebut no obligation to do so exists.

All the source code in this book, and various other public domain examples, can be found at the book's Web site. If such an electronic form is more convenient for you, we hope this helps you. In fact, if you are able, you might benefit from visiting this location, where you might find updated versions of examples or other useful utilities not mentioned in the book.

First things out of the way, let us turn to second things. Little of the source code in this book is intended as a final say on how to perform a given task. Many of the examples are easy enough to copy directly into your own program, or to use as standalone utilities. But the real goal in presenting the examples is educational. We really hope you will *think* about what the examples do, and why they do it the way they do. In fact, we hope readers will think of better, faster, and more general ways of performing the same tasks. If the examples work their best, they should be better as inspirations than as instructions.

0.6 External Resources

0.6.1 General Resources

A good clearinghouse for resources and links re to this book is the book's Web site. Over time, add errata and additional examples, questions, answers, utilities, and so on to the site, so chec from time to time:

<http://gnosis.cx/TPiP>

The first place you should probably turn for *any* question on Python programming (after this bo is:

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

The Python newsgroup <comp.lang.python> is amazingly useful resource, with discussion that generally both friendly and erudite. You may als post to and follow the newsgroup via a mirrore mailing list:

<http://mail.python.org/mailman/listinfo/pyth list>

0.6.2 Books

This book generally aims at an intermediate rea Other Python books are better introductory tex (especially for those fairly new to programming generally). Some good introductory texts are:

Core Python Programming, Wesley J. Chun, Prentice Hall, 2001. ISBN: 0-130-26036-3.

Learning Python, Mark Lutz & David Ascher, O'Reilly, 1999. ISBN: 1-56592-464-9.

The Quick Python Book, Daryl Harms & Kenne McDonald, Manning, 2000. ISBN: 1-884777-7 0.

As introductions, I would generally recommend these books in the order listed, but learning sty vary between readers.

Two texts that overlap this book somewhat, but
focus more narrowly on referencing the standar
library, are:

Python Essential Reference, Second Edition,
David M. Beazley, New Riders, 2001. ISBN: 0-
7357-1091-0.

Python Standard Library, Fredrik Lundh, O'Re
2001. ISBN: 0-596-00096-0.

For coverage of XML, at a far more detailed lev
than this book has room for, is the excellent tex

Python & XML, Christopher A. Jones & Fred L.
Drake, Jr., O'Reilly, 2002. ISBN: 0-596-00128

0.6.3 Software Directories

Currently, the best Python-specific directory for
software is the Vaults of Parnassus:

<http://www.vex.net/parnassus/>

SourceForge is a general open source software
resource. Many projectsPython and otherwiseai
hosted at that site, and the site provides search
capabilities, keywords, category browsing, and

like:

<http://sourceforge.net/>

Freshmeat is another widely used directory of software projects (mostly open source). Like th Vaults of Parnassus, Freshmeat does not directl host project files, but simply acts as an informa clearinghouse for finding relevant projects:

<http://freshmeat.net/>

0.6.4 Specific Software

A number of Python projects are discussed in tl book. Most of those are listed in one or more o software directories mentioned above. A genera search engine like Google, <http://google.com also useful in locating project home pages. Belc are a number of project URLs that are current a time of this writing. If any of these fall out of d by the time you read this book, try searching ir search engine or software directory for an upda URL.

The author's *Gnosis Utilities* contains a number Python packages mentioned in this book, incluc *gnosis.indexer, gnosis.xml.indexer,* *gnosis.xml.pickle,* and others. You can downloa most current version from:

<http://gnosis.cx/download/Gnosis_Utils-current.tar.gz>

eGenix.com provides a number of useful Pythoı extensions, some of which are documented in t book. These include *mx.TextTools, mx.DateTim* severeral new datatypes, and other facilities:

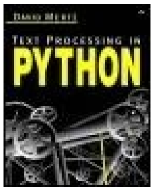
<http://egenix.com/files/python/eGenix-mx-Extensions.html>

SimpleParse is hosted by SourceForge, at:

<http://simpleparse.sourceforge.net/>

The *PLY* parsers has a home page at:

<http://systems.cs.uchicago.edu/ply/ply.html



Text Processing in PythonBy David Mertz

Acknowledgments

Portions of this book are adapted from my column *Charming Python* and other writing first published by *IBM developerWorks*, <http://ibm.com/developerWorks/>. I wish to thank IBM for publishing me, for granting permission to use this material, and most especially for maintaining such a general and useful resource for programmers.

The Python community is a wonderfully friendly place. I made drafts of this book, while in progress, available on the Internet. I received numerous helpful and kind responses, many that helped make the book better than it would otherwise have been.

In particular, the following folks made suggestions and contributions to the book while in draft form. I apologize to any correspondents I may have omitted from the list; your advice was appreciated even if momentarily lost in the bulk of my saved

email.

Sam Penrose <sam@ddmweb.com>

UserDict string substitution hacks.

Roman Suzi <rnd@onego.ru>

More on string substitution hacks.

Samuel S. Chessman <chessman@tux.org>

Helpful observations of various typos.

John W. Krahn <krahnj@acm.org>

Helpful observations of various typos.

Terry J. Reedy <tjreedy@udel.edu>

Found lots of typos and made good organizational suggestions.

Amund Tveit <amund.tveit@idi.ntnu.no>

Pointers to word-based Huffman compression for Appendix B.

Pascal Oberndoerfer

<oberndoerfer@mac.com>

Suggestions about focus of parser discussion.

Bob Weiner <bob@deepware.com>

Suggestions about focus of parser discussion.

Max M <maxm@mxm.dk>

Thought provocation about XML and Unicode entities.

John Machin <sjmachin@lexicon.net>

Nudging to improve sample regular expression functions.

Magnus Lie Hetland <magnus@hetland.org>

Called use of default "static" arguments "spooky code" and failed to appreciate the clarity of the <> operator.

Tim Andrews <Tim.Andrews@adpro.com.au>

Found lots of typos in Chapters 3 and 2.

Marc-Andre Lemburg <mal@lemburg.com>

Wrote *mx.TextTools* in the first place and made helpful comments on my coverage of it.

Mike C. Fletcher <mcfletch@users.sourceforge.net>

Wrote *SimpleParse* in the first place and made helpful comments on my coverage of it.

Lorenzo M. Catucci <lorenzo@sancho.ccd.uniroma2.it>

Suggested glossary entries for CRC and hash.

David LeBlanc <whisper@oz.net>

Various organizational ideas while in draft. Then he wound up acting as one of my technical reviewers and provided a huge amount of helpful advice on both content and organization.

Mike Dussault <dussault@valvesoftware.com>

Found an error in combinatorial HOFs and made good suggestions on Appendix A.

Guillermo Fernandez <guillermo.fernandez@epfl.ch>

Advice on clarifying explanations of compression techniques.

Roland Gerlach <roland@rkga.com.au>

Typos are boundless, but a bit less for his email.

Antonio Cuni <cuni@programmazione.it>

Found error in original Schwartzian sort example and another in map()/zip() discussion.

Michele Simionato <mis6+@pitt.edu>

Acted as a nice sounding board for deciding on final organization of the appendices.

Jesper Hertel <jh@magnus.dk>

Was frustrated that I refused to take his well-reasoned advice for code conventions.

Andrew MacIntyre <andymac@bullseye.apana.org.au>

Did not comment on this book, but has maintained the OS/2 port of Python for several versions. This made my life easier by letting me test and write examples on my favorite machine.

Tim Churches <tchur@optushome.com.au>

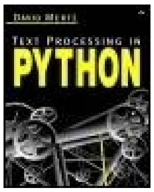
A great deal of subversive entertainment, despite not actually fixing anything in this book.

Moshe Zadka <moshez@twistedmatrix.com>

Served as technical reviewer of this book in manuscript and brought both erudition and an eye for detail to the job.

Sergey Konozenko
<sergey_konozenko@ieee.org>

Boosted my confidence in final preparation with the enthusiasm he brought to his technical reviewand even more so with the acuity with which he "got" my attempts to impose mental challenge on my readers.



Text Processing in PythonBy David Mertz

Table of Contents

Chapter 1. Python Basics

This chapter discusses Python capabilities that are likely to be used in text processing applications. For an introduction to Python syntax and semantics per se, readers might want to skip ahead to Appendix A (A Selective and Impressionistic Short Review of Python); Guido van Rossum's *Python Tutorial* at <http://python.org/doc/current/tut/tut.html> is also quite excellent. The focus here occupies a somewhat higher level: not the Python language narrowly, but also not yet specific to text processing.

In Section 1.1, I look at some programming techniques that flow out of the Python language itself, but that are usually not obvious to Python beginnersand are sometimes not obvious even to intermediate Python programmers. The programming techniques that are discussed are ones that tend to be applicable to text processing contextsother programming tasks are likely to have their own tricks and idioms that are not explicitly documented in this book.

In Section 1.2, I document modules in the Python standard library that you will probably use in your text processing application, or at the very least want to keep in the back of your mind. A number of other Python standard library modules are far enough afield of text processing that you are unlikely to use them in this type of application. Such remaining modules are documented very briefly with one- or two-line descriptions. More details on each module can be found with Python's standard documentation.

Chapter 1. Python Basics

1.1 Techniques and Patterns

1.1.1 Utilizing Higher-Order Functions in Text

This first topic merits a warning. It jumps feet-
fairly sophisticated level and may be unfamiliar
Do not be too frightened by this first topicyou c
it. If the functional programming (FP) concepts
recommend you jump ahead to Appendix A, es

In text processing, one frequently acts upon a s
homogeneous. Most often, these chunks are lin
sometimes other sorts of fields and blocks are r
functions and syntax for reading in lines from a
Obviously, these chunks are not entirely homog
the level we worry about during processing, ea
instruction or information.

As an example, consider an imperative style co
text that match a criterion isCond():

```
selected = []                   # temp
fp = open(filename):
for line in fp.readlines():     # Py2.
    if isCond(line):            # (2.2
        selected.append(line)
del line                        # Clea
```

There is nothing *wrong* with these few lines (se
take a few seconds to read through them. In m
not parse as a *single thought*, even though its
is slightly superfluous (and it retains a value as
conceivably step on a previously defined value)

```
selected = filter(isCond, open(filer
# Py2.2 -> filter(isCond, open(filer
```

In the concrete, a textual source that one frequ
log file. All sorts of applications produce log file
system changes that might need to be examine
actions intermittently. For example, the Python
produces a file called INSTALL.LOG that contair
Below is a highly abridged copy of this file from

INSTALL.LOG sample data file

```
Title: Python 2.2
Source: C:\DOWNLOAD\PYTHON-2.2.EXE |
Made Dir: D:\Python22
File Copy: D:\Python22\UNWISE.EXE |
RegDB Key: Software\Microsoft\Window
RegDB Val: Python 2.2
File Copy: D:\Python22\w9xpopen.exe
Made Dir: D:\PYTHON22\DLLs
File Overwrite: C:\WINDOWS\SYSTEM\MS
RegDB Root: 2
RegDB Key: Software\Microsoft\Window
RegDB Val: D:\PYTHON22\Python.exe
Shell Link: C:\WINDOWS\Start Menu\Pr
Link Info: D:\Python22\UNWISE.EXE |
Shell Link: C:\WINDOWS\Start Menu\Pr
Link Info: D:\Python22\python.exe |
```

You can see that each action recorded belongs
application would presumably handle each type
action has different data fields associated with
functions that identify line types, for example:

```
def isFileCopy(line):
```

```
    return line[:10]=='File Copy:' #
def isFileOverwrite(line):
    return line[:15]=='File Overwrit
```

The string method *"".startswith()* is less error p
versions, but these examples are compatible w
functional programming style, you can also wri

```
isRegDBRoot = lambda line: line[:11]
isRegDBKey = lambda line: line[:10]=
isRegDBVal = lambda line: line[:10]=
```

Selecting lines of a certain type is done exactly

```
lines = open(r'd:\python22\install.l
regroot_lines = filter(isRegDBRoot,
```

But if you want to select upon multiple criteria,
cumbersome. For example, suppose you are int
write a new custom function for this filter:

```
def isAnyRegDB(line):
    if   line[:11]=='RegDB Root:': r
    elif line[:10]=='RegDB Key:':  r
    elif line[:10]=='RegDB Val:':  r
    else:                          r
```

```
# For recent Pythons, line.startswit
```

Programming a custom function for each combi
functions. More importantly, each such custom
and has a nonzero chance of introducing a bug.
satisfied, you can either write custom functions
example:

```
shortline = lambda line: len(line) <
short_regvals = filter(shortline, fi
```

In this example, we rely on previously defined
will be in either shortline() or isRegDBVal(), but
isShortRegVal(). Such nested filters, however, a
are involved.

Calls to *map()* are sometimes similarly nested i
the same string. For a fairly trivial example, su
normalize whitespace in lines of text. Creating
they could be nested in *map()* calls:

```
from string import upper, join, spli
def flip(s):
    a = list(s)
    a.reverse()
    return join(a,'')
normalize = lambda s: join(split(s),
```

```
cap_flip_norms = map(upper, map(flip
```

This type of *map()* or *filter()* nest is difficult to
can sometimes be drawn into nesting alternatir
still worse. For example, suppose you want to p
lines that meet several criteria. To avoid this tra
verbose imperative coding style that simply wra
temporary variables for intermediate results.

Within a functional programming style, it is nor
excessive call nesting. The key to doing this is a
higher-order functions. In general, a higher-ord
returns as result a function object. First-order f
and produce a datum as an answer (perhaps a
contrast, the "inputs" and "outputs" of a HOF a
intended to be eventually called somewhere lat

One example of a higher-order function is a *fur*
returns a function, or collection of functions, th
their creation. The "Hello World" of function fac
World," an adder factory exists just to show wh
useful by itself. Pretty much every explanation
as:

```
>>> def adder_factory(n):
...     return lambda m, n=n: m+n
...
```

```
>>> add10 = adder_factory(10)
>>> add10
<function <lambda> at 0x00FB0020>
>>> add10(4)
14
>>> add10(20)
30
>>> add5 = adder_factory(5)
>>> add5(4)
9
```

For text processing tasks, simple function facto
combinatorial HOFs. The idea of a combinatoria
(usually first-order) functions as arguments an
synthesizes the operations of the argument fun
combinatorial higher-order functions that achie
lines:

combinatorial.py

```
from operator import mul, add, truth
apply_each = lambda fns, args=[]: ma
bools = lambda lst: map(truth, lst)
bool_each = lambda fns, args=[]: boo
conjoin = lambda fns, args=[]: reduc
```

```
all = lambda fns: lambda arg, fns=fr
both = lambda f,g: all((f,g))
all3 = lambda f,g,h: all((f,g,h))
and_ = lambda f,g: lambda x, f=f, g=
disjoin = lambda fns, args=[]: reduc
some = lambda fns: lambda arg, fns=f
either = lambda f,g: some((f,g))
anyof3 = lambda f,g,h: some((f,g,h))
compose = lambda f,g: lambda x, f=f,
compose3 = lambda f,g,h: lambda x, f
ident = lambda x: x
```

Even with just over a dozen lines, many of thes
convenience functions that wrap other more ge
use these HOFs to simplify some of the earlier
results, so look above for comparisons:

Some examples using higher-order functions

```
# Don't nest filters, just produce f
short_regvals = filter(both(shortlir

# Don't multiply ad hoc functions, j
regroot_lines = \
```

```
    filter(some([isRegDBRoot, isRegI

# Don't nest transformations, make c
capFlipNorm = compose3(upper, flip,
cap_flip_norms = map(capFlipNorm, li
```

In the example, we bind the composed functior corresponding *map()* line expresses just the *sir* to all the lines. But the binding also illustrates s functions. By condensing the several operations can save the combined operation for reuse else

As a rule of thumb, I recommend not using mo given line of code. If these "list application" fun readability is preserved by saving results to inte functional programming style calls themselves wonderful thing about Python is the degree to v different programming styles. For example:

```
intermed = filter(niceProperty, map(
final = map(otherTransform, intermec
```

Any nesting of successive *filter ()* or *map()* calls functions using the proper combinatorial HOFs. needed is pretty much always quite small. How offset by the lines used for giving names to cor usually about one-half the length of imperative mean correspondingly fewer bugs).

A nice feature of combinatorial functions is that
algebra for functions that have not been called
operator.mul in combinatorial.py is more than a
a collection of simple values, you might express
values as:

```
satisfied = (this or that) and (foo
```

In the case of text processing on chunks of tex
predicative functions applied to a chunk:

```
satisfied = (thisP(s) or thatP(s)) a
```

In an expression like the above one, several pr
string (or other object), and a set of logical rel
expression is itself a logical predicate of the str
wish to evaluate the same predicate more than
function expressing the predicate:

```
satisfiedP = both(either(thisP,thatP
```

Using a predicative function created with combi
other function:

```
selected = filter(satisfiedP, lines)
```

1.1.2 Exercise: More on combinatorial functic

The module combinatorial.py presented above
combinatorial higher-order functions. But there
example. Creating a personal or organization lil
reusability of your current text processing librai

QUESTIONS

1: Some of the functions defined in combinator
combinatorial. In a precise sense, a combina
functions as arguments and return one or m
arguments. Identify which functions are not
exactly what type of thing each one *does* ret

The functions both() and and_() do almost t
important, albeit subtle, way. and_(), like th
its evaluation. Consider these lines:

```
>>> f = lambda n: n**2 > 10
>>> g = lambda n: 100/n > 10
>>> and_(f,g)(5)
1
>>> both(f,g)(5)
```

```
1
>>> and_(f,g)(0)
0
```

2:

```
>>> both(f,g)(0)
Traceback (most recent call last):
...
```

The shortcutting and_() can potentially allow
second one. The second function never gets
value on a given argument.

a. Create a similarly shortcutting combi

- Create general shortcutting functions short
similarly to the functions all() and some(), re

- Describe some situations where nonshortcu
all(), or anyof3() are more desirable than sir

The function ident() would appear to be poir
is passed to it. In truth, ident() is an almost
3: collection. Explain the significance of ident()

Hint: Suppose you have a list of lines of text
strings. What filter can you apply to find all t

4: The function not_() might make a nice addit
define this function as:

```
>>> not_ = lambda f: lambda x, f=f
```

Explore some situations where a not_() func

5: The function apply_each() is used in combin
the utility of apply_each() is more general th
trivial usage of apply_each() might look som

```
>>> apply_each(map(adder_factory,
[10, 11, 12, 13, 14]
```

Explore some situations where apply_each()
chunk of text.

6: Unlike the functions all() and some(), the fu
fixed number of input functions as argument
that takes a list of input functions, of any ler

7: What other combinatorial higher-order funct
likely to prove useful in text processing? Cor
functions into useful operations, and add the
these enhanced HOFs?

1.1.3 Specializing Python Datatypes

Python comes with an excellent collection of sta
built-in type. At the same time, an important p
less important than programmers coming from
Python's "principle of pervasive polymorphism"
an object *does* than what it *is.* Another commo
like a duck and quacks like a duck, treat it like

Broadly, the idea behind polymorphism is lettin
things of different types. In C++ or Java, for ex
method overloading to let an operation apply to
needed). For example:

C++ signature-based polymorphism

```
#include <stdio.h>
class Print {
public:
  void print(int i)    { printf("int
  void print(double d) { printf("dou
  void print(float f)  { printf("flo
};
main() {
```

```
  Print *p = new Print();
  p->print(37);      /* --> "int 37"
  p->print(37.0);    /* --> "double
}
```

The most direct Python translation of signature
performs type checks on its argument(s). It is s

Python "signature-based" polymorphism

```
def Print(x):
    from types import *
    if type(x) is FloatType:  print
    elif type(x) is IntType:  print
    elif type(x) is LongType: print
```

Writing signature-based functions, however, is
performing these sorts of explicit type checks,
problem you want to solve correctly! What you
type x is, but rather whether x can perform the
what type of thing it is strictly).

PYTHONIC POLYMORPHISM

Probably the single most common case where p
identifying "file-like" objects. There are many o
such as those created with *urllib, cStringIO, zip*
can perform only subsets of what actual files ca
others can seek, and so on. But for many purpo
"file-like" capabilityit is good enough to make s
capabilities you actually need.

Here is a typical example. I have a module that
would like users to be able to specify an XML so
of an XML file, passing a file-like object that co
DOM object to work with (built with any of seve
my module may get their XML from novel place
over sockets, etc.). By looking at what a candid
whichever capabilities that object *has:*

Python capability-based polymorphism

```
def toDOM(xml_src=None):
    from xml.dom import minidom
    if hasattr(xml_src, 'documentEle
        return xml_src    # it is al
    elif hasattr(xml_src,'read'):
        # it is something that knows
        return minidom.parseString(x
    elif type(xml_src) in (StringTyp
```

```
        # it is a filename of an XMI
        xml = open(xml_src).read()
        return minidom.parseString(x
    else:
        raise ValueError, "Must be i
              "filename, file-like c
```

Even simple-seeming numeric types have varyi
should not usually care about the internal repre
what it can do. Of course, as one way to assure
appropriate to coerce it to a type using the buil
list(), long(), str(), tuple(), and *unicode()*. All o
transform anything that looks a little bit like the
instance of it. It is usually not necessary, howe
prescribed types; again we can just check capa

For example, suppose that you want to remove
numberperhaps because they represent measu
numbersints or longsyou might mask out some
might round to a given precision. Rather than t
numeric capabilities. One common way to test
something, and catch any exceptions that occu
simple example:

Checking what numbers can do

```
def approx(x):                    # int
    if hasattr(x,'__and__'):  # supp
        return x & ~0x0FL
    try:                          # supp
        return (round(x.real,2)+rour
    except AttributeError:
        return round(x,2)
```

ENHANCED OBJECTS

The reason that the principle of pervasive polyr
it easy to create new objects that behave most
objects were already mentioned as examples; y
datatype precisely. But even basic datatypes lik
can be easily specialized and/or emulated.

There are two details to pay attention to when
important matter to understand is that the capa
syntactic constructsare generally implemented
leading and trailing double underscores. Any ob
act like a basic datatype in those contexts that
datatype is just an object with some well-optim
methods.

The second detail concerns exactly how you get
make use of existing implementations. There is

version of any basic datatype, except for the pi
quite a few such details, and the easiest way tc
specialize an existing class. Under all non-ancie
provides the pure-Python modules *UserDict, Us*
custom datatypes. You can inherit from an appr
methods as needed. No sample parents are prc
however.

Under Python 2.2 and above, a better option is
inherit from the underlying C implementations
these parent classes have become the self-sam
types and construct objects: *int(), list(), unicoc*
subtle profundities that accompany new-style c
worry about these. All you need to know is that
than one that inherits from *UserString;* likewise
UserDict (assuming your scripts all run on a rec

Custom datatypes, however, need not specializ
to create classes that implement "just enough"
used for a given purpose. Of course, in practice
datatypes is either because you want them to c
because you want them to implement the magi
datatypes. For example, below is a custom data
approx() function, and that also provides a (slig

```
>>> class I:  # "Fuzzy" integer data
...     def __init__(self, i):  self
...     def __and__(self, i):   retu
```

```
...     def err_range(self):
...         lbound = approx(self.i)
...         return "Value: [%d, %d)"
...
>>> i1, i2 = I(29), I(20)
>>> approx(i1), approx(i2)
(16L, 16L)
>>> i2.err_range()
'Value: [16, 31)'
```

Despite supporting an extra method and being
function, I is not a very versatile datatype. If y
"fuzzy integers," you will raise a TypeError. Sin
an older Python version you would need to imp

Using new-style classes in Python 2.2+, you co
underlying int datatype. A partial implementati

```
>>> class I2(int):    # New-style fu
...     def __add__(self, j):
...         vals = map(int, [approx(
...         k = int.__add__(*vals)
...         return I2(int.__add__(k,
...     def err_range(self):
...         lbound = approx(self)
...         return "Value: [%d, %d)"
```

```
...
>>> i1, i2 = I2(29), I2(20)
>>> print "i1 =", i1.err_range(),":
i1 = Value: [16, 31) : i2 = Value: [
>>> i3 = i1 + i2
>>> print i3, type(i3)
47 <class '__main__.I2'>
```

Since the new-style class int already supports b
again. With new-style classes, you refer to data
attribute that holds the data (e.g., self.i in class
syntactic operators within magic methods that
the .__add__() method of the parent int rather
method.

In practice, you are less likely to want to create
emulate container types. But it is worth unders
integers are a fuzzy concept in Python (the fuzz
than the fuzziness of I2 integers, though). Even
need not operate on objects of IntType or Long
desired protocols.

1.1.4 Base Classes for Datatypes

There are several magic methods that are often
fact, these methods are useful even for classes

sense, every object is a datatype since it can co
supports special syntax such as arithmetic oper
method that you can define is documented in th
datatype each is most relevant to. Moreover, ea
few additional magic methods; those covered e
are particularly important.

In documenting class methods of base classes,
for documenting module functions. The one spe
is the use of self as the first argument to all me
arbitrary, this convention is less special than it
following uses of self are equally legal:

```
>>> import string
>>> self = 'spam'
>>> object.__repr__(self)
'<str object at 0x12c0a0>'
>>> string.upper(self)
'SPAM'
```

However, there is usually little reason to use cla
in and module functions with the same purpose
classes are used only in child classes that overr

```
>>> class UpperObject(object):
...         def __repr__(self):
...             return object.__repr__
```

```
...
>>> uo = UpperObject()
>>> print uo
<__MAIN__.UPPEROBJECT OBJECT AT 0X1C
```

object • Ancestor class for new-style data

Under Python 2.2+, object has become a base
enables a custom class to use a few new capab
usually if you are interested in creating a custo
of object, such as list, float, or dict.

METHODS

object.__eq__(self, other)

Return a Boolean comparison between self and
to the == operator. The parent class object doe
object equality means the same thing as identi
implement this in order to affect comparisons.

object.__ne__(self, other)

Return a Boolean comparison between self and
to the != and <> operators. The parent class o
default object inequality means the same thing
Although it might seem that equality and inequ
methods are not explicitly defined in terms of e
with:

```
>>> class EQ(object):
...     # Abstract parent class for
...     def __eq__(self, o): return
...     def __ne__(self, o): return
...
>>> class Comparable(EQ):
...     # By def'ing inequlty, get e
...     def __ne__(self, other):
...         return someComplexCompar
```

object.__nonzero__(self)

Return a Boolean value for an object. Determin
comparisons or, and, and not, and to if and filte
.__nonzero__() method returns a true value is

object.__len__(self)

len(object)

Return an integer representing the "length" of t
straightforwardhow many objects are in the col
behavior to some other meaningful value.

object.__repr__(self)
repr(object)
object.__str__(self)
str(object)

Return a string representation of the object sel
the *repr()* and *str()* built-in functions, to the pr

Where feasible, it is desirable to have the .__re
sufficient information in it to reconstruct an ide
equality obj==eval(repr(obj)). In many cases,
information in a string, and the repr() of an obj
detailed than, the str() representation of the sa

SEE ALSO: repr *96;* operator *47*;

file • New-style base class for file objects

Under Python 2.2+, it is possible to create a cu
built-in class file. In older Python versions you
the methods that define an object as "file-like."
inheritance from file buys you littleif the data c
native filesystem, you will have to reimplement

Even more than for other object types, what m
Depending on your purpose you may be happy
that can only write. You may need to seek with
linear stream. In general, however, file-like obj
Custom classes only need implement those me
should only be used in contexts where their ca

In documenting the methods of file-like objects
for other built-in types. Since actually inheritin
name FILE to indicate a general file-like object.
examples (and implement all the methods nam
equally good FILE instances.

BUILT-IN FUNCTIONS

open(fname [,mode [,buffering]])
file(fname [,mode [,buffering]])

Return a file object that attaches to the filenam
describes the capabilities and access style of th

writing (truncating any existing content); a for
these modes may also have the binary flag b fo
text and binary files. The flag + may be used to
argument buffering may be 0 for none, 1 for lir
bytes.

```
>>> open('tmp','w').write('spam and
>>> print open('tmp','r').read(),
spam and eggs
>>> open('tmp','w').write('this and
>>> print open('tmp','r').read(),
this and that
>>> open('tmp','a').write('something
>>> print open('tmp','r').read(),
this and that
something else
```

METHODS AND ATTRIBUTES

FILE.close()

Close a file object. Reading and writing are disa

FILE.closed

Return a Boolean value indicating whether the

FILE.fileno()

Return a file descriptor number for the file. File
should not implement this method.

FILE.flush()

Write any pending data to the underlying file. F
still implement this method as pass.

FILE.isatty()

Return a Boolean value indicating whether the
documentation says that file-like objects that d
implement this method, but implementing it to
approach.

FILE.mode

Attribute containing the mode of the file, norm
to the object's initializer.

FILE.name

The name of the file. For file-like objects withou
the object should be put into this attribute.

FILE.read ([size=sys.maxint])

Return a string containing up to size bytes of co
is encountered or upon another condition that r
file position forward immediately past the read
as the default value.

FILE.readline([size=sys.maxint])

Return a string containing one line from the file
maximum of size bytes are read. The file positi
negative size argument is treated as the defaul

FILE.readlines([size=sys.maxint])

Return a list of lines from the file, each line incl
size is given, limit the read to *approximately* si
moved forward past the read in bytes. A negati

value.

FILE.seek(offset [,whence=0])

Move the file position by offset bytes (positive
where the initial file position is prior to the mov
EOF.

FILE.tell()

Return the current file position.

FILE.truncate([size=0])

Truncate the file contents (it becomes size leng

FILE.write(s)

Write the string s to the file, starting at the cur
forward past the written bytes.

FILE.writelines(lines)

Write the lines in the sequence lines to the file.
file position is moved forward past the written b

FILE.xreadlines()

Memory-efficient iterator over lines in a file. In
generator that returns one line per each yield.

SEE ALSO: xreadlines *72;*

int • New-style base class for integer obje

long • New-style base class for long intege

In Python, there are two standard datatypes fo
IntType have a fixed range that depends on the
and minus 2**31. Objects of type LongType are
operations on integers that exceed the range of
to long objects. However, no operation on a lon
(even if the result is of small magnitude)with th

From a user point of view ints and longs provid
between them is only in underlying implementa

faster to operate on (since they use raw CPU in
methods integers have are shared by floating p
below. For example, consult the discussion of *fl*
corresponding *int.__mul__()* method. The spec
point numbers is their ability to perform bitwise

Under Python 2.2+, you may create a custom c
earlier versions, you would need to manually de
utilize (generally a lot of work, and probably nc

Each binary bit operation has a left-associative
both versions and perform an operation on two
is chosen. However, if you perform an operation
custom right-associative method will be chosen

```
>>> class I(int):
...     def __xor__(self, other):
...         return "X0R"
...     def __rxor__(self, other):
...         return "RX0R"
...
>>> 0xFF ^ 0xFF
0
>>> 0xFF ^ I(0xFF)
'RXOR'
>>> I(0xFF) ^ 0xFF
```

```
'XOR'
>>> I(0xFF) ^ I(0xFF)
'X0R'
```

METHODS

int.__and__(self, other)
int.__rand__(self, other)

Return a bitwise-and between self and other. D
operator.

int.__hex__(self)

Return a hex string representing self. Determin
hex() function.

int.__invert__(self)

Return a bitwise inversion of self. Determines h

int.__lshift__(self, other)

int.__rlshift__(self, other)

Return the result of bit-shifting self to the left b
shifts other by self bits. Determines how a data

int.__oct__(self)

Return an octal string representing self. Determ
oct() function.

int.__or__(self, other)
int.__ror__(self, other)

Return a bitwise-or between self and other. Det
operator.

int.__rshift__(self, other)
int.__rrshift__(self, other)

Return the result of bit-shifting self to the right
shifts other by self bits. Determines how a data

int.__xor__(self, other)
int.__rxor__(self, other)

Return a bitwise-xor between self and other. De
operator.

SEE ALSO: float *19;* int *421;* long *422;* sys.max

float • New-style base class for floating po

Python floating point numbers are mostly imple
library of your platform; that is, to a greater or
standard. A complex number is just a Python o
extra operations on these pairs.

DIGRESSION

Although the details are far outside the scope o
Floating point math is harder than you think! If
IEEE 754 math is, you are not yet aware of all o
Python luminary and erstwhile professor of num
2001 (on <comp.lang.python>):

> Anybody who thinks he knows what he's doin

naive, or Tim Peters (well, it COULD be W. Ka
here).

Fellow Python guru Tim Peters observed:

> I find it's possible to be both (wink). But **noth**
> even Kahan works his butt off to come up witl

Peters illustrated further by way of Donald Knut
Edition, Addison-Wesley, 1997; ISBN: 0201896

> Many serious mathematicians have attemptec
> operations rigorously, but found the task so fc
> with plausibility arguments instead.

The trick about floating point numbers is that a
representing real-life (fractional) quantities, op
rules we learned in middle school: associativity
very ordinary-seeming numbers can be represe
numbers. For example:

```
>>> 1./3
0.33333333333333331
>>> .3
0.29999999999999999
>>> 7 == 7./25 * 25
0
>>> 7 == 7./24 * 24
```

CAPABILITIES

In the hierarchy of Python numeric types, floati
than integers, and complex numbers higher tha
get promoted upwards. However, the magic me
strictly a subset of those associated with intege
floats apply equally to ints and longs (or intege
support a few addition methods.

Under Python 2.2+, you may create a custom c
under earlier versions, you would need to manu
wished to utilize (generally a lot of work, and p

Each binary operation has a left-associative and
both versions and perform an operation on two
is chosen. However, if you perform an operation
the custom right-associative method will be cho
example under *int*.

METHODS

float.__abs__(self)

Return the absolute value of self. Determines h
function *abs()*.

float.__add__(self, other)
float.__radd__(self, other)

Return the sum of self and other. Determines h

float.__cmp__(self, other)

Return a value indicating the order of self and o
to the numeric comparison operators <, >, <=
behavior of the built-in *cmp()* function. Should
and 1 for self>other. If other comparison metho
.__cmp__(): .__ge__(), .__gt__(), .__le__(), a

float.__div__(self, other)
float.__rdiv__(self, other)

Return the ratio of self and other. Determines h
Python 2.3+, this method will instead determin
division operator //.

float.__divmod__(self, other)
float.__rdivmod__(self, other)

Return the pair (div, remainder). Determines h
divmod() function.

float.__floordiv__(self, other)
float.__rfloordiv__(self, other)

Return the number of whole times self goes int
responds to the Python 2.2+ floor division oper

float.__mod__(self, other)
float.__rmod__(self, other)

Return the modulo division of self into other. De
operator.

float.__mul__(self, other)
float.__rmul__(self, other)

Return the product of self and other. Determine

float.__neg__(self)

Return the negative of self. Determines how a (

float.__pow__(self, other)
float.__rpow__(self, other)

Return self raised to the other power. Determin
operator.

float.__sub__(self, other)
float.__rsub__(self, other)

Return the difference between self and other. D
binary - operator.

float.__truediv__(self, other)
float.__rtruediv__(self, other)

Return the ratio of self and other. Determines h
true division operator /.

SEE ALSO: complex *22;* int *18;* float *422;* oper

complex • New-style base class for compl

Complex numbers implement all the above doc
and a few additional ones.

Inequality operations on complex numbers are
even though they were previously. In Python 2.
complex.__gt__(), complex.__le__(), and *com*
return Boolean values indicating the order. The
as complex numbers do not have a "natural" or
with this changethis is one of the few changes
using it, that I feel was a real mistake. The imp
sort a list of various things, some of which migl

```
>>> lst = ["string", 1.0, 1, 1L, ('t
>>> lst.sort()
>>> lst
[1.0, 1, 1L, 'string', ('t', 'u', 'p
>>> lst.append(1j)
>>> lst.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot compare complex nu
```

It is true that there is no obvious correct orderi

number (complex or otherwise), but there is als
tuple, and a number. Nonetheless, it is frequent
order to create a canonical (even if meaningles
this shortcoming of recent Python versions in th
of luck):

```
>>> class C(complex):
...   def __lt__(self, o):
...     if hasattr(o, 'imag'):
...       return (self.real,self.ima
...     else:
...       return self.real < o
...   def __le__(self, o): return se
...   def __gt__(self, o): return no
...   def __ge__(self, o): return se
...
>>> lst = ["str", 1.0, 1, 1L, (1,2,3
>>> lst.sort()
>>> lst
[1.0, 1, 1L, (1+1j), (2-2j), 'str',
```

Of course, if you adopt this strategy, you have
the custom datatype C. And unfortunately, unle
binary operation between a C object and anoth
datatype. The reader can work out the details o

METHODS

complex.conjugate(self)

Return the complex conjugate of self. A quick r
n-mj.

complex.imag

Imaginary component of a complex number.

complex.real

Real component of a complex number.

SEE ALSO: float *19;* complex *422;*

UserDict • Custom wrapper around diction

dict • New-style base class for dictionary

Dictionaries in Python provide a well-optimized
other Python objects (see Glossary entry on "in
datatypes that respond to various dictionary op
operations associated with dictionaries, all invo
with numeric datatypes, there are several regu
as part of the general interface for dictionary-li

If you create a dictionary-like datatype by subc
special methods defined by the parent are prox
object's .data member. If, under Python 2.2+, y
inherits dictionary behaviors. In either case, yo
wish. Below is an example of the two styles for

```
>>> from sys import stderr
>>> from UserDict import UserDict
>>> class LogDictOld(UserDict):
...     def __setitem__(self, key, va
...         stderr.write("Set: "+str(k
...         self.data[key] = val
...
>>> ldo = LogDictOld()
>>> ldo['this'] = 'that'
Set: this->that
>>> class LogDictNew(dict):
...     def __setitem__(self, key, va
...         stderr.write("Set: "+str(k
```

```
...         dict.__setitem__(self, key
...
>>> ldn = LogDictOld()
>>> ldn['this'] = 'that'
Set: this->that
```

METHODS

dict.__cmp__(self, other)
UserDict.UserDict.__cmp__(self, other)

Return a value indicating the order of self and o
to the numeric comparison operators <, >, <=
behavior of the built-in *cmp()* function. Should
and 1 for self>other. If other comparison meth
.__cmp__(): .__ge__(), .__gt__(), .__le__(), a

dict.__contains__(self, x)
UserDict.UserDict.__contains__(self, x)

Return a Boolean value indicating whether self
contained in a dictionary means matching one o
by overriding it (e.g., check whether x is in a va
datatype responds to the in operator.

dict.__delitem__(self, x)
UserDict.UserDict.__delitem__(self, x)

Remove an item from a dictionary-like datatype
removing the pair whose key equals x. Determi
statement, as in: del self [x].

dict.__getitem__(self, x)
UserDict.UserDict.__getitem__(self, x)

By default, return the value associated with the
to indexing with square braces. You may overri
return special values. For example:

```
>>> class BagOfPairs(dict):
...     def __getitem__(self, x):
...         if self.has_key(x):
...             return (x, dict.__ge
...         else:
...             tmp = dict([(v,k) fo
...             return (dict.__getit
...
>>> bop = BagOfPairs({'this':'that',
>>> bop['this']
```

```
('this', 'that')
>>> bop['eggs']
('spam', 'eggs')
>>> bop['bacon'] = 'sausage'
>>> bop
{'this': 'that', 'bacon': 'sausage',
>>> bop ['nowhere']
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 7, in __getit
KeyError: nowhere
```

dict.__len__(self)
UserDict.UserDict.__len__(self)

Return the length of the dictionary. By default t
you could perform a different calculation if you
size of a record set returned from a database q
how a datatype responds to the built-in *len()* fu

dict.__setitem__(self, key, val)
UserDict.UserDict.__setitem__(self, key, val)

Set the dictionary key key to value val. Determ
assignment; that is, self[key]=val. A custom ve
calculation based on val and/or key before addi

dict.clear(self)
UserDict.UserDict.clear(self)

Remove all items from self.

dict.copy(self)
UserDict.UserDict.copy(self)

Return a copy of the dictionary self (i.e., a disti

dict.get(self, key [,default=None])
UserDict.UserDict.get(self, key [,default=Non

Return the value associated with the key key. I
instead of raising a KeyError.

dict.has_key(self, key)
UserDict.UserDict.has_key(self, key)

Return a Boolean value indicating whether self

dict.items(self)
UserDict.UserDict.items(self)
dict.iteritems(self)
UserDict.UserDict.iteritems(self)

Return the items in a dictionary, in an unspecifi
list of (key,val) pairs, while the .iteritems() met
object that successively yields items. The latter
true in-memory structure, but rather some sort
method responds externally similarly to a for lo

```
>>> d = {1:2, 3:4}
>>> for k,v in d.iteritems(): print
...
1 2 : 3 4 :
>>> for k,v in d.items(): print k,v,
...
1 2 : 3 4 :
```

dict.keys(self)
UserDict.UserDict.keys(self)
dict.iterkeys(self)

UserDict.UserDict.iterkeys(self)

Return the keys in a dictionary, in an unspecifie
list of keys, while the .iterkeys() method (in Py

SEE ALSO: dict.items() *26;*

dict.popitem(self)
UserDict.UserDict.popitem(self)

Return a (key,val) pair for the dictionary, or rai
Removes the returned item from the dictionary
in which items are popped is unspecified (and c

dict.setdefault(self, key [,default=None])
UserDict.UserDict.setdefault(self, key [,defau

If key is currently in the dictionary, return the c
the dictionary, set self[key]=default, then retur

SEE ALSO: dict.get() *26;*

dict.update(self, other)
UserDict.UserDict.update(self, other)

Update the dictionary self using the dictionary o
the corresponding value from other is used in s
is added.

dict.values(self)
UserDict.UserDict.values(self)
dict.itervalues(self)
UserDict.UserDict.itervalues(self)

Return the values in a dictionary, in an unspeci
true list of keys, while the .itervalues() method

SEE ALSO: dict.items() *26;*

SEE ALSO: dict *428;* list *28;* operator *47;*

UserList • Custom wrapper around list obj

list • New-style base class for list objects

tuple • New-style base class for tuple obje

A Python list is a (possibly) heterogeneous mut
similar immutable sequence (see Glossary entr
methods of lists and tuples are the same, but a
associated with internal transformation.

If you create a list-like datatype by subclassing
methods defined by the parent are proxies to t
member. If, under Python 2.2+, you subclass fr
inherits list (tuple) behaviors. In either case, yo
wish. The discussion of *dict* and *UserDict* shows
specialization.

The difference between a list-like object and a t
might think. Mutability is only really important
dictionaries only check the mutability of an obje
object's .__hash__() method. If this method fa
considered mutable (and ineligible to serve as a
useful as keys is because every tuple composed
lists (or dictionaries), by contrast, may also hav
matter (since either can be changed).

You can easily give a hash value to a list-like da
wrong way to do so:

```
>>> class L(list):
...     __hash__ = lambda self: hash
...
>>> lst = L([1,2,3])
```

```
>>> dct = {lst:33, 7:8}
>>> print dct
{[1, 2, 3]: 33, 7: 8}
>>> dct[1st]
33
>>> 1st.append(4)
>>> print dct
{[1, 2, 3, 4]: 33, 7: 8}
>>> dct[1st]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: [1, 2, 3, 4]
```

As soon as 1st changes, its hash changes, and
to it. What you need is something that does no

```
>>> class L(list):
...     __hash__ = lambda self: id(s
...
>>> 1st = L([1,2,3])
>>> dct = {lst:33, 7:8}
>>> dct[1st]
33
>>> 1st.append(4)
>>> dct
```

```
{[1, 2, 3, 4]: 33, 7: 8}
>>> dct[lst]
33
```

As with most everything about Python datatype
protocol that you can choose to support or not

Sequence datatypes may choose to support ord
The methods .__cmp__(), .__ge__(), .__gt__(
meanings for sequences that they do for other
details.

METHODS

list.__add__(self, other)
UserList.UserList.__add__(self, other)
tuple.__add__(self, other)
list.__iadd__(self, other)
UserList.UserList.__iadd__(self, other)

Determine how a datatype responds to the + a
("in-place add") are supported in Python 2.0+.
statements lst+=other and lst=lst+other hav
version might be more efficient.

Under standard meaning, addition of the two se
sequence object with all the items in both self a
mutates the left-hand object without creating a
choose to give a special meaning to addition, p
object added in. For example:

```
>>> class XList(list):
...     def __iadd__(self, other):
...         if issubclass(other.__cl
...             return list.__iadd__
...         else:
...             from operator import
...             return map(add, self
...
>>> xl = XList([1,2,3])
>>> xl += [4,5,6]
>>> xl
[1, 2, 3, 4, 5, 6]
>>> xl += 10
>>> xl
[11, 12, 13, 14, 15, 16]
```

list.__contains__(self, x)
UserList.UserList.__contains__(self, x)
tuple.__contains__(self, x)

Return a Boolean value indicating whether self
datatype responds to the in operator.

list.__delitem__(self, x)
UserList.UserList.__delitem__(self, x)

Remove an item from a list-like datatype. Dete
statement, as in del self[x].

list.__delslice__(self, start, end)
UserList.UserList.__delslice__(self, start, end

Remove a range of items from a list-like dataty
the del statement applied to a slice, as in del se

list.__getitem__(self, pos)
UserList.UserList.__getitem__(self, pos)
tuple.__getitem__(self, pos)

Return the value at offset pos in the list. Deter
with square braces. The default behavior on list
nonexistent offsets.

list.__getslice__(self, start, end)
UserList.UserList.__getslice__(self, start, enc
tuple.__getslice__(self, start, end)

Return a subsequence of the sequence self. Det
indexing with a slice parameter, as in self[start

list.__hash__(self)
UserList.UserList.__hash__(self)
tuple.__hash__(self)

Return an integer that distinctly identifies an ob
to the built-in *hash()* functionand probably mor
dictionaries. By default, tuples (and other immu
will raise a TypeError. Dictionaries will handle h
to make hashes unique per object.

```
>>> hash(219750523), hash((1,2))
(219750523, 219750523)
>>> dct = {219750523:1, (1,2):2}
>>> dct[219750523]
1
```

list.__len__(self

UserList.UserList.__len__(self
tuple.__len__(self

Return the length of a sequence. Determines h
function.

list.__mul__(self, num)
UserList.UserList.__mul__(self, num)
tuple.__mul__(self, num)
list.__rmul__(self, num)
UserList.UserList.__rmul__(self, num)
tuple.__rmul__(self, num)
list.__imul__(self, num)
UserList.UserList.__imul__(self, num)

Determine how a datatype responds to the * ar
("in-place add") are supported in Python 2.0+.
statements lst*=other and lst=lst*other have t
might be more efficient.

The right-associative version .__rmul__() deter
associative .__mul__() determines the value of
product of a sequence and a number produces
items in self duplicated num times:

```
>>> [1,2,3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

list.__setitem__(self, pos, val)
UserList.UserList.__setitem__(self, pos, val)

Set the value at offset pos to value value. Dete
assignment; that is, self[pos]=val. A custom ve
calculation based on val and/or key before addi

list.__setslice__(self, start, end, other)
UserList.UserList.__setslice__(self, start, end

Replace the subsequence self[start:end] with th
sequences are not necessarily the same length,
or shorter than self. Determines how a datatyp
self[start:end]=other.

list.append(self, item)
UserList.UserList.append(self, item)

Add the object item to the end of the sequence

list.count(self, item)
UserList.UserList.count(self, item)

Return the integer number of occurrences of ite

list.extend(self, seq)
UserList.UserList.extend (self, seq)

Add each item in seq to the end of the sequenc
len(seq).

list.index(self, item)
UserList.UserList.index(self, item)

Return the offset index of the first occurrence c

list.insert(self, pos, item)
UserList.UserList.insert(self, pos, item)

Add the object item to the sequence self before
by one.

list.pop(self [,pos=-1])
UserList.UserList.pop(self [,pos=-1])

Return the item at offset pos of the sequence s
sequence. By default, remove the last item, wh
and .append() operations.

list.remove(self, item)
UserList.UserList.remove(self, item)

Remove the first occurrence of item in self. Dec

list.reverse(self)
UserList.UserList.reverse(self)

Reverse the list self in place.

list.sort(self [cmpfunc])
UserList.UserList.sort(self [,cmpfunc])

Sort the list self in place. If a comparison funct
using that function.

SEE ALSO: list *427;* tuple *427;* dict *24;* operato

UserString • Custom wrapper around strin

str • New-style base class for string objec

A string in Python is an immutable sequence of "immutable"). There is special syntax for creati escaping, and so onbut in terms of object beha string does a tuple does, too. Both may be slice arithmetic operators + and *.

For the *str* and *UserString* magic methods that of strings, see the corresponding *tuple* docume *str.__getitem__(), str.__getslice__(), str.__has* *str.__rmul__().* Each of these methods is also c also includes a few explicit definitions of magic class: *UserString.__iadd__(), UserString.__imu* you may define your own implementations of th Python 2.2+). In any case, internally, in-place c

Strings have quite a number of nonmagic meth datatype that can be utilized in the same functi specialize some of these common string metho documented in the discussion of the *string* moc

not also defined in the *string* module. However,
provides very reasonable default behaviors for

SEE ALSO: "".capitalize() *132;* "".title() *133;* ""
134; "".expandtabs() *134;* "".find() *135;* "".ind
"".isdigit() *136;* "".islower() *136;* "".isspace() *1*
"".join() *137;* ""ljust() *138;* "".lower() *138;* "".l
"".rindex() *141;* "".rjust() *141;* "".rstrip() *142;*
"".startswith() *144;* "".strip() *144;* "".swapcase
"".encode() *188;*

METHODS

str.__contains__(self, x)
UserString.UserString.__contains__(self, x)

Return a Boolean value indicating whether self
datatype responds to the in operator.

In Python versions through 2.2, the in operator
tends to trip me up. Fortunately, Python 2.3+ h
Python versions, in can only be used to determ
stringthis makes sense if you think of a string a
nonetheless intuitively want something like the

```
>>> s = "The cat in the hat"
```

```
>>> if "the" in s: print "Has defini
...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'in <string>' requires ch
```

It is easy to get the "expected" behavior in a cu
producing the same result whenever x is indee

```
>>> class S(str):
...     def __contains__(self, x):
...         for i in range(len(self)
...             if self.startswith(x
...
>>> s = S("The cat in the hat")
>>> "the" in s
1
>>> "an" in s
0
```

Python 2.3 strings behave the same way as my

SEE ALSO: string *422;* string *129;* operator *47,*

1.1.5 Exercise: Filling out the forms (or decid

DISCUSSION

A particular little task that was quite frequent a
has become absolutely ubiquitous for slightly d
encounters is that one has a certain general fo
but miscellaneous little details differ from insta
common case where one comes across this pat
Web pages rule the roost of templating techniq

It turns out that everyone and her sister has de
Creating a templating system is a very appealin
just a little while after they have gotten a firm
discussed in Chapter 5, but many others are no
systems will be HTML/CGI oriented and will ofte
calculation of fill-in valuesthe inspiration in thes
ColdFusion, Java Server Pages, Active Server P
gets sprinkled around in documents that are pr

At the very simplest, Python provides interpola
similar to the C sprintf() function. So a simple e

```
>>> form_letter="""Dear %s %s,
...
... You owe us $%s for account (#%s)
...
... The Company"""
>>> fname = 'David'
```

```
>>> lname = 'Mertz'
>>> due = 500
>>> acct = '123-T745'
>>> print form_letter % (fname,lname
Dear David Mertz,

You owe us $500 for account (#123-T7

The Company
```

This approach does the basic templating, but it
composing the tuple of insertion values. And m
templatesuch as the addition or subtraction of a

A bit more robust approach is to use Python's c
example:

```
>>> form_letter="""Dear %(fname)s %(
...
... You owe us $%(due)s for account
...
... The Company"""
>>> fields = {'lname':'Mertz', 'fnam
>>> fields['acct'] = '123-T745'
>>> fields['due'] = 500
>>> fields['last_letter'] = '01/02/2
```

```
>>> print form_letter % fields
Dear David Mertz,

You owe us $500 for account (#123-T7

The Company
```

With this approach, the fields need not be listed
Furthermore, if the order of fields is rearranged
used for a different template, the fields dictiona
fields has unused dictionary keys, it doesn't hu

The dictionary interpolation approach is still sub
Two improvements using the *UserDict* module c
incompatible) ways. In Python 2.2+ the built-in
class"; if available everywhere you need it to ru
UserDict.UserDict. One approach is to avoid all

```
>>> form_letter="""%(salutation)s %(
...
... You owe us $%(due)s for account
...
... %(closing)s The Company"""
>>> from UserDict import UserDict
>>> class AutoFillingDict(UserDict):
...     def __init__(self,dict={}):
```

```
...     def __getitem__(self,key):
...         return UserDict.get(self
>>> fields = AutoFillingDict()
>>> fields['salutation'] = 'Dear'
>>> fields
{'salutation': 'Dear'}
>>> fields['fname'] = 'David'
>>> fields['due'] = 500
>>> fields ['closing'] = 'Sincerely,
>>> print form_letter % fields
Dear David ,

You owe us $500 for account (#). Ple

Sincerely, The Company
```

Even though the fields lname and acct are not s
produce a basically sensible letter (instead of c

Another approach is to create a custom dictiona
interpolation." This approach is particularly use
for the final string over the course of the progra

```
>>> form_letter="""%(salutation)s %(
...
... You owe us $%(due)s for account
```

```
...
... %(closing)s The Company"""
>>> from UserDict import UserDict
>>> class ClosureDict(UserDict):
...     def __init__(self,dict={}):
...     def __getitem__(self,key):
...         return UserDict.get(self
>>> name_dict = ClosureDict({'fname'
>>> print form_letter % name_dict
%(salutation)s David Mertz,

You owe us $%(due)s for account (#%(

%(closing)s The Company
```

Interpolating using a ClosureDict simply fills in
knows, then returns a new string that is closer

SEE ALSO: dict *24;* UserDict *24;* UserList *28;* U

QUESTIONS

1: What are some other ways to provide "smar
that the *UserList* or *UserString* modules migh

interpolation?

2: Consider other "magic" methods that you mi *UserDict.UserDict*. How might these addition more powerful?

3: How far do you think you can go in using Py technique? At what point would you decide y regular expression substitutions or a parser?

4: What sorts of error checking might you impl simple list or dictionary interpolation could f trappable errors (they let the application kn create a system with both flexible interpolati completeness of the final result?

1.1.6 Problem: Working with lines from a larg

At its simplest, reading a file in a line-oriented .readlines(), and .xreadlines() methods of a file syntax for this frequent operation by letting the (strictly in forward sequence). To read in an en possibly split it into lines or other chunks using

```
>>> for line in open('chap1.txt'): #
...     # process each line in some
...     pass
...
>>> linelist = open('chap1.txt').rea
>>> print linelist[1849],
  EXERCISE: Working with lines from
>>> txt = open('chap1.txt').read()
>>> from os import linesep
>>> linelist2 = txt.split(linesep)
```

For moderately sized files, reading the entire co
make time and memory issues more important
example, might be multiple megabytes, or even
such files do not strictly exceed the size of avai
consuming. A related technique to those discus
Reading a file backwards by record, line, or par

Obviously, if you *need* to process every line in a
xreadlines does so in a memory-friendly way, a
sequentially. But for applications that only need
to make improvements. The most important m

A CACHED LINE LIST

It is straightforward to read a particular line fro

```
>>> import linecache
>>> print linecache.getline('chap1.t
  PROBLEM: Working with lines from a
```

Notice that *linecache.getline()* uses one-based
indexing in the prior example. While there is no
have an object that combined the efficiency of
lists. Existing code might exist to process lists
that is agnostic about the source of a list of line
and index, it would be useful to be able to slice
do to real lists (including with extended slices,

cachedlinelist.py

```
import linecache, types
class CachedLineList:
    # Note: in Python 2.2+, it is pr
    # __slots__ = ('_fname')
    # ...and inheriting from 'object
    def __init__(self, fname):
        self._fname = fname
    def __getitem__(self, x):
        if type(x) is types.SliceTyp
```

```
            return [linecache.getlir
                    for n in range(x
        else:
            return linecache.getline
    def __getslice__(self, beg, end)
        # pass to __getitem__ which
        return self[beg:end:1]
```

Using these new objects is almost identical to u
open(fname).readlines(), but more efficient (es

```
>>> from cachedlinelist import Cache
>>> cll = CachedLineList('../chap1.t
>>> cll [1849]
'  PROBLEM: Working with lines from
>>> for line in cll[1849:1851]: prir
...
  PROBLEM: Working with lines from a
  ----------------------------------
>>> for line in cll[1853:1857:2]: pr
...
  a matter of using the '.readline()
  simplified syntax for this frequer
```

A RANDOM LINE

Occasionallyespecially for testing purposesyou
oriented file. It is easy to fall into the trap of m
few lines of a file, and maybe for the last few, t
Unfortunately, the first and last few lines of ma
headers or footers are used; sometimes a log f
development rather than usage; and so on. The
might provide more data than you want to wor
processing, complete testing could be time con

On most systems, seeking to a particular positi
bytes up to that position. Even using *linecache*,
the point of a cached line. A fast approach to fi
seek to a random position within a file, then rea
that position, identifying a line within that chun

randline.py

```
#!/usr/bin/python
"""Iterate over random lines in a fi
From command-line use: % randline.py
"""
import sys
from os import stat, linesep
from stat import ST_SIZE
from random import randrange
```

```
MAX_LINE_LEN = 4096

#__ Iterable class
class randline(object):
    __slots__ = ('_fp','_size','_lim
    def __init__(self, fname, limit=
        self._size = stat(fname)[ST_
        self._fp = open(fname,'rb')
        self._limit = limit
    def __iter__(self):
        return self
    def next(self):
        if self._limit <= 0:
            raise StopIteration
        self._limit -= 1
        pos = randrange(self._size)
        priorlen = min(pos, MAX_LINE
        self._fp.seek(pos-priorlen)
        # Add extra linesep at beg/e
        prior = linesep + self._fp.r
        post = self._fp.read(MAX_LIN
        begln = prior.rfind(linesep)
        endln = post.find(linesep)
        return prior[begln:]+post[:e
```

```
#-- Use as command-line tool
if __name__=='__main__':
    fname, numlines = sys.argv[1], i
    for line in randline(fname, numl
        print line
```

The presented *randline* module may be used ei
a command-line tool. In the latter case, you co
another application, as in:

```
% randline.py reallybig.log 1000 | t
```

A couple details should be noted in my impleme
more than once in a line iteration. If you choos
this probably will not happen (but the so-called
collision more likely than you might expect; see
line that contains a random position in the file,'
to be chosen than long lines. That distribution c
needs. In practical terms, for testing "enough"
all that important.

SEE ALSO: xreadlines *72;* linecache *64;* randor

1.2 Standard Modules

There are a variety of tasks that many or most
perform, but that are not themselves text proc
typically live inside files, so for a concrete appli
whether files exist, whether you have access to
attributes; you might also want to read their co
does not happen until the text makes it into a P
local memory is a necessary step.

Another task is making Python objects persiste
processing results can be saved in computer-us
applications often benefit from being able to ca
work with the results of those calls.

Yet another class of modules helps you deal wit
beyond what the inherent syntax does. I have r
to which such "Python internal" modules are su
in text processing applications; a number of "in

line descriptions under the "Other Modules" top

1.2.1 Working with the Python Interpreter

Some of the modules in the standard library co important to Python as the basic syntax. Such Python's design, but users of other languages for reading command-line arguments, catching like in external modules.

copy • Generic copying operations

Names in Python programs are merely bindings objects are mutable. This point is simple, but it beginning Python programmerand even a few e The problem is that binding another name (incl entry, or attribute) to an object leaves you with object. If you change the underlying object usi points to a changed object. Sometimes you wa

One variant of the binding trap is a particularly table of values, initialized as zeros. Later on, yo row/column position as, for example, table[2][languages). Here is what you would probably tr

```
>>> row = [0]*4
>>> print row
[0, 0, 0, 0]
>>> table = [row]*4   # or 'table =
>>> for row in table: print row
...
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
>>> table[2][3] = 7
>>> for row in table: print row
...
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 7]
>>> id(table[2]), id(table[3])
(6207968, 6207968)
```

The problem with the example is that table is a *exact same* list object. You cannot change just one object. What you need instead is a *copy* of

Python provides a number of ways to create co

names). Such a copy is a "snapshot" of the stat
independently of changes to the original. A few
are:

```
>>> table1 = map(list, [(0,)*4]*4)
>>> id(table1[2]), id(table1[3])
(6361712, 6361808)
>>> table2 = [lst[:] for lst in [[0]
>>> id(table2[2]), id(table2[3])
(6356720, 6356800)
>>> from copy import copy
>>> row = [0]*4
>>> table3 = map(copy, [row]*4)
>>> id(table3[2]), id(table3[3])
(6498640, 6498720)
```

In general, slices always create new lists. In Py
dict() likewise construct new/copied lists/dicts (
association types as arguments).

But the most general way to make a new copy
with the *copy* module. If you use the *copy* mod
issues of whether a given sequence is a list, or
coercion forces into a list.

FUNCTIONS

copy.copy(obj)

Return a shallow copy of a Python object. Most
objects can be copied. A shallow copy binds its
objects as bound in the originalbut the object it

```
>>> import copy
>>> class C: pass
...
>>> o1 = C()
>>> o1.lst = [1,2,3]
>>> o1.str = "spam"
>>> o2 = copy.copy(o1)
>>> o1.lst.append(17)
>>> o2.lst
[1, 2, 3, 17]
>>> o1.str = 'eggs'
>>> o2.str
'spam'
```

copy.deepcopy(obj)

Return a deep copy of a Python object. Each el
recursively copied. For nested containers, it is u

deep copyotherwise you can run into problems

```
>>> o1 = C()
>>> o1.lst = [1,2,3]
>>> o3 = copy.deepcopy(o1)
>>> o1.lst.append(17)
>>> o3.lst
[1, 2, 3]
>>> o1.lst
[1, 2, 3, 17]
```

exceptions • Standard exception class hie

Various actions in Python raise exceptions, and
an except clause. Although strings can serve as
compatibility reasons, it is greatly preferable to

When you catch an exception in using an excep
descendent exceptions. By utilizing a hierarchy
exception classes, you can tailor exception han
requirements.

```
>>> class MyException(StandardError)
...
>>> try:
```

```
...     raise MyException
... except StandardError:
...     print "Caught parent"
... except MyException:
...     print "Caught specific class
... except:
...     print "Caught generic leftov
...
Caught parent
```

In general, if you need to raise exceptions man
exception close to your situation, or inherit fror
in Figure 1.1 shows the exception classes defin

Figure 1.1. Standard

Exception	Root class for all built-in exceptions
StandardError	Base for "normal" exceptions
ArithmeticError	Base for arithmetic exceptions
OverflowError	Number too large to represent
ZeroDivisionError	Dividing by zero
FloatingPointError	Problem in floating point operation
LookupError	Problem accessing a value in a collection
IndexError	Problem accessing a value in a sequence
KeyError	Problem accessing a value in a mapping
NameError	Problem accessing local or global name
UnboundLocalError	Reference to non-existent name
AttributeError	Problem accessing or setting an attribute
TypeError	Operation or function applied to wrong type
ValueError	Operation or function on unusable value
UnicodeError	Problem encoding or decoding
EnvironmentError	Problem outside of Python itself
IOError	Problem performing I/O
OSError	Error passed from the operating system
WindowsError	Windows-specific OS problem
AssertionError	Failure of an assert statement
EOFError	End-of-file without a read
ImportError	Problem importing a module
ReferenceError	Problem accessing collected weakref
KeyboardInterrupt	User pressed interrupt (`ctrl-c`) key
MemoryError	Operation runs out of memory (try `del`'ing)
SyntaxError	Problem parsing Python code
SystemError	Internal (recoverable) error in Python
RuntimeError	Error not falling under any other category
NotImplementedError	Functionality not yet available
StopIteration	Iterator has no more items available
SystemExit	Raised by `sys.exit()`

getopt • Parser for command line options

Utility applicationswhether for text processing c
of command-line switches to configure their be
practice, all that you need to do to process com
list sys.argv[1:] and handle each element of th
my own small "sys.argv parser" more than onc
too much.

The *getopt* module provides some automation a

It takes just a few lines of code to tell *getopt* w
which switch prefixes and parameter styles to u
the final word in parsing command lines. Pytho
module <http://optik.sourceforge.net/> renam
Matrix library contains *twisted.python.usage*
<http://www.twistedmatrix.com/documents/ho
other third-party tools, were written because of

For most purposes, *getopt* is a perfectly good t
module is included in later Python versions, eit
backwards compatible or *getopt* will remain in t
scripts.

SEE ALSO: sys.argv *49;*

FUNCTIONS

getopt.getopt(args, options [,long_options]])

The argument args is the actual list of options l
sys.argv[1:]. The argument options and the op
formats for acceptable options. If any options s
acceptable format, a *getopt.GetoptError* except
with either a single dash for single-letter optior
(DOS-style leading slashes are not usable, unfo

The return value of *getopt.getopt()* is a pair co
additional arguments. The latter is typically a li
on. The option list is a list of pairs of the form (
of Python, you can convert an option list to a d
likely to be useful.

The options format string is a sequence of lette
colon. Any option letter followed by a colon tak
option.

The format for long_options is a list of strings i
the leading dashes). If an option name ends wi
after the option.

It is easiest to see *getopt* in action:

```
>>> import getopt
>>> opts='-al -b -c 2 --foo=bar --ba
>>> optlist, args = getopt.getopt(op
>>> optlist
[('-a', '1'), ('-b', ''), ('-c', '2'
('--baz', '')]
>>> args
['file1', 'file2']
>>> nodash = lambda s: \
...             s.translate(''.join(map
>>> todict = lambda l: \
```

```
...           dict([(nodash(opt),val)
>>> optdict = todict(optlist)
>>> optdict
{'a': '1', 'c': '2', 'b': '', 'baz':
```

You can examine options given either by loopin
optdict.get(key, default) type tests as needed i

operator • Standard operations as functio

All of the standard Python syntactic operators a
the *operator* module. In most cases, it is more
in a few cases functions are useful. The most c
conjunction with functional programming const

```
>>> import operator
>>> lst = [1, 0, (), '', 'abc']
>>> map(operator.not_, lst)    # fp-s
[0, 1, 1, 1, 0]
>>> tmplst = []                # impe
>>> for item in lst:
...     tmplst.append(not item)
...
>>> tmplst
```

```
[0, 1, 1, 1, 0]
>>> del tmplst                    # must
```

As well as being shorter, I find the FP style mor
provides *sample* implementations of the functic
implementations are faster and are written dire
what each function does.

operator2.py

```
### Comparison functions
lt = __lt__ = lambda a,b: a < b
le = __le__ = lambda a,b: a <= b
eq = __eq__ = lambda a,b: a == b
ne = __ne__ = lambda a,b: a != b
ge = __ge__ = lambda a,b: a >= b
gt = __gt__ = lambda a,b: a > b
### Boolean functions
not_ = __not__ = lambda o: not o
truth = lambda o: not not o
# Arithmetic functions
abs = __abs__ = abs   # same as buil
add = __add__ = lambda a,b: a + b
and_ = __and__ = lambda a,b: a & b
```

```
div = __div__ = \
      lambda a,b: a/b  # depends on
floordiv = __floordiv__ = lambda a,b
inv = invert = __inv__ = __invert__
lshift = __lshift__ = lambda a,b: a
rshift = __rshift__ = lambda a,b: a
mod = __mod__ = lambda a,b: a % b
mul = __mul__ = lambda a,b: a * b
neg = __neg__ = lambda o: -o
or_ = __or__ = lambda a,b: a | b
pos = __pos__ = lambda o: +o # ident
sub = __sub__ = lambda a,b: a - b
truediv = __truediv__ = lambda a,b:
xor = __xor__ = lambda a,b: a ^ b
### Sequence functions (note overloa
concat = __concat__ = add
contains = __contains__ = lambda a,b
countOf = lambda seq,a: len([x for x
def delitem(seq,a): del seq[a]
__delitem__ = delitem
def delslice(seq,b,e): del seq[b:e]
__delslice__ = delslice
getitem = __getitem__ = lambda seq,i
getslice = __getslice__ = lambda seq
```

```
index0f = lambda seq,o: seq.index(o)
repeat = __repeat__ = mul
def setitem(seq,i,v): seq[i] = v
__setitem__ = setitem
def setslice(seq,b,e,v): seq[b:e] = 
__setslice__ = setslice
### Functionality functions (not imp
# The precise interfaces required tc
#      are ill-defined, and might var
#      Python versions and custom dat
import operator
isCallable = callable      # just use
isMappingType = operator.isMappingTy
isNumberType = operator.isNumberType
isSequenceType = operator.isSequence
```

sys • Information about current Python in

As with the Python "userland" objects you creat
interpreter itself is very open to introspection. U
examine and modify many aspects of the Pythc
with much of the functionality in the *os* module
esoteric to address in this book about text proc
Reference for information on those attributes a

The module attributes *sys.exc_type, sys.exc_v* been deprecated in favor of the function *sys.ex* *sys.last-type, sys.last-value, sys.last_traceb* into exceptions and stack frames to a finer deg statements do. *sys.exec_prefix* and *sys.execut* paths for Python.

The functions *sys.displayhook()* and *sys.except* goes, and *sys.__displayhook__* and *sys.__exce* (e.g., STDOUT and STDERR). *sys.exitfunc* affec *sys.ps1* and *sys.ps2* control prompts in the Pyt

Other attributes and methods simply provide m to know for text processing applications. The a are Windows specific; *sys.setdlopenf lags ()*, ar Methods like *sys.builtin_module_names, sys._g* *sys.getrecursionlimit(), sys.setprofile(), sys.set* *sys.setrecursionlimit(), sys.modules*, and also s internals. Unicode behavior is affected by the s overridable with arguments anyway.

ATTRIBUTES

sys.argv

A list of command-line arguments passed to a

is the script name itself, so you are normally in
arguments.

SEE ALSO: getopt *44;* sys.stdin *51;* sys.stdout

sys.byteorder

The native byte order (endianness) of the curre
and little. Available in Python 2.0+.

sys.copyright

A string with copyright information for the curr

sys.hexversion

The version number of the current Python inter
increases with every version, even nonproducti
human-readable; *sys.version* or *sys.version_in*

SEE ALSO: sys.version *51;* sys.version_info *52*

sys.maxint

The largest positive integer supported by Pytho
platforms, 2**31-1. The largest negative integ

sys.maxunicode

The integer of the largest supported code point
current configuration. Unicode characters are s

sys.path

A list of the pathnames searched for modules. `
module loading.

sys.platform

A string identifying the OS platform.

SEE ALSO: os.uname() *81;*

sys.stderr
sys.__stderr__

File object for standard error stream (STDERR)

value in case *sys.stderr* is modified during prog
warnings from the Python interpreter are writte
of *sys.stderr* is for application messages that in
example:

```
% cat cap_file.py
#!/usr/bin/env python
import sys, string
if len(sys.argv) < 2:
    sys.stderr.write("No filename sp
else:
    fname = sys.argv[1]
    try:
        input = open(fname).read()
        sys.stdout.write(string.uppe
    except:
        sys.stderr.write("Could not
% ./cap_file.py this > CAPS
% ./cap_file.py nosuchfile > CAPS
Could not read 'nosuchfile'
% ./cap_file.py > CAPS
No filename specified
```

SEE ALSO: sys.argv *49;* sys.stdin *51;* sys.stdou

sys.stdin
sys.__stdin__

File object for standard input stream (STDIN). s
value in case *sys.stdin* is modified during progr
are read from *sys.stdin*, but the most typical us
redirected streams on the command line. For ex

```
% cat cap_stdin.py
#!/usr/bin/env python
import sys, string
input = sys.stdin.read()
print string.upper(input)
% echo "this and that" | ./cap_stdir
THIS AND THAT
```

SEE ALSO: sys.argv *49;* sys.stderr *50;* sys.stdo

sys.stdout
sys.__stdout__

File object for standard output stream (STDOUT
value in case *sys.stdout* is modified during prog
of the *print* statement goes to *sys.stdout*, and y
such as *sys.stdout.write()*.

SEE ALSO: sys.argv *49;* sys.stderr *50;* sys.stdi

sys.version

A string containing version information on the c
the string is version (#build_num, build_date, b

```
>>> print sys.version
1.5.2 (#0 Apr 13 1999, 10:51:12) [MS
```

Or:

```
>>> print sys.version
2.2 (#1, Apr 17 2002, 16:11:12)
[GCC 2.95.2 19991024 (release)]
```

This version-independent way to find the major
components should work for 1.5-2.3.x (at least

```
>>> from string import split
>>> from sys import version
>>> ver_tup = map(int, split(split(v
>>> major, minor, point = ver_tup[:3
>>> if (major, minor) >= (1, 6):
...     print "New Way"
```

```
... else:
...     print "Old Way"
...
New Way
```

sys.version_info

A 5-tuple containing five components of the ver
interpreter: (major, minor, micro, releaselevel,
phrase; the other are integers.

```
>>> sys.version_info
(2, 2, 0, 'final', 0)
```

Unfortunately, this attribute was added to Pyth
useful in requiring a minimal version for some c

SEE ALSO: sys.version *51;*

FUNCTIONS

sys.exit ([code=0])

Exit Python with exit code code. Cleanup actior

statements are honored, and it is possible to in
the SystemExit exception. You may specify a nu
that codify them; you may also specify a string
(with the actual exit code set to 1).

sys.getdefaultencoding()

Return the name of the default Unicode string e

sys.getrefcount(obj)

Return the number of references to the object c
than you might expect, because it includes the
argument.

```
>>> x = y = "hi there"
>>> import sys
>>> sys.getrefcount(x)
3
>>> lst = [x, x, x]
>>> sys.getrefcount(x)
6
```

SEE ALSO: os *74*;

types • Standard Python object types

Every object in Python has a type; you can find
type(). Often Python functions use a sort of *ad*
implemented by checking features of objects pa
coming from languages like C or Java are some
they are accustomed to seeing multiple "type s
types the function can accept. But that is not th

Experienced Python programmers try not to rel
even in an inheritance sense. This attitude is al
programmers of other languages (especially sta
important to a Python program is what an obje
become much more complicated to describe wh
"type/class unification" in Python 2.2 and above
this book).

For example, you might be inclined to write an
manner:

Naive overloading of argument

```
import types, exceptions
def overloaded_get_text(o):
    if type(o) is types.FileType:
```

```
        text = o.read()
    elif type(o) is types.StringType
        text = o
    elif type(o) in (types.IntType,
                     types.LongType,
        text = repr(o)
    else:
        raise exceptions.TypeError
    return text
```

The problem with this rigidly typed code is that
necessary. Something need not be an actual Fil
be sufficiently "file-like" (e.g., a *urllib.urlopen()*
like enough for this purpose). Similarly, a new-
types.StringType or a *UserString.UserString()* c
as such, and similarly for other numeric types.

A better implementation of the function above i

"Quacks like a duck" overloading of argumen

```
def overloaded_get_text(o):
    if hasattr(o,'read'):
        return o.read()
    try:
```

```
        return ""+o
    except TypeError:
        pass
    try:
        return repr(0+o)
    except TypeError:
        pass
    raise
```

At times, nonetheless, it is useful to have symb
object types. In many such cases, an empty or
may be used in conjunction with the *type()* fun
stylistic:

```
>>> type('') == types.StringType
1
>>> type(0.0) == types.FloatType
1
>>> type(None) == types.NoneType
1
>>> type([]) == types.ListType
1
```

BUILT-IN

type(o)

Return the datatype of any object o. The returr
object of the type *types.TypeType*. TypeType ol
.__repr__() methods to create readable descrip

```
>>> print type(1)
<type 'int'>
>>> print type(type(1))
<type 'type'>
>>> type(1) is type(0)
1
```

CONSTANTS

types.BuiltinFunctionType
types.BuiltinMethodType

The type for built-in functions like *abs(), len()*,
"standard" C extensions like *sys* and *os*. Howev
actually Python wrappers for C extensions, so t
types.FuntionType. A general Python programm
details.

types.BufferType

The type for objects created by the built-in buff

types.Class Type

The type for user-defined classes.

```
>>> from operator import eq
>>> from types import *
>>> map(eq, [type(C), type(C()), typ
...         [ClassType, InstanceType
[1, 1, 1]
```

SEE ALSO: types.InstanceType *56*; types.Metho

types.CodeType

The type for code objects such as returned by c

types.ComplexType

Same as type(0+0j).

types.DictType
types.DictionaryType

Same as type({}).

types.EllipsisType

The type for built-in Ellipsis object.

types.FileType

The type for open file objects.

```
>>> from sys import stdout
>>> fp = open('tst','w')
>>> [type(stdout), type(fp)] == [typ
1
```

types.FloatType

Same as type (0.0).

types.FrameType

The type for frame objects such as tb.tb_frame
types.TracebackType.

types.FunctionType
types.LambdaType

Same as type(lambda:0).

types.GeneratorType

The type for generator-iterator objects in Pythc

```
>>> from __future__ import generator
>>> def foo(): yield 0
...
>>> type(foo) == types.FunctionType
1
>>> type(foo()) == types.GeneratorTy
1
```

SEE ALSO: types.FunctionType *56;*

types.InstanceType

The type for instances of user-defined classes.

SEE ALSO: types.ClassType *55;* types.MethodTy

types.IntType

Same as type(0).

types.ListType

Same as type().

types.LongType

Same as type(OL).

types.MethodType
types.Unbound MethodType

The type for methods of user-defined class inst

SEE ALSO: types.ClassType *55*; types.Instance

types.ModuleType

The type for modules.

```
>>> import os, re, sys
>>> [type(os), type(re), type(sys)]
1
```

types.NoneType

Same as type(None).

types.StringType

Same as type("").

types.TracebackType

The type for traceback objects found in *sys.exc*

types.TupleType

Same as type(()).

types.UnicodeType

Same as type(u"").

types.SliceType

The type for objects returned by slice().

types.StringTypes

Same as (types.StringType,types.UnicodeType)

SEE ALSO: types.StringType *57*; types.Unicode

types.TypeType

Same as type (type (obj)) (for any obj).

types.XRangeType

Same as type(xrange(1)).

1.2.2 Working with the Local Filesystem

dircache • Read and cache directory listin

The *dircache* module is an enhanced version of
function, *dircache* keeps prior directory listings
new call to the filesystem. Since *dircache* is sm
directory has been touched since last caching,
os.listdir() (with possible minor speed gains).

FUNCTIONS

dircache.listdir(path)

Return a directory listing of path path. Uses a li

dircache.opendir(path)

Identical to *dircache.listdir()*. Legacy function t

dircache.annotate(path, lst)

Modify the list lst in place to indicate which iten
files. The string path should indicate the path t

```
>>> l = dircache.listdir('/tmp')
>>> l
['501', 'md10834.db']
>>> dircache.annotate('/tmp', l)
>>> l
['501/', 'md10834.db']
```

filecmp • Compare files and directories

The *filecmp* module lets you check whether twc
directories contain some identical files. You hav
thorough of a comparison is performed.

FUNCTIONS

filecmp.cmp(fname1, fname2 [,shallow=1 [,us

Compare the file named by the string fname1 w
fname2. If the default true value of shallow is u
the mode, size, and modification time of the tw
files are compared byte by byte. Unless you are
deliberately falsify timestamps on files (as in a
comparison is quite reliable. However, tar and u

```
>>> import filecmp
>>> filecmp.cmp('dir1/file1', 'dir2/
0
>>> filecmp.cmp('dir1/file2', 'dir2/
1
```

The use_statcache argument is not relevant for
versions, the *statcache* module provided (slight
stats, but its use is no longer needed.

filecmp.cmpfiles(dirname1, dirname2, fnamel

Compare those filenames listed in fnamelist if t
dirname1 and the directory dirname2. *filecmp.c*
(some of the lists may be empty): (matches, m
identical files in both directories, mismatches a
directories. errors will contain names if a file ex
two directories, or if either file cannot be read f
problems, etc.).

```
>>> import filecmp, os
>>> filecmp.cmpfiles('dirl','dir2',[
(['this'], ['that'], ['other'])
>>> print os.popen('ls -l dir1').rea
-rwxr-xr-x    1 quilty   staff      1
-rwxr-xr-x    1 quilty   staff      6
-rwxr-xr-x    1 quilty   staff      7
-rwxr-xr-x    1 quilty   staff      5
>>> print os.popen('ls -l dir2').rea
-rwxr-xr-x    1 quilty   staff      1
-rwxr-xr-x    1 quilty   staff      6
```

The shallow and use_statcache arguments are

CLASSES

filecmp.dircmp(dirname1, dirname2 [,ignore=

Create a directory comparison object. dirname1
compare. The optional argument ignore is a seq
defaults to ["RCS","CVS","tags"]; hide is a sequ
defaults to [os.curdir,os.pardir] (i.e., [".",".."]).

METHODS AND ATTRIBUTES

The attributes of *filecmp.dircmp* are read-only.

filecmp.dircmp.report()

Print a comparison report on the two directorie

```
>>> mycmp = filecmp.dircmp('dir1','c
>>> mycmp.report()
diff dir1 dir2
Only in dir1 : ['other', 'spam']
Identical files : ['this']
Differing files : ['that']
```

filecmp.dircmp.report_partial_closure()

Print a comparison report on the two directorie
The method name has nothing to do with the th
functional programming.

filecmp.dircmp.report_partial_closure()

Print a comparison report on the two directorie
subdirectories.

filecmp.dircmp.left_list

Pathnames in the dirname1 directory, filtering o

filecmp.dircmp.right_list

Pathnames in the dirname2 directory, filtering o

filecmp.dircmp.common

Pathnames in both directories.

filecmp.dircmp.left_only

Pathnames in dirname 1 but not dirname2.

filecmp.dircmp.right_only

Pathnames in dirname2 but not dirname1.

filecmp.dircmp.common_dirs

Subdirectories in both directories.

filecmp.dircmp.common_files

Filenames in both directories.

filecmp.dircmp.common_funny

Pathnames in both directories, but of different t

filecmp.dircmp.same_files

Filenames of identical files in both directories.

filecmp.dircmp.diff_files

Filenames of nonidentical files whose name occ

filecmp.dircmp.funny_files

Filenames in both directories where something

filecmp.dircmp.subdirs

A dictionary mapping *filecmp.dircmp.common_*
filecmp.dircmp objects; for example:

```
>>> usercmp = filecmp.dircmp('/Users
>>> usercmp.subdirs['Public'].commor
['Drop Box']
```

SEE ALSO: os.stat() *79;* os.listdir() *76;*

fileinput • Read multiple files or STDIN

Many utilities, especially on Unix-like systems,
files and/or on redirected input. A flexibility in t
homogeneous fashion is part of the "Unix philos
you to write a Python application that uses thes
no special programming to adjust to input sour

A common, minimal, but extremely useful Unix
input to STDOUT (allowing redirection of STDOU
examples of cat:

```
% cat a
AAAAA
```

```
% cat a b
AAAAA
BBBBB
% cat - b < a
AAAAA
BBBBB
% cat < b
BBBBB
% cat a < b
AAAAA
% echo "XXX" | cat a -
AAAAA
XXX
```

Notice that STDIN is read only if either "-" is gi
are given at all. We can implement a Python ve
as follows:

cat.py

```
#!/usr/bin/env python
import fileinput
for line in fileinput.input():
        print line,
```

FUNCTIONS

fileinput.input([files=sys.argv[1:] [,inplace=0 |

Most commonly, this function will be used witho
in the introductory example of cat.py. However,
special cases.

The argument files is a sequence of filenames t
the arguments given on the command line. Cor
treat some of these arguments as flags rather t
- or /). Any list of filenames you like may be us
not it is built from sys.argv.

If you specify a true value for inplace, output w
than to STDOUT. Input taken from STDIN, howe
place operation, a temporary backup file is crea
given the extension indicated by the backup ar

```
% cat a b
AAAAA
BBBBB
% cat modify.py
#!/usr/bin/env python
import fileinput, sys
for line in fileinput.input(sys.argv
```

```
        print "MODIFIED", line,
% echo "XXX" | ./modify.py a b -
MODIFIED XXX
% cat a b
MODIFIED AAAAA
MODIFIED BBBBB
```

fileinput.close()

Close the input sequence.

fileinput.nextfile()

Close the current file, and proceed to the next
file will not be counted towards the line total.

There are several functions in the *fileinput* mod
current input state. These tests can be used to
dependent way.

fileinput.filelineno()

The number of lines read from the current file.

fileinput.filename()

The name of the file from which the last line wa
function returns None.

fileinput.isfirstline()

Same as fileinput.filelineno()==1.

fileinput.isstdin()

True if the last line read was from STDIN.

fileinput.lineno()

The number of lines read during the input loop,

CLASSES

fileinput.FileInput([files [,inplace=0 [,backup=

The methods of *fileinput.FileInput* are the same
an additional .readline() method that matches t
objects also have a .__getitem__() method to s

The arguments to initialize a *fileinput.FileInput*
to the *fileinput.input ()* function. The class exist
subclassing. For normal usage, it is best to just

SEE ALSO: multifile *285;* xreadlines *72;*

glob • Filename globing utility

The *glob* module provides a list of pathnames r
fnmatch module is used internally to determine

FUNCTIONS

glob.glob(pat)

Both directories and plain files are returned, so
of path, use *os.path.isdir()* or *os.path.isfile()*; c
other filters.

Pathnames returned by *glob.glob()* contain as r
information as the pattern pat gives. For examp

```
>>> import glob, os.path
>>> glob.glob('/Users/quilty/Book/ch
['/Users/quilty/Book/chap3.txt', '/U
>>> glob.glob('chap[3-6].txt')
['chap3.txt', 'chap4.txt', 'chap5.tx
>>> filter(os.path.isdir, glob.glob(
['/Users/quilty/Book/SCRIPTS', '/Use
```

SEE ALSO: fnmatch *232;* os.path *65;*

linecache • Cache lines from files

The module *linecache* can be used to simulate r
the lines in a file. Lines that are read are cache

FUNCTIONS

linecache.getline(fname, linenum)

Read line linenum from the file named fname. I
function will catch the error and return an emp
the filename if it is not found in the current dire

```
>>> import linecache
>>> linecache.getline('/etc/hosts',
'192.168.1.108   hermes  hermes.gnos
```

linecache.clearcache()

Clear the cache of read lines.

linecache.checkcache()

Check whether files in the cache have been mo

os.path • Common pathname manipulatio

The *os.path* module provides a variety of functi
filesystem paths in a cross-platform fashion.

FUNCTIONS

os.path.abspath(pathname)

Return an absolute path for a (relative) pathna

```
>>> os.path.abspath('SCRIPTS/mk_book
'/Users/quilty/Book/SCRIPTS/mk_book'
```

os.path.basename(pathname)

Same as os.path.split(pathname)[1].

os .path.commonprefix(pathlist)

Return the path to the most nested parent dire
sequence pathlist.

```
>>> os.path.commonprefix(['/usr/X11R
...                        '/usr/sbir
...                        '/usr/loca
'/usr/'
```

os.path.dirname(pathname)

Same as os.path.split(pathname)[0].

os.path.exists(pathname)

Return true if the pathname pathname exists.

os.path.expanduser(pathname)

Expand pathnames that include the tilde charac
initial tilde refers to a user's home directory, ar
the named user's home directory. This function
platforms.

```
>>> os.path.expanduser('~dqm')
'/Users/dqm'
>>> os.path.expanduser('~/Book')
'/Users/quilty/Book'
```

os.path.expandvars(pathname)

Expand pathname by replacing environment va
function is in the *os.path* module, you could eq
Python, generally (this is not necessarily a goo

```
>>> os.path.expandvars('$HOME/Book')
'/Users/quilty/Book'
```

```
>>> from os.path import expandvars a
>>> if ev('$HOSTTYPE')=='macintosh'
...     print ev("The vendor is $VEN
...
The vendor is apple, the CPU is powe
```

os.path.getatime(pathname)

Return the last access time of pathname (or rai
possible).

os.path.getmtime(pathname)

Return the modification time of pathname (or r
possible).

os.path.getsize(pathname)

Return the size of pathname in bytes (or raise

os.path.isabs(pathname)

Return true if pathname is an absolute path.

os.path.isdir(pathname)

Return true if pathname is a directory.

os.path.isfile(pathname)

Return true if pathname is a regular file (includ

os.path.islink(pathname)

Return true if pathname is a symbolic link.

os.path.ismount(pathname)

Return true if pathname is a mount point (on P

os.path.join(path1 [,path2 [...]])

Join multiple path components intelligently.

```
>>> os.path.join('/Users/quilty/','E
'/Users/quilty/Book/SCRIPTS/mk_book'
```

os.path.normcase(pathname)

Convert pathname to canonical lowercase on ca
convert slashes on Windows systems.

os.path.normpath(pathname)

Remove redundant path information.

```
>>> os.path.normpath('/usr/local/bir
'/usr/local/include/slang.h'
```

os.path.realpath(pathname)

Return the "real" path to pathname after de-ali
Python 2.2+.

```
>>> os.path.realpath('/usr/bin/newal
'/usr/sbin/sendmail'
```

os.path.samefile(pathname1, pathname2)

Return true if pathname1 and pathname2 are t

SEE ALSO: filecmp *58;*

os.path.sameopenfile(fp1, fp2)

Return true if the file handles fp1 and fp2 refer
Windows.

os.path.split(pathname)

Return a tuple containing the path leading up t
directory or filename in isolation.

```
>>> os.path.split('/Users/quilty/Boo
('/Users/quilty/Book', 'SCRIPTS')
```

os.path.splitdrive(pathname)

Return a tuple containing the drive letter and th
do not use a drive letter, the drive letter is emp

Windows-like systems).

os.path.walk(pathname, visitfunc, arg)

For every directory recursively contained in pat
pathnames) for each path.

```
>>> def big_files(minsize, dirname,
...     for file in files:
...         fullname = os.path.join(
...         if os.path.isfile(fullna
...             if os.path.getsize(f
...                 print fullname
...
>>> os.path.walk('/usr/', big_files,
/usr/lib/libSystem.B_debug.dylib
/usr/lib/libSystem.B_profile.dylib
```

shutil • Copy files and directory trees

The functions in the *shutil* module make workir
nothing in this module that you could not do us
functions, but *shutil* often provides a more dire

for you. The functions in *shutil* match fairly clos
Unix filesystem utilities like cp and rm.

FUNCTIONS

shutil.copy(src, dst)

Copy the file named src to the pathname dst. I
given the name os.path.join(dst+os.path.baser

SEE ALSO: os.path.join() *66;* os.path.basenam

shutil.copy2(src, dst)

Same as *shutil.copy()* except that the access ar
values in src.

shutil.copyfile(src, dst)

Copy the file named src to the filename dst (ov
has the same effect as open(dst,"wb").write(op

shutil.copyfileobj(fpsrc, fpdst [,buffer=-1])

Copy the file-like object fpsrc to the file-like obj
buffer is given, only the specified number of by
this allows copying very large files.

shutil.copymode(src, dst)

Copy the permission bits from the file named sr

shutil.copystat(src, dst)

Copy the permission and timestamp data from

shutil.copytree(src, dst [,symlinks=0])

Copy the directory src to the destination dst rec
symlinks is a true value, copy symbolic links as
of copying the content of the link target. This fu
every platform and filesystem.

shutil.rmtree(dirname [ignore [,errorhandler]]

Remove an entire directory tree rooted at dirna
true value, errors will be silently ignored. If err

handler is used to catch errors. This function m
platform and filesystem.

SEE ALSO: open() *15;* os.path *65;*

stat • Constants/functions for os.stat()

The *stat* module provides two types of support
os.lstat(), and *os.fstat()* calls.

Several functions exist to allow you to perform
check one predicate of a file, it is more direct t
functions, but for performing several such tests
and perform several *stat.S_*()* tests.

As well as helper functions, *stat* defines symbol
10-tuple returned by *os.stat()* and friends. For

```
>>> from stat import *
>>> import os
>>> fileinfo = os.stat('chap1.txt')
>>> fileinfo[ST_SIZE]
68666L
>>> mode = fileinfo [ST_MODE]
>>> S_ISSOCK(mode)
0
```

```
>>> S_ISDIR(mode)
0
>>> S_ISREG(mode)
1
```

FUNCTIONS

stat.S_ISDIR(mode)

Mode indicates a directory.

stat.S_ISCHR(mode)

Mode indicates a character special device file.

stat.S_ISBLK(mode)

Mode indicates a block special device file.

stat.S_ISREG(mode)

Mode indicates a regular file.

stat.S_ISFIFO(mode)

Mode indicates a FIFO (named pipe).

stat.S_ISLNK(mode)

Mode indicates a symbolic link.

stat.S_ISSOCK(mode)

Mode indicates a socket.

CONSTANTS

stat.ST_MODE

I-node protection mode.

stat.ST_INO

I-node number.

stat.ST_DEV

Device.

stat.ST_NLINK

Number of links to this i-node.

stat.ST_UID

User id of file owner.

stat.ST_GID

Group id of file owner.

stat.ST_SIZE

Size of file.

stat.ST_ATIME

Last access time.

stat.ST_MTIME

Modification time.

stat.ST_CTIME

Time of last status change.

tempfile • Temporary files and filenames

The *tempfile* module is useful when you need t
interface. In contrast to the file-like interface o
filesystem for storage rather than simulating th
memory-constrained contexts, therefore, *temp*

The temporary files created by *tempfile* are as
as is supported by the underlying platform. You
temporary data will not be read or changed eith
afterwards (temporary files are deleted when c

tempfile to provide you with cryptographic-leve
accidents and casual inspection.

FUNCTIONS

tempfile.mktemp([suffix=""])

Return an absolute path to a unique temporary
is specified, the name will end with the suffix st

tempfile.TemporaryFile([mode="w+b" [,buffsi

Return a temporary file object. In general, ther
mode argument of w+b; there is no existing fil
and it does little good to write temporary data
optional suffix argument generally will not ever
when closed. The default buffsize uses the platf
needed.

```
>>> tmpfp = tempfile.TemporaryFile()
>>> tmpfp.write('this and that\n')
>>> tmpfp.write('something else\n')
>>> tmpfp.tell()
29L
```

```
>>> tmpfp.seek(0)
>>> tmpfp.read()
'this and that\nsomething else\n'
```

SEE ALSO: StringIO *153*; cStringIO *153*;

xreadlines • Efficient iteration over a file

Reading over the lines of a file had some pitfall
was a memory-friendly way, and there was a fa
meet. These techniques were:

```
>>> fp = open('bigfile')
>>> line = fp.readline()
>>> while line:
...     # Memory-friendly but slow
...     # ...do stuff...
...     line = fp.readline()

>>> for line in open('bigfile').reac
...     # Fast but memory-hungry
...     # ...do stuff...
```

Fortunately, with Python 2.1 a more efficient te
2.2+, this efficient technique was also wrapped

keeping with the new iterator). With Python 2.3
in favor of the idiom "for line in file:".

FUNCTIONS

xreadlines.xreadlines(fp)

Iterate over the lines of file object fp in an effic
memory usage).

```
>>> for line in xreadlines.xreadline
...     # Efficient all around
...     # ...do stuff...
```

Corresponding to this *xreadlines* module functio
objects.

```
>>> for line in open('tmp').xreadlir
...     # As a file object method
...     # ...do stuff...
```

If you use Python 2.2 or above, an even nicer v

```
>>> for line in open('tmp'):
...     # ...do stuff...
```

SEE ALSO: linecache *64;* FILE.xreadlines() *17;*

1.2.3 Running External Commands and Acces

commands • Quick access to external com

The *commands* module exists primarily as a co
os.popen()* functions on Unix-like systems. ST
results.

FUNCTIONS

commands.getoutput(cmd)

Return the output from running cmd. This funct

```
>>> def getoutput(cmd):
...     import os
...     return os.popen('{ '+cmd+';
```

commands.getstatusoutput(cmd)

Return a tuple containing the exit status and ou could also be implemented as:

```
>>> def getstatusoutput(cmd):
...     import os
...     fp = os.popen('{ '+cmd+'; }
...     output = fp.read()
...     status = fp.close()
...     if not status: status=0 # Wa
...     return (status, output)
...
>>> getstatusoutput('ls nosuchfile')
(256, 'ls: nosuchfile: No such file
>>> getstatusoutput('ls c*[1-3].txt'
(0, 'chap1.txt\nchap2.txt\nchap3.txt
```

commands.getstatus(filename)

Same as commands.getoutput('ls -ld '+filenam

SEE ALSO: os.popen() *77*; os.popen2() *77*; os.

os • Portable operating system services

The *os* module contains a large number of func
calling on or determining features of the operat
many cases, functions in *os* are internally imple
riscos, or *mac*, but for portability it is better to

Not everything in the *os* module is documented
those features that are unlikely to be used in te
Python Library Reference that accompanies Pyt

Functions and constants not documented here
functions and attributes *os.confstr(), os.confstr*
os.sysconf_names let you probe system configu
specific to process permissions on Unix-like sys
os.geteuid(), os.getgid(), os.getgroups(), os.ge
os.getuid(), os.setegid(), os.seteuid(), os.setgi
os.setpgid(), os.setreuid(), os.setregid(), os.se

The functions *os.abort(), os.exec*(), os._exit()*
os.spawn(), os.times(), os.wait(), os.waitpid(,*
os.WSTOPSIG()', and *os.WTERMSIG()* and the
deal with process creation and management. T
since creating and managing multiple processe
processing tasks. However, I briefly document t
os.nice(), os.startfile(), and *os.system()* and in
omitted functionality can also be found in the *c*

A number of functions in the os module allow y
descriptors. In general, it is simpler to perform

built-in *open()* function or the *os.popen*()* fam
like *FILE.readline(), FILE.write(), FILE.seek()*, a
files can be determined using the *os.stat()* func
shutil modules. Therefore, the functions *os.clos*
os.fpathconf(), os.fstat(), os.fstatvfs(), os.ftrun
os.open(), os.openpty(), os.pathconf(), os.pipe
os.tcgetpgrp(), os.tcsetpgrp(), os.ttyname(), o
covered here. As well, the supporting constants
omitted.

SEE ALSO: commands *73;* os.path *65;* shutil *6*

FUNCTIONS

os.access(pathname, operation)

Check the permission for the file or directory pa
specified is allowed, return a true value. The ar
between 0 and 7, inclusive, and encodes four fe
and readable. These features have symbolic na

```
>>> import os
>>> os.F_OK, os.X_OK, os.W_OK, os.R_
(0, 1, 2, 4)
```

To query a specific combination of features, you features.

```
>>> os.access('myfile', os.W_OK | os
1
>>> os.access('myfile', os.X_OK + os
0
>>> os.access('myfile', 6)
1
```

os.chdir(pathname)

Change the current working directory to the pa

SEE ALSO: os.getcwd() *75;*

os.chmod(pathname, mode)

Change the mode of file or directory pathname
page for the chmod utility for more information

os.chown(pathname, uid, gid)

Change the owner and group of file or directory

See the man page for the chown utility for mor

os.chroot(pathname)

Change the root directory under Unix-like syste
page for the chroot utility for more information

os.getcwd()

Return the current working directory as a string

```
>>> os.getcwd()
'/Users/quilty/Book'
```

SEE ALSO: os.chdir() *75*;

os.getenv(var [,value=None])

Return the value of environment variable var. If
defined, return value. An equivalent call is os. e

SEE ALSO: os.environ *81;* os.putenv() *78;*

os.getpid()

Return the current process id. Possibly useful fc
process id's.

SEE ALSO: os.kill() *76;*

os.kill(pid, sig)

Kill an external process on Unix-like systems. Y
the pid argument by some means, such as a ca
sig sent to the process may be found in the *sig*
example:

```
>>> from signal import  *
>>> SIGHUP, SIGINT, SIGQUIT, SIGIOT,
(1, 2, 3, 6, 9)
>>> def kill_by_name(progname):
...     pidstr = os.popen('ps|grep '
...     pid = int(pidstr.split()[0])
...     os.kill(pid, 9)
...
>>> kill_by_name('myprog')
```

os.link(src, dst)

Create a hard link from path src to path dst on
on the ln utility for more information.

SEE ALSO: os.symlink() *80;*

os.listdir(pathname)

Return a list of the names of files and directorie
entries for the current and parent directories (t
the list.

os.lstat(pathname)

Information on file or directory pathname. See
not follow symbolic links.

SEE ALSO: os.stat() *79;* stat *69;*

os.mkdir(pathname [,mode=0777])

Create a directory named pathname with the n
operating systems, mode is ignored. See the m

information on modes.

SEE ALSO: os.chmod() *75;* os.mkdirs() *77*;

os.mkdirs(pathname [,mode=0777])

Create a directory named pathname with the n
this function will create any intermediate direct

SEE ALSO: os.mkdir() *76;*

os.mkfifo(pathname [,mode=0666])

Create a named pipe on Unix-like systems.

os.nice(increment)

Decrease the process priority of the current ap
is useful if you do not wish for your application

The four functions in the *os.popen*()* family all
capture their STDOUT and STDERR and/or set t
family differ somewhat in how these three pipe

os.popen(cmd [,mode="r" [,bufsize]])

Open a pipe to or from the external command c
is an open file object connected to the pipe. Th
or w for write. The exit status of the command
closed. An optional buffer size bufsize may be s

```
>>> import os
>>> def ls(pat):
...     stdout = os.popen('ls '+pat)
...     result = stdout.read()
...     status = stdout.close()
...     if status: print "Error stat
...     else: print result
...
>>> ls('nosuchfile')
ls: nosuchfile: No such file or dire
Error status 256
>>> ls('chap[7-9].txt')
chap7.txt
```

os.popen2(cmd [,mode [,bufsize]])

Open both STDIN and STDOUT pipes to the ext

is a pair of file objects connecting to the two re
as with *os.popen()*.

SEE ALSO: os.popen3() *78;* os.popen() *77*;

os.popen3(cmd [,mode [,bufsize]])

Open STDIN, STDOUT, and STDERR pipes to th
value is a 3-tuple of file objects connecting to t
bufsize work as with *os.popen()*.

```
>>> import os
>>> stdin, stdout, stderr = os.poper
>>> print >>stdin, 'line one'
>>> print >>stdin, 'line two'
>>> stdin.write('line three\n)'
>>> stdin.close()
>>> stdout.read()
'LINE one\nLINE two\nLINE three\n'
>>> stderr.read()
''
```

os.popen4(cmd [,mode [,bufsize]])

Open STDIN, STDOUT, and STDERR pipes to th
to *os.popen3(), os.popen4()* combines STDOUT
return value is a pipe of file objects connecting
bufsize work as with *os.popen()*.

SEE ALSO: os.popen3() *78;* os.popen() *77*;

os.putenv(var, value)

Set the environment variable var to the value v
environment only affect subprocesses of the cu
with *os.system()* or *os.popen()*, not the whole

Calls to *os.putenv()* will update the environmer
Therefore, it is better to update *os.environ* dire
environment).

SEE ALSO: os.environ *81;* os.getenv() *75;* os.p

os.readlink(linkname)

Return a string containing the path symbolic lin
like systems.

SEE ALSO: os.symlink() *80;*

os.remove(filename)

Remove the file named filename. This function
cannot be removed, an OSError is raised.

SEE ALSO: os.unlink() *81;*

os.removedirs(pathname)

Remove the directory named pathname and an
function will not remove directories with files, a
to do so.

SEE ALSO: os.rmdir() *79;*

os.rename(src, dst)

Rename the file or directory src as dst. Depend
operation may raise an OSError if dst already e

SEE ALSO: os.renames() *79;*

os.renames(src, dst)

Rename the file or directory src as dst. Unlike c
any intermediate directories needed for a neste

SEE ALSO: os.rename() *79;*

os.rmdir(pathname)

Remove the directory named pathname. This fu
directories and will raise an OSError if you atter

SEE ALSO: os.removedirs() *79;*

os.startfile(path)

Launch an application under Windows system.
was double-clicked in a Drives window or as if y
line. Using Windows associations, a data file ca
an actual executable application.

SEE ALSO: os.system() *80;*

os.stat(pathname)

Create a stat_result object that contains inform

pathname. A stat_result object has a number o
tuple of numeric values. Before Python 2.2, onl
attributes of a stat_result object are named the
module, but in lowercase.

```
>>> import os, stat
>>> file_info = os.stat('chap1.txt')
>>> file_info.st_size
87735L
>>> file_info [stat.ST_SIZE]
87735L
```

On some platforms, additional attributes are av
systems usually have .st_blocks, .st_blksize, ar
.st_rsize, .st_creator, and .st_type; RISCOS ha

SEE ALSO: stat *69;* os.lstat() *76;*

os.strerror(code)

Give a description for a numeric error code cod
os.popen(bad_cmd).close().

SEE ALSO: os.popen() *77*;

os.symlink(src, dst)

Create a soft link from path src to path dst on U
on the ln utility for more information.

SEE ALSO: os.link() *76;* os.readlink() *78;*

os.system(cmd)

Execute the command cmd in a subshell. Unlike
output of the executed process is not captured
terminal as the current Python application). In
on non-Windows systems to detach an applicat
. For example, under MacOSX, you could launch

```
>>> import os
>>> cmd="/Applications/TextEdit.app/
>>> os.system(cmd)
0
```

SEE ALSO: os.popen() *77;* os.startfile() *79;* com

os.tempnam([dir [,prefix]])

Return a unique filename for a temporary file. I
that directory will be used in the path; if prefix
indicated prefix. For most purposes, it is more s
obtain a file object rather than first generating

SEE ALSO: tempfile *71;* os.tmpfile() *80;*

os.tmpfile()

Return an "invisible" file object in update mode
entry, but simply acts as a transient buffer for d

SEE ALSO: tempfile *71;* StringIO *153;* cStringIO

os.uname()

Return detailed information about the current o
systems. The returned 5-tuple contains sysnam
machine, each as descriptive strings.

os.unlink(filename)

Remove the file named filename. This function
cannot be removed, an OSError is raised.

SEE ALSO: os.remove() *78;*

os.utime(pathname, times)

Set the access and modification timestamps of
mtime) specified in times. Alternately, if times i
current time.

SEE ALSO: time *86;* os.chmod() *75;* os.chown(

CONSTANTS AND ATTRIBUTES

os.altsep

Usually None, but an alternative path delimiter

os.curdir

The string the operating system uses to refer to
"." on Unix or ":" on Macintosh (before MacOSX

os.defpath

The search path used by exec*p*() and spawn
variable.

os.environ

A dictionary-like object containing the current e

```
>>> os.environ['TERM']
'vt100'
>>> os.environ['TERM'] = 'vt220'
>>> os.getenv('TERM')
'vt220'
```

SEE ALSO: os.getenv() *75;* os.putenv() *78;*

os.linesep

The string that delimits lines in a file; for exam
"\r\n" on Windows.

os.name

A string identifying the operating system the cu on. Possible strings include posix, nt, dos, mac,

os.pardir

The string the operating system uses to refer t
".." on Unix or "::" on Macintosh (before MacOS

os.pathsep

The string that delimits search paths; for exam

os.sep

The string the operating system uses to refer t
Unix, "\" on Windows, ":" on Macintosh.

SEE ALSO: sys *49;* os.path *65;*

1.2.4 Special Data Values and Formats

random • Pseudo-random value generator

Python provides better pseudo-random number
with a rand() function, but not good enough for
of Python's Wichmann-Hill generator is about 7
indicates how long it will take a particular seed
will produce a different sequence of numbers. P
Twister generator, which has a longer period an
practical purposes, pseudorandom numbers gen
adequate for random-seeming behavior in appl

The underlying pseudo-random numbers gener
mapped into a variety of nonuniform patterns a
capture and tinker with the state of a pseudo-ra
subclass the *random.Random* class that operat
latter sort of specialization is outside the scope
random.Random and functions *random.getstate*
random.setstate() are omitted from this discus
and *random.randint()* are deprecated.

FUNCTIONS

random.betavariate(alpha, beta)

Return a floating point value in the range [0.0,

random.choice(seq)

Select a random element from the nonempty se

random.cunifvariate(mean, arc)

Return a floating point value in the range [mea
uniform distribution. Arguments and result are

random.expovariate(lambda_)

Return a floating point value in the range [0.0,
The argument lambda_ gives the *inverse* of the

```
>>> import random
>>> t1,t2 = 0,0
>>> for x in range(100):
...     t1 += random.expovariate(1./
...     t2 += random.expovariate(20.
...
>>> print t1/100, t2/100
18.4021962198 0.0558234063338
```

random.gamma(alpha, beta)

Return a floating point value with a gamma dist

random.gauss(mu, sigma)

Return a floating point value with a Gaussian di
sigma is sigma. *random.gauss()* is slightly faste

random.lognormvariate(mu, sigma)

Return a floating point value with a log normal
this distribution is Gaussian with mean mu and

random.normalvariate(mu, sigma)

Return a floating point value with a Gaussian di
sigma is sigma.

random.paretovariate(alpha)

Return a floating point value with a Pareto distr
parameter.

random.random()

Return a floating point value in the range [0.0,

random.randrange([start=0,] stop [,step=1])

Return a random element from the specified ra
expression random.choice(range(start,stop,ste
range object. Use *random.randrange()* in place

random.seed([x=time.time()])

Initialize the Wichmann-Hill generator. You do r
random.seed(), since the current system time i
module import. But if you wish to provide more
pass any hashable object as argument x. Your l
integer less than 27814431486575L, whose val
independent means.

random.shuffle(seq [,random=random.randor

Permute the mutable sequence seq in place. Ar
specified to use an alternate random generator,

one. Possible permutations get very big very qu
sequences, not every permutation will occur.

random.uniform(min, max)

Return a random floating point value in the ran

random.vonmisesvariate(mu, kappa)

Return a floating point value with a von Mises c
expressed in radians, and kappa is the concent

random.weibullvariate(alpha, beta)

Return a floating point value with a Weibull dist
and beta is the shape parameter.

struct • Create and read packed binary str

The *struct* module allows you to encode compa
module may also be used to read C structs that
formatting codes are only useful for reading C s

raised if a format does not match its string or v

A format string consists of a sequence of alpha
represented by zero or more bytes in the encod
formatting code may be preceded by a number
The entire format string may be preceded by a
platform-native data sizes and endianness are
sizes are used. The flag = explicitly indicates pl
endian representations; > or ! indicates big-en

The available formatting codes are listed below
your platform for its sizes if platform-native siz

Formatting codes for struct module

x	pad byte	0 b
c	char	1 b
b	signed char	1 b
B	unsigned char	1 b
h	short int	2 b
H	unsigned short	2 b
i	int	4 b
I	unsigned int	4 b
l	long int	4 b
L	unsigned long	4 b

```
q       long long int              8 b
Q       unsigned long long         8 b
f       float                      4 b
d       double                     8 b
s       string                     pac
p       Pascal string              pac
P       char pointer               4 b
```

Some usage examples clarify the encoding:

```
>>> import struct
>>> struct.pack('5s5p2c', 'sss','ppp
'sss\x00\x00\x03ppp\x00cc'
>>> struct.pack('h', 1)
'\x00\x01'
>>> struct.pack('I', 1)
'\x00\x00\x00\x01'
>>> struct.pack('l', 1)
'\x00\x00\x00\x01'
>>> struct.pack('<l', 1)
'\x01\x00\x00\x00'
>>> struct.pack('f', 1)
'?\x80\x00\x00'
>>> struct.pack('hil', 1,2,3)
'\x00\x01\x00\x00\x00\x00\x00\x02\x0
```

FUNCTIONS

struct.calcsize(fmt)

Return the length of the string that corresponds

struct.pack(fmt, v1 [,v2 [...]])

Return a string with values v1, et alia, packed a

struct.unpack(fmt, s)

Return a tuple of values represented by string s

time • Functions to manipulate date/time

The *time* module is useful both for computing a increments, and for simple benchmarking of ap purposes, eGenix.com's *mx.Date* module is mo than is *time*. You may obtain *mx.Date* from:

<http://egenix.com/files/python/eGenix-mx-E

Time tuplesused by several functionsconsist of
second, weekday, Julian day, and Daylight Savi
Month, day, and Julian day (day of year) are or
weekday are zero-based (Monday is 0). The Da
for Standard Time, and -1 for "best guess."

CONSTANTS AND ATTRIBUTES

time.accept2dyear

Boolean to allow two-digit years in date tuples.
the first matching date since time.gmtime(0) is

```
>>> import time
>>> time.accept2dyear
1
>>> time.localtime(time.mktime((99,1
(1999, 1, 1, 0, 0, 0, 4, 1, 0)
>>> time.gmtime(0)
(1970, 1, 1, 0, 0, 0, 3, 1, 0)
```

time.altzone
time.daylight
time.timezone

time.tzname

These several constants show information on th
locations use Daylight Savings adjustments dur
usually but not always a one-hour adjustment.
such an adjustment is available in *time.altzone*
seconds west of UTC the current zone is; *time.a*
Savings if possible. *time.tzname* gives a tuple o

```
>>> time.daylight, time.tzname
(1, ('EST', 'EDT'))
>>> time.altzone, time.timezone
(14400, 18000)
```

FUNCTIONS

time.asctime([tuple=time.localtime()])

Return a string description of a time tuple.

```
>>> time.asctime((2002, 10, 25, 1, 5
'Fri Oct 25 01:51:48 2002'
```

SEE ALSO: time.ctime() *87;* time.strftime() *88,*

time.clock()

Return the processor time for the current proce
inherent meaning, but the value is guaranteed
the amount of CPU time used by the process. T
comparative benchmarking of various operatior
should not be compared between different CPU
on one machine. For example:

```
import time
start1 = time.clock()
approach_one()
time1 = time.clock()-start1
start2 = time.clock()
approach_two()
time2 = time.clock()-start2
if time1 > time2:
    print "The second approach seems
else:
    print "The first approach seems
```

Always use *time.clock()* for benchmarking rathe
low-resolution "wall clock" only.

time.ctime([seconds=time.time()])

Return a string description of seconds since epo

```
>>> time.ctime(1035526125)
'Fri Oct 25 02:08:45 2002'
```

SEE ALSO: time.asctime() *87;*

time.gmtime([seconds=time.time()])

Return a time tuple of seconds since epoch, giv

```
>>> time.gmtime(1035526125)
(2002, 10, 25, 6, 8, 45, 4, 298, 0)
```

SEE ALSO: time.localtime() *88;*

time.localtime([seconds=time.time()])

Return a time tuple of seconds since epoch, giv

```
>>> time.localtime(1035526125)
(2002, 10, 25, 2, 8, 45, 4, 298, 1)
```

SEE ALSO: time.gmtime() *88;* time.mktime() *8*

time.mktime(tuple)

Return a number of seconds since epoch corres

```
>>> time.mktime((2002, 10, 25, 2, 8,
1035526125.0
```

SEE ALSO: time.localtime() *88;*

time.sleep(seconds)

Suspend execution for approximately seconds r
time). The argument seconds is a floating point
timer) and is fully thread safe.

time.strftime(format [,tuple=time.localtime()])

Return a custom string description of a time tu
format may contain the following fields: %a/%
weekday name; %b/%B/%m for abbreviated/fu
abbreviated/full year; %d for day-of-month; %
day-of-year; %M for minute; %p for AM/PM; %
year (Sunday/Monday start); %c/%x/%X for lo
%Z for timezone name. Other characters may

appear as literals (a literal % can be escaped).

```
>>> import time
>>> tuple = (2002, 10, 25, 2, 8, 45,
>>> time.strftime("%A, %B %d '%y (we
"Friday, October 25 '02 (week 42)"
```

SEE ALSO: time.asctime() *87;* time.ctime() *87;*

time.strptime(s [,format="%a %b %d %H:%M:

Return a time tuple based on a string descriptic
string format follows the same rules as in *time.*
platforms.

SEE ALSO: time.strftime() *88;*

time.time()

Return the number of seconds since the epoch
specifically determine the epoch using time.ctir
functions in the *time* module to generate usefu
also generally nondecreasing in its return value
benchmarking purposes.

```
>>> time.ctime(0)
'Wed Dec 31 19:00:00 1969'
>>> time.time()
1035585490.484154
>>> time.ctime(1035585437)
'Fri Oct 25 18:37:17 2002'
```

SEE ALSO: time.clock() *87;* time.ctime() *87;*

SEE ALSO: calendar *100;*

1.3 Other Modules in the Standard L

If your application performs other types of task
this module list can suggest where to look for r
who find themselves maintaining code written b
unfamiliar modules are imported by the existin
summarized in the list below, nor documented
third-party module. For standard library module
you a sense of the general purpose of a given r

__builtin__

Access to built-in functions, exceptions, and oth
exposing its own internals, but "normal" develo

1.3.1 Serializing and Storing Python Objects

In object-oriented programming (OOP) languag
structured data is frequently represented at rur
objects belong to basic datatypeslists, tuples, a
you reach a certain degree of complexity, hiera
become more likely.

For simple objects, especially sequences, serial
straightforward. For example, lists can easily be
length strings. Lists-of-lists can be saved in line
delimited fields, or in rows of RDBMS tables. Bu
sequences goes past two, and even more so fo
traditional table-oriented storage is a less-obvio

While it is *possible* to create "object/relational a
flat tables, that usually requires custom progra
solutions exist, both in the Python standard libr
actually two separate issues involved in storing
to convert them into strings in the first place; t
general persistence mechanism for such seriali
course, it is simple enough to store (and retriev
you would any other stringto a file, a database,
create a "dictionary on disk," while the *shelve* r
serialization to write arbitrary objects as values

Several third-party modules support object seri
need an XML dialect for your object representat
xmlrpclib are useful. The YAML format is both h
support libraries for Python, Perl, Ruby, and Jav

can exchange objects between these several pr

SEE ALSO: gnosis.xml.pickle *410;* yaml *415*; xı

DBM • Interfaces to dbm-style databases

A dbm-style database is a "dictionary on disk."
to store a set of key/val pairs to a file, or files,
and set them as if they were an in-memory dic
standard dictionary, always maps strings to stri
objects, you will need to convert them to string
wrapper).

Depending on your platform, and on which exte
dbm modules might be available. The performa
modules vary significantly. As well, some DBM ı
functionality. Most of the time, however, your b
supported DBM module using the wrapper mod
select the best available DBM for the current en
user having to worry about the underlying supp

Functions and methods are documents using th
real usage, you would use the name of a specif
get or set DBM values using standard named in
methods characteristic of dictionaries are also s
special to DBM databases.

SEE ALSO: shelve *98;* dict *24;* UserDict *24;*

FUNCTIONS

DBM.open(fname [,flag="r" [,mode=0666]])

Open the filename fname for dbm access. The
the database is accessed. A value of r is for rea
w opens an already existing file for read/write
an existing one, with read/write access; the op
database, erasing the one named in fname if it
argument specifies the Unix mode of the file(s)

METHODS

DBM.close()

Close the database and flush any pending write

DBM.first()

Return the first key/val pair in the DBM. The or

the *DBM.first()* method, combined with repeate
item in the dictionary.

In Python 2.2+, you can implement an items()
.items() method of dictionaries for DBMs:

```
>>> from __future__ import generator
>>> def items(db):
...     try:
...         yield db.first()
...         while 1:
...             yield db.next()
...     except KeyError:
...         raise StopIteration
...
>>> for k,v in items(d):   # typical
...     print k,v
```

DBM.has_key(key)

Return a true value if the DBM has the key key.

DBM.keys()

Return a list of string keys in the DBM.

DBM.last()

Return the last key/val pair in the DBM. The ord
the *DBM.last()* method, combined with repeate
every item in the dictionary in reverse order.

DBM.next()

Return the next key/val pair in the DBM. A poir
maintained, so the methods *DBM.next()* and *D*
relative items.

DBM.previous()

Return the previous key/val pair in the DBM. A
maintained, so the methods *DBM.next()* and *D*
relative items.

DBM.sync()

Force any pending data to be written to disk.

SEE ALSO: FILE.flush() *16;*

MODULES

anydbm

Generic interface to underlying DBM support. C
of the "best available" DBM module. If you ope
guessed and usedassuming the current machin

SEE ALSO: whichdb *93;*

bsddb

Interface to the Berkeley DB library.

dbhash

Interface to the BSD DB library.

dbm

Interface to the Unix (n)dbm library.

dumbdbm

Interface to slow, but portable pure Python DBI

gdbm

Interface to the GNU DBM (GDBM) library.

whichdb

Guess which db package to use to open a db fil
function *whichdb.whichdb()*. If you open an exi
function is called automatically behind the scen

SEE ALSO: shelve *98;*

cPickle • Fast Python object serialization

pickle • Standard Python object serializati

The module *cPickle* is a comparatively fast C im
module. The streams produced and read by *cPi*
The only time you should prefer *pickle* is in the
subclass the pickling base class; *cPickle* is man
pickle.Pickler is not documented here.

The *cPickle* and *pickle* modules support a both
designed for human readability, but it is not hu
Nonetheless, if readability is a goal, *yaml* or *gn*
Binary format produces smaller pickles that are

It is possible to fine-tune the pickling behavior
.__getstate__(), .__setstate__(), and .__getini
invocations involved in defining these methods,
book and are rarely necessary for "normal" obj
structures).

Use of the *cPickle* or *pickle* module is quite sim

```
>>> import cPickle
>>> from somewhere import my_complex
>>> s = cPickle.dumps(my_complex_obj
>>> new_obj = cPickle.loads(s)
```

FUNCTIONS

pickle.dump(o, file [,bin=0])
cPickle.dump(o, file [,bin=0])

Write a serialized form of the object o to the fil
argument bin is given a true value, use binary

pickle.dumps(o [,bin=0])
cPickle.dumps(o [,bin=0])

Return a serialized form of the object o as a str
given a true value, use binary format.

pickle.load(file)
cPickle.load(file)

Return an object that was serialized as the con

pickle.loads(s)
cPickle.load(s)

Return an object that was serialized in the strin

SEE ALSO: gnosis.xml.pickle *410;* yaml *415;*

marshal

Internal Python object serialization. For more g
cPickle, or *gnosis.xml.pickle*, or the YAML tools
limited-purpose serialization to the pseudo-com
.pyc files.

pprint • Pretty-print basic datatypes

The module *pprint* is similar to the built-in func
purpose of *pprint* is to represent objects of bas
especially in cases where collection types nest i
pprint.pformat and *repr()* produce the same res
uses newlines and indentation to illustrate the s
possible, the string representation produced by
create objects with the built-in *eval()* .

I find the module *pprint* somewhat limited in th
helpful representation of objects of custom type
compound data. Instance attributes are very fr
dictionary keys. For example:

```
>>> import pprint
>>> dct = {1.7:2.5, ('t','u','p'):['
>>> dct2 = {'this':'that', 'num':38,
```

```
>>> class Container: pass
...
>>> inst = Container()
>>> inst.this, inst.num, inst.dct =
>>> pprint.pprint(dct2)
{'dct': {('t', 'u', 'p'): ['l', 'i',
 'num': 38,
 'this': 'that'}
>>> pprint.pprint(inst)
<__main__.Container instance at 0x41
```

In the example, dct2 and inst have the same st
chosen in an application as a data container. Bu
tells us the barest information about *what* an o
mini-module below enhances pretty-printing:

pprint2.py

```
from pprint import pformat
import string, sys
def pformat2(o):
    if hasattr(o,'__dict__'):
        lines = []
        klass = o.__class__.__name__
```

```
        module = o.__module__
        desc = '<%s.%s instance at 0
        lines.append(desc)
        for k,v in o.__dict__.items(
            lines.append('instance.%
        return string.join(lines,'\n
    else:
        return pprint.pformat(o)

def pprint2(o, stream=sys.stdout):
    stream.write(pformat2(o)+'\n')
```

Continuing the session above, we get a more u

```
>>> import pprint2
>>> pprint2.pprint2(inst)
<__main__.Container instance at 0x41
instance.this='that'
instance.dct={('t', 'u', 'p'): ['l',
instance.num=38
```

FUNCTIONS

pprint.isreadable(o)

Return a true value if the equality below holds:

```
o == eval(pprint.pformat(o))
```

pprint.isrecursive(o)

Return a true value if the object o contains recu
themselves at any nested level cannot be resto

pprint.pformat(o)

Return a formatted string representation of the

pprint.pprint(o [,stream=sys.stdout])

Print the formatted representation of the object

CLASSES

pprint.PrettyPrinter(width=80, depth=..., inden

Return a pretty-printing object that will format

recursion to depth depth, and will indent each r
pprint.PrettyPrinter.pprint() will write to the file

```
>>> pp = pprint.PrettyPrinter(width=
>>> pp.pprint(dct2)
{'dct': {1.7: 2.5,
         ('t', 'u', 'p'): ['l',
                           'i',
                           's',
                           't']},
 'num': 38,
 'this': 'that'}
```

METHODS

The class *pprint.PrettyPrinter* has the same me
The only difference is that the stream used for
configured when an instance is initialized rathe

SEE ALSO: gnosis.xml.pickle *410;* yaml *415;*

repr • Alternative object representation

The module *repr* contains code for customizing

its default behavior the function *repr.repr()* pro
representation of objectsin the case of large co
can be unwieldy, and unnecessary for merely d

```
>>> dct = dict([(n,str(n)) for n in
>>> repr(dct)      # much worse for,
"{0: '0', 1: '1', 2: '2', 3: '3', 4:
>>> from repr import repr
>>> repr(dct)
"{0: '0', 1: '1', 2: '2', 3: '3', ..
>>>   'dct'
"{0: '0', 1: '1', 2: '2', 3: '3', 4:
```

The back-tick operator does not change behavi
replaced.

You can change the behavior of the *repr.repr()*
object *repr.aRepr*.

```
>>> dct = dict([(n,str(n)) for n in
>>> repr(dct)
"{0: '0', 1: '1', 2: '2', 3: '3', 4:
>>> import repr
>>> repr.repr(dct)
"{0: '0', 1: '1', 2: '2', 3: '3', ..
>>> repr.aRepr.maxdict = 5
```

```
>>> repr.repr(dct)
"{0: '0', 1: '1', 2: '2', 3: '3', 4:
```

In my opinion, the choice of the name for this r
identical to that of the built-in function. You car
the as form of importing, as in:

```
>>> import repr as _repr
>>> from repr import repr as newrepr
```

For fine-tuned control of object representation,
Potentially, you could use substitutable repr() fu
application output, but if you anticipate such a
name that indicates this; for example, overrida

CLASSES

repr.Repr()

Base for customized object representations. Th
exists in the module namespace, so this class is
change an attribute, it is simplest just to set it

ATTRIBUTES

repr.maxlevel

Depth of recursive objects to follow.

repr.maxdict
repr.maxlist
repr.maxtuple

Number of items in a collection of the indicated
Sequences default to 6, dicts to 4.

repr.maxlong

Number of digits of a long integer to stringify. D

repr.maxstring

Length of string representation (e.g., s[:N]). De

repr.maxother

"Catch-all" maximum length of other represent

FUNCTIONS

repr.repr(o)

Behaves like built-in *repr()*, but potentially with
created.

repr.repr_TYPE(o, level)

Represent an object of the type TYPE, where th
names. The argument level indicates the level
(you might want to decide what to print based
the object is). The *Python Library Reference* giv

```
class MyRepr(repr.Repr):
    def repr_file(self, obj, level):
        if obj.name in ['<stdin>', '
            return obj.name
        else:
            return 'obj'
aRepr = MyRepr()
print aRepr.repr(sys.stdin)
```

shelve • General persistent dictionary

The module *shelve* builds on the capabilities of
step forward. Unlike with the DBM modules, yo
values in a *shelve* database. The keys in *shelve*
strings.

The methods of *shelve* databases are generally
DBMs. However, shelves do not have the .first(
methods; nor do they have the .items () metho
the time you will simply use name-indexed ass
time, the available *shelve.get(), shelve.keys(),*
shelve.close() methods are useful.

Usage of a shelve consists of a few simple step

```
>>> import shelve
>>> sh = shelve.open('test_shelve')
>>> sh.keys()
['this']
>>> sh['new_key'] = {1:2, 3:4, ('t',
>>> sh.keys()
['this', 'new_key']
>>> sh['new_key']
{1: 2, 3: 4, ('t', 'u', 'p'): ['l',
>>> del sh['this']
```

```
>>> sh.keys()
['new_key']
>>>  sh.close()
```

In the example, I opened an existing shelve, ar
was available. Deleting a key/value pair is the s
dictionary. Opening a new shelve automatically

Although *shelve* only allows strings to be used
generate strings that characterize other types c
reasons that you do not generally want to use r
also a bad idea to use mutable objects as *shelv*
method is a good way to generate stringsbut ke
strictly guarantee uniqueness, so it is possible
entries using this hack:

```
>>> '%x' % hash((1,2,3,4,5))
'866123f4'
>>> '%x' % hash(3.1415)
'6aad0902'
>>> '%x' % hash(38)
'26'
>>> '%x' % hash('38')
'92bb58e3'
```

Integers, notice, are their own hash, and string
you adopted this approach, you would want to

as keys. There is no real problem with doing so
you need to remember to use consistently:

```
>>> sh['%x' % hash('another_key')] =
>>> sh.keys()
['new_key', '8f9ef0ca']
>>> sh['%x' % hash('another_key')]
'another value'
>>> sh['another_key']
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/sw/lib/python2.2/shelve.py"
    f = StringIO(self.dict[key])
KeyError: another_key
```

If you want to go beyond the capabilities of *she*
investigate the third-party library Zope Object I
arbitrary objects to be persistent, not only dicti
you store data in ways other than in local files,
simultaneous access. Look for details at:

<http://www.zope.org/Wikis/ZODB/Standalon

SEE ALSO: DBM 90; dict *24;*

The rest of the listed modules are comparativel
processing applications. Some modules are spe
is indicated parenthetically. Recent distributions
included" approachmuch more is included in a l
other free programming languages (but other p
existing libraries that can be downloaded separ

1.3.2 Platform-Specific Operations

_winreg

Access to the Windows registry (Windows).

AE

AppleEvents (Macintosh; replaced by *Carbon.Ar*

aepack

Conversion between Python variables and Appl

aetypes

AppleEvent objects (Macintosh).

applesingle

Rudimentary decoder for AppleSingle format fil

buildtools

Build MacOS applets (Macintosh).

calendar

Print calendars, much like the Unix cal utility. A
or stringify calendars for various time frames. F

```
>>> print calendar.month(2002,11)
    November 2002
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Carbon.AE, Carbon.App, Carbon.CF, Carbon.
Carbon.Evt, Carbon.Fm, Carbon.Help, Carbor
Carbon.Qd, Carbon.Qdoffs, Carbon.Qt, Carbc
Carbon.TE, Carbon.Win

Interfaces to Carbon API (Macintosh).

cd

CD-ROM access on SGI systems (IRIX).

cfmfile

Code Fragment Resource module (Macintosh).

ColorPicker

Interface to the standard color selection dialog

ctb

Interface to the Communications Tool Box (Mac

dl

Call C functions in shared objects (Unix).

EasyDialogs

Basic Macintosh dialogs (Macintosh).

fcntl

Access to Unix fcntl() and iocntl() system funct

findertools

AppleEvents interface to MacOS finder (Macinto

fl, FL, flp

Functions and constants for working with the F

fm, FM

Functions and constants for working with the F

fpectl

Floating point exception control (Unix).

FrameWork, MiniAEFrame

Structured development of MacOS applications

gettext

The module *gettext* eases the development of r
translations must be performed manually, this r
translation and runtime substitutions of langua

grp

Information on Unix groups (Unix).

locale

Control the language and regional settings for a
the behavior of several functions, such as *time.*
module is also useful for creating strings such a
currency strings for specific nations.

mac, macerrors, macpath

Macintosh implementation of *os* module functio
directly and let it call *mac* where needed (Macir

macfs, macfsn, macostools

Filesystem services (Macintosh).

MacOS

Access to MacOS Python interpreter (Macintosh

macresource

Locate script resources (Macintosh).

macspeech

Interface to Speech Manager (Macintosh).

mactty

Easy access serial to line connections (Macintos

mkcwproject

Create CodeWarrior projects (Macintosh).

msvcrt

Miscellaneous Windows-specific functions provi
libraries (Windows).

Nac

Interface to Navigation Services (Macintosh).

nis

Access to Sun's NIS Yellow Pages (Unix).

pipes

Manage pipes at a finer level than done by *os.p*
varies between platforms (Unix).

PixMapWrapper

Wrap PixMap objects (Macintosh).

posix, posixfile

Access to operating system functionality under
portable version of the same functionality and s

preferences

Application preferences manager (Macintosh).

pty

Pseudo terminal utilities (IRIX, Linux).

pwd

Access to Unix password database (Unix).

pythonprefs

Preferences manager for Python (Macintosh).

py_resource

Helper to create PYC resources for compiled ap

quietconsole

Buffered, nonvisible STDOUT output (Macintosh

resource

Examine resource usage (Unix).

syslog

Interface to Unix syslog library (Unix).

tty, termios, TERMIOS

POSIX tty control (Unix).

W

Widgets for the Mac (Macintosh).

waste

Interface to the WorldScript-Aware Styled Text

winsound

Interface to audio hardware under Windows (W

xdrlib

Implements (a subset of) Sun eXternal Data Re
is similar to the *struct* module, but the format i

1.3.3 Working with Multimedia Formats

aifc

Read and write AIFC and AIFF audio files. The i
sunau and *wave* modules.

al, AL

Audio functions for SGI (IRIX).

audioop

Manipulate raw audio data.

chunk

Read chunks of IFF audio data.

colorsys

Convert between RGB color model and YIQ, HLS

gl, DEVICE, GL

Functions and constants for working with Silico

imageop

Manipulate image data stored as Python strings
the third-party *Python Imaging Library* (usually
<http://www.pythonware.com/products/pil/>)

imgfile

Support for imglib files (IRIX).

jpeg

Read and write JPEG files on SGI (IRIX). The *Py*
(<http://www.pythonware.com/products/pil/>)
working with a large number of image formats

rgbimg

Read and write SGI RGB files (IRIX).

sunau

Read and write Sun AU audio files. The interfac
and *wave* modules.

sunaudiodev, SUNAUDIODEV

Interface to Sun audio hardware (SunOS/Solari

videoreader

Read QuickTime movies frame by frame (Macin

wave

Read and write WAV audio files. The interface t
sunau modules.

1.3.4 Miscellaneous Other Modules

array

Typed arrays of numeric values. More efficient t
applicable.

atexit

Exit handlers. Same functionality as *sys.exitfun*

BaseHTTPServer, SimpleHTTPServer, Simple

HTTP server classes. *BaseHTTPServer* should us
The other modules provide sufficient customiza
indicated by their names. All may be customize

Bastion

Restricted object access. Used in conjunction w

bisect

List insertion maintaining sort order.

cmath

Mathematical functions over complex numbers.

cmd

Build line-oriented command interpreters.

code

Utilities to emulate Python's interactive interpre

codeop

Compile possibly incomplete Python source cod

compileall

Module/script to compile .py files to cached byt

compile, compile.ast, compile.visitor

Analyze Python source code and generate Pytho

copy_reg

Helper to provide extensibility for pickle/cPickle

curses, curses.ascii, curses.panel, curses.tex

Full-screen terminal handling with the (n)curse

dircache

Cached directory listing. This module enhances

dis

Disassembler of Python byte-code into mnemoı

distutils

Build and install Python modules and packages
mechanism for creating distribution packages c
for installing them on target machines. Althoug
processing applications that are distributed to ι
working with *distutils* is outside the scope of th
found in the Python standard documentation, e
Python Modules and *Installing Python Modules*.

doctest

Check the accuracy of _*doc*_ strings.

errno

Standard errno system symbols.

fpformat

General floating point formatting functions. Duן

functionality.

gc

Control Python's (optional) cyclic garbage collec

getpass

Utilities to collect a password without echoing t

imp

Access the internals of the import statement.

inspect

Get useful information from live Python objects

keyword

Check whether string is a Python keyword.

math

Various trigonometric and algebraic functions a
operate on floating point numbersuse *cmath* fo

mutex

Work with mutual exclusion locks, typically for

new

Create special Python objects in customizable w
create a module object without using a file of tl
while bypassing the normal .__init__() call. "Nc
text processing applications.

pdb

A Python debugger.

popen2

Functions to spawn commands with pipes to ST
In Python 2.0+, this functionality is copied to tl
Generally you should use the os module (unless
earlier).

profile

Profile the performance characteristics of Pytho
your application, your first step in solving any p
code. But details of using *profile* are outside th
usually a bad idea to *assume* speed is a probler

pstats

Print reports on profiled Python code.

pyclbr

Python class browser; useful for implementing
editing Python.

pydoc

Extremely useful script and module for examini
included with Python 2.1+, but is compatible w
pydoc can provide help similar to Unix man pag
also a Web browser interface to documentation
while developing Python applications, but its de

py_compile

"Compile" a .py file to a .pyc (or .pyo) file.

Queue

A multiproducer, multiconsumer queue, especia

readline, rlcompleter

Interface to GNU readline (Unix).

rexec

Restricted execution facilities.

sched

General event scheduler.

signal

Handlers for asynchronous events.

site, user

Customizable startup module that can be modif
Python installation.

statcache

Maintain a cache of *os.stat()* information on file

statvfs

Constants for interpreting the results of *os.stat*

thread, threading

Create multithreaded applications with Python.
applicationslike other applicationsmight use a t
the scope of this book. Most, but not all, Pytho
applications.

Tkinter, ScrolledText, Tix, turtle

Python interface to TCL/TK and higher-level wi
platforms, but not on all Python installations.

traceback

Extract, format, and print information about Py
applications.

unittest

Unit testing framework. Like a number of other
modules, *unittest* is a useful facilityand its usa
applications in general. But this module is not s
applications to be addressed in this book.

warnings

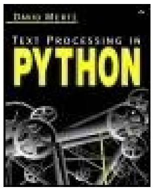
Python 2.1 added a set of warning messages fc
but that fall below the threshold for raising exc
printed to STDERR, but the *warning* module ca
warning messages.

weakref

Create references to objects that do not limit g
references seem strange, and the strangeness
do not know why you would want to use these,
to.

whrandom

Wichmann-Hill random number generator. Depr
necessary to use directly before thatuse the mc
values.



Text Processing in PythonBy David Mertz

Chapter 2. Basic String Operations

> The cheapest, fastest and most reliable comp
> of a computer system are those that aren't th
>
> Gordon Bell, Encore Computer Corporation

If you are writing programs in Python to accom
processing tasks, most of what you need to kn
this chapter. Sure, you will probably need to kn
to do some basic things with pipes, files, and a
to get your text to process (covered in Chapter
for actually *processing* the text you have gotte
string module and string methodsand Python's
data structuresdo most all of what you need do
almost all the time. To a lesser extent, the vari
custom modules to perform encodings, encrypt
compressions are handy to have around (and y
certainly do not want the work of implementing
yourself). But at the heart of text processing a
transformations of bits of text. That's what *stri*
functions and string methods do.

There are a lot of interesting techniques elsewh
this book. I wouldn't have written about them i
not find them important. But be cautious befor

interesting things. Specifically, given a fixed tas
mind, before cracking this book open to any of
chapters, consider very carefully whether your
can be solved using the techniques in this chap
you can answer this question affirmatively, you
usually eschew the complications of using the h
level modules and techniques that other chapte
discuss. By all means read all of this book for th
and edification that I hope it provides; but still
the "Zen of Python," and prefer simple to comp
simple is enough.

This chapter does several things. Section 2.1 lo
number of common problems in text processing
(and should) be solved using (predominantly) t
techniques documented in this chapter. Each of
"Problems" presents working solutions that can
adopted with little change to real-life jobs. But
goal is to provide readers with a starting point
adaptation of the examples. It is not my goal to
mere collections of packaged utilities and modu
of those exist on the Web, and resources like th
of Parnassus <http://www.vex.net/parnassus/>
Python Cookbook
<http://aspn.activestate.com/ASPN/Python/Co
are worth investigating as part of any project/ta
new and better utilities will be written between
I write this and when you read it). It is better f

readers to receive a solid foundation and startir
from which to develop the functionality they ne
their own projects and tasks. And even better t
spurring adaptation, these examples aim to enc
contemplation. In presenting examples, this bo
to embody a way of thinking about problems ar
attitude towards solving them. More than any ir
technique, such ideas are what I would most lik
share with readers.

Section 2.2 is a "reference with commentary" o
Python standard library modules for doing basi
manipulations. The discussions interspersed wi
module try to give some guidance on why you
want to use a given module or function, and th
reference documentation tries to contain more
of actual typical usage than does a plain refere
many cases, the examples and discussion of in
functions addresses common and productive de
patterns in Python. The cross-references are in
contextualize a given function (or other thing) i
of related ones (and to help you decide which i
you). The actual listing of functions, constants,
and the like is in alphabetical order within type

Section 2.3 in many ways continues Section 2.
also provides some aids for using this book in a
context. The problems and solutions presented

Section 2.3 are somewhat more open-ended th
in Section 2.1. As well, each section labeled as
"Discussion" is followed by one labeled "Questi
These questions are ones that could be assigne
teacher to students; but they are also intended
issues that general readers will enjoy and bene
contemplating. In many cases, the questions p
limitations of the approaches initially presented
readers to think about ways to address or move
these limitationsexactly what readers need to d
writing their own custom code to accomplish ou
tasks. However, each Discussion in Section 2.3
stand on its own, even if the Questions are skip
by the reader.

Chapter 2. Basic String Operations

2.1 Some Common Tasks

2.1.1 Problem: Quickly sorting lines on custo

Sorting is one of the real meat-and-potatoes al
most programming. Fortunately for Python dev
extraordinarily fast. Moreover, Python lists with
elements can be sortedPython cannot rely on th
unfortunate exception to this general power wa
comparisons of complex numbers raise a TypeE
reason; Unicode strings in lists can cause simila

SEE ALSO: complex *22*;

The list sort method is wonderful when you wan
the order that Python considers natural, in the
a lot of times, you want to sort things in "unnat
any order that is not simple alphabetization of t

contain meaningful bits of information in positi
last name may occur as the second word of a li
the first word); an IP address may occur severa
may occur at position 70 of each line; and so o
style of meaningful order that Python doesn't q

The list sort method *[].sort()* supports an optio
The job this function has is to return -1 if the fi
things are equal order-wise, and return 1 if the
function *cmp()* does this in a manner identical t
speed, 1st.sort() is much faster than 1st.sort(c
custom comparison function is probably the bes
with an in-line lambda function as the custom c
handy idiom.

When it comes to speed, however, use of custo
the problem is Python's function call overhead,
slowness. Fortunately, a technique called "Schw
custom sorts. Schwartzian Transforms are nam
technique for working with Perl; but the technic

The pattern involved in the Schwartzian Transfo
can more precisely be called the Guttman-Rosl
Schwartzian Transform):

1. **Transform the list in a reversible way in**

- Call Python's native *[].sort()* method.

- Reverse the transformation in (1) to restore tl

The reason this technique works is that, for a li transformation operations, which is easy to am compare/flip operations for large lists. The sort that makes the sort more efficient is a win in th

Below is an example of a simple, but plausible, fourth and subsequent words of a list of input li sort to the bottom. Running the test against a f megabyteperformed the Schwartzian Transform 12 seconds for the custom comparison function number of factors will change the exact relative generally be expected.

schwartzian_sort.py

```
# Timing test for "sort on fourth wo
# Specifically, two lines >= 4 words
#   lexographically on the 4th, 5th,
#   Any line with fewer than four wo
#   the end, and will occur in "natu

import sys, string, time
wrerr = sys.stderr.write
```

```
# naive custom sort
def fourth_word(ln1,ln2):
    lst1 = string.split(ln1)
    lst2 = string.split(ln2)
    #-- Compare "long" lines
    if len(lst1) >= 4 and len(lst2)
        return cmp(lst1[3:],lst2[3:]
    #-- Long lines before short line
    elif len(lst1) >= 4 and len(lst2
        return -1
    #-- Short lines after long lines
    elif len(lst1) < 4 and len(lst2)
        return 1
    else:                       # Natura
        return cmp(ln1,ln2)

# Don't count the read itself in the
lines = open(sys.argv[1]).readlines(

# Time the custom comparison sort
start = time.time()
lines.sort(fourth_word)

end = time.time()
wrerr("Custom comparison func in %3.
```

```
# open('tmp.custom','w').writelines(

# Don't count the read itself in the
lines = open(sys.argv[1]).readlines(

# Time the Schwartzian sort
start = time.time()
for n in range(len(lines)):        #
    lst = string.split(lines[n])
    if len(lst) >= 4:              #
        lines[n] = (lst[3:], lines[r
    else:                          #
        lines[n] = (['\377'], lines[

lines.sort()                       #

for n in range(len(lines)):        #
    lines[n] = lines[n] [1]

end = time.time()
wrerr("Schwartzian transform sort ir
# open('tmp.schwartzian','w').writel
```

Only one particular example is presented, but r
technique to any sort they need to perform frec

2.1.2 Problem: Reformatting paragraphs of te

While I mourn the decline of plaintext ASCII as unnecessarily complicated and large (and often left in text files full of prose. READMEs, HOWTC are written in plaintext (or at least something c processing techniques are valuable). Moreover, frequently enough hand-edited that their plaint

One task that is extremely common when work paragraphs to conform to desired margins. Pytl performs more limited reformatting than the cc done within text editors, which are indeed quite sometimes it would be nice to automate the for that it is slightly surprising that Python has no s the class *formatter.DumbWriter*, or the possibil *formatter.AbstractWriter*. These classes are dis of customization and sophistication needed to u way out of proportion for the task at hand.

Below is a simple solution that can be used eith STDIN and writing to STDOUT) or by import to

reformat_para.py

```
# Simple paragraph reformatter.  All
```

```
# of left and right margins, and of
# (using constants defined in module

LEFT,RIGHT,CENTER = 'LEFT','RIGHT','

def reformat_para(para='',left=0,ric
    words = para.split()
    lines = []
    line  = ''
    word = 0
    end_words = 0
    while not end_words:
        if len(words[word]) > right-
            line = words[word]
            word +=1
            if word >= len(words):
                end_words = 1
        else:
            while len(line)+len(worc
                line += words[word]+
                word += 1
                if word >= len(words
                    end_words = 1
                    break
        lines.append(line)
```

```
        line = ''
    if just==CENTER:
        r, l = right, left
        return '\n'.join([' '*left+l
    elif just==RIGHT:
        return '\n'.join([line.rjust
    else: # left justify
        return '\n'.join([' '*left+l

if __name__=='__main__':
    import sys
    if len(sys.argv) <> 4:
        print "Please specify left_m
    else:
        left  = int(sys.argv[1])
        right = int(sys.argv[2])
        just  = sys.argv[3].upper()

              # Simplistic approach
              for p in sys.stdin.rea
                  print reformat_par
```

A number of enhancements are left to readers,
indents or indented first lines, for example. Or
be appropriate for wrapping (e.g., headers). A

input paragraphs differently, either by a differe
paragraphs internally in some manner.

2.1.3 Problem: Column statistics for delimite

Data feeds, DBMS dumps, log files, and flat-file
similar recordsone per linewith a collection of fi
separated either by a specified delimiter or by s
occur.

Parsing these structured text records is quite ea
equally straightforward. But in working with a v
is easy to keep writing almost the same code o
computation.

The example below provides a generic framewo
structured text database.

fields_stats.py

```
# Perform calculations on one or mor
# fields in a structured text databa

import operator
from types import *
```

```
from xreadlines import xreadlines #
                                  #
#-- Symbolic Constants
DELIMITED = 1
FLATFILE = 2

#-- Some sample "statistical" func (
nillFunc = lambda lst: None
toFloat = lambda lst: map(float, lst
avg_lst = lambda lst: reduce(operato
sum_lst = lambda lst: reduce(operato
max_lst = lambda lst: reduce(max, to

class FieldStats:
    """Gather statistics about struc
text_db may be either string (incl.
style may be in (DELIMITED, FLATFILE
delimiter specifies the field separa
column_positions lists all field pos
                 using one-based inc
         E.g.:  (1, 7, 40) would ta
                 from columns 1, 7,
field_funcs is a dictionary with col
            and functions on lists a
     E.g.:  {1:avg_lst, 4:sum_lst, 5
```

```
            average of column one, t
            max of column 5.  All ot
            are ignored.

    """
    def __init__(self,
                 text_db='',
                 style=DELIMITED,
                 delimiter=',',
                 column_positions=(1,),
                 field_funcs={} ):
        self.text_db = text_db
        self.style = style
        self.delimiter = delimiter
        self.column_positions = column_p
        self.field_funcs = field_funcs

    def calc(self):
        """Calculate the column statisti
        """
        #-- 1st, create a list of lists
        used_cols = self.field_funcs.key
        used_cols.sort()
        # one-based column naming: colum
        columns = []
```

```
for n in range(1+used_cols[-1]):
    # hint: '[[]]*num' creates r
    columns.append([])

      #-- 2nd, fill lists used f
              # might use a stri
      if type(self.text_db) in (
          for line in self.text_
              fields = self.spli
              for col in used_cc
                  field = fields
                    columns[col]
        else:   # Something file
            for line in xreadlir
                fields = self.sp
                for col in used_
                    field = fiel
                    columns[col]

        #-- 3rd, apply the fielc
        results = [None] * (1+us
        for col in used_cols:
            results[col] = \
                 apply(self.fiel
```

```
        #-- Finally, return the
        return results

    def splitter(self, line):
        """Split a line into fields
        if self.style == DELIMITED:
            return line.split(self.d
        elif self.style == FLATFILE:
            fields = []
            # Adjust offsets to Pyth
            # and also add final pos
            num_positions = len(self
            offsets = [(pos-1) for p
            offsets.append(len(line)
            for pos in range(num_pos
                start = offsets[pos]
                end = offsets[pos+1]
                fields.append(line[s
            return fields
        else:
            raise ValueError, \
                  "Text database mus

#-- Test data
# First Name, Last Name, Salary, Yea
```

```
delim = '''
Kevin,Smith,50000,5,Media Relations
Tom,Woo,30000,7,Accounting
Sally,Jones,62000,10,Management
'''.strip()     # no leading/trailir

# Comment      First     Last      Sa
flat = '''
tech note      Kevin     Smith     5(
more filler    Tom       Woo       3(
yet more...    Sally     Jones     62
'''.strip()      # no leading/trailir

#-- Run self-test code
if __name__ == '__main__':
    getdelim = FieldStats(delim, fie
    print 'Delimited Calculations:'
    results = getdelim.calc()
    print '  Average salary -', resu
    print '  Max years worked -', re

    getflat = FieldStats(flat, fielc
                                 style
                                 colum
    print 'Flat Calculations:'
```

```
results = getflat.calc()
print '  Average salary -', resu
print '  Max years worked -', re
```

The example above includes some efficiency co
working with large data sets. In the first place,
file-like object, rather than keeping the whole s
generator *xreadlines.xreadlines()* is an extreme
Python 2.1+otherwise use *FILE.readline()* or *FI*
efficiency, respectively). Moreover, only the dat
lists, in order to save memory. However, rather
statistics on multiple fields, as many field colun
used in one pass.

One possible improvement would be to allow m
field during a pass. But that is left as an exercis

2.1.4 Problem: Counting characters, words, li

There is a wonderful utility under Unix-like syst
so obvious, that it is hard to imagine working w
words, and lines of files (or STDIN). A few com
displayed, but I rarely use them.

In writing this chapter, I found myself on a syst
order. The example below is actually an "enhan
lacks the command-line switches). Unlike the e

directly within Python and is available anywher
is oneis a compact use of the *"".join()* and *"".sp*
could also be used, for example, to be compatil

wc.py

```
# Report the chars, words, lines, pa
# on STDIN or in wildcard filename p
import sys, glob
if len(sys.argv) > 1:
    c, w, l, p = 0, 0, 0, 0
    for pat in sys.argv[1:]:
        for file in glob.glob(pat):
            s = open(file).read()
            wc = len(s), len(s.split
                 len(s.split('\n')),
            print '\t'.join(map(str,
            c, w, l, p = c+wc[0], w+
    wc = (c,w,l,p)
    print '\t'.join(map(str, wc)), '
else:
    s = sys.stdin.read()
    wc = len(s), len(s.split()), len
         len(s.split('\n\n'))
```

```
    print '\t'.join(map(str, wc)), '
```

This little functionality could be wrapped up in a
bother with doing so. Most of the work is in the
the counting basically taking only two lines.

The solution above is quite likely the "one obvi
the other hand a slightly more adventurous rea
fun):

```
>>> wc  = map(len,[s]+map(s.split,(N
```

A real daredevil might be able to reduce the en

2.1.5 Problem: Transmitting binary data as AS

Many channels require that the information tha
with a high-order first bit of one will be handlec
protocols like Simple Mail Transport Protocol (S
(NNTP), or HTTP (depending on content encodi
many standard tools like editors. In order to en
techniques have been invented over time.

An obvious, but obese, encoding technique is t
hexadecimal digits. UUencoding is an older star
transmit binary files over the Usenet and on BE
MacOS world. In recent years, base64which is

styles of encoding. All of the techniques are bas
are used to represent three binary bytesbut the
conventions (as well as in the encoding as such
of variable encoding length. In quoted printable
unchanged, but a few special characters and al

Python provides modules for all the encoding st
binhex, base64, and *quopri* all operate on input
data therein. They also each have slightly differ
for example, closes its output file after encodin
a *cStringIO* file-like object. All of the high-level
module *binascii. binascii*, in turn, implements tl
assumes that it will be passed the right size blo

The standard library, therefore, does not contai
functionality for when the goal is just encoding
to wrap that up, though:

encode_binary.py

```
# Provide encoders for arbitrary bin
# in Python strings.  Handles block
# transparently, and returns a strin
# Precompression of the input string
# or eliminate any size penalty for
```

```
import sys
import zlib
import binascii

UU = 45
BASE64 = 57
BINHEX = sys.maxint

def ASCIIencode(s='', type=BASE64, c
    """ASCII encode a binary string"
    # First, decide the encoding sty
    if type == BASE64:   encode = bi
    elif type == UU:     encode = bi
    elif type == BINHEX: encode = bi
    else: raise ValueError, "Encodir
    # Second, compress the source if
    if compress: s = zlib.compress(s
    # Third, encode the string, bloc
    offset = 0
    blocks = []
    while 1:
        blocks.append(encode(s[offse
        offset += type
        if offset > len(s):
            break
```

```
    # Fourth, return the concatenate
    return ''.join(blocks)

def ASCIIdecode(s='', type=BASE64, c
    """Decode ASCII to a binary stri
    # First, decide the encoding sty
    if type == BASE64:    s = binasci
    elif type == BINHEX: s = binasci
    elif type == UU:
        s = ''.join([binascii.a2b_uu
    # Second, decompress the source
    if compress: s = zlib.decompress
    # Third, return the decoded bina
    return s

# Encode/decode STDIN for self-test
if __name__ == '__main__':
    decode, TYPE = 0, BASE64
    for arg in sys.argv:
        if   arg.lower()=='-d': deco
        elif arg.upper()=='UU': TYPE
        elif arg.upper()=='BINHEX':
        elif arg.upper()=='BASE64':
    if decode:
        print ASCIIdecode(sys.stdin.
```

```
    else:
        print ASCIIencode(sys.stdin.
```

The example above does not attach any header
for that, a wrapper like *uu, mimify*, or *MimeWri*
around encode_binary.py.

2.1.6 Problem: Creating word or letter histogr

A histogram is an analysis of the relative occurr
possible values. In terms of text processing, th
either words or byte values. Creating histogram
but the technique is not always immediately ob
below has a good generality, provides several u
and can be used in a command-line operation r

histogram.py

```
# Create occurrence counts of words
# A few utility functions for preser
# Avoids requirement of recent Pytho

from string import split, maketrans,
import sys
from types import *
```

```
import types

def word_histogram(source):
    """Create histogram of normalize
    hist = {}
    trans = maketrans('','')
    if type(source) in (StringType,U
        for word in split(source):
            word = translate(word, t
            if len(word) > 0:
                hist[word] = hist.ge
    elif hasattr(source,'read'):
        try:
            from xreadlines import x
            for line in xreadlines(s
                for word in split(li
                    word = translate
                    if len(word) > 0
                        hist[word] =
        except ImportError:
            line = source.readline()
            while line:
                for word in split(li
                    word = translate
                    if len(word) > 0
```

```
                    hist[word] =
                line = source.readli
    else:
        raise TypeError, \
            "source must be a stri
    return hist

def char_histogram(source, sizehint=
    hist = {}
    if type(source) in (StringType,U
        for char in source:
            hist[char] = hist.get(ch
    elif hasattr(source,'read'):
        chunk = source.read(sizehint
        while chunk:
            for char in chunk:
                hist[char] = hist.ge
            chunk = source.read(size
    else:
        raise TypeError, \
            "source must be a stri
    return hist

def most_common(hist, num=1):
    pairs = []
```

```
    for pair in hist.items():
        pairs.append((pair[1],pair[0
    pairs.sort()
    pairs.reverse()
    return pairs[:num]

def first_things(hist, num=1):
    pairs = []
    things = hist.keys()
    things.sort()
    for thing in things:
        pairs.append((thing,hist[thi
    pairs.sort()
    return pairs[:num]

if __name__ == '__main__':
    if len(sys.argv) > 1:
        hist = word_histogram(open(s
    else:
        hist = word_histogram(sys.st

    print "Ten most common words:"
    for pair in most_common(hist, 10
        print '\t', pair[1], pair[0]
```

```
print "First ten words alphabeti
for pair in first_things(hist, 1
    print '\t', pair[0], pair[1]

# a more practical command-line 
# for pair in most_common(hist,l
#     print pair[1],'\t',pair[0]
```

Several of the design choices are somewhat arb
stripped to identify "real" words. But on the oth
may not be what is desired. The sorting functio
return an initial sublist. Perhaps it would be bet
slice the result. It is simple to customize aroun

2.1.7 Problem: Reading a file backwards by re

Reading a file line by line is a common task in P
server logs, configuration files, structured text
information into logical records, one per line. Ve
some calculation on each record in turn.

Python provides a number of convenient metho
reading. *FILE.readlines()* reads a whole file at o
is very fast, but requires the whole contents of
files, this can be a problem. *FILE.readline()* is r
and can be called repeatedly until the EOF is re

solution for recent Python versions is *xreadline*
2.1+. These techniques are memory-friendly, w
list" of lines (by way of Python's new generator

The above techniques work nicely for reading a
to start at the end of a file and work backwards
encountered when you want to read log files th
when you want to look at the most recent recor
There is a very easy technique if memory usag

```
>>> open('lines','w').write('\n'.joi
>>> fp = open('lines')
>>> lines = fp.readlines()
>>> lines.reverse()
>>> for line in lines [1:5]:
...     # Processing suite here
...     print line,
...
98
97
96
95
```

For large input files, however, this technique is
something analogous to *xreadlines* here. The e
(the example works equally well for file-like ob

read_backwards.py

```
# Read blocks of a file from end to
# Blocks may be defined by any delim
#  constants LINE and PARA are usefu
# Works much like the file object me
#  repeated calls continue to get "r
#  function returns empty string onc

# Define constants
from os import linesep
LINE = linesep
PARA = linesep*2
READSIZE = 1000

# Global variables
buffer = ''

def read_backwards(fp, mode=LINE, si
    """Read blocks of file backwards
    # Trick of mutable default argum
    if not _init[0]:
        fp.seek(0,2)
        _init[0] = 1
```

```
    # Find a block (using global buf
    global buffer
    while 1:
        # first check for block in b
        delim = buffer.rfind(mode)
        if delim <> -1:     # block
            block = buffer[delim+len
            buffer = buffer[:delim]
            return block+mode
        #-- BOF reached, return rema
        elif fp.tell()==0:
            block = buffer
            buffer = ''
            return block
        else:               # Read some
            readsize = min(fp.tell()
            fp.seek(-readsize,1)
            buffer = fp.read(readsiz
            fp.seek(-readsize,1)
#-- Self test of read_backwards()
if __name__ == '__main__':
    # Let's create a test file to re
    fp = open('lines','wb')
    fp.write(LINE.join(['--- %d ---'
    # Now open for reading backwards
```

```
fp = open('lines','rb')
# Read the blocks in, one per ca
block = read_backwards(fp)
while block:
    print block,
    block = read_backwards(fp)
```

Notice that *anything* could serve as a block deli
to work for lines and block paragraphs (and blo
of line breaks). But other delimiters could be us
read backwards word-by-worda space delimiter
right for other whitespace. However, reading a
generally good enough.

Another enhancement is possible with Python 2
read_backwards() could be programmed as an
The performance will not differ significantly, bu
(and a "list-like" interface like *FILE.readlines()*

QUESTIONS

1: Write a generator-based version of read_bac
the self-test code to utilize the generator ins

2: Explore and explain some pitfalls with the us
argument. Explain also how the style allows
with the encapsulation of class instances.

2.2 Standard Modules

2.2.1 Basic String Transformations

The module *string* forms the core of Python's te
certainly the place to look before other module
you should note, have been copied to methods
methods of string objects are a little bit faster t
functions. A few new methods of string objects
but are still documented here.

SEE ALSO: str *33;* UserString *33*;

string • A collection of string operations

There are a number of general things to notice
(which is composed entirely of functions and cc

1. **Strings are immutable (as discussed in such thing as changing a string "in plac languages, such as C, by changing the b Whenever a *string* module function take returns a brand-new string object and l very common pattern of binding the sar was passed on the right side within the conceals this fact. For example:**

```
>>> import string
>>> str = "Mary had a little lamb'
>>> str = string.replace(str, 'ha
>>> str
'Mary ate a little lamb'
```

The first string object never gets modifi is no longer bound to any name after th garbage collection and will disappear fr module function will not change any exi make it look like they changed.

- Many *string* module functions are now also av string object methods, there is no need to imp usually slightly more concise. Moreover, using a than the corresponding *string* module function. of each function/method that exists as both a s method is contained in this reference to the *str*

- The form string.join(string.split (...)) is a freq
discussion is contained in the reference items fo
general, combining these two functions is very
processing the parts, then putting together the

- Think about clever *string.replace()* patterns. B
with use of "place holder" string patterns, a sur
(especially when also manipulating the interme
reference item for *string.replace()* for some dis

- A mutable string of sorts can be obtained by u
can contain a collection of substrings, each one
individually. The *array* module can define arrays
modifiable, included with slice notation. The fur
be used to re-create true strings; for example:

```
>>> lst = ['spam','and','eggs']
>>> lst[2] = 'toast'
>>> print ''.join(lst)
spamandtoast
>>> print ' '.join(lst)
spam and toast
```

Or:

```
>>> import array
>>> a = array.array('c','spam and eg
```

```
>>> print ''.join(a)
spam and eggs
>>> a[0] = 'S'
>>> print ''.join(a)
Spam and eggs
>>> a[-4:] = array.array('c','toast'
>>> print ''.join(a)
Spam and toast
```

CONSTANTS

The *string* module contains constants for a num
characters. Each of these constants is itself sim
collection). As such, it is easy to define constar
module, should you need them. For example:

```
>>> import string
>>> string.brackets = "[]{}()<>"
>>> print string.brackets
[]{}()<>
```

string.digits

The decimal numerals ("0123456789").

string.hexdigits

The hexadecimal numerals ("0123456789abcde

string.octdigits

The octal numerals ("01234567").

string.lowercase

The lowercase letters; can vary by language. Ir

```
>>> import string
>>> string.lowercase
'abcdefghijklmnopqrstuvwxyz'
```

You should not modify *string.lowercase* for a sc
attribute, such as string.spanish_lowercase wit
depend on this constant).

string.uppercase

The uppercase letters; can vary by language. I

```
>>> import string
>>> string.uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

You should not modify *string.uppercase* for a sc
attribute, such as string.spanish_uppercase wit
depend on this constant).

string.letters

All the letters (string.lowercase+string.upperca

string.punctuation

The characters normally considered as punctua
versions of Python (most systems):

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_'{|}~'
```

string.whitespace

The "empty" characters. Normally these consis
carriage return, and space (in that order):

```
>>> import string
>>> string.whitespace
'\011\012\013\014\015 '
```

You should not modify *string.whitespace* (some

string.printable

All the characters that can be printed to any de
(string.digits+string.letters+string.punctuation

FUNCTIONS

string.atof(s=...)

Deprecated. Use *float()*.

Converts a string to a floating point value.

SEE ALSO: eval() *445*; float() *422*;

string.atoi(s=...[,base=10])

Deprecated with Python 2.0. Use *int()* if no cus

Converts a string to an integer value (if the stri
than 10, the base may be specified as the seco

SEE ALSO: eval() *445*; int() *421*; long() *422*;

string.atol(s=...[,base=10])

Deprecated with Python 2.0. Use *long()* if no cu

Converts a string to an unlimited length intege
in a base other than 10, the base may be speci

SEE ALSO: eval() *445*; long() *422*; int() *421*;

string.capitalize(s=...)
"".capitalize()

Return a string consisting of the initial characte
all other characters converted to lowercase (if a

```
>>> import string
```

```
>>> string.capitalize("mary had a li
'Mary had a little lamb!'
>>> string.capitalize("Mary had a Li
'Mary had a little lamb!'
>>> string.capitalize("2 Lambs had M
'2 lambs had mary!'
```

For Python 1.6+, use of a string object method
preferred in most cases:

```
>>> "mary had a little lamb".capital
'Mary had a little lamb'
```

SEE ALSO: string.capwords() *133*; string.lower

string.capwords(s=...)
"".title()

Return a string consisting of the capitalized wor

```
string.join(map(string.capitalize,st
```

But *string.capwords()* is a clearer way of writin
whitespace is "normalized" by the process:

```
>>> import string
>>> string.capwords("mary HAD a litt
'Mary Had A Little Lamb!'
>>> string.capwords("Mary      had a
'Mary Had A Little Lamb!'
```

With the creation of string methods in Python 1
renamed as a string method to *"".title()*.

SEE ALSO: string.capitalize() *132*; string.lower

string.center(s=. . . , width=...)
"".center(width)

Return a string with s padded with symmetrical
truncated) to occupy length width (or more).

```
>>> import string
>>> string.center(width=30,s="Mary h
'    Mary had a little lamb '
>>> string.center("Mary had a little
'Mary had a little lamb'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a little lamb".center(
'  Mary had a little lamb '
```

SEE ALSO: string.ljust() *138;* string.rjust() *141*

string.count(s, sub [,start [,end]])
"".count(sub [,start [,end]])

Return the number of nonoverlapping occurren
arguments are specified, only the correspondin

```
>>> import string
>>> string.count("mary had a little
4
>>> string.count("mary had a little
2
```

For Python 1.6+, use of a string object method

```
>>> 'mary had a little lamb'.count("
4
```

"".endswith(suffix [,start [,end]])

This string method does not have an equivalent
indicating whether the string ends with the suff
start is specified, only consider the terminal su
argument end is given, only consider the slice [

SEE ALSO: "".startswith() *144*; string.find() *13*

string.expandtabs(s=...[,tabsize=8])
"".expandtabs([,tabsize=8])

Return a string with tabs replaced by a variable
text blocks to line up at "tab stops." If no secon
up at multiples of 8 spaces. A newline implies a

```
>>> import string
>>> s = 'mary\011had a little lamb'
>>> print s
mary    had a little lamb
>>> string.expandtabs(s, 16)
'mary            had a little lamb'
>>> string.expandtabs(tabsize=1, s=s
'mary had a little lamb'
```

For Python 1.6+, use of a string object method

```
>>> 'mary\011had a little lamb'.expa
'mary                    had a litt
```

string.find(s, sub [,start [,end]])
"".find(sub [,start [,end]])

Return the index position of the first occurrence
arguments are specified, only the correspondin
in s as a whole). Return -1 if no occurrence is f
list indexing:

```
>>> import string
>>> string.find("mary had a little l
1
>>> string.find("mary had a little l
6
>>> string.find("mary had a little l
21
>>> string.find("mary had a little l
-1
```

For Python 1.6+, use of a string object method

```
>>> 'mary had a little lamb'.find("a
6
```

SEE ALSO: string.index() *135;* string.rfind() *14*

string.index(s, sub [,start [,end]])
"".index(sub [,start [,end]])

Return the same value as does *string.find()* wit
instead of returning -1 when sub does not occu

```
>>> import string
>>> string.index("mary had a little
21
>>> string.index("mary had a little
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "d:/py20sl/lib/string.py", li
    return s.index(*args)
ValueError: substring not found in s
```

For Python 1.6+, use of a string object method

```
>>> 'mary had a little lamb'.index("
6
```

SEE ALSO: string.find() *135;* string.rindex() *14*

Several string methods that return Boolean valu
property. None of the .is*() methods, however,

"".isalpha()

Return a true value if all the characters are alpl

"".isalnum()

Return a true value if all the characters are alpl

"".isdigit()

Return a true value if all the characters are digi

"".islower()

Return a true value if all the characters are low
character:

```
>>> "ab123".islower(), '123'.islower
(1, 0, 0)
```

SEE ALSO: "".lower() *138;*

"".isspace()

Return a true value if all the characters are whi

"".istitle()

Return a true value if all the string has title cas

SEE ALSO: "".title() *133;*

"".isupper()

Return a true value if all the characters are upp
character.

SEE ALSO: "".upper() *146;*

string.join(words=...[,sep=" "])
"".join (words)

Return a string that results from concatenating

sep between each. The function *string.join()* dif
that it takes a list (of strings) as a primary argu

It is worth noting *string.join()* and *string.split()*
both; in other words, string.join(string.split(s,s

Typically, *string.join()* is used in contexts where
example, here is a small program to output the
STDOUT, one per line:

list_capwords.py

```
import string,sys
capwords = []

for line in sys.stdin.readlines():
    for word in line.split():
        if word == word.upper() and
            capwords.append(word)
print string.join(capwords, '\n')
```

The technique in the sample list_capwords.py s
building up a string by direct concatenation. Ho
reduces the performance difference:

```
>>> import string
```

```
>>> s = "Mary had a little lamb"
>>> t = "its fleece was white as snc
>>> s = s +" "+ t     # relatively "e
>>> s += " " + t      # "cheaper" tha
>>> lst = [s]
>>> lst.append(t)     # "cheapest" wa
>>> s = string.join(lst)
```

For Python 1.6+, use of a string object method
However, just as *string.join()* is special in takin
method *"".join()* is unusual in being an operatio
(required) words list (this surprises many new

SEE ALSO: string.split() *142;*

string.joinfields(...)

Identical to *string.join().*

string.ljust(s=..., width=...)
"".ljust(width)

Return a string with s padded with trailing spac
(or more).

```
>>> import string
>>> string.ljust(width=30,s="Mary ha
'Mary had a little lamb         '
>>> string.ljust("Mary had a little 
'Mary had a little lamb'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a little lamb".ljust(2
'Mary had a little lamb    '
```

SEE ALSO: string.rjust() *141;* string.center() *1.*

string.lower(s=...)
"".lower()

Return a string with any uppercase letters conv

```
>>> import string
>>> string.lower("mary HAD a little 
'mary had a little lamb!'
>>> string.lower("Mary had a Little 
'mary had a little lamb!'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a Little Lamb!".lower(
'mary had a little lamb!'
```

SEE ALSO: string.upper() *146;*

string.lstrip(s=...)
"".lstrip([chars=string.whitespace])

Return a string with leading whitespace charact
object method is stylistically preferred in many

```
>>> import string
>>> s = """
...      Mary had a little lamb
>>> string.lstrip(s)
'Mary had a little lamb      \011'
>>> s.lstrip()
'Mary had a little lamb      \011'
```

Python 2.3+ accepts the optional argument cha
in the string chars will be removed.

SEE ALSO: string.rstrip() *142;* string.strip() *14*

string.maketrans(from, to)

Return a translation table string for use with *st*
be the same length. A translation table is a stri
position defines a translation from the *chr()* val
that index position.

```
>>> import string
>>> ord('A')
65
>>> ord('z')
122
>>> string.maketrans('ABC','abc')[65
'abcDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_'ab
>>> string.maketrans('ABCxyz','abcXY
'abcDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_'ab
```

SEE ALSO: string.translate() *145;*

string.replace(s=..., old=..., new=...[,maxsplit=
"".replace(old, new [,maxsplit])

Return a string based on s with occurrences of
maxsplit is specified, only replace maxsplit initi

```
>>> import string
>>> string.replace("Mary had a littl
```

```
'Mary had some lamb'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a little lamb".replace
'Mary had some lamb'
```

A common "trick" involving *string.replace()* is t
Obviously, simply to replace several different su
operations are almost inevitable. But there is a
can be used to create an intermediate string wi
particular context. The same goal can always b
sometimes staged *string.replace()* operations a

```
>>> import string
>>> line = 'variable = val      # se
>>> # we'd like '#3' and '#4' spelle
>>> string.replace(line,'#','number
'variable = val      number  see com
>>> place_holder=string.replace(line
>>> place_holder
'variable = val      !!! see comment
>>> place_holder=place_holder.replac
>>> place_holder
'variable = val      !!! see comment
>>> line = string.replace(place_hold
```

```
>>> line
'variable = val      # see comments
```

Obviously, for jobs like this, a placeholder must
strings undergoing "staged transformation"; bu
placeholders may be as long as needed.

SEE ALSO: string.translate() *145;* mx.TextTools

string.rfind(s, sub [,start [,end]])
"".rfind(sub [,start [,end]])

Return the index position of the last occurrence
arguments are specified, only the correspondin
in s as a whole). Return -1 if no occurrence is f
list indexing:

```
>>> import string
>>> string.rfind("mary had a little
19
>>> string.rfind("mary had a little
9
>>> string.rfind("mary had a little
21
>>> string.rfind("mary had a little
```

```
-1
```

For Python 1.6+, use of a string object method

```
>>> 'mary had a little lamb'.rfind('
6
```

SEE ALSO: string.rindex() *141;* string.find() *13*

string.rindex(s, sub [,start [,end]])
"".rindex(sub [,start [,end]])

Return the same value as does *string.rfind()* wi
instead of returning -1 when sub does not occu

```
>>> import string
>>> string.rindex("mary had a little
21
>>> string.rindex("mary had a little
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "d:/py20sl/lib/string.py", li
    return s.rindex(*args)
ValueError: substring not found in s
```

For Python 1.6+, use of a string object method

```
>>> 'mary had a little lamb'.index("
6
```

SEE ALSO: string.rfind() *140;* string.index() *13*

string.rjust(s=..., width=...)
"".rjust(width)

Return a string with s padded with leading spac
(or more).

```
>>> import string
>>> string.rjust(width=30,s="Mary ha
'        Mary had a little lamb'
>>> string.rjust("Mary had a little
'Mary had a little lamb'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a little lamb".rjust(2
'    Mary had a little lamb'
```

SEE ALSO: string.ljust() *138;* string.center() *13*

string.rstrip(s=...)
"".rstrip([chars=string.whitespace])

Return a string with trailing whitespace charact
object method is stylistically preferred in many

```
>>> import string
>>> s = """
...      Mary had a little lamb
>>> string.rstrip(s)
'\012    Mary had a little lamb'
>>> s.rstrip()
'\012    Mary had a little lamb'
```

Python 2.3+ accepts the optional argument cha
in the string chars will be removed.

SEE ALSO: string.lstrip() *139;* string.strip() *14*

string.split(s=...[,sep=...[,maxsplit=...]])
"".split([,sep [,maxsplit]])

Return a list of nonoverlapping substrings of s.
substrings are divided around the occurrences
are divided around *any* whitespace characters.

resultant list. If the third argument maxsplit is
maxsplit parts is appended to the list, giving th

```
>>> import string
>>> s = 'mary had a little lamb    .
>>> string.split(s, ' a ')
['mary had', 'little lamb     ...wit
>>> string.split(s)
['mary', 'had', 'a', 'little', 'lamb
'of', 'sherry']
>>> string.split(s,maxsplit=5)
['mary', 'had', 'a', 'little', 'lamb
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a Little Lamb!".split(
['Mary', 'had', 'a', 'Little', 'Lamb
```

The *string.split()* function (and corresponding s
for working with texts, especially ones that rese
all whitespace as a single divider allows *string.s*

```
>>> wc = lambda s: len(s.split())
>>> wc("Mary had a Little Lamb")
5
>>> s = """Mary had a Little Lamb
```

```
... its fleece as white as snow.
... And everywhere that Mary went  .
>>> print s
Mary had a Little Lamb
its fleece as white as snow.
And everywhere that Mary went   ...
>>> wc(s)
23
```

The function *string.split()* is very often used in
involved is "pull the string apart, modify the pa
be words, but this also works with lines (dividir

```
>>> import string
>>> s = """Mary had a Little Lamb
... its fleece as white as snow.
... And everywhere that Mary went
>>> string.join(string.split(s))
'Mary had a Little Lamb its fleece a
... that Mary went the lamb was sure
```

A Python 1.6+ idiom for string object methods

```
>>> "-".join(s.split())
'Mary-had-a-Little-Lamb-its-fleece-a
...-that-Mary-went--the-lamb-was-sur
```

SEE ALSO: string.join() *137;* mx.TextTools.sets
mx.TextTools.splitat() *315;* mx.TextTools.splitli

string.splitfields(...)

Identical to *string.split()*.

"".splitlines([keepends=0])

This string method does not have an equivalent
the string. The optional argument keepends de
included in the line strings.

"".startswith(prefix [,start [,end]])

This string method does not have an equivalent
indicating whether the string begins with the p
start is specified, only consider the terminal su
third argument end is given, only consider the

SEE ALSO: "".endswith() *134;* string.find() *135*

string.strip(s=...)

"".strip([chars=string.whitespace])

Return a string with leading and trailing whites
use of a string object method is stylistically pre

```
>>> import string
>>> s = """
...     Mary had a little lamb      \
>>> string.strip(s)
'Mary had a little lamb'
>>> s.strip()
'Mary had a little lamb'
```

Python 2.3+ accepts the optional argument cha
in the string chars will be removed.

```
>>> s = "MARY had a LITTLE lamb STEW
>>> s.strip("ABCDEFGHIJKLMNOPQRSTUVW
' had a LITTLE lamb '
```

SEE ALSO: string.rstrip() *142;* string.lstrip() *13*

string.swapcase(s=...)
"".swapcase()

Return a string with any uppercase letters conv
converted to uppercase.

```
>>> import string
>>> string.swapcase("mary HAD a litt
'MARY had A LITTLE LAMB!'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a Little Lamb!".swapca
'MARY had A LITTLE LAMB!'
```

SEE ALSO: string.upper() *146;* string.lower() *1*

string.translate(s=..., table=...[,deletechars=""
"".translate(table [,deletechars=""])

Return a string, based on s, with deletechars d
with any remaining characters translated accor

```
>>> import string
>>> tab = string.maketrans('ABC','ak
>>> string.translate('MARY HAD a lit
'MRY HD a ie LMb'
```

For Python 1.6+, use of a string object method
However, if *string.maketrans()* is used to create
the *string* module anyway:

```
>>> 'MARY HAD a little LAMB'.transla
'MRY HD a ie LMb'
```

The *string.translate()* function is a *very* fast wa
table takes some getting used to, but the resul
procedural technique such as:

```
>>> (new,frm,to,dlt) = ("",'ABC','ab
>>> for c in 'MARY HAD a little LAMB
...     if c not in dlt:
...         pos = frm.find(c)
...         if pos == -1: new += c
...         else:         new += to[
...
>>> new
'MRY HD a ie LMb'
```

SEE ALSO: string.maketrans() *139;*

string.upper(s=...)
"".upper()

Return a string with any lowercase letters conv

```
>>> import string
>>> string.upper("mary HAD a little
'MARY HAD A LITTLE LAMB!'
>>> string.upper("Mary had a Little
'MARY HAD A LITTLE LAMB!'
```

For Python 1.6+, use of a string object method

```
>>> "Mary had a Little Lamb!".upper(
'MARY HAD A LITTLE LAMB!'
```

SEE ALSO: string.lower() *138;*

string.zfill(s=..., width=...)

Return a string with s padded with leading zero (or more). If a leading sign is present, it "floats general, *string.zfill()* is designed for alignment see if a string looks number-like.

```
>>> import string
>>> string.zfill("this", 20)
'0000000000000000this'
```

```
>>> string.zfill("-37", 20)
'-0000000000000000037'
>>> string.zfill("+3.7", 20)
'+00000000000000003.7'
```

Based on the example of *string.rjust()*, one mig
however, no such method exists.

SEE ALSO: string.rjust() *141;*

2.2.2 Strings as Files, and Files as Strings

In many ways, strings and files do a similar job
unlimited amount of (textual) information that
the bytes. A first inclination is to suppose that t
of persistencefiles hang around when the curre
distinction is not really tenable. On the one har
pickle, and *marshal*and third-party modules like
making strings persist (but not thereby corresp
other hand, many files are not particularly pers
under Unix-like systems exist only for program
similar "device files" are really just streams; ar
disks, or get deleted with program cleanup, are

The real difference between files and strings in
techniques available to operate on them. File ol
on themselves. Notably, file objects have a con

imaginary "read-head" passing over the physica
can be sliced and indexedfor example, str[4:10
string object methods and by functions of modu
special-purpose Python objects act "file-like" wi
gzip.open() and *urllib.urlopen()* . Of course, Py
for just how "file-like" something has to be to w
to figure that out for each type of object she wi
time things "just work" right).

Happily, Python provides some standard modul
interoperable.

mmap • Memory-mapped file support

The *mmap* module allows a programmer to cre
special *mmap* objects enable most of the techn
and simultaneously most of the techniques you
the hinted caveat about "most," however: Many
using the corresponding string object methods.
"string-like," it basically only implements the .f
associated with slicing and indexing. This is eno

When a string-like change is made to a *mmap*
underlying file, and the change is persistent (as
that the object called .flush() before destructio
to "persistent strings."

Some examples of working with memory-mapp

```
>>> # Create a file with some test c
>>> open('test','w').write(' #'.joir
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(),1000)
>>> len(mm)
1000
>>> mm[-20:]
'218 #219 #220 #221 #'
>>> import string   # apply a string
>>> mm.seek(string.find(mm, '21'))
>>> mm.read(10)
'21 #22 #23'
>>> mm.read(10)      # next ten bytes
' #24 #25 #'
>>> mm.find('21')    # object method
402
>>> try: string.rfind(mm, '21')
... except AttributeError: print "Ur
...
Unsupported string function
>>> '/'.join(re.findall('..21..',mm)
' #21 #/121 #/ #210 / #212 / #214 /
```

It is worth emphasizing that the bytes in a file c
mmap.mmap.resize() method to write into diffe
the file from the middle, only by adding to the e

CLASSES

mmap.mmap(fileno, length [,tagname]) (Wind
mmap.mmap(fileno, length [,flags=MAP_SHA

Create a new memory-mapped file object. filen
mapping on. Generally this number should be o
object. length specifies the length of the mappi
for length to specify the current length of the fi
specified, only the initial portion of the file will b
file is specified, the file can be extended with a

The underlying file for a memory-mapped file o
"+" mode modifier.

According to the official Python documentation
may be specified. If it is, multiple memory-map
practice, however, each instance of *mmap.mma*
not a tagname is specified. In any case, this all
underlying file, generally at different positions i

```
>>> open('test','w').write(' #'.join
```

```
>>> fp = open('test','r+')
>>> import mmap
>>> mm1 = mmap.mmap(fp.fileno(),100(
>>> mm2 = mmap.mmap(fp.fileno(),100(
>>> mm1.seek(500)
>>> mm1.read(10)
'122 #123 #'
>>> mm2.read(10)
'0 #1 #2 #3'
```

Under Unix, the third argument flags may be M
MAP_SHARED is specified for flags, all processe
to a *mmap* object. Otherwise, the changes are
argument, prot, may be used to disallow certai
mapped file regions.

METHODS

mmap.mmap.close()

Close the memory-mapped file object. Subsequ
object will raise an exception. Under Windows,
is somewhat erratic, however. Note that closing
same as closing the underlying file object. Clos
inaccessible, but closing the memory-mapped f

object.

SEE ALSO: FILE.close() *16;*

mmap.mmap.find(sub [,pos])

Similar to *string.find()* . Return the index positi
object. If the optional second argument pos is s
relative to pos. Return -1 if no occurrence is fou

```
>>> open('test','w').write(' #'.join
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(), 0)
>>> mm.find('21')
74
>>> mm.find('21',100)
-26
>>> mm.tell()
0
```

SEE ALSO: mmap.mmap.seek() *152;* string.fin

mmap.mmap.flush([offset, size])

Writes changes made in memory to *mmap* obje
second argument size must either both be spec
specified, only the position starting at offset or

mmap.mmap.flush() is necessary to guarantee
guarantee is given that changes *will not* be writ
interpreter housekeeping. *mmap* should not be
(since changes may not be cancelable).

SEE ALSO: FILE.flush() *16;*

mmap.mmap.move(target, source, length)

Copy a substring within a memory-mapped file
argument length. The target location is the first
from the position source. It is allowable to have
target range, but it must not go past the last p

```
>>> open('test','w').write(''.join([
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(),0)
>>> mm[:]
'AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDD
>>> mm.move(40,0,5)
>>> mm[:]
```

```
'AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDI
```

mmap.mmap.read(num)

Return a string containing num bytes, starting
moved to the end of the read string. In contras
mmap.mmap.read() always requires that a byt
map file object not fully substitutable for a file
following is safe for both true file objects and m

```
>>> open('test','w').write(' #'.joir
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(),0)
>>> def safe_readall(file):
...     try:
...         length = len(file)
...         return file.read(length)
...     except TypeError:
...         return file.read()
...
>>> s1 = safe_readall(fp)
>>> s2 = safe_readall(mm)
>>> s1 == s2
```

1

SEE ALSO: mmap.mmap.read_byte() *151;* mm
mmap.mmap.write() *153;* FILE.read() *17;*

mmap.mmap.read_byte()

Return a one-byte string from the current file p
one. Same as mmap.mmap.read (1).

SEE ALSO: mmap.mmap.read() *150;* mmap.mı

mmap.mmap.readline()

Return a string from the memory-mapped file c
and going to the next newline character. Advan
read.

SEE ALSO: mmap.mmap.read() *150;* mmap.mı

mmap.mmap.resize(newsize)

Change the size of a memory-mapped file obje
underlying file or merely to expand the area of

file is padded with null bytes (\000) unless othe
operations on *mmap* objects, changes to the ur
.flush() is performed.

SEE ALSO: mmap.mmap.flush() *150;*

mmap.mmap.seek(offset [,mode])

Change the current file position. If a second arg
can be selected. The default is 0, absolute file p
current file position. Mode 2 is relative to the er
smaller than the whole size of the underlying fi
distance to move the current file positionin moc
be negative, in mode 1 the current position car

SEE ALSO: FILE.seek() *17;*

mmap.mmap.size()

Return the length of the underlying file. The siz
less than the whole file is mapped:

```
>>> open('test','w').write('X'*100)
>>> fp = open('test','r+')
>>> import mmap
```

```
>>> mm = mmap.mmap(fp.fileno(),50)
>>> mm.size()
100
>>> len(mm)
50
```

SEE ALSO: len() *14;* mmap.mmap.seek() *152;*

mmap.mmap.tell()

Return the current file position.

```
>>> open('test','w').write('X'*100)
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(), 0)
>>> mm.tell()
0
>>> mm.seek(20)
>>> mm.tell()
20
>>> mm.read(20)
'XXXXXXXXXXXXXXXXXXXX'
>>> mm.tell()
```

```
40
```

SEE ALSO: FILE.tell() *17;* mmap.mmap.seek()

mmap.mmap.write(s)

Write s into the memory-mapped file object at
position is updated to the position following the
useful for functions that expect to be passed a
However, for new code, it is generally more nat
operations to write contents. For example:

```
>>> open('test','w').write('X'*50)
>>> fp = open('test','r+')
>>> import mmap
>>> mm = mmap.mmap(fp.fileno(), 0)
>>> mm.write('AAAAA')
>>> mm.seek(10)
>>> mm.write('BBBBB')
>>> mm[30:35] = 'SSSSS'
>>> mm[:]
'AAAAAXXXXXBBBBBXXXXXXXXXXXXXXXSSSSS
>>> mm.tell()
15
```

SEE ALSO: FILE.write() *17;* mmap.mmap.read(

mmap.mmap.write_byte(c)

Write a one-byte string to the current file positi
Same as mmap.mmap.write(c) where c is a on

SEE ALSO: mmap.mmap.write() *153;*

StringIO • File-like objects that read from

cStringIO • Fast, but incomplete, StringIO

The *StringIO* and *cStringIO* modules allow a pr
"string buffers." These special *StringIO* objects
apply to "true" file objects, but without any cor

The most common use of string buffer objects i
with byte-streams in files are to be applied to s
buffer object behaves in a file-like manner and
objects.

cStringIO is much faster than *StringIO* and sho
provide a StringIO class whose instances are th

cannot be subclassed (and therefore cannot pro
handle Unicode strings. One rarely needs to sul
support in *cStringIO* could be a problem for ma
support write operations, which makes its strin
against an in-memory file can be accomplished

A string buffer object may be initialized with a s
so, that is the initial content of the buffer. Belov
handling):

```
>>> from cStringIO import StringIO a
>>> from StringIO import StringIO as
>>> alef, omega = unichr(1488), unic
>>> sentence = "In set theory, the G
...              "ordinal limit of the
...              alef+" represents the
>>> sio = SIO(sentence)
>>> try:
...     csio = CSIO(sentence)
...     print "New string buffer fro
... except TypeError:
...     csio = CSIO(sentence.encode(
...     print "New string buffer fro
...
New string buffer from ENCODED strin
>>> sio.getvalue() == unicode(csio.g
```

```
1
>>> try:
...     sio.getvalue() == csio.getva
... except UnicodeError:
...     print "Cannot even compare U
...
Cannot even compare Unicode with str
>>> lines = csio.readlines()
>>> len(lines)
3
>>> sio.seek(0)
>>> print sio.readline().encode('utf
In set theory, the Greek  represents
>>> sio.tell(), csio.tell()
(51, 124)
```

CONSTANTS

cStringIO.InputType

The type of a *cStringIO.StringIO* instance that
StringIO.StringIO instances are simply Instance

SEE ALSO: cStringIO.StringIO *155;*

cStringIO.OutputType

The type of *cStringIO.StringIO* instance that ha
read/write). All *StringIO.StringIO* instances are

SEE ALSO: cStringIO.StringIO *155;*

CLASSES

StringIO.StringIO ([buf=...])
cStringIO.StringIO([buf])

Create a new string buffer. If the first argumen
string content. If the *cStringIO* module is used,
whether write access to the buffer is enabled. A
must be initialized with no argument, otherwise
buffer, however, is always read/write.

METHODS

StringIO.StringIO.close()
cStringIO.StringIO.close()

Close the string buffer. No access is permitted a

SEE ALSO: FILE.close() *16;*

StringIO.StringIO.flush()
cStringIO.StringIO.flush()

Compatibility method for file-like behavior. Data
there is no need to finalize a write to disk.

SEE ALSO: FILE.close() *16;*

StringIO.StringIO.getvalue()
cStringIO.StringIO.getvalue()

Return the entire string held by the string buffe
Basically, this is the way you convert back from

StringIO.StringIO.isatty()
cStringIO.StringIO.isatty()

Return 0. Compatibility method for file-like beh

SEE ALSO: FILE.isatty() *16;*

StringIO.StringIO.read ([num])
cStringIO.StringIO.read ([num])

If the first argument num is specified, return a
num characters are not available, return as ma
all the characters from current file position to e
position by the amount read.

SEE ALSO: FILE.read() *17;* mmap.mmap.read(

StringIO.StringIO.readline([length=...])
cStringIO.StringIO.readline([length])

Return a string from the string buffer, starting f
next newline character. Advance the current file

SEE ALSO: mmap.mmap.readline() *151;* String
StringIO.StringIO.readlines() *156;* FILE.readlin

StringIO.StringIO.readlines([sizehint=...])
cStringIO.StringIO.readlines([sizehint]

Return a list of strings from the string buffer. Ea
including the trailing newline character(s). If ar

approximately sizehint characters worth of lines

SEE ALSO: StringIO.StringIO.readline() *156;* FI

cStringIO.StringIO.reset()

Sets the current file position to the beginning o
cStringIO.StringIO.seek(0).

SEE ALSO: StringIO.StringIO.seek() *156;*

StringIO.StringIO.seek(offset [,mode=0])
cStringIO.StringIO.seek(offset [,mode])

Change the current file position. If the second a
mode can be selected. The default is 0, absolut
current file position. Mode 2 is relative to the er
offset specifies the distance to move the currer
in mode 2 it should be negative, in mode 1 the
or backward.

SEE ALSO: FILE.seek() *17*; mmap.mmap.seek(

StringIO.StringIO.tell()
cStringIO.StringIO.tell()

Return the current file position in the string buf

SEE ALSO: StringIO.StringIO.seek() *156;*

StringIO.StringIO.truncate([len=0])
cStringIO.StringIO.truncate ([len])

Reduce the length of the string buffer to the fir
only reduce characters later than the current fil
cStringIO.StringIO.reset() can be used to assur

SEE ALSO: StringIO.StringIO.seek() *156;* cStri
StringIO.StringIO.close() *155;*

StringIO.StringIO.write(s=...)
cStringIO.StringIO.write(s)

Write the first argument s into the string buffer
position is updated to the position following the

SEE ALSO: StringIO.StringIO.writelines() *157;*
StringIO.StringIO.read() *156;* FILE.write() *17;*

StringIO.StringIO.writelines(list=...)
cStringIO.String IO.writelines(list)

Write each element of list into the string buffer
position is updated to the position following the
an actual list. For the *StringIO* method, other s
best to coerce an argument into an actual list fi
strings, or a TypeError will occur.

Contrary to what might be expected from the n
never inserts newline characters. For the list el
string buffer, each element string must already
following variants on writing a list to a string bu

```
>>> from StringIO import StringIO
>>> sio = StringIO()
>>> lst = [c*5 for c in 'ABC']
>>> sio.writelines(lst)
>>> sio.write(''.join(lst))
>>> sio.write('\n'.join(lst))
>>> print sio.getvalue()
AAAAABBBBBCCCCCAAAAABBBBBCCCCCAAAAA
BBBBB
CCCCC
```

SEE ALSO: FILE.writelines() *17;* StringIO.String

2.2.3 Converting Between Binary and ASCII

The Python standard library provides several m
7-bit ASCII. At the low level, *binascii* is a C ext
high level, *base64, binhex, quopri*, and *uu* prov
binascii.

base64 • Convert to/from base64 encodin

The *base64* module is a wrapper around the fu
binascii.b2a-base64(). As well as providing a fil
string conversions, *base64* handles the chunkir
provides for the direct encoding of arbitrary inp
headers to encoded data; MIME standards for h
other modules that utilize *base64*. Base64 enco

FUNCTIONS

base64.encode(input=..., output=...)

Encode the contents of the first argument input
input and output should be file-like objects; inp
writable.

base64.encodestring(s=...)

Return the base64 encoding of the string passe

base64.decode(input=..., output=...)

Decode the contents of the first argument inpu
input and output should be file-like objects; inp
writable.

base64.decodestring(s=...)

Return the decoding of the base64-encoded str

SEE ALSO: email *345*; rfc822 *397*; mimetools *3*
396; binascii *159;* quopri *162*;

binascii • Convert between binary data an

The *binascii* module is a C implementation of a
data. Each function in the *binascii* module take
as an argument, and returns the string result o
apply to the length of strings passed to some fu
operate on specific block sizes).

FUNCTIONS

binascii.a2b_base64(s)

Return the decoded version of a base64-encode
encoding blocks should be passed as the argum

binascii.a2b_hex(s)

Return the decoded version of a hexadecimal-e
number of hexadecimals digits should be passe

binascii.a2b_hqx(s)

Return the decoded version of a binhex-encode
number of encoded binary bytes should be pass

binascii.a2b_qp(s [,header=0])

Return the decoded version of a quoted printab
number of encoded binary bytes should be pass
argument header is specified, underscores will

binascii.a2b_uu(s)

Return the decoded version of a UUencoded str
encoding block should be passed as the argume
returned).

binascii.b2a_base64(s)

Return the based64 encoding of a binary string
string no longer than 57 bytes should be passe

binascii.b2a_hex(s)

Return the hexadecimal encoding of a binary st
passed as the argument s.

binascii.b2a_hqx(s)

Return the binhex4 encoding of a binary string.
passed as the argument s. Run-length compres
(use *binascii.rlecode_hqx()* first, if needed).

binascii.b2a_qp(s [,quotetabs=0 [,istext=1 [he

Return the quoted printable encoding of a binar
be passed as the argument s. The optional argu
spaces and tabs; istext specifies *not* to newline
as underscores (and escape underscores). New

binascii.b2a_uu(s)

Return the UUencoding of a binary string (inclu
blocksand newline after block). A binary string
the argument s.

binascii.crc32(s [,crc])

Return the CRC32 checksum of the first argume
it will be used as an initial checksum. This allov
continuation. For example:

```
>>> import binascii
>>> crc = binascii.crc32('spam')
>>> binascii.crc32(' and eggs', crc)
739139840
>>> binascii.crc32('spam and eggs')
```

```
739139840
```

binascii.crc_hqx(s, crc)

Return the binhex4 checksum of the first argun
argument. This allows partial computation of a

```
>>> import binascii
>>> binascii.crc_hqx('spam and eggs'
17918
>>> crc = binascii.crc_hqx('spam', 0
>>> binascii.crc_hqx(' and eggs', cr
17918
```

SEE ALSO: binascii.crc32 *160;*

binascii.hexlify(s)

Identical to *binascii.b2a_hex()*.

binascii.rlecode_hqx(s)

Return the binhex4 run-length encoding (RLE)

0x90 is used as an indicator byte. Independent of precompression for encoded strings.

SEE ALSO: zlib.compress() *182;*

binascii.rledecode_hqx(s)

Return the expansion of a binhex4 run-length e

binascii.unhexlify(s)

Identical to *binascii.a2b_hex()*

EXCEPTIONS

binascii.Error

Generic exception that should only result from

binascii.Incomplete

Exception raised when a data block is incomple errors in reading blocks, but it could indicate da

SEE ALSO: base64 *158;* binhex *161;* uu *163;*

binhex • Encode and decode binhex4 files

The *binhex* module is a wrapper around the fur
binascii.rlecode_hqx(), binascii.rledecode_hqx(
a file-based interface on top of the underlying s
encoding of encoded files and attaches the nee
MacOS, the resource fork of a file is encoded al
other platforms).

FUNCTIONS

binhex.binhex(inp=..., out=...)

Encode the contents of the first argument inp t
filename; out may be either a filename or a file
object is not "file-like" enough since it will be cl
value lost. You could override the . close() meth
this limitation.

binhex.hexbin(inp=...[,out=...])

Decode the contents of the first argument to ar
specified, it will be used as the output filename
the binhex header. The argument inp may be e

CLASSES

A number of internal classes are used by *binhe*
examined in $PYTHONHOME/lib/binhex.py if de
this).

SEE ALSO: binascii *159;*

quopri • Convert to/from quoted printable

The *quopri* module is a wrapper around the fun
binascii.b2a_qp(). The module *quopri* has the s
adds no content headers to encoded data; MIM
wrapping are handled by other modules that ut
specified in RFC 1521.

FUNCTIONS

quopri.encode(input, output, quotetabs)

Encode the contents of the first argument input
input and output should be file-like objects; inp
writable. If quotetabs is a true value, escape ta

quopri.encodestring(s [,quotetabs=0])

Return the quoted printable encoding of the str
quotetabs is a true value, escape tabs and spac

quopri.decode(input=..., output=...[,header=0]

Decode the contents of the first argument inpu
input and output should be file-like objects; inp
writable. If header is a true value, encode spac

quopri.decodestring(s [,header=0])

Return the decoding of the quoted printable str
a true value, decode underscores as spaces.

SEE ALSO: email *345*; rfc822 *397*; mimetools 3

396; binascii *159;* base64 *158;*

uu • UUencode and UUdecode files

The *uu* module is a wrapper around the functio
well as providing a file-based interface on top c
handles the chunking of binary files into UUenc
header and footer.

FUNCTIONS

uu.encode(in, out, [name=...[,mode=0666]])

Encode the contents of the first argument in to
out should be file objects, but filenames are als
special filename "-" can be used to specify STD
objects are passed as arguments, in must be re
argument name can be used to specify the filer
by default it is the name of in. The fourth argur
UUencoding header.

uu.decode(in, [,out_file=...[, mode=...])

Decode the contents of the first argument in to
out_file is specified, it will be used as the outpu
from the UUencoding header. Arguments in and
are also accepted (the latter is deprecated). If t
out_file is either unspecified or is a filename), c

SEE ALSO: binascii *159;*

2.2.4 Cryptography

Python does not come with any standard and g
included capabilities are fairly narrow in purpos
standard library consist of several cryptographi
encryption algorithm. A quick survey of cryptog
absent from the standard library:

Symmetrical Encryption: Any technique by w
with a key K to produce a cyphertext C. Applica
C is called "decryption" and produces as output
form of symmetrical encryption.

Cryptographic Hash: Any technique by which
message M that has several additional properti
any M' such that the cryptographic hash of M' is
M', there is a very low probability that the cryp
Sometimes a third property is included: (3) Giv
hash H', examining the relationship between H

whose hash is H'. The standard modules *crypt,*
hashes.

Asymmetrical Encryption: Also called "public
pair of keys K_{pub} and K_{priv} can be generated t
an asymmetrical encryption technique will be c
plaintext message M, M equals P(K_{priv},P(M,K_{pu}
difficult to obtain a private-key K_{priv} that assur
P(M,K_{pub}), it is difficult to obtain M. In general
user generates K_{pub} and K_{priv}, then releases K
secret. There is no support for asymmetrical er

Digital Signatures: Digital signatures are real
cases, the same underlying algorithm is used fo
which a pair of keys K_{ver} and K_{sig} can be gene
algorithm for a digital signature will be called S
M equals P(K_{ver},P(M,K_{sig})). (2) Given only a v
signature key K_{sig} that assures the equality in
find any C' such that P(K_{ver},C) is a plausible m
is not a forgery). In general, in a digital signatu
then releases K_{ver} to other users but retains K
signatures in the standard library.

Those outlined are the most important cryptogr
introductions to cryptology and cryptography ca

tutorial is *Introduction to Cryptology Concepts*

<http://gnosis.cx/publish/programming/crypt

Further material is in *Introduction to Cryptolog*

<http://gnosis.cx/publish/programming/crypt

And more advanced material is in *Intermediate*

<http://gnosis.cx/publish/programming/crypt

A number of third-party modules have been cre
guide to these third-party tools is the Vaults of
<http://www.vex.net/parnassus/apyllo.py?i=9
library will be covered here specifically, since al
of the topic of text processing as such. Moreove
non-Python libraries, which will not be present
necessarily be maintained as new Python versi

The most important third-party modules are lis
believes are likely to be maintained and that pr
algorithms.

mxCrypto
amkCrypto

Marc-Andre Lemburg and Andrew Kuchlingboth
moduleshave played a game of leapfrog with ea
amkCrypto, respectively. Each release of either
providing compatible interfaces and overlapping
you read this is the best bet. Current informatic

<http://www.amk.ca/python/code/crypto.htm

Python Cryptography

Andrew Kuchling, who has provided a great dea
documents these cryptography modules at:

<http://www.amk.ca/python/writing/pycrypt/

M2Crypto

The *mxCrypto* and *amkCrypto* modules are mos
similar range of cryptographic capabilities for a
Siong's *M2Crypto*. Information and documentat

<http://www.post1.com/home/ngps/m2/>

fcrypt

Carey Evans has created *fcrypt*, which is a pure
standard library's *crypt* module. While probably
implementation, *fcrypt* will run anywhere that P
this functionality). *fcrypt* may be obtained at:

<http://home.clear.net.nz/pages/c.evans/sw/

crypt • Create and verify Unix-style passw

The crypt() function is a frequently used, but so
creation/verification tool. Under Unix-like syste
and may be called from wrapper functions in la
cryptographic hash based on the Data Encryptio
crypt() is based on an 8-byte key and a 2-byte
repeated encryption of a constant string, using
perturb the encryption in one of 4,096 ways. B
alphanumerics plus dot and slash.

By using a cryptographic hash, passwords may
imposter cannot easily produce a false passwor
stored in the password file, even given access t
"dictionary attacks" more difficult. If an impost
try applying crypt() to a candidate password ar
password file. Without a salt, the chances of m
higher. The salt (a random value should be use
guess by 4,096 times.

The *crypt* module is only installed on some Pytl
Moreover, the module, if installed, relies on an
approach to password creation, the third-party
Python reimplementation.

FUNCTIONS

crypt.crypt(passwd, salt)

Return an ASCII 13-byte encrypted password. T
to eight characters in length (extra characters a
The second argument salt must be a string up t
are truncated). The value of salt forms the first

```
>>> from crypt import crypt
>>> crypt('mypassword','XY')
'XY5XuULXk4pcs'
>>> crypt('mypasswo','XY')
'XY5XuULXk4pcs'
>>> crypt('mypassword...more.charact
'XY5XuULXk4pcs'
>>> crypt('mypasswo','AB')
'AB061nfYxWIKg'
>>> crypt('diffpass','AB')
```

'AB105BopaFYNs'

SEE ALSO: fcrypt *165;* md5 *167;* sha *170;*

md5 • Create MD5 message digests

RSA Data Security, Inc.'s MD5 cryptographic ha RFC1321. Like *sha*, and unlike *crypt, md5* allow arbitrary strings (Unicode strings may not be h considerationssuch as compatibility with other currently considered a better algorithm than MI cryptographic hashes. The operation of *md5* ob that the final hash value may be built progressi MD5 algorithm produces a 128-bit hash.

CONSTANTS

md5.MD5Type

The type of an *md5.new* instance.

CLASSES

md5.new([s])

Create an *md5* object. If the first argument s is
the initial string s. An MD5 hash can be comput

```
>>> import md5
>>> md5.new('Mary had a little lamb'
'e946adb45d4299def2071880d30136d4'
```

md5.md5([s])

Identical to *md5.new*.

METHODS

md5.copy()

Return a new *md5* object that is identical to the
terminal strings can be concatenated to the clo

```
>>> import md5
>>> m = md5.new('spam and eggs')
>>> m.digest()
```

```
'\xb5\x81f\x0c\xff\x17\xe7\x8c\x84\x
>>> m2 = m.copy()
>>> m2.digest()
'\xb5\x81f\x0c\xff\x17\xe7\x8c\x84\x
>>> m.update(' are tasty')
>>> m2.update(' are wretched')
>>> m.digest()
'*\x94\xa2\xc5\xceq\x96\xef&\x1a\xc9
>>> m2.digest()
'h\x8c\xfam\xe3\xb0\x90\xe8\x0e\xcb\
```

md5.digest()

Return the 128-bit digest of the current state o
byte will contain a full 8-bit range of possible va

```
>>> import md5              # Python 2.1
>>> m = md5.new('spam and eggs')
>>> m.digest()
'\xb5\x81f\x0c\xff\x17\xe7\x8c\x84\x

>>>  import md5             # Python <=
>>> m = md5.new('spam and eggs')
>>> m.digest()
```

```
'\265\201f\014\377\027\347\214\204\3
```

md5.hexdigest()

Return the 128-bit digest of the current state o
encoded string. Each byte will contain only valu
represents 8-bits of hash, and this format may
email.

```
>>> import md5
>>> m = md5.new('spam and eggs')
>>> m.hexdigest()
'b581660cff17e78c84c3a84ad02e6785'
```

md5.update(s)

Concatenate additional strings to the *md5* obje
The number of concatenation steps that go into
only the actual string that would result from co
However, for large strings that are determined
md5.update() numerous times. For example:

```
>>> import md5
>>> m1 = md5.new('spam and eggs')
```

```
>>> m2 = md5.new('spam')
>>> m2.update(' and eggs')
>>> m3 = md5.new('spam')
>>> m3.update(' and ')
>>> m3.update('eggs')
>>> m1.hexdigest()
'b581660cff17e78c84c3a84ad02e6785'
>>> m2.hexdigest()
'b581660cff17e78c84c3a84ad02e6785'
>>> m3.hexdigest()
'b581660cff17e78c84c3a84ad02e6785'
```

SEE ALSO: sha *170;* crypt *166;* binascii.crc32()

rotor • Perform Enigma-like encryption ar

The *rotor* module is a bit of a curiosity in the Py
encryption performed by *rotor* is similar to that
interesting and important Enigma algorithm. Gi
inventing the theory of computability, but also i
there is a nice literary quality to the inclusion o
mistaken for a robust modern encryption algori
there are two types of encryption algorithms: t
reading your messages, and those that will stop
organization from reading your messages. *roto*

rather bright little sisters. But *rotor* will not hel
On the other hand, there is nothing else in the
military-grade encryption, either.

CLASSES

rotor.newrotor(key [,numrotors])

Return a *rotor* object with rotor permutations a
If the second argument numrotors is specified,
can be used (more is stronger). A rotor encrypt

```
>>> rotor.newrotor('mypassword').enc
'\x10\xef\xf1\x1e\xeaor\xe9\xf7\xe5\
```

Object style encryption and decryption is perfo

```
>>> import rotor
>>> C = rotor.newrotor('pass2').encr
>>> r1 = rotor.newrotor('mypassword'
>>> C2 = r1.encrypt('Mary had a litt
>>> r1.decrypt(C2)
'Mary had a little lamb'
>>> r1.decrypt(C)    # Let's try it
'\217R$\217/sE\311\330~#\310\342\200
```

```
>>> r1.setkey('pass2')
>>> r1.decrypt(C)   # Let's try it
'Mary had a little lamb'
```

METHODS

rotor.decrypt(s)

Return a decrypted version of cyphertext string
initial positions.

rotor.decryptmore(s)

Return a decrypted version of cyphertext string
current positions.

rotor.encrypt(s)

Return an encrypted version of plaintext string
initial positions.

rotor.encryptmore(s)

Return an encrypted version of plaintext string
current positions.

rotor.setkey (key)

Set a new key for a *rotor* object.

sha • Create SHA message digests

The National Institute of Standards and Techno
best well-known cryptographic hash for most p
allows one to find the cryptographic hash of arb
hashed, however). Absent any other considerat
programsSHA is currently considered a better a
should be used for cryptographic hashes. The o
binascii.crc32() hashes in that the final hash va
concatenated strings. The SHA algorithm produ

CLASSES

sha.new([s])

Create an *sha* object. If the first argument s is the initial string s. An SHA hash can be comput

```
>>> import sha
>>> sha.new('Mary had a little lamb'
'bac9388d0498fb378e528d35abd05792291
```

sha.sha ([s])

Identical to *sha.new*.

METHODS

sha.copy()

Return a new *sha* object that is identical to the
terminal strings can be concatenated to the clo

```
>>> import sha
>>> s = sha.new('spam and eggs')
>>> s.digest()
'\276\207\224\213\255\375x\024\245b\
>>> s2 = s.copy()
>>> s2.digest()
```

```
'\276\207\224\213\255\375x\024\245b\
>>> s.update(' are tasty')
>>> s2.update(' are wretched')
>>> s.digest()
'\013^C\366\253?I\323\206nt\2443\251
>>> s2.digest()
'\013\210\237\216\014\3337X\333\221h
```

sha.digest()

Return the 160-bit digest of the current state o
will contain a full 8-bit range of possible values

```
>>> import sha               # Python 2.1
>>> s = sha.new('spam and eggs')
>>> s.digest()
'\xbe\x87\x94\x8b\xad\xfdx\x14\xa5b\

>>> import sha               # Python <=
>>> s = sha.new('spam and eggs')
>>> s.digest()
'\276\207\224\213\255\375x\024\245b\
```

sha.hexdigest()

Return the 160-bit digest of the current state o
encoded string. Each byte will contain only valu
represents 8-bits of hash, and this format may
email.

```
>>> import sha
>>> s = sha.new('spam and eggs')
>>> s.hexdigest()
'be87948badfd7814a5621e43d20faa3820
```

sha.update(s)

Concatenate additional strings to the *sha* objec
The number of concatenation steps that go into
only the actual string that would result from co
However, for large strings that are determined
sha.update() numerous times. For example:

```
>>> import sha
>>> s1 = sha.sha('spam and eggs')
>>> s2 = sha.sha('spam')
>>> s2.update(' and eggs')
>>> s3 = sha.sha('spam')
>>> s3.update(' and ')
>>> s3.update('eggs')
```

```
>>> s1.hexdigest()
'be87948badfd7814a5621e43d20faa3820
>>> s2.hexdigest()
'be87948badfd7814a5621e43d20faa3820
>>> s3.hexdigest()
'be87948badfd7814a5621e43d20faa3820
```

SEE ALSO: md5 *167;* crypt *166;* binascii.crc32(

2.2.5 Compression

Over the history of computers, a large number invented, mostly as variants on Lempel-Ziv anc for all sorts of data streams, but file-level archi and known application. Under MS-DOS and Wir ARJ, CAB, RAR, and other formatsbut the ZIP f variant. Under Unix-like systems, compress (.Z the most popular format on these systems, but compression rates. Under MacOS, the most pop additional variants on archive formats, but ZIPa on a number of platforms.

The Python standard library includes support fo module performs low-level compression of raw itself called by the high-level modules below fo

The modules *gzip* and *zipfile* provide file-level i

notable difference in the operation of *gzip* and *z*
underlying GZ and ZIP formats. gzip (GZ) opera
of concatenating collections of files to tools like
Unix-like systems) files like foo.tar.gz or foo.tgz
collection of files, then applying gzip to the resu
compression and archiving aspects in a single t
to create file-like objects based directly on the
to provide more specialized methods for naviga
individual compressed file images therein.

Also see Appendix B (A Data Compression Prim

gzip • Functions that read and write gzipp

The *gzip* module allows the treatment of the co
directly in a file-like manner. Uncompressed da
written back in, all without a caller knowing or
simple example illustrates this:

gzip_file.py

```
# Treat a GZ as "just another file"
import gzip, glob
print "Size of data in files:"
```

```
for fname in glob.glob('*'):
    try:
        if fname[-3:] == '.gz':
            s = gzip.open(fname).rea
        else:
            s = open(fname).read()
        print ' ',fname,'-',len(s),'
    except IOError:
        print 'Skipping',file
```

The module *gzip* is a wrapper around *zlib*, with
and decompression tasks. In many respects, *gz*
emulating and/or wrapping a file object.

SEE ALSO: mmap *147;* StringIO *153;* cStringIO

CLASSES

gzip.GzipFile([filename=...[,mode="rb" [,comp

Create a *gzip* file-like object. Such an object su
exception of .seek() and .tell(). Either the first
fileobj should be specified (likely by argument n

The second argument mode takes the mode of

(r, rb, a, ab, w, or wb may be specified with the
The third argument compresslevel specifies the
highest level, 9; an integer down to 1 may be s
operation (compression level of a read file com

gzip.open(filename=...[mode='rb [,compressle

Same as *gzip.GzipFile* but with extra arguments
gzip.open is always opened by name, not by ur

METHODS AND ATTRIBUTES

gzip.close()

Close the *gzip* object. No access is permitted af
object, the underlying file object is not closed,

SEE ALSO: FILE.close() *16;*

gzip.flush()

Write outstanding data from memory to disk.

SEE ALSO: FILE.close() *16;*

gzip.isatty()

Return 0. Compatibility method for file-like beh

SEE ALSO: FILE.isatty() *16;*

gzip.myfileobj

Attribute holding the underlying file object.

gzip.read([num])

If the first argument num is specified, return a
num characters are not available, return as ma
all the characters from current file position to e
position by the amount read.

SEE ALSO: FILE.read() *17;*

gzip.readline([length])

Return a string from the *gzip* object, starting fr
next newline character. The argument length lir
file position by the amount read.

SEE ALSO: FILE.readline() *17;*

gzip.readlines([sizehint=...])

Return a list of strings from the *gzip* object. Ea
including the trailing newline character(s). If ar
approximately sizehint characters worth of lines

SEE ALSO: FILE.readlines() *17;*

gzip.write(s)

Write the first argument s into the *gzip* object a
position is updated to the position following the

SEE ALSO: FILE.write() *17;*

gzip.writelines(list)

Write each element of list into the *gzip* object a

position is updated to the position following the
list must contain only strings, or a TypeError wi

Contrary to what might be expected from the n
newline characters. For the list elements actual
each element string must already have a newli
StringIO.StringIO.writelines() for an example.

SEE ALSO: FILE.writelines() *17;* StringIO.String

SEE ALSO: zlib *181;* zipfile *176;*

zipfile • Read and write ZIP files

The *zipfile* module enables a variety of operatio
created by applications such as PKZip, Info-Zip
inclusion of multiple file images within a single
directly file-like manner as *gzip* does. Nonethel
archive, add new file images to one, create a n
and directory information of a ZIP file.

An initial example of working with the *zipfile* m

```
>>> for name in 'ABC':
...     open(name,'w').write(name*10
...
```

```
>>> import zipfile
>>> z = zipfile.ZipFile('new.zip','w
>>> z.write('A')
>>> z.write('B','B.newname',zipfile.
>>> z.write('C','C.newname')
>>> z.close()
>>> z = zipfile.ZipFile('new.zip')
>>> z.testzip()
>>> z.namelist()
['A', 'B.newname', 'C.newname']
>>> z.printdir()
File Name
A
B.newname
C.newname
>>> A = z.getinfo('A')
>>> B = z.getinfo('B.newname')
>>> A.compress_size
11
>>> B.compress_size
1000
>>> z.read(A.filename)[:40]
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>>> z.read(B.filename)[:40]
'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

```
>>> # For comparison, see what Info-
>>> import os
>>> print os.popen('unzip -v new.zip
Archive: new.zip
 Length  Method   Size  Ratio   Date
 ------  ------   ----  -----   ----
   1000  Defl:N     11   99%  07-18-
   1000  Stored   1000    0%  07-18-
   1000  Defl:N     11   99%  07-18-
 ------          ------  ---
   3000           1022  66%
```

The module *gzip* is a wrapper around *zlib*, with
and decompression tasks.

CONSTANTS

Several string constants (*struct* formats) are us
ZIP format. These constants are not normally u

```
zipfile.stringCentralDir = 'PK\x01\x
zipfile.stringEndArchive = 'PK\x05\x
zipfile.stringFileHeader = 'PK\x03\x
zipfile.structCentralDir = '<4s4B4H3
```

```
zipfile.structEndArchive = '<4s4H21H
zipfile.structFileHeader = '<4s2B4H3
```

Symbolic names for the two supported compres

```
zipfile.ZIP_STORED = 0
zipfile.ZIP_DEFLATED = 8
```

FUNCTIONS

zipfile.is_zipfile(filename=...)

Check if the argument filename is a valid ZIP a
not recognized as valid archives. Return 1 if val
guarantee archive is fully intact, but it does pro

CLASSES

zipfile.PyZipFile(pathname)

Create a *zipfile.ZipFile* object that has the extra
method allows you to recursively add all *.py[c
purpose, but a special feature to aid *distutils*.

zipfile.ZipFile(file=...[,mode='r' [,compression

Create a new *zipfile.ZipFile* object. This object i
first argument file must be specified and is sim
manipulated. The second argument mode may
archive in read-only mode; w to truncate the fi
existing archive and add to it. The third argume
methodZIP_DEFLATED requires that *zlib* and th

zipfile.ZipInfo()

Create a new *zipfile.ZipInfo* object. This object
archived filename and its file image. Normally,
but only look at the *zipfile.ZipInfo* objects that
zipfile.ZipFile.infolist(), zipfile.ZipFile.getinfo(),
special cases like *zipfile.ZipFile.writestr()*, it is u

METHODS AND ATTRIBUTES

zipfile.ZipFile.close()

Close the *zipfile.ZipFile* object, and flush any ch
closed to perform updates.

zipfile.ZipFile.getinfo(name=...)

Return the *zipfile.ZipInfo* object corresponding
ZIP archive, a KeyError is raised.

zipfile.ZipFile.infolist()

Return a list of *zipfile.ZipInfo* objects contained
is simply a list of instances of the same type. If
zipfile.ZipFile.getinfo() is a better method to us
however, *zipfile.ZipFile.infolist()* provides a nice

zipfile.ZipFile.namelist()

Return a list of the filenames of all the archived

zipfile.ZipFile.printdir()

Print to STDOUT a pretty summary of archived
are similar to running Info-Zip's unzip with the

zipfile.ZipFile.read(name=...)

Return the contents of the archived file with file

zipfile.ZipFile.testzip()

Test the integrity of the current archive. Return
with corruption. If everything is valid, return N

zipfile.ZipFile.write(filename=...[,arcname=...[,

Add the file filename to the *zipfile.ZipFile* objec
specified, use arcname as the stored filename (
argument compress_type is specified, use the i
archive must be opened in w or a mode.

zipfile.ZipFile.writestr(zinfo=..., bytes=...)

Write the data contained in the second argume
meta-information must be contained in attribut
data, and time should be included; other inforn
be opened in w or a mode.

zipfile.ZipFile.NameToInfo

Dictionary that maps filenames in archive to co
method *zipfile.ZipFile.getinfo()* is simply a wrap

zipfile.ZipFile.compression

Compression type currently in effect for new *zi*
due caution (most likely not at all after initializ

zipfile.ZipFile.debug = 0

Attribute for level of debugging information ser
(no output) to 3 (verbose). May be modified.

zipfile.ZipFile.filelist

List of *zipfile.ZipInfo* objects contained in the *z*
zipfile.ZipFile.infolist() is simply a wrapper to re
(most likely not at all).

zipfile.ZipFile.filename

Filename of the *zipfile.ZipFile* object. DO NOT m

zipfile.ZipFile.fp

Underlying file object for the *zipfile.ZipFile* obje

zipfile.ZipFile.mode

Access mode of current *zipfile.ZipFile* object. D

zipfile.ZipFile.start_dir

Position of start of central directory. DO NOT m

zipfile.ZipInfo.CRC

Hash value of this archived file. DO NOT modify

zipfile.ZipInfo.comment

Comment attached to this archived file. Modify
zipfile.ZipFile.writestr()).

zipfile.ZipInfo.compress_size

Size of the compressed data of this archived fil

zipfile.ZipInfo.compress_type

Compression type used with this archived file. I
zipfile.ZipFile.writestr()).

zipfile.ZipInfo.create_system

System that created this archived file. Modify w
zipfile.ZipFile.writestr()).

zipfile.ZipInfo.create_version

PKZip version that created the archive. Modify
zipfile.ZipFile.writestr()).

zipfile.ZipInfo.date_time

Timestamp of this archived file. Modify with du *zipfile.ZipFile.writestr()*).

zipfile.ZipInfo.external_attr

File attribute of archived file when extracted.

zipfile.ZipInfo.extract_version

PKZip version needed to extract the archive. M
zipfile.ZipFile.writestr()).

zipfile.ZipInfo.file_offset

Byte offset to start of file data. DO NOT modify

zipfile.ZipInfo.file size

Size of the uncompressed data in the archived

zipfile.ZipInfo.filename

Filename of archived file. Modify with due cauti

zipfile.ZipInfo.header_offset

Byte offset to file header of the archived file. D

zipfile.ZipInfo.volume

Volume number of the archived file. DO NOT m

EXCEPTIONS

zipfile.error

Exception that is raised when corrupt ZIP file is

zipfile.BadZipFile

Alias for *zipfile.error*.

SEE ALSO: zlib *181;* gzip *173;*

zlib • Compress and decompress with zlib

zlib is the underlying compression engine for al modules. Moreover, *zlib* is extremely useful in i data that does not necessarily live in files (or w if it winds up in them indirectly). The Python *zl* system library.

There are two basic modes of operation for *zlib* an uncompressed string to *zlib.compress()* and *zlib.decompress()* is symmetrical. In a more co or decompression objects that are able to recei streams, and return partial results based on wh operation is similar to the way one uses *sha.sh* *rotor.encryptmore()*, or *binascii.crc32()* (albeit For large byte-streams that are determined, it compression/decompression objects than it wou string at once (for example, if the input or resu

CONSTANTS

zlib.ZLIB_VERSION

The installed zlib system library version.

zlib.Z_BEST_COMPRESSION = 9

Highest compression level.

zlib.Z_BEST_SPEED = 1

Fastest compression level.

zlib.Z_HUFFMAN_ONLY = 2

Intermediate compression level that uses Huffn

FUNCTIONS

zlib.adler32(s [,crc])

Return the Adler-32 checksum of the first argu
specified, it will be used as an initial checksum.
checksum and continuation. An Adler-32 check
a CRC32 checksum. Unlike *md5* or *sha*, an Adle
cryptographic hashes, but merely for detection

SEE ALSO: zlib.crc32() *182*; md5 *167;* sha *17*

zlib.compress(s [,level])

Return the zlib compressed version of the strin
argument level is specified, the compression te
level ranges from 1 to 9 and may also be specif
Z_BEST_COMPRESSION and Z_BEST_SPEED. T
the desired compression level (usually within a
and within a few percent of the size of Z_BEST_

SEE ALSO: zlib.decompress() *182;* zlib.compres

zlib.crc32(s [,crc])

Return the CRC32 checksum of the first argume
it will be used as an initial checksum. This allov
continuation. Unlike *md5* or *sha*, a CRC32 chec
hashes, but merely for detection of accidental c

Identical to *binascii.crc32()* (example appears t

SEE ALSO: binascii.crc32() *160;* zlib.adler32()

zlib.decompress(s [,winsize [,buffsize]])

Return the decompressed version of the zlib co

second argument winsize is specified, it determ
size. The default winsize is 15. If the third argu
size of the decompression buffer. The default bu
allocated if needed. One rarely needs to use wi
defaults.

SEE ALSO: zlib.compress() *182*; zlib.decompres

CLASS FACTORIES

zlib does not define true classes that can be sp
zlib.decompressobj() are actually factory-functi
instance objects, just as classes do, but they do
most users, the difference is not important: To
object, you just call that factory-function in the

zlib.compressobj([level])

Create a compression object. A compression ob
strings that are fed to it while maintaining the s
compressed byte-streams. If argument level is
fine-tuned. The compression level ranges from
usually the desired compression level.

SEE ALSO: zlib.compress() *182*; zlib.decompres

zlib.decompressobj([winsize])

Create a decompression object. A decompressi
new strings that are fed to it while maintaining
decompressed byte-streams. If the argument w
logarithm of the history buffer size. The default

SEE ALSO: zlib.decompress() *182*; zlib.compres

METHODS AND ATTRIBUTES

zlib.compressobj.compress(s)

Add more data to the compression object. If th
is returned, otherwise an empty string. All retu
zlib.compressobj.compress() should be concate
a string or a decompression object). The examp
lets one examine the buffering behavior of com

zlib_objs.py

```
# Demonstrate compression object str
import zlib, glob
decom = zlib.decompressobj()
```

```
com = zlib.compressobj()
for file in glob.glob('*'):
    s = open(file).read()
    c = com.compress(s)
    print 'COMPRESSED:', len(c), 'by
    d = decom.decompress(c)
    print 'DECOMPRESS:', len(d), 'by
    print 'UNUSED DATA:', len(decom.
    raw_input('-- %s (%s bytes) --'
f = com.flush()
m = decom.decompress(f)
print 'DECOMPRESS:', len(m), 'bytes
print 'UNUSED DATA:', len(decom.unus
```

SEE ALSO: zlib.compressobj.flush() *184*; zlib.d
zlib.compress() *182*;

zlib.compressobj.flush([mode])

Flush any buffered data from the compression
zlib.compressobj.compress(), the output of a *z*
concatenated to the same decompression byte-
are. If the first argument mode is left empty, o
compression object cannot be used further, and
Z_SYNC_FLUSH or Z_FULL_FLUSH are specified

but some uncompressed data may not be recov

SEE ALSO: zlib.compress() *182*; zlib.compresso

zlib.decompressobj.unused_data

As indicated, *zlib.decompressobj.unused-data* i
If any partial compressed stream cannot be de
stream received, the remainder is buffered in t
a compression object forms a complete decomp
instance attribute. However, if data is received
decompression may be possible on a particular

SEE ALSO: zlib.decompress() *182*; zlib.decomp

zlib.decompressobj.decompress (s)

Return the decompressed data that may be der
state and the argument s data passed in. If all
remainder is left in *zlib.decompressobj.unused-*

zlib.decompressobj.flush()

Return the decompressed data from any bytes this call, the decompression object cannot be u

EXCEPTIONS

zlib.error

Exception that is raised by compression or deco

SEE ALSO: gzip *173;* zipfile *176;*

2.2.6 Unicode

Note that Appendix C (Understanding Unicode)

Unicode is an enhanced set of character entitie defined in ASCII encoding and the codepage-sp characters each. The full Unicode character set of codepoints already fixedcan contain literally representation of a large number of national ch even the large character sets of Chinese-Japan

Although Unicode defines a unique codepoint fo are numerous *encodings* that correspond to ea defines ASCII characters as single bytes with st characters, a variable number of bytes (up to 6 "escape" to Unicode being indicated by high-bit

UTF-16 is similar, but uses either 2 or 4 bytes t
UTF-32 is a format that uses a fixed 4-byte valu
however, is not currently supported by Python.

Native Unicode support was added to Python 2.
that Python supports Unicodeit brings the world
computer applications. But in practice, you hav
because it is all too easy to encounter glitches

```
>>> alef, omega = unichr(1488), unic
>>> unicodedata.name(alef)
>>> print alef
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeError: ASCII encoding error:
>>> print chr(170)

>>> if alef == chr(170): print "Hebr
...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeError: ASCII decoding error:
```

A Unicode string that is composed of only ASCI
(but not identical) to a Python string of the sam

```
>>> u"spam" == "spam"
1
>>> u"spam" is "spam"
0
>>> "spam" is "spam"   # string inte
1
>>> u"spam" is u"spam" # unicode int
1
```

Still, the care you take should not discourage y
as Unicode enables. It is really amazingly powe
talking dog: It is not that he speaks so *well,* bu

Built-In Unicode Functions/Methods

The Unicode string method *u"".encode()* and th
operations. The Unicode string method returns
represent it (using the specified or default enco
these encoded strings and produces the Unicod
Specifically, suppose we define the function:

```
>>> chk_eq = lambda u,enc: u == unic
```

The call *chk_eq(u, enc*) should return 1 for eve
encoding name and u is capable of being repre

The set of encodings supported for both built-ir
be registered using the *codecs* module. Each er
it, and the case of the string is normalized befo
encodings):

ascii, us-ascii

Encode using 7-bit ASCII.

base64

Encode Unicode strings using the base64 4-to-3

latin-1, iso-8859-1

Encode using common European accent charac
character's *ord()* values are identical to their U

quopri

Encode in quoted printable format.

rot13

Not really a Unicode encoding, but "rotate 13 c
example and convenience.

utf-7

Encode using variable byte-length encoding tha
utf-8, ASCII characters encode themselves.

utf-8

Encode using variable byte-length encoding tha

utf-16

Encoding using 2/4 byte encoding. Include "enc

utf-16-le

Encoding using 2/4 byte encoding. Assume "litt
indicator bytes.

utf-16-be

Encoding using 2/4 byte encoding. Assume "big
indicator bytes.

unicode-escape

Encode using Python-style Unicode string const

raw-unicode-escape

Encode using Python-style Unicode raw string c

The error modes for both built-ins are listed be
be handled in any of several ways:

strict

Raise UnicodeError for all decoding errors. Defa

ignore

Skip all invalid characters.

replace

Replace invalid characters with ? (string target)

u"".encode([enc [,errmode]])
"".encode([enc [,errmode]])

Return an encoded string representation of a U
representation is in the style of encoding enc (
writing to a file or stream that other applicatior
several encodings:

```
>>> alef = unichr(1488)
>>> s = 'A'+alef
>>> s
u'A\u05d0'
>>> s.encode('unicode-escape')
'A\\u05d0'
>>> s.encode('utf-8')
'A\xd7\x90'
>>> s.encode('utf-16')
'\xff\xfeA\x00\xd0\x05'
```

```
>>> s.encode('utf-16-le')
'A\x00\xd0\x05'
>>> s.encode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeError: ASCII encoding error:
>>> s.encode('ascii','ignore')
'A'
```

unicode(s [,enc [,errmode]])

Return a Unicode string object corresponding to
argument s. The string s might be a string that
application. The representation is treated as co
the second argument is specified, or system de
handled in the default strict style or in a style s

unichr(cp)

Return a Unicode string object containing the s
codepoint is passed in the argument cp.

---

codecs • Python Codec Registry, API, and

The *codecs* module contains a lot of sophisticat
Python's Unicode handling. Most of those capat
who are just interested in text processing need
module, therefore, will break slightly with the s
only two very useful wrapper functions within t

codecs.open(filename=...[,mode='rb' [,encodi

This wrapper function provides a simple and dir
treating its contents directly as Unicode. In con
built-in *open()* function are written and read as
file involves multiple passes through *u"".encod*

The first argument filename specifies the name
mode is specified, the read/write mode can be
those used by *open()*. If the third argument en
to interpret the file (an incorrect encoding will
handling may be modified by specifying the fou
as with the built-in *unicode()* function). A fifth a
specific buffer size (on platforms that support t

An example of usage clarifies the difference bet

```
>>> import codecs
>>> alef = unichr(1488)
```

```
>>> open('unicode_test','wb').write(
>>> open('unicode_test').read()   #
'A\xd7\x90'
>>> # Now read directly as Unicode
>>> codecs.open('unicode_test', encc
u'A\u05d0'
```

Data written back to a file opened with *codecs.*

SEE ALSO: open() *15;*

codecs.EncodedFile(file=..., data_encoding=.

This function allows an already opened file to b layer. The mode and buffering are taken from t argument data_encoding and a third argument strings in one encoding within an application, tl file encoding. As with *codecs.open()* and *unicoc* with the fourth argument errors.

The most likely purpose for *codecs.EncodedFile* byte-streams from multiple sources, encoded a wrapping file objects (or file-like objects) in an in one encoding can be transparently written tc An example clarifies:

```
>>> import codecs
>>> alef = unichr(1488)
>>> open('unicode_test','wb').write(
>>> fp = open('unicode_test','rb+')
>>> fp.read()      # Plain string w/
'A\xd7\x90'
>>> utf16_writer = codecs.EncodedFil
>>> ascii_writer = codecs.EncodedFil
>>> utf16_writer.tell()    # Wrapper
3
>>> s = alef.encode('utf-16')
>>> s              # Plain string as
'\xff\xfe\xd0\x05'
>>> utf16_writer.write(s)
>>> ascii_writer.write('XYZ')
>>> fp.close()                  # File sh
>>> open('unicode_test').read()
'A\xd7\x90\xd7\x90XYZ'
```

SEE ALSO: codecs.open() *189;*

unicodedata • Database of Unicode charac

The module *unicodedata* is a database of Unico

unicodedata take as an argument one Unicode
the character contained in a plain (non-Unicode
essentially informational, rather than transform
decisions about the transformations performed
unicodedata. The short utility below provides al
codepoint:

unichr_info.py

```
# Return all the information [unicod
# about the single unicode character
# is specified as a command-line arg
# Arg may be any expression evaluati
from unicodedata import  *
import sys
char = unichr(eval(sys.argv[1]))
print 'bidirectional', bidirectional
print 'category     ', category(char
print 'combining    ', combining(cha
print 'decimal      ', decimal(char,
print 'decomposition', decompositior
print 'digit        ', digit(char,0)
print 'mirrored     ', mirrored(char
print 'name         ', name(char,'NC
```

```
print 'numeric        ', numeric(char,
try: print 'lookup        ', 'lookup(
except: print "Cannot lookup"
```

The usage of unichr_info.py is illustrated below

```
% python unichr_info.py 1488
bidirectional R
category      Lo
combining     0
decimal       0
decomposition
digit         0
mirrored      0
name          HEBREW LETTER ALEF
numeric       0
lookup        u'\u05d0'

% python unichr_info.py ord('1')
bidirectional EN
category      Nd
combining     0
decimal       1
decomposition
digit         1
```

```
mirrored        0
name            DIGIT ONE
numeric         1.0
lookup          u'1'
```

For additional information on current Unicode c

<http://www.unicode.org/Public/UNIDATA/Un

FUNCTIONS

unicodedata.bidirectional(unichr)

Return the bidirectional characteristic of the ch Possible values are AL, AN, B, BN, CS, EN, ES, S, and WS. Consult the URL above for details o to-right), R (right-to-left), and WS (whitespace

unicodedata.category (unichr)

Return the category of the character specified i Cf, Cn, Ll, Lm, Lo, Lt, Lu, Mc, Me, Mn, Nd, Nl, N Zl, Zp, and Zs. The first (capital) letter indicate (punctuation), S (symbol), Z (separator), or C mnemonic within the major category of the firs

unicodedata.combining(unichr)

Return the numeric combining class of the char include values such as 218 (below left) or 210 (details.

unicodedata.decimal(unichr [,default])

Return the numeric decimal value assigned to t If the second argument default is specified, ret raise ValueError).

unicodedata.decomposition(unichr)

Return the decomposition mapping of the chara empty string if none exists. Consult the URL ab characters may be broken into component char

```
>>> from unicodedata import *
>>> name(unichr(190))
'VULGAR FRACTION THREE QUARTERS'
```

```
>>> decomposition(unichr(190))
'<fraction> 0033 2044 0034'
>>> name(unichr(0x33)), name(unichr(
('DIGIT THREE', 'FRACTION SLASH', 'I
```

unicodedata.digit(unichr [,default])

Return the numeric digit value assigned to the
the second argument default is specified, retur
ValueError).

unicodedata.lookup(name)

Return the Unicode character with the name sp
must be exact, and ValueError is raised if no m

```
>>> from unicodedata import *
>>> lookup('GREEK SMALL LETTER ETA')
u'\u03b7'
>>> lookup('ETA')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: undefined character name
```

SEE ALSO: unicodedata.name() *193;*

unicodedata.mirrored(unichr)

Return 1 if the character specified in the argum
bidirection text. Return 0 otherwise.

unicodedata.name(unichr)

Return the name of the character specified in tl
have a regular form by descending category im

SEE ALSO: unicodedata.lookup() *193;*

unicodedata.numeric(unichr [,default])

Return the floating point numeric value assigne
unichr. If the second argument default is specif
(otherwise raise ValueError).

2.3 Solving Problems

2.3.1 Exercise: Many ways to take out the gar

DISCUSSION

Recall, if you will, the dictum in "The Zen of Pyt
only oneobvious way to do it." As with most dic
Also as with most dictums, this is not necessari

A discussion on the newsgroup <comp.lang.pyt
simple problem. The immediate problem was th
with a variety of dividers and delimiters inside t
7890, or 123/456-7890 might all represent the
be encountered in textual data sources (such a
field. For purposes of this problem, the canonic

The problem mentioned here can be generalize
interested in only some of the characters withir
and the rest is simply filler. So the general prob
filler.

The first and "obvious" approach might be a pr
version of this approach might look like:

```
>>> s = '(123)/456-7890'
>>> result = ''
>>> for c in s:
...     if c in '0123456789':
...         result = result + c
...
>>> result
'1234567890'
```

This first approach works fine, but it might seen
single action. And it might also seem odd that y
rather than just transform the whole string.

One possibly simpler approach is to use a regul
the next chapter, or who know regular expressi

```
>>> import re
>>> s = '(123)/456-7890'
>>> re.sub(r'\D', '', s)
```

```
'1234567890'
```

The actual work done (excluding defining the ir one short expression. Good enough, but one ca frequently far slower than basic string operatio example presented, but for processing megaby

Using a functional style of programming is one tersely, and perhaps more efficiently. For exam

```
>>> s = '(123)/456-7890'
>>> filter(lambda c:c.isdigit(), s)
'1234567890'
```

We also get something short, without needing t technique that utilizes string object methods ar hopes on the great efficiency of Python dictiona

```
>>> isdigit = {'0':1,'1':1,'2':1,'3'
...            '5':1,'6':1,'7':1,'8'
>>> " .join([x for x in s if isdigit
'1234567890'
```

QUESTIONS

1: Which content extraction technique seems m
use? Explain why.

2: What intuitions do you have about the perfo
to large data sets? Are there differences in c
operating on one single large string input an
inputs?

3: Construct a program to verify or refute your

4: Can you think of ways of combining these te
other techniques available that might be eve
string.translate() does)? Construct a faster t

5: Are there reasons other than raw processing
others? Explain these reasons, if they exist.

2.3.2 Exercise: Making sure things are what t

DISCUSSION

The concept of a "digital signature" was introdu

Python standard library does not include (direct
to characterize a digital signature is as some in
other information really is what it purports to b
broader set of things than just digital signature
talk about the "threat model" a crypto-system

Data may be altered by malicious tampering, b
storage-media errors, or by program errors. Th
easiest threat to defend against. The standard
and send that also. The receiver of the data ca
herselfusing the same algorithmand compare it
the one below does this:

crc32.py

```
# Calculate CRC32 hash of input file
# Incremental read for large input s
# Usage:      python crc32.py [file1
#    or:      python crc32.py < STDIN

import binascii
import fileinput
filelist = []
crc = binascii.crc32('')
for line in fileinput.input():
```

```
    if fileinput.isfirstline():
        if fileinput.isstdin():
            filelist.append('STDIN')
        else:
            filelist.append(fileinpu
    crc = binascii.crc32(line,crc)
print 'Files:', ' '.join(filelist)
print 'CRC32:', crc
```

A slightly faster version could use *zlib.adler32(* randomly corrupted file would have the right CI enough not to worry about most times.

A CRC32 hash, however, is far too weak to be u will almost surely not create a chance hash coll parlancecan find one relatively easily. Specifica find an M' such that CRC32(M) equals CRC32(M M' appears plausible as a message to the recei difficult.

To thwart fraudulent messages, it is necessary *SHA* or *MD5*. Doing so is almost the same utilit

sha.py

```
# Calculate SHA hash of input files
```

```
# Usage:     python sha.py [file1 [f
#    or:     python sha.py < STDIN

import sha, fileinput, os, sys
filelist = []
sha = sha.sha()
for line in fileinput.input():
    if fileinput.isfirstline():
        if fileinput.isstdin():
            filelist.append('STDIN')
        else:
            filelist.append(fileinpu
    sha.update(line[:-1]+os.linesep)
sys.stderr.write('Files: '+' '.join(
print sha.hexdigest()
```

An SHA or MD5 hash cannot be forged practica tamperer, we need to worry about whether the can produce a false SHA hash that matches her common procedure is to attach the hash to the last line of the data file, or within some wrappe channel" transmission. One alternative is "out c cryptographic hashes. For example, a set of cry placed on a Web page. Merely transmitting the but it does require Mallory to attack both chanr

By using encryption, it is possible to transmit a
encrypt the hash and attach that encrypted ver
identifying information before the encryption, t
Otherwise, one could simply include both the h
encryption of the hash, an asymmetrical encryp
Python standard library, the best we can do is t
rotor. For example, we could use the utility bel

hash_rotor.py

```
#!/usr/bin/env python
# Encrypt hash on STDIN using sys.ar
import rotor, sys, binascii
cipher = rotor.newrotor(sys.argv[1])
hexhash = sys.stdin.read()[:-1]  # r
print hexhash
hash = binascii.unhexlify(hexhash)
sys.stderr.write('Encryption: ')
print binascii.hexlify(cipher.encryp
```

The utilities could then be used like:

```
%  cat mary.txt
Mary had a little lamb
% python sha.py mary.txt I hash_rotc
```

```
Files: mary.txt
SHA: Encryption:
% cat mary.txt
Mary had a little lamb
c49bf9a7840f6c07ab00b164413d7958e094
63a9d3a2f4493d957397178354f21915cb36
```

The penultimate line of the file now has its SHA
the hash. The password used will somehow nee
validate the appended document (obviously, th
and more proprietary documents than in the ex

QUESTIONS

1: How would you wrap up the suggestions in t
complete "digital_signatures.py" utility or m
completed utility?

2: Why is CRC32 not suitable for cryptographic
should not need to know the details of the a
coverage of hash results important for any h

Explain in your own words why hashes serve

3: malicious attacker in the scenarios above, ho
crypto-systems outlined here? What lines of
sketched out or programmed in (1)?

4: If messages are subject to corruptions, inclu
short length of hashes may make problems i
might you enhance the document verificatio
hash itself? How might you allow more accu
of a large document (it may be desirable to
document)?

5: Advanced: The RSA public-key algorithm is a
modulo exponentiation operations and some
among other places, at the author's *Introduc*
<http://gnosis.cx/publish/programming/cry

Try implementing an RSA public-key algorithm
signature system you developed above.

2.3.3 Exercise: Finding needles in haystacks

DISCUSSION

Many texts you deal with are loosely structured

ordered records. For documents of that sort, a
"What is (or isn't) in the documents?"at a more
might obtain by actually *reading* the document
collection of documents to determine the (com
relevant to a given area of interest.

A certain category of questions about documen
processing. For example, to locate all the files
having a certain file size, some basic use of the
utility to do such a search, which includes some
The search itself is only a few lines of code:

findfile1.py

```
# Find files matching date and size
_usage = """
Usage:
   python findfile1.py [-start=days_
                       [-small=min_s
Example:
  python findfile1.py -start=10 -enc
"""
import os.path
import time
import glob
```

```
import sys

def parseargs(args):
    """Somewhat flexible argument pa

    Switches can start with - or /,
    No error checking for bad argume
    """
    now = time.time()
    secs_in_day = 60*60*24
    start = 0              # start of e
    end = time.time()      # right now
    small = 0              # empty file
    large = sys.maxint     # max file s
    pat = '*'              # match all
    for arg in args:
        if arg[0] in '-/':
            if   arg[1:6]=='start': st
            elif arg[1:4]=='end':   en
            elif arg[1:6]=='small': sm
            elif arg[1:6]=='large': la
            elif arg[1] in 'h?': print
        else:
            pat = arg
    return (start,end,small,large,pa
```

```
if __name__ == '__main__':
    if len(sys.argv) > 1:
        (start,end,small,large,pat)
        for fname in glob.glob(pat):
            if not os.path.isfile(fr
                continue          #
            modtime = os.path.getmti
            size = os.path.getsize(f
            if small <= size <= larg
                print time.ctime(moc
    else: print _usage
```

What about searching for text inside files? The contents quickly and could be used to search fil collections, hits may be common. To make sens number of hits can help. The utility below perfo without the argument parsing of findfile1.py):

findfile2.py

```
# Find files that contain a word
_usage = "Usage: python findfile.py
import os.path
```

```
import glob
import sys

if len(sys.argv) == 2:
    search_word = sys.argv[1]
    results = []
    for fname in glob.glob('*'):
        if os.path.isfile(fname):  #
            text = open(fname).read()
            fsize = len(text)
            hits = text.count(search_
            density = (fsize > 0) anc
            if density > 0:         #
                results.append((densi
    results.sort()
    results.reverse()
    print 'RANKING  FILENAME'
    print '-------  ---------------
    for match in results:
        print '%6d  '%int(match[0] *
else:
    print _usage
```

Variations on these are, of course, possible. Bu
searches and rankings by adding new search o

example, adding some regular expression optic
the grep utility.

The place where a word search program like th
locating documents in *very* large document coll
optimized, as grep simply takes a while to sear
to *shortcut* this search time, as well as add son

A technique for rapid searching is to perform a
create an indexi.e., databaseof those generic se
not *really* search contents, but only check the a
searches. The utility indexer.py is a functional e
most current version may be downloaded from

The utility indexer.py allows very rapid searchir
words within a file. For example, one might wa
sources, such as VARCHAR database fields) tha
Supposing there are many thousands of candid
basis could be slow. But indexer.py creates a co
dictionaries that provide answers to such inquir

The full source code to indexer.py is worth read
persistence mechanisms and with an object-ori
reuse. The underlying idea is simple, however.
collection of documents:

```
*Indexer.fileids:      fileid --> fil
*Indexer.files:        filename --> (
```

```
*Indexer.words:        word --> {file
```

The essential mapping is *Indexer.words. For e
often? The mappings *Indexer.fileids and *Inde
shorter numeric aliases to be used instead of lo
performance boost and storage saver). The sec
wordcount for each file. This allows a ranking o
thought is that a megabyte file with ten occurre
Python than is a kilobyte file with the same ten

Both generating and utilizing the mappings abo
one basically simply needs the intersection of t
*Indexer.words dictionary, one value for each w
incrementing counts in the nested dictionary of

QUESTIONS

1: One of the most significantand surprisingly s
indexes is figuring out just what a "word" is.
determine word identities? How might you h
How might you disallow binary strings that a
identification tests against real-world docum

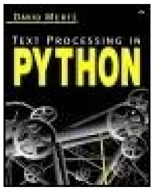
Could other data structures be used to store
above? If other data structures are used, wh

2: disadvantages do you expect to encounter? A
allow for additional search capabilities than t
other indexed search capabilities would have

3: Consider adding integrity guarantees to inde
synchronization with the underlying docume
integrity? Hint: consider *binascii.crc32, sha*,
would be needed for integrity checks? Imple

4: The utility indexer.py has some ad hoc exclu
index, based simply on some file extensions.
nontextual data? What does it mean for a do
istextual.py that will identify text and nontex
satisfaction?

5: Advanced: indexer.py implements several dif
mechanisms might you use from those imple
do better than SlicedZPickleIndexer (the bes



Text Processing in PythonBy David Mertz

Table of Contents

Chapter 3. Regular Expressions

Regular expressions allow extremely valuable t processing techniques, but ones that warrant c explanation. Python's *re* module, in particular, a numerous enhancements to basic regular expre (such as named backreferences, lookahead ass backreference skipping, non-greedy quantifiers others). A solid introduction to the subtleties of expressions is valuable to programmers engage processing tasks.

The prequel of this chapter contains a tutorial c expressions that allows a reader unfamiliar with expressions to move quickly from simple to cor elements of regular expression syntax. This tut aimed primarily at beginners, but programmers with regular expressions in other programming benefit from a quick read of the tutorial, which the particular regular expression dialect in Pyth

It is important to note up-front that regular exp while very powerful, also have limitations. In br regular expressions cannot match patterns that arbitrary depths. If that statement does not ma read Chapter 4, which discusses parsersto a lar

extent, parsing exists to address the limitations
regular expressions. In general, if you have dou
whether a regular expression is sufficient for yo
try to understand the examples in Chapter 4, p
the discussion of how you might spell a floating
number.

Section 3.1 examines a number of text process
problems that are solved most naturally using r
expressions. As in other chapters, the solutions
presented to problems can generally be adopte
as little utilities for performing tasks. However,
elsewhere, the larger goal in presenting probler
solutions is to address a style of thinking about
class of problems than those whose solutions a
presented directly in this book. Readers who ar
interested in a range of ready utilities and mod
probably want to check additional resources on
such as the Vaults of Parnassus
<http://www.vex.net/parnassus/> and the Pyt
Cookbook
<http://aspn.activestate.com/ASPN/Python/Co

Section 3.2 is a "reference with commentary" o
Python standard library modules for doing regu
expression tasks. Several utility modules and b
compatibility regular expression engines are av
but for most readers, the only important modul

re itself. The discussions interspersed with each
try to give some guidance on why you would w
a given module or function, and the reference
documentation tries to contain more examples
typical usage than does a plain reference. In m
cases, the examples and discussion of individu
functions address common and productive desi
patterns in Python. The cross-references are in
contextualize a given function (or other thing) i
of related ones (and to help a reader decide wh
right for her). The actual listing of functions, co
classes, and the like are in alphabetical order w
category.

Chapter 3. Regular Expressions

3.1 A Regular Expression Tutorial

Some people, when confronted with a probler
expressions." Now they have two problems.

Jamie Zawinski, <alt.religion.emacs> (08/12/

3.1.1 Just What Is a Regular Expression, Any

Many readers will have some background with
any. Those with experience using regular expre
probably skip this tutorial section. But readers
called regexes by users) should read this sectic
from a refresher.

A regular expression is a compact way of descr
them to search for patterns and, once found, tc
can also be used to launch programmatic actior

Jamie Zawinski's tongue-in-cheek comment in t
expressions are amazingly powerful and deeply
writing them is just as error-prone as writing a
always better to solve a genuinely simple probl
simple, think about regular expressions.

A large number of tools other than Python inco
functionality. Unix-oriented command-line tools
for regular expression processing. Many text ed
on regular expressions. Many programming lan
such as Perl and TCL, build regular expressions
command-line shells, such as Bash or the Wind
expressions as part of their command syntax.

There are some variations in regular expressior
them, but for the most part regular expressions
inside bigger languages like Python. The examp
documentation in the rest of the chapter) will fo
chapter transfers easily to working with other p

As with most of this book, examples will be illus
sessions that readers can type themselves, so t
examples. However, the *re* module has little rea
illustrates matches in the shell. Therefore, the a
below is implied in the examples:

re_show.py

```
import re
def re_show(pat, s):
    print re.compile(pat, re.M).sub(

s = '''Mary had a little lamb
And everywhere that Mary
went, the lamb was sure
to go'''
```

Place the code in an external module and impo
not worry about what the above function does f
argument to re_show() will be a regular expres
a string to be matched against. The matches w
pattern for purposes of matching beginnings ar
whatever is contained between curly braces.

3.1.2 Matching Patterns in Text: The Basics

The very simplest pattern matched by a regula
sequence of literal characters. Anything in the t
characters in exactly the order listed will match
uppercase version, and vice versa. A space in a
literal space in the target (this is unlike most pr
where a variable number of spaces separate ke

```
>>> from re_show import re_show, s
```

```
>>> re_show('a', s)
M{a}ry h{a}d {a} little l{a}mb.
And everywhere th{a}t M{a}ry
went, the l{a}mb w{a}s sure
to go.

>>> re_show('Mary', s)
{Mary} had a little lamb.
And everywhere that {Mary}
went, the lamb was sure
to go.
```

A number of characters have special meanings
special meaning can be matched, but to do so i
character (this includes the backslash characte
the regular expression should include \\). In Py
available that will not perform string interpolati
same backslash-prefixed codes as do Python st
expression strings by quoting them as "raw stri

```
>>> from re_show import re_show
>>> s = '''Special characters must b
>>> re_show(r'.*', s)
{Special characters must be escaped.
```

```
>>> re_show(r'\.\*', s)
Special characters must be escaped{.

>>> re_show('\\\\', r'Python \ escap
Python {\} escaped {\} pattern

>>> re_show(r'\\', r'Regex \ escapec
Regex {\} escaped {\} pattern
```

Two special characters are used to mark the be
dollar sign ("$"). To match a caret or dollar sigr
(i.e., precede it by a backslash "\").

An interesting thing about the caret and dollar
That is, the length of the string matched by a c
rest of the regular expression can still depend c
expression tools provide another zero-width pa
be divided by whitespace like spaces, tabs, new
boundary pattern matches the actual point whe
whitespace characters.

```
>>> from re_show import re_show, s
>>> re_show(r'^Mary', s)
{Mary} had a little lamb
```

```
And everywhere that Mary
went, the lamb was sure
to go

>>> re_show(r'Mary$', s)
Mary had a little lamb
And everywhere that {Mary}
went, the lamb was sure
to go

>>> re_show(r'$','Mary had a little
Mary had a little lamb{}
```

In regular expressions, a period can stand for a
is not included, but optional switches can force
re module functions). Using a period in a patter
occurs here, without having to decide what.

Readers who are familiar with DOS command-li
filling the role of "some character" in command
question mark has a different meaning, and the

```
>>> from re_show import re_show, s
>>> re_show(r'.a', s)
{Ma}ry {ha}d{ a} little {la}mb
```

```
And everywhere t{ha}t {Ma}ry
went, the {la}mb {wa}s sure
to go
```

A regular expression can have literal characters
Each literal character or positional pattern is an
group several atoms together into a small regu
expression. One might be inclined to call such a
called an atom.

In older Unix-oriented tools like grep, subexpre
parentheses; for example, \ (Mary\). In Python
done with bare parentheses, but matching a lit
pattern.

```
>>> from re_show import re_show, s
>>> re_show(r'(Mary)( )(had)', s)
{Mary had} a little lamb
And everywhere that Mary
went, the lamb was sure
to go

>>> re_show(r'\(.*\)', 'spam (and eg
spam {(and eggs)}
```

Rather than name only a single character, a pat
a set of characters.

A set of characters can be given as a simple list
will match any single lowercase vowel. For lette
and last letter of a range, with a dash in the mi
lowercase or uppercase letter in the first half of

Python (as with many tools) provides escape-st
character class, such as \s for a whitespace cha
define these character classes with square brac
expressions more compact and more readable.

```
>>> from re_show import re_show, s
>>> re_show(r'[a-z]a', s)
Mary {ha}d a little {la}mb
And everywhere t{ha}t Mary
went, the {la}mb {wa}s sure
to go
```

The caret symbol can actually have two differen
the time, it means to match the zero-length pa
the beginning of a character class, it reverses t
not included in the listed character set is match

```
>>> from re_show import re_show, s
```

```
>>> re_show(r'[^a-z]a', s)
{Ma}ry had{ a} little lamb
And everywhere that {Ma}ry
went, the lamb was sure
to go
```

Using character classes is a way of indicating th
in a particular spot. But what if you want to spe
occur in a position in the regular expression? Fo
vertical bar ("|"). This is the symbol that is also
and is sometimes called the pipe character.

The pipe character in a regular expression indic
group enclosing it. What this means is that eve
right of a pipe character, the alternation greedil
the scope of the alternation, you must define a
may match. The example illustrates this:

```
>>> from re_show import re_show
>>> s2 = 'The pet store sold cats, c
>>> re_show(r'cat|dog|bird', s2)
The pet store sold {cat}s, {dog}s, a

>>> s3 = '=first first= # =second se
>>> re_show(r'=first|second=', s3)
```

```
{=first} first= # =second {second=}
>>> re_show(r'(=)(first)|(second)(=)
{=first} first= # =second {second=}

>>> re_show(r'=(first|second)=', s3)
=first first= # =second second= # {=
```

One of the most powerful and common things y
specify how many times an atom occurs in a co
want to specify something about the occurrenc
interested in specifying the occurrence of a cha

There is only one quantifier included with "basi
("*"); in English this has the meaning "some or
specify that any number of an atom may occur
asterisk.

Without quantifiers, grouping expressions does
can add a quantifier to a subexpression we can
subexpression as a whole. Take a look at the ex

```
>>> from re_show import re_show
>>> s = '''Match with zero in the mi
... Subexpression occurs, but...: @=
... Lots of occurrences: @=!==!==!==
```

```
... Must repeat entire pattern: @=!=
>>> re_show(r'@(=!=)*@', s)
Match with zero in the middle: {@@}
Subexpression occurs, but...: @=!=AB
Lots of occurrences: {@=!==!==!==!==
Must repeat entire pattern: @=!==!=!
```

3.1.3 Matching Patterns in Text: Intermediate

In a certain way, the lack of any quantifier sym
It says the atom occurs exactly once. Extended
numbers to "once exactly" and "zero or more ti
more times" and the question mark ("?") mean
far the most common enumerations you wind u

If you think about it, you can see that the exte
you "say" anything the basic ones do not. They
readable way. For example, (ABC)+ is equivale
equivalent to XABCY|XY. If the atoms being qua
subexpressions, the question mark and plus sig

```
>>> from re_show import re_show
>>> s = '''AAAD
... ABBBBCD
... BBBCD
... ABCCD
```

```
... AAABBBC'''
>>> re_show(r'A+B*C?D', s)
{AAAD}
{ABBBBCD}
BBBCD
ABCCD
AAABBBC
```

Using extended regular expressions, you can sp
using a more verbose syntax than the question
curly braces ("{" and "}") can surround a preci
looking for.

The most general form of the curly-brace quant
must be no larger than the second, and both m
count is specified this way to fall between the r
As shorthand, either argument may be left emp
as zero/infinity, respectively. If only one argum
that number of occurrences are matched.

```
>>> from re_show import re_show
>>> s2 = '''aaaaa bbbbb ccccc
... aaa bbb ccc
... aaaaa bbbbbbbbbbbbbb ccccc'''
>>> re_show(r'a{5} b{,6} c{4,8}', s2
```

```
{aaaaa bbbbb ccccc}
aaa bbb ccc
aaaaa bbbbbbbbbbbbbb ccccc

>>> re_show(r'a+ b{3,} c?', s2)
{aaaaa bbbbb c}cccc
{aaa bbb c}cc
{aaaaa bbbbbbbbbbbbbb c}cccc

>>> re_show(r'a{5} b{6,} c{4,8}', s2
aaaaa bbbbb ccccc
aaa bbb ccc
{aaaaa bbbbbbbbbbbbbb ccccc}
```

One powerful option in creating search patterns
matched earlier in a regular expression is matc
using backreferences. Backreferences are name
the backslash/escape character when used in tl
each successive group in the match pattern, as
backreference refers to the group that, in this e
number.

It is important to note something the example
backreference is the same literal string matche
matched the string could have matched other s

subexpression later in the regular expression d
backreference (but you have to decide what it i

Backreferences refer back to whatever occurre
order those grouped expressions occurred. Up
However, Python also allows naming backrefere
the backreferences are pointing to. The initial p
the corresponding backreference must contain

```
>>> from re_show import re_show
>>> s2 = '''jkl abc xyz
... jkl xyz abc
... jkl abc abc
... jkl xyz xyz
... '''
>>> re_show(r'(abc|xyz) \1', s2)
jkl abc xyz
jkl xyz abc
jkl {abc abc}
jkl {xyz xyz}

>>> re_show(r'(abc|xyz) (abc|xyz)',
jkl {abc xyz}
jkl {xyz abc}
jkl {abc abc}
jkl {xyz xyz}
```

```
>>> re_show(r'(?P<let3>abc|xyz) (?P=
jkl abc xyz
jkl xyz abc
jkl {abc abc}
jkl {xyz xyz}
```

Quantifiers in regular expressions are greedy. T
can.

Probably the easiest mistake to make in compo
much. When you use a quantifier, you want it t
the point where you want to finish your match.
quantifiers, it is easy to forget that the last bit
than the one you are interested in.

```
>>> from re_show import re_show
>>> s2 = '''-- I want to match the w
... -- with 'th' and end with 's'.
... this
... thus
... thistle
... this line matches too much
... '''
>>> re_show(r'th.*s', s2)
```

```
-- I want to match {the words that s
-- wi{th 'th' and end with 's}'.
{this}
{thus}
{this}tle
{this line matches} too much
```

Often if you find that regular expressions are m
reformulate the problem in your mind. Rather t
match later in the expression?" ask yourself, "W
part?" This often leads to more parsimonious pa
pattern is to use the complement operator and
think about how it works.

The trick here is that there are two different wa
Either you can think you want to keep matching
want to keep matching *unless* you get to XYZ. T

For people who have thought about basic proba
rolling a 6 on a die in one roll is $1/6$. What is the
calculation puts the odds at $1/6+1/6+1/6+1/6+1/6+1/6$, o
all, the chance after twelve rolls isn't 200 perce
avoid rolling a 6 for six rolls?" (i.e., $5/6$ x $5/6$ x $5/6$ x
of getting a 6 is the same chance as not avoidi
imagine transcribing a series of die rolls, you co
record, and similar thinking applies.

```
>>> from re_show import re_show
>>> s2 = '''-- I want to match the w
... -- with 'th' and end with 's'.
... this
... thus
... thistle
... this line matches too much
... '''
>>> re_show(r'th[^s]*.', s2)
-- I want to match {the words} {that
-- wi{th 'th' and end with 's}'.
{this}
{thus}
{this}tle
{this} line matches too much
```

Not all tools that use regular expressions allow
locate the matched pattern; the mostly widely
which is a tool for searching only. Text editors,
replacement in their regular expression search

Python, being a general programming language
accompany matches. Since Python strings are i
objects in place, but instead return the modifie
module, one can always rebind a particular vari

re modification.

Replacement examples in this tutorial will call a
module function *re.sub ()*. Original strings will b
results will appear below the call and with the s
areas as re_show() used. Be careful to notice t
will not be returned by standard *re* functions, b
import the following function in the examples b

re_new.py

```
import re
def re_new(pat, rep, s):
    print re.sub(pat, '{'+rep+'}', s
```

Let us take a look at a couple of modification ex
covered. This one simply substitutes some liter
string.replace() can achieve the same result an

```
>>> from re_new import re_new
>>> s = 'The zoo had wild dogs, bobc
>>> re_new('cat','dog',s)
The zoo had wild dogs, bob{dog}s, li
```

Most of the time, if you are using regular expre
to match more general patterns than just litera
replaced (even if it is several different strings ir

```
>>> from re_new import re_new
>>> s = 'The zoo had wild dogs, bobc
>>> re_new('cat|dog','snake',s)
The zoo had wild {snake}s, bob{snake
>>> re_new(r'[a-z]+i[a-z]*','nice',s
The zoo had {nice} dogs, bobcats, {r
```

It is nice to be able to insert a fixed string ever
frankly, doing that is not very context sensitive
fixed strings, but rather to insert something tha
patterns. Fortunately, backreferences come to c
in the pattern matches themselves, but it is eve
replacement patterns. By using replacement ba
the matched patterns to use just the parts of ir

As well as backreferencing, the examples below
regular expressions. In most programming code
examples differ solely in an extra space within
return value is importantly different.

```
>>> from re_new import re_new
>>> s = 'A37 B4 C107 D54112 E1103 XX
```

```
>>> re_new(r'([A-Z])([0-9]{2,4})',r'
{37:A} B4 {107:C} {5411:D}2 {1103:E}
>>> re_new(r'([A-Z])([0-9]{2,4}) ',r
{37:A }B4 {107:C }D54112 {1103:E }XX
```

This tutorial has already warned about the dan
expression patterns. But the danger is so much
that it is worth repeating. If you replace a patte
thought of when you composed the pattern, yo
data from your target.

It is always a good idea to try out regular expre
representative of production usage. Make sure
matching. A stray quantifier or wildcard can ma
what you thought was a specific pattern. And s
pattern for a while, or find another set of eyes,
after you see what matches. Familiarity might b
competence.

3.1.4 Advanced Regular Expression Extensio

Some very useful enhancements to basic regul
with many other tools). Many of these do not s
expressions, but they *do* manage to make expr

Earlier in the tutorial, the problems of matching

workarounds were suggested. Python is nice er
optional "non-greedy" quantifiers. These quanti
matching whatever comes next in the pattern (

Non-greedy quantifiers have the same syntax a
quantifier followed by a question mark. For exa
A[A-Z] *?B. In English, this means "match an A
are needed to find a B."

One little thing to look out for is the fact that th
capital letters. No longer matches are ever nee
pattern. If you use non-greedy quantifiers, wat
symmetric danger.

```
>>> from re_show import re_show
>>> s = '''-- I want to match the wo
... -- with 'th' and end with 's'.
... this line matches just right
... this # thus # thistle'''
>>> re_show(r'th.*s',s)
-- I want to match {the words that s
-- wi{th 'th' and end with 's}'.
{this line matches jus}t right
{this # thus # this}tle

>>> re_show(r'th.*?s',s)
```

```
-- I want to match {the words} {that
-- wi{th 'th' and end with 's}'.
{this} line matches just right
{this} # {thus} # {this}tle

>>> re_show(r'th.*?s ',s)
-- I want to match {the words }that
-- with 'th' and end with 's'.
{this }line matches just right
{this }# {thus }# thistle
```

Modifiers can be used in regular expressions or
A modifier affects, in one way or another, the ir
A modifier, unlike an atom, is global to the part
anything, it instead constrains or directs what t

When used directly within a regular expression
whole pattern, as in (?Limsux). For example, to
case of the letters, one could use (?i)cat. The s
argument as bitmasks (i.e., with a | between e
the *re* module, not to all. For example, the two

```
>>> import re
>>> re.search(r'(?Li)cat','The Cat i
4
```

```
>>> re.search(r'cat','The Cat in the
4
```

However, some function calls in *re* have no argu
either use the modifier prefix pseudo-group or
use it in string form. For example:

```
>>> import re
>>> re.split(r'(?i)th','Brillig and
['Brillig and ', 'e Sli', 'y Toves']
>>> re.split(re.compile('th',re.I),'
['Brillig and ', 'e Sli', 'y Toves']
```

See the *re* module documentation for details or

The modifiers listed below are used in *re* expres
may be accustomed to a g option for "global" m
as their default unit, and "global" means to ma
passed string as its unit, so "global" is simply t
the regular expressions have to be tailored to l
characters, or the strings being operated on sh
means.

```
* L (re.L) - Locale customization of
* i (re.I) - Case-insensitive match
```

```
* m (re.M) - Treat string as multipl
* s (re.S) - Treat string as single
* u (re.U) - Unicode customization c
* x (re.X) - Enable verbose regular
```

The single-line option ("s") allows the wildcard otherwise). The multiple-line option ("m") caus end of each line in the target, not just the begi The insensitive option ("i") ignores differences Unicode options ("L" and "u") give different inte alphanumeric ("\w") escaped patternsand their

The verbose option ("x") is somewhat different may contain nonsignificant whitespace and inlir different interpretation of regular expression pa easily readable complex patterns. Some examp

Let's take a look first at how case-insensitive ar behavior.

```
>>> from re_show import re_show
>>> s = '''MAINE # Massachusetts # C
... mississippi # Missouri # Minneso
>>> re_show(r'M.*[ise] ', s)
{MAINE # Massachusetts }# Colorado #
```

```
mississippi # {Missouri }# Minnesota

>>> re_show(r'(?i)M.*[ise] ', s)
{MAINE # Massachusetts }# Colorado #
{mississippi # Missouri }# Minnesota

>>> re_show(r'(?si)M.*[ise] ', s)
{MAINE # Massachusetts # Colorado #
mississippi # Missouri }# Minnesota
```

Looking back to the definition of re_show(), we
multiline option. So patterns displayed with re_
a couple of examples that use *re.findall()* instea

```
>>> from re_show import re_show
>>> s = '''MAINE # Massachusetts # C
... mississippi # Missouri # Minneso
>>> re_show(r'(?im)^M.*[ise] ', s)
{MAINE # Massachusetts }# Colorado #
{mississippi # Missouri }# Minnesota

>>> import re
>>> re.findall(r'(?i)^M.*[ise] ', s)
['MAINE # Massachusetts ']
>>> re.findall(r'(?im)^M.*[ise] ', s
```

```
['MAINE # Massachusetts ', 'mississi
```

Matching word characters and word boundaries
being alphanumeric. Character codepages for le
differ among national alphabets. Python versior
regular expressions can optionally use the curr

Of greater long-term significance is the *re* mod
Unicode categories of characters, and decide w
category. Locale settings work OK for European
clearer and less error prone. The "u" modifier c
are recognized or merely ASCII ones:

```
>>> import re
>>> alef, omega = unichr(1488), unic
>>> u = alef +' A b C d '+omega+' X
>>> u, len(u.split()), len(u)
(u'\u05d0 A b C d \u03c9 X y Z', 9,
>>> ':'.join(re.findall(ur'\b\w\b',
u'A:b:C:d:X:y:Z'
>>> ':'.join(re.findall(ur'(?u)\b\w\
u'\u05d0:A:b:C:d:\u03c9:X:y:Z'
```

Backreferencing in replacement patterns is very

in a complex regular expression, which can be
refer to the parts of a replacement pattern in s
re patterns allow "grouping without backreferer

A group that should not also be treated as a ba
beginning of the group, as in (?:pattern). In fac
backreferences are in the search pattern itself:

```
>>> from re_new import re_new
>>> s = 'A-xyz-37 # B:abcd:142 # C-w
>>> re_new(r'([A-Z])(?:-[a-z]{3}-)([
{A37} # B:abcd:142 # {C66} # {D93}
>>> # Groups that are not of interes
...
>>> re_new(r'([A-Z])(-[a-z]{3}-)([0-
{A-xyz-} # B:abcd:142 # {C-wxy-} # {
>>> # One could lose track of groups
...
```

Python offers a particularly handy syntax for re
than just play with the numbering of matched g
pointed out the syntax for named backreferenc
P=name). However, a bit different syntax is ne
use the \g operator along with angle brackets a

```
>>> from re_new import re_new
>>> s = "A-xyz-37 # B:abcd:142 # C-w
>>> re_new(r'(?P<prefix>[A-Z])(-[a-z
...         r'\g<prefix>\g<id>',s)
{A37} # B:abcd:142 # {C66} # D93}
```

Another trick of advanced regular expression to
similar to regular grouped subexpression, excep
There are two advantages to using lookahead a
assertion can function in a similar way to a grou
match something without counting it in backref
lookahead assertion can specify that the next c
different (more general) subexpression actually
backreferencing that other subexpression).

There are two kinds of lookahead assertions: p
positive assertion specifies that something does
something does not come next. Emphasizing th
groups, the syntax for lookahead assertions is s
and (?!pattern) for negative assertions.

```
>>> from re_new import re_new
>>> s = 'A-xyz37 # B-ab6142 # C-Wxy6
>>> # Assert that three lowercase le
...
```

```
>>> re_new(r'([A-Z]-)(?=[a-z]{3})([\
{xyz37A-} # B-ab6142 # C-Wxy66 # {qr
>>> # Assert three lowercase letts c
...
>>> re_new(r'([A-Z]-)(?![a-z]{3})([\
A-xyz37 # {ab6142B-} # {Wxy66C-} # I
```

Along with lookahead assertions, Python 2.0+ a
similara pattern is of interest only if it is (or is r
Lookbehind assertions are somewhat more rest
may only look backwards by a fixed number of
general quantifiers are allowed in lookbehind as
expressed using lookbehind assertions.

As with lookahead assertions, lookbehind asser
The former assures that a certain pattern does
the pattern *does* precede the match.

```
>>> from re_new import re_new
>>> re_show('Man', 'Manhandled by Th
{Man}handled by The {Man}
>>> re_show('(?<=The )Man', 'Manhanc
Manhandled by The {Man}

>>> re_show('(?<!The )Man', 'Manhanc
```

{Man}handled by The Man

In the later examples we have started to see ju
get. These examples are not the half of it. It is
to-understand things with regular expression (b

There are two basic facilities that Python's "ver
expressions. One is allowing regular expression
whitespace like trailing spaces and newlines). T
regular expressions. When patterns get complic

The example given is a fairly typical example o
commented, regular expression:

```
>>> from re_show import re_show
>>> s = '''The URL for my site is: h
... might also enjoy ftp://yoursite.
... place to download files.'''
>>> pat = r'''  (?x)( # verbose ider
... (http|ftp|gopher) # make sure we
...                :// # ...needs to
...         [^ \n\r]+ # some stuff t
...                 \w # URL always e
...       (?=[\s\.,]) # assert: foll
...                  ) # end of match
```

```
>>> re_show(pat, s)
The URL for my site is: {http://mysi
might also enjoy {ftp://yoursite.com
place to download files.
```

3.2 Some Common Tasks

3.2.1 Problem: Making a text block flush left

For visual clarity or to identify the role of text,
prose-oriented documents (but log files, configu
initial fields). For downstream purposes, indent
incorrect, since the indentation is not part of th
However, it often makes matters even worse to
indented textsimply remove leading whitespace
decoration, the relative indentations of lines wi
functions (for example, the blocks of text migh

The general procedure you need to take in max
But it is easy to throw more code at it than is n
nested loops of *string.find()* and *string.replace(*
regular expressionscombined with the concisen
give you a quick, short, and direct transformati

flush_left.py

```
# Remove as many leading spaces as p
from re import findall,sub
# What is the minimum line indentati
indent = lambda s: reduce(min,map(le
# Remove the block-minimum indentati
flush_left = lambda s: sub('(?m)^ {%

if __name__ == '__main__':
    import sys
    print flush_left(sys.stdin.read(
```

The flush_left() function assumes that blocks a
combined with spacesan initial pass through th
$PYTHONPATH/tools/scripts/) can convert block

A helpful adjunct to flush_left() is likely to be th
Chapter 2, Problem 2. Between the two of thes
"batch-oriented word processor." (What other c

3.2.2 Problem: Summarizing command-line o

Documentation of command-line options to pro
places like manpages, docstrings, READMEs an

expect to see command-line options indented a
by one or more lines of description, and usually
users browsing documentation, but is of sufficie
expressions are well suited to finding the right

A specific scenario where you might want a sur
understanding configuration files that call multi
Unix-like systems is a good example of such a
themselves often have enough complexity and
have difficulty parsing them.

The utility below will look for every service laur
summary documentation of all the options usec

show_services.py

```
import re, os, string, sys

def show_opts(cmdline):
    args = string.split(cmdline)
    cmd = args[0]
    if len(args) > 1:
        opts = args[1:]
    # might want to check error outp
    (in_, out_, err) = os.popen3('ma
```

```
    manpage = out_.read()
    if len(manpage) > 2:      # foun
        print '\n%s' % cmd
        for opt in opts:
            pat_opt = r'(?sm)^\s*'+c
            opt_doc = re.search(pat_
            if opt_doc is not None:
                print opt_doc.group(
            else:               # try
                mentions = []
                for para in string.s
                   if re.search(opt,
                       mentions.appe
                if not mentions:
                   print '\n    ',op
                else:
                   print '\n    ',op
                   print '\n'.join(m
     else:                    # no m
         print cmdline
         print '    No documentatio

def services(fname):
    conf = open(fname).read()
    pat_srv = r'''(?xm)(?=^[^#])
```

```
                          (?:(?:[\w/]+\s+){6
                          (.*$)
    return re.findall(pat_srv, conf)

if __name__ == '__main__':
    for service in services(sys.argv
        show_opts(service)
```

The particular tasks performed by show_opts()
systems, but the general techniques are more l
comment character and number of fields in /etc
scripts, but the use of regular expressions to fi
If the man and col utilities are not on the releva
such as reading in the docstrings from Python m
the samples in $PYTHONPATH/tools/ use compa

Another thing worth noting is that even where
data, you need not do everything with regular e
identify paragraphs in show_opts() is still the q
re.split() could do the same thing.

Note: Along the lines of paragraph splitting, he
expression that matches every whole paragraph
For purposes of the puzzle, assume that a para
doubled newlines ("\n\n").

3.2.3 Problem: Detecting duplicate words

A common typo in prose texts is doubled words
except in those few cases where they are inten
programming language code, configuration files
suited to detecting this occurrence, which just a
easy to wrap the regex in a small utility with a

dupwords.py

```
# Detect doubled words and display w
# Include words doubled across lines

import sys, re, glob
for pat in sys.argv[1:]:
    for file in glob.glob(pat):
        newfile = 1
        for para in open(file).read(
            dups = re.findall(r'(?m)
            if dups:
                if newfile:
                    print '%s\n%s\n'
                    newfile = 0
                for dup in dups:
                    print '[%s] -->'
```

This particular version grabs the line or lines or

context (along with a prompt for the duplicate i
assumption made by dupwords.py is that a dou
to the beginning of another, ignoring whitespac
paragraphs is not likewise noteworthy.

3.2.4 Problem: Checking for server errors

Web servers are a ubiquitous source of informa
documents is largely hit-or-miss. Every Web ma
month or two, thereby breaking bookmarks and
surfers, it is worse for robots faced with the dif
content and errors. By-the-by, it is easy to accu
error messages rather than desired content.

In principle, Web servers can and should return
practice, Web servers almost always return dyn
requests. Such pages are basically perfectly no
like "Error 404: File not found!" Most of the tim
containing custom graphics and layout, links to
tags, and all sorts of other stuff. It is actually q
send in response to requests for nonexistent UI

Below is a very simple Python script to examine
requests. Getting an error page is usually as si
http://somewebsite.com/phony-url or the like (
discussed in Chapter 5, but its details are not ir

url_examine.py

```
import sys
from urllib import urlopen

if len(sys.argv) > 1:
    fpin = urlopen(sys.argv[1])
    print fpin.geturl()
    print fpin.info()
    print fpin.read()
else:
    print "No specified URL"
```

Given the diversity of error pages you might re
regular expression (or any program) that deter
document is an error page. Furthermore, some
quite errors, but not really quite content either
suggestions on how to get to content). But som
content from errors. One noteworthy heuristic i
404 or 403 (not a sure thing, but good enough
the "error probability" of HTML documents:

error_page.py

```
import re, sys
page = sys.stdin.read()

# Mapping from patterns to probabili
err_pats = {r'(?is)<TITLE>.*?(404|40
            r'(?is)<TITLE>.*?ERROR.*
            r'(?is)<TITLE>ERROR</TIT
            r'(?is)<TITLE>.*?ERROR.*
            r'(?is)<META .*?(404|403
            r'(?is)<META .*?ERROR.*?
            r'(?is)<TITLE>.*?File No
            r'(?is)<TITLE>.*?Not Fou
            r'(?is)<BODY.*(404|403).
            r'(?is)<H1>.*?(404|403).
            r'(?is)<BODY.*not found.
            r'(?is)<H1>.*?not found.
            r'(?is)<BODY.*the reques
            r'(?is)<BODY.*the page y
            r'(?is)<BODY.*page.{1,50
            r'(?is)<BODY.*request.{1
            r'(?i)does not exist': 0
           }
err_score = 0
for pat, prob in err_pats.items():
    if err_score > 0.9: break
```

```
    if re.search(pat, page):
        # print pat, prob
        err_score += prob

if err_score > 0.90:   print 'Page i
elif err_score > 0.75: print 'It is
elif err_score > 0.50: print 'Better
elif err_score > 0.25: print 'Fair i
else:                 print 'Page is
```

Tested against a fair number of sites, a collecti
threshold confidences works quite well. Within
an error page, erro_page.py has gotten no fals
lowest warning level for every true error page.

The patterns chosen are all fairly simple, and b
determined entirely subjectively by the author.
technique can be used to solve many "fuzzy lo
with Web server errors).

Code like that above can form a general approa
is worth, the scripts url_examine.py and error_
from the first to the second. For example:

```
% python urlopen.py http://gnosis.cx
Page is almost surely an error repor
```

3.2.5 Problem: Reading lines with continuatio

Many configuration files and other types of com
facility to treat multiple lines as if they were a s
usually desirable as a first step to turn all these
(or more likely, to transform both single and co
iterate through later). A continuation character
before a newline, or possibly the last thing othe
partial) table of continuation characters used by
below:

```
\ Python, JavaScript, C/C++, Bash, T
_ Visual Basic, PAW
& Lyris, COBOL, IBIS
; Clipper, TOP
- XSPEC, NetREXX
= Oracle Express
```

Most of the formats listed are programming lan
than just identifying the lines. More often, it is
interest in simple parsing, and most of the time
of using trailing backslashes for continuation lin

One *could* manage to parse logical lines with a
and performed concatenations when needed. B
problem to a single regular expression. The mo

logical_lines.py

```
# Determine the logical lines in a f
# continuation characters.  'logical
# list.  The self-test prints the lo
# physical lines (for all specified 

import re

def logical_lines(s, continuation='\
    c = continuation
    if strip_trailing_space:
        s = re.sub(r'(?m)(%s)(\s+)$'
    pat_log = r'(?sm)^.*?$(?<!%s)'%[
    return [t.replace(c+'\n','') for

if  __name__ == '__main__':
     import sys
     files, strip, contin = ([], 0, 
     for arg in sys.argv[1:]:
         if arg[:-1] == '--continue=
         elif arg[:-1] == '-c': cont
         elif arg in ('--string','-s
         else: files.append(arg)
```

```
    if not files: files.append(sys.
    for file in files:
        s = open(sys.argv[1]).read(
        print '\n'.join(logical_lir
```

The comment in the pat_log definition shows a
times. The comment is the pattern that is used
dense as it is with symbols, you can still read it
version of the same line with the verbose modi

```
>>> pat = r'''
... (?x)     # This is the verbose ve
... (?s)     # In the pattern, let ".
... (?m)     # Allow ^ and $ to match
... ^        # Start the match at the
.... *?      # Non-greedily grab ever
...          # where the rest of the
... $        # End the match at an er
... (?<!     # Only count as a match
...          # the immediately last t
... \\)      # It wasn't an (escaped)
```

3.2.6 Problem: Identifying URLs and email ad

A neat feature of many Internet and news clien

that the applications can act upon. For URL reso
"clickable"; for an email address it usually mea
address. Depending on the nature of an applica
each identified resource. For a text processing a
something more batch-oriented: extraction, tra

Fully and precisely implementing RFC1822 (for
possible within regular expressions. But doing s
needed to identify 99% of resources. Moreover,
world" are not strictly compliant with the releva
"almost correct" resource identifiers. The utility
balance of other well-implemented and practica
intended to look like a resource, and *almost* no

find_urls.py

```
# Functions to identify and extract

import re, fileinput

pat_url = re.compile(  r'''
                 (?x)( # verbose ide
     (http|ftp|gopher) # make sure w
                   :// # ...needs to
        (\w+[:.]?){2,} # at least tw
```

```
                 (/?| # could be ju
            [^ \n\r"]+ # or stuff th
               [\w/]) # resource na
       (?=[\s\.,>)'"\]]) # assert: fol
                     ) # end of matc
                       ''')
pat_email = re.compile(r'''
               (?xm)  # verbose ide
            (?=^.{11} # Mail header
       (?<!Message-ID:|  # rule out Me
          In-Reply-To))  # ...and also
              (.*?)(  # must grab t
    ([A-Za-z0-9-]+\.)?  # maybe an in
         [A-Za-z0-9-]+  # definitely
                    @  # ...needs an
          (\w+\.?){2,}  # at least tw
     (?=[\s\.,>)'"\]])  # assert: fol
                     ) # end of matc
                       ''')
extract_urls = lambda s: [u[0] for u
extract_email = lambda s: [(e[1]) fo

if __name__ == '__main__':
    for line in fileinput.input():
        urls = extract_urls(line)
```

```
        if urls:
            for url in urls:
                print fileinput.file
        emails = extract_email(line)
        if emails:
            for email in emails:
                print fileinput.file
```

A number of features are notable in the utility a
done within the regular expressions themselves
extract_email() are each a single line, using the
especially list comprehensions (four or five line
style helps emphasize where the work is done)
STDOUT, but you could do something else with

A bit of testing of preliminary versions of the re
complications to them. In part this lets readers
greater part, this helps weed out what I would
least two domain groupsthis rules out LOCALHO
colon to end a domain group, we allow for spec
http://gnosis.cx:8080/resource/.

Email addresses have one particular special con
addresses happen to be actual mail archives, yo
these headers is very similar to that of email ad
Message-IDs). By combining a negative look-be
can make sure that everything that gets extrac

little complicated to combine these things corre

3.2.7 Problem: Pretty-printing numbers

In producing human-readable documents, Pyth leaves something to be desired. Specifically, th of 1,000 in written large numerals are not prod reading large numbers difficult. For example:

```
>>> budget = 12345678.90
>>> print 'The company budget is $%s
The company budget is $12345678.9
>>> print 'The company budget is %10
The company budget is 12345678.90
```

Regular expressions can be used to transform r alternative would be to process numeric values stringifying the chunks). A few basic utility func

pretty_nums.py

```
# Create/manipulate grouped string v

import re
```

```
def commify(f, digits=2, maxgroups=5
    template = '%%1.%df' % digits
    s = template % f
    pat = re.compile(r'(\d+)(\d{3})(
    if european:
        repl = r'\1.\2\3\4'
    else:   # could also use locale.
        repl = r'\1,\2\3\4'
    for i in range(maxgroups):
        s = re.sub(pat,repl,s)
    return s

def uncommify(s):
    return s.replace(',','')

def eurify(s):
    s = s.replace('.','\000')    # pl
    s = s.replace(',','.')       # ch
    s = s.replace('\000',',')    # de
    return s

def anglofy(s):
    s = s.replace(',','\000')    # pl
    s = s.replace('.',',')       # ch
    s = s.replace('\000','.')    # de
```

```
    return s

vals = (12345678.90, 23456789.01, 34
sample = '''The company budget is $%
Its debt is $%s, against assets
of $%s'''

if __name__ == '__main__':
     print sample % vals, '\n-----'
     print sample % tuple(map(commif
     print eurify(sample % tuple(map
```

The technique used in commify() has virtues ar
slightly kludgey inasmuch as it loops through th
argument, it is no good for numbers bigger tha
smaller than this). If purity is a goaland it prob
with a single regular expression to do the whol
the "place holder" idea that was mentioned in t

3.3 Standard Modules

3.3.1 Versions and Optimizations

> Rules of Optimization:
>
> Rule 1: Don't do it.
>
> Rule 2 (for experts only): Don't do it yet.
>
> M.A. Jackson

Python has undergone several changes in its re
by *pre* in Python 1.5; *pre*, in turn, by *sre* in Pyt
include the older modules in its standard library
deprecated when the newer versions are includ
served as a wrapper to the underlying regular e
Python 2.0+ has used *re* to wrap *sre*, *pre* is stil
underlying *pcre* C extension module that can te

Each version has generally improved upon its p
as regular expressions there are always a few l
Unicode support and is faster for most operatio
insensitive searches. Subtle details of regular e
regex module perform faster than the newer or
be extremely complicated and dependent upon

Readers might start to feel their heads swim wi
out of historic interest, you really do not need t
regular expression support. The simple rule is j
what it wrapsthe interface is compatible betwee

The real virtue of regular expressions is that th
cryptic) description of complex patterns in text.
are *fast enough*; there is rarely any point in opt
does what it needs to do fast enough that spee
"We should forget about small efficiencies, say
is the root of all evil." ("Computer Programming
Lecture Notes Number 27, Stanford University
Information, 1992).

In case regular expression operations prove to
bottleneck in an application, there are four step
these in order:

1. **Think about whether there is a way to s
Most especially, is it possible to reduce
pattern matching? You should always te**

however; performance characteristics r

- Consider whether regular expressions are *rea* surprising frequency, faster and simpler operati other modules) do what needs to be done. Actu first one.

- Write the search or transformation in a faster Low-level modules will inevitably involve more about the problem. But order-of-magnitude spe

- Code the application (or the relevant parts of is the absolutely first consideration in an applic Tools like swigwhile outside the scope of this bo modules to perform bottleneck operations. The *must* be solved with regular expressions that P means).

3.3.2 Simple Pattern Matching

fnmatch • Glob-style pattern matching

The real purpose of the *fnmatch* module is to m *fnmatch* is used indirectly through the *glob* mo files (for example to process each matching file about filesystems, it simply provides a way of c

language used by *fnmatch* is much simpler thai
bad, depending on your needs. As a plus, most
or Unix command line is already familiar with tl
shell-style expansions.

Four subpatterns are available in *fnmatch* patte
grouping and no quantifiers. Obviously, the disc
than with *re*. The subpatterns are as follows:

Glob-style subpatterns

```
*       Match everything that follows
?       Match any single character.
[set]   Match one character from a se
        follows the same rules as a r
        character class.  It may incl
        and zero or more enumerated c
[!set] Match any one character that
```

A pattern is simply the concatenation of one or

FUNCTIONS

fnmatch.fnmatch(s, pat)

Test whether the pattern pat matches the strin
case-insensitive. A cross-platform script should
match actual filenames.

```
>>> from fnmatch import fnmatch
>>> fnmatch('this', '[T]?i*')  # On
0

>>> fnmatch('this', '[T]?i*')  # On
1
```

SEE ALSO: fnmatch.fnmatchcase() *233*;

fnmatch.fnmatchcase(s, pat)

Test whether the pattern pat matches the strin
platform.

```
>>> from fnmatch import fnmatchcase
>>> fnmatchcase('this', '[T]?i*')
0
>>> from string import upper
>>> fnmatchcase(upper('this'), upper
1
```

SEE ALSO: fnmatch.fnmatch() *233*;

fnmatch.filter(lst, pat)

Return a new list containing those elements of
fnmatch.fnmatch() rather than like *fnmatch.fnr*
dependent. The example below shows a (slowe
on all platforms.

```
>>> import fnmatch              # Assumi
>>> fnmatch.filter(['This','that','c
['This', 'thing']
>>> fnmatch.filter(['This','that','c
['that', 'other', 'thing']
>>> from fnmatch import fnmatchcase
>>> mymatch = lambda s: fnmatchcase(
>>> filter(mymatch, ['This','that','
['that', 'other', 'thing']
```

For an explanation of the built-in function *filter*

SEE ALSO: fnmatch.fnmatch() *233;* fnmatch.fn

SEE ALSO: glob *64;* re *236;*

3.3.3 Regular Expression Modules

pre • Pre-sre module

pcre • Underlying C module for pre

The Python-written module *pre*, and the C-writt
regular expression engine, are the regular expr
backwards compatibility, they continue to be in
space of *pre* is intended to be equivalent to imp
Python 2.0+, with the exception of the handlin
is, the lines below are almost equivalent, other
specific operations:

```
>>> import pre as re
>>> import re
```

However, there is very rarely any reason to use
pre should know far more about the internals o
this book. Of course, prior to Python 2.0, impor
Python wrappers later renamed *pre*).

SEE ALSO: re *236;*

reconvert • Convert [regex] patterns to [r

This module exists solely for conversion of old r
1.5 versions of Python, or possibly from regular
sed, awk, or grep. Conversions are not guarant
a starting point for a code update.

FUNCTIONS

reconvert.convert(s)

Return as a string the modern *re*-style pattern
passed in argument s. For example:

```
>>> import reconvert
>>> reconvert.convert(r'\<\(cat\|dog
'\\b(cat|dog)\\b'
>>> import re
>>> re.findall(r'\b(cat | dog)\b', "
['dog']
```

SEE ALSO: regex *235;*

regex • Deprecated regular expression mc

The *regex* module is distributed with recent Pyt
compatibility of scripts. Starting with Python 2.
DeprecationWarning:

```
% python -c "import regex"
-c:1:  DeprecationWarning: the regex
please use the re module
```

For all users of Python 1.5+, *regex* should not l
to convert its usage to *re* calls.

SEE ALSO: reconvert *235;*

sre • Secret Labs Regular Expression Engi

Support for regular expressions in Python 2.0+
simply wraps *sre* in order to have a backwards-
almost never be any reason to import *sre* itself
deprecate *sre* also. As with *pre*, anyone decidin
the internals of regular expression engines thar

SEE ALSO: re *236;*

PATTERN SUMMARY

Figure 3.1 lists regular expression patterns; fol
more detailed explanation of patterns in action,
in this chapter. The utility function re_show() d
descriptions.

Figure 3.1. Regular

Summary of Regular Expression Patterns

Atoms	
Plain symbol:	...
Escape:	\
Grouping operators:	( )
Backreference:	\#, \##
Character class:	[ ]
Digit character class:	\d
Non-digit character class:	\D
Alphanumeric char class:	\w
Non-alphanum char class:	\W
Whitespace char class:	\s
Non-whitespace char class:	\S
Wildcard character:	.
Beginning of line:	^
Beginning of string:	\A
End of line:	$
End of string:	\Z
Word boundary:	\b
Non-word boundary:	\B
Alternation operator:	|

Constants	
re.IGNORECASE	re.I
re.LOCALE	re.L
re.MULTILINE	re.M
re.DOTALL	re.S
re.UNICODE	re.U
re.VERBOSE	re.X

Quantifiers	
Universal quantifier:	*
Non-greedy universal quantifier:	*?
Existential quantifier:	+
Non-greedy existential quantifier:	+?
Potentiality quantifier:	?
Non-greedy potentiality quantifier:	??
Exact numeric quantifier:	{*num*}
Lower-bound quantifier:	{*min*,}
Bounded numeric quantifier:	{*min*,*max*}
Non-greedy bounded quantifier:	{*min*,*max*}?

Group-Like Patterns	
Pattern modifiers:	(?Limsux)
Comments:	(?#...)
Non-backreferenced atom:	(?:...)
Positive Lookahead assertion:	(?=...)
Negative Lookahead assertion:	(?!...)
Positive Lookbehind assertion:	(?<=...)
Negative Lookbehind assertion:	(?<!...)
Named group identifier:	(?P<*name*>)
Named group backreference:	(?P=*name*)

ATOMIC OPERATORS

Plain symbol

Any character not described below as having a target string. An "A" matches exactly one "A" i

Escape: "\"

The escape character starts a special sequence
summary must be escaped to be treated as lite
character itself). The letters "A", "b", "B", "d", '
patterns if preceded by an escape. The escape
group with up to two decimal digits. The escape
special escaped meaning.

Since Python string escapes overlap regular ex
strings for regular expressions that potentially

```
>>> from re_show import re_show
>>> re_show(r'\$ \\ \^', r'\$ \\ \^
\$ \\ \^ {$ \ ^}

>>> re_show(r'\d \w', '7 a 6 # ! C')
{7 a} 6 # ! C
```

Grouping operators: "(", ")"

Parentheses surrounding any pattern turn that
pattern). Quantifiers refer to the immediately p
the preceding character or character class. For

```
>>> from re_show import  re_show
>>> re_show(r'abc+', 'abcabc abc abc
{abc}{abc} {abc} {abccc}
```

```
>>> re_show(r'(abc)+', 'abcabc abc a
{abcabc} {abc}  {abc}cc
```

Backreference: "\d", "\dd"

A backreference consists of the escape characte
digit in a back reference may not be a zero. A b
by an earlier group, where the enumeration of

```
>>> from re_show import  re_show
>>> re_show(r'([abc])(.*)\1',  'all
{all the boys a}re coy
```

An attempt to reference an undefined group wi

Character classes: "[", "]"

Specify a set of characters that may occur at a
enumerated with no delimiter. Predefined chara
custom character classes. A range of characters
are allowed within a class. If a dash is meant to
should occur as the first listed character. A char
it with a caret ("^"). If a caret is meant to be in

occur in a noninitial position. Most special chara
meaning inside a character class and are merel
"\", and "-" should be escaped with a backslash

```
>>> from re_show import re_show
>>> re_show(r'[a-fA-F]', 'A X c G')
{A} X {c} G

>>> re_show(r'[-A$BC\]]', r'A X - \
{A} X {-} \ {]} [ {$}

>>> re_show(r'[^A-Fa-f]', r'A X c G'
A{ }{X}c{ }{G}
```

Digit character class: "\d"

The set of decimal digits. Same as "*0-9*".

Non-digit character class: "\D"

The set of all characters *except* decimal digits.

Alphanumeric character class: "\w"

The set of alphanumeric characters. If re.LOCA
the same as [a-zA-ZO-9_]. Otherwise, the set i
appropriate to the locale or with an indicated U

Non-alphanumeric character class: "\W"

The set of nonalphanumeric characters. If re.L(
this is the same as [^a-zA-ZO-9_]. Otherwise,
indicated by the locale or Unicode character pro

Whitespace character class: "\s"

The set of whitespace characters. Same as [\t\

Non-whitespace character class: "\S"

The set of nonwhitespace characters. Same as

Wildcard character: "."

The period matches any single character at a p
will match a newline. Otherwise, it will match a

Beginning of line: "^"

The caret will match the beginning of the targe
"^" will match the beginning of each line withir

Beginning of string: "\A"

The "\A" will match the beginning of the target
specified, "\A" behaves the same as "^". But ev
the beginning of the entire target.

End of line: "$"

The dollar sign will match the end of the target
"$" will match the end of each line within the ta

End of string: "\Z"

The "\Z" will match the end of the target string
"\Z" behaves the same as "$". But even if the r
the entire target.

Word boundary: "\b"

The "\b" will match the beginning or end of a w
alphanumeric characters according to the curre
width match.

Non-word boundary: "\B"

The "\B" will match any position that is *not* the
defined as a sequence of alphanumeric characte
and "$", "\B" is a zero-width match.

Alternation operator: " |"

The pipe symbol indicates a choice of multiple a
groups) separated by a pipe will match. For exa

```
>>> from re_show import re_show
>>> re_show(r'A|c|G', r'A X c G')
{A} X {c} {G}

>>> re_show(r'(abc)|(xyz)', 'abc efg
{abc} efg {xyz} lmn
```

QUANTIFIERS

Universal quantifier: "*"

Match zero or more occurrences of the precedir
empty string. For example:

```
>>> from re_show import re_show
>>> re_show('a* ', ' a aa aaa aaaa b
{ }{a }{aa }{aaa}{aaaa }b
```

Non-greedy universal quantifier: "*?"

Match zero or more occurrences of the precedir
allowable. For example:

```
>>> from re_show import re_show
>>> re_show('<.*>', '<> <tag>Text</t
{<> <tag>Text</tag>}

>>> re_show('<.*?>', '<> <tag>Text</
{<>} {<tag>}Text{</tag>}
```

Existential quantifier: "+"

Match one or more occurrences of the precedin
target string to satisfy the "+" quantifier. For ex

```
>>> from re_show import re_show
>>> re_show('a+ ', ' a aa aaa aaaa b
 {a }{aa }{aaa }{aaaa }b
```

Non-greedy existential quantifier: "+?"

Match one or more occurrences of the precedin
allowable. For example:

```
>>> from re_show import  re_show
>>> re_show('<.+>', '<>  <tag>Text</
{<> <tag>Text</tag>}

>>> re_show('<.+?>', '<> <tag>Text</
{<> <tag>}Text{</tag>}
```

Potentiality quantifier: "?"

Match zero or one occurrence of the preceding empty string. For example:

```
>>> from re_show import  re_show
>>> re_show('a? ', ' a  aa aaa aaaa
{ }{a }a{a }aa{a }aaa{a  }b
```

Non-greedy potentiality quantifier: "??"

Match zero or one occurrence of the preceding

```
>>> from re_show import re_show
>>> re_show(' a?', ' a aa aaa aaaa b
{ a}{ a}a{ a}aa{ a}aaa{ }b

>>> re_show(' a??', ' a aa aaa aaaa
{ }a{ }aa{ }aaa{ }aaaa{ }b
```

Exact numeric quantifier: "{num}"

Match exactly num occurrences of the precedin

```
>>> from re_show import re_show
>>> re_show('a{3} ', ' a aa aaa aaaa
```

```
 a aa {aaa }a{aaa }b
```

Lower-bound quantifier: "{min,}"

Match *at least* min occurrences of the precedin

```
>>> from re_show import re_show
>>> re_show('a{3,} ', ' a aa aaa aaa
 a aa {aaa }{aaaa }b
```

Bounded numeric quantifier: "{min,max}"

Match *at least* min and *no more than* max occu

```
>>> from re_show import re_show
>>> re_show('a{2,3} ', ' a aa aaa aa
 a {aa }{aaa }a{aaa }
```

Non-greedy bounded quantifier: "{min,max}?

Match *at least* min and *no more than* max occu as few occurrences as allowable. Scanning is fr produced in terms of right-side groupings. For

```
>>> from re_show import re_show
>>> re_show(' a{2,4}?', ' a aa aaa a
 a{ aa}{ aa}a{ aa}aa b

>>> re_show('a{2,4}? ', ' a aa aaa a
 a {aa }{aaa }{aaaa }b
```

GROUP-LIKE PATTERNS

Python regular expressions may contain a num matches in some manner. With the exception o in backreferencing. All pseudo-group patterns h

Pattern modifiers: "(?Limsux)"

The pattern modifiers should occur at the very or more letters in the set "Limsux" may be inclu interpretation of the pattern is changed globally or the tutorial for details.

Comments: "(?#...)"

Create a comment inside a pattern. The comme

no effect on what is matched. In most cases, u
formatted comments than does "(?#...)".

```
>>> from re_show import re_show
>>> re_show(r'The(?#words in caps) C
{The Cat} in the Hat
```

Non-backreferenced atom: "(?:...)"

Match the pattern "...", but do not include the r
Moreover, methods like *re.match.group ()* will r
atom.

```
>>> from re_show import re_show
>>> re_show(r'(?:\w+) (\w+).* \1', '
{abc xyz xyz} abc

>>> re_show(r'(\w+) (\w+).* \1', 'ab
{abc xyz xyz abc}
```

Positive Lookahead assertion: "(?=...)"

Match the entire pattern only if the subpattern
substring matched by "..." as part of the match

same characters, or some of them).

```
>>> from re_show import re_show
>>> re_show(r'\w+ (?=xyz)', 'abc xyz
{abc }{xyz }xyz abc
```

Negative Lookahead assertion: "(?!...)"

Match the entire pattern only if the subpattern

```
>>> from re_show import re_show
>>> re_show(r'\w+ (?!xyz)', 'abc xyz
abc xyz {xyz }abc
```

Positive Lookbehind assertion: "(?< =...)"

Match the rest of the entire pattern only if the s
current match point. But do not include the targ
match (the same characters may or may not be
pattern). The pattern "..." must match a fixed n
general quantifiers.

```
>>> from re_show import re_show
>>> re_show(r'\w+(?<=[A-Z]) ', 'Word
```

```
Words {THAT }end in {capS }X
```

Negative Lookbehind assertion: "(?<!...)"

Match the rest of the entire pattern only if the s
to the current match point. The same character
group(s) in the entire pattern. The pattern "..."
therefore not contain general quantifiers.

```
>>> from re_show import re_show
>>> re_show(r'\w+(?<![A-Z]) ', 'Worc
{Words }THAT {end }{in }capS X
```

Named group identifier: "(?P<name>)"

Create a group that can be referred to by the n
backreferences. The forms below are equivalen

```
>>> from re_show import re_show
>>> re_show(r'(\w+) (\w+).* \1', 'ak
{abc xyz xyz abc}

>>> re_show(r'(?P<first>\w+) (\w+).*
{abc xyz xyz abc}
```

```
>>> re_show(r'(?P<first>\w+) (\w+).*
{abc xyz xyz abc}
```

Named group backreference: "(?P=name)"

Backreference a group by the name name rath
name must have been defined earlier by (?P<n

CONSTANTS

A number of constants are defined in the *re* mo
These constants are independent bit-values, so
bitwise disjunction of modifiers. For example:

```
>>> import re
>>> c = re.compile('cat | dog', re.I
```

re.I, re.IGNORECASE

Modifier for case-insensitive matching. Lowerca
patterns modified with this modifier. The prefix
achieve the same effect.

re.L, re.LOCALE

Modifier for locale-specific matching of \w, \W,
inside the pattern to achieve the same effect.

re.M, re.MULTILINE

Modifier to make ^ and $ match the beginning
string rather than the beginning and end of the
used inside the pattern to achieve the same eff

re.S, re.DOTALL

Modifier to allow . to match a newline character
newline characters. The prefix (?s) may also be
effect.

re.U, re.UNICODE

Modifier for Unicode-property matching of \w, \
The prefix (?u) may also be used inside the pat

re.X, re.VERBOSE

Modifier to allow patterns to contain insignificar
significantly improve readability of patterns. Th
to achieve the same effect.

re.engine

The regular expression engine currently in use.
normally is set to the string sre. The presence a
make sure which underlying implementation is

FUNCTIONS

For all *re* functions, where a regular expression
either a compiled regular expression or a string

re.escape(s)

Return a string with all nonalphanumeric chara
conversion makes an arbitrary string suitable fo
all literals in original string).

```
>>> import  re
>>> print  re.escape("(*@&^$@|")
\(\*\@\&\^\$\@\|
```

re.findall(pattern=..., string=...)

Return a list of all nonoverlapping occurrences
groups, return a list of tuples where each tuple
matches are included in the returned list, if the

```
>>> import re
>>> re.findall(r'\b[a-z]+\d+\b', 'ab
['abc123', 'xyz666', 'def77']
>>> re.findall(r'\b([a-z]+)(\d+)\b',
[('abc', '123'), ('xyz', '666'), ('c
```

SEE ALSO: re.search() *249;* mx.TextTools.finda

re.purge()

Clear the regular expression cache. The *re* moc
expression patterns. The number of patterns ca
recent versions generally keeping 100 items in
it is flushed automatically. You could use *re.pur*

However, such tuning is approximate at best: P
off explicitly compiled with *re.compile()* and the

re.split(pattern=..., string=...[,maxsplit=0])

Return a list of substrings of the second argum
regular expression that delimits the substrings.
included in the resultant list. Otherwise, those s
only the substrings between occurrences of pat

If the third argument maxsplit is specified as a
are parsed into the list, with any leftover conta

```
>>> import re
>>> re.split(r'\s+', 'The Cat in the
['The', 'Cat', 'in', 'the', 'Hat']
>>> re.split(r'\s+', 'The Cat in the
['The', 'Cat', 'in', 'the Hat']
>>> re.split(r'(\s+)', 'The Cat in t
['The', ' ', 'Cat', ' ', 'in', ' ',
>>> re.split(r'(a)(t)', 'The Cat in
['The C', 'a', 't', ' in the H', 'a'
>>> re.split(r'a(t)', 'The Cat in th
['The C', 't', ' in the H', 't', '']
```

SEE ALSO: string.split() *142*;

re.sub(pattern=..., repl=..., string=...[,count=0]

Return the string produced by replacing every r
pattern with the second argument repl in the th
count is specified, no more than count replacer

The second argument repl is most often a regu
Backreferences to groups matched by pattern r
backreferences using the usual escaped numbe
may also be referred to using the form \g<nam
pat). As well, enumerated backreferences may
\g<num>, where num is an integer between 1

```
>>> import re
>>> s = 'abc123 xyz666 lmn-11 def77'
>>> re.sub(r'\b([a-z]+)(\d+)', r'\2\
'123abc : 666xyz : lmn-11 77def :'
>>> re.sub(r'\b(?P<lets>[a-z]+)(?P<r
'123abc : 666xyz : lmn-11 77def :'
>>> re.sub('A', 'X', 'AAAAAAAAAA', c
'XXXXAAAAAA'
```

A variant manner of calling *re.sub ()* uses a fur
a callback function should take a MatchObject a

function is invoked for each match of pattern, a
result for whatever pattern matched. For exam

```
>>> import re
>>> sub_cb = lambda pat: '('+'len(pa
>>> re.sub(r'\w+', sub_cb, 'The leng
'(3)The (6)length (2)of (4)each (4)w
```

Of course, if repl is a function object, you can t
instead of) simply returning modified strings. F

```
>>> import re
>>> def side_effects(match):
...     # Arbitrarily complicated be
...     print len(match.group()), ma
...     return match.group()  # unch
...
>>> new = re.sub(r'\w+', side_effect
3 The
6 length
2 of
4 each
4 word
>>> new
'The length of each word'
```

Variants on callbacks with side effects could be
principle, a parser and execution environment f
contained in the callback function, for example)

SEE ALSO: string.replace() *139;*

re.subn(pattern=..., repl=..., string=...[,count=0

Identical to *re.sub () ,* except return a 2-tuple w
replacements made.

```
>>> import re
>>> s = 'abc123 xyz666 lmn-11 def77'
>>> re.subn(r'\b([a-z]+)(\d+)', r'\2
('123abc : 666xyz : lmn-11 77def :',
```

SEE ALSO: re.sub() *246*;

CLASS FACTORIES

As with some other Python modules, primarily c
classes that can be specialized. Instead, *re* has
objects. The practical difference is small for mo
attributes of returned instances in the same ma

re.compile(pattern=...[,flags=...])

Return a PatternObject based on pattern string
specified, use the modifiers indicated by flags.
string as an argument to *re* functions. However
application should be compiled in advance to as
execution. Moreover, a compiled PatternObject
achieve effects equivalent to *re* functions, but w
contexts. For example:

```
>>> import re
    >>> word = re.compile('[A-Za-z]+
>>> word.findall('The Cat in the Hat
['The', 'Cat', 'in', 'the', 'Hat']
>>> re.findall(word, 'The Cat in the
['The', 'Cat', 'in', 'the', 'Hat']
```

re.match(pattern=..., string=...[,flags=...])

Return a MatchObject if an initial substring of th
in the first argument pattern. Otherwise return
of methods and attributes to manipulate the ma
itself a string.

Since *re.match()* only matches initial substrings constrained to itself match only initial substring

SEE ALSO: re.search() *249*; re.compile.match(

re.search(pattern=..., string=...[,flags=...])

Return a MatchObject corresponding to the left
that matches the pattern in the first argument
matched string can be of zero length if the patt
desired). A MatchObject, if returned, has a vari
matched patternbut notably a MatchObject is *n*

SEE ALSO: re.match() *248*; re.compile.search(

METHODS AND ATTRIBUTES

re.compile.findall(s)

Return a list of nonoverlapping occurrences of t
with the PatternObject.

SEE ALSO *re.findall()*

re.compile.flags

The numeric sum of the flags passed to *re.com*
guarantee is given by Python as to the values a

```
>>> import re
>>> re.I,re.L,re.M,re.S,re.X
(2, 4, 8, 16, 64)
>>> c = re.compile('a', re.I | re.M)
>>> c.flags
10
```

re.compile.groupindex

A dictionary mapping group names to group nu
pattern, the dictionary is empty. For example:

```
>>> import re
>>> c = re.compile(r'(\d+)(\[A-Z]+)(
>>> c.groupindex
{}
>>> c=re.compile(r'(?P<nums>\d+)(?P<
>>> c.groupindex
{'nums': 1, 'caps': 2, 'lwrs': 3}
```

re.compile.match(s [,start [,end]])

Return a MatchObject if an initial substring of tl
Otherwise, return None. A MatchObject, if retur
manipulate the matched patternbut notably a M

In contrast to the similar function *re.match()* ,
arguments start and end that limit the match t
start and end is similar to taking a slice of s as
used, "^" will only match the true start of s. Fo

```
>>> import re
>>> s = 'abcdefg'
>>> c = re.compile('^b')
>>> print c.match(s, 1)
None
>>> c.match(s[1:])
<SRE_Match object at 0x10c440>
>>> c = re.compile('.*f$')
>>> c.match(s[:-1])
<SRE_Match object at 0x116d80>
>>> c.match(s,1,6)
<SRE_Match object at 0x10c440>
```

SEE ALSO: re.match() *248*; re.compile.search(

re.compile.pattern

The pattern string underlying the compiled Mat

```
>>> import re
>>> c = re.compile('^abc$')
>>> c.pattern
'^abc$'
```

re.compile.search(s [,start [,end]])

Return a MatchObject corresponding to the left
matches the PatternObject. If no match is poss
zero length if the pattern allows that (usually n
returned, has a variety of methods and attribut
a MatchObject is *not* itself a string.

In contrast to the similar function *re.search()* ,
arguments start and end that limit the match t
specifying start and end is similar to taking a sl
and end are used, "^" will only match the true

```
>>> import re
>>> s = 'abcdefg'
>>> c = re.compile('^b')
>>> c = re.compile('^b')
>>> print c.search(s, 1),c.search(s[
```

```
None <SRE_Match object at 0x117980>
>>> c = re.compile('.*f$')
>>> print c.search(s[:-1]),c.search(
<SRE_Match object at 0x51040> <SRE_M
```

SEE ALSO: re.search() *249*; re.compile.match(

re.compile.split(s [,maxsplit])

Return a list of substrings of the first argument
groups are included in the resultant list. Otherw
are dropped, and only the substrings between

If the second argument maxsplit is specified as
are parsed into the list, with any leftover conta

re.compile.split() is identical in behavior to *re.s*
documentation of the latter for examples of usa

SEE ALSO: re.split() *246*;

re.compile.sub(repl, s [,count=0])

Return the string produced by replacing every
with the first argument repl in the second argu

specified, no more than count replacements wil

The first argument repl may be either a regular
function. Backreferences may be named or enu

re.compile.sub () is identical in behavior to *re.s*
documentation of the latter for a number of exa

SEE ALSO: re.sub() *246*; re.compile.subn() *25*

re.compile.subn()

Identical to *re.compile.sub()* , except return a 2
replacements made.

re.compile.subn() is identical in behavior to *re.*
documentation of the latter for examples of usa

SEE ALSO: re.subn() *248*; re.compile.sub() *25*

Note: The arguments to each "MatchObject" m
ellipses given on the *re.search()* line. All argum
re.search() return the very same type of object

re.match.end([group])
re.search.end ([group])

The index of the end of the target substring ma
group is specified, return the ending index of th
the ending index of group 0 (i.e., the whole ma
alternation operator that is not used in the curr
the same non-negative value as *re.search.start*

```
>>> import  re
>>> m = re.search('(\w+)((\d*)| )(\w
>>> m.groups()
('The', ' ', None, 'Cat')
>>> m.end(0), m.end(1), m.end(2), m.
(7, 3, 4, -1, 7)
```

re.match.endpos, re.search.endpos

The end position of the search. If *re.compile.se*
value, otherwise it is the length of the target st
the search, the value is always the length of th

SEE ALSO: re.compile.search() *250*; re.search(

re.match.expand(template)
re.search.expand(template)

Expand backreferences and escapes in the argu
by the MatchObject. The expansion rules are th
Any nonescaped characters may also be include

```
>>> import re
>>> m = re.search('(\w+) (\w+)','The
>>> m.expand(r'\g<2> : \1')
'Cat : The'
```

re.match.group([group [,...]])
re.search.group([group [,...]])

Return a group or groups from the MatchObject
matched substring. If one argument group is sp
the target string. If multiple arguments group1
corresponding substrings of the target.

```
>>> import re
>>> m = re.search(r'(\w+)(/)(\d+)','
>>> m.group()
'abc/123'
>>> m.group(1)
'abc'
>>> m.group(1,3)
('abc', '123')
```

SEE ALSO: re.search.groups() *253*; re.search.g

re.match.groupdict([defval])
re.search.groupdict([defval])

Return a dictionary whose keys are the named
Enumerated but unnamed groups are not incluc
dictionary are the substrings matched by each
part of an alternation operator that is not used
that key is None, or defval if an argument is sp

```
>>> import re
>>> m = re.search(r'(?P<one>\w+)((?E
>>> m.groupdict()
{'one': 'abc', 'tab': None, 'two': '
>>> m.groupdict('---')
{'one': 'abc', 'tab': '---', 'two':
```

SEE ALSO: re.search.groups() *253*;

re.match.groups([defval])
re.search.groups([defval])

Return a tuple of the substrings matched by gr

alternation operator that is not used in the curr
None, or defval if an argument is specified.

```
>>> import re
>>> m = re.search(r'(\w+)((\t)|(/))(
>>> m.groups()
('abc', '/', None, '/', '123')
>>> m.groups('---')
('abc', '/', '---', '/', '123')
```

SEE ALSO: re.search.group() *253*; re.search.gr

re.match.lastgroup, re.search.lastgroup

The name of the last matching group, or None
compose the match.

re.match.lastindex, re.search.lastindex

The index of the last matching group, or None i

re.match.pos, re.search.pos

The start position of the search. If *re.compile.s*
value, otherwise it is 0. If *re.search()* or *re.mat*
0.

SEE ALSO: re.compile.search() *250;* re.search(

re.match.re, re.search.re

The PatternObject used to produce the match.
must be retrieved from the PatternObject's patt

```
>>> import re
>>> m = re.search('a','The Cat in th
>>> m.re.pattern
'a'
```

re.match.span ([group])
re.search.span([group])

Return the tuple composed of the return values
(group). If the argument group is not specified,

```
>>> import re
>>> m = re.search('(\w+)((\d*)| )(\w
```

```
>>> m.groups()
('The', ' ', None, 'Cat')
>>> m.span(0), m.span(1), m.span(2),
((0, 7), (0, 3), (3, 4), (-1, -1), (
```

re.match.start ([group])
re.search.start ([group])

The index of the end of the target substring ma
group is specified, return the ending index of th
the ending index of group 0 (i.e., the whole ma
alternation operator that is not used in the curr
the same non-negative value as *re.search.start*

```
>>> import re
>>> m = re.search('(\w+)((\d*)| )(\w
>>> m.groups()
('The', ' ', None, 'Cat')
>>> m.start(0), m.start(1), m.start(
(0, 0, 3, -1, 4)
```

re.match.string, re.search.string

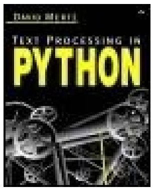
The target string in which the match occurs.

```
>>> import re
>>> m = re.search('a','The Cat in th
>>> m.string
'The Cat in the Hat'
```

EXCEPTIONS

re.error

Exception raised when an invalid regular expres
produce a compiled regular expression (includi


DAVID MERTZ
TEXT PROCESSING IN
PYTHON

Chapter 4. Parsers and State Machines

All the techniques presented in the prior chapters of this book have something in common, but something that is easy to overlook. In a sense, every basic string and regular expression operation treats strings as *homogeneous.* Put another way: String and regex techniques operate on *flat* texts. While said techniques are largely in keeping with the "Zen of Python" maxim that "Flat is better than nested," sometimes the maxim (and homogeneous operations) cannot solve a problem. Sometimes the data in a text has a deeper *structure* than the linear sequence of bytes that make up strings.

It is not entirely true that the prior chapters have eschewed data structures. From time to time, the examples presented broke flat texts into lists of lines, or of fields, or of segments matched by patterns. But the structures used have been quite simple and quite regular. Perhaps a text was treated as a list of substrings, with each substring manipulated in some manneror maybe even a list of lists of such substrings, or a list of tuples of data fields. But overall, the data structures have had limited (and mostly fixed) nesting depth and have consisted of sequences of items that are themselves treated similarly. What this chapter introduces is the notion of thinking about texts as *trees* of nodes, or even still more generally as graphs.

Before jumping too far into the world of nonflat texts, I should repeat a warning this book has issued from time to time. If you do not *need* to use the techniques in this chapter, you are better off sticking with the simpler and more maintainable techniques discussed in the prior chapters. Solving too general a problem too soon is a pitfall for application developmentit is almost always better to do less than to do more. Fullscale parsers and state machines fall to the "more" side of such a choice. As we have seen already, the class of problems you can solve using regular expressionsor even only string operationsis quite broad.

There is another warning that can be mentioned at this point. This book does not attempt to explain parsing theory or the design of parseable languages. There are a lot of intricacies to these matters, about which a reader can consult a specialized text like the so-called "Dragon Book"Aho, Sethi, and Ullman's *Compilers: Principle, Techniques and Tools* (Addison-Wesley, 1986; ISBN: 0201100886)or Levine, Mason, and Brown's *Lex & Yacc* (Second Edition, O'Reilly, 1992; ISBN: 1-56592-000-7). When Extended Backus-Naur Form (EBNF) grammars or other parsing descriptions are discussed below, it is in a general fashion that does not delve into algorithmic resolution of ambiguities or big-O efficiencies (at least not in much detail). In practice, everyday Python programmers who are processing textsbut who are not designing new programming languagesneed not worry about those parsing subtleties omitted from this book.

4.1 An Introduction to Parsers

4.1.1 When Data Becomes Deep and Texts Be

Regular expressions can match quite complicat
comes to matching arbitrarily nested subpatter
quite often in programming languages and text
places sometimes). For example, in HTML docu
nested inside each other. For that matter, chara
nest arbitrarilythe following defines a valid HTM

```
>>>  s = '''<p>Plain text, <i>italic
            <i>italicized subphrase<
            subphrase</b></i>, <i>ot
            phrase</i></p>'''
```

The problem with this fragment is that most an
less or more than a desired <i> element body.

```
>>> ital = r'''(?sx)<i>.+</i>'''
>>> for phrs in re.findall(ital, s):
...     print phrs, '\n-----'
...
<i>italicized phrase,
       <i>italicized subphrase</i>,
       subphrase</b></i>, <i>other i
       phrase</i>
-----
>>> ital2 = r'''(?sx)<i>.+?</i>'''
>>> for phrs in re.findall(ital2, s)
...     print phrs, '\n-----'
...
<i>italicized phrase,
       <i>italicized subphrase</i>
-----
<i>other italic
       phrase</i>
-----
```

What is missing in the proposed regular expres
imagine reading through a string character-by-
match must do within the underlying regex eng
of "How many layers of italics tags am I in?" W
would be possible to figure out which opening t

meant to match. But regular expressions are n

You encounter a similar nesting in most progra
suppose we have a hypothetical (somewhat BA
IF/THEN/END structure. To simplify, suppose th
the regex cond\d+, and every action matches a
IF/THEN/END structures can nest within each o
define the following three top-level structures:

```
>>> s = '''
IF cond1 THEN act1 END
-----
IF cond2 THEN
  IF cond3 THEN act3 END
END
-----
IF cond4 THEN
  act4
END
'''
```

As with the markup example, you might first tr
a regular expression like:

```
>>> pat = r'''(?sx)
IF \s+
```

```
cond\d+ \s+
THEN \s+
act\d+ \s+
END'''
>>> for stmt in re.findall(pat, s):
...     print stmt, '\n-----'
...
IF cond1 THEN act1 END
-----
IF cond3 THEN act3 END
-----
IF cond4 THEN
  act4
END
-----
```

This indeed finds three structures, but the wror
structure should be the compound statement th
cond3. It is not too difficult to allow a nested IF
substitute for a simple action; for example:

```
>>> pat2 = '''(?sx)(
IF \s+
cond\d+ \s+
THEN \s+
```

```
(   (IF \s+ cond\d+ \s+ THEN \s+ act\
 | (act\d+)
) \s+
END
)'''
>>> for stmt in re.findall(pat2, s):
...     print stmt[0], '\n-----'
...
IF cond1 THEN act1 END
-----
IF cond2 THEN
  IF cond3 THEN act3 END
END
-----
IF cond4 THEN
  act4
END
-----
```

By manually nesting a "first order" IF/THEN/EN
simple action, we can indeed match the examp
assumed that nesting of IF/THEN/END structur
"second order" structure is nested inside a "thi
infinitum? What we would like is a means of de
a text, in a manner similar to, but more genera

describe.

4.1.2 What Is a Grammar?

In order to parse nested structures in a text, yo "grammar." A grammar is a specification of a se "productions") arranged into a strictly hierarchi have a nameand perhaps some other propertie collection of child nodes. When a document is p node can ever be a descendent of itself; this is produces a tree rather than a graph.

In many actual implementations, such as the fa grammar is expressed at two layers. At the firs produces a stream of "tokens" for a "parser" to frequently what you might think of as words or the text differently than does our normal idea o nonoverlapping subsequences of the original te specification used, some subsequences may be "zero-case" lexer is one that simply treats the a parser operates on (some modules discussed d

The second layer of a grammar is the actual pa sequence of tokens and generates a "parse tree generated under the assumption that the under according to the grammarthat is, there is a way grammar specification. With most parser tools, on EBNF.

An EBNF grammar consists of a set of rule decl
similar quantification and alternation as that in
use slightly different syntax for specifying gram
expressivity and available quantifiers. But almo
their grammar specifications. Even the DTDs us
Chapter 5) have a very similar syntax to other
sense since an XML dialect is a particular gramm

```
<!ELEMENT body  ((example-column | i
```

In brief, under the sample DTD, a <body> elem
occurrences of a "first thing"that first thing bei
<image-column>. Following the optional first c
must occur. Of course, we would need to see th
in a <text-column>, or to see what other elem
in. But each such rule is similar in form.

A familiar EBNF grammar to Python programme
On many Python installations, this grammar as
location like [...]/Python22/Doc/ref/grammar.tx
Python Language Reference excerpts from the
example, a floating point number in Python is i

EBNF-style description of Python floating poi

```
floatnumber   ::= pointfloat | expor
pointfloat    ::= [intpart] fractior
exponentfloat ::= (intpart | pointfl
intpart       ::= digit+
fraction      ::= "." digit+
exponent      ::= ("e" | "E") ["+" |
digit         ::= "0"..."9"
```

The Python grammar is given in an EBNF variar expressivity. Most of the tools this chapter disc are still ultimately capable of expressing just as verbosely). Both literal strings and character ra production. Alternation is expressed with "|". Q are used. These features are very similar to th Additionally, optional groups are indicated with mandatory groups with parentheses. Conceptu regex "?" quantifier.

Where an EBNF grammar goes beyond a regula named terms as parts of patterns. At first glanc substitute regular expression patterns for name floating point pattern presented, we could simp

Regular expression to identify a floating poin

```
pat = r'''(?x)
```

```
    (                        # exponent
      (                      # intpart
        (                    # pointflo
          (\d+)?[.]\d+       # optional
          |
          \d+[.]             # intpart
        )                    # end poir
        |
        \d+                  # intpart
      )                      # end intp
      [eE][+-]?\d+           # exponent
    )                        # end expo
    |
    (                        # pointflo
      (\d+)?[.]\d+           # optional
      |
      \d+[.]                 # intpart
    )                        # end poir
    '''
```

As a regular expression, the description is hard
documentation added to a verbose regex. The l
documenting. Moreover, some care had to be ta
expressionthe exponentfloat alternative is requ
alternative since the latter can form a subseque

need for a little tweaking and documentation, t
as generaland exactly equivalentto the Python

You might wonder, therefore, what the point of
floating point number is an unusually simple st
floatnumber requires no recursion or self-refere
makes up a floatnumber is something simpler,
those simpler components is itself made up of
in defining a Python floating point number.

In the general case, structures can recursively
by containing other structures that in turn cont
entirely absurd to imagine floating point numbe
language had them would not be Python, howe
a "googol" was defined in 1938 by Edward Kasn
(otherwise called "10 dotrigintillion"). As a Pyth
as 1e100. Kasner also defined a "googolplex" a
much larger than anyone needs for any practic
Python expression to name a googolplexfor exa
conceive a programming language that allowed
googolplex. By the way: If you try to actually *c*
other programming language), you will be in fo
computer and/or some sort of crash or overflo
most language grammars are quite a bit more
can actually do anything with.

Suppose that you wanted to allow these new "e
language. In terms of the grammar, you could

description:

```
exponent ::= ("e" | "E") ["+" | "-"]
```

In the regular expression, the change is a prob
expression identifies the (optional) exponent:

```
[eE][+-]?\d+       # exponent
```

In this case, an exponent is just a series of digi
floating point terms, the regular expression wo
regular expression in place of \d+. Unfortunate
replacement would still contain the insufficient
require substitution. The sequence of substituti
regular expression is infinitely long.

4.1.3 An EBNF Grammar for IF/THEN/END Str

The IF/THEN/END language structure presente
example of nestable grammatical structures th
numbers. In fact, Pythonalong with almost eve
precisely such if statements inside other if stat
we might describe our hypothetical simplified I
EBNF variant used for Python's grammar.

Recall first our simplified rules for allowable str
and END, and they always occur in that order w

in this language are always in all capitals. Any
insignificant, except that each term is separate
whitespace. Every condition is spelled to match
Every IF "body" either contains an action that r
act\d+, *or* it contains another IF/THEN/END str
three IF/THEN/END structures, one of which co

```
IF cond1 THEN act1 END
-----
IF cond2 THEN
  IF cond3 THEN act3 END
END
-----
IF cond4 THEN
  act4
END
```

Let us try a grammar:

EBNF grammar for IF/THEN/END structures

```
if_expr   ::= "IF" ws cond ws "THEN"
whitechar ::= " " | "\t" | "\n" | "\
ws        ::= whitechar+
digit     ::= "0"..."9"
```

```
number    ::= digit+
cond      ::= "cond" number
action    ::= simpleact | if_expr
simpleact ::= "act" number
```

This grammar is fairly easy to follow. It defines
ws and number that consist of repetitions of sir
as an explicit alternation of individual character
Taken to the extreme, every production could a
verbose if_expr productionyou would just subst
productions for the names in the if_expr produ
much easier to read. The most notable aspect c
production, since an action can itself recursivel

For this problem, the reader is encouraged to d
robust variations on the very simple IF/THEN/E
evident, it is difficult to actually do much with t
actions and conditions are given semantic mear
can invent their own variations, but a few are p

4.1.4 Pencil-and-Paper Parsing

To test a grammar at this point, just try to expa
some production that is allowed at that point in
and paper. Think of the text of test cases as a t
production (if so, write the satisfied production
the symbol is added to the "unsatisfied register

with pencil and paper, however: It is better to s
subsequence than a shorter one. If a parent pr
the children must be satisfied in the specified o
For now, assume only one character of lookahe
example, suppose you find the following sequer

```
"IF   cond1..."
```

Your steps with the pencil would be something

1. **Read the "I"no production is satisfied.**

- Read the "F", unsatisfied becomes "I"-"F". No in if_expr (a literal is considered a production). quantifiers or alternates, write down the "IF" pr
- Read the space, Unsatisfied becomes simply a ws, but hold off for a character since ws contai substring to satisfy it.
- Read the second space, unsatisfied becomes s production ws. But again hold off for a characte
- Read the third space, unsatisfied becomes spa the production ws. But keep holding off for the
- Read the "c", unsatisfied becomes "space-spa production, so revert to the production in 5. Un

- Et cetera.

If you get to the last character, and everything
case is valid under the grammar. Otherwise, th
few IF/THEN/END structures that you think are
grammar.

4.1.5 Exercise: Some variations on the langu

1. **Create and test an IF/THEN/END gramm
occur between the THEN and the END. F
structures are valid under this variation**

```
IF cond1 THEN act1 act2 act3 END
-----
IF cond2 THEN
  IF cond3 THEN act3 END
  IF cond4 THEN act4 END
END
-----
IF cond5 THEN IF cond6 THEN act6 a
```

- Create and test an IF/THEN/END grammar th
numbers as conditions (as an enhancement of
comparison consists of two numbers with one c
There might or might not be any whitespace be

surrounding numbers. Use your judgment abou
Python floating point grammar might provide a
simpler).

- Create and test an IF/THEN/END grammar tha
action. A loop consists of the keyword LOOP, fo
by action(s), and terminated by the END keywo
actions, and therefore ifs and loops can be cont
example:

```
IF cond1 THEN
  LOOP 100
    IF cond2 THEN
      act2
    END
  END
END
```

You can make this LOOP-enhanced grammar ar
you wish.

- Create and test an IF/THEN/END grammar tha
If an ELSE occurs, it is within an IF body, but E
own body that can contain action(s). For examp

```
IF cond1 THEN
  act1
```

```
  act2
ELSE
  act3
  act4
END
```

- Create and test an IF/THEN/END grammar tha IF, ELSE, or LOOP body. For example, the follov variant:

```
IF cond1 THEN
ELSE act2
END
-*-
IF cond1 THEN
  LOOP 100 END
ELSE
END
```

4.2 An Introduction to State Machin

State machines, in a theoretical sense, underla related. But a Python programmer does not ne matters in writing programs. Nonetheless, ther problems where the best and most natural app solution. At heart, a state machine is just a wa application.

A parser is a specialized type of state machine structured texts. Generally a parser is accompa that describes the states and transitions used b in turn applied to text obeying a "grammar."

In some text processing problems, the processi of text depends upon what we have done so fa can be naturally expressed using a parser gram with the semantics of the prior text than with it properties a portion of a text has is generally o

Concretely, we might calculate some arithmetic
name encountered in a text file in a database,
processing. Where the parsing of a text depend
useful approach.

Implementing an elementary and generic state
used for a variety of purposes. The third-party
discussed later in this chapter, can also be used
processors.

4.2.1 Understanding State Machines

A much too accurate description of a state mac
set of nodes and a set of transition functions. S
events; each event is in the domain of the tran
range is a subset of the nodes. The function ret
subset of the nodes are end-states; if an end-s

An abstract mathematical descriptionlike the or
programming problems. Equally picayune is the
programming language like Python is a state m
really in a declarativefunctional or constraint-ba
Furthermore, every regular expression is logica
parser implements an abstract state machine.
without really thinking about it, but that fact pr
techniques.

An informal, heuristic definition is more useful t
program requirement that includes a handful of
Furthermore, it is sometimes the case that indi
determine which type of treatment is appropria
identifying"). The state machines discussed in t
intended to express clearly the programming re
sense to talk about your programming problem
events, it is likely to be a good idea to program

4.2.2 Text Processing State Machines

One of the programming problems most likely t
text files. Processing a text file very often consi
file (typically either a character or a line), and d
In some cases, this processing is "stateless"tha
to determine exactly what to do in response to
though the text file is not 100 percent stateless
(for example, the line number might matter for
line number). But in other common text proces
highly "stateful"the meaning of a chunk depend
maybe even on what chunks come next). Files
readable texts, programming source files, and d
example of a stateful chunk is a line that might

```
myObject = SomeClass(this, that, oth
```

That line means something very different if it h

```
"""How to use SomeClass:
myObject = SomeClass(this, that, oth
"""
```

That is, we needed to know that we were in a "
comment rather than an action. Of course, a pr
general way will usually use a parser and gram

4.2.3 When Not to Use a State Machine

When we begin the task of writing a processor
should ask ourselves is "What types of things d
is a candidate for a state. These types should b
indefinite, a state machine is probably not the r
solution is appropriate. Or maybe the problem
be that many types of things.

Moreover, we are not quite ready for a state ma
It might turn out that even though our text file
where each chunk is a single type of thing. A st
the transitions between types of text require sc
single state-block.

An example of a somewhat stateful text file tha
state machine is a Windows-style .ini file (gene
data-with-API Windows registry). Those files cc
and a number of value assignments. For examp

File: hypothetical.ini

```
; set the colorscheme and userlevel
[colorscheme]
background=red
foreground=blue
title=green

[userlevel]
login=2
; admin=0
title=1
```

This example has no real-life meaning, but it w
.ini format. (1) In one sense, the type of each l
semicolon, left brace, or alphabetic). (2) In anc
keyword "title" presumably means something ir
could program a text processor that had a COL
processed the value assignments of each state.
handle this problem.

On the one hand, we could simply create the na
code like:

Chunking Python code to process .ini file

```
txt = open('hypothetical.ini').read(
from string import strip, split
nocomm = lambda s: s[0] != ';'
eq2pair = lambda s: split(s,'=')
def assignments(sect):
    name, body = split(sect,']')
    assigns = split(body,'\n')
    assigns = filter(strip, assigns)
    assigns = filter(None, assigns)
    assigns = filter(nocomm, assigns
    assigns = map(eq2pair, assigns)
    assigns = map(tuple, assigns)
    return (name, assigns)
sects = split(txt,'[')
sects = map(strip, sects)
sects = filter(nocomm, sects)
config = map(assignments, sects)
pprint.pprint(config)
```

Applied to the hypothetical.ini file above, this c

```
[('colorscheme',
  [('background', 'red'),
   ('foreground', 'blue'),
   ('title', 'green')]),
```

```
 ('userlevel',
  [('login', '2'),
   ('title', '1')])]
```

This particular list-oriented data structure may
enough to transform this into dictionary entries
slightly modified code could generate other dat

An alternative approach is to use a single curre
and process lines accordingly:

```
for line in open('hypothetical.ini')
    if line[0] == '[':
        current_section = line[1:-2]
    elif line[0] == ';':
        pass    # ignore comments
    else:
        apply_value(current_section,
```

Sidebar: A digression on functional programı

Readers will have noticed that the .ini chunking
functional programming (FP) style to it than do
wrote the presented code this way for two reas
emphasize the contrast with a state machine aı

its eschewal of state (see the discussion of func
example is, in a sense, even farther from a stat
that used a few nested loops in place of the ma

The more substantial reason I adopted a functi
type of problem is precisely the sort that can of
clearly using FP constructs. Basically, our sourc
homogeneous at each level. Each section is sim
assignment is similar to others. A clearand stat
structures is applying an operation uniformly to
do a given set of operations to find the assignm
well just map() that set of operations to the col
approach is more terse than a bunch of nested
better expressing the underlying intention of th

Use of a functional programming style, howeve
to map(), reduce(), and filter() can quickly bec
function/variable names are not chosen careful
Python" code (a popular competition for other l
constructs. Warnings in mind, it is possible to c
of the .ini chunking code (that produces identic
considerably short of obfuscated, but will still b
programmers. On the plus side, it is half the le
accidental side effects:

Strongly functional code to process .ini file

```
from string import strip, split
eq2tup = lambda s: tuple(split(s,'='
splitnames = lambda s: split(s,']')
parts = lambda s, delim: map(strip,
useful = lambda ss: filter(lambda s:
config = map(lambda _:(_[0], map(eq2
             map(splitnames, useful(
pprint.pprint(config)
```

In brief, this functional code says that a configu
(2) a list of key/value pairs. Using list compreh
the example code is compatible back to Python
and parts() go a long way towards keeping the
are, furthermore, potentially worth saving in a
makes the relevant .ini chunking code even sho

A reader exercise is to consider how the higher
on functional programming could further impro
presented in this subsection.

4.2.4 When to Use a State Machine

Now that we have established not to use a stat
should look at a case where a state machine is
Appendix D. Txt2Html converts "smart ASCII" f

In very brief recap, smart ASCII format is a tex
distinguish different types of text blocks, such a
samples. While it is easy for a human reader or
these text block types, there is no simple way t
Unlike in the .ini file example, text block types
no single delimiter that separates blocks in all c
blank line within a code sample does not neces
be separated by blank lines). But we do need t
on each text block type for the correct final XM
natural solution here.

The general behavior of the Txt2Html reader is
Read a line of the text file and go to current sta
met to leave the current state and enter anothe
appropriate for the current state. This example
but it expresses the pattern described:

A simple state machine input loop in Python

```
global state, blocks, newblock
for line in fpin.readlines():
    if state == "HEADER":         #
        if blankln.match(line):   ne
        elif textln.match(line):  st
        elif codeln.match(line):  st
        else:
```

```
            if newblock: startHead(l
            else: blocks[-1] += line
    elif state == "TEXT":         #
        if blankln.match(line):   ne
        elif headln.match(line):  st
        elif codeln.match(line):  st
        else:
            if newblock: startText(l
        else: blocks[-1] += line
elif state == "CODE":         # blar
    if blankln.match(line):   blocks
    elif headln.match(line):  startH
    elif textln.match(line):  startT
    else: blocks[-1] += line
else:
    raise ValueError, "unexpected in
```

The only real thing to notice is that the variable
in functions like startText(). The transition conc
expression patterns, but they could just as well
actually done later in the program; the state m
in the blocks list. In a sense, the state machine
processor.

4.2.5 An Abstract State Machine Class

It is easy in Python to abstract the form of a st
state machine model of the program stand out
block in the previous example (which doesn't ri
other conditional). Furthermore, the class prese
job of isolating in-state behavior. This improves
cases.

File: statemachine.py

```
class InitializationError(Exception)

class StateMachine:
    def __init__(self):
        self.handlers = []
        self.startState = None
        self.endStates = []

    def add_state(self, handler, enc
        self.handlers.append(handler
        if end_state:
            self.endStates.append(na

    def set_start(self, handler):
        self.startState = handler
```

```
def run(self, cargo=None):
    if not self.startState:
        raise InitializationErro
              "must call .set_st
    if not self.endStates:
        raise InitializationErro
              "at least one stat
    handler = self.startState
    while 1:
        (newState, cargo) = hanc
        if newState in self.endS
            newState(cargo)
            break
        elif newState not in sel
            raise RuntimeError,
        else:
            handler = newState
```

The StateMachine class is really all you need fo fewer lines than something similar would requi passing function objects in Python. You could e check and the self.handlers list, but the extra f intention.

To actually *use* the StateMachine class, you nee

want to use. A handler must follow a particular
in any case it must have some breakout conditi
should process another event of the state's typ
handler should check for breakout conditions ar
transition to. At the end, a handler should pass
and any cargo the new state handler will need.

An encapsulation device is the use of cargo as a
necessarily called cargo by the handlers). This
one state handler to take over where the last st
consist of a file handle, which would allow the n
point where the last state handler stopped. But
complex class instance, or a tuple with several

4.2.6 Processing a Report with a Concrete Sta

A moderately complicated report format provide
to a state machine programming styleand spec
The hypothetical report below has a number of
to buyer orders, but at other times the identica
Blank lines, for example, are processed differen
processed according to different rules, each get
order, a degree of stateful processing is perforn
calculations:

Sample Buyer/Order Report

```
MONTHLY REPORT -- April 2002
===================================

Rules:
 - Each buyer has price schedule for
 - Each buyer has a discount schedul
 - Discounts are per-order (i.e., cc
 - Buyer listing starts with line cc
 - Item quantities have name-whitesp
 - Comment sections begin with line
   and ends with first line that enc

>> Acme Purchasing

  widgets      100
  whatzits    1000
  doodads     5000
  dingdongs   20

* Note to Donald: The best contact f
* 413-555-0001.  Fallback is Sue For

>> Megamart

doodads   10k
```

```
whatzits  5k

>> Fly-by-Night Sellers
   widgets         500
   whatzits       4
   flazs          1000

* Note to Harry: Have Sales contact

*
Known buyers:
>>  Acme
>>  Megamart
>>  Standard (default discounts)
*

*** LATE ADDITIONS ***

>> Acme Purchasing
widgets      500     (rush shipment)
```

The code to processes this report below is a bit
is devoted merely to deciding when to leave th
of the "buyer states" is sufficiently similar that
parameterized state; but in a real-world applica

detailed custom programming for both in-state
For example, a report might allow different forr

buyer_invoices.py

```
from statemachine import StateMachir
from buyers import STANDARD, ACME, M
from pricing import discount_schedul
import sys, string

#-- Machine States
def error(cargo):
    # Don't want to get here! Unider
    sys.stderr.write('Unidentifiable

def eof(cargo):
    # Normal termination -- Cleanup
    sys.stdout.write('Processing Suc

def read_through(cargo):
    # Skip through headers until buy
    fp, last = cargo
    while 1:
        line = fp.readline()
```

```
        if not line:                retu
        elif line[:2] == '>>':  retu
        elif line[0] == '*':    retu
        else:                       cont

def comment(cargo):
    # Skip comments
    fp, last = cargo
    if len(last) > 2 and string.rstr
        return read_through, (fp, ''
    while 1:
        # could save or process comm
        line = fp.readline()
        lastchar = string.rstrip(lin
        if not line:                retu
        elif lastchar == '*':   retu
def STANDARD(cargo, discounts=discou
                prices=item_pric
    fp, company = cargo
    invoice = 0
    while 1:
        line = fp.readline()
        nextstate = buyerbranch(line
        if nextstate == 0: continue
        elif nextstate == 1:
```

```
            invoice = invoice + calc
        else:
            pr_invoice(company, 'sta
            return nextstate, (fp, l

def ACME(cargo, discounts=discount_s
                prices=item_prices[A
    fp, company = cargo
    invoice = 0
    while 1:
        line = fp.readline()
        nextstate = buyerbranch(line
        if nextstate == 0: continue
        elif nextstate == 1:
            invoice = invoice + calc
        else:
            pr_invoice(company, 'neg
            return nextstate, (fp, l

def MEGAMART(cargo, discounts=discou
                    prices=item_pric
    fp, company = cargo
    invoice = 0
    while 1:
        line = fp.readline()
```

```
        nextstate = buyerbranch(line
        if nextstate == 0: continue
        elif nextstate == 1:
            invoice = invoice + calc
        else:
            pr_invoice(company, 'neg
            return nextstate, (fp, l

#-- Support function for buyer/state
def whichbuyer(line):
    # What state/buyer does this lin
    line = string.upper(string.repla
    find = string.find
    if find(line,'ACME') >= 0:
    elif find(line,'MEGAMART')>= 0:
    else:

def buyerbranch(line):
    if not line:
    elif not string.strip(line):
    elif line[0] == '*':
    elif line[:2] == '>>':
    else:

#-- General support functions
```

```
def calc_price(line, prices):
    product, quant = string.split(li
    quant = string.replace(string.up
    quant = int(quant)
    return quant*prices[product]

def discount(invoice, discounts):
    multiplier = 1.0
    for threshhold, percent in disco
        if invoice >= threshhold: mu
    return invoice*multiplier

def pr_invoice(company, disctype, am
    print "Company name:", company[3
    print "Invoice total: $", amount

if __name__== "__main__":
    m = StateMachine()
    m.add_state(read_through)
    m.add_state(comment)
    m.add_state(STANDARD)
    m.add_state(ACME)
    m.add_state(MEGAMART)
    m.add_state(error, end_state=1)
    m.add_state(eof, end_state=1)
```

```
    m.set_start(read_through)
    m.run((sys.stdin, ''))
```

The body of each state function consists mostly
returning a new target state, along with a carg
of a file handle and the last line read. In some
also needed for use by the subsequent state. T
flow diagram lets you see the set of transitions

All of the buyer states are "initialized" using de
during calls by a normal state machine .run() c
as classes instead of as functions, but that feel
specific initializer values are contained in a sup

pricing.py support data

```
from buyers import STANDARD, ACME, M

# Discount consists of dollar requir
# Each buyer can have an ascending s
# one applicable to a month is used.
discount_schedules = {
    STANDARD  : [(5000,10),(10000,20
    ACME      : [(1000,10),(5000,15)
    MEGAMART  : [(2000,10),(5000,20)
```

```
    BAGOBOLTS : [(2500,10),(5000,15)
  }
item_prices = {
    STANDARD  : {'widgets':1.0, 'wha
                 'dingdongs':1.3, 'f
    ACME      : {'widgets':0.9, 'wha
                 'dingdongs':0.9, 'f
    MEGAMART  : {'widgets':1.0, 'wha
                 'dingdongs':1.2, 'f
    BAGOBOLTS : {'widgets':0.8, 'wha
                 'dingdongs':1.3, 'f
  }
```

In place of reading in such a data structure, a f
read them from a database of some sort. None
abstract flow into separate modules makes for

4.2.7 Subgraphs and State Reuse

Another benefit of the state machine design ap
states without touching the state handlers at al
doing soif a state branches to another state, th
"registered" states. You can, however, add hom
states. For example:

Creating end states for subgraphs

```
from statemachine import StateMachir
from BigGraph import *

def subgraph_end(cargo): print "Leav
foo = subgraph_end
bar = subgraph_end

def spam_return(cargo): return spam,
baz = spam_return
if __name__=='__main__':
    m = StateMachine()
    m.add_state(foo, end_state=1)
    m.add_state(bar, end_state=1)
    m.add_state(baz)
    map(m.add_state, [spam, eggs, ba
    m.set_start(spam)
    m.run(None)
```

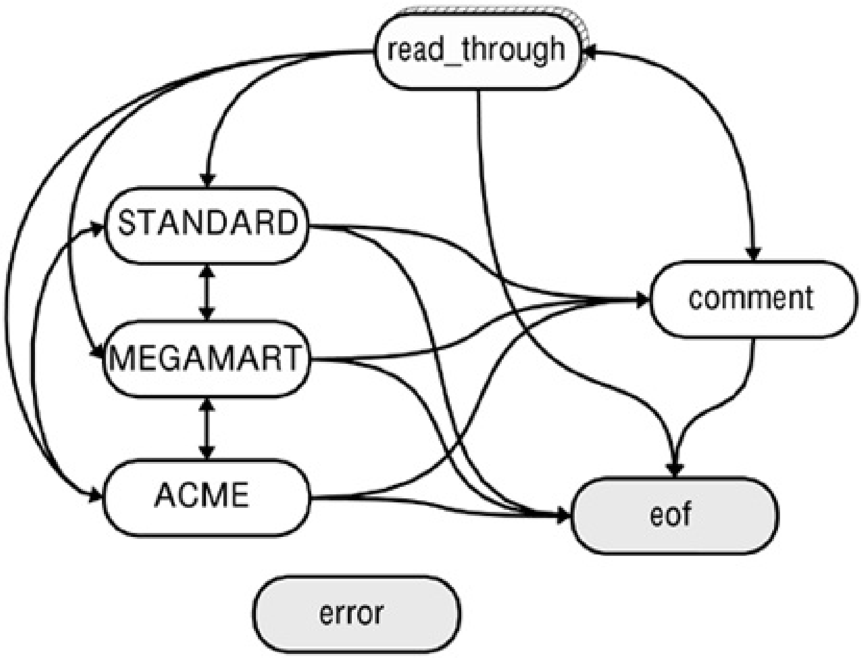
In a complex state machine graph, you often er
particular collection of statesi.e., nodesmight ha
few connections out to the rest of the graph. Us
related set of functionality.

For processing the buyer report discussed earli
meaningful subgraphs really exist. But in the su
BigGraph module contains hundreds or thousar
complex complete graph. Supposing that the st
subgraph, and all branches out of the subgraph
an entire new application.

The example redefined foo and bar as end state
StateMachine object) ends when they are reach
into the spam-eggs-bacon subgraph. A subgrap
state machine. It is actually the end_state flag
as an end state, it would raise a RuntimeError

If you create large graphsespecially with the int
is often useful to create a state diagram. Pencil
this; a variety of flow-chart software also exists
to allow you to identify clustered subgraphs and
of a functional subgraph. A state diagram from
A quick look at Figure 4.1, for example, allows
which might not have been evident in the code
enhancement to the diagram and handlers mig
written into it.

Figure 4.1. Buyer s



4.2.8 Exercise: Finding other solutions

1. **On the face of it, a lot of "machinery" w complicated a report above. The goal of robust and to allow for expansion to lar machine approach in your mind, how el: the presented type (assume that "reaso the same type).**

   Try writing a fresh report processing ap the presented application (or at least sc against the sample report and against a

   What errors did you encounter running more concise than the presented one? V presented application? Is your applicati

**another programmer? Which approach w
other report formats? In what respect is
state machine example?**

- The error state is never actually reached in th
transition conditions into the error state would
types of corruption or mistakes in reports do yo
reports, or other documents, are flawed, but it
possible. What are good approaches to recover
those approaches in state machine terms, using
framework?

4.3 Parser Libraries for Python

4.3.1 Specialized Parsers in the Standard Libı

Python comes standard with a number of modu
variety of custom formats are in sufficiently wic
standard library support for them. Aside from t
the *email* and *xml* packages, and the modules *ı*
which performs parsing of sorts. A number of a
handle and process audio and image formats, i
tools. However, these media formats are better
than as token streams of the sort parsers hand

The specialized tools discussed under this secti
Python Library Reference for detailed documen
worth knowing what is available, but for space
specifics of these few modules.

ConfigParser

Parse and modify Windows-style configuration f

```
>>> import ConfigParser
>>> config = ConfigParser.ConfigPars
>>> config.read(['test.ini','nonesuc
>>> config.sections()
['userlevel', 'colorscheme']
>>> config.get('userlevel','login')
'2'
>>> config.set('userlevel','login',5
>>> config.write(sys.stdout)
[userlevel]
login = 5
title = 1

[colorscheme]
background = red
foreground = blue
```

difflib
.../Tools/scripts/ndiff.py

The module *difflib*, introduced in Python 2.1, co
you determine the difference and similarity of p
enough to work with sequences of all kinds, bu
lines or sequences of characters.

Word similarity is useful for determining likely r
required between strings. The function *difflib.g*
"fuzzy matching" of a string against patterns. T

```
>>> users = ['j.smith', 't.smith', '
>>> maxhits = 10
>>> login = 'a.smith'
>>> difflib.get_close_matches(login,
['t.smith', 'j.smith', 'p.smyth']
>>> difflib.get_close_matches(login,
['t.smith', 'j.smith']
>>> difflib.get_close_matches(login,
['t.smith', 'j.smith', 'p.smyth', 'a
```

Line matching is similar to the behavior of the U
utility is able to take a source and a difference,
(file). The functions *difflib.ndiff()* and *difflib.res*
time, however, the bundled ndiff.py tool perforr
the "patches" with an -r# option).

```
%. ./ndiff.py chap4.txt chap4.txt~ |
-:  chap4.txt
```

```
+:   chap4.txt~
+       against patterns.
-      against patterns.  The require
-
-      >>> users = ['j.smith', 't.smi
-      >>> maxhits = 10
-      >>> login = 'a.smith'
```

There are a few more capabilities in the *difflib* r
possible.

formatter

Transform an abstract sequence of formatting e
objects. Writer objects, in turn, produce concre
parent formatter and writer classes are contain

In a way, *formatter* is an "anti-parser"that is, w
program events, *formatter* transforms a series

The purpose of the *formatter* module is to struc
processor file formats. The module *htmllib* utiliz
details provide calls related to features like font

For highly structured output of prose-oriented c
albeit requiring learning a fairly complicated AP

classes included to create simple tools. For exa
equivalent to lynx -dump:

urldump.py

```
#!/usr/bin/env python
import sys
from urllib import urlopen
from htmllib import HTMLParser
from formatter import AbstractFormat
if len(sys.argv) > 1:
    fpin = urlopen(sys.argv[1])
    parser = HTMLParser(AbstractForm
    parser.feed(fpin.read())
    print '-------------------------
    print fpin.geturl()
    print fpin.info()
else:
    print "No specified URL"
```

SEE ALSO: htmllib *285;* urllib *388;*

htmllib

Parse and process HTML files, using the service
module, *htmllib* relies on the user constructing
callbacks from HTML events, usually utilizing th
a "writer" (also usually based on the *formatter*
layers of indirection in the *htmllib* API to make

SEE ALSO: HTMLParser *384*; formatter *284*; sg

multifile

The class *multifile.MultiFile* allows you to treat a
as if it were several files, each with their own F
.seek(), and .tell() methods. In iterator fashion
with the method *multifile.MultiFile.next()*.

SEE ALSO: fileinput *61;* mailbox *372;* email.Par

parser
symbol
token
tokenize

Interface to Python's internal parser and tokeni
arguably a text processing task, the complexiti
book.

robotparser

Examine a robots.txt access control file. This fil
behavior of automatic indexers and Web crawle
requests.

sgmllib

A partial parser for SGML. Standard Generalize
complex document standard; in its full generali
rather a grammar for describing concrete forma
XML is (almost) a simplified subset of SGML.

Although it might be nice to have a Python libr
such a thing. Instead, *sgmllib* implements just
with *htmllib*. You might be able to coax parsing
but Python's standard XML tools are far more r

SEE ALSO: htmllib *285;* xml.sax *405;*

shlex

A lexical analyzer class for simple Unix shell-lik
implement small command language within Pyt

tabnanny

This module is generally used as a command-li
applications. The module/script *tabnanny* check
tabs and spaces within the same block. Behind
tokenized, but normal usage consists of someth

```
% /sw/lib/python2.2/tabnanny.py SCRI
SCRIPTS/cmdline.py 165 '\treturn 1\r
'SCRIPTS/HTMLParser_stack.py': Toker
                                 mult
SCRIPTS/outputters.py 18 '\tself.wri
SCRIPTS/txt2bookU.py 148 '\ttry:\n'
```

The tool is single purpose, but that purpose add
programming.

SEE ALSO: tokenize *285;*

4.3.2 Low-Level State Machine Parsing

mx.TextTools • Fast Text Manipulation Too

Marc-Andre Lemburg's *mx.TextTools* is a remar

gestalt of. *mx.TextTools* can be blazingly fast a
as difficult as it might be to "get" the mindset c
application written with it working just right. Or
mx.TextTools can process a larger class of text
simultaneously operating much faster. But debu
you wish you were merely debugging a cryptic

In recent versions, *mx.TextTools* has come in a
other "mx Extensions for Python." Most of the c
implementations of datatypes not found in a ba

mx.TextTools stands somewhere between a sta
the module *SimpleParse*, discussed below, is ar
mx.TextTools. As a state machine, *mx.TextToo*
statemachine module presented in the prior sec
very close to a high-level parser. This is how Le
accompanying *mx.TextTools*:

> mxTextTools is an extension package for Pytho
> types that implement high-performance text r
> addition to a very flexible and extendable stat
> scanning and processing text based on low-lev
> tuples. It gives you access to the speed of C w
> steps every time you change the parsing desc
>
> Applications include parsing structured text, fi
> using translation tables) and recombining stri

The Python standard library has a good set of t
powerful, flexible, and easy to work with. But P
fast. Mind you, for most problems, Python by it
class of problems, being able to choose *mx.Tex*

The unusual structure of *mx.TextTools* applicat
usage. After a few sample applications are pres
commands, modifiers, and functions is given.

BENCHMARKS

A familiar computer-industry paraphrase of Mar
dictates that there are "Lies, Damn Lies, and B
certainly do not want readers to put too great a
Nonetheless, in exploring *mx.TextTools*, I want
here is a rough idea.

The second example below presents part of a r
Txt2Html application reproduced in Appendix D
is the regular expression replacements perform
ASCII inline markup of words and phrases.

In order to get a timeable test case, I concaten
file a bit over 2MB, and about 41k lines and 30
one text block, first using an *mx.TextTools* vers

Processing time of the same test file went from

slowish Linux test machine (running Python 1.5
about a 3x speedup over what I get with the *re*
particular applications might gain significantly r
Moreover, 34 seconds is a long time in an intera
a batch process done once a day, or once a wee

Example: Buyer/Order Report Parsing

Recall (or refer to) the sample report presented
State Machines." A report contained a mixture
comments. The state machine we used looked
based on context whether the new line indicate
to write almost the same algorithm utilizing *mx*
that is not what we will do.

A more representative use of *mx.TextTools* is t
interesting components of the report document
"grammar" that describes every valid "buyer re
procedural/grammar approach is much easier,
report.

An *mx.TextTools* tag table is a miniature state
portion of a string. Matching, in this context, m
while nonmatching means that a "failure" end s
table is a success state. Each individual state ir
construct by reading from the "read-head" and
either success or failure, program flow jumps t

success or failure state for the tag table as a w
often different from the jump target for failureb
jump targets, unlike the *statemachine* module's

Notably, one of the types of states you can incl
state can "externally" look like a simple match
subpatterns and machine flow in order to deter
as in an EBNF grammar, you can build nested c
States can also have special behavior, such as f
mx.TextTools tag table state is simply a binary

Let us look at an *mx.TextTools* parsing applicat
works:

buyer_report.py

```
from mx.TextTools import *

word_set = set(alphanumeric+white+'-
quant_set = set(number+'kKmM')

item   = ( (None, AllInSet, newline_
           (None, AllInSet, white_se
           ('Prod', AllInSet, a2z_se
           (None, AllInSet, white_se
```

```
           ('Quant', AllInSet, quant
           (None, WordEnd, '\n', -5)

buyers = ( ('Order', Table,
                  ( (None, WordEnd,
                    ('Buyer', AllInS
                    ('Item', Table,
                  Fail, +0), )

comments = ( ('Comment', Table,
                  ( (None, Word, '\r
                    (None, WordEnd,
                    (None, Skip, -1)
                  +1, +2),
             (None, Skip, +1),
             (None, EOF, Here, -2) )

def unclaimed_ranges(tagtuple):
    starts = [0] + [tup[2] for tup i
    stops = [tup[1] for tup in tagtu
    return zip(starts, stops)

def report2data(s):
    comtuple = tag(s, comments)
    taglist = comtuple[1]
```

```
    for beg,end in unclaimed_ranges(
        taglist.extend(tag(s, buyers
    taglist.sort(cmp)
    return taglist

if __name__=='__main__':
    import sys, pprint
    pprint.pprint(report2data(sys.st
```

Several tag tables are defined in *buyer_report:* such as those in each tag table are general mat patterns; after working with *mx.TextTools* for a tag tables. As mentioned above, states in tag t name or inline. For example, buyers contains a utilizes the tag table named item.

Let us take a look, step by step, at what the bu tag table needs to be passed as an argument t string to match against. That is done in the rep general, buyersor any tag tablecontains a list o example, all such states are numbered in comn state, which contains a subtable with three stat

Tag table state in buyers

1. **Try to match the subtable. If the match**

taglist of matches. If the match fails, do jump back into the one state (i.e., +0). succeeds, advancing the read-head on e

Subtable states in buyers

1. **Try to find the end of the "word" \n>> than symbols at the beginning of a line. past the point that first matched. If this (sub)table as a whole fails to match. No match, so the default jump of +1 is take anything to the taglist upon a state mat**

- Try to find some word_set characters. This se various other sets are defined in *mx.TextTools* Buyer to the taglist of matches. As many contig matched. The match is considered a failure if th state match fails, jump to Fail, as in state (1).

- Try to match the item tag table. If the match matches. What gets added, moreover, includes match fails, jump to MatchOkthat is, the (sub)t succeeds, jump +0that is, keep looking for ano

What *buyer_report* actually does is to first iden between comments for buyer orders. This appr the design of *mx.TextTools* allows us to do this

not involve actually pulling out the slices that m
numerically the offset ranges where they occur.
performing repeated slices, or otherwise creatin

The following is important to notice: As of versi
mx.TextTools.tag() function that accompanies m
optional third and fourth arguments are passed
offsets within a larger string to scan, *not* the st
versions will fix the discrepancy (either approac
breakage in existing code).

What *buyer_report* produces is a data structure
something like:

buyer_report.py data structure

```
$ python ex_mx.py < recs.tmp
[('Order', 0,  638,
  [('Buyer', 547, 562, None),
   ('Item', 562, 583,
    [('Prod', 566, 573, None), ('Qua
   ('Item', 583, 602,
     [('Prod', 585, 593, None), ('Qu
   ('Item', 602, 621,
     [('Prod', 604, 611, None), ('Qu
```

```
    ('Item', 621, 638,
      [('Prod', 623, 632, None), ('Qu
  ('Comment', 638, 763, []),
  ('Order', 763, 805,
   [('Buyer', 768, 776, None),
    ('Item', 776, 792,
     [('Prod', 778, 785, None), ('Qua
    ('Item', 792, 805,
     [('Prod', 792, 800, None), ('Qua
  ('Order', 805, 893,
   [('Buyer', 809, 829, None),
    ('Item', 829, 852,
     [('Prod', 833, 840, None), ('Qua
    ('Item', 852, 871,
     [('Prod', 855, 863, None), ('Qua
    ('Item', 871, 893,
     [('Prod', 874, 879, None), ('Qua
  ('Comment', 893, 952, []),
  ('Comment', 952, 1025, []),
  ('Comment', 1026, 1049, []),
  ('Order', 1049, 1109,
   [('Buyer', 1054, 1069, None),
    ('Item',1069, 1109,
     [('Prod', 1070, 1077, None), ('Q
```

While this is "just" a new data structure, it is qu
reports. For example, here is a brief function th
taglist. You could even arrange for it to be valic
(see Chapter 5 for details about XML, DTDs, etc

```
def taglist2xml(s, taglist, root):
    print '<%s>' % root
    for tt in taglist:
        if tt[3] :
            taglist2xml(s, tt[3], tt
        else:
            print '<%s>%s</%s>' % (t
    print '</%s>' % root
```

Example: Marking up smart ASCII

The "smart ASCII" format uses email-like conve
emphasis, source code, and URL links. This forr
to produce the book you hold (which was writte
obeying just a few conventions (that are almos
email), a writer can write without much clutter,

The Txt2Html utility uses a block-level state ma
regular expressions, to identify and modify mar
Python's regular expression engine is moderate
only a couple seconds. In practice, Txt2Html is

documents. However, it is easy to imagine a no
converting multimegabyte documents and/or d
a high-volume Web site. In such a case, Pythor
expressions, would simply be too slow.

mx.TextTools can do everything regular expres
cannot. In particular, a taglist can contain recur
regular expressions cannot. The utility mxTypo
capabilities the prior example did not use. Rath
mxTypography.py utilizes a number of callback
match event. As well, mxTypography.py adds s
Something similar to these techniques is almos
updated over time (or simply to aid the initial d
application should.

mx.TextTools version of Typography()

```
from mx.TextTools import *
import string, sys

#-- List of all words with  markup,
ws, head_pos, loops = [], None, 0

#-- Define "emitter" callbacks for e
def emit_misc(tl,txt,l,r,s):
```

```
    ws.append(txt[l:r])
def emit_func(tl,txt,l,r,s):
    ws.append('<code>'+txt[l+1:r-1]+
def emit_modl(tl,txt,l,r,s):
    ws.append('<em><code>'+txt[l+1:r
def emit_emph(tl,txt,l,r,s):
    ws.append('<em>'+txt[l+1:r-1]+'<
def emit_strg(tl,txt,l,r,s):
    ws.append('<strong>'+txt[l+1:r-1
def emit_titl(tl,txt,l,r,s):
    ws.append('<cite>'+txt[l+1:r-1]+
def jump_count(tl,txt,l,r,s):
    global head_pos, loops
    loops = loops+1
    if head_pos is None: head_pos =
    elif head_pos == r:
        raise "InfiniteLoopError", \
              txt[l-20:l]+'{'+txt[l]
    else: head_pos = r

#-- What can appear inside, and what
punct_set = set("'!@#$%^&*()_-+=|\{}
markable = alphanumeric+whitespace+"
markable_func = set(markable+"*-_[]"
markable_modl = set(markable+"*-_'")
```

```
markable_emph = set(markable+"*_'[]"
markable_strg = set(markable+"-_'[]"
markable_titl = set(markable+"*-'[]"
markup_set    = set("-*'[]_")

#-- What can precede and follow mark
darkins = '(/"'
leadins = whitespace+darkins      #
darkouts = '/.),:;?!"'
darkout_set = set(darkouts)
leadouts = whitespace+darkouts    #
leadout_set = set(leadouts)

#-- What can appear inside plain wor
word_set = set(alphanumeric+'{}/@#$%
wordinit_set = set(alphanumeric+"$#+

#-- Define the word patterns (global
# Special markup
def markup_struct(lmark, rmark, call
    struct = \
      ( callback, Table+CallTag,
        ( (None, Is, lmark),
          (None, AllInSet, markables
          (None, Is, rmark),
```

```
          (None, IsInSet, leadout_se
          (None, Skip, -1,+1, MatchO
          (None, IsIn, x_post, Match
          (None, Skip, -1,+1, MatchO
        )
      )
    return struct
funcs   = markup_struct("'", "'", em
modules = markup_struct("[", "]", em
emphs   = markup_struct("-", "-", em
strongs = markup_struct("*", "*", em
titles  = markup_struct("_", "_", em

# All the stuff not specially markec
plain_words = \
 ( ws, Table+AppendMatch,
   ( (None, IsInSet,
          wordinit_set, MatchFail),
     (None, Is, "'",+1),
     (None, AllInSet, word_set,+1),
     (None, Is, "'", +2),
     (None, IsIn, "st",+1),
     (None, IsInSet,
          darkout_set,+1, MatchOk),
     (None, IsInSet,
```

```
        whitespace_set, MatchFail),
      (None, Skip, -1)
    ) )
# Catch some special cases
bullet_point = \
 ( ws, Table+AppendMatch,
    ( (None, Word+CallTag, "* "),
    ) )
horiz_rule = \
 ( None, Table,
    ( (None, Word, "-"*50),
      (None, AllIn, "-"),
    ) )
into_mark = \
 ( ws, Table+AppendMatch,
    ( (None, IsInSet, set(darkins)),
      (None, IsInSet, markup_set),
      (None, Skip, -1)
    ) )
stray_punct = \
 ( ws, Table+AppendMatch,
    ( (None, IsInSet, punct_set),
      (None, AllInSet, punct_set),
      (None, IsInSet, whitespace_set)
      (None, Skip, -1)
```

```
    ) )
leadout_eater = (ws, AllInSet+Appenc

#-- Tag all the (possibly marked-up)
tag_words = \
 ( bullet_point+(+1,),
   horiz_rule + (+1,),
   into_mark  + (+1,),
   stray_punct+ (+1,),
   emphs   + (+1,),
   funcs   + (+1,),
   strongs + (+1,),
   modules + (+1,),
   titles  + (+1,),
   into_mark+(+1,),
   plain_words +(+1,),             #
   leadout_eater+(+1,-1),          #
   (jump_count, Skip+CallTag, 0),  #
   (None, EOF, Here, -13)          #
 )
def Typography(txt):
    global ws
    ws = []    # clear the list befc
    tag(txt, tag_words, 0, len(txt),
    return string.join(ws, '')
```

```
if __name__ == '__main__':
    print Typography(open(sys.argv[1
```

mxTypographify.py reads through a string and d
of the markup patterns in tag_words. Or rather
application just will not know what action to tak
subtable matches, a callback function is called,
being appended to the global list ws. In the end

Several of the patterns given are mostly fallbac
table detects the condition where the next bit o
alone without abutting any words. In most case
a pattern, but *mxTypographify* has to do *somet*

Making sure that every subsequence is matched
are a few examples of matches and failures for
not match this subtable needs to match some o

```
-- spam       # matches "--"
& spam        # fails at "AllInSet" s
#@$ %% spam   # matches "#@$"
**spam        # fails (whitespace isn
```

After each success, the read-head is at the spa
After a failure, the read-head remains where it

Like stray_punct, emphs, funcs, strongs, plain_
in tag_words has its appropriate callback functi
they "emit" the match, along with surrounding
appended to their tuple; what this does is spec
That is, even if these patterns fail to match, we
positionto try matching against the other patter

After the basic word patterns each attempt a m
mxTypography.py, a "leadout" is the opposite o
might precede a word pattern, and the former
leadout_set includes whitespace characters, bu
and question mark, which might end a word. Th
As designed, it preserves exactly the whitespac
normalize whitespace here by emitting somethi
space always).

The jump_count is extremely important; we wi
enough to say that we *hope* the line never does

The EOF line is our flow control, in a way. The
that nothing is actually *done* with any match. T
is just a filler value that occupies the tuple posi
end of the read buffer. On success, the whole ta
succeeded, processing stops. EOF failure is mor
the end of our string, we jump -13 states (to b
starts over, hopefully with the read-head advan
start of the list of tuples, we continue eating su
exhausted (calling callbacks along the way).

The tag() call simply launches processing of the contained in txt). In our case, we do not care a is handled in callbacks. However, in cases wher tuple can be used to determine if there is reasc buffer.

DEBUGGING A TAG TABLE

Describing it is easy, but I spent a large numbe tables that would match every pattern I was int something it wasn't. While smart ASCII markup few complications (e.g., markup characters bei characters and other punctuation appearing in format that is complicated enough to warrant u have similar complications.

Without question, the worst thing that can go w above is that *none* of the listed states match fr happens, your program winds up in a tight infir so you cannot get at it with Python code directl process *countless* times during my first brush a

Fortunately, there is a solution to the infinite lo jump_count.

mxTypography.py infinite loop catcher

```
def jump_count(taglist,txt,l,r,subta
    global head_pos
    if head_pos is None: head_pos =
    elif head_pos == r:
        raise "InfiniteLoopError", \
              txt[1-20:1]+'{'+txt[1]
    else: head_pos = r
```

The basic purpose of jump_count is simple: We
has been run through multiple times without m
to check whether the last read-head position is
cannot get anywhere, since we have reached th
is fated to happen forever. mxTypography.py si
reports a little bit of buffer context to see what

It is also possible to move the read-head manu
position. To manipulate the read head in this fa
table items. But a better approach is to create
from a Python loop. This Python loop can look
the next call if no match occurred. Either way,
way than with the loop tag table approach, less

Not as bad as an infinite loop, but still undesira
when they are not supposed to or not match w
has to match, or we would have an infinite loop
examining this situation much easier. During de
changes to my emit_* callbacks to print or log

output from these temporary print statements, lies.

CONSTANTS

The *mx.TextTools* module contains constants fc characters. Many of these character classes are of these constants also has a set version prede character class that may be used in tag tables a obtain a character set from a (custom) characte

```
>>> from mx.TextTools import a2z, se
>>> varname_chars = a2z + '_'
>>> varname_set = set(varname_chars)
```

mx.TextTools.a2z
mx.TextTools.a2z_set

English lowercase letters ("abcdefghijklmnopqrs

mx.TextTools.A2Z
mx.TextTools.A2Z_set

English uppercase letters ("ABCDEFGHIJKLMNC

mx.TextTools.umlaute
mx.TextTools.umlaute_set

Extra German lowercase hi-bit characters.

mx.TextTools.Umlaute
mx.TextTools.Umlaute_set

Extra German uppercase hi-bit characters.

mx.TextTools.alpha
mx.TextTools.alpha_set

English letters (A2Z + a2z).

mx.TextTools.german_alpha
mx.TextTools.german_alpha_set

German letters (A2Z + a2z + umlaute + Umlau

mx.TextTools.number
mx.TextTools.number_set

The decimal numerals ("0123456789").

mx.TextTools.alphanumeric
mx.TextTools.alphanumeric_set

English numbers and letters (alpha + number).

mx.TextTools.white
mx.TextTools.white_set

Spaces and tabs (" \t\v"). This is more restricte

mx.TextTools.newline
mx.TextTools.newline_set

Line break characters for various platforms ("\r

mx.TextTools.formfeed
mx.TextTools.formfeed_set

Formfeed character ("\f").

mx.TextTools.whitespace
mx.TextTools.whitespace_set

Same as *string.whitespace* (white+newline+for

mx.TextTools.any
mx.TextTools.any_set

All characters (0x00-0xFF).

SEE ALSO: string.digits *130*; string.hexdigits *1*
string.uppercase *131*; string.letters *131*; string
string.printable *132*;

COMMANDS

Programming in *mx.TextTools* amounts mostly
tag table requires just one call to the *mx.TextT*
mini-languagesomething close to a specialized

Each tuple within a tag table contains several e

```
(tagobj, command[+modifiers], argume
         [,jump_no_match=MatchFail [
```

The "tag object" may be None, a callable objec
pattern may match, but nothing is added to a t
invoked. If a callable object (usually a function)
string is used, it is used to name a part of the t
mx.TextTools.tag().

A command indicates a type of pattern to matc
occurs in case of such a match. Some comman
to specify behaviors to take if they are reached
values that are allowed and how they are inter

Two jump conditions may optionally be specifie
defaults to MatchFailthat is, unless otherwise sp
causes the tag table as a whole to fail. If a valu
the specified number of states forward or backv
in forward branches. Branches backward will be

```
# Branch forward one state if next c
# ... branch backward three states i
tupX = (None, Is, 'X', +1, -3)
# assume all the tups are defined sc
tagtable = (tupA, tupB, tupV, tupW,
```

If no value is given for jump_match, branching

Version 2.1.0 of *mx.TextTools* adds named jum
maintain) than numeric offsets. An example is

```
tag_table = ('start',
             ('lowercase',AllIn,a2z,
             ('upper',AllIn,A2Z,'ski
             'skip',
             (None,AllIn,white+newli
             (None,AllNotIn,alpha+wh
             (None,EOF,Here,'start')
```

It is easy to see that if you were to add or rem
jump to, for example, skip than to change ever

UNCONDITIONAL COMMANDS

mx.TextTools.Fail
mx.TextTools.Jump

Nonmatch at this tuple. Used mostly for docum
Here or To placeholder. The tag tables below ar

```
table1 = ( ('foo', Is, 'X', MatchFai
table2 = ( ('foo', Is, 'X', +1, +2),
```

```
            ('Not_X', Fail, Here) )
```

The Fail command may be preferred if several c
condition needs to be documented explicitly.

Jump is equivalent to Fail, but it is often better
other; for example:

```
tup1 = (None, Fail, Here, +3)
tup2 = (None, Jump, To, +3)
```

mx.TextTools.Skip
mx.TextTools.Move

Match at this tuple, and change the read-head
relative amount, Move to an absolute offset (wi
example:

```
# read-head forward 20 chars, jump t
tup1 = (None, Skip, 20)
# read-head to position 10, and jump
tup2 = (None, Move, 10, 0, -4)
```

Negative offsets are allowed, as in Python list in

MATCHING PARTICULAR CHARACTERS

mx.TextTools.AllIn
mx.TextTools.AllInSet
mx.TextTools.AllInCharSet

Match all characters up to the first that is not ir
string while AllInSet uses a set as argument. Fc
to match CharSet objects. In general, the set o
The following are functionally the same:

```
tup1 = ('xyz', AllIn, 'XYZxyz')
tup2 = ('xyz', AllInSet, set('XYZxyz
tup3 = ('xyz', AllInSet, CharSet('XY
```

At least one character must match for the tuple

mx.TextTools.AllNotIn

Match all characters up to the first that *is* incluc
mx.TextTools does not include an AllNotInSet c
functionally the same (the second usually faste

```
from mx.TextTools import AllNotIn, A
```

```
tup1 = ('xyz', AllNotIn, 'XYZxyz')
tup2 = ('xyz', AllInSet, invset('xyz
```

At least one character must match for the tupl

mx.TextTools.Is

Match specified character. For example:

```
tup = ('X', Is, 'X')
```

mx.TextTools.IsNot

Match any one character except the specified c

```
tup = ('X', IsNot, 'X')
```

mx.TextTools.IsIn
mx.TextToo1s.IsInSet
mx.TextTools.IsInCharSet

Match exactly one character if it is in argument
a set as argument. For version 2.1.0, you may

In general, the set or CharSet form will be fast
functionally the same:

```
tup1 = ('xyz', IsIn, 'XYZxyz')
tup2 = ('xyz', IsInSet, set('XYZxyz'
tup3 = ('xyz', IsInSet, CharSet('XYZ
```

mx.TextTools.IsNotIn

Match exactly one character if it is *not* in argum
not include an 'AllNotInSet command. However
(the second usually faster):

```
from mx.TextTools import IsNotIn, Is
tup1 = ('xyz', IsNotIn, 'XYZxyz')
tup2 = ('xyz', IsInSet, invset('xyzX
```

MATCHING SEQUENCES

mx.TextTools.Word

Match a word at the current read-head position

```
tup = ('spam', Word, 'spam')
```

mx.TextTools.WordStart
mx.TextTools.sWordStart
mx.TextTools.WordEnd
mx.TextTools.sWordEnd

Search for a word, and match up to the point o
manner are extremely fast, and this is one of tl
commands sWordStart and sWordEnd use "sear
significantly faster).

WordStart and sWordStart leave the read-head
match succeeds. WordEnd and sWordEnd leave
word. On failure, the read-head is not moved fo

```
>>> from mx.TextTools import *
>>> s = 'spam and eggs taste good'
>>> tab1 = ( ('toeggs', WordStart, '
>>> tag(s, tab1)
(1, [('toeggs', 0, 9, None)], 9)
>>> s[0:9]
'spam and '
>>> tab2 = ( ('pasteggs', sWordEnd,
>>> tag(s, tab2)
(1, [('pasteggs', 0, 13, None)], 13)
>>> s[0:13]
```

```
'spam and eggs'
```

SEE ALSO: mx.TextTools.BMS() *307*; mx.TextTo

mx.TextTools.sFindWord

Search for a word, and match only that word. A
ignored. This command accepts a search object
head is positioned immediately after the match

```
>>> from mx.TextTools import *
>>> s = 'spam and eggs taste good'
>>> tab3 = ( ('justeggs', sFindWord,
>>> tag(s, tab3)
(1, [('justeggs', 9, 13, None)], 13)
>>> s[9:13]
'eggs'
```

SEE ALSO: mx.TextTools.sWordEnd *302*;

mx.TextTools.EOF

Match if the read-head is past the end of the st
argument Here, for example:

```
tup = (None, EOF, Here)
```

COMPOUND MATCHES

mx.TextTools.Table
mx.TextTools.SubTable

Match if the table given as argument matches a
difference between the Table and the SubTable
When the Table command is used, any matches
structure associated with the tuple. When SubT
current level taglist. For example:

```
>>> from mx.TextTools import *
>>> from pprint import pprint
>>> caps = ('Caps', AllIn, A2Z)
>>> lower = ('Lower', AllIn, a2z)
>>> words = ( ('Word', Table, (caps,
...           (None, AllIn, whitespa
>>> from pprint import pprint
>>> pprint(tag(s, words))
(0,
 [('Word', 0, 4, [('Caps', 0, 1, Nor
  ('Word', 5, 19, [('Caps', 5, 6, Nc
```

```
  ('Word', 20, 29, [('Caps', 20, 24,
  ('Word', 30, 35, [('Caps', 30, 32,
 ],
 35)
>>> flatwords = ( (None, SubTable, (
...               (None, AllIn, whit
>>> pprint (tag(s, flatwords))
(0,
 [('Caps', 0, 1, None),
  ('Lower', 1, 4, None),
  ('Caps', 5, 6, None),
  ('Lower', 6, 19, None),
  ('Caps', 20, 24, None),
  ('Lower', 24, 29, None),
  ('Caps', 30, 32, None),
  ('Lower', 32, 35, None)],
 35)
```

For either command, if a match occurs, the rea
match.

The special constant ThisTable can be used inst
recursively.

mx.TextTools.TableInList

mx.TextTools.SubTableInList

Similar to Table and SubTable except that the a
index). The advantage (and the danger) of this
added after the tuple definedin particular, the c
list_of_tables to allow recursion. Note, howeve
with the Table or SubTable commands and is us

SEE ALSO: mx.TextTools.Table *304*; mx.TextToo

mx.TextTools.Call

Match on any computable basis. Essentially, wh
parsing/matching is turned over to Python rath
function that is called must accept arguments s
pos is the current read-head position, and end
called function must return an integer for the n
from pos, the match is a success.

As an example, suppose you want to match at
make up a dictionary word. Perhaps an efficien
the dictionary word list. You might check dictio

```
tup = ('DictWord', Call, inDict)
```

Since the function inDict is written in Python, it *mx.TextTools* pattern tuple.

mx.TextTools.CallArg

Same as Call, except CallArg allows passing ad
dictionary example given in the discussion of C
maximum word length for a match:

```
tup = ('DictWord', Call, (inDict,['E
```

SEE ALSO: mx.TextTools.Call *305*;

MODIFIERS

mx.TextTools.CallTag

Instead of appending (tagobj, l, r, subtags) to t
function indicated as the tag object (which mus
The function called must accept the arguments
is the present taglist, s is the underlying string,
match, and subtags is the nested taglist. The fu
modify taglist or subtags as part of its action. F
include:

```
>>> def todo_flag(taglist, s, start,
...     sys.stderr.write("Fix issue
...
>>> tup = (todo_flag, Word+CallTag,
>>> tag('XXX more stuff', (tup,))
Fix issue at offset 0
(1, [], 3)
```

mx.TextTools.AppendMatch

Instead of appending (tagobj,start,end,subtags
append the match found as string. The produce
the same manner as "normal" taglist data struc
joining or for list processing styles.

```
>>> from mx.TextTools import *
>>> words = (('Word', AllIn+AppendMa
...          (None, AllIn, whitespac
>>> tag('this and that', words)
(0, ['this', 'and', 'that'], 13)
>>> join(tag('this and that', words)
'this-and-that'
```

SEE ALSO: string.split() *142*;

mx.TextTools.AppendToTagobj

Instead of appending (tagobj,start,end,subtags
the .append() method of the tag object. The ta
in Python 2.2+).

```
>>> from mx.TextTools import *
>>> ws = []
>>> words = ((ws, AllIn+AppendToTago
...           (None, AllIn, whitespac
>>> tag('this and that', words)
(0, [], 13)
>>> ws
[(None, 0, 4, None), (None, 5, 8, No
```

SEE ALSO: mx.TextTools.CallTag *305*;

mx.TextTools.AppendTagobj

Instead of appending (tagobj,start,end,subtags
append the tag object. The produced taglist is u
same manner as "normal" taglist data structure
joining or for list processing styles.

```
>>> from mx.TextTools import *
```

```
>>> words = (('word', AllIn+AppendTa
...           (None, AllIn, whitespac
>>> tag('this and that', words)
(0, ['word', 'word', 'word'], 13)
```

mx.TextTools.LookAhead

If this modifier is used, the read-head position
name suggests, this modifier allows you to crea
lookaheads.

```
>>> from mx.TextTools import *
>>> from pprint import pprint
>>> xwords = ((None, IsIn+LookAhead,
...            ('xword', AllIn, alpha
...            ('other', AllIn, alpha
...            (None, AllIn, whitespa
>>> pprint(tag('Xylophone trumpet xr
(0,
 [('xword', 0, 9, None),
  ('other', 10, 17, None),
  ('xword', 18, 22, None),
  ('other', 23, 29, None)],
 29)
```

CLASSES

mx.TextTools.BMS(word [,translate])
mx.TextTools.FS(word [,translate])
mx.TextTools.TextSearch(word [,translate [,al

Create a search object for the string word. This
expression. A search object has several methoc
string. The BMS name is short for "Boyer-Moore
name FS is reserved for accessing the "Fast Sea
both classes use Boyer-Moore. For *mx.TextTool*
.TextSearch() constructor.

If a translate argument is given, the searched s
equivalent to transforming the string with *string*

SEE ALSO: string.translate() *145*;

mx.TextTools.CharSet(definition)

Version 2.1.0 of *mx.TextTools* adds the Unicode
may be initialized to support character ranges,
definition="a-mXYZ". In most respects, CharSe

METHODS AND ATTRIBUTES

mx.TextTools.BMS.search(s [,start [,end]])
mx.TextTools.FS.search(s [,start [,end]])
mx.TextTools.TextSearch.search(s [,start [,en

Locate as a slice the first match of the search c
end are used, only the slice s[start:end] is cons
documentation that accompanies *mx.TextTools*
search object methods as indicating the length

mx.TextTools.BMS.find(s, [,start [,end]])
mx.TextTools.FS.find(s, [,start [,end]])
mx.TextTools.TextSearch.search(s [,start [,en

Similar to *mx.TextTools.BMS.search()*, except r
The behavior is similar to that of *string.find()*.

SEE ALSO: string.find() *135*; mx.TextTools.find

mx.TextTools.BMS.findall(s [,start [,end]])
mx.TextTools.FS.findall(s [,start [,end]])
mx.TextTools.TextSearch.search(s [,start [,en

Locate as slices *every* match of the search obje
and end are used, only the slice s[start:end] is

```
>>> from mx.TextTools import BMS, ar
>>> foosrch = BMS('FOO', upper(any))
>>> foosrch.search('foo and bar and 
(0, 3)
>>> foosrch.find('foo and bar and FO
0
>>> foosrch.findall('foo and bar anc
[(0, 3), (16, 19)]
>>> foosrch.search('foo and bar and 
(16, 19)
```

SEE ALSO: re.findall *245*; mx.TextTools.findall(

mx.TextTools.BMS.match
mx.TextTools.FS.match
mx.TextTools.TextSearch.match

The string that the search object will look for in

mx.TextTools.BMS.translate
mx.TextTools.FS.translate

mx.TextTools.TextSearch.match

The translation string used by the object, or No

mx.TextTools.CharSet.contains(c)

Return a true value if character c is in the Char

mx.TextTools.CharSet.search(s [,direction [,s

Return the position of the first CharSet charact
there is no match. You may specify a negative

SEE ALSO: re.search() *249*;

mx.TextTools.CharSet.match(s [,direction [,st

Return the length of the longest contiguous ma
s[start:end].

mx.TextTools.CharSet.split(s [,start=0 [,stop=

Return a list of substrings of s[start:end] divide

SEE ALSO: re.search() *249*;

mx.TextTools.CharSet.splitx(s [,start=0 [,stop

Like *mx.TextTools.CharSet.split()* except retain
elements.

mx.TextTools.CharSet.strip(s [,where=0 [,star

Strip all characters in s[start:stop] appearing ir

FUNCTIONS

Many of the functions in *mx.TextTools* are usec
higher-level utility functions that do not require
are listed under a separate heading and genera
string module.

mx.TextTools.cmp(t1, t2)

Compare two valid taglist tuples on their slice p

passes of *mx.TextTools.tag()*, or combined by c
string order. This custom comparison function i

```
>>> import mx.TextTools
>>> from pprint import pprint
>>> t1 = [('other', 10, 17, None),
...        ('other', 23, 29, None),
...        ('xword', 0, 9, None),
...        ('xword', 18, 22, None)]
>>> t1.sort(mx.TextTools.cmp)
>>> pprint(tl)
[('xword', 0, 9, None),
 ('other', 10, 17, None),
 ('xword', 18, 22, None),
 ('other', 23, 29, None)]
```

mx.TextTools.invset(s)

Identical to mx.TextTools.set(s, 0).

SEE ALSO: mx.TextTools.set() *310*;

mx.TextTools.set(s [,includechars=1])

Return a bit-position encoded character set. Bit
like InSet and AllInSet operate more quickly th
AllIn).

If includechars is set to 0, invert the character

SEE ALSO: mx.TextTools.invset() *310*;

mx.TextTools.tag(s, table [,start [,end [,taglist

Apply a tag table to a string. The return value i
success is a binary value indicating whether the
after the match attempt. Even on a nonmatch
advanced to some degree by member tuples m
data structure generated by application. Modifie
the composition of taglist; but in the normal ca
of the form (tagname, start, end, subtaglist).

Assuming a "normal" taglist is created, tagnam
object in a tuple within the tag table. start and
subtaglist is either None or a taglist for a subta

If start or end are given as arguments to *mx.Te*
slice s[start:end] (or s[start:] if only start is us
object is used instead of a new list. This allows
example. If None is passed as taglist, no taglist

See the application examples and command illu
mx.TextTools.tag().

UTILITY FUNCTIONS

mx.TextTools.charsplit(s, char, [start [,end]])

Return a list split around each char. Similar to s
arguments start and end are used, only the slic

SEE ALSO: string.split() *142*; mx.TextTools.sets

mx.TextTools.collapse(s, sep=' ')

Return a string with normalized whitespace. Th
(s),sep), but faster.

```
>>> from mx.TextTools import collaps
>>> collapse('this and that','-')
'this-and-that'
```

SEE ALSO: string.join() *137*; string.split() *142*;

mx.TextTools.countlines(s)

Returns the number of lines in s in a platform-p
style), LF (Unix-style), or CRLF (DOS-style), in

SEE ALSO: FILE.readlines() *17*; mx.TextTools.s

mx.TextTools.find(s, search_obj, [start, [,end]

Return the position of the first match of search_
and end are used, only the slice s[start:end] is
search object method of the same name; the s
synonyms:

```
from mx.TextTools import BMS, find
s = 'some string with a pattern in i
pos1 = find(s, BMS('pat'))
pos2 = BMS('pat').find(s)
```

SEE ALSO: string.find() *135*; mx.TextTools.BMS

mx.TextTools.findall(s, search_obj [,start [,en

Return as slices *every* match of search_obj aga
are used, only the slice s[start:end] is consider
object method of the same name; the syntax is
synonyms:

```
from mx.TextTools import BMS, findal
s = 'some string with a pattern in i
pos1 = findall(s, BMS('pat'))
pos2 = BMSCpat').findall(s)
```

SEE ALSO: mx.TextTools.find() *312*; mx.TextTo

mx.TextTools.hex2str(hexstr)

Returns a string based on the hex-encoded stri

```
>>> from mx.TextTools import hex2str
>>> str2hex('abc')
'616263'
>>> hex2str('616263')
'abc'
```

SEE ALSO: mx.TextTools.str2hex() *315*;

mx.TextTools.is_whitespace(s [,start [,end]])

Returns a Boolean value indicating whether s[s
start and end are optional, and will default to 0

mx.TextTools.isascii(s)

Returns a Boolean value indicating whether s c

mx.TextTools.join(joinlist [,sep="" [,start [,en

Return a string composed of slices from other s
form (s, start, end, ...) each indicating the sou
Negative offsets do not behave like Python slic
item tuple contains extra entries, they are igno

If the optional argument sep is specified, a deli
start and end are specified, only joinlist[start:e

```
>>> from mx.TextTools import join
>>> s = 'Spam and eggs for breakfast
>>> t = 'This and that for lunch'
>>> j1 = [(s, 0, 4), (s, 9, 13), (t,
>>> join(j1, '/', 1, 4)
'/eggs/This/that'
```

SEE ALSO: string.join() *137*;

mx.TextTools.lower(s)

Return a string with any uppercase letters conv
string.lower() , but much faster.

SEE ALSO: string.lower() *138*; mx.TextTools.up

mx.TextTools.prefix(s, prefixes [,start [,stop [,

Return the first prefix in the tuple prefixes that
specified, only operate on the slice s[start:end]

If a translate argument is given, the searched s
equivalent to transforming the string with strin

```
>>> from mx.TextTools import prefix
>>> prefix('spam and eggs', ('spam',
'spam'
```

SEE ALSO: mx.TextTools.suffix() *316*;

mx.TextTools.multireplace(s ,replacements [,s

Replace multiple nonoverlapping slices in s with
tuples of the form (new, left, right). Indexing is
replacement changes the length of the result. I
the slice s[start:end].

```
>>> from mx.TextTools import findall
>>> s = 'spam, bacon, sausage, and s
>>> repls = [('X',l,r) for l,r in fi
>>> multireplace(s, repls)
'X, bacon, sausage, and X'
>>> repls
[('X', 0, 4), ('X', 26, 30)]
```

mx.TextTools.replace(s, old, new [,start [,stop

Return a string where the pattern matched by s
start and end are specified, only operate on the
than *string.replace()* , since a search object is u

```
>>> from mx.TextTools import replace
>>> s = 'spam, bacon, sausage, and s
>>> spam = BMS('spam')
>>> replace(s, spam, 'eggs')
'eggs, bacon, sausage, and eggs'
>>> replace(s, spam, 'eggs', 5)
' bacon, sausage, and eggs'
```

SEE ALSO: string.replace() *139*; mx.TextTools.I

mx.TextTools.setfind(s, set [,start [,end]])

Find the first occurence of any character in set.
end is specified, look only in s[start:end]. The a

```
>>> from mx.TextTools import *
>>> s = 'spam and eggs'
>>> vowel = set('aeiou')
>>> setfind(s, vowel)
2
>>> setfind(s, vowel, 7, 10)
9
```

SEE ALSO: mx.TextTools.set() *310*;

mx.TextTools.setsplit(s, set [,start [,stop]])

Split s into substrings divided at any characters
substrings of s[start:]; if end is specified, use s

SEE ALSO: string.split() *142*; mx.TextTools.set(

mx.TextTools.setsplitx(text,set[,start =0, stop

Split s into substrings divided at any characters
returned list. Adjacent characters in set are ret
specified, create a list of substrings of s[start:]
argument set must be a set.

```
>>> s = 'do you like spam'
>>> setsplit(s, vowel)
['d', ' y', ' l', 'k', ' sp', 'm']
>>> setsplitx(s, vowel)
['d', 'o', ' y', 'ou', ' l', 'i', 'k
```

SEE ALSO: string.split() *142*; mx.TextTools.set(

mx.TextTools.splitat(s, char, [n=1 [,start [end]

Return a 2-element tuple that divides s around
specified, only operate on the slice s[start:end]

```
>>> from mx.TextTools import splitat
>>> s = 'spam, bacon, sausage, and s
>>> splitat(s, 'a', 3)
('spam, bacon, s', 'usage, and spam'
>>> splitat(s, 'a', 3, 5, 20)
(' bacon, saus', 'ge')
```

mx.TextTools.splitlines(s)

Return a list of lines in s. Line-ending combinat
recognized in any combination, which makes th
string.split(s,"\n") or *FILE.readlines()*.

SEE ALSO: string.split() *142*; FILE.readlines() *1*
mx.TextTools.countlines() *311*;

mx.TextTools.splitwords(s)

Return a list of whitespace-separated words in

SEE ALSO: string.split() *142*;

mx.TextTools.str2hex(s)

Returns a hexadecimal representation of a strir
s.encode("hex").

SEE ALSO: "".encode() *188*; mx.TextTools.hex2

mx.TextTools.suffix(s, suffixes [,start [,stop [,

Return the first suffix in the tuple suffixes that
specified, only operate on the slice s[start:end]

If a translate argument is given, the searched s
equivalent to transforming the string with *strin*

```
>>> from mx.TextTools import suffix
>>> suffix('spam and eggs', ('spam',
'eggs'
```

SEE ALSO: mx.TextTools.prefix() *313*;

mx.TextTools.upper(s)

Return a string with any lowercase letters conv
string.upper(), but much faster.

SEE ALSO: string.upper() *146*; mx.TextTools.lo

4.3.3 High-Level EBNF Parsing

SimpleParse • A Parser Generator for mx.

SimpleParse is an interesting tool. To use this r

module installed. While there is nothing you ca
mx.TextTools by itself, *SimpleParse* is often mu
modules to provide higher-level APIs for *mx.Te*
useful of these, and the only one that this book
written against *SimpleParse* version 1.0, but th
features of 2.0. Version 2.0 is fully backward co

SimpleParse substitutes an EBNF-style gramma
mx.TextTools tag tables. Or more accurately, *S*
based on friendlier and higher-level EBNF gram
and modify tag tables before passing them to *n*
want to stick wholly with *SimpleParse*'s EBNF v
grammatical description of the text format.

An application based on *SimpleParse* has two m
that defines the structure of a processed text. T
generated *mx.TextTools* taglist. *SimpleParse* 2.
taglists present a data structure that is quite ea
tools in *SimpleParse* 2.0 are not covered here,
mx.TextTools illustrate such traversal.

Example: Marking up smart ASCII (Redux)

Elsewhere in this book, applications to process
Appendix D lists the Txt2Html utility, which use
paragraphs and regular expressions for identify
example was given in the discussion of *mx.Tex*

table was developed to recognize inline markup
grammar is yet another way to perform the sar
styles will highlight a number of advantages tha
concise, and applications built around it can be

The application simpleTypography.py is quite si
in creating a grammar to describe smart ASCII
read, but designing one *does* require a bit of th

typography.def

```
para            := (plain / markup)+
plain           := (word / whitespace
<whitespace>    := [ \t\r\n]+
<alphanums>     := [a-zA-Z0-9]+
<word>          := alphanums, (wordpu
<wordpunct>     := [-_]
<contraction>   := "'", ('am'/'clock'
markup          := emph / strong / mc
emph            := '-', plain, '-'
strong          := '*', plain, '*'
module          := '[', plain, ']'
code            := "'", plain, "'"
title           := '_', plain, '_'
<punctuation>   := (safepunct / mdash
```

```
<mdash>          := '--'
<safepunct>      := [!@#$%^&()+=|\{}:;
```

This grammar is almost exactly the way you wo
verbally, which is a nice sort of clarity. A paragr
marked-up text. Plaintext consists of some coll
Marked-up text might be emphasized, or strong
Strongly emphasized text is surrounded by aste
what a "word" really is, or just what a contracti
syntax of EBNF doesn't get in the way.

Notice that some declarations have their left sid
productions will not be written to the taglistthis
mx.Texttools tag table. Of course, if a producti
cannot be, either. By omitting some production
structure is produced (with only those element

In contrast to the grammar above, the same sc
using regular expressions. This is what the Txt2
program does. But this terseness is much hard
code below expresses largely (but not precisely

Python regexes for smart ASCII markup

```
# [module] names
re_mods =   r"""([\(\s'/">]|^)\[(.*?
```

```
# *strongly emphasize* words
re_strong = r"""([\(\s'/"]|^)\*(.*?)
# -emphasize- words
re_emph =   r"""([\(\s'/"]|^)-(.*?)-
# _Book Title_ citations
re_title =  r"""([\(\s'/"]|^)_(.*?)_
# 'Function()' names
re_funcs =  r"""([\(\s/"]|^)'(.*?)'(
```

If you discover or invent some slightly new vari
with the EBNF grammar than with those regula
therefore *mx.TextTools*will generally be even fa
patterns.

GENERATING AND USING A TAGLIST

For simpleTypography.py, I put the actual gram
is a good organization to use. Changing the gra
changing the application logic, and the files refl
string, so in principle you could include it in the
generate it in some way).

Let us look at the entirecompacttagging applica

simpleTypography.py

```
from sys import stdin, stdout, stder
from simpleparse import generator
from mx.TextTools import TextTools
from typo_html import codes
from pprint import pprint

src = stdin.read()
decl = open('typography.def').read()
parser = generator.buildParser(decl)
taglist = TextTools.tag(src, parser)
pprint(taglist, stderr)

for tag, beg, end, parts in taglist[
    if tag == 'plain':
        stdout.write(src[beg:end])
    elif tag == 'markup':
        markup = parts[0]
        mtag, mbeg, mend = markup[:3
        start, stop = codes.get(mtag

        stdout.write(start + src[m
    else:
        raise TypeError, "Top level
```

With version 2.0 of *SimpleParse*, you may use a taglist:

```
from simpleparse.parser import Parse
parser = Parser(open('typography.def
taglist = parser.parse(src)
```

Here is what it does. First read in the grammar
grammar. The generated parser is similar to the
mxTypography.py module discussed earlier (bu
structure). Next, apply the tag table/parser to t
through the taglist, and emit some new marked
anything else desired with each production enc

For the particular grammar used for smart ASC
fall into either a "plain" production or a "marku
across a single level in the taglist (except when
markup production, such as "title"). But a more
programming languagescould easily recursively
production names at every level. For example,
codes, this recursive style would probably be us
figuring out how to adjust the grammar (hint: F
mutually recursive).

The particular markup codes that go to the out
not essential, reasons. A little trick of using a d
(although the otherwise case remains too narro
organization is that we might in the future wan
say, HTML, DocBook, LaTeX, or others. The particu

like:

typo_html.py

```
codes = \
{ 'emph'    : ('<em>', '</em>'),
  'strong'  : ('<strong>', '</strong>
  'module'  : ('<em><code>', '</code>
  'code'    : ('<code>', '</code>'),
  'title'   : ('<cite>', '</cite>'),
}
```

Extending this to other output formats is straig

THE TAGLIST AND THE OUTPUT

The *tag table* generated from the grammar in t
includes numerous recursions. Only the excepti
manuallet alone automatedmodification of tag t
average user need not even look at these tags,
simpleTypography.py.

The *taglist* produced by applying a grammar, in
run of simpleTypography.py against a small inp

```
% python simpleTypography.py < p.txt
(1,
 [('plain', 0, 15, []),
  ('markup', 15, 27, [('emph', 15, 2
  ('plain', 27, 42, []),
  ('markup', 42, 51, [('module', 42,
  ('plain', 51, 55, []),
  ('markup', 55, 70, [('code', 55, 7
  ('plain', 70, 90, []),
  ('markup', 90, 96, [('strong', 90,
  ('plain', 96, 132, []),
  ('markup', 132, 145, [('title', 13
  ('plain', 145, 174, [])],
 174)
```

Most productions that were satisfied are not wr
needed for the application. You can control this
without angle braces on the left side of their de
expect:

```
% cat p.txt
Some words are -in italics-, others
name [modules] or 'command lines'.
Still others are *bold* -- that's ho
it goes. Maybe some _book titles_.
```

```
And some in-fixed dashes.
% cat p.html
Some words are <em>in italics</em>,
name <em><code>modules</code></em> c
Still others are <strong>bold</stron
it goes. Maybe some <cite>book title
And some in-fixed dashes.
```

GRAMMAR

The language of *SimpleParse* grammars is itsel
grammar. In principle, you could refine the lang
variable declaration in bootstrap.py, or simplep
example, extended regular expressions, W3C X
integer occurrence quantification. To specify tha
use the following declaration in *SimpleParse*:

```
foos := foo, foo, foo, foo?, foo?, f
```

Hypothetically, it might be more elegant to writ

```
foos := foo{3,7}
```

In practice, only someone developing a custom

reason to fiddle quite so deeply; "normal" prog
defined by default. Nonetheless, taking a look a
in understanding the module.

DECLARATION PATTERNS

A *SimpleParse* grammar consists of a set of one
generally occurs on a line by itself; within a line
to improve readability. A common strategy is to
other use of internal whitespace is acceptable.
assignment symbol ":=", followed by a definitio
declaration, following an unquoted "#" (just as

In contrast to most imperative-style programm
occur in any order. When a parser generator's .
level" of the grammar is given as an argument.
call of the form:

```
from simpleparse import generator
parser = generator.buildParser(decl)
from mx.TextTools import TextTools
taglist = TextTools.tag(src, parser)
```

Under *SimpleParse* 2.0, you may simplify this t

```
from simpleparse.parser import Parse
```

```
parser = Parser(decl,'toplevel')
taglist = parser.parse(src)
```

A left side term may be surrounded by angle br
from being written into a taglist produced by *m*
"unreported" production. Other than in relation
acts just like a reported one. Either type of terr
productions in the same manner (without angle

In *SimpleParse* 2.0 you may also use reversed
production, but not the production itself. As wit
functions normally in matching inputs; it differs

```
PRODUCTIONS                    TAGLIST
-------------------------------------
a    := (b,c)                  ('a', l, r, [
b    := (d,e)                      ('b', l,
c    := (f,g)                      ('c', l,
-------------------------------------
a    := (b,c)                  ('a', l, r, [
<b> := (d,e)                       # no b, a
c    := (f,g)                      ('c', l,
-------------------------------------
# Only in 2.0+                 ('a', l, r, [
a    := (b,c)                      # no b, b
>b< := (d,e)                       ('d', l,
```

```
c    := (f,g)                         ('e', l,
                                      ('c', l,
-----------------------------------------------
```

The remainder of the documentation of the *Sim*
occur on the right sides of declarations. In addi
another production may occur anywhere any el
recursive relations to one another.

LITERALS

Literal string

A string enclosed in single quotes matches the
used for the characters \a, \b, \f, \n, \r, \t, and
may used. To include a literal backslash, it shou

```
foo := "bar"
```

Character class: "[", "]"

Specify a set of characters that may occur at a
be enumerated with no delimiter. A range of ch
Multiple ranges are allowed within a class.

To include a "]" character in a character class, r
character must be either the first (after the opt

```
varchar := [a-zA-Z_0-9]
```

QUANTIFIERS

Universal quantifier: "*"

Match zero or more occurrences of the precedir
precedence than alternation or sequencing; grc
scope as well.

```
any_Xs     := "X"*
any_digits := [0-9]*
```

Existential quantifier: "+"

Match one or more occurrences of the precedin
precedence than alternation or sequencing; grc
scope as well.

```
some_Xs     := "X"+
some_digits := [0-9]+
```

Potentiality quantifier: "?"

Match at most one occurrence of the preceding
precedence than alternation or sequencing; grc
scope as well.

```
maybe_Xs      := "X"?
maybe_digits := [0-9]?
```

Lookahead quantifier: "?"

In *SimpleParse* 2.0+, you may place a questior
but should not actually claim the pattern. As wi
positive or negative lookahead assertions.

```
next_is_Xs          := ?"X"
next_is_not_digits := ?-[0-9]
```

Error on Failure: "!"

In *SimpleParse* 2.0+, you may cause a descript
does not match, rather than merely stopping pa

```
require_Xs    := "X"!
```

```
require_code := ([A-Z]+, [0-9])!
contraction := "'", ('clock'/'d'/'ll
```

For example, modifying the contraction product
every apostrophe is followed by an ending. Sin
like:

```
% python typo2.py < p.txt
Traceback (most recent call last):
[...]
simpleparse.error.ParserSyntaxError:
Failed parsing production "contracti
Expected syntax: ('clock'/'d'/'ll'/'
Got text: 'command lines'. Still oth
```

STRUCTURES

Alternation operator: "/"

Match the first pattern possible from several alt
patterns to match. Some EBNF-style parsers wi
SimpleParse more simply matches the *first* pos

```
>>> from mx.TextTools import tag
```

```
>>> from simpleparse import generatc
>>> decl = '''
... short := "foo", " "*
... long  := "foobar", " "*
... sl    := (short / long)*
... ls    := (long / short)*
... '''
>>> parser = generator.buildParser(c
>>> tag('foo foobar foo bar', parser
[('short', 0, 4, []), ('short', 4, 7
>>> parser = generator.buildParser(c
>>> tag('foo foobar foo bar', parser
[('short', 0, 4, []), ('long', 4, 11
```

Sequence operator: ","

Match the first pattern followed by the second p
present, ...). Whenever a definition needs seve
sequence operator is used.

```
term := someterm, [0-9]*, "X"+, (oth
```

Negation operator: "-"

Match anything that the next pattern *does not*
simple term or a compound expression.

```
nonletters    := -[a-zA-Z]
nonfoo        := -foo
notfoobarbaz := -(foo, bar, baz)
```

An expression modified by the negation operat
expression with a negative lookahead assertion

```
>>> from mx.TextTools import tag
>>> from simpleparse import generatc
>>> decl = '''not_initfoo : = [ \t]*
>>> p = generator.buildParser(decl).
>>> tag(' foobar and baz', p)      #
(0, [], 0)
>>> tag(' bar, foo and baz', p)    #
(1, [], 5)
>>> tag(' bar foo and baz', p)     #
(1, [], 17)
```

Grouping operators: "(", ")"

Parentheses surrounding any pattern turn that
larger expression). Quantifiers and operators re

one is defined, otherwise to the adjacent literal

```
>>> from mx.TextTools import tag
>>> from simpleparse import generatc
>>> decl = '''
... foo       := "foo"
... bar       := "bar"
... foo_bars  := foo, bar+
... foobars   := (foo, bar)+
... '''
>>> p1 = generator.buildParser(decl)
>>> p2 = generator.buildParser(decl)
>>> tag('foobarfoobar', p1)
(1, [('foo', 0, 3, []), ('bar', 3, 6
     ('foo', 6, 9, []), ('bar', 9, 1
>>> tag('foobarfoobar', p2)
(1, [('foo', 0, 3, []), ('bar', 3, 6
>>> tag('foobarbarbar', p1)
(1, [('foo', 0, 3, []), ('bar', 3, 6
>>> tag('foobarbarbar', p2)
(1, [('foo', 0, 3, []), ('bar', 3, 6
     ('bar', 6, 9, []), ('bar', 9, 1
```

USEFUL PRODUCTIONS

In version 2.0+, *SimpleParse* includes a numbe
your grammars. See the examples and docume
on the many included productions and their usa

The included productions, at the time of this wr

simpleparse.common.calendar_names

Locale-specific names of months, and days of t

simpleparse.common.chartypes

Locale-specific categories of characters, such as
locale_decimal_point, and so on.

simpleparse.common.comments

Productions to match comments in a variety of
end-of-line comments (Python, Bash, Perl, etc.
others.

simpleparse.common.iso_date

Productions for strictly conformant ISO date an

simpleparse.common.iso_date_loose

Productions for ISO date and time formats with
formatting.

simpleparse.common.numbers

Productions for common numeric formats, such
numbers, and so on.

simpleparse.common.phonetics

Productions to match phonetically spelled word
bravo, charlie, ..." spelling is the only style sup

simpleparse.common.strings

Productions to match quoted strings as used in

simpleparse.common.timezone_names

Productions to match descriptions of timezones
data/time fields.

GOTCHAS

There are a couple of problems that can easily
you are having problems in your application, ke

1. **Bad recursion. You might fairly naturall**

   ```
   a := b, a?
   ```

   **Unfortunately, if a long string of b rules
   can either exceed the C-stack's recursio
   memory to construct nested tuples. Use**

   ```
   a := b+
   ```

   **This will grab all the b productions in or
   parse out each b if necessary).**

- Quantified potentiality. That is a mouthful; co

```
a := (b? / c)*
x := (y?, z?)+
```

The first alternate b? in the firstand both y? an

characters (if a b or y or z do not occur at the c
possible" zero-width patterns, you get into an i
always simple; it might not be b that is qualifie
productions *in* b productions, etc.).

- No backtracking. Based on working with regul
productions to use backtracking. They do not. F

```
a := ((b/c)*, b)
```

If this were a regular expression, it would matc
match the final b. As a *SimpleParse* production,
productions occur, they will be claimed by (b/c)

4.3.4 High-Level Programmatic Parsing

PLY • Python Lex-Yacc

One module that I considered covering to roun
module. This module is both widely used in the
However, I believe that the audience of this boo
Beazley's *PLY* module than with the older *Spark*

In the documentation accompanying *PLY*, Beazl
Spark on his design and development. While th
*Spark*the APIs are significantly differentthere is

module. Both modules require a very different
do *mx.TextTools, SimpleParse*, or the state ma
particular, both *PLY* and *Spark* make heavy use
state machines out of specially named variable

Within an overall similarity, *PLY* has two main a
context. The first, and probably greatest, advar
PLY has implemented some rather clever optim
for repeated runsthe main speed difference lies
slightly less powerful, parsing algorithm. For te
compiler development), *PLY's* LR parsing is pler

A second advantage *PLY* has over every other F
flexible and fine-grained error reporting and err
processing context, this is particularly importar

For compiling a programming language, it is ge
the case of even small errors. But for processin
you usually want to be somewhat tolerant of m
possible from a text automatically is frequently
job of handling "allowable" error conditions gra

PLY consists of two modules: a lexer/tokenizer
choice of names is taken from the popular C-or
correspondingly similar. Parsing with *PLY* usuall
at the beginning of this chapter: (1) Divide the
using lex.py. (2) Generate a parse tree from th

When processing text with *PLY*, it is possible to
event. Depending on application requirements,
SimpleParse. For example, each time a specific
modify the stored token according to whatever
different application action. Likewise, during pa
constructed, the node can be modified and/or c
SimpleParse simply delivers a completed parse
separately. However, while *SimpleParse* does no
PLY does, *SimpleParse* offers a higher-level and
the two modules is full of pros and cons.

Example: Marking up smart ASCII (yet again)

This chapter has returned several times to appl
machine in Appendix D; a functionally similar e
with *SimpleParse*. This email-like markup forma
presents just enough complications to make for
techniques and libraries. In many ways, an app
version aboveboth use grammars and parsing s

GENERATING A TOKEN LIST

The first step in most *PLY* applications is the cre
by a series of regular expressions attached to s
By convention, the *PLY* token types are in all ca

string is merely assigned to a variable. If actior
the rule name is defined as a function, with the
passed to the function is a LexToken object (wi
may be modified and returned. The pattern is c

wordscanner.py

```
# List of token names. This is alway
tokens = [ 'ALPHANUMS','SAFEPUNCT','
           'UNDERSCORE','APOSTROPHE',

# Regular expression rules for simpl
t_ALPHANUMS     = r"[a-zA-Z0-9]+"
t_SAFEPUNCT     = r'[!@#$%^&()+=|\{}
t_BRACKET       = r'[][]'
t_ASTERISK      = r'[*]'
t_UNDERSCORE    = r'_'
t_APOSTROPHE    = r"'"
t_DASH          = r'-'

# Regular expression rules with acti
def t_newline(t):
    r"\n+"
    t.lineno += len(t.value)
```

```
# Special case (faster) ignored char
t_ignore = " \t\r"

# Error handling rule
def t_error(t):
    sys.stderr.write("Illegal charac
                     % (t.value[0],
    t.skip(1)

import lex, sys
def stdin2tokens():
    lex.input(sys.stdin.read())
    toklst = []
    while 1:
        t = lex.token()
        if not t: break    # No more
        toklst.append(t)
    return toklst

if __name__=='__main__':
    lex.lex()
    for t in stdin2tokens():
        print '%s<%s>' % (t.value.lj
```

You are required to list the token types you wis
such token, and any special patterns that are n
variable or as a function. After that, you just in
off sequentially. Let us look at some results:

```
% cat p.txt
-Itals-, [modname]--let's add ~ unde
% python wordscanner.py < p.txt
Illegal character '~' (1)
-              <DASH>
Itals          <ALPHANUMS>
-              <DASH>
,              <SAFEPUNCT>
[              <BRACKET>
modname        <ALPHANUMS>
]              <BRACKET>
-              <DASH>
-              <DASH>
let            <ALPHANUMS>
'              <APOSTROPHE>
s              <ALPHANUMS>
add            <ALPHANUMS>
underscored    <ALPHANUMS>
var            <ALPHANUMS>
-              <UNDERSCORE>
```

```
name                <ALPHANUMS>
.                   <SAFEPUNCT>
```

The output illustrates several features. For one
nondiscarded substring as constituting some to
tilde character is handled gracefully by being or
something different if desired, of course. White
tokenizerthe special t-ignore variable quickly ig
function contains some extra code to maintain

The simple tokenizer above has some problems
dash or to mark italicized phrases; apostrophes
for a function name; underscores can occur bot
Readers who have used *Spark* will know of its c
inheritance; *PLY* cannot do that, but it can utili
exactly the same effect:

wordplusscanner.py

```
"Enhanced word/markup tokenization"
from wordscanner import  *
tokens.extend(['CONTRACTION','MDASH'
t_CONTRACTION   = r"(?<=[a-zA-Z])'(a
t_WORDPUNCT     = r'(?<=[a-zA-Z0-9])
def t_MDASH(t): # Use HTML style mda
```

```
    r'--'
    t.value = '&mdash;'
    return t

if __name__=='__main__':
    lex.lex()
    for t in stdin2tokens():
        print '%s<%s>' % (t.value.lj
```

Although the tokenization produced by wordsca
grammar rules, producing more specific tokens
In the case of t_MDASH(), wordplusscanner.py
recognition:

```
% python wordplusscanner.py < p.txt
Illegal character '~' (1)
-              <DASH>
Itals          <ALPHANUMS>
-              <DASH>
,              <SAFEPUNCT>
[              <BRACKET>
modname        <ALPHANUMS>
]              <BRACKET>
&mdash;        <MDASH>
let            <ALPHANUMS>
```

```
's              <CONTRACTION>
add             <ALPHANUMS>
underscored     <ALPHANUMS>
var             <ALPHANUMS>
_               <WORDPUNCT>
name            <ALPHANUMS>
.               <SAFEPUNCT>
```

Parsing a token list

A parser in *PLY* is defined in almost the same m
named functions of the form p_rulename() are
to match (or a disjunction of several such patte
YaccSlice object, which is list-like in assigning e
indexed position.

The code within each function should assign a u
t[1:]. If you would like to create a parse tree o
class of some sort and assign each right-hand r
example:

```
def p_rulename(t):
    'rulename : somerule SOMETOKEN c
    #   ^           ^          ^
    #  t[0]        t[1]       t[2]
```

```
    t[0] = Node('rulename', t[1:])
```

Defining an appropriate Node class is left as an
would be a traversable tree structure.

It is fairly simple to create a set of rules to com
wordplusscanner.py. In the sample application,
markupbuilder.py simply creates a list of match
codes. Other data structures are possible too, a
time a rule is matched (e.g., write to STDOUT).

markupbuilder.py

```
import yacc
from wordplusscanner import *

def p_para(t):
    '''para : para plain
            | para emph
            | para strong
            | para module
            | para code
            | para title
            | plain
            | emph
```

```
                | strong
                | module
                | code
                | title '''
    try:    t[0] = t[1] + t[2]
    except: t[0] = t[1]

def p_plain(t):
    '''plain : ALPHANUMS
             | CONTRACTION
             | SAFEPUNCT
             | MDASH
             | WORDPUNCT
             | plain plain '''
    try:    t[0] = t[1] + t[2]
    except: t[0] = [t[1]]

def p_emph(t):
    '''emph : DASH plain DASH'''
    t[0] = ['<i>'] + t[2] + ['</i>']

def p_strong(t):
    '''strong : ASTERISK plain ASTEF
    t[0] = ['<b>'] + t[2] + ['</b>']
```

```
def p_module(t):
    '''module : BRACKET plain BRACKE
    t[0] = ['<em><tt>'] + t[2] + ['<

def p_code(t):
    '''code : APOSTROPHE plain APOST
    t[0] = ['<code>'] + t[2] + ['</c

def p_title(t):
    '''title : UNDERSCORE plain UNDE
    t[0] = ['<cite>'] + t[2] + ['</c

def p_error(t):
    sys.stderr.write('Syntax error a
                      % (t.value,t.li

if __name__=='__main__':
    lex.lex()                   # Build
    yacc.yacc()                 # Build
    result = yacc.parse(sys.stdin.re
    print result
```

The output of this script, using the same input

```
% python markupbuilder.py < p.txt
```

```
Illegal character '~' (1)
['<i>', 'Itals', '</i>', ',', '<em><
'</tt></em>', '&mdash;', 'let', "'s"
'var', '_', 'name', '.']
```

One thing that is less than ideal in the *PLY* grar
SimpleParse or another EBNF library, we might

```
plain := (ALPHANUMS | CONTRACTION |
```

Quantification can make declarations more dire
using self-referential rules whose left-hand terr
is similar to recursive definitions, for example:

```
plain : plain plain
      | OTHERSTUFF
```

For example, markupbuilder.py, above, uses thi

If a tree structure were generated in this parse
containing lower plain nodes (and terminal leav
Traversal would need to account for this possib
issue, in this case. A particular plain object mig
smaller lists, but either way it is a list by the tir

LEX

A *PLY* lexing module that is intended as suppor
A lexing module that constitutes a stand-alone

1. **Import the *lex* module:**

```
import lex
```

- Define a list or tuple variable tokens that cont allowed to produce. A list may be modified in-p an importing module; for example:

```
tokens = ['FOO', 'BAR', 'BAZ', 'FLAM
```

- Define one or more regular expression pattern tokens should have a corresponding pattern; ot corresponding substrings will not be included ir

Token patterns may be defined in one of two wa string to a specially named variable. (2) By def docstring is a regular expression string. In the is matched. In both styles, the token name is p it should return the LexToken object passed to do not wish to include the token in the token st

```
t_FOO = r"[Ff] [Oo]{1,2}"
t_BAR = r"[Bb][Aa][Rr]"
def t_BAZ(t):
```

```
    r"([Bb] [Aa] [Zz])+"
    t.value = 'BAZ'     # canonical
    return t
def t_FLAM(t):
    r"(FLAM|flam)*"
    # flam's are discarded (no retur
```

Tokens passed into a pattern function have thre
contains the current line number within the stri
change the reported position, even if the token
normally the string matched by the regular exp
like a tuple or instance, may be assigned instea
string naming the token (the same as the part

There is a special order in which various token
patterns used, several patterns could grab the
desired pattern first claim on a substring. Each
the order it is defined in the lexer file; all patte
considered *after* every function-defined pattern
however, are not considered in the order they a
The purpose of this ordering is to let longer pat
"==" would be claimed before "=" , allowing th
correctly, rather than as sequential assignment

The special variable t_ignore may contain a stri
matching. These characters are skipped more e
return value. The token name ignore is, therefo

token (if the all-cap token name convention is f

The special function t_error() may be used to p
the passed-in LexToken will contain the remain
match). If you want to skip past a problem are
in the body of t_error()), use the .skip() metho

- Build the lexer. The *lex* module performs a bit
not need to name the built lexer. Most applicati
you wish toor if you need multiple lexers in the
name. For example:

```
mylexer = lex.lex()    # named lexer
lex.lex()              # default lexe
mylexer.input(mytext) # set input fo
lex.input(othertext)  # set input fo
```

- Give the lexer a string to process. This step is
conjunction with *lex*, and nothing need be done
input string using lex.input() (or similarly with

- Read the token stream (for stand-alone token
lex.token() function or the .token() method of a
PLY does not treat the token stream as a Pytho
iterator wrapper with:

```
from __future__ import generators
# ...define the lexer rules, etc...
```

```
def tokeniterator(lexer=lex):
    while 1:
        t = lexer.token()
        if t is None:
            raise StopIteration
        yield t
# Loop through the tokens
for t in tokeniterator():
    # ...do something with each toke
```

Without this wrapper, or generally in earlier ver
with a break condition:

```
# ...define the lexer rules, etc...
while 1:
    t = lex.token()
    if t is None:   # No more input
        break
    # ... do something with each tok
```

YACC

A *PLY* parsing module must do five things:

1. **Import the yacc module:**

```
import yacc
```

- Get a token map from a lexer. Suppose a lexe
requirements 1 through 4 in the above LEX des

```
from mylexer import *
```

Given the special naming convention t_* used f
pollution from import * is minimal.

You could also, of course, simply include the ne
itself.

- Define a collection of grammar rules. Gramma
functions. Specially named functions having a p
corresponding action code. Whenever a produc
function matches, the body of that function run

Productions in *PLY* are described with a simplifi
are available in rules; only sequencing and alte
with recursion and component productions).

The left side of each rule contains a single rule
spaces, a colon, and an additional one or more
following this. The right side of a rule can occu
allowed to fulfill a rule name, each such patterr
("|"). Within each right side line, a production i
termswhich may be either tokens generated by

production may be included in the same p_*()
each function to one production (you are free t

```
def p_rulename(t):
    '''rulename    : foo SPACE bar
                   | foo bar baz
                   | bar SPACE baz
       otherrule   : this that other
                   | this SPACE that
#...action code...
```

The argument to each p_*() function is a YaccS
the rule to an indexed position. The left side ru
term/token on the right side is listed thereafter
large enough to contain every term needed; th
production is fulfilled on a particular call.

Empty productions are allowed by *yacc* (matchi
empty production in a grammar, but this empty
higher-level productions. An empty production
of (potentiality) quantification in *PLY*; for exam

```
def p_empty(t):
    '''empty : '''
    pass
def p_maybefoo(t):
```

```
    '''foo : FOOTOKEN
           | empty '''
    t[0] = t[1]
def p_maybebar(t):
    '''bar : BARTOKEN
           | empty '''
    t[0] = t[1]
```

If a fulfilled production is used in other product
code should assign a meaningful value to index
production. Moreover what is returned by the a
production. For example:

```
# Sum N different numbers: "1.0 + 3
def p_sum(t):
    '''sum : number PLUS number'''
    #   ^      ^      ^     ^
    #  t[0]   t[1]  t[2]  t[3]
    t[0] = t[1] + t[3]
def p_number(t):
    '''number : BASICNUMBER
              | sum          '''
    #    ^        ^
    #   t[0]     t[1]
    t[0] = float(t[1])
```

```
# Create the parser and parse some s
yacc.yacc()
print yacc.parse('1.0')
```

The example simply assigns a numeric value wi
position 0 of the YaccSlice a list, Node object, o
higher-level productions.

- To build the parser the *yacc* module performs
do not need to name the built parser. Most app
However, if you wish toor if you need multiple p
built parser to a name. For example:

```
myparser = yacc.yacc()        # named
yacc.yacc()                   # defaul
r1 = myparser.parse(mytext) # set ir
r0 = yacc.parse(othertext)   # set ir
```

When parsers are built, *yacc* will produce diagn
the grammar.

- Parse an input string. The lexer is implicitly ca
rules. The return value of a parsing action can b
builds. It might be an abstract syntax tree, if a
might be a simple list as in the smart ASCII exa
concatenations and modifications during parsin
parsing was done wholly to trigger side effects

index position 0 of the root rule's LexToken.

MORE ON PLY PARSERS

Some of the finer points of *PLY* parsers will not
accompanying *PLY* contains some additional im
more systematically to parsing theory will addr
least be touched on.

Error Recovery

A *PLY* grammar may contain a special p_error(
matched (at the current position) by any other
enters an "error-recovery" mode. If the parser
successfully, a traceback is generated. You may
catch errors that occur at specific points in the

To implement recovery within the p_error() fun
yacc.token(), yacc.restart(), and yacc.errok().
this tokenor some sequence of tokensmeets so
yacc.restart() or yacc.errok(). The first of these
statebasically, only the final sub-string of the in
data structure you have built will remain as it w
in its last state and just ignore any bad tokens
p_error() itself, or via calls to yacc.token() in th

The Parser State Machine

When a parser is first compiled, the files parset
parsetab.py, contains more or less unreadable
subsequent parser invocations. These structure
applications; timestamps and signatures are co
changed. Pregenerating state tables speeds up

The file parser.out contains a fairly readable de
by *yacc*. Although you cannot manually modify
can help you in understanding error messages
grammars.

Precedence and Associativity

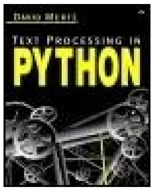
To resolve ambiguous grammars, you may set
precedence and the associativity of tokens. Abs
new symbol rather than reduce a rule where bo

The *PLY* documentation gives an example of an
4 + 5. After the tokens 3, *, and 4 have been
allow reduction of the product. But at the same
PLUS NUMBER, which would allow a lookahead
token). Moreover, the same token can have diff
the unary-minus and minus operators in 3 - 4

To solve both the precedence ambiguity and th
can declare an explicit precedence and associat

Declaring precedence and associativity

```
precedence = (
    ('left', 'PLUS', 'MINUS'),
    ('left', 'TIMES, 'DIVIDE'),
    ('right', 'UMINUS'),
)
def p_expr_uminus(t):
    'expr : MINUS expr % prec UMINUS
    t[0] = -1 * t[2]
def p_expr_minus(t):
    'expr : expr MINUS expr'
    t[0] = t[1] - t[3]
def p_expr_plus(t):
    'expr : expr PLUS expr'
    t[0] = t[1] + t[3]
```



Text Processing in PythonBy David Mertz

Table of Contents

Chapter 5. Internet Tools and Techniques

> Be strict in what you send, and lenient in what you accept.
>
> Internet Engineering Task Force

Internet protocols in large measure are descriptions of textual formats. At the lowest level, TCP/IP is a binary protocol, but virtually every layer run on top of TCP/IP consists of textual messages exchanged between servers and clients. Some basic messages govern control, handshaking, and authentication issues, but the information content of the Internet predominantly consists of texts formatted according to two or three general patterns.

The handshaking and control aspects of Internet protocols usually consist of short commandsand sometimes challengessent during an initial conversation between a client and server. Fortunately for Python programmers, the Python standard library contains intermediate-level modules to support all the most popular communication protocols: *poplib, smtplib, ftplib, httplib, telnetlib, gopherlib*, and *imaplib*. If you want to use any of these protocols, you can simply provide required setup information, then call module functions or classes to handle all the lower-level interaction. Unless you want to do something exoticsuch as programming a custom or less common network protocolthere is never a need to utilize the lower-level services of the *socket* module.

The communication level of Internet protocols is not primarily a text processing issue. Where text processing comes in is with parsing and production of compliant texts, to contain the *content* of these protocols. Each protocol is characterized by one or a few message types that are typically transmitted over the protocol. For example, POP3, NNTP, IMAP4, and SMTP protocols are centrally means of transmitting texts that conform to RFC-822, its updates, and associated RFCs. HTTP is firstly a means of transmitting Hypertext Markup Language (HTML) messages.

Following the popularity of the World Wide Web, however, a dizzying array of other message types also travel over HTTP: graphic and sounds formats, proprietary multimedia plug-ins, executable byte-codes (e.g., Java or Jython), and also more textual formats like XML-RPC and SOAP.

The most widespread text format on the Internet is almost certainly human-readable and human-composed notes that follow RFC-822 and friends. The basic form of such a text is a series of headers, each beginning a line and separated from a value by a colon; after a header comes a blank line; and after that a message body. In the simplest case, a message body is just free-form text; but MIME headers can be used to nest structured and diverse contents within a message body. Email and (Usenet) discussion groups follow this format. Even other protocols, like HTTP, share a top envelope structure with RFC-822.

A strong second as Internet text formats go is HTML. And in third place after that is XML, in various dialects. HTML, of course, is the lingua franca of the Web; XML is a more general standard for defining custom "applications" or "dialects," of which HTML is (almost) one. In either case, rather than a header composed of line-oriented fields followed by a body, HTML/XML contain hierarchically nested "tags" with each tag indicated by surrounding angle brackets. Tags like HTML's <body>, <cite>, and <blockquote> will be familiar already to most readers of this book. In any case, Python has a strong collection of tools in its standard library for parsing and producing HTML and XML text documents. In the case of XML, some of these tools assist with specific XML dialects, while lower-level underlying libraries treat XML sui generis. In some cases, third-party modules fill gaps in the standard library.

Various Python Internet modules are covered in varying depth in this chapter. Every tool that comes with the Python standard library is examined at least in summary. Those tools that I feel are of greatest importance to application programmers (in text processing applications) are documented in fair detail and accompanied by usage examples, warnings, and tips.

Chapter 5. Internet Tools and Techniqu

5.1 Working with Email and Newsgr

Python provides extensive support in its standa newsgroup) messages. There are three general supported by one or more Python modules.

1. **Communicating with network servers to messages. The modules *poplib, imaplib,* the protocol contained in its name. Thes text processing per se, but are often im email. The discussion of each of these n only those methods necessary to condu the first three modules/protocols. The r here under the assumption that email is processed than are Usenet articles. Ind almost always frowned upon, while aut (within limits).**

- Examining the contents of message folders. V

messages in a variety of formats, many providi
module *mailbox* provides a uniform API for rea
popular folder formats. In a way, *imaplib* serve
IMAP4 server can also structure folders, but fol
only cursorilythat topic also falls afield of text p
are definitely text formats, and *mailbox* makes

- The core text processing task in working with
the actual messages. RFC-822 describes a form
franca for Internet communication. Not every M
Agent (MTA) strictly conforms to the RFC-822 (
standardbut they all generally try to do so. The
rfc822, rfc1822, mimify, mimetools, MimeWrite
parsing and processing email messages.

Although existing applications are likely to use
and *multifile*, the package *email* contains more
implementations of the same capabilities. The f
synopsis while the various subpackages of *ema*

There is one aspect of working with email that
unnecessary. Unfortunately, in the real-world, a
viruses, and frauds; any application that works
demands a way to filter out the junk messages
the scope of this discussion, readers might ben
Techniques," at:

<http://gnosis.cx/publish/programming/filteri

A flexible Python project for statistical analysis
Bayesian and related models, is SpamBayes:

<http://spambayes.sourceforge.net/>

5.1.1 Manipulating and Creating Message Tex

email • Work with email messages

Without repeating the whole of RFC-2822, it is
email or newsgroup message. Messages may th
that impose larger-level structure, but here we
single message. An RFC-2822 message, like mo
format, often restricted to true 7-bit ASCII.

A message consists of a header and a body. A b
"payloads." In fact, MIME multipart/* type payl
payloads, but such nesting is comparatively un
payload in a body is divided by a simple, but fa
is pseudo-random, and you need to examine th
either contain text or binary data using base64
encoding (even 8-bit, which is not generally saf
either have MIME type text/* or compose the w
payload delimiter).

An RFC-2822 header consists of a series of field

beginning of a line and is followed by a colon a
the field name, starting on the same line, but p
continued field value cannot be left aligned, but
one space or tab. There are some moderately c
contents can split between lines, often depende
holds. Most field names occur only once in a he
their order of occurrence is not important to en
field namesnotably Receivedtypically occur mul
Complicating headers further, field values can c
ASCII character set.

The most important element of the *email* packa
whose instances provide a data structure and c
structure of RFC-2822 messages. Various capal
message, and for parsing a whole message into
contained in subpackages of the *email* package
wrapped in convenience functions in the top-lev

A version of the *email* package was introduced
However, *email* has been independently upgrad
releases. At the time this chapter was written,
and this discussion reflects that version (and th
most likely to remain consistent in later versior
use the version accompanying your Python inst
of the *email* package from <http://mimelib.sou
package. The current (and expected future) ver
compatible with Python versions back to 2.1. S
<http://gnosis.cx/TPiP/>, for instructions on us

incompatible with versions of Python before 2.0

CLASSES

Several children of *email.Message.Message* allo
with special properties and convenient initializa
technically contained in a module named in the
directly in the *email* namespace, but each is ve

email.MIMEBase.MIMEBase(maintype, subtyp

Construct a message object with a Content-Typ
is used only as a parent for further subclasses,

```
>>> mess = email.MIMEBase.MIMEBase('
>>> print mess
From nobody Tue Nov 12 03:32:33 2002
Content-Type: text/html; charset="us
MIME-Version: 1.0
```

email.MIMENonMultipart.MIMENonMultipart(m

Child of *email.MIMEBase.MIMEBase*, but raises

.attach(). Generally this class is used for furthe

email.MIMEMultipart.MIMEMultipart([subtype[,params]]]])**

Construct a multipart message object with subt boundary with the argument boundary, but spe to be calculated. If you wish to populate the me as additional arguments. Keyword arguments a Type header.

```
>>> from email.MIMEBase import MIMEB
>>> from email.MIMEMultipart import
>>> mess = MIMEBase('audio','midi')
>>> combo = MIMEMultipart('mixed', M
>>> print combo
From nobody Tue Nov 12 03:50:50 2002
Content-Type: multipart/mixed; chars
        boundary="===============595
MIME-Version: 1.0

--===============5954819931142521==
Content-Type: audio/midi
MIME-Version: 1.0
```

```
--===============5954819931142521==-
```

email.MIMEAudio.MIMEAudio(audiodata [,sub

Construct a single part message object that hol
specified as a string in the argument audiodata
sndhdr is used to detect the signature of the au
specify the argument subtype instead. An enco
with the encoder argument (but usually should
parameters to the Content-Type header.

```
>>> from email.MIMEAudio import MIME
>>> mess = MIMEAudio(open('melody.mi
```

SEE ALSO: sndhdr *397*;

email.MIMEImage.MIMEImage(imagedata [,su

Construct a single part message object that hol
specified as a string in the argument imagedata
imghdr is used to detect the signature of the im
specify the argument subtype instead. An enco
with the encoder argument (but usually should
parameters to the Content-Type header.

```
>>> from email.MIMEImage import MIME
>>> mess = MIMEImage(open('landscape
```

SEE ALSO: imghdr *396*;

email.MIMEText.MIMEText(text [,subtype [,cha

Construct a single part message object that hol
string in the argument text. A character set ma

```
>>> from email.MIMEText import MIMET
>>> mess = MIMEText(open('TPiP.tex')
```

FUNCTIONS

email.message_from_file(file [,_class=email.M

Return a message object based on the message
This function call is exactly equivalent to:

```
email.Parser.Parser(_class, strict).
```

SEE ALSO: email.Parser.Parser.parse() *363*;

email.message_from_string(s [,_class=email.

Return a message object based on the messag
function call is exactly equivalent to:

```
email.Parser.Parser(_class, strict).
```

SEE ALSO: email.Parser.Parser.parsestr() *363*;

email.Encoders • Encoding message paylo

The module *email.Encoder* contains several fun
part message objects. Each of these functions s
to an appropriate value after encoding the body
.get_payload() message method can be used to

FUNCTIONS

email.Encoders.encode_quopri(mess)

Encode the message body of message object m
sets the header Content-Transfer-Encoding.

email.Encoders.encode_base64(mess)

Encode the message body of message object m
header Content-Transfer-Encoding.

email.Encoders.encode_7or8bit(mess)

Set the Content-Transfer-Encoding to 7bit or 8b
not modify the payload itself. If message mess
header, calling this will create a second oneit is
calling this function.

SEE ALSO: email.Message.Message.get_payloa

email.Errors • Exceptions for [email] pack

Exceptions within the *email* package will raise s
desired level of generality. The exception hierar

Figure 5.1. Standard ema

exceptions.Exception	Root class for all built-in exceptions
MessageError	Base for email exceptions
MessageParseError	Base for message parsing exceptions
BoundaryError	Could not find boundary
HeaderParseError	Problem parsing the header
MultipartConversionError	Also child of `exceptions.TypeError`

SEE ALSO: exceptions *44*;

email.Generator • Create text representat

The module *email.Generator* provides support f
email.Message.Message objects. In principle, y
message objects to specialized formatsfor exan
email.Message.Message object to store values t
practice, you almost always want to write mess
2822 message texts. Several of the methods of
utilize *email.Generator*.

CLASSES

email.Generator.Generator(file [,mangle_from

Construct a generator instance that writes to th
mangle_from_ is specified as a true value, any
begins with the string From followed by a space
reversible) transformation prevents BSD mailbo

argument maxheaderlen specifies where long h
such is possible).

email.Generator.DecodedGenerator(file [,man

Construct a generator instance that writes RFC-
initializers as its parent *email.Generator.Genera*
argument fmt.

The class *email.Generator.DecodedGenerator* o
of a multipart message payload. Nontext parts
may contain keyword replacement values. For e

```
[Non-text (%(type)s) part of message
```

Any of the keywords type, maintype, subtype, f
used as keyword replacements in the string fmt
the payload, a simple description of its unavaila

METHODS

email.Generator.Generator.clone()
email.Generator.DecodedGenerator.clone()

Return a copy of the instance with the same op

email.Generator.Generator.flatten(mess [,unix
email.Generator.DecodedGenerator.flatten(m

Write an RFC-2822 serialization of message obj
instance was initialized with. If the argument u
BSD mailbox From_ header is included in the s

email.Generator.Generator.write(s)
email.Generator.DecodedGenerator.write(s)

Write the string s to the file-like object the inst
generator object itself act in a file-like manner,

SEE ALSO: email.Message *355*; mailbox *372*;

email.Header • Manage headers with non-

The module *email.Charset* provides fine-tuned
conversions and maintaining a character set re
provided by *email.Header* provides all the capa
friendlier form.

The basic reason why you might want to use th
want to encode multinational (or at least non-U
bodies are somewhat more lenient than header
restricted to using only 7-bit ASCII to encode o
email.Header provides a single class and two co
non-ASCII characters in email headers is descri
2045, RFC-2046, RFC-2047, and most directly

CLASSES

email.Header.Header([s="" [,charset [,maxline [,continuation_ws=" "]]]]])

Construct an object that holds the string or Uni
charset to use in encoding s; absent any argum
as needed.

Since the encoded string is intended to be used
to wrap the string to multiple lines (depending
specifies where the wrapping will occur; header
anticipate using the encoded string withit is sig
specified header_name, no width is set aside fo
continuation_ws specified what whitespace strin
lines; it must be a combination of spaces and ta

Instances of the class *email.Header.Header* imp

therefore respond to the built-in *str()* function a
built-in techniques are more natural, but the m
performs an identical action. As an example, le

```
>>> from unicodedata import lookup
>>> lquot = lookup("LEFT-POINTING DO
>>> rquot = lookup("RIGHT-POINTING I
>>> s = lquot + "Euro-style" + rquot
>>> s
u'\xabEuro-style\xbb quotation'
>>> print s.encode('iso-8859-1')
Euro-style quotation
```

Using the string s, let us encode it for an RFC-2

```
>>> from email.Header import Header
>>> print Header(s)
=?utf-8?q?=C2=ABEuro-style=C2=BB_quo
>>> print Header(s,'iso-8859-1')
=?iso-8859-1?q?=ABEuro-style=BB_quot
>>> print Header(s, 'utf-16')
=?utf-16?b?/v8AqwBFAHUAcgBvACOAcwBOA
 =?utf-16?b?/v8AuwAgAHEAdQBvAHQAYQBC
>>> print Header(s,'us-ascii')
=?utf-8?q?=C2=ABEuro-style=C2=BB_quo
```

Notice that in the last case, the *email.Header.H*
my request for an ASCII character set, since it
However, the class is happy to skip the encodin

```
>>> print Header('"US-style" quotati
"US-style" quotation
>>> print Header('"US-style" quotati
=?utf-8?q?=22US-style=22_quotation?=
>>> print Header('"US-style" quotati
"US-style" quotation
```

METHODS

email.Header.Header.append(s [,charset])

Add the string or Unicode string s to the end of
character set charset. Note that the charset of
that of the existing content.

```
>>> subj = Header(s,'latin-1',65)
>>> print subj
=?iso-8859-1?q?=ABEuro-style=BB_quot
>>> unicodedata.name(omega), unicode
('GREEK SMALL LETTER OMEGA', 'GREEK
```

```
>>> subj.append(', Greek: ', 'us-asc
>>> subj.append(Omega, 'utf-8')
>>> subj.append(omega, 'utf-16')
>>> print subj
=?iso-8859-1?q?=ABEuro-style=BB_quot
 =?utf-8?b?zqk=?= =?utf-16?b?/v8DyQ=
 >>> unicode(subj)
 u'\xabEuro-style\xbb quotation, Gre
```

email.Header.Header.encode()
email.Header.Header.__str__()

Return an ASCII string representation of the in

FUNCTIONS

email.Header.decode_header(header)

Return a list of pairs describing the component
header object header. Each pair in the list conta
encoding name.

```
>>> email.Header.decode_header(Heade
```

```
[('spam and eggs', None)]
>>> print subj
=?iso-8859-1?q?=ABEuro-style=BB_quot
 =?utf-8?b?zqk=?= =?utf-16?b?/v8DyQ=
>>> for tup in email.Header.decode_h
...
('\xabEuro-style\xbb quotation', 'is
(', Greek:', None)
('\xce\xa9', 'utf-8')
('\xfe\xff\x03\xc9', 'utf-16')
```

These pairs may be used to construct Unicode s
function. However, plain ASCII strings show an
to the *unicode()* function.

```
>>> for s,enc in email.Header.decode
...     enc = enc or 'us-ascii'
...     print `unicode(s, enc)'
...
u'\xabEuro-style\xbb quotation'
u', Greek:'
u'\u03a9'
u'\u03c9'
```

SEE ALSO: unicode() *423*; email.Header.make_

email.Header.make_header(decoded_seq [,m [,continuation_ws]]])

Construct a header object from a list of pairs of *email.Header.decode-header()*. You may also, c decoded_seq manually, or by other means. The header_name, and continuation_ws are the sar

```
>>> email.Header.make.header([('\xce
...                               ('-mar
'=?utf-8?b?zqk=?=-man'
```

SEE ALSO: email.Header.decode_header() *353;*

email.Iterators • Iterate through compon

The module *email.Iterators* provides several co messages in ways different from *email.Message* *email.Message.Message.walk()*.

FUNCTIONS

email.Iterators.body_line_iterator(mess)

Return a generator object that iterates through
mess. The entire body that would be produced
content types and nesting of parts. But any MII
returned lines.

```
>>> import email.MIMEText, email.Ite
>>> mess1 = email.MIMEText.MIMEText(
>>> mess2 = email.MIMEText.MIMEText(
>>> combo = email.Message.Message()
>>> combo.set_type('multipart/mixed'
>>> combo.attach(mess1)
>>> combo.attach(mess2)
>>> for line in email.Iterators.body
...     print line
...
message one
message two
```

email.Iterators.typed_subpart_iterator(mess [

Return a generator object that iterates through
matches maintype. If a subtype subtype is spec
maintype/subtype.

email.Iterators._structure(mess [,file=sys.std

Write a "pretty-printed" representation of the s
Output to the file-like object file.

```
>>> email.Iterators._structure(combo
multipart/mixed
    multipart/digest
        image/png
        text/plain
    audio/mp3
    text/html
```

SEE ALSO: email.Message.Message.get_payloa
362;

email.Message • Class representing an em

A message object that utilizes the *email.Messag*
syntactic conveniences and support methods fo
The class *email.Message.Message* is a very goo
built-in *str()* functionand therefore also the prin
produce its RFC-2822 serialization.

In many ways, a message object is dictionary-l

implemented in it to support keyed indexing an
containment testing with the in keyword, and k
expects to find in a Python dict are all impleme
email.Message.Message:has_key(), .keys(), .va
examples are helpful:

```
>>> import mailbox, email, email.Par
>>> mbox = mailbox.PortableUnixMailb
...                           email.Par
>>> mess = mbox.next()
>>> len(mess)                    # numb
16
>>> 'X-Status' in mess           # memb
1
>>> mess.has_key('X-AGENT')      # also
0
>>> mess['x-agent'] = "Python Mail A
>>> print mess['X-AGENT']        # acce
Python Mail Agent
>>> del mess['X-Agent']          # dele
>>> print mess['X-AGENT']
None
>>> [fld for (fld,val) in mess.items
['Received', 'Received', 'Received',
```

This is dictionary-like behavior, but only to an e
email header rules. Moreover, a given key may
key will return only the first such value, but me
will return a list of all the entries. In some othe
is more like a list of tuples, chiefly in guarantee
fields.

A few more details of keyed indexing should be
add an *additional* header, rather than replace a
operation is more like a *list.append()* method. I
every matching header. If you want to replace a
assign.

The special syntax defined by the *email.Messag*
headers. But a message object will typically als
If the Content-Type header contains the value m
zero or more payloads, each one itself a messa
(including where none is explicitly specified), th
an encoded one. The message instance method
either a list of message objects or a string. Use
which return type is expected.

As the epigram to this chapter suggests, you sh
messages you construct yourself. But in real-w
messages with badly mismatched headers and
to be multipart, and vice versa. Moreover, the M
loose indication of what payloads actually conta
spammers and virus writers trying to exploit th

security of Microsoft applicationsa malicious pa
Windows will typically launch apps based on file
problems arise not out of malice, but simply ou
Depending on the source of your processed me
about the allowable structure and headers of m

SEE ALSO: UserDict *24*; UserList *28*;

CLASSES

email.Message.Message()

Construct a message object. The class accepts

METHODS AND ATTRIBUTES

email.Message.Message.add_header(field, va

Add a header to the message headers. The hea
effect is the same as keyed assignment to the o
parameters using Python keyword arguments.

```
>>> import email.Message
>>> msg = email.Message.Message()
```

```
>>> msg['Subject'] = "Report attachm
>>> msg.add_header('Content-Disposit
...                  filename='report
>>> print msg
From nobody Mon Nov 11 15:11:43 2002
Subject: Report attachment
Content-Disposition: attachment; fil
```

email.Message.Message.as_string([unixfrom=

Serialize the message to an RFC-2822-complian specified with a true value, include the BSD ma Serialization with *str()* or *print* includes the "Fr

email.Message.Message.attach(mess)

Add a payload to a message. The argument me *email.Message.Message* object. After this call, t message objects (perhaps of length one, if this calling this method causes the method .is_mult need to separately set a correct multipart/* co object.

```
>>> mess = email.Message.Message()
```

```
>>> mess.is_multipart()
0
>>> mess.attach(email.Message.Messag
>>> mess. is_multipart ()
1
>>> mess.get_payload()
[<email.Message.Message instance at
>>> mess.get_content_type()
'text/plain'
>>> mess.set_type('multipart/mixed')
>>> mess.get_content_type()
'multipart/mixed'
```

If you wish to create a single part payload for a
email.Message.Message.set-payload().

SEE ALSO: email.Message.Message.set_payloa

email.Message.Message.del_param(param [,h

Remove the parameter param from a header. If
is taken, but also no exception is raised. Usuall
header, but you may specify a different header
argument requote controls whether the parame
no harm).

```
>>> mess = email.Message.Message()
>>> mess.set_type('text/plain')
>>> mess.set_param('charset','us-asc
>>> print mess
From nobody Mon Nov 11 16:12:38 2002
MIME-Version: 1.0
Content-Type: text/plain; charset="u

>>> mess.del_param('charset')
>>> print mess
From nobody Mon Nov 11 16:13:11 2002
MIME-Version: 1.0
content-type: text/plain
```

email.Message.Message.epilogue

Message bodies that contain MIME content deli
the area between the first and final delimiter. A
stored in *email.Message.Message.epilogue*.

SEE ALSO: email.Message.Message.preamble 3

email.Message.Message.get_all(field [,failobj=

Return a list of all the headers with the field na
value specified in argument failobj. In most cas
at all), but a few fields such as Received typica

The default nonmatch return value of None is p
Returning an empty list will let you use this me

```
>>> for rcv in mess.get_all('Receive
...     print rcv
...
About that time
A little earlier
>>> if mess.get_all('Foo',[]):
...     print "Has Foo header(s)"
```

email.Message.Message.get_boundary([failol

Return the MIME message boundary delimiter f
boundary is defined; this *should* always be the

email.Message.Message.get_charsets([failob

Return a list of string descriptions of contained

email.Message.Message.get_content_charset

Return a string description of the message char

email.Message.Message.get_content_maintyp

For message mess, equivalent to mess.get_con

email.Message.Message.get_content_subtyp

For message mess, equivalent to mess.get_con

email.Message.Message.get_content_type()

Return the MIME content type of the message c
lowercase and contains both the type and subty

```
>>> msg_photo.get_content_type()
'image/png'
>>> msg_combo.get_content_type()
'multipart/mixed'
>>> msg_simple.get_content_type()
```

```
'text/plain'
```

email.Message.Message.get_default_type()

Return the current default type of the message
payloads that are not accompanied by an explic

email.Message.Message.get_filename([failobj

Return the filename parameter of the Content-I
exists (perhaps because no such header exists)

email.Message.Message.get_param(param [,f

Return the parameter param of the header hea
header. If the parameter does not exist, return
specified as a true value, the quote marks are r

```
>>> print mess.get_param('charset',u
us-ascii
>>> print mess.get_param('charset',u
"us-ascii"
```

SEE ALSO: email.Message.Message.set_param(

email.Message.Message.get_params([,failobj

Return all the parameters of the header header
header. If the header does not exist, return fail
list of key/val pairs. The argument unquote ren

```
>>> print mess.get_params(header="To
[('<mertz@gnosis.cx>', '')]
>>> print mess.get_params(unquote=0)
[('text/plain', ''), ('charset', '"u
```

email.Message.Message.get_payload([i [,deco

Return the message payload. If the message m
method returns a list of component message ob
string with the message body. Note that if the m
email.Parser.HeaderParser, then the body is tre
MIME delimiters.

Assuming that the message is multipart, you m
the indexed component. Specifying the i argum
returned list without specifying i. If decode is sp
single part, the returned payload is decoded (i.

I find that dealing with a payload that may be e
awkward. Frequently, you would like to simply
whether or not MIME multiparts are contained i
uniformity:

write_payload_list.py

```
#!/usr/bin/env python
"Write payload list to separate file
import email, sys
def get_payload_list(msg, decode=1):
    payload = msg.get_payload(decode
    if type(payload) in [type(""), t
        return [payload]
    else:
        return payload
mess = email.message_from_file(sys.s
for part,num in zip(get_payload_list
    file = open('%s.%d' % (sys.argv[
    print >> file, part
```

SEE ALSO: email.Parser *363*; email.Message.M
email.Message.Message.walk() *362*;

email.Message.Message.get_unixfrom()

Return the BSD mailbox "From_" envelope hea

SEE ALSO: mailbox *372*;

email.Message.Message.is_multipart()

Return a true value if the message is multipart.
multipart is having multiple message objects in
not guaranteed to be multipart/* when this me
it *should* be).

SEE ALSO: email.Message.Message.get_payloa

email.Message.Message.preamble

Message bodies that contain MIME content deli
the area between the first and final delimiter. A
is stored in *email.Message.Message.preamble*.

SEE ALSO: email.Message.Message.epilogue *3*

email.Message.Message.replace_header(field

Replaces the first occurrence of the header with
matching header is found, raise KeyError.

email.Message.Message.set_boundary(s)

Set the boundary parameter of the Content-Typ
have a Content-Type header, raise HeaderParse
create a boundary manually, since the *email* m
it own for multipart messages.

email.Message.Message.set_default_type(cty

Set the current default type of the message to
decoding payloads that are not accompanied by

email.Message.Message.set_param(param, va [,requote=1 [,charset [,language]]]])

Set the parameter param of the header header
requote is specified as a true value, the parame
language may be used to encode the paramete

email.Message.Message.set_payload(payload

Set the message payload to a string or to a list
overwrites any existing payload the message h
you must use this method to configure the mes
subclass to construct the message in the first p

SEE ALSO: email.Message.Message.attach() *35*
email.MIMEImage.MIMEImage *348*; email.MIMI

email.Message.Message.set_type(ctype [,hea

Set the content type of the message to ctype, l
is. If the argument requote is specified as a tru
also specify an alternative header to write the c
cannot think of any reason you would want to.

email.Message.Message.set_unixfrom(s)

Set the BSD mailbox envelope header. The argu
a space, usually followed by a name and a date

SEE ALSO: mailbox *372*;

email.Message.Message.walk()

Recursively traverse all message parts and sub
iterator will yield each nested message object i

```
>>> for part in mess.walk():
...     print part.get_content_type()
multipart/mixed
text/html
audio/midi
```

SEE ALSO: email.Message.Message.get_payloa

email.Parser • Parse a text message into a

There are two parsers provided by the *email.Pa*
child *email.Parser.HeaderParser*. For general us
latter allows you to treat the body of an RFC-28
Skipping the parsing of message bodies can be
improperly formatted message bodies (somethi
spam messages that lack any content value as

The parsing methods of both classes accept an
Specifying headersonly has a stronger effect th
class. If headersonly is specified in the parsing
body is skipped altogetherthe message object c
the other hand, if *email.Parser.HeaderParser* is
is specified as false (the default), the body is al

content type is multipart/*.

CLASSES

email.Parser.Parser([_class=email.Message.M

Construct a parser instance that uses the class
There is normally no reason to specify a differe
parsing with the strict option will cause excepti
conform fully to the RFC-2822 specification. In
useful.

email.Parser.HeaderParser([_class=email.Mes

Construct a parser instance that is the same as
that multipart messages are parsed as if they w

METHODS

email.Parser.Parser.parse(file [,headersonly=
email.Parser.HeaderParser.parse(file [,header

Return a message object based on the messag
the optional argument headersonly is given a tı
discarded.

email.Parser.Parser.parsestr(s [,headersonly=
email.Parser.HeaderParser.parsestr(s [,heade

Return a message object based on the messag
argument headersonly is given a true value, th

email.Utils • Helper functions for working

The module *email.Utils* contains a variety of co
with special header fields.

FUNCTIONS

email.Utils.decode_rfc2231(s)

Return a decoded string for RFC-2231 encoded

```
>>> Omega = unicodedata.lookup("GREE
```

```
>>> print email.Utils.encode_rfc2231
%3A9-man%40gnosis.cx
>>> email.Utils.decode_rfc2231("utf-
('utf-8', '', ':9-man@gnosis.cx')
```

email.Utils.encode_rfc2231(s [,charset [,langu

Return an RFC-2231-encoded string from the s
optionally be specified.

email.Utils.formataddr(pair)

Return a formatted address from pair (realnam

```
>>> email.Utils.formataddr(('David M
'David Mertz <mertz@gnosis.cx>'
```

email.Utils.formataddr([timeval [,localtime=0]

Return an RFC-2822-formatted date based on a
time.localtime(). If the argument localtime is sp
timezone rather than UTC. With no options, use

```
>>> email.Utils.formatdate()
'Wed, 13 Nov 2002 07:08:01 -0000'
```

email.Utils.getaddresses(addresses)

Return a list of pairs (realname,addr) based on
argument addresses.

```
>>> addrs = ['"Joe" <jdoe@nowhere.la
>>> email.Utils.getaddresses(addrs)
[('Joe', 'jdoe@nowhere.lan'), ('Jane
```

email.Utils.make_msgid([seed])

Return a unique string suitable for a Message-I
incorporate that string into the returned value;
name or other identifying information.

```
>>> email.Utils.make_msgid('gnosis')
'<20021113071050.3861.13687.gnosis@l
```

email.Utils.mktime_tz(tuple)

Return a timestamp based on an *email.Utils.pa*

```
>>> email.Utils.mktime_tz((2001, 1,
979224542.0
```

email.Utils.parseaddr(address)

Parse a compound address into the pair (realna

```
>>> email.Utils.parseaddr('David Mer
('David Mertz', 'mertz@gnosis.cx')
```

email.Utils.parsedate(datestr)

Return a date tuple based on an RFC-2822 date

```
>>> email.Utils.parsedate('11 Jan 20
(2001, 1, 11, 14, 49, 2, 0, 0, 0)
```

SEE ALSO: time *86*;

email.Utils.parsedate_tz(datestr)

Return a date tuple based on an RFC-2822 date
but adds a tenth tuple field for offset from UTC

email.Utils.quote(s)

Return a string with backslashes and double qu

```
>>> print email.Utils.quote(r'"MyPat
\"MYPath\" is d:\\this\\that
```

email.Utils.unquote(s)

Return a string with surrounding double quotes

```
>>> print email.Utils.unquote('<mert
mertz@gnosis.cx
>>> print email.Utils.unquote('"us-a
us-ascii
```

5.1.2 Communicating with Mail Servers

imaplib • IMAP4 client

The module *imaplib* supports implementing cus
in RFC-1730 and RFC-2060. As with the discuss
documentation aims only to cover the basics of
methods and functions are omitted here. In pa
able to retrieve messagescreating new mailbox
this book.

The *Python Library Reference* describes the PO
recommends the use of IMAP4 if your server su
technicallyIMAP indeed has some advantagesin
more widespread among both clients and serve
your specific requirements will dictate the choi

Aside from using a more efficient transmission
sends whole messages), IMAP4 maintains mult
automates filtering messages by criteria. A typi
might look like the one below. To illustrate a fe
the promising subject lines, after deleting any t
itself retrieve regular messages, only their hea

check_imap_subjects.py

```
#!/usr/bin/env python
import imaplib, sys
if len(sys.argv) == 4:
    sys.argv.append('INBOX')
```

```
(host, user, passwd, mbox) = sys.arg
i = imaplib.IMAP4(host, port=143)
i.login(user, passwd)
resp = i.select(mbox)
if r[0] <> 'OK':
    sys.stderr.write("Could not sele
    sys.exit()
# delete some spam messages
typ, spamlist = i.search(None, '(SUB
i.store(','.join(spamlist.split()),'
i.expunge()
typ, messnums = i.search(None,'ALL')
for mess in messnums:
    typ, header = i.fetch(mess, 'RFC
    for line in header[0].split('\n'
        if string.upper(line[:9]) ==
            print line[9:]
i.close()
i.logout()
```

There is a bit more work to this than in the POP additional capabilities. Unfortunately, much of t passing strings with flags and commands, none *Python Library Reference* or in the source to th protocol is probably necessary for complex clie

CLASSES

imaplib.IMAP4([host="localhost" [port=143]])

Create an IMAP instance object to manage a ho

METHODS

imaplib.IMAP4.close()

Close the currently selected mailbox, and delet
method *imaplib.IMAP4.logout()* is used to actua

imaplib.IMAP4.expunge()

Permanently delete any messages marked for d

imaplib.IMAP4.fetch(message_set, message_

Return a pair (typ,datalist). The first field typ is
The second field datalist is a list of returned str
message_set is a comma-separated list of mes

message_parts describe the components of the
and so on.

imaplib.IMAP4.list([dirname="" [,pattern="*"]]

Return a (typ,datalist) tuple of all the mailboxe
glob-style pattern pattern. datalist contains a li
this method with *imaplib.IMAP4.search()*, whicl
from the currently selected mailbox.

imaplib.IMAP4.login(user, passwd)

Connect to the IMAP server specified in the inst
authentication information given by user and p

imaplib.IMAP4.logout()

Disconnect from the IMAP server specified in th

imaplib.IMAP4.search(charset, criterion1 [,cri

Return a (typ,messnums) tuple where messnur
numbers of matching messages. Message criter

either be ALL for all messages or flags indicatin

imaplib.IMAP4.select([mbox="INBOX" [,readc

Select the current mailbox for operations such
imaplib.IMAP4.expunge(). The argument mbox
readonly allows you to prevent modification to

SEE ALSO: email *345*; poplib *368*; smtplib *370*

poplib • A POP3 client class

The module *poplib* supports implementing cust
in RFC-1725. As with the discussion of other pr
only to cover the basics of communicating with
may be omitted here.

The *Python Library Reference* describes the PO
recommends the use of IMAP4 if your server su
technicallyIMAP indeed has some advantagesin
more widespread among both clients and serve
your specific requirements will dictate the choic

A typical (simple) POP3 client application might
few methods, this application will print all the p

delete any that look like spam. The example dc
only their headers.

new_email_subjects.py

```
#!/usr/bin/env python
import poplib, sys, string
spamlist = []
(host, user, passwd) = sys.argv[1:]
mbox = poplib.POP3(host)
mbox.user(user)
mbox.pass_(passwd)

for i in range(1, mbox.stat()[0]+1):
    # messages use one-based indexir
    headerlines = mbox.top(i, 0)[1]
    for line in headerlines:
        if string.upper(line[:9]) ==
            if -1 <> string.find(lir
                spam = string.join(m
                spamlist.append(spam
                mbox.dele(i)
            else:
                print line[9:]
```

```
mbox.quit()
for spam in spamlist:
    report_to_spamcop(spam)      # as
```

CLASSES

poplib.POP3(host [,port=110])

The *poplib* module provides a single class that c
at host host, using port port.

METHODS

poplib.POP3.apop(user, secret)

Log in to a server using APOP authentication.

poplib.POP3.dele(messnum)

Mark a message for deletion. Normally the actu
with *poplib.POP3.quit()*, but server implementa

poplib.POP3.pass_(password)

Set the password to use when communicating v

poplib.POP3.quit()

Log off from the connection to the POP server. I
deletions to be carried out. Call this method as
connection to the POP server; while you are co
receiving any incoming messages.

poplib.POP3.retr(messnum)

Return the message numbered messnum (usin
of the form (resp,linelist,octets), where linelist
message. To re-create the whole message, you

poplib.POP3.rset()

Unmark any messages marked for deletion. Sir
good practice to mark messages using *poplib.P*
you want to erase them. However, *poplib.POP3.*
unusual circumstances occur before the connec

poplib.POP3.top(messnum, lines)

Retrieve the initial lines of message messnum.
lines lines from the body. The return format is t
you will typically be interested in offset 1 of the

poplib.POP3.stat()

Retrieve the status of the POP mailbox in the fo
gives you the total number of message pending
messages.

poplib.POP3.user(username)

Set the username to use when communicating

SEE ALSO: email *345*; smtplib *370*; imaplib *36*

smtplib • SMTP/ESMTP client class

The module *smtplib* supports implementing cus
in RFC-821 and RFC-1869. As with the discussi
documentation aims only to cover the basics of

methods and functions are omitted here. The m
retrieve incoming email, and the module *smtpl*

A typical (simple) SMTP client application migh
a command-line tool that accepts as a paramet
header, constructs the From using environment
STDIN. The To and From are also added as RFC

send_email.py

```
#!/usr/bin/env python
import smtplib
from sys import argv, stdin
from os import getenv
host = getenv('HOST', 'localhost')
if len(argv) >= 2:
    to_ = argv[1]
else:
    to_ = raw_input('To: ').strip()
if len(argv) >=3:
    subject = argv[2]
    body = stdin.read()
else:
    subject = stdin.readline()
    body = subject + stdin.read()
```

```
from_ = "%s@%s" % (getenv('USER', 'u
mess = '''From: %s\nTo: %s\n\n%s' %
server = smtp.SMTP(host)
server.login
server.sendmail(from_, to_, mess)
server.quit()
```

CLASSES

smtplib.SMTP([host="localhost" [,port=25]])

Create an instance object that establishes a co
using port port.

METHODS

smtplib.SMTP.login(user, passwd)

Login to an SMTP server that requires authenti
fails.

Not allor even mostSMTP servers use password
direct authentication, but since not all clients s

often disabled. One commonly used strategy to
malicious/spam messages to be sent through tl
arrangement, an IP address is authorized to us
that same address has successfully authenticat
machine. The timeout period is typically a few r

smtplib.SMTP.quit()

Terminate an SMTP connection.

smtplib.SMTP.sendmail(from_, to_, mess [,ma

Send the message mess with From envelope fr
may either be a string containing a single addre
message should include any desired RFC-822 h
arguments mail_options and rcpt_options.

SEE ALSO: email *345*; poplib *368*; imaplib *366*,

5.1.3 Message Collections and Message Parts

mailbox • Work with mailboxes in various

The module *mailbox* provides a uniform interfa
popular formats. Each class in the *mailbox* mod
appropriate format, and returns an instance wit
method returns each consecutive message with
Moreover, the .next () method is conformant wi
which lets you loop over messages in recent ve

By default, the messages returned by mailbox i
rfc822.Mailbox. These message objects provide
attributes. However, the recommendation of thi
in place of the older *rfc822*. Fortunately, you m
optional message constructor. The only constra
callable object that accepts a file-like object as
two logical choices here.

```
>>> import mailbox, email, email.Par
>>> mbox = mailbox.PortableUnixMailb
>>> mbox.next()
<rfc822.Message instance at 0x41d77(
>>> mbox = mailbox.PortableUnixMailb
...                           email.mes
>>> mbox.next()
<email.Message.Message instance at (
>>> mbox = mailbox.PortableUnixMailb
...                           email.Par
>>> mbox.next()
```

```
<email.Message.Message instance at C
```

In Python 2.2+ you might structure your applic

Looping through a mailbox in 2.2+

```
#!/usr/bin/env python
from mailbox import PortableUnixMail
from email import message_from_file
import sys
folder = open(sys.argv[1])
for message in PortableUnixMailbox(f
    # do something with the message.
    print message['Subject']
```

However, in earlier versions, this same code wil
.__getitem__() magic method. The slightly less
application in an older Python is:

Looping through a mailbox in any version

```
#!/usr/bin/env python
"Subject printer, older Python and r
import sys
```

```
from mailbox import PortableUnixMail
mbox = PortableUnixMailbox(open(sys.
while 1:
    message = mbox.next()
    if message is None:
        break
    print message.getheader('Subject
```

CLASSES

mailbox.UnixMailbox(file [,factory=rfc822.Mes

Read a BSD-style mailbox from the file-like obj
specified, it must be a callable object that acce
(in this case, that object is a portion of an unde

A BSD-style mailbox divides messages with a b
In this strict case, the "From_" line must have
matches a regular expression. In most cases, y
mailbox.PortableUnixMailbox, which relaxes the
message in a file.

mailbox.PortableUnixMailbox(file [,factory=rf

The arguments to this class are the same as fo
messages within the mailbox file depends only
beginning of a line. In practice, this is as much
guarantee that all mailboxes of interest will be
version.

mailbox.BabylMailbox(file [,factory=rfc822.Me

The arguments to this class are the same as fo
files in Babyl format.

mailbox.MmdfMailbox(file [,factory=rfc822.Me

The arguments to this class are the same as fo
files in MMDF format.

mailbox.MHMailbox(dirname [,factory=rfc822

The MH format uses the directory structure of t
organize mail folders. Each message is held in a
mailbox.MHMailbox is a string giving the name
factory argument is the same as with *mailbox.U*

mailbox.Maildir(dirname [,factory=rfc822.Mes

The QMail format, like the MH format, uses the
native filesystem to organize mail folders. The i
string giving the name of the directory to be pr
same as with *mailbox.UnixMailbox*.

SEE ALSO: email *345*; poplib *368*; imaplib *366*,

mimetypes • Guess the MIME type of a file

The *mimetypes* module maps file extensions to
is a dictionary, but several convenience functio
files containing additional mappings, and also c
ways. As well as actual MIME types, the *mimet*
for example, compression wrapper.

In Python 2.2+, the *mimetypes* module also pr
lets instances each maintain their own MIME ty
multiple distinct mapping is rare enough not to

FUNCTIONS

mimetypes.guess_type(url [,strict=0])

Return a pair (typ, encoding) based on the file
by url. If the strict option is specified with a tru
considered. Otherwise, a larger number of wide
either type or encoding cannot be guessed, Nor

```
>>> import mimetypes
>>> mimetypes.guess_type('x.abc.gz')
(None, 'gzip')
>>> mimetypes.guess_type('x.tgz')
('application/x-tar', 'gzip')
>>> mimetypes.guess_type('x.ps.gz')
('application/postscript', 'gzip')
>>> mimetypes.guess_type('x.txt')
('text/plain', None)
>>> mimetypes.guess_type('a.xyz')
(None, None)
```

mimetypes.guess_extension(type [,strict=0])

Return a string indicating a likely extension ass
extensions are possible, one is returned (gener
this is not guaranteed). The argument strict ha
mimetypes.guess-type().

```
>>> print mimetypes.guess_extension(
```

```
None
>>> print mimetypes.guess_extension(
.pdf
>>> print mimetypes.guess_extension(
.ai
```

mimetypes.init([list-of-files])

Add the definitions from each filename listed in
Several default files are examined even if this f
configuration files may be added as needed on
system, which uses somewhat different directo
run:

```
>>> mimetypes.init(['/private/etc/ht
...                  '/private/etc/ht
```

Notice that even if you are specifying only one
enclose its name inside a list.

mimetypes.read_mime_types(fname)

Read the single file named fname and return a
types.

```
>>> from mimetypes import read_mime_
>>> types = read_mime_types('/privat
>>> for _ in range(5): print types.p
...
('.wbxml', 'application/vnd.wap.wbxm
('.aiff', 'audio/x-aiff')
('.rm', 'audio/x-pn-realaudio')
('.xbm', 'image/x-xbitmap')
('.avi', 'video/x-msvideo')
```

ATTRIBUTES

mimetypes.common_types

Dictionary of widely used, but unofficial MIME t

mimetypes.inited

True value if the module has been initialized.

mimetypes.encodings_map

Dictionary of encodings.

mimetypes.knownfiles

List of files checked by default.

mimetypes.suffix_map

Dictionary of encoding suffixes.

mimetypes.types_map

Dictionary mapping extensions to MIME types.

5.2 World Wide Web Applications

5.2.1 Common Gateway Interface

cgi • Support for Common Gateway Interf

The module *cgi* provides a number of helpful to
elements to CGI, basically: (1) Reading query v
requesting browser. The first of these elements
just a matter of formatting suitable text to retu
is its primary interface; it also contains several
here because their use is uncommon (and not h
specific needs). See the *Python Library Referen*

A CGI PRIMER

A primer on the Common Gateway Interface is
applicationin any programming languagethat ru
recognizes a request for a CGI application, sets
control to the CGI application. By default, this i
for the CGI application to run in, but technologi
some tricks to avoid extra process creation. Th
but change little from the point of view of the C

A Python CGI script is called in exactly the sam
between a CGI and a static URL is that the form
serverconventionally, such scripts are confined
another directory name is used); Web servers g
scripts may live. When a CGI script runs, it is e
STDOUT, followed by a blank line, then finally s
often an HTML document. That is really all ther

CGI requests may utilize one of two methods: I
associated query data to the STDIN of the CGI
script). A GET request puts the query in an env
There is not a lot of difference between the two
query information in a Uniform Resource Identi
without HTML forms and saved/bookmarked. Fo
query to a script example discussed below:

<http://gnosis.cx/cgi-bin/simple.cgi?this=tha

You do not actually *need* the *cgi* module to crea
the script simple.cgi mentioned above:

simple.cgi

```
#!/usr/bin/python
import os,sys
print "Content-Type: text/html"
print
print "<html><head><title>Environmen
for k,v in os.environ.items():
    print k, "::",
    if len(v)<=40: print v
    else:          print v[:37]+"...
print "&lt;STDIN&gt; ::", sys.stdin.
print "</pre></body></html>"
```

I happen to have composed the above sample c
script from another Web page. Here is one that

http://gnosis.cx/simpleform.html

```
<html><head><title>Test simple.cgi</
<form action="cgi-bin/simple.cgi" me
<input type="hidden" name="this" val
<input type="text" value="" name="sp
<input type="submit" value="GET">
```

```
</form>
<form action="cgi-bin/simple.cgi" me
<input type="hidden" name="this" val
<input type="text" value="" name="sp
<input type="submit" value="POST">
</form>
</body></html>
```

It turns out that the script simple.cgi is modera
what it has to work with. For example, the quer
by the GET form on simpleform.html) returns a
(edited):

```
DOCUMENT_ROOT :: /www/gnosis
HTTP_ACCEPT_ENCODING :: gzip, deflat
CONTENT_TYPE :: application/x-www-fo
SERVER_PORT :: 80
REMOTE_ADDR :: 151.203.xxx.xxx
SERVER_NAME :: www.gnosis.cx
HTTP_USER_AGENT :: Mozilla/5.0 (Maci
REQUEST_URI :: /cgi-bin/simple.cgi?t
QUERY_STRING :: this=that&spam=eggs+
SERVER_PROTOCOL :: HTTP/1.1
HTTP_HOST :: gnosis.cx
REQUEST_METHOD :: GET
```

```
SCRIPT_NAME :: /cgi-bin/simple.cgi
SCRIPT_FILENAME :: /www/gnosis/cgi-b
HTTP_REFERER :: http://gnosis.cx/sim
<STDIN> ::
```

A few environment variables have been omittec
Web servers and setups. The most important va
perhaps want to make other decisions based or
HTTP_USER_AGENT, or HTTP_REFERER (yes, th
that STDIN is empty in this case. However, usir
will give a slightly different response (trimmed)

```
CONTENT_LENGTH :: 28
REQUEST_URI :: /cgi-bin/simple.cgi
QUERY_STRING ::
REQUEST_METHOD :: POST
<STDIN> :: this=that&spam=eggs+are+g
```

The CONTENT_LENGTH environment variable is
and STDIN contains the query. The rest of the c

A CGI script need not utilize any query data an
example, on some of my Web pages, I utilize a
reports back who "looks" at it. Web bugs have a
send HTML mail and want to verify receipt cove
some additional information about visitors to a
might contain, at bottom:

```
<img src="http://gnosis.cx/cgi-bin/v
```

The script itself is:

visitor.cgi

```
#!/usr/bin/python
import os
from sys import stdout
addr = os.environ.get("REMOTE_ADDR",
agent = os.environ.get("HTTP_USER_AG
fp = open('visitor.log','a')
fp.write('%s\t%s\n' % (addr, agent))
fp.close()
stdout.write("Content-type: image/gi
stdout.write('GIF89a\001\000\001\000
stdout.write('\000\000\000!\371\004\
stdout.write('\000\000\000\001\000\0
```

CLASSES

The point where the *cgi* module becomes usefu
class *cgi.FieldStorage* will determine the details
made, and decode the urlencoded query into a

these checks manually, but *cgi* makes it much e

cgi.FieldStorage([fp=sys.stdin [,headers [,ob [,keep_blank_values=0 [,strict_parsing=0]]]]]]

Construct a mapping object containing query in default arguments and construct a standard ins to use name indexing and also supports severa object will determine all relevant details of the

```
import cgi
query = cgi.FieldStorage()
eggs = query.getvalue('eggs','defaul
numfields = len(query)
if query.has_key('spam'):
    spam = query['spam']
[...]
```

When you retrieve a *cgi.FieldStorage* value by n string, but either an instance of *cgi.FieldStorag* or a list of such objects. The string query is in t may contain multiple fields with the same name of such values is returned. The safe way to read whether a list is returned:

```
if type(eggs) is type([]): # several
    for egg in eggs:
        print "<dt>Egg</dt>\n<dd>",
else:
    print "<dt>Eggs</dt>\n<dd>", egg
```

For special circumstances you might wish to ch
specifying an optional (named) argument. The
read for POST requests. The argument headers
headers to valuesusually consisting of {"Conter
the environment if no argument is given. The a
environment mapping is found. If you specify a
will be included for a blank HTML form fieldmap
is specified, a ValueError will be raised if there

METHODS

The methods .keys(), .values(), and .has_key()
The method .items(), however, is not supportec

cgi.FieldStorage.getfirst(key [,default=None])

Python 2.2+ has this method to return exactly
You cannot rely on which such string value will
form fields have the same namebut you are ass

a list.

cgi.FieldStorage.getlist(key [,default=None])

Python 2.2+ has this method to return a list of matches on the key key. This allows you to loop about whether they are a list or a single string.

```
>>> spam = form.getlist('spam')
>>> for s in spam:
...     print s
```

cgi.FieldStorage.getvalue(key [,default=None

Return a string or list of strings that are the val argument default is specified, return the specifi indexing by name, this method retrieves actual .value attribute.

```
>>> import sys, cgi, os
>>> from cStringIO import StringIO
>>> sys.stdin = StringIO("this=that&
>>> os.environ['REQUEST_METHOD'] = '
>>> form = cgi.FieldStorage()
```

```
>>> form.getvalue('this')
['that', 'other']
>>> form['this']
[MiniFieldStorage('this','that'),Mir
```

ATTRIBUTES

cgi.FieldStorage.file

If the object handled is an uploaded file, this at
While you can read the entire file contents as a
attribute, you may want to read it line-by-line i
.readlines() method of the file object.

cgi.FieldStorage.filename

If the object handled is an uploaded file, this at
HTML form to upload a file looks something like

```
<form action="upload.cgi" method="PC
      enctype="multipart/form-data">
  Name: <input name="" type="file" s
  <input type="submit" value="Uploac
```

```
</form>
```

Web browsers typically provide a point-and-clic

cgi.FieldStorage.list

This attribute contains the list of mapping obje
each object in the list is itself a *cgi.MiniStorage*
if you upload files that themselves contain mult

```
>>> form.list
[MiniFieldStorage('this', 'that'),
MiniFieldStorage('this', 'other'),
MiniFieldStorage('spam', 'good eggs'
```

SEE ALSO: cgi.FieldStorage.getvalue() *380*;

cgi.FieldStorage.value
cgi.MiniFieldStorage.value

The string value of a storage object.

SEE ALSO: urllib *388*; cgitb *382*; dict *24*;

cgitb • Traceback manager for CGI scripts

Python 2.2 added a useful little module for deb
it for earlier Python versions from <http://lfw.c
developing CGI scripts is that their normal outp
the underlying Web server and forwarded to an
traceback occurs due to a script error, that outp
at in a CGI context). A more useful action is eit
display them in the client browser.

Using the *cgitb* module to examine CGI script e
the top of your CGI script, simply include the li

Traceback enabled CGI script

```
import cgitb
cgitb.enable()
```

If any exceptions are raised, a pretty-formatted
a name starting with @).

METHODS

cgitb.enable([display=1 [,logdir=None [contex

Turn on traceback reporting. The argument dis
to the browseryou might not want this to happ
will have little idea what to make of such a rep
letting them see it). If logdir is specified, trace
The argument context indicates how many line
point where an error occurred.

For earlier versions of Python, you will have to
approach is:

Debugging CGI script in Python

```
import sys
sys.stderr = sys.stdout
def main():
    import cgi
    # ...do the actual work of the C
    # perhaps ending with:
    print template % script_dictiona
print "Content-type: text/html\n\n"
main()
```

This approach is not bad for quick debugging; e
Unfortunately, though, the traceback (if one oc
that you need to go to "View Source" in a brow

traceback. With a few more lines, we can add a

Debugging/logging CGI script in Python

```
import sys, traceback
print "Content-type: text/html\n\n"
try:                  # use explicit ex
    import my_cgi  # main CGI functi
    my_cgi.main()
except:
    import time
    errtime = '--- '+ time.ctime(tim
    errlog = open('cgi_errlog', 'a')
    errlog.write(errtime)
    traceback.print_exc(None, errlog
    print "<html>\n<head>"
    print "<title>CGI Error Encounte
    print "<body><p>A problem was en
    print "<p>Please check the serve
    print "</body></html>"
```

The second approach is quite generic as a wrap
write. Just import a different CGI module as ne
more detailed or friendlier.

SEE ALSO: cgi *376*;

5.2.2 Parsing, Creating, and Manipulating HTI

htmlentitydefs • HTML character entity re

The module *htmlentitydefs* provides a mapping
symbolic names of corresponding HTML 2.0 ent
have equivalents in the ISO-8859-1 character s
HTML numeric references instead.

ATTRIBUTES

htmlentitydefs.entitydefs

A dictionary mapping symbolic names to charac

```
>>> import htmlentitydefs
>>> htmlentitydefs.entitydefs['omega
'&#969;'
>>> htmlentitydefs.entitydefs['uuml'
'\xfc'
```

For some purposes, you might want a reverse c
8859-1 characters.

```
>>> from htmlentitydefs import entit
>>> iso8859_1 = dict([(v,k) for k,v
>>> iso8859_1['\xfc']
'uuml'
```

HTMLParser • Simple HTML and XHTML pa

The module *HTMLParser* is an event-based fram
contrast to *htmllib*, which is based on *sgmllib, H*
expressions to identify the parts of an HTML do
and so on. The different internal implementatio
of the modules.

I find the module *HTMLParser* much more straig
therefore *HTMLParser* is documented in detail ir
htmllib more or less *requires* the use of the anc
no extra difficultly in letting *HTMLParser* make c
to do this, for example, if you have an existing
format.

Both *HTMLParser* and *htmllib* provide an interfa
expat XML parsers. That is, a documentHTML o
events, with no data structure created to repres

documents, another processing API is the Docu
document as an in-memory hierarchical data st

In principle, you could use *xml.sax* or *xml.dom*
conformed with XHTMLthat is, tightened up HTI
problem is that very little existing HTML is XHT
HTML does not require closing tags in many cas
to be closed. But implicit closing tags can be in
with certain names). A popular tool like tidy do
this way. The more significant problem is sema
quite lax about tag matchingWeb browsers that
pages are quite complex software projects.

For example, a snippet like that below is quite l

```
<p>The <a href="http://ietf.org">IET
   <i>Be lenient in what you <b>acce
```

If you know even a little HTML, you know that t
wanted the whole quote in italics, the word acc
a data structure such as a DOM object is difficu
fairly lenient about what it will process; howeve
any other problem), the module will raise the e

SEE ALSO: htmllib *285*; xml.sax *405*;

CLASSES

HTMLParser.HTMLParser()

The *HTMLParser* module contains the single cla
is fairly useful, since it does not actually do any
Utilizing *HTMLParser.HTMLParser()* is a matter
handle the events you are interested in.

If it is important to keep track of the structural
document, you will need to maintain a data stru
certain that the document you are processing is
example:

HTMLParser_stack.py

```
#!/usr/bin/env python
import HTMLParser
html = """<html><head><title>Advice<
<p>The <a href="http://ietf.org">IET
   <i>Be strict in what you <b>send<
</body></html>
"""
tagstack = []
class ShowStructure(HTMLParser.HTMLP
    def handle_starttag(self, tag, a
```

```
    def handle_endtag(self, tag): ta
    def handle_data(self, data):
        if data.strip():
            for tag in tagstack: sys
            sys.stdout.write(' >> %s
ShowStructure().feed(html)
```

Running this optimistic parser produces:

```
% ./HTMLParser_stack.py
/html/head/title >> Advice
/html/body/p >> The
/html/body/p/a >> IETF admonishes:
/html/body/p/a/i >> Be strict in wha
/html/body/p/a/i/b >> send
/html/body/p/a/i >> .
```

You could, of course, use this context informati
particular bit of content (or when you process t

A more pessimistic approach is to maintain a "f
that will remove the most recent starttag corre
<p> and <blockquote> tags from nesting if no
do more along this line for a production applica
good start:

```
class TagStack:
    def __init__(self, lst=[]): self
    def __getitem__(self, pos): retu
    def append(self, tag):
        # Remove every paragraph-lev
        if tag.lower() in ('p','bloc
            self.lst = [t for t in s
                            if t not i
        self.lst.append(tag)
    def pop(self, tag):
        # "Pop" by tag from nearest
        self.lst.reverse()
        try:
            pos = self.lst.index(tag
        except ValueError:
            raise HTMLParser.HTMLPar
        del self.lst[pos]
        self.lst.reverse()
tagstack = TagStack()
```

This more lenient stack structure suffices to pa
given in the module discussion.

METHODS AND ATTRIBUTES

HTMLParser.HTMLParser.close()

Close all buffered data, and treat any current d

HTMLParser.HTMLParser.feed(data)

Send some additional HTML data to the parser
data. You may feed the instance with whatever
be processed, maintaining the previous state.

HTMLParser.HTMLParser.getpos()

Return the current line number and offset. Gen
report or analyze the state of the processing of

HTMLParser.HTMLParser.handle_charref(nan

Method called when a character reference is en
references may be interspersed with element te
construct a Unicode character from a character
Unicode (or raw character reference) to *HTMLPa*

```
class CharacterData(HTMLParser.HTMLP
```

```
    def handle_charref(self, name):
        import unicodedata
        char = unicodedata.name(unic
        self.handle_data(char)
    [...other methods...]
```

HTMLParser.HTMLParser.handle_comment(d

Method called when a comment is encountered
with --->. The argument data contains the cont

HTMLParser.HTMLParser.handle_data(data)

Method called when content data is encountere
the argument data, but if character or entity re
respective handler methods will be called in an

HTMLParser.HTMLParser.handle_decl(data)

Method called when a declaration is encountere
>. The argument data contains the contents of
like a type of declaration, but are handled by th
HTMLParser.HTMLParser.handle_comment() me

HTMLParser.HTMLParser.handle_endtag(tag)

Method called when an endtag is encountered.
(without brackets).

HTMLParser.HTMLParser.handle_entityref(na

Method called when an entity reference is enco
references occur in the middle of an element te
with calls to *HTMLParser.HTMLParser.handle_da*
the latter method with decoded entities; for exa

```
class EntityData(HTMLParser.HTMLPars
    def handle_entityref(self, name)
        import htmlentitydefs
        self.handle_data(htmlentityc
    [...other methods...]
```

HTMLParser.HTMLParser.handle_pi(data)

Method called when a processing instruction (P
end with ?>. They are less common in HTML th
data contains the contents of the PI.

HTMLParser.HTMLParser.handle_startendtag

Method called when an XHTML-style empty tag

```
<img src="foo.png" alt="foo"/>
```

The arguments tag and attrs are identical to th
HTMLParser.HTMLParser.handle_starttag().

HTMLParser.HTMLParser.handle_starttag(tag

Method called when a starttag is encountered.
(without brackets), and the argument attrs con
such as [("href","http://ietf.org)].

HTMLParser.HTMLParser.lasttag

The last tagstart or endthat was encountered.
structure like those discussed is more useful. B
You should treat it as read-only.

HTMLParser.HTMLParser.reset()

Restore the instance to its initial state, lose any
within unclosed tags).

5.2.3 Accessing Internet Resources

urllib • Open an arbitrary URL

The module *urllib* provides convenient, high-lev
While *urllib* lets you connect to a variety of prot
connectionsespecially issues of complex authen
instead. However, *urllib does* provide hooks for

The interface to *urllib* objects is file-like. You ca
connection for almost any function or class that
the World Wide Web, File Transfer Protocol (FTP
treated, almost transparently, as if it were part

Although the module provides two classes that
tuned control, generally in practice the function
need to the *urllib* module.

FUNCTIONS

urllib.urlopen(url [,data])

Return a file-like object that connects to the Ur
named in url. This resource may be an HTTP, FT
argument data can be specified to make a POS
urlencoded string, which may be created by the
is specified with an HTTP URL, the GET method

Depending on the type of resource specified, a
the instance, but each provides the methods: .
.close(), .info(), and .geturl() (but not .xreadlir

Most of the provided methods are shared by fil
interfacearguments and return valuesas actual
contains the URL that the object connects to, u

The method .info() returns *mimetools.Message*
documented in detail in this book, this object is
email.Message.Message objectspecifically, it re
and dictionary-like indexing:

```
>>> u = urllib.urlopen('urlopen.py')
>>> print 'u.info() '
<mimetools.Message instance at 0x62f
>>> print u.info()
Content-Type: text/x-python
Content-Length: 577
Last-modified: Fri, 10 Aug 2001 06:0
```

```
>>> u.info().keys()
['last-modified', 'content-length',
>>> u. info() ['content-type']
'text/x-python'
```

SEE ALSO: urllib.urlretrieve() *390*; urllib.urlenc

urllib.urlretrieve(url [,fname [,reporthook [,dat

Save the resources named in the argument url
fname is specified, that filename will be used; c
generated. The optional argument data may co
HTTP POST request, as with *urllib.urlopen()*.

The optional argument reporthook may be used
implement a progress meter for downloads. The
repeatedly with the arguments bl_transferred,
smaller than the block size will typically call rep
file_size will *approximately* equal bl_transferred

The return value of *urllib.urlretrieve()* is a pair
name of the created filethe same as the fname
return value is a *mimetools.Message* object, lik
urllib.urlopen object.

SEE ALSO: urllib.urlopen() *389*; urllib.urlencode

urllib.quote(s [,safe="/"])

Return a string with special characters escaped for being quoted.

```
>>> urllib.quote('/~username/special
'/%7Eusername/special%26odd%21'
```

urllib.quote_plus(s [,safe="/"])

Same as *urllib.quote()*, but encode spaces as +

urllib.unquote(s)

Return an unquoted string. Inverse operation o

urllib.unquote_plus(s)

Return an unquoted string. Inverse operation o

urllib.urlencode(query)

Return a urlencoded query for an HTTP POST o
either a dictionary-like object or a sequence of
preserved in the generated query.

```
>>> query = urllib.urlencode([('hl',
...                            ('q','
>>> print query
hl=en&q=Text+Processing+in+Python
>>> u = urllib.urlopen('http://googl
```

Notice, however, that at least as of the momen
results on this request because a Python shell i
provides a SOAP interface that is more lenient,
create a custom *urllib* class that spoofed an acc

CLASSES

You can change the behavior of the basic *urllib.*
by substituting your own class into the module
to use *urllib* classes:

```
import urllib
class MyOpener(urllib.FancyURLopener
    pass
urllib._urlopener = MyOpener()
```

```
u = urllib.urlopen("http://some.url"
```

urllib.URLopener([proxies [,x509]])**

Base class for reading URLs. Generally you sho
urllib.FancyURLopener unless you need to impl

The argument proxies may be specified with a
resources through a proxy. The keyword argum
authentication; specifically, you should give nar
case.

```
import urllib
proxies = {'http':'http://192.168.1.
urllib._urlopener = urllib.URLopener
```

urllib.FancyURLopener([proxies [,x509]])**

The optional initialization arguments are the sa
subclass further to use other arguments. This c
HTTP redirect codes, as well as 401 authenticat
urllib.FancyURLopener is the one actually used
it to add custom capabilities.

METHODS AND ATTRIBUTES

urllib.URLFancyopener.get_user_passwd(hos

Return the pair (user,passwd) to use for auther
the method .prompt_user_passwd() in turn. In
provide a GUI login interface or obtain authenti
such as a database.

urllib.URLopener.open(url [,data])
urllib.URLFancyopener.open(url [,data])

Open the URL url, optionally using HTTP POST

SEE ALSO: urllib.urlopen() *389*;

urllib.URLopener.open_unknown (url [,data])
urllib.URLFancyopener.open_unknown (url [,

If the scheme is not recognized, the .open() me
You can implement error reporting or fallback b

urllib.URLFancyopener.prompt_user_passwd

Prompt for the authentication pair (user,passwo
prompt within a GUI. If the authentication is no
means, directly overriding .get_user_passwd()

urllib.URLopener.retrieve(url [,fname [,reportl
urllib.URLFancyopener.retrieve(url [,fname [,r

Copies the URL url to the local file named fnam
reporthook if specified. Use the optional HTTP F

SEE ALSO: urllib.urlretrieve() *390*;

urllib.URLopener.version
urllib.URFancyLopener.version

The User Agent string reported to a server is co
urllib/##, where the *urllib* version number is us

urlparse • Parse Uniform Resource Locato

The module *urlparse* support just one fairly sim
enough for quick implementations to get wrong
resources on the Internet: access protocol, net

fragment. Using *urlparse*, you can break out ar
or generate URLs. The format of URLs is based

Notice that the *urlparse* module does not parse
but merely returns them as a field. For exampl
ftp://guest:gnosis@192.168.1.102:21//tmp/M
network (at least at the moment this is written
retrieve this file. Parsing this fairly complicated

```
>>> import urlparse
>>> url = 'ftp://guest:gnosis@192.16
>>> urlparse.urlparse(url)
('ftp', 'guest:gnosis@192.168.1.102:
'', '', '',)
```

While this information is not incorrect, this netw
all but the host are optional. The actual structu
bracket nesting to indicate optional component

```
[user[:password]@]host[:port]
```

The following mini-module will let you further p

location_parse.py

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

```
def location_parse(netloc):
    "Return tuple (user, passwd, hos
    if '@' not in netloc:
        netloc = ':@' + netloc
    login, net = netloc.split('@')
    if ':' not in login:
        login += ':'
    user, passwd = login.split(':')
    if ':' not in net:
        net += ':'
    host, port = net.split(':')
    return (user, passwd, host, port

#-- specify network location on comm
if __name__=='__main__':
    import sys
    print location_parse(sys.argv[1]
```

FUNCTIONS

urlparse.urlparse(url [,def_scheme="" [,fragm

Return a tuple consisting of six components of
query, fragment). A URL is assumed to follow tl

query#fragment. If a default scheme def_scher
in case no scheme is encoded in the URL itself.
fragments will not be split from other fields.

```
>>> from urlparse import urlparse
>>> urlparse('gnosis.cx/path/sub/fil
('http', '', 'gnosis.cx/path/sub/fil
>>> urlparse('gnosis.cx/path/sub/fil
('http', '', 'gnosis.cx/path/sub/fil
>>> urlparse('http://gnosis.cx/path/
...          'gopher', 1)
('http', 'gnosis.cx', '/path/file.cg
>>> urlparse('http://gnosis.cx/path/
...          'gopher', 0)
('http', 'gnosis.cx', '/path/file.cg
```

urlparse.urlunparse(tup)

Construct a URL from a tuple containing the fie
returned URL has canonical form (redundancy e
urlparse.urlunparse() are not precisely inverse
urlunparse (urlparse (s)) should be idempotent

urlparse.urljoin(base, file)

Return a URL that has the same base path as b
example:

```
>>> from urlparse import urljoin
>>> urljoin('http://somewhere.lan/pa
...                   'sub/other.html
'http://somewhere.lan/path/sub/other
```

In Python 2.2+ the functions *urlparse.urlsplit()*
These differ from *urlparse.urlparse()* and *urlpar*
does not split out params from path.

5.3 Synopses of Other Internet Modules

There are a variety of Internet-related modules in the standard library that will not be covered here in their specific usage. In the first place, there are two general aspects to writing Internet applications. The first aspect is the parsing, processing, and generation of messages that conform to various protocol requirements. These tasks are solidly inside the realm of text processing and should be covered in this book. The second aspect, however, are the issues of actually sending a message "over the wire": choosing ports and network protocols, handshaking, validation, and so on. While these tasks are important, they are outside the scope of this book. The synopses below will point you towards appropriate modules, though; the standard documentation, Python interactive help, or other texts can help with the details.

A second issue comes up also, moreover. As Internet standardsusually canonicalized in RFCshave evolved, and as Python libraries have become more versatile and robust, some newer modules have superceded older ones. In a similar way, for example, the *re* module replaced the older *regex* module. In the interests of backwards compatibility, Python has not dropped any Internet modules from its standard distributions. Nonetheless, the *email* module represents the current "best practice" for most tasks related to email and newsgroup message handling. The modules *mimify, mimetools, MimeWriter, multifile*, and *rfc822* are likely to be utilized in existing code, but for new applications, it is better to use the capabilities in *email* in their stead.

As well as standard library modules, a few third-party tools deserve special mention (at the bottom of this section). A large number of Python developers have created tools for various Internet-related tasks, but a small number of projects have reached a high degree of sophistication and a widespread usage.

5.3.1 Standard Internet-Related Tools

asyncore

Asynchronous socket service clients and servers.

Cookie

Manage Web browser cookies. Cookies are a common mechanism for managing state in Web-based applications. RFC-2109 and RFC-2068 describe the encoding used for cookies, but in practice MSIE is not very standards compliant, so the parsing is relaxed in the *Cookie* module.

SEE ALSO: cgi *376*; httplib *396*;

email.Charset

Work with character set encodings at a fine-tuned level. Other modules within the *email* package utilize this module to provide higher-level interfaces. If you need to dig deeply into character set conversions, you might want to use this module directly.

SEE ALSO: email *345*; email.Header *351*; unicode *423*; codecs *189*;

ftplib

Support for implementing custom File Transfer Protocol (FTP) clients. This protocol is detailed in RFC-959. For a full FTP application, *ftplib* provides a very good starting point; for the simple capability to retrieve publicly accessible files over FTP, *urIIib.urlopen()* is more direct.

SEE ALSO: urllib *388*; urllib2 *398*;

gopherlib

Gopher protocol client interface. As much as I am still personally fond of the gopher protocol, it is used so rarely that it is not worth documenting here.

httplib

Support for implementing custom Web clients. Higher-level access to the HTTP and HTTPS protocols than using raw *sockets* on ports 80 or 443, but lower-level, and more communications oriented, than using the higher-level *urllib* to access Web resources in a file-like way.

SEE ALSO: urllib *388*; socket *397*;

ic, icopen

Internet access configuration (Macintosh).

icopen

Internet Config replacement for open()

(Macintosh).

imghdr

Recognize image file formats based on their first few bytes.

mailcap

Examine the mailcap file on Unix-like systems. The files /etc/mailcap, /usr/etc/mailcap, /usr/local/etc/mailcap, and $HOME/.mailcap are typically used to configure MIME capabilities in client applications like mail readers and Web browsers (but less so now than a few years ago). See RFC-1524.

mhlib

Interface to MH mailboxes. The MH format consists of a directory structure that mirrors the folder organization of messages. Each message is contained in its own file. While the MH format is in many ways *better*, the Unix mailbox format seems to be more widely used. Basic access to a single folder in an MH hierarchy can be achieved with the *mailbox.MHMailbox* class, which satisfies most working requirements.

SEE ALSO: mailbox *372*; email *345*;

mimetools

Various tools used by MIME-reading or MIME-writing programs.

MimeWriter

Generic MIME writer.

mimify

Mimification and unmimification of mail messages.

netrc

Examine the netrc file on Unix-like systems. The file $HOME/.netrc is typically used to configure FTP clients.

SEE ALSO: ftplib *395*; urllib *388*;

nntplib

Support for Network News Transfer Protocol (NNTP) client applications. This protocol is defined in RFC-977. Although Usenet has a different distribution system from email, the message format of NNTP messages still follows the format defined in RFC-822. In particular, the *email* package, or the *rfc822* module, are useful for creating and modifying news messages.

SEE ALSO: email *345*; rfc822 *397*;

nsremote

Wrapper around Netscape OSA modules (Macintosh).

rfc822

RFC-822 message manipulation class. The *email* package is intended to supercede *rfc822*, and it is better to use *email* for new application development.

SEE ALSO: email *345*; poplib *368*; mailbox *372*; smtplib *370*;

select

Wait on I/O completion, such as sockets.

sndhdr

Recognize sound file formats based on their first few bytes.

socket

Low-level interface to BSD sockets. Used to communicate with IP addresses at the level underneath protocols like HTTP, FTP, POP3, Telnet, and so on.

SEE ALSO: ftplib *395*; gopherlib *395*; httplib *396*; imaplib *366*; nntplib *397*; poplib *368*; smtplib *370*; telnetlib *397*;

SocketServer

Asynchronous I/O on sockets. Under Unix, pipes can also be monitored with *select.socket* supports SSL in recent Python versions.

telnetlib

Support for implementing custom telnet clients. This protocol is detailed in RFC-854. While possibly useful for intranet applications, Telnet is an entirely unsecured protocol and should not really be used on the Internet. Secure Shell (SSH) is an encrypted protocol that otherwise is generally similar in capability to Telnet. There is no support for SSH in the Python standard library, but third-party options exist, such as *pyssh*. At worst, you can script an SSH client using a tool like the third-party *pyexpect*.

urllib2

An enhanced version of the *urllib* module that adds specialized classes for a variety of protocols. The main focus of *urllib2* is the handling of authentication and encryption methods.

SEE ALSO: urllib *388*;

Webbrowser

Remote-control interfaces to some browsers.

5.3.2 Third-Party Internet Related Tools

There are many very fine Internet-related tools that this book cannot discuss, but to which no slight is intended. A good index to such tools is the relevant page at the Vaults of Parnassus:

<http://py.vaults.ca/apyllo.py/812237977>

Quixote

In brief, *Quixote* is a templating system for HTML delivery. More so than systems like PHP, ASP, and JSP to an extent, *Quixote* puts an emphasis on Web application structure more than page appearance. The home page for *Quixote* is <http://www.mems-exchange.org/software/quixote/>

Twisted

To describe *Twisted*, it is probably best simply to quote from Twisted Matrix Laboratories' Web site <http://www.twistedmatrix.com/>:

> Twisted is a framework, written in Python, for writing networked applications. It includes implementations of a number of commonly used network services such as a Web server, an IRC chat server, a mail server, a relational database interface and an object broker. Developers can build

applications using all of these services as well as custom services that they write themselves. Twisted also includes a user authentication system that controls access to services and provides services with user context information to implement their own security models.

While *Twisted* overlaps significantly in purpose with *Zope, Twisted* is generally lower-level and more modular (which has both pros and cons). Some protocols supported by *Twisted*usually both server and clientand implemented in pure Python are SSH; FTP; HTTP; NNTP; SOCKSv4; SMTP; IRC; Telnet; POP3; AOL's instant messaging TOC; OSCAR, used by AOL-IM as well as ICQ; DNS; MouseMan; finger; Echo, discard, chargen, and friends; Twisted Perspective Broker, a remote object protocol; and XML-RPC.

Zope

Zope is a sophisticated, powerful, and just plain *complicated* Web application server. It incorporates everything from dynamic page generation, to database interfaces, to Web-based administration, to back-end scripting in several styles and languages. While the learning curve is steep, experienced Zope developers can develop and manage Web applications more easily, reliably, and faster than users of pretty much any other technology.

The home page for Zope is <http://zope.org/>.

5.4 Understanding XML

Extensible Markup Language (XML) is a text for
of storage and transport requirements. Parsing
element of many text processing applications. T
techniques for dealing with XML in Python. Whi
simplifying the exchange of complex and hierar
into a standard of considerable complexity. This
details of XML tools; an excellent book dedicate

Python & XML, Christopher A. Jones & Fred L.
00128-2.

The XML format is sufficiently rich to represent
straightforwardly than others. A task that XML
marked-up textdocumentation, books, articles,
XML is probably used more often to represent c
containers, and so on. In many of these cases,
extra verbosity. XML itself is more like a metala

syntax constraints that any XML document mus
document formats are defined as XML *dialects.*
set of tags that are used within a type of docur
to use those tags. What I refer to as an XML di
called "an *application* of XML."

THE DATA MODEL

At base, XML has two ways to represent data. A
values. Both names and values are Unicode str
but values frequently encode other basic dataty
XML Schemas. Attribute names are mildly restr
XML markup; attribute values can encode any s
escaped. XML attribute values are whitespace r
can itself also be escaped. A bare example is:

```
>>> from xml.dom import minidom
>>> x = '''<x a="b" d="e f g" num="3
>>> d = minidom.parseString(x)
>>> d.firstChild.attributes.items()
[(u'a', u'b'), (u'num', u'38'), (u'c
```

As with a Python dictionary, no order is defined
tag.

The second way XML represents data is by nest
a tag together with a corresponding "close tag"

an ordered sequence of *subelements.* The sube
nested subelements. A general term for any pa
element, an attribute, or one of the special par
example of an element that contains some sub

```
>>> x = '''<?xml version="1.0" encoc
... <root>
...   <a>Some data</a>
...   <b data="more data" />
...   <c data="a list">
...     <d>item 1</d>
...     <d>item 2</d>
...   </c>
... </root>'''
>>> d = minidom.parseString(x)
>>> d.normalize()
>>> for node in d.documentElement.ch
...     print node
...
<DOM Text node "
  ">
<DOM Element: a at 7033280>
<DOM Text node "
  ">
<DOM Element: b at 7051088>
```

```
<DOM Text node "
  ">
<DOM Element: c at 7053696>
<DOM Text node "
">
>>> d.documentElement.childNodes[3].
[(u'data', u'more data')]
```

There are several things to notice about the Py

1. **The "document element," named root ir subelement nodes, named a, b, and c.**

- Whitespace is preserved within elements. The between the subelements make up several text intermix, each potentially meaningful. Spacing nonetheless also often used for visual clarity (a

- The example contains an XML declaration, <? included.

- Any given element may contain attributes *anc*

OTHER XML FEATURES

Besides regular elements and text nodes, XML "special" nodes. Comments are common and us

be hand edited at some point (or even potentia
how a document is to be handled. Document ty
validity rules for where elements and attributes
CDATA lets you embed mini-XML documents or
documents, while leaving markup untouched. E

```
<?xml version="1.0" ?>
<!DOCTYPE root SYSTEM "sometype.dtd"
<root>
<!-- This is a comment -->
This is text data inside the &lt;roc
<![CDATA[Embedded (not well-formed)
         <this><that> >>string<< </t
</root>
```

XML documents may be either "well-formed" or
indicates that a document obeys the proper syr
general: All tags are either self-closed or follow
characters are escaped; tags are properly hiera
particular documents can also fail to be well-for
documents sensu stricto, but merely fragments
formed XML can be found at <http://www.w3.c
<http://www.w3.org/TR/xml11/>.

Beyond well-formedness, some XML documents
document matches a further grammatical speci
Definition (DTD), or in an XML Schema. The mo

W3C XML Schema specification, found in forma
<http://www.w3.org/TR/xmlschema-0/> and in
schema specifications, howeverone popular alte
documented at <http://www.oasis-open.org/cc

The grammatical specifications indicated by DT
can specify that certain subelements must occu
cardinality and order. Or, certain attributes may
simple case, the following DTD is one that the p
would conform to. There are an infinite number
but each one describes a slightly different *rang*

```
<!ELEMENT root ((a|OTHER-A)?, b, c*)
<!ELEMENT a (#PCDATA)>
<!ELEMENT b EMPTY>
<!ATTLIST b data CDATA #REQUIRED
            NOT-THERE (this | that)
<!ELEMENT c (d+)>
<!ATTLIST c data CDATA #IMPLIED>
<!ELEMENT d (#PCDATA)>
```

The W3C recommendation on the XML standard
features of the above DTD example can be note
attribute NOT-THERE are permitted by this DTD
sample XML document. The quantifications ?, *
sequence operator have similar meaning as in r
Attributes may be required or optional as well a

value types; for example, the data attribute mu
THERE attribute may contain this or that only.

Schemas go farther than DTDs, in a way. Beyor
attributes must contain strings describing partic
dates, schemas allow more flexible quantificatic
example, the following W3C XML Schema migh
purchases:

```
<xsd:element name="item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="USPrice" ty
      <xsd:element name="shipDate" t
                   minOccurs="0" max
    </xsd:sequence>
    <xsd:attribute name="partNum" ty
  </xsd:complexType>
</xsd:element>
<!-- Stock Keeping Unit, a code for
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string
      <xsd:pattern value="\d{3}-[A-Z
   </xsd:restriction>
</xsd:simpleType>
```

An XML document that is valid under this scher

```
<item partNum="123-XQ">
  <USPrice>21.95</USPrice>
  <shipDate>2002-11-26</shipDate>
</item>
```

Formal specifications of schema languages can
this example is meant simply to illustrate the ty

In order to check the validity of an XML docume
validating parser. Some stand-alone tools perfo
messages in cases of invalidity. As well, certain
within larger applications. As a rule, however, *n*
and check only for well-formedness.

Quite a number of technologies have been built
specified by W3C, OASIS, or other standards gr
aware of is XSLT. There are a number of thick b
matter is too complex to document here. But ir
declarative programming language whose synta
document is processed using a set of rules in a
output, often a different XML document. The ele
describe a pattern that might occur in a source
that will be produced if that pattern is encounte
anyway; in the details, "patterns" can have loo
find XSLT to be more complicated than genuine
technology for my own purposes, but you are fa

processes if you work with existing XML applica

5.4.1 Python Standard Library XML Modules

There are two principle APIs for accessing and widespread use: DOM and SAX. Both are suppo these two APIs make up the bulk of Python's X programming language neutral, and using them similar to using them in Python.

The Document Object Model (DOM) represents Nodes may be of several typesa document type comments, elements, and attribute mapsbut w strictly nested hierarchy. Typically, nodes have some nodes are *leaf nodes* without children. Th actions on nodes: delete nodes, add nodes, fin and other actions. The DOM itself does not spe is transformed (parsed) into a DOM representa serialized to an XML document. In practice, how *xml.dom*incorporate these capabilities. Formal

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

and:

<http://www.w3.org/TR/2000/REC-DOM-Leve

The Simple API for XML (SAX) is an *event-base* which envisions XML as a rooted tree of nodes, occurring linearly in a file, text, or other stream the sense of telling you very little inherently ab and also in the sense of being extremely memc sense that once a tag or content is processed, i manually save it in a data structure). However, to assure well-formedness of parsed documents case of problems in well-formedness; you may these. Formal specification of SAX can be founc

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

xml.dom

The module xml.dom is a Python implementatic Model, Level 2. As much as possible, its API fol conveniences are added as well. A brief exampl

```
>>> from xml.dom import minidom
>>> dom = minidom.parse('address.xml
>>> addrs = dom.getElementsByTagName
>>> print addrs[1].toxml()
<address city="New York" number="344
>>> jobs = dom.getElementsByTagName(
```

```
>>> for key, val in jobs[3].attribut
...     print key,'=',val
employee-type = Part-Time
is-manager = no
job-description = Hacker
```

SEE ALSO: gnosis.xml.objectify *409*;

xml.dom.minidom

The module *xml.dom.minidom* is a lightweight
You may pass in a custom SAX parser object w
default, *xml.dom.minidom* uses the fast, nonva

xml.dom.pulldom

The module *xml.dom.pulldom* is a DOM implem
building the portions of a DOM tree that are rec
some cases, this approach can be considerably
xml.dom.minidom or another DOM parser; how
somewhat underdocumented and experimental

xml.parsers.expat

Interface to the expat nonvalidating XML parser
xml.dom.minidom modules utilize the services
functionality lives mostly in a C library. You can
but since the interface uses the same general e
there is usually no reason to.

xml.sax

The package *xml.sax* implements the Simple AP
the underlying *xml.parser.expat* parser, but any
methods may be used instead. In particular, the
the *PyXML* package.

When you create a SAX application, your main
handlers that will process events generated dur
handler is a ContentHandler, but you may also
ErrorHandler. Generally you will specialize the b
own applications. After defining and registering
.parse() method of the parser that you register
incremental processing, you can use the feed()

A simple example illustrates usage. The applica
an equivalent, but not necessarily identical, doc
used as a canonical form of the document:

xmlcat.py

```
#!/usr/bin/env python
import sys
from xml.sax import handler, make_pa
from xml.sax.saxutils import escape

class ContentGenerator(handler.Conte
    def __init__(self, out=sys.stdou
        handler.ContentHandler.__ini
        self._out = out
    def startDocument(self):
        xml_decl = '<?xml version="1
        self._out.write(xml_decl)
    def endDocument(self):
        sys.stderr.write("Bye bye!\r
    def startElement(self, name, att
        self._out.write('<' + name)
        name_val = attrs.items()
        name_val.sort()
        for (name, value) in name_va
            self._out.write(' %s="%s
        self._out.write('>')
    def endElement(self, name):
        self._out.write('</%s>' % na
    def characters(self, content):
        self._out.write(escape(conte
```

```
    def ignorableWhitespace(self, co
        self._out.write(content)
    def processingInstruction(self,
        self._out.write('<?%s %s?>'

if __name__=='__main__':
    parser = make_parser()
    parser.setContentHandler(Content
    parser.parse(sys.argv[1])
```

xml.sax.handler

The module *xml.sax.handler* defines classes Co
and ErrorHandler that are normally used as par

xml.sax.saxutils

The module *xml.sax.saxutils* contains utility fur
Several functions allow escaping and munging s

xml.sax.xmlreader

The module *xml.sax.xmlreader* provides a fram

will be usable by the *xml.sax* module. Any new
conventions can be plugged in to the *xml.sax.n*

xmllib

Deprecated module for XML parsing. Use *xml.s*

xmlrpclib
SimpleXMLRPCServer

XML-RPC is an XML-based protocol for remote p
For the most part, the XML aspect is hidden fro
xmlrpclib to call remote methods and the modu
your own server that supports such method cal

```
>>> import xmlrpclib
>>> betty = xmlrpclib.Server("http:/
>>> print betty.examples.getStateNam
South Dakota
```

The XML-RPC format itself is a bit verbose, eve
you to pass argument values to a remote meth

```
>>> import xmlrpclib
```

```
>>> print xmlrpclib.dumps((xmlrpclib
<params>
<param>
<value><boolean>1</boolean></value>
</param>
<param>
<value><int>37</int></value>
</param>
<param>
<value><array><data>
<value><double>11.19999999999999</c
<value><string>spam</string></value>
</data></array></value>
</param>
</params>
```

SEE ALSO: gnosis.xml.pickle *410*;

5.4.2 Third-Party XML-Related Tools

A number of projects extend the XML capabiliti
principle author of several XML-related modules
package. Information on the current release ca

<http://gnosis.cx/download/Gnosis_Utils.ANN

The package itself can be downloaded as a *dist*

<http://gnosis.cx/download/Gnosis_Utils-curr

The Python XML-SIG (special interest group) pr
PyXML. The work of this group is incorporated i
Python releasesnot every *PyXML* tool, however,
given moment, the most sophisticatedand ofter
downloading the latest *PyXML* package. Be awa
overrides the default Python XML support and r

<http://pyxml.sourceforge.net/>

Fourthought, Inc. produces the *4Suite* package
Fourthought releases *4Suite* as free software, a
incorporated into the *PyXML* project (albeit at a
Fourthought is a for-profit company that also o
for *4Suite*. The community page for *4Suite* is:

<http://4suite.org/index.xhtml>

The Fourthought company Web site is:

<http://fourthought.com/>

Two other modules are discussed briefly below.
However, both *PYX* and *yaml* fill many of the sa
being easier to manipulate with text processing
edit by hand. There is a contrast between these

semantically identical to XML, merely using a d
has a quite different semantics from XMLI pres
concrete applications where developers might i
"buzz"), YAML is a better choice.

The home page for *PYX* is:

<http://pyxie.sourceforge.net/>

I have written an article explaining PYX in more

<http://gnosis.cx/publish/programming/xml_

The home page for YAML is:

<http://yaml.org>

I have written an article contrasting the utility a

<http://gnosis.cx/publish/programming/xml_

gnosis.xml.indexer

The module *gnosis.xml.indexer* builds on the fu
example in Chapter 2 (and contained in the *gn*
of file contents, *gnosis.xml.indexer* creates indi

for a kind of "reverse XPath" search. That is, w
package, lets you see the contents of an XML n
gnosis.xml.indexer identifies the XPaths to the
module may be used either in a larger applicati
example:

```
% indexer symmetric
./crypto1.xml::/section[2]/panel[8]/
./crypto1.xml::/section[2]/panel[8]/
./crypto1.xml::/section[2]/panel[7]/
./crypto2.xml::/section[4]/panel[6]/
4 matched wordlist: ['symmetric']
Processed in 0.100 seconds (SlicedZP

% indexer "-filter=*::/*/title" symm
./cryptol.xml::/section[2]/panel[8]/
./cryptol.xml::/section[2]/panel[7]/
2 matched wordlist: ['symmetric']
Processed in 0.080 seconds (SlicedZP
```

Indexed searches, as the example shows, are v
more details on this module:

<http://gnosis.cx/publish/programming/xml_

gnosis.xml.objectify

The module *gnosis.xml.objectify* transforms arb
that have a "native" feel to them. Where XML is
believe that using *gnosis.xml.objectify* is the qu
data in a Python application.

The Document Object Model defines an OOP mo
programming languages. But while DOM is nom
are distinctly un-Pythonic. For example, here is
(skipping whitespace text nodes for some indic

```
>>> from xml.dom import minidom
>>> dom_obj = minidom.parse('address
>>> dom_obj.normalize()
>>> print dom_obj.documentElement.ch
...                                .at
Los Angeles
```

In contrast, *gnosis.xml.objectify* feels like you a

```
>>> from gnosis.xml.objectify import
>>> xml_obj = XML_Objectify('address
>>> py_obj = xml_obj.make_instance()
>>> py_obj.person[2].address.city
u'Los Angeles'
```

gnosis.xml.pickle

The module *gnosis.xml.pickle* lets you serialize format. In most respects, the purpose is the sa target is useful for certain purposes. You may p standard XML parsers, XSLT processors, XML ed

In several respects, *gnosis.xml.pickle* offers fin module does. You can control security permissi representation of object types within an XML fil during the pickle/unpickle cycle; and several ot possible. However, in basic usage, *gnosis.xml.p* An example illustrates both the usage and the f

```
>>> class Container: pass
...
>>> inst = Container()
>>> dct = {1.7:2.5, ('t','u','p'):'t
>>> inst.this, inst.num, inst.dct =
>>> import gnosis.xml.pickle
>>> print gnosis.xml.pickle.dumps(in
<?xml version="1.0"?>
<!DOCTYPE PyObject SYSTEM "PyObjects
<PyObject module="__main__" class="C
<attr name="this" type="string" valu
<attr name="dct" type="dict" id="600
  <entry>
    <key type="tuple" id="5973680" >
```

```
      <item type="string" value="t"
      <item type="string" value="u"
      <item type="string" value="p"
    </key>
    <val type="string" value="tuple"
  </entry>
  <entry>
    <key type="numeric" value="1.7"
    <val type="numeric" value="2.5"
  </entry>
</attr>
<attr name="num" type="numeric" valu
</PyObject>
```

SEE ALSO: pickle *93*; cPickle *93*; yaml *415*; pp

gnosis.xml.validity

The module *gnosis.xml.validity* allows you to de
their containment according to XML validity cor
always produce string representations that are
formed ones. When you attempt to add an item
that is not permissible, a descriptive exception
specify quantification, subelement types, and s

For example, suppose you wish to create docur
Document Type Definition:

dissertation.dtd

```
<!ELEMENT dissertation (dedication?,
<!ELEMENT dedication (#PCDATA)>
<!ELEMENT chapter (title, paragraph+
<!ELEMENT title (#PCDATA)>
<!ELEMENT paragraph (#PCDATA I figur
<!ELEMENT figure EMPTY>
<!ELEMENT table EMPTY>
<!ELEMENT appendix (#PCDATA)>
```

You can use *gnosis.xml.validity* to assure your
documents. First, you create a Python version

dissertation.py

```
from gnosis.xml.validity import *
class appendix(PCDATA):   pass
class table(EMPTY):       pass
class figure(EMPTY):      pass
class _mixedpara(Or):     _disjoins
```

```
class paragraph(Some):     _type = _m
class title(PCDATA):       pass
class _paras(Some):        _type = pa
class chapter(Seq):        _order = (
class dedication(PCDATA): pass
class _apps(Any):          _type = ap
class _chaps(Some):        _type = ch
class _dedi(Maybe):        _type = de
class dissertation(Seq):   _order = (
```

Next, import your Python validity constraints, a

```
>>> from dissertation import *
>>> chap1 = LiftSeq(chapter,('About
>>> paras_ch1 = chap1[1]
>>> paras_ch1 += [paragraph('OOP car
>>> print chap1
<chapter><title>About Validity</titl
<paragraph>It is a good thing</parag
<paragraph>OOP can enforce it</parag
</chapter>
```

If you attempt an action that violates constrain
example:

```
>>> try:
```

```
..      paras_ch1.append(dedication("
.. except ValidityError, x:
...     print x
Items in _paras must be of type <cla
(not <class 'dissertation.dedicatior
```

PyXML

The *PyXML* package contains a number of capa
standard library. *PyXML* was at version 0.8.1 at
number indicates, it remains an in-progress/be
last released version of Python was 2.2.2, with
this, *PyXML* will probably be at a later number a
current features will have been incorporated int
where is a moving target.

Some of the significant features currently availa
library are listed below. You may install *PyXML*
override the existing XML support.

- A validating XML parser written in Python ca
  program rather than a C extension, *xmlproc*
  underlying *expat* parser).

- A SAX extension called *xml.sax.writers* that
  or other formats.

- A fully compliant DOM Level 2 implementatio
- Support for canonicalization. That is, two XM even though they are not byte-wise identica attribute orders, character entities, and som *meaning* of the document. Two canonicalized identical if and only if they are byte-wise ide
- XPath and XSLT support, with implementatio faster XSLT implementations around, howev
- A DOM implementation, called *xml.dom.pull* nodes has been incorporated into recent ver Python versions, this is available in *PyXML*.
- A module with several options for serializing comparable to *gnosis.xml.pickle*, but I like th

PYX

PYX is both a document format and a Python m format. As well as the Python module, tools wri documents between XML and PYX format.

The idea behind PYX is to eliminate the need fo node in an XML document is represented, in the prefix character to indicate the node type. Most

exception of document type declarations, comn
could be incorporated into an updated PYX forn

Documents in the PYX format are easily proces
processing tools like sed, grep, awk, sort, wc, a
basic *FILE.readline()* loop are equally able to pı
makes it much easier to use familiar text proce
is with XML. A brief example illustrates the PYX

```
% cat test.xml
<?xml version="1.0"?>
<?xml-stylesheet href="test.css" typ
<Spam flavor="pork">
  <Eggs>Some text about eggs.</Eggs>
  <MoreSpam>Ode to Spam (spam="smoke
</Spam>
% ./xmln test.xml
?xml-stylesheet href="test.css" type
Aflavor pork
-\n
(Eggs
-Some text about eggs. )Eggs
-\n
(MoreSpam
-Ode to Spam (spam="smoked-pork")
)MoreSpam
```

```
-\n
)Spam
```

4Suite

The tools in *4Suite* focus on the use of XML doc
server element of the *4Suite* software is useful
documents, searching them, transforming them
address a variety of XML technologies. In some
technologies not found in the Python standard l
4Suite provides more advanced implementation

Among the XML technologies implemented in *4*
XPointer, XLink and XPath, and SOAP. Among th
performing XSLT transformations. *4xpath* lets y
powerful XPath descriptions of how to reach the
documents use to identify their semantic chara

I detail *4Suite* technologies in a bit more detail

<http://gnosis.cx/publish/programming/xml_

yaml

The native data structures of object-oriented pr

straightforward to represent in XML. While XML
represent any compound data, the only inheren
that only maps strings to strings. Moreover, eve
a given data structure, the XML is quite verbos
especially to edit manually.

The YAML format is designed to match the stru
languages: Python, Perl, Ruby, and Java all hav
writing. Moreover, the YAML format is extremel
acronym cutely stands for "YAML Ain't Markup l
as a better pretty-printer than *pprint*, while sim
be used for configuration files or to exchange d
languages.

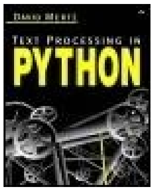
There is no fully general and clean way, howeve
can use the *yaml* module to read YAML data file
to read and write to one particular XML format.
XML dialects than *gnosis.xml.pickle*, there are a
and YAML representations of the same data. On
plusthere is essentially a straight-forward and c
Python data structures and YAML representatio

In the YAML example below, refer back to the s
gnosis.xml.pickle and *pprint* in their respective
in this case unlike *pprint*the serialization can be
object (or to create a different object after edit

```
>>> class Container: pass
```

```
...
>>> inst = Container()
>>> dct = {1.7:2.5, ('t','u','p'):'t
>>> inst.this, inst.num, inst.dct =
>>> import yaml
>>> print yaml.dump(inst)
--- !!__main__.Container
dct:
    1.7: 2.5
    ?
        - t
        - u
        - p
: tuple
num: 38
this: that
```

SEE ALSO: pprint *94*; gnosis.xml.pickle *410*;



Text Processing in PythonBy David Mertz

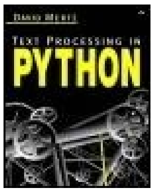
Table of Contents

Appendix A. A Selective and Impressionistic Short Review of Python

A reader who is coming to Python for the first time would be well served reading Guido van Rossum's *Python Tutorial*, which can be downloaded from <http://python.org/>, or picking up one of the several excellent books devoted to teaching Python to novices. As indicated in the Preface, the audience of this book is a bit different.

The above said, some readers of this book might use Python only infrequently, or not have used Python for a while, or may be sufficiently versed in numerous other programming languages, that a quick review on Python constructs suffices for understanding. This appendix will briefly mention each major element of the Python language itself, but will not address any libraries (even standard and ubiquitous ones that may be discussed in the main chapters). Not all fine points of syntax and semantics will be covered here, either. This review, however, should suffice for a reader to understand all the examples in this book.

Even readers who are familiar with Python might enjoy skimming this review. The focus and spin of this summary are a bit different from most introductions. I believe that the way I categorize and explain a number of language features can provide a moderately novelbut equally accurateperspective on the Python language. Ideally, a Python programmer will come away from this review with a few new insights on the familiar constructs she uses every day. This appendix does not shy away from using some abstract terms from computer scienceif a particular term is not familiar to you, you will not lose much by skipping over the sentence it occurs in; some of these terms are explained briefly in the Glossary.



Appendix A. A Selective and Impressionistic Short Review of Python

A.1 What Kind of Language Is Python?

Python is a byte-code compiled programming language that supports multiple programming paradigms. Python is sometimes called an interpreted and/or scripting language because no separate compilation step is required to run a Python program; in more precise terms, Python uses a virtual machine (much like Java or Smalltalk) to run machine-abstracted instructions. In most situations a byte-code compiled version of an application is cached to speed future runs, but wherever necessary compilation is performed "behind the scenes."

In the broadest terms, Python is an imperative programming language, rather than a declarative (functional or logical) one. Python is dynamically and strongly typed, with very late binding compared to most languages. In addition, Python is an object-oriented language with strong introspective facilities, and one that generally relies on conventions rather than enforcement mechanisms to control access and visibility of names. Despite its object-oriented core, much of the syntax of Python is designed to allow a convenient procedural style that masks the underlying OOP mechanisms. Although Python allows basic functional programming (FP) techniques, side effects are the norm, evaluation is always strict, and no compiler optimization is performed for tail recursion (nor on almost any other construct).

Python has a small set of reserved words, delimits blocks and structure based on indentation only, has a fairly rich collection of built-in data structures, and is generally both terse and readable compared to other programming languages. Much of the strength of Python lies in its standard library and in a flexible system of importable modules and packages.

A.2 Namespaces and Bindings

The central concept in Python programming is t
scope) in a Python program has available to it a
namespaces; each namespace contains a set o
object. In older versions of Python, namespace
"three-scope rule" (builtin/global/local), but Pyt
nested scoping. In most cases you do not need
scoping works the way you would expect (the s
of lexical scoping are mostly ones with nested f

There are quite a few ways of binding a name t
namespace/scope and/or within some other sco
below.

A.2.1 Assignment and Dereferencing

A Python statement like x=37 or y="foo" does

"foo"does not exist, Python creates one. If such
Next, the name x or y is added to the current r
and that name is bound to the corresponding o
current namespace, it is re-bound. Multiple nar
scopes/namespaces, can be bound to the same

A simple assignment statement binds a name ir
name has been declared as global. A name dec
(module-level) namespace instead. A qualified
statement binds a name into a specified names
object, or to the namespace of a module/packa

```
>>> x = "foo"                  # bind 'x
>>> def myfunc():              # bind 'm
...     global x, y            # specify
...     x = 1                  # rebind
...     y = 2                  # create
...     z = 3                  # create
...
>>> import package.module      # bind na
>>> package.module.w = 4       # bind 'w
>>> from mymod import obj      # bind ob
>>> obj.attr = 5               # bind na
```

Whenever a (possibly qualified) name occurs or
a line by itself, the name is dereferenced to the
bound inside some accessible scope, it cannot b

raises a NameError exception. If the name is fo
(possibly with comma-separated expressions be
invoked/called after it is dereferenced. Exactly
controlled and overridden for Python objects; b
method runs some code, and invoking a class c

```
>>> pkg.subpkg.func()    # invoke a f
>>> x = y                # deref 'y'
```

A.2.2 Function and Class Definitions

Declaring a function or a class is simply the pre
binding it to a name. But the def and class decl
assignments. In the case of functions, the *lamb*
right of an assignment to bind an "anonymous"
direct technique for classes, but their declaratio

```
>>> add1 = lambda x,y: x+y # bind 'a
>>> def add2(x, y):        # bind 'a
...     return x+y
...
>>> class Klass:           # bind 'K
...    def meth1(self):    # bind 'm
...        return 'Myself'
```

A.2.3 import Statements

Importing, or importing *from,* a module or a pa
current namespace. The import statement has
effect.

Statements of the forms

```
>>> import modname
>>> import pkg.subpkg.modname
>>> import pkg.modname as othername
```

add a new module object to the current names
define namespaces that you can bind values in

Statements of the forms

```
>>> from modname import foo
>>> from pkg.subpkg.modname import f
```

instead add the names foo or bar to the curren
import, any statements in the imported module
the forms is simply the effect upon namespaces

There is one more special form of the import st

```
>>> from modname import *
```

The asterisk in this form is not a generalized gl
special syntactic form. "Import star" imports ev

the current namespace (except those named w
be explicitly imported if needed). Use of this fo
risks adding names to the current namespace t
that may rebind existing names.

A.2.4 for Statements

Although for is a looping construct, the way it v
of an iterable object to a name (in the current r
are (almost) equivalent:

```
>>> for x in somelist:  # repeated b
...     print x
...
>>> ndx = 0             # rebinds 'r
>>> while 1:            # repeated b
...    x = somelist[ndx]
...    print x
...    ndx = ndx+1
...    if ndx >= len(somelist):
...        del ndx
...        break
```

A.2.5 except Statements

The except statement can optionally bind a nar

```
>>> try:
...     raise "ThisError", "some mes
... except "ThisError", x:     # Binc
...     print x
...
some message
```

Team-Fly

A.3 Datatypes

Python has a rich collection of basic datatypes.
you to hold heterogeneous elements inside the
minor limitations). It is straightforward, therefo
Python.

Unlike many languages, Python datatypes come
immutable. All of the atomic datatypes are imm
The collections list and dict are mutable, as are
datatype is simply a question of whether object
place"an immutable object can only be created
during its existence. One upshot of this distinct
as dictionary keys, but mutable objects may no
want a data structureespecially a large onethat
program operation, you should choose a mutab

Most of the time, if you want to convert values
explicit conversion/encoding call is required, bu

rules to allow numeric expressions over a mixtu
listed below with discussions of each. The built-
the datatype of an object.

A.3.1 Simple Types

bool

Python 2.3+ supports a Boolean datatype with
earlier versions of Python, these values are typ
2.3+, the Boolean values behave like numbers
micro-releases of Python (e.g., 2.2.1) include th
Boolean datatype.

int

A signed integer in the range indicated by the r
platform. For most current platforms, integers r
(2**31)-1. You can find the size on your platfor
are the bottom numeric type in terms of promo
integer, but integers are sometimes promoted t
string may be explicitly converted to an int usin

SEE ALSO: int *18;*

long

An (almost) unlimited size integral number. A lo
followed by an 1 or L (e.g., 34L, 98765432101)
that overflow *sys.maxint* are automatically prom
may be explicitly converted to a long using the

float

An IEEE754 floating point number. A literal floa
an int or long by containing a decimal point and
37., .453e-12). A numeric expression that invo
promotes all component types to floats before p
long, or string may be explicitly converted to a

SEE ALSO: float *19;*

complex

An object containing two floats, representing re
number. A numeric expression that involves bot
types promotes all component types to complex
There is no way to spell a literal complex in Pyt
the usual way of computing a complex value. A
indicates an imaginary number. An int, long, or

complex using the *complex()* function. If two fl
complex(), the second is the imaginary compor
complex(1.1,2)).

string

An immutable sequence of 8-bit character valu
languages, there is no "character" type in Pyth
length one. String objects have a variety of me
methods always return a new string object rath
The built-in *chr()* function will return a length-c
passed integer. The *str()* function will return a s
object. For example:

```
>>> ord('a')
97
>>> chr(97)
'a'
>>> str(97)
'97'
```

SEE ALSO: string *129;*

unicode

An immutable sequence of Unicode characters.
Unicode character, but Unicode strings of lengtl
Unicode strings contain a similar collection of m
latter, Unicode methods return new Unicode ob
object. See Chapter 2 and Appendix C for addit

A.3.2 String Interpolation

Literal strings and Unicode strings may contain
contains format codes, values may be *interpola*
and a tuple or dictionary giving the values to su

Strings that contain format codes may follow ei
pattern uses format codes with the syntax %[fl
Interpolating a string with format codes on this
tuple of matching length and content datatypes
interpolated, you may give the bare item rather
example:

```
>>> "float %3.1f, int %+d, hex %06x"
'float 1.2, int +1234, hex 0004d2'
>>> '%e' % 1234
'1.234000e+03'
>>> '%e' % (1234,)
'1.234000e+03'
```

The (slightly) more complex pattern for format

format code, which is then used as a string key
syntax of this pattern is %(key)[flags][len[.pre
with this style of format codes requires % comb
all the named keys, and whose corresponding v
example:

```
>>> dct = {'ratio':1.234, 'count':12
>>> "float %(ratio)3.1f, int %(count
'float 1.2, int +1234, hex 0004d2'
```

You *may not* mix tuple interpolation and diction
string.

I mentioned that datatypes must match format
different range of datatypes, but the rules are a
Generally, numeric data will be promoted or de
complex types cannot be used for numbers.

One useful style of using dictionary interpolatio
namespace dictionary. Regular bound names de
strings.

```
>>> s = "float %(ratio)3.1f, int %(c
>>> ratio = 1.234
>>> count = 1234
>>> offset = 1234
>>> s % globals()
```

```
'float 1.2, int +1234, hex 0004d2'
```

If you want to look for names across scope, yo
both local and global names:

```
>>> vardct = {}
>>> vardct.update(globals())
>>> vardct.update(locals())
>>> interpolated = somestring % varc
```

The flags for format codes consist of the followi

```
0 Pad to length with leading zeros
- Align the value to the left withir
- (space) Pad to length with leading
+ Explicitly indicate the sign of pc
```

When a length is included, it specifies the *minir*
formatting. Numbers that will not fit within a le
specified. When a precision is included, the len
decimal are included in the total length:

```
>>> '[%f]' % 1.234
'[1.234000]'
>>> '[%5f]' % 1.234
'[1.234000]'
```

```
>>> '[%.1f]' % 1.234
'[1.2]'
>>> '[%5.1f]' % 1.234
'[  1.2]'
>>> '[%05.1f]' % 1.234
'[001.2]'
```

The formatting types consist of the following:

```
d Signed integer decimal
i Signed integer decimal
o Unsigned octal
u Unsigned decimal
x Lowercase unsigned hexadecimal
X Uppercase unsigned hexadecimal
e Lowercase exponential format float
E Uppercase exponential format float
f Floating point decimal format
g Floating point: exponential format
G Uppercase version of 'g'
c Single character: integer for chr(
r Converts any Python object using r
s Converts any Python object using s
% The '%' character, e.g.: '%%%d' %
```

One more special format code style allows the u
case, the interpolated tuple must contain an ex
each format code, preceding the value to forma

```
>>> "%0*d # %0*.2f" % (4, 123, 4, 1.
'0123 # 1.23'
>>> "%0*d # %0*.2f" % (6, 123, 6, 1.
'000123 # 001.23'
```

A.3.3 Printing

The least-sophisticated form of textual output i
particular, the STDOUT and STDERR streams ca
sys.stdout and *sys.stderr*. Writing to these is ju
example:

```
>>> import sys
>>> try:
...     # some fragile action
...     sys.stdout.write('result of a
... except:
...     sys.stderr.write('could not c
...
result of action
```

You cannot seek within STDOUT or STDERRgen pure sequential outputs.

Writing to STDOUT and STDERR is fairly inflexil
statement accomplishes the same purpose mor
sys.stdout.write() only accept a single string as
any number of arguments of any type. Each arg
equivalent of repr(obj). For example:

```
>>> print "Pi: %.3f" % 3.1415, 27+11
Pi: 3.142 38 {1: 2, 3: 4} (1, 2, 3)
```

Each argument to the print statment is evaluat
argument is passed to a function. As a consequ
an object is printed, rather than the exact form
example, the dictionary prints in a different ord
spacing of the list and dictionary is slightly diffe
peformed and is a very common means of defir

There are a few things to watch for with the pri
between each argument to the statement. If yo
a separating space, you will need to use string
get the right result. For example:

```
>>> numerator, denominator = 3, 7
>>> print repr(numerator)+"/"+repr(c
3/7
>>> print "%d/%d" % (numerator, denc
```

By default, a print statement adds a linefeed to
eliminate the linefeed by adding a trailing comm
up with a space added to the end:

```
>>> letlist = ('a','B','Z','r','w')
>>> for c in letlist: print c,   # i
...
a B Z r w
```

Assuming these spaces are unwanted, you mus
otherwise calculate the space-free string you w

```
>>> for c in letlist+('\n',): # no s
...     sys.stdout.write(c)
...
aBZrw
>>> print ''.join(letlist)
aBZrw
```

There is a special form of the print statement tl
other than STDOUT. The print statement itself c
signs, then a writable file-like object, then a co
(printed) arguments. For example:

```
>>> print >> open('test','w'), "Pi:
```

```
>>> open('test').read()
'Pi: 3.142 38\n'
```

Some Python programmers (including your aut
"noisy," but it *is* occassionally useful for quick c

If you want a function that would do the same
one does so, but without any facility to eliminat
output:

```
def print_func(*args):
    import sys
    sys.stdout.write(' '.join(map(re
```

Readers could enhance this to add the missing
statement is the clearest approach, generally.

SEE ALSO: sys.stderr *50;* sys.stdout *51;*

A.3.4 Container Types

tuple

An immutable sequence of (heterogeneous) ob
membership and length of a tuple cannot be m
elements and subsequences can be accessed b

tuples can be constructed from such elements a
"records" in some other programming language

The constructor syntax for a tuple is commas b
parentheses around a constructed list are requi
constructs such as function arguments, but it is
construct a tuple. Some examples:

```
>>> tup = 'spam','eggs','bacon','sau
>>> newtup = tup[1:3] + (1,2,3) + (t
>>> newtup
('eggs', 'bacon', 1, 2, 3, 'sausage'
```

The function *tuple()* may also be used to constr
(either a list or custom sequence type).

SEE ALSO: tuple *28;*

list

A mutable sequence of objects. Like a tuple, lis
subscripting and slicing; unlike a tuple, list met
can modify the length and membership of a list

The constructor syntax for a list is surrounding
constructed with no objects between the braces

an object name; longer lists separate each eler
slices, of course, also use square braces, but th
Python grammar (and common sense usually p
examples:

```
>>> lst = ['spam', (1,2,3), 'eggs',
>>> lst[:2]
['spam', (1, 2, 3)]
```

The function *list()* may also be used to construc
(either a tuple or custom sequence type).

SEE ALSO: list *28;*

dict

A mutable mapping between immutable keys a
dict exists for a given key; adding the same ke
overrides the previous entry (much as with bin
unordered, and entries are accessed either by k
contained objects using the methods .keys(), .v
Python versionswith the .popitem() method. All
objects in an unspecified order.

The constructor syntax for a dict is surrounding
constructed with no objects between the bracke

dict is separated by a colon, and successive pai
example:

```
>>> dct = {1:2, 3.14:(1+2j), 'spam':
>>> dct['spam']
'eggs'
>>> dct['a'] = 'b'    # add item to
>>> dct.items()
[('a', 'b'), (1, 2), ('spam', 'eggs'
>>> dct.popitem()
('a', 'b')
>>> dct
{1: 2, 'spam': 'eggs', 3.14: (1+2j)}
```

In Python 2.2+, the function *dict()* may also be
sequence of pairs or from a custom mapping ty

```
>>> d1 = dict([('a','b'), (1,2), ('s
>>> d1
{'a': 'b', 1: 2, 'spam': 'eggs'}
>>> d2 = dict(zip([1,2,3],['a','b','
>>> d2
{1: 'a', 2: 'b', 3: 'c'}
```

SEE ALSO: dict *24;*

sets.Set

Python 2.3+ includes a standard module that ir
Python versions, a number of developers have
sets. If you have at least Python 2.2, you can c
<http://tinyurl.com/2d31> (or browse the Pytl
definition True,False=1, 0 to your local version,

A set is an unordered collection of hashable obj
in a set more than once; a set resembles a dict
utilize bitwise and Boolean syntax to perform b
test does not have a special syntactic form, ins
.issuperset() methods. You may also loop throu
order. Some examples illustrate the type:

```
>>> from sets import Set
>>> x = Set([1,2,3])
>>> y = Set((3,4,4,6,6,2)) # init wi
>>> print x, '//', y       # make su
Set([1, 2, 3]) // Set([2, 3, 4, 6])
>>> print x | y            # union c
Set([1, 2, 3, 4, 6])
>>> print x & y            # interse
Set([2, 3])
>>> print y-x              # differe
Set([4, 6])
```

```
>>> print x ^ y                # symmetr
Set([1, 4, 6])
```

You can also check membership and iterate ove

```
>>> 4 in y                    # membersh
1
>>> x.issubset(y)             # subset c
0
>>> for i in y:
...     print i+10,
...
12 13 14 16
>>> from operator import add
>>> plus_ten = Set(map(add, y, [10]*
>>> plus_ten
Set([16, 12, 13, 14])
```

sets.Set also supports in-place modification of s
does not allow modification.

```
>>> x = Set([1,2,3])
>>> x |= Set([4,5,6])
>>> x
Set([1, 2, 3, 4, 5, 6])
>>> x &= Set([4,5,6])
```

```
>>> x
Set([4, 5, 6])
>>> x ^= Set ([4, 5])
>>> x
Set([6])
```

A.3.5 Compound Types

class instance

A class instance defines a namespace, but this act as a data container (but a container that als has methods). A class instance (or any namesp of creating a mapping between names and valu be set or modified using standard qualified nam methods by qualifying with the namespace of tl conventionally called self. For example:

```
>>> class Klass:
...     def setfoo(self, val):
...         self.foo = val
...
>>> obj = Klass()
>>> obj.bar = 'BAR'
>>> obj.setfoo(['this','that','other
```

```
>>> obj.bar, obj.foo
('BAR', ['this', 'that', 'other'])
>>> obj.__dict__
{'foo': ['this', 'that', 'other'], '
```

Instance attributes often dereference to other c
hierarchically organized namespace quantificati
Moreover, a number of "magic" methods named
underscores provide optional syntactic convenie
The most common of these magic methods is ._
(often utilizing arguments). For example:

```
>>> class Klass2:
...     def __init__(self, *args, **
...         self.listargs = args
...         for key, val in kw.items
...             setattr(self, key, v
...
>>> obj = Klass2(1, 2, 3, foo='FOO',
>>> obj.bar.blam = 'BLAM'
>>> obj.listargs, obj.foo, obj.bar.b
((1, 2, 3), 'FOO', 'BAZ', 'BLAM')
```

There are quite a few additional "magic" metho
Many of these methods let class instances beha
maintaining special class behaviors). For examp

methods control the string representation of an
.__setitem__() methods allow indexed access t
indices, or list-like numbered indices); methods
.__pow__(), and .__abs__() allow instances to
Python Reference Manual discusses magic meth

In Python 2.2 and above, you can also let insta
by inheriting classes from these built-in types.
datatype whose "shape" contains both a mutab
attribute. Two ways to define this datatype are:

```
>>> class FooList(list):          # wo
...     def __init__(self, lst=[], f
...         list.__init__(self, lst)
...         self.foo = foo
...
>>> foolist = FooList([1,2,3], 'FOO'
>>> foolist[1], foolist.foo
(2, 'FOO')
>>> class oldFooList:             # wo
...     def __init__(self, lst=[], f
...         self._lst, self.foo = ls
...     def append(self, item):
...         self._lst.append(item)
...     def __getitem__(self, item):
...         return self._lst[item]
```

```
...     def __setitem__(self, item,
...         self._lst [item] = val
...     def __delitem__(self, item):
...         del self._lst[item]
...
>>> foolst2 = oldFooList([1,2,3], 'F
>>> foolst2[1], foolst2.foo
(2, 'FOO')
```

If you need more complex datatypes than the b
whose class has magic methods, often these ca
whose attributes are bound in link-like fashion t
be constructed according to various topologies,
modeling graphs). As a simple example, you ca
using the following node class:

```
>>> class Node:
...     def __init__(self, left=None
...         self.left, self.value, s
...     def __repr__(self):
...         return self.value
...
>>> tree = Node(Node(value="Left Lea
...             "Tree Root",
...             Node(left=Node(value
...                  right=Node(valu
```

```
>>> tree,tree.left,tree.left.left,tr
(Tree Root, Left Leaf, None, RightLe
```

In practice, you would probably bind intermedi
easy pruning and rearrangement.

SEE ALSO: int *18*; float *19*; list *28*; string *129*;
UserString *33*;

A.4 Flow Control

Depending on how you count it, Python has abo
mechanisms, which is much simpler than most
Python's collection of mechanisms is well chose
highdegree of orthogonality between them.

From the point of view of this appendix, except
flow control techniques. In a language like Java
"happy" if it does not throw any exceptions at a
exceptions less "exceptional"a perfectly good d
when an exception is raised.

Two additional aspects of the Python language
flow control, but nonetheless amount to such w
functional programming style operations on list
heart, flow control constructs.

A.4.1 if/then/else Statements

Choice between alternate code paths is general
its optional elif and else components. An if bloc
at the end of the compound statement, zero or
followed by a Boolean expression and a colon. I
expression and colon. The else statement, if it
it, just a colon. Each statement introduces a bl
(indented on the following lines or on the same

Every expression in Python has a Boolean value
literal. Any empty container (list, dict, tuple) is
Unicode string is false; the number 0 (of any n
whose class defines a .__nonzero__() or .__len
return a false value. Without these special meth
time, Boolean expressions consist of compariso
actually evaluate to the canonical objects "0" o
<=, <>, !=, is, is not, in, and not in. Sometime
an expression.

Only one block in an "if/elif/else" compound sta
multiple conditions hold, the first one that evalu

```
>>> if 2+2 <= 4:
...    print "Happy math"
...
Happy math
```

```
>>> x = 3
>>> if x > 4: print "More than 4"
... elif x > 3: print "More than 3"
... elif x > 2: print "More than 2"
... else: print "2 or less"
...
More than 2
>>> if isinstance(2, int):
...     print "2 is an int"     # 2.
... else:
...     print "2 is not an int"
```

Python has no "switch" statement to compare c
matches. Occasionally, the repetition of an expr
lines looks awkward. A "trick" in such a case is
following are equivalent, for example:

```
>>> if var.upper() == 'ONE':     val
... elif var.upper() == 'TWO':   val
... elif var.upper() == 'THREE': val
... elif var.upper() == 'FOUR':  val
... else:                        val
...
>>> switch = {'ONE':1, 'TWO':2, 'THF
>>> val = switch.get(var.upper(), 0)
```

A.4.2 Boolean Shortcutting

The Boolean operators or and and are "lazy." Tl
evaluates only as far as it needs to determine t
disjoin of an or is true, the value of that disjoin
without evaluating the rest; if the first conjoin
becomes the value of the whole expression.

Shortcutting is formally sufficient for switching
concise than "if/elif/else" blocks. For example:

```
>>> if this:              # 'if' compour
...     result = this
... elif that:
...     result = that
... else:
...     result = 0
...
>>> result = this or that or 0 # boc
```

Compound shortcutting is also possible, but not

```
>>> (cond1 and func1()) or (cond2 ar
```

A.4.3 for/continue/break Statements

The for statement loops over the elements of a
utilizes an iterator object (which may not have
sequences like lists, tuples, and strings are aut
statements. In earlier Python versions, a few s
xrange() also act as iterators.

Each time a for statement loops, a sequence/it
variable. The loop variable may be a tuple with
for multiple names in each loop. For example:

```
>>> for x,y,z in [(1,2,3),(4,5,6),(7
...
1 2 3 * 4 5 6 * 7 8 9 *
```

A particularly common idiom for operating on e

```
>>> for key,val in dct.items():
...     print key, val, '*',
...
1 2 * 3 4 * 5 6 *
```

When you wish to loop through a block a certai
use the range() or xrange() built-in functions t
length. For example:

```
>>> for _ in range(10):
...     print "X",      # '_' is not
```

```
...
X X X X X X X X X X
```

However, if you find yourself binding over a ran
indicates that you have not properly understoo
operating on a collection of related *things* that
loop, not just a need to do exactly the same th

If the continue statement occurs in a for loop, t
executing later lines in the block. If the break s
passes past the loop without executing later lin
occurs in a try).

A.4.4 map(), filter(), reduce(), and List Compre

Much like the for statement, the built-in functio
actions based on a sequence of items. Unlike a
a value resulting from this application to each i
programming style functions accepts a function
sequence(s) as a subsequent argument(s).

The *map()* function returns a list of items of the
where each item in the result is a "transformati
explicitly want such transformed items, use of
clearer than an equivalent for loop; for example

```
>>> nums = (1,2,3,4)
```

```
>>> str_nums = []
>>> for n in nums:
...     str_nums.append(str(n))
...
>>> str_nums
['1', '2', '3', '4']
>>> str_nums = map(str, nums)
>>> str_nums
['1', '2', '3', '4']
```

If the function argument of *map()* accepts (or c
sequences can be given as later arguments. If s
lengths, the shorter ones are padded with None
given as the function argument, producing a se
argument sequences.

```
>>> nums = (1,2,3,4)
>>> def add(x, y):
...     if x is None: x=0
...     if y is None: y=0
...     return x+y
...
>>> map(add, nums, [5,5,5])
[6, 7, 8, 4]
>>> map(None, (1,2,3,4), [5,5,5])
```

```
[(1, 5), (2, 5), (3, 5), (4, None)]
```

The *filter()* function returns a list of those items
condition given by the function argument. The
parameter, and its return value is interpreted a
example:

```
>>> nums = (1,2,3,4)
>>> odds = filter(lambda n: n%2, num
>>> odds
(1, 3)
```

Both *map()* and *filter()* can use function argum
making it possiblebut not usually desirableto re
filter() function. For example:

```
>>> for x in seq:
...     # bunch of actions
...     pass
...
>>> def actions(x):
...     # same bunch of actions
...     return 0
...
>>> filter(actions, seq)
[]
```

Some epicycles are needed for the scoping of b
statements. But as a general picture, it is wortl
between these very different-seeming techniqu

The *reduce()* function takes as a function argur
addition to a sequence second argument, *reduc*
as an initializer. For each item in the input sequ
aggregate result with the item, until the sequer
map() and *filter()*has a loop-like effect of opera
main purpose is to create some sort of aggrega
many items. For example:

```
>>> from operator import add
>>> sum = lambda seq: reduce(add, se
>>> sum([4,5,23,12])
44
>>> def tastes_better(x, y):
...     # some complex comparison of
...     # either return x, or returr
...     # ...
...
>>> foods = [spam, eggs, bacon, toas
>>> favorite = reduce(tastes_better,
```

List comprehensions (listcomps) are a syntactic
2.0. It is easiest to think of list comprehensions
the *map()* or *filter()* functions. That is, like the

produce lists of items, based on "input" sequen
for and if that are familiar from statements. Mo
a compound list comprehension expression tha
map() and *filter()* functions.

For example, consider the following small probl
string of characters; you would like to construc
number from the list and a character from the s
larger than the number. In traditional imperativ

```
>>> bigord_pairs = []
>>> for n in (95,100,105):
...     for c in 'aei':
...         if ord(c) > n:
...             bigord_pairs.append(
...
>>> bigord_pairs
[(95, 'a'), (95, 'e'), (95, 'i'), (1
```

In a functional programming style you might w

```
>>> dupelms=lambda lst,n: reduce(lam
...                              map
>>> combine=lambda xs,ys: map(None,x
>>> bigord_pairs=lambda ns,cs: filte
...
```

```
>>> bigord_pairs((95,100,105),'aei')
[(95, 'a'), (95, 'e'), (100, 'e'), (
```

In defense of this FP approach, it has not *only* a
provided the general combinatorial function cor
still rather obfuscated.

List comprehensions let you write something th

```
>>> [(n,c) for n in (95,100,105) for
[(95, 'a'), (95, 'e'), (95, 'i'), (1
```

As long as you have listcomps available, you ha
since it just amounts to repeating the for claus

Slightly more formally, a list comprehension co
square brackets (like a list constructor, which it
not by requirement, contains some names that
more for clauses that bind a name repeatedly (
clauses that limit the results. Generally, but not
some names that were bound by the for clause

List comprehensions may nest inside each othe
listcomp loops over a list that is defined by ano
listcomp is even used inside a listcomp's expres
as easy to produce difficult-to-read code by exc
nesting *map()* and *filter()* functions. Use cautio
nesting.

It is worth noting that list comprehensions are functional programming style calls. Specifically, bound in the enclosing scope (or global if the n put a minor extra burden on you to choose dist listcomps.

A.4.5 while/else/continue/break Statements

The while statement loops over a block as long remains true. If an else block is used within a c the expression becomes false, the else block is if the while expression is initially false.

If the continue statement occurs in a while loop without executing later lines in the block. If the control passes past the loop without executing break occurs in a try). If a break occurs in a wh

If a while statement's expression is to go from name in the expression will be re-bound within will depend on an external condition, such as a a call to a function whose Boolean value change the most common Python idiom for while stater a block. Some examples:

```
>>> command = ''
>>> while command != 'exit':
```

```
...     command = raw_input('Commanc
...     # if/elif block to dispatch
...
Command > someaction
Command > exit
>>> while socket.ready():
...     socket.getdata()  # do somet
... else:
...     socket.close()    # cleanup
...
>>> while 1:
...     command = raw_input('Commanc
...     if command == 'exit': break
...     # elif's for other commands
...
Command > someaction
Command > exit
```

A.4.6 Functions, Simple Generators, and the

Both functions and object methods allow a kinc
but one that is quite restrictive. A function or m
enters at its top, executes any statements enco
context as soon as a return statement is reache
invocation of a function or method is basically a

Python 2.2 introduced a flow control construct,
style of nonlocal branching. If a function or met
then it becomes a *generator function,* and invo
iterator instead of a simple value. A generator i
method that returns values. Any instance objec
generator iterator's method is special in having

In a standard function, once a return statemen
discards all information about the function's flo
returned value might contain some information
always gone. A generator iterator, in contrast, '
all local bindings, between each invocation of it
a calling context each place a yield statement is
body, but the calling context (or any context wi
able to jump back to the flow point where this l

In the abstract, generators seem complex, but
example:

```
>>> from __future__ import generator
>>> def generator_func():
...     for n in [1,2]:
...         yield n
...     print "Two yields in for loc
...     yield 3
...
>>> generator_iter = generator_func(
```

```
>>> generator_iter.next()
1
>>> generator_iter.next()
2
>>> generator_iter.next()
Two yields in for loop
3
>>> generator_iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
StopIteration
```

The object generator_iter in the example can b
to and returned from functions, just like any ot
generator_iter.next() jumps back into the last f
body yielded.

In a sense, a generator iterator allows you to p
statements of some (older) languages, but still
programming. The most common usage for ger
Most of the time, generators are used as "iterat

```
>>> for n in generator_func():
...     print n
...
1
```

```
2
Two yields  in for loop
3
```

In recent Python versions, the StopIteration ex
loop. The generator iterator's .next() method is
possible by the for statement. The name indica
bound to the values the yield statement(s) retu

A.4.7 Raising and Catching Exceptions

Python uses exceptions quite broadly and proba
programming language. In fact there are certai
awkward to express by means other than raisir

There are two general purposes for exceptions
actions can be invalid or disallowed in various w
zero; you cannot open (for reading) a filename
require arguments of specific types; you canno
of an assignment; and so on. The exceptions ra
names of the form [AZ].*Error. Catching *error* e
recover from a problem condition and restore a
such error exceptions are not caught in an appl
debugging clues since they appear in traceback

The second purpose for exceptions is for circum
"exceptional." But understand "exceptional" in a

indicates a programming or computer error, bu
the norm." For example, Python 2.2+ iterators
more items can be generated. Most such implie
however; it is merely the case that they contai
run out only once at the end. It's not "the norm
it is often expected that this will happen eventu

In a sense, raising an exception can be similar
cause control flow to leave a block. For exampl

```
>>> n = 0
>>> while 1:
...     n = n+1
...     if n > 10: break
...
>>> print n
11
>>> n = 0
>>> try:
...     while 1:
...         n = n+1
...         if n > 10: raise "ExitLc
... except:
...     print n
...
11
```

In two closely related ways, exceptions behave
the first place, exceptions could be described a
contexts is considered a sin akin to "GOTO," bu
know at compile time exactly where an excepti
else, it is caught by the Python interpreter). It
or a containing block, and so on; or it might be
called it, or something that called the caller, an
its way through execution contexts until it finds
propagation of exceptions is quite opposite to t
scoped bindings (or even to the earlier "three-s

The corollary of exceptions' dynamic scope is th
exit gracefully from deeply nested loops. The "Z
is better than nested." And indeed it is so, if yo
you should probably refactor (e.g., break loops
nesting *just deeply enough,* dynamically scope
Consider the following small problem: A "Ferma
integers (i,j,k) such that "i**2 + j**2 == k**2
any Fermat triples exist with all three integers i
(but entirely nonoptimal) solution is:

```
>>> def fermat_triple(beg, end):
...     class EndLoop(Exception): pa
...     range_ = range(beg, end)
...     try:
...         for i in range_:
...             for j in range_:
```

```
...                  for k in range_:
...                      if i**2 + j*
...                          raise Er
...     except EndLoop, triple:
...         # do something with 'tri
...         return i,j,k
...
>>> fermat_triple(1,10)
(3, 4, 5)
>>> fermat_triple(120,150)
>>> fermat_triple(100,150)
(100, 105, 145)
```

By raising the EndLoop exception in the middle
catch it again outside of all the loops. A simple
out of the most deeply nested block, which is p
for setting a "satisfied" flag and testing for this
approach is much simpler. Since the except blo
with the triple, it could have just been returned
case, other actions can be required before a re

It is not uncommon to want to leave nested loo
the sense of an "*Error" exception. Sometimes
discover a problem condition within nested bloc
outside the nesting. Some typical examples are
missing dictionary keys or list indices, and so o

statements to the calling position that really ne
support functions as if nothing can go wrong. F

```
>>> try:
...     result = complex_file_operat
... except IOError:
...     print "Cannot open file", fi
```

The function complex_file_operation() should n
what to do if a bad filename is given to itthere
context. Instead, such support functions can si
upwards, until some caller takes responsibility f

The try statement has two forms. The try/exce
the try/finally form is useful for "cleanup handl

In the first form, a try block must be followed b
except may specify an exception or tuple of exc
may omit an exception (tuple), in which case it
caught by an earlier except block. After the exc
an else block. The else block is run only if no ex
example:

```
>>> def except_test(n):
...     try: x = 1/n
...     except IOError: print "IO Er
...     except ZeroDivisionError: pr
```

```
...     except: print "Some Other Er
...     else: print "All is Happy"
...
>>> except_test(1)
All is Happy
>>> except_test(0)
Zero Division
>>> except_test('x')
Some Other Error
```

An except test will match either the exception a
exception. It tends to make sense, therefore, ir
from related ones in the *exceptions* module. Fo

```
>>> class MyException(IOError): pass
>>> try:
...    raise MyException
... except IOError:
...    print "got it"
...
got it
```

In the try/finally form of the try statement, the
cleanup code. If no exception occurs in the try
that. If an exception *was* raised in the try block
original exception is re-raised at the end of the

statement is executed in a finally blockor if a n
(including with the raise statement)the finally b
original exception disappears.

A finally statement acts as a cleanup block eve
contains a return, break, or continue statement
not run all the way through, finally is still enter
accomplish. A typical use of this compound stat
resource at the very start of the try block, then
may not succeed in the rest of the block; the fi
file gets closed, whether or not all the actions c

The try/finally form is never strictly needed sin
last exception. It is possible, therefore, to have
statement to propagate an error upward after t
cleanup action is desired whether or not except
form can save a few lines and express your int

```
>>> def finally_test(x):
...     try:
...         y = 1/x
...         if x > 10:
...             return x
...     finally:
...         print "Cleaning up..."
...     return y
...
```

```
>>> finally_test(0)
Cleaning up...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in finally
ZeroDivisionError: integer division
>>> finally_test(3)
Cleaning up...
0
>>> finally_test(100)
Cleaning up...
100
```

A.4.8 Data as Code

Unlike in languages in the Lisp family, it is *usua* programs that execute data values. It is *possib* strings during program runtime using several b *codeop, imp*, and *new* provide additional capab interactive shell itself is an example of a progra input, then executes them. So clearly, this appr

Other than in providing an interactive environm know Python), a possible use for the "data as c themselves generate Python code, either to rur application. At a simple level, it is not difficult t

based on templatized functionality; for this to b
program to contain some customization that wa

eval(s [,globals=globals() [,locals=locals()]])

Evaluate the expression in string s and return t
specify optional arguments globals and locals to
name lookup. By default, use the regular globa
that only an expression can be evaluated, not a

Most of the time when a (novice) programmer t
some valueoften numericbased on data encode
line in a report file contains a list of dollar amou
these numbers. A naive approach to the proble

```
>>> line = "$47  $33  $51  $76"
>>> eval("+".join([d.replace('$', ''
207
```

While this approach is generally slow, that is no
significant issue is that *eval()* runs code that is
could contain Python code that causes harm to
an application to malfunction. Imagine that inst
contained os.rmdir("/"). A better approach is to
int(), *float()*, and so on.

```
>>> nums = [int(d.replace('$', ''))
>>> from operator import add
>>> reduce(add, nums)
207
```

exec

The *exec* statement is a more powerful sibling (
code may be run if passed to the *exec* stateme
allows optional namespace specification, as wit

```
exec code [in globals [,locals]]
```

For example:

```
>>> s = "for i in range(10):\n  prir
>>> exec s in globals(), locals()
0 1 2 3 4 5 6 7 8 9
```

The argument code may be either a string, a co
eval(), the security dangers and speed penaltie
convenience provided. However, where code is
are occasionally uses for this statement.

__import__(s [,globals=globals() [,locals=loca

Import the module named s, using namespace
argument fromlist may be omitted, but if specif
[""]the fully qualified subpackage will be import
statement is the way you import modules, but
of s is not determined until runtime, use __*imp*

```
>>> op = __import__('os.path',global
>>> op.basename('/this/that/other')
'other'
```

input([prompt])

Equivalent to eval(raw_input (prompt)), along w
eval() generally. Best practice is to always use
existing programs.

raw_input([prompt])

Return a string from user input at the terminal.
console-based applications.

```
>>> s = raw_input('Last Name: ')
Last Name: Mertz
>>> s
```

'Mertz'

A.5 Functional Programming

This section largely recapitulates briefer descrip
common unfamiliarity with functional programm
Additional material on functional programming
naturecan be found in articles at:

<http://gnosis.cx/publish/programming/charr

<http://gnosis.cx/publish/programming/charr

<http://gnosis.cx/publish/programming/charr

It is hard to find any consensus about exactly w
either its proponents or detractors. It is not rea
feature of languages, and to what extent a feat
a book about Python, we can leave aside discus
languages like Lisp, Scheme, Haskell, ML, Ocan
we can focus on what makes a Python program

Programs that lean towards functional program
paradigms, tend to have many of the following

1. **Functions are treated as first-class obje
   to other functions and methods, and ret**

- Solutions are expressed more in terms of *wha*
how the computation is performed.

- Side effects, especially rebinding names repea
referentially transparent (see Glossary).

- Expressions are emphasized over statements;
describe how a result collection is related to a p
objects.

- The following Python constructs are used prev
filter(), reduce(), apply(), zip(), and *enumerate*
operator; list comprehensions; and switches ex

Many experienced Python programmers conside
as a feature. The main drawback of a functiona
elsewhere) is that it is easy to write unmaintair
using it. Too many *map(), reduce()*, and *filter()*
all the self-evidence of Python's simple stateme
unnamed *lambda* functions into the mix makes
discussion in Chapter 1 of higher-order functior

A.5.1 Emphasizing Expressions Using lambd

The *lambda* operator is used to construct an "a
more common def declaration, a function creat
single expression as a result, not a sequence o
There are inelegant ways to emulate statement
should think of *lambda* as a less-powerful cousi

Not all Python programmers are happy with the
benefit in readability to giving a function a desc
style below is clearly more readable than the fi

```
>>> from math import sqrt
>>> print map(lambda (a,b): sqrt(a**
[5.0, 13.038404810405298, 35.9026461
>>> sides = ((3,4),(7,11),(35,8))
>>> def hypotenuse(ab):
...     a,b = ab[:]
...     return sqrt(a**2+b**2)
...
>>> print map(hypotenuse, sides)
[5.0, 13.038404810405298, 35.9026461
```

By declaring a named function hypotenuse(), th
much more clear. Once in a while, though, a fu
(e.g., in *Tkinter, xml.sax*, or *mx.TextTools*) rea

only adds noise.

However, you may notice in this book that I fai
to define a name. For example, you might see

```
>>> hypotenuse = lambda (a,b): sqrt(
```

This usage is mostly for documentation. A side
saved in assigning an anonymous function to a
concision is not particularly important. This fun
explicitly that I do not expect any side effectsli
structureswithin the hypotenuse() function. Wh
that fact is not advertised; you have to look thr
Strictly speaking, there are wayslike calling *set*
lambda, but as a convention, I avoid doing so,

Moreover, a second documentary goal is servec
above. Whenever this form occurs, it is possibl
expression anywhere the left-hand name occur
parentheses usually, however). By using this fo
simply a short-hand for the defined expression.

```
>>> hypotenuse = lambda a,b: sqrt(a*
>>> (lambda a,b: sqrt(a**2+b**2))(3,
(5.0, 5.0)
```

Bindings with def, in general, lack substitutabili

A.5.2 Special List Functions

Python has two built-in functions that are strict
are frequently useful in conjunction with the "fu

zip(seq1 [,seq2 [,...]])

The *zip()* function, in Python 2.0+, combines m
tuples. Think of the teeth of a zipper for an ima

The function *zip()* is almost the same as map(N
reaches the end of the shortest sequence. For e

```
>>> map(None, (1,2,3,4), [5,5,5])
[(1, 5), (2, 5), (3, 5), (4, None)]
>>> zip((1,2,3,4), [5,5,5])
[(1, 5), (2, 5), (3, 5)]
```

Especially in combination with *apply()*, extende
unpacking, *zip()* is useful for operating over mu
example:

```
>>> lefts, tops = (3, 7, 35), (4, 11
>>> map(hypotenuse, zip(lefts, tops)
[5.0, 13.038404810405298, 35.9026461
```

A little quirk of *zip()* is that it is *almost* its own
syntax is needed for inversion, though. The exp
(as an exercise, play with variations). Consider

```
>>> sides = [(3, 4), (7, 11), (35, 8
>>> zip(*zip(*sides))
[(3, 4), (7, 11), (35, 8)]
```

enumerate(collection)

Python 2.3 adds the *enumerate()* built-in functi
index positions at the same time. Basically, enu
zip(range(len(seq)),seq), but *enumerate()* is a
the entire list to loop over. A typical usage is:

```
>>> items = ['a','b']
>>> i = 0        # old-style explicit
>>> for thing in items:
...     print 'index',i,'contains',t
...     i += 1
index 0 contains a
index 1 contains b
>>> for i,thing in enumerate(items):
...     print 'index',i,'contains',t
...
```

```
index 0 contains a
index 1 contains b
```

A.5.3 List-Application Functions as Flow Con

I believe that text processing is one of the area judicious use of functional programming techni conciseness. A strength of FP stylespecifically tl *filter()*, and *reduce()*is that they are not merely *sequences.* In text processing contexts, most lo of text, frequently over lines. When you wish to items, FP style allows the code to focus on the side issues of loop constructs and transient var

In part, a *map(), filter()*, or *reduce()* call is a ki an instruction to perform an action a number o functions. For example:

```
for x in range(100):
    sys.stdout.write(str(x))
```

and:

```
filter(sys.stdout.write, map(str, ra
```

are just two different ways of calling the str() fu sys.stdout.write() method with each result). Th

does not bother rebinding a name for each itera
application function returns a valuea list for *ma*
value for *reduce()* . Functions/methods like *sys*
their side effects almost always return None; by
around these, you avoid constructing a throwaw
empty list.

A.5.4 Extended Call Syntax and apply()

To call a function in a dynamic way, it is someti
arguments in data structures prior to the call. U
several positional arguments is awkward, and u
arguments simply cannot be done with the Pyth
example, consider the salutation() function:

```
>>> def salutation(title,first,last,
...     print prefix,
...     if use_title: print title,
...     print '%s %s,' % (first, las
...
>>> salutation('Dr.','David','Mertz'
To: Dr. David Mertz,
```

Suppose you read names and prefix strings fro
call salutation() with arguments determined at

```
>>> rec = get_next_db_record()
>>> opts = calculate_options(rec)
>>> salutation(rec[0], rec[1], rec[2
...              use_title=opts.get('u
...              prefix=opts.get('pref
```

This call can be performed more concisely as:

```
>>> salutation(*rec, **opts)
```

Or as:

```
>>> apply(salutation, rec, opts)
```

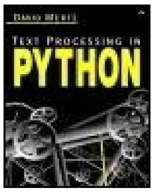
The calls func(*args,**keywds) and apply(func
argument args must be a sequence of the same
The (optional) argument keywds is a dictionary
matching keyword arguments (if not, it has no

In most cases, the extended call syntax is more
resembles the *declaration* syntax of generic pos
a few casesparticularly in higher-order function
still useful. For example, suppose that you have
an action immediately or defer it for later, depe
program this application as:

```
defer_list = []
```

```
if some_runtime_condition():
    doIt = apply
else:
    doIt = lambda *x: defer_list.app
#...do stuff like read records and c
doIt(operation, args, keywds)
#...do more stuff...
#...carry out deferred actions...
map(lambda (f,args,kw): f(*args,**kw
```

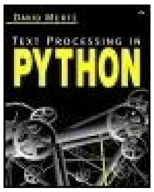
Since *apply()* is itself a first-class function rathe
aroundor in the example, bind it to a name.



Text Processing in PythonBy David Mertz

Table of Contents

Appendix B. A Data Compression Primer



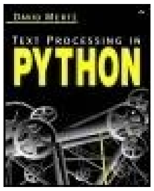
Appendix B. A Data Compression Primer

B.1 Introduction

See Section 2.2.5 for details on compression capabilities included in the Python standard library. This appendix is intended to provide readers who are unfamiliar with data compression a basic background on its techniques and theory. The final section of this appendix provides a practical exampleaccompanied by some demonstration codeof a Huffman-inspired custom encoding.

Data compression is widely used in a variety of programming contexts. All popular operating systems and programming languages have numerous tools and libraries for dealing with data compression of various sorts. The right choice of compression tools and libraries for a particular application depends on the characteristics of the data and application in question: streaming versus file; expected patterns and regularities in the data; relative importance of CPU usage, memory usage, channel demands, and storage requirements; and other factors.

Just what is data compression, anyway? The short answer is that data compression removes *redundancy* from data; in information-theoretic terms, compression increases the *entropy* of the compressed text. But those statements are essentially just true by definition. Redundancy can come in a lot of different forms. Repeated bit sequences (11111111) are one type. Repeated byte sequences are another (XXXXXXXX). But more often redundancies tend to come on a larger scale, either regularities of the data set taken as a whole, or sequences of varying lengths that are relatively common. Basically, what data compression aims at is finding algorithmic transformations of data representations that will produce more compact representations given "typical" data sets. If this description seems a bit complex to unpack, read on to find some more practical illustrations.



Appendix B. A Data Compression Primer

B.2 Lossless and Lossy Compression

There are actually two fundamentally different "styles" of data compression: lossless and lossy. This appendix is generally about lossless compression techniques, but the reader would be served to understand the distinction first. Lossless compression involves a transformation of the representation of a data set such that it is possible to reproduce *exactly* the original data set by performing a decompression transformation. Lossy compression is a representation that allows you to reproduce something "pretty much like" the original data set. As a plus for the lossy techniques, they can frequently produce far more compact data representations than lossless compression techniques can. Most often lossy compression techniques are used for images, sound files, and video. Lossy compression may be appropriate in these areas insofar as human observers do not perceive the literal bit-pattern of a digital image/sound, but rather more general "gestalt" features of the underlying image/sound.

From the point of view of "normal" data, lossy compression is not an option. We do not want a program that does "about the same" thing as the one we wrote. We do not want a database that contains "about the same" kind of information as what we put into it. At least not for most purposes (and I know of few practical uses of lossy compression outside of what are already approximate mimetic representations of the real world, likes images and sounds).

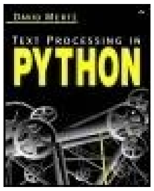
B.3 A Data Set Example

For purposes of this appendix, let us start with
representation. Here is an easy-to-understand
Greenfield, MA, the telephone prefixes are 772-
readers: In the USA, local telephone numbers a
conventionally represented in the form ###-#
geographic blocks.) Suppose also that the first
assigned of the three. The suffix portions might
equal distribution. The data set we are interest
telephone numbers currently in active use." On
why this might be interesting for programmatic
specify that herein.

Initially, the data set we are interested in come
representation: a multicolumn report (perhaps
query or compilation process). The first few line

```
=======================================
```

```
772-7628        772-8601        772-0113
773-4319        774-3920        772-0893
773-1134        772-4930        772-9390
[...]
```



B.4 Whitespace Compression

Whitespace compression can be characterized most generally as "removing what we are not interested in." Even though this technique is technically a lossy-compression technique, it is still useful for many types of data representations we find in the real world. For example, even though HTML is far more readable in a text editor if indentation and vertical spacing is added, none of this "whitespace" should make any difference to how the HTML document is rendered by a Web browser. If you happen to know that an HTML document is destined only for a Web browser (or for a robot/spider), then it might be a good idea to take out all the whitespace to make it transmit faster and occupy less space in storage. What we remove in whitespace compression never really had any functional purpose to start with.

In the case of our example in this article, it is possible to remove quite a bit from the described report. The row of "=" across the top adds nothing functional, nor do the "-" within numbers, nor the spaces between them. These are all useful for a person reading the original report, but do not matter once we think of it as data. What we remove is not precisely whitespace in traditional terms, but the intent is the same.

Whitespace compression is extremely "cheap" to perform. It is just a matter of reading a stream of data and excluding a few specific values from the output stream. In many cases, no "decompression" step is involved at all. But even where we would wish to re-create something close to the original somewhere down the data stream, it should require little in terms of CPU or memory. What we reproduce may or may not be exactly what we started with, depending on just what rules and constraints were involved in the original. An HTML page typed by a human in a text editor will probably have spacing that is idiosyncratic. Then again, automated tools often produce "reasonable" indentation and spacing of HTML. In the case of the rigid report format in our example, there is no reason that the original representation could not be precisely produced by a "decompressing formatter" down the data stream.

B.5 Run-Length Encoding

Run-length encoding (RLE) is the simplest widely used lossless-compression technique. Like whitespace compression, it is "cheap"especially to decode. The idea behind it is that many data representations consist largely of strings of repeated bytes. Our example report is one such data representation. It begins with a string of repeated "=", and has strings of spaces scattered through it. Rather than represent each character with its own byte, RLE will (sometimes or always) have an iteration count followed by the character to be repeated.

If repeated bytes are predominant within the expected data representation, it might be adequate and efficient to always have the algorithm specify one or more bytes of iteration count, followed by one character. However, if one-length character strings occur, these strings will require two (or more) bytes to encode them; that is, 00000001 01011000 might be the output bit stream required for just one ASCII "X" of the input stream. Then again, a hundred "X" in a row would be output as 01100100 01011000, which is quite good.

What is frequently done in RLE variants is to selectively use bytes to indicate iterator counts and otherwise just have bytes represent themselves. At least one byte-value has to be reserved to do this, but that can be escaped in the output, if needed. For example, in our example telephone-number report, we know that everything in the input stream is plain ASCII characters. Specifically, they all have bit one of their ASCII value as 0. We could use this first ASCII bit to indicate that an iterator count was being represented rather than representing a regular character. The next seven bits of the iterator byte could be used for the iterator count, and the next byte could represent the character to be repeated. So, for example, we could represent the string "YXXXXXXXX" as:

```
"Y"       Iter(8)  "X"
01001111  10001000 01011000
```

This example does not show how to escape iterator byte-values, nor does it allow iteration of more than 127 occurrences of a character. Variations on RLE deal with issues such as these, if needed.

B.6 Huffman Encoding

Huffman encoding looks at the symbol table of
whole data set. The compression is achieved by
finding the "weights" of each symbol in the dat
set. Some symbols occur more frequently than
others, so Huffman encoding suggests that the
frequent symbols need not be encoded using as
many bits as the less-frequent symbols. There
are variations on Huffman-style encoding, but t
original (and frequent) variation involves lookin
for the most common symbol and encoding it
using just one bit, say 1. If you encounter a 0,
you know you're on the way to encoding a long
variable length symbol.

Let's imagine we apply Huffman encoding to ou
local phone-book example (assume we have
already whitespace-compressed the report). W

might get:

```
Encoding    Symbol
 1            7
 010          2
 011          3
 00000        4
 00001        5
 00010        6
 00011        8
 00100        9
 00101        0
 00111        1
```

Our initial symbol set of digits could already be straightforwardly encoded (with no-compressio as 4-bit sequences (nibbles). The Huffman encoding given will use up to 5-bits for the wor case symbols, which is obviously worse than th nibble encoding. However, our best case will us only *1* bit, and we know that our best case is al the most frequent case, by having scanned the data set. So we might encode a particular phor number like:

```
772 7628 --> 1 1 010 1 00010 010 000
```

The nibble encoding would take 28-bits to represent a phone number; in this particular case, our encoding takes 19-bits. I introduced spaces into the example above for clarity; you can see that they are not necessary to unpack the encoding, since the encoding table will determine whether we have reached the end of an encoded symbol (but you have to keep track of your place in the bits).

Huffman encoding is still fairly cheap to decode cycle-wise. But it requires a table lookup, so it cannot be quite as cheap as RLE, however. The encoding side of Huffman is fairly expensive, though; the whole data set has to be scanned and a frequency table built up. In some cases a "shortcut" is appropriate with Huffman coding. Standard Huffman coding applies to a particula data set being encoded, with the set-specific symbol table prepended to the output data stream. However, if the whole type of data encodednot just the single data sethas the sam regularities, we can opt for a global Huffman table. If we have such a global Huffman table, can hard-code the lookups into our executables which makes both compression and decompression quite a bit cheaper (except for t initial global sampling and hard-coding). For example, if we know our data set would be English-language prose, letter-frequency tables are well known and quite consistent across data sets.

B.7 Lempel Ziv-Compression

Probably the most significant lossless-compress
explained here is LZ78, but LZ77 and other var
LZ78 is to encode a streaming byte sequence u
compressing a bit stream, the LZ table is filled
blank slots. Various size tables are used, but fo
number example above, let's suppose that we u
example, although much too small for most oth
ten slots with our alphabet (digits). As new byt
that grabs the longest sequence possible, then
sequence. In the worst case, we are using 5-bi
we'll wind up getting to use 5-bits for multiple s
machine might do this (a table slot is noted wit

```
7 --> Lookup: 7 found       --> nothi
7 --> Lookup: 77 not found --> add '
2 --> Lookup: 72 not found --> add '
```

```
7 --> Lookup: 27 not found --> add '
6 --> Lookup: 76 not found --> add '
2 --> Lookup: 62 not found --> add '
8 --> Lookup: 28 not found --> add '
```

So far, we've got nothing out of it, but let's con

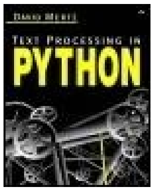
```
7 --> Lookup: 87 not found  --> add
7 --> Lookup: 77 found      --> noth
2 --> Lookup: 772 not found --> add
8 --> Lookup: 28 found      --> noth
6 --> Lookup: 286 not found --> add
...
```

The steps should suffice to see the pattern. We
but notice that we've already managed to use s
symbols with one output in each case. We've al
772 in slot 18, which would prove useful later i

What LZ78 does is fill up one symbol table with
it, and start a new one. In this regard, 32 entri
since that will get cleared before a lot of reuse
symbol table is easy to illustrate.

In typical data sets, Lempel-Ziv variants achiev
Huffman or RLE. On the other hand, Lempel-Zi
use large tables in memory. Most real-life comp

of Lempel-Ziv and Huffman techniques.



B.8 Solving the Right Problem

Just as choosing the right algorithm can often create orders-of-magnitude improvements over even heavily optimized wrong algorithms, choosing the right data representation is often even more important than compression methods (which are always a sort of post hoc optimization of desired features). The simple data set example used in this appendix is a perfect case where reconceptualizing the problem would actually be a much better approach than using *any* of the compression techniques illustrated.

Think again about what our data

represents. It is not a very general collection of data, and the rigid a priori constraints allow us to reformulate our whole problem. What we have is a maximum of 30,000 telephone numbers (7720000 through 7749999), some of which are active, and others of which are not. We do not have a "duty," as it were, to produce a full representation of each telephone number that is active, but simply to indicate the binary fact that it *is* active. Thinking of the problem this way, we can simply allocate 30,000 bits of memory and storage, and have each bit say "yes" or "no" to the presence of one telephone number. The ordering of the bits in the bit-array can be simple ascending order from the lowest to the highest telephone number in the range.

This bit-array solution is the best in almost every respect. It allocates exactly 3750 bytes to represent the data set; the various compression techniques will use a varying amount of storage depending both on the number of telephone numbers in the set and the efficiency of the compression. But if 10,000 of the 30,000 possible telephone numbers are active, and even a very efficient compression technique requires several bytes per telephone number, then the bit-array is an order-of-magnitude better. In terms of CPU demands, the bit-array is not only better than any of the discussed compression methods, it is also quite likely to be better than the naive noncompression method of listing all the numbers as strings. Stepping through a bit-array and incrementing a "current-telephone-number" counter can be done quite efficiently and mostly within the on-chip cache of a modern CPU.

The lesson to be learned from this very simple example is certainly not that every problem has some magic shortcut (like this one does). A lot of problems genuinely require significant memory, bandwidth, storage, and CPU resources, and in many of those cases compression techniques can help easeor shiftthose burdens. But a more moderate lesson could be suggested: Before compression techniques are employed, it is a good idea to make sure that one's starting conceptualization of the data representation is a good one.

B.9 A Custom Text Compressor

Most styles of compression require a decompre
useful with a source document. Many (de)comp
only the needed bytes of a compressed or deco
formats even insert recovery or bookkeeping by
documents (rather than from the very beginnin
compressed documents or strings look like plai
Nonetheless, even streaming decompressors re
plaintext content of a compressed document.

An excellent example of a streaming (de)compr
Although not entirely transparent, you can com
explicit call to a (de)compression function using
like interface, but it is also easy to operate on a
cStringIO.StringIO(). For example:

```
>>> from gzip import GzipFile
>>> from cStringIO import StringIO
```

```
>>> sio = StringIO()
>>> writer = GzipFile(None, 'wb', 9,
>>> writer.write('Mary had a little
>>> writer.write('its fleece as whit
>>> writer.close()
>>> sio.getvalue()[:20]
'\x1f\x8b\x08\x00k\xc1\x9c<\x02\xff'
>>> reader = GzipFile(None, 'rb', 9,
>>> reader.read()[:20]
'Mary had a little la'
>>> reader.seek(30)
>>> reader.read()
'ece as white as snow\n'
```

One thing this example shows is that the under
gibberish. Although the file-like API hides the d
decompression process is also stateful in its de
sequence in the compressed text. You cannot e
middle of the compressed text without a knowl

A different approach to compression can have s
language textual sources. A group of researche
for "word-based Huffman compression." The ge
whole words as the symbol set for a Huffman ta
natural languages, a limited number of (various
frequency, and savings result if such words are

general, such reduced representation is commo
based Huffman takes the additional step of reta
mapping, as with other Huffman variants).

A special quality of word-based Huffman compr
decompression to be searched. This quality ma
compressed form, without incurring the require
useful. Instead, if one is searching for words di
merely precompress the search terms, then use
can be either against an in-memory string or ag
against a precompressed target will be *faster* th
one would use snippets similar to:

```
small_text = word_Huffman_compress(b
search_term = "Foobar"
coded_term = word_Huffman_compress(s
offset = small_text.find(coded_term)
coded_context = small_text[offset-10
plain_context = word_Huffman_expand(
```

A sophisticated implementation of word-based
compression sizes than does *zlib*. For simplicity
compression to the goal of clarity and brevity o
a number of features.

The presented module *word-huffman* uses a fix
symbol table. This number of bytes can be sele

to a generous 2 million entries). The module al
from the actual compression/decompression. T
various documents get encoded using the same
based on a set of canonical documents. In this
symbol table generation can happen just once,
transmitted along with each compressed docum
treating the document being processed current
(thereby somewhat improving compression).

In the algorithm utilized by *word-huffman*, only
The lower 128 ASCII characters represent them
sequence that is not in the symbol table is repr
would not benefit from encoding. Any high-bit
are escaped by being preceded by an OxFF byt
using two bytes; this technique is clearly only u
binary files. Moreover, only character values 0x
always signals a literal high-bit character in the

The *word_huffman* algorithm is not entirely sta
in a compressed text can be expanded without
required. Any low-bit character always literally
might be either an escaped literal, a first byte
symbol table entry. In the worst case, where a
look back two bytes from an arbitrary position
Normally, only one byte lookback is necessary.
separated from each other in the uncompresse
whitespace), so parsing compressed entries is

word_huffman.py

```
wordchars = '-_ABCDEFGHIJKLMNOPQRSTU

def normalize_text(txt):
    "Convert non-word characters to
    trans = [' '] * 256
    for c in wordchars: trans[ord(c)
    return txt.translate('' .join(tr

def build_histogram(txt, hist={}):
    "Incrementally build a histogram
    for word in txt.split():
        hist[word] = hist.get(word,
    return hist

def optimal_Nbyte(hist, entrylen=2):
    "Build optimal word list for non
    slots = 127**entrylen
    words = []
    for word, count in hist.items():
        gain = count * (len(word)-en
        if gain > 0: words.append((g
    words.sort()
```

```
    words.reverse()
    return [w[1] for w in words[:slc

def tables_from_words(words):
    "Create symbol tables for compre
    # Determine ACTUAL best symbol t
    if len(words) < 128: entrylen = 
    elif len(words) <= 16129: entryl
    else: entrylen = 3 # assume < ~2
    comp_table = {}
    # Escape hibit characters
    for hibit_char in map(chr, range
        comp_table[hibit_char] = chr
    # Literal low-bit characters
    for lowbit_char in map(chr, rang
        comp_table[lowbit_char] = lo
    # Add word entries
    for word, index in zip(words, ra
        comp_table[word] = symbol(in
    # Reverse dictionary for expansi
    exp_table = {}
    for key, val in comp_table.items
        exp_table[val] = key
    return (comp_table, exp_table, e
```

```
def symbol(index, entrylen):
    "Determine actual symbol from wo
    if entrylen == 1:
        return chr(128+index)
    if entrylen == 2:
        byte1, byte2 = divmod(index,
        return chr(128+byte1)+chr(12
    if entrylen == 3:
        byte1, rem = divmod(index, 1
        byte2, byte3 = divmod(rem, 1
        return chr(128+byte1)+chr(12
    raise ValueError, "symbol byte l

def word_Huffman_compress(text, comp
    "Compress text based on word-to-
    comp_text = []
    maybe_entry = []
    for c in text+chr(0):   # force
        if c in wordchars:
            maybe_entry.append(c)
        else:
            word = ''.join(maybe_ent
            comp_text.append(comp_ta
            maybe_entry = []
```

```
            comp_text.append(comp_ta
    return ''.join(comp_text[:-1])

def word_Huffman_expand(text, exp_ta
    "Expand text based on symbol-to-
    exp_text = []
    offset = 0
    end = len(text)
    while offset < end:
        c = text[offset]
        if ord(c) == 255:   # escape
            exp_text.append(text[off
            offset += 2
        elif ord(c) >= 128: # symbol
            symbol = text[offset:off
            exp_text.append(exp_tabl
            offset += entrylen
        else:
            exp_text.append(c)
            offset += 1
    return ''.join(exp_text)

def Huffman_find(pat, comp_text, com
    "Find a (plaintext) substring in
    comp_pat = word_Huffman_compress
```

```
    return comp_text.find(comp_pat)

if __name__=='__main__':
    import sys, glob
    big_text = []
    for fpat in sys.argv[1:]:
        for fname in glob.glob(fpat)
            big_text.append(open(fna
    big_text = ''.join(big_text)
    hist = build_histogram(normalize
    for entrylen in (1, 2, 3):
        comp_words = optimal_Nbyte(h
        comp_table, exp_table, entry
        comp_text = word_Huffman_com
        exp_text = word_Huffman_expa
        print "Nominal/actual symbol
              (entrylen, entrylen_,
        print "Compression ratio: %i
              ((100*len(comp_text))/
        if big_text == exp_text:
            print "*** Compression/e
        else:
            print "*** Failure in co
        # Just for fun, here's a sea
        pos = Huffman_find('Foobar',
```

The *word_huffman* module, while simple and fa
the basis for a fleshed-out variant. The compre
comparatively modest 5060 percent of the size
given that locality of decompression of subsegr
nearly no disadvantage to this transformation f
quicker basically in direct proportion to the leng

One likely improvement would be to add run-le
of nonalpha characters); doing so would lose n
algorithm is designed around, and in typical ele
significant additional compression. Moreover, a
transformation is that transformed documents
based techniques (i.e., cumulatively). In other
documents with *word-huffman* if you intend to
tools.

More aggressive improvements might be obtair
table entries and/or by claiming some additiona
(and escaping literals in the original text). You
results might vary somewhat depending upon t
texts.

Search capabilities might also be generalizedbu
In the referenced research article below, the au
expression searching against word-based Huffn
implementation allows certain straightforward t
literal words occur within them) for searching a

caveats and restrictions apply. Overcoming mos
Python's underlying regular expression engine,

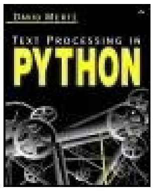
B.10 References

A good place to turn for additional theoretical a
practical information on compression is at the
<comp.compression> FAQ:

<http://www.faqs.org/faqs/compression-faq/

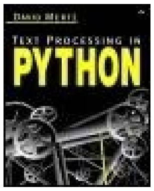
A research article on word-based Huffman enco
inspired my simple example of word-based com
The article "Fast and Flexible Word Searching o
Compressed Text," by Edleno Silva de Moura, G
Navarro, Nivio Ziviani, and Ricardo Baeza-Yates
found at:

<http://citeseer.nj.nec.com/silvademoura00fa



Text Processing in PythonBy David Mertz

Appendix C. Understanding Unicode



Appendix C. Understanding Unicode

C.1 Some Background on Characters

Before we see what Unicode is, it makes sense to step back slightly to think about just what it means to store "characters" in digital files. Anyone who uses a tool like a text editor usually just thinks of what they are doing as entering some charactersnumbers, letters, punctuation, and so on. But behind the scene a little bit more is going on. "Characters" that are stored on digital media must be stored as sequences of ones and zeros, and some encoding and decoding must happen to make these ones and zeros into characters we see on a screen or type in with a keyboard.

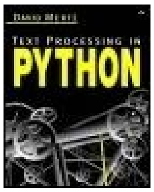
Sometime around the 1960s, a few decisions were made about just what ones and zeros (bits) would represent characters. One important choice that most modern computer users give no thought to was the decision to use 8-bit bytes on nearly all computer platforms. In other words, bytes have 256 possible values. Within these 8-bit bytes, a consensus was reached to represent one character in each byte. So at that point, computers needed a particular *encoding* of characters into byte values; there were 256 "slots" available, but just which character would go in each slot? The most popular encoding developed was Bob Bemers' American Standard Code for Information Interchange (ASCII), which is now specified in exciting standards like ISO-14962-1997 and ANSI-X3.4-1986(R1997). But other options, like IBM's mainframe EBCDIC, linger on, even now.

ASCII itself is of somewhat limited extent. Only the values of the lower-order 7-bits of each byte might contain ASCII-encoded characters. The top 7-bits worth of positions (128 of them) are "reserved" for other uses (back to this). So, for example, a byte that contains "01000001" *might* be an ASCII encoding of the letter "A", but a byte containing "11000001" cannot be an ASCII encoding of anything. Of course, a given byte may or may not *actually* represent a character; if it is part of a text file, it probably does, but if it is part of object code, a compressed archive, or other binary data, ASCII decoding is misleading. It depends on context.

The reserved top 7-bits in common 8-bit bytes have been used for a number of things in a character-encoding context. On traditional textual terminals (and printers, etc.) it has been common to allow switching between *codepages* on terminals to allow display of a variety of national-language characters (and special characters like box-drawing borders), depending on the needs of a user. In the world of Internet communications, something very similar to the codepage system exists with the various ISO-8859-* encodings. What all these systems do is assign a set of characters to the 128 slots that ASCII reserves for other uses. These might be accented Roman characters (used in many Western European languages) or they might be non-Roman character sets like Greek, Cyrillic, Hebrew, or Arabic (or in the future, Thai and Hindi). By using the right codepage, 8-bit bytes can be made quite suitable for encoding reasonable sized (phonetic) alphabets.

Codepages and ISO-8859-* encodings, however, have some definite limitations. For one thing, a terminal can only display one codepage at a given time, and a document with an ISO-8859-* encoding can only contain one character set. Documents that need to contain text in multiple languages are not possible to represent by these encodings. A second issue is equally important: Many ideographic and pictographic character sets have far more than 128 or 256 characters in them (the former is all we would have in the codepage system, the latter if we used the whole byte and discarded the ASCII part). It is simply not possible to encode languages like Chinese, Japanese, and Korean in 8-bit bytes. Systems like ISO-

2022-JP-1 and codepage 943 allow larger character sets to be represented using two or more bytes for each character. But even when using these language-specific multibyte encodings, the problem of mixing languages is still present.



C.2 What Is Unicode?

Unicode solves the problems of previous character-encoding schemes by providing a unique code number for *every* character needed, worldwide and across languages. Over time, more characters are being added, but the allocation of available ranges for future uses has already been planned out, so room exists for new characters. In Unicode-encoded documents, no ambiguity exists about how a given character should display (for example, should byte value 0x89 appear as e-umlaut, as in codepage 850, or as the per-mil mark, as in codepage 1004?). Furthermore, by giving each character its own code, there is no problem or ambiguity in creating multilingual documents that utilize multiple character sets at the same time. Or rather, these documents actually utilize the single (very large) character set of Unicode itself.

Unicode is managed by the Unicode Consortium (see Resources), a nonprofit group with corporate, institutional, and individual members. Originally, Unicode was planned as a 16-bit specification. However, this original plan failed to leave enough room for national variations on related (but distinct) ideographs across East Asian languages (Chinese, Japanese, and Korean), nor for specialized alphabets used in mathematics and the scholarship of historical languages.

As a result, the code space of Unicode is currently 32-bits (and anticipated to remain fairly sparsely populated, given the 4 billion allowed characters).

Appendix C. Understanding Unicode

C.3 Encodings

A full 32-bits of encoding space leaves plenty o
want to represent, but it has its own problems.
character we want to encode, that makes for ra
streams). Furthermore, these verbose files are
for legacy tools. As a solution to this, Unicode i
Transformation Formats" (abbreviated as UTF-*
use rather clever techniques to encode characte
with the most common situation being the use
the encoding name. In addition, the use of spe
characters is designed in such a way as to be fr
an available encoding, one that simply uses all

The design of UTF-8 is such that US-ASCII char
themselves. For example, the English letter "e"
both ASCII and in UTF-8. However, the non-Eng
Unicode character OxOOEB, is encoded with the
the UTF-16 representation of every character is

sometimes 4 bytes). UTF-16 has the rather stra
letters "e" and "e-umlaut" as 0x65 0x00 and 0x
the odd value for the e-umlaut in UTF-8 come f
encoded UTF-8 character is allowed to be in the
confusion. So the UTF-8 scheme uses some bit
character using up to 6 bytes. But the byte valu
arranged in such a manner as not to allow conf
you read a file nonsequentially).

Let's look at another example, just to see it laid
encoded in several ways. The view presented is
hex-mode file viewer. This way, it is easy to see
representation (on a legacy, non-Unicode termi
underlying hexadecimal values each byte conta

Hex view of several character string encoding

```
------------------- Encoding = us-as
55 6E 69 63 6F 64 65 20 20 20 20 20
------------------- Encoding = utf-8
55 6E 69 63 6F 64 65 20 20 20 20 20
------------------- Encoding = utf-1
FF FE 55 00 6E 00 69 00 63 00 6F 00
```

C.4 Declarations

We have seen how Unicode characters are actu
applications know to use a particular decoding
How applications are alerted to a Unicode enco
stream in question.

Normal text files do not have any special heade
explicitly specify type. However, some operatin
BeOSWindows and Linux only in a more limited
extended attributes to files; increasingly, MIME
extended attributes. If this happens to be the c
information such as:

```
Content-Type: text/plain; charset=UT
```

Nonetheless, having MIME headers attached to
Fortunately, the actual byte sequences in Unico
Unicode-aware application, absent contrary ind

given file is encoded with UTF-8. A non-Unicode
will find a file that contains a mixture of ASCII
multibyte UTF-8 encodings). All the ASCII-rang
they were ASCII encoded. If any multibyte UTF
appear as non-ASCII bytes and should be treat
application. This may result in nonprocessing o
pretty much the best we could expect from a le
does not know how to deal with the extended c

For UTF-16 encoded files, a special convention
file. One of the sequences 0xFF 0xFE or 0xFE 0
choice of which header specifies the endianness
platforms are little-endian and will use 0xFF 0x
of a legacy file beginning with these bytes was
as a reliable indicator for UTF-16 encoding. Wit
ASCII characters will appear every other byte,
course, extended characters will produce non-r
(4 byte) representations. But a legacy tool that
doing the right thing with UTF-16 encoded files

Many communications protocolsand more recer
explicit encoding specification. For example, an
server) can return a header such as the followi
client:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset:UTF
```

Similarly, an NNTP, SMTP/POP3 message can ca
that makes explicit the encoding to follow (mos
text/html, however; or at least we can hope).

HTML and XML documents can contain tags and
explicit. An HTML document can provide a hint

```
<META HTTP-EQUIV="Content-Type" CONT
```

However, a META tag should properly take lowe
situation where both are part of the communica
HTTP header does not exist).

In XML, the actual document declaration should

```
<?xml version="1.0" encoding="UTF-8"
```

Other formats and protocols may provide explic
means.

C.5 Finding Codepoints

Each Unicode character is identified by a unique
character codepoints on official Unicode Web sit
of characters is by generating an HTML page wi
below does this:

mk_unicode_chart.py

```
# Create an HTML chart of Unicode ch
import sys
head = '<html><head><title>Unicode C
       '<META HTTP-EQUIV="Content-Ty
             'CONTENT="text/html; ch
       '</head><body>\n<h1>Unicode C
foot = '</body></html>'
```

```
fp = sys.stdout
fp.write(head)
num_blocks = 32 # Up to 256 in theor
for block in range(0,256*num_blocks,
    fp.write('\n\n<h2>Range %5d-%5d<
    start = unichr(block).encode('ut
    fp.write('\n<pre>     ')
    for col in range(16): fp.write(s
    fp.write('</pre>')
    for offset in range(0,256,16):
        fp.write('\n<pre>')
        fp.write('+'+str(offset).rju
        line = '  '.join([unichr(n+b
        fp.write(line.encode('UTF-8'
        fp.write('</pre>')
fp.write(foot)
fp.close()
```

Exactly what you see when looking at the gene
browser and OS platform the page is viewed or
factors. Generally, any character that cannot be
appear as some sort of square, dot, or questior
generally accurate. Once a character is visually
generated with the *unicodedata* module:

```
>>> import unicodedata
```

```
>>> unicodedata.name(unichr(1488))
'HEBREW LETTER ALEF'
>>> unicodedata.category(unichr(148
'Lo'
>>> unicodedata.bidirectional(unichr
'R'
```

A variant here would be to include the informat
generated HTML chart, although such a listing
example above.

C.6 Resources

More-or-less definitive information on all matte
be found at:

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

The Unicode Consortium:

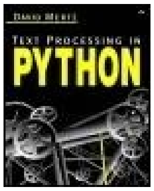
<http://www.unicode.org/unicode/consortium

Unicode Technical Report #17Character Encodi

<http://www.unicode.org/unicode/reports/tr1

A brief history of ASCII:

<http://www.bobbemer.com/ASCII.HTM>



Text Processing in Python By David Mertz

Table of Contents

Appendix D. A State Machine for Ad

This book was written entirely in plaintext edito
ASCII." In spirit and appearance, smart ASCII r
developed on email and Usenet. In fact, I have
number of years to produce articles, tutorials, a
additional conventions in the earlier smart ASC
that made almost all the individual typographic
toolchain only came to exist through many hou
by other developers.

The printed version of this book used tools I wr
frontmatter, and endmatter, and then to add LaTeX
number of custom LaTeX macros are included in th
people lets me convert LaTeX source into the PDF
printed copies.

For information on the smart ASCII format, see
book, chiefly in Chapter 4. You may also downl
site at <http://gnosis.cx/TPiP/>, along with a s
used. Readers might also be interested in a for
in spirit, but both somewhat "heavier" and mor
semiofficial status in the Python community sin
for information see:

<http://docutils.sourceforge.net/rst.html>

In this appendix, I include the full source code
text of this book into an HTML document. I beli
demonstration of the design and structure of a
structure, book2html.py uses a line-oriented st
appropriate document elements. Under this app
part, determined by the context of the lines tha
decisions on how to categorize each line with a
collection of regular expression patterns, the bl
HTML output. In principle, it would not be diffic
steps involved are modular.

The Web site for this book has a collection of ut
I have adapted the skeleton to deal with variati
overlap between all of them. Using this utility is

```
% book2html.py "Text Processing in P
```

The title is optional, and you may pipe STDIN a
HTML, I decided it would be nice to colorize sou
support module:

colorize.py

```
#!/usr/bin/python
import keyword, token, tokenize, sys
from cStringIO import StringIO
```

```
PLAIN = '%s'
BOLD  = '<b>%s</b>'
CBOLD = '<font color="%s"><b>%s</b><
_KEYWORD = token.NT_OFFSET+1
_TEXT    = token.NT_OFFSET+2
COLORS   = { token.NUMBER:       'blac
             token.OP:           'dark
             token.STRING:       'gree
             tokenize.COMMENT: 'dark
             token.NAME:         None,
             token.ERRORTOKEN: 'red'
             _KEYWORD:           'blue
             _TEXT:              'blac

class ParsePython:
    "Colorize python source"
    def __init__(self, raw):
        self.inp  = StringIO(raw.exp
    def toHTML(self):
        "Parse and send the colored
        raw = self.inp.getvalue()
        self.out = StringIO()
        self.lines = [0,0]      # st
        self.lines += [i+1 for i in
```

```
        self.lines += [len(raw)]
        self.pos = 0
        try:
            tokenize.tokenize(self.i
            return self.out.getvalue
        except tokenize.TokenError,
            msg,ln = ex [0],ex [1] [
            sys.stderr.write("ERROR:
                             (msg, r
            return raw
    def __call__(self,toktype,toktex
        "Token handler"
        # calculate new positions
        oldpos = self.pos
        newpos = self.lines[srow] +
        self.pos = newpos + len(tokt
        if toktype in [token.NEWLINE
            self.out.write('\n')
            return
        if newpos > oldpos:     # se
            self.out.write(self.inp.
        if toktype in [token.INDENT,
            self.pos = newpos   # sk
            return
        if token.LPAR <= toktype anc
```

```
            toktype = token.OP  # ma
        elif toktype == token.NAME a
            toktype = _KEYWORD
        color = COLORS.get(toktype,
        if toktext:             # se
            txt = Detag(toktext)
            if color is None:    txt
            elif color=='black': txt
            else:                txt
            self.out.write(txt)

Detag = lambda s: \
    s.replace('&','&amp;').replace('

if __name__=='__main__':
    parsed = ParsePython(sys.stdin.r
    print '<pre>'
    print parsed.toHTML()
    print '</pre>'
```

The module *colorize* contains its own self-test c
own. The main module consists of:

book2html.py

```
#!/usr/bin/python
"""Convert ASCII book source files f

Usage: python book2html.py [title] <
"""
__author__=["David Mertz (mertz@gnos
__version__="November 2002"

from __future__ import generators
import sys, re, string, time
from colorize import ParsePython
from cgi import escape

#-- Define some HTML boilerplate
html_open =\
"""<!DOCTYPE HTML PUBLIC "-//IETF//I
<html>
<head>
<title>%s</title>
<style>
  .code-sample {background-color:#EE
             width:90%%; margin-l
  .module      {color : darkblue}
  .libfunc     {color : darkgreen}
</style>
```

```
</head>
<body>
"""
html_title = "Automatically Generate
html_close = "</body></html>"
code_block = \
"""<table class="code-sample"><tr><t
<tr><td><pre>%s</pre></td></tr>
</table>"""
#-- End of boilerplate

#-- State constants
for s in ("BLANK CHAPTER SECTION SUE
          "MODNAME PYSHELL CODESAMP
          "SUBBODY TERM DEF RULE VEF
    exec "%s = '%s'" % (s,s)
markup = {CHAPTER:'h1', SECTION:'h2'
          BODY:'p', QUOTE:'blockquot
          DEF:'blockquote'}
divs = {RULE:'hr', VERTSPC:'br'}

class Regexen:
    def __init__(self):
        # blank line is empty, space
        self.blank    = re.compile("
```

```
        self.chapter  = re.compile('
        self.section  = re.compile('
        self.subsect  = re.compile('
        self.subsub   = re.compile('
        self.modline  = re.compile('
        self.pyshell  = re.compile('
        self.codesamp = re.compile('
        self.numlist  = re.compile('
        self.body     = re.compile('
        self.quote    = re.compile('
        self.subbody  = re.compile('
        self.rule     = re.compile('
        self.vertspc  = re.compile('

def Make_Blocks(fpin=sys.stdin, r=Re
    #-- Initialize the globals
    global state, blocks, laststate
    state, laststate = BLANK, BLANK
    blocks = [[BLANK]]
    #-- Break the file into relevant
    for line in fpin.xreadlines():
        line = line.rstrip()
        #-- for "one-line states" ju
        if r.blank.match(line):
            if inState(PYSHELL):
```

```
    else:
elif r.rule.match(line):
elif r.vertspc.match(line):
elif r.chapter.match(line):
elif r.section.match(line):
elif r.subsect.match(line):
elif r.subsub.match(line):
elif r.modline.match(line):
elif r.numlist.match(line):
elif r.pyshell.match(line):
    if not inState(PYSHELL):
elif r.codesamp.match(line):
#-- now the multi-line state
elif r.body.match(line):
    if not inState(BODY):
elif r.quote.match(line):
    if inState(MODLINE):
    elif r.blank.match(line)
    elif not inState(QUOTE):
#-- now the "multi-line stat
elif inState(MODLINE, PYSHEI
    "stay in this state unti
    "...or other one-line pr
elif r.subbody.match(line):
    "Sub-body is tricky: it
```

```
            "PYSHELL, CODESAMP, NUMI
            if inState(BODY):
            elif inState(BLANK):
                if laststate==DEF:
            elif inState(DEF, CODESA
                pass
        else:
            raise ValueError, \
                  "unexpected input
        if inState(MODLINE, RULE, VE
        elif r.blank.match(line): pa
        else: blocks[-1].append(line
    return LookBack(blocks)

def LookBack(blocks):
    types = [f [0] for f in blocks]
    for i in range(len(types)-1):
        this, next = types[i:i+2]
        if (this,next)==(BODY,DEF):
            blocks[i][0] = TERM
    return blocks

def newState(name):
    global state, laststate, blocks
    if name not in (BLANK, MODLINE):
```

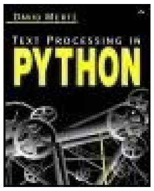
```
        blocks.append([name])
    laststate = state
    state = name

def instate(*names) :
    return state in names

def Process_Blocks(blocks, fpout=sys
    fpout.write(html_open % title)
    for block in blocks:        # Ma
        typ, lines = block[0], block
        tag = markup.get(typ, None)
        div = divs.get(typ, None)
        if tag is not None:
            map(fpout.write, wrap_ht
        elif div is not None:
            fpout.write('<%s />\n' %
        elif typ in (PYSHELL, CODESA
            fpout.write(fixcode('\n'
        elif typ in (MODNAME,):
            mod = '<hr/><h3 class="m
            fpout.write(mod)
        elif typ in (TERM,):
            terms = '<br />\n'.join(
            fpout.write('<h4 class="
```

```
        else:
            sys.stderr.write(typ+'\r
    fpout.write(html_close)

#-- Functions for start of block-typ
def wrap_html(lines, tag):
    txt = '\n'.join(lines)
    for para in txt.split('\n\n'):
        if para: yield '<%s>%s</%s>\
                        (tag,URLify(

def fixcode(block, style=CODESAMP):
    block = LeftMargin(block)
    # Pull out title if available
    title = 'Code Sample'
    if style==CODESAMP:
        re_title = re.compile('^#\*?
        if_title = re_title.match(bl
        if if_title:
            title = if_title.group(1
            block = re_title.sub('',
    # Decide if it is Python code
    firstline = block[:block.find('\
    if re.search(r'\.py_?|[Pp]ython|
        # Has .py, py_, Python/pytho
```

```
        block = ParsePython(block.rs
        return code_block % (Typogra
    # elif the-will-and-the-way-is-t
    else:
        return code_block % (Typogra

def LeftMargin(txt):
    "Remove as many leading spaces a
    for l in range(12,-1,-1):
        re_lead = '(?sm)'+' '*l+'\S'
        if re.match(re_lead, txt): b
    txt = re.sub('(?sm)^'+' '*l, '',
    return txt

def URLify(txt):
    # Conv special IMG URL's: Alt Te
    # (don't actually try quite as h
    txt = re.sub('(?sm){(.*?):\s*(ht
                 '<img src="\\2" alt
    # Convert regular URL's
    txt = re.sub('(?:[^="])((?:http|
                 '<a href="\\1">\\1<
    return txt

def Typography(txt):
```

```
    rc = re.compile      # cut down l
    MS = re.M | re.S
    # [module] names
    r = rc(r"""([\(\s'/">]|^)\[(.*?)
    txt = r.sub('\\1<i class="module
    # *strongly emphasize* words
    r = rc(r"""([\(\s'/"]|^)\*(.*?)\
    txt = r.sub('\\1<strong>\\2</str
    # -emphasize- words
    r = rc(r"""([\(\s'/"]|^)-(.+?)-(
    txt = r.sub('\\1<em>\\2</em>\\3'
    # _Book Title_ citations
    r = rc(r"""([\(\s'/"]|^)_(.*?)_(
    txt = r.sub('\\1<cite>\\2</cite>
    # 'Function()' names
    r = rc(r"""([\(\s/"]|^)'(.*?)'([
    txt = r.sub("\\1<code>\\2</code>
    # 'library. func() ' names
    r = rc(r"""([\(\s/"]|^)'(.*?)'([
    txt = r.sub('\\1<i clas    s="li
    return txt

if __name__ == '__main__':
    blocks = Make_Blocks()
    if len(sys.argv) > 1:
```

```
    Process_Blocks(blocks, title=sys
else:
    Process_Blocks(blocks)
```



Text Processing in PythonBy David Mertz

Appendix E. Glossary

Asymmetrical Encryption

Encryption using a pair of keysthe first encry
decrypts. In the most common protocol, the
but the encryption key may be widely reveal
publish your encryptionor "public"key, which
that only you can decrypt. The person who f
course, has initial access to it, but any third-
"private"key cannot access the message. Se
of cryptographic capabilities.

Big-O Notation, Complexity

Big-O notation is a way of describing the gov
of an algorithm. Often such complexity is de
an expression on "n" following in parenthese
letter or a special typeface for the "O". The '
"order" of complexity.

The insight behind big-O notation is that ma
calculation time that can be expressed as a
data set or domain at issue. For the most im
constant startup times and even speed mult
underlying complexity. For example, suppos

that takes 100 seconds to initialize some dat
seconds to perform the main calculation. If y
runtime will be 260 seconds; saving that 10(
seem worthwhile, if possible. However, if you
objects, you are looking at 1,100 seconds in
minor component. Moreover, you might thinl
(N^2) seconds to only 2*(N^2) secondssay,
programming language. Once you consider t
to calculate for N=100, even the multiplier is
particular if you had a better algorithm that
seconds (bigger multiplier), you would be a
50,000 seconds.

In noting complexity orders, constants and r
omitted, leaving only the dominant factor. C

O(1)	constant
O(log(n))	logarithmic
O((log(n))^c)	polylogarith
O(n)	linear
O(n*log(n))	frequent in
O(n^2)	quadratic
O(n^c)	polynomial
O(c^n)	exponential

Birthday Paradox

The name "birthday paradox" comes from th
peoplethat in a room with just 23 people the
two of them sharing a birthday. A naive hun
365 days, it should instead take something l
likelihood.

In a broader sense the probability of collisio
outcomes are possible, reaches 50 percent w
items are collected. This is a concern when y
selections, and the like to consist of only dis

Cryptographic Hash

A hash with a strong enough noncollision pr
produce a false message yielding the same h
message. See Section 2.2.4 for a discussion

Cyclic Redundancy Check (CRC32)

Based on mod 2 polynomial operations, CRC
"fingerprint" of a set of data.

See also **[Hash]**

Digital Signatures

A means of proving the authenticity of a me
encryption, digital signatures involve two ke
secret, but a published validation key can be
the signing key used it to authenticate a me
discussion of cryptographic capabilities.

Hash

A short value that is used as a "fingerprint"
should be unlikely that two data sets will yie
can be used to check for data errors, by com
hash value (mismatch suggests data error).
noncollision properties to be used cryptogra

Idempotent Function

The property that applying a function to its

value. That is, if and only if *F* is idempotent
In a nod to Chaos Theory, we can observe th
finite repetitions of composition with F is ide
attractorthat is, if G is idempotent for G=lan
interesting fact is completely unnecessary to
book.

Immutable

Literally, "cannot be changed." Some data co
and strings, in Pythonconsist of a set of item
change over the life of the object. In contras
dictionaries can continue to be the same obj
membership. Since you generally access obj
index positions), it is sometimes easy to con
be used at different times to point to differe
objects. For example, a pattern with tuples l

```
>>> tup = (1,2,3)
>>> id(tup)
248684
>>> tup = tup+(4,5,6)
>>> tup
(1, 2, 3, 4, 5, 6)
```

```
>>> id(tup)
912076
```

Even though the name tup is re-bound durir
bound object changes. Moreover, creating a
later produces the same identity:

```
>>> tup2 = (1,2,3)
>>> id(tup2)
248684
```

Immutable objects are particularly useful as
continue to hash the same way over prograr
a stricter constraint than immutabilityit is ne
an immutable object itself be (recursively) ir
hashable.

Mutable

Literally, "can be changed." Data collection c
and arrays from the *array* module are mutal
stays the same, even as their membership c
(usually) suitable as dictionary keys, howeve
used to hold *records* of a data collection, wh
fields within a record. The insight underlying

record contained different field data, it woul
individual self-identical records can be adde
collection, depending on outside events and

Public-key Encryption

See [**Assymmetrical Encryption**]

Referential Transparency

The property of a function or block construc
same value every time it is called with the s
functions are referentially transparent, by de
results depend on global state, external con
referentially opaque.

Shared-key Encryption

See [**Symmetrical Encryption**]

Structured Text Database

A text file that is used to encode multiple re
composed of the same fields. Records and fi
and columns, respectively. A structured text

format that contains little or no explicit mark
are delimited files and fixed-width files, both
and elsewhere. Most of the time, structured
oriented, with one conceptual record per line
indentation are used to indicate dependent s

Symmetrical Encryption

Encryption using a single "key" that must be
Section 2.2.4 for a discussion of cryptograph
