Other Modules

Overview

This chapter describes a number of less common modules. Some are useful, others are quite obscure, and some are just plain obsolete.

The pyclbr module

This module contains a basic Python class parser.

In 1.5.2, the module exports a single function, **readmodule**, which parses a given module, and returns a list of all classes defined at the module's top level.

Example: Using the pyclbr module

```
# File:pyclbr-example-1.py

import pyclbr

mod = pyclbr.readmodule("cgi")

for k, v in mod.items():
    print k, v
```

```
MiniFieldStorage <pyclbr.Class instance at 7873b0>
InterpFormContentDict <pyclbr.Class instance at 79bd00>
FieldStorage <pyclbr.Class instance at 790e20>
SvFormContentDict <pyclbr.Class instance at 79b5e0>
StringIO <pyclbr.Class instance at 77dd90>
FormContent <pyclbr.Class instance at 79bd60>
FormContentDict <pyclbr.Class instance at 79a9c0>
```

In 2.0 and later, there's also an alternative interface, **readmodule_ex**, which returns global functions as well.

Example: Using the pyclbr module to read classes and functions

```
# File:pyclbr-example-3.py

import pyclbr

# available in Python 2.0 and later
mod = pyclbr.readmodule_ex("cgi")

for k, v in mod.items():
    print k, v
```

```
MiniFieldStorage <pyclbr.Class instance at 00905D2C>
parse_header <pyclbr.Function instance at 00905BD4>
test <pyclbr.Function instance at 00906FBC>
print_environ_usage <pyclbr.Function instance at 00907C94>
parse_multipart <pyclbr.Function instance at 00905294>
FormContentDict <pyclbr.Class instance at 008D3494>
initlog <pyclbr.Function instance at 00904AAC>
parse <pyclbr.Function instance at 00904EFC>
StringIO <pyclbr.Class instance at 00903EAC>
SvFormContentDict <pyclbr.Class instance at 00906824>
...
```

To get more information about each class, use the various attributes in the **Class** instances:

Example: Using the pyclbr module

```
# File:pyclbr-example-2.py

import pyclbr
import string

mod = pyclbr.readmodule("cgi")

def dump(c):
    # print class header
    s = "class " + c.name
    if c.super:
        s = s +  "(" + string.join(map(lambda v: v.name, c.super), ", ") + ")"
    print s + ":"
    # print method names, sorted by line number
    methods = c.methods.items()
    methods.sort(lambda a, b: cmp(a[1], b[1]))
    for method, lineno in methods:
        print "  def " + method
    print

for k, v in mod.items():
    dump(v)
```

```
class MiniFieldStorage:
  def __init__
  def __repr__

class InterpFormContentDict(SvFormContentDict):
  def __getitem__
  def values
  def items

...
```

The filecmp module

(New in 2.0) This module contains functions to compare files and directories.

Example: Using the filecmp module

```
# File:filecmp-example-1.py

import filecmp

if filecmp.cmp("samples/sample.au", "samples/sample.wav"):
    print "files are identical"
else:
    print "files differ!"
```

```
files differ!
```

In 1.5.2 and earlier, you can use the **cmp** and **dircmp** modules instead.

The cmd module

This module provides a simple framework for command-line interfaces (CLI). This is used by the **pdb** debugger module, but you can also use it for your own programs.

To implement your own command-line interface, subclass the **Cmd** class, and define **do** and **help** methods. The base class automatically turns all **do** methods into commands, and uses the **help** methods to show help information.

Example: Using the cmd module

```
# File:cmd-example-1.py

import cmd
import string, sys

class CLI(cmd.Cmd):

    def __init__(self):
        cmd.Cmd.__init__(self)
        self.prompt = '> '

    def do_hello(self, arg):
        print "hello again", arg, "!"

    def help_hello(self):
        print "syntax: hello [message]",
        print "-- prints a hello message"

    def do_quit(self, arg):
        sys.exit(1)

    def help_quit(self):
        print "syntax: quit",
        print "-- terminates the application"

    # shortcuts
    do_q = do_quit

#
# try it out

cli = CLI()
cli.cmdloop()
```

```
> help

Documented commands (type help <topic>):
========================================
hello          quit

Undocumented commands:
======================
help           q

> hello world
hello again world !
> q
```

The rexec module

This module provides versions of **exec**, **eval**, and **import** which executes code in a restricted execution environment. In this environment, functions that can damage resources on the local machine are no longer available.

Example: Using the rexec module

```
# File:rexec-example-1.py

import rexec

r = rexec.RExec()
print r.r_eval("1+2+3")
print r.r_eval("__import__('os').remove('file')")
```

```
6
Traceback (innermost last):
  File "rexec-example-1.py", line 5, in ?
    print r.r_eval("__import__('os').remove('file')")
  File "/usr/local/lib/python1.5/rexec.py", line 257, in r_eval
    return eval(code, m.__dict__)
  File "<string>", line 0, in ?
AttributeError: remove
```

The Bastion module

This module allows you to control how a given object is used. It can be used to pass objects from unrestricted parts of your application to code running in restricted mode.

To create a restricted instance, simply call the **Bastion** wrapper. By default, all instance variables are hidden, as well as all methods that start with an underscore.

Example: Using the Bastion module

```
# File:bastion-example-1.py

import Bastion

class Sample:
    value = 0

    def _set(self, value):
        self.value = value

    def setvalue(self, value):
        if 10 < value <= 20:
            self._set(value)
        else:
            raise ValueError, "illegal value"

    def getvalue(self):
        return self.value

#
# try it

s = Sample()
s._set(100) # cheat
print s.getvalue()

s = Bastion.Bastion(Sample())
s._set(100) # attempt to cheat
print s.getvalue()
```

```
100
Traceback (innermost last):
...
AttributeError: _set
```

You can control which functions to publish. In the following example, the internal method can be called from outside, but the **getvalue** no longer works:

Example: Using the Bastion module with a non-standard filter

```
# File:bastion-example-2.py

import Bastion

class Sample:
    value = 0

    def _set(self, value):
        self.value = value

    def setvalue(self, value):
        if 10 < value <= 20:
            self._set(value)
        else:
            raise ValueError, "illegal value"

    def getvalue(self):
        return self.value

#
# try it

def is_public(name):
    return name[:3] != "get"

s = Bastion.Bastion(Sample(), is_public)
s._set(100) # this works
print s.getvalue() # but not this
```

```
100
Traceback (innermost last):
...
AttributeError: getvalue
```

The readline module

(Optional). This module activates input editing on Unix, using the GNU readline library (or compatible).

Once imported, this module provides improved command line editing, and also command history. It also enhances the **input** and **raw_input** functions.

Example: Using the readline module

```
# File:readline-example-1.py

import readline # activate readline editing
```

The rlcompleter module

(Optional, Unix only). This module provides word completion for the **readline** module.

To enable word completion, just import this module. By default, the completion function is bound to the **ESCAPE** key. Press **ESCAPE** twice to finish the current word. To change the key, you can use something like:

```
import readline
readline.parse_and_bind("tab: complete")
```

The following script shows how to use the completion functions from within a program.

Example: Using the rlcompleter module to expand names

```
# File:rlcompleter-example-1.py

import rlcompleter
import sys

completer = rlcompleter.Completer()

for phrase in "co", "sys.p", "is":
    print phrase, "=>",
    # emulate readline completion handler
    try:
        for index in xrange(sys.maxint):
            term = completer.complete(phrase, index)
            if term is None:
                break
            print term,
    except:
        pass
    print
```

```
co => continue compile complex coerce completer
sys.p => sys.path sys.platform sys.prefix
is => is isinstance issubclass
```

The statvfs module

This module contains a number of constants and test functions that can be used with the optional **os.statvfs** function, which returns information about a file system. This is usually only available on Unix systems.

Example: Using the statvfs module

```
# File:statvfs-example-1.py

import statvfs
import os

st = os.statvfs(".")

print "preferred block size", "=>", st[statvfs.F_BSIZE]
print "fundamental block size", "=>", st[statvfs.F_FRSIZE]
print "total blocks", "=>", st[statvfs.F_BLOCKS]
print "total free blocks", "=>", st[statvfs.F_BFREE]
print "available blocks", "=>", st[statvfs.F_BAVAIL]
print "total file nodes", "=>", st[statvfs.F_FILES]
print "total free nodes", "=>", st[statvfs.F_FFREE]
print "available nodes", "=>", st[statvfs.F_FAVAIL]
print "max file name length", "=>", st[statvfs.F_NAMEMAX]
```

```
preferred block size => 8192
fundamental block size => 1024
total blocks => 749443
total free blocks => 110442
available blocks => 35497
total file nodes => 92158
total free nodes => 68164
available nodes => 68164
max file name length => 255
```

The calendar module

This is a Python reimplementation of the Unix **cal** command. It simply prints the calendar for any given month or year to standard output.

prmonth(year, month) prints the calendar for a given month.

Example: Using the calender module

```
# File:calendar-example-1.py

import calendar
calendar.prmonth(1999, 12)
```

```
    December 1999
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 31
```

prcal(year) prints the calendar for a given year.

Example: Using the calender module

```
# File:calendar-example-2.py

import calendar
calendar.prcal(2000)
```

```
                                  2000

      January               February              March
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2      1  2  3  4  5  6      1  2  3  4  5
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   6  7  8  9 10 11 12
10 11 12 13 14 15 16  14 15 16 17 18 19 20  13 14 15 16 17 18 19
17 18 19 20 21 22 23  21 22 23 24 25 26 27  20 21 22 23 24 25 26
24 25 26 27 28 29 30  28 29                 27 28 29 30 31
31

       April                 May                  June
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2   1  2  3  4  5  6  7            1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30
31

       July                 August              September
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2      1  2  3  4  5  6               1  2  3
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   4  5  6  7  8  9 10
10 11 12 13 14 15 16  14 15 16 17 18 19 20  11 12 13 14 15 16 17
17 18 19 20 21 22 23  21 22 23 24 25 26 27  18 19 20 21 22 23 24
24 25 26 27 28 29 30  28 29 30 31           25 26 27 28 29 30
31

      October              November             December
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                   1      1  2  3  4  5               1  2  3
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   4  5  6  7  8  9 10
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  11 12 13 14 15 16 17
16 17 18 19 20 21 22  20 21 22 23 24 25 26  18 19 20 21 22 23 24
23 24 25 26 27 28 29  27 28 29 30           25 26 27 28 29 30 31
30 31
```

Note that the calendars are printed using European conventions; in other words, Monday is the first day of the week.

This module contains a number of support functions which can be useful if you want to output calendars in other formats. It's probably easiest to copy the entire file, and tweak it to suit your needs.

The sched module

This is a simple event scheduler for non-threaded environments.

Example: Using the sched module

```
# File:sched-example-1.py

import sched
import time, sys

scheduler = sched.scheduler(time.time, time.sleep)

# add a few operations to the queue
scheduler.enter(0.5, 100, sys.stdout.write, ("one\n",))
scheduler.enter(1.0, 300, sys.stdout.write, ("three\n",))
scheduler.enter(1.0, 200, sys.stdout.write, ("two\n",))

scheduler.run()
```

```
one
two
three
```

The statcache module

This module contains a function which returns information about files. It's an extension of the **os.stat** function, in that it keeps a cache with recently collected information.

Example: Using the statcache module

```
# File:statcache-example-1.py

import statcache
import os, stat, time

now = time.time()
for i in range(1000):
    st = os.stat("samples/sample.txt")
print "os.stat", "=>", time.time() - now

now = time.time()
for i in range(1000):
    st = statcache.stat("samples/sample.txt")
print "statcache.stat", "=>", time.time() - now

print "mode", "=>", oct(stat.S_IMODE(st[stat.ST_MODE]))
print "size", "=>", st[stat.ST_SIZE]
print "last modified", "=>", time.ctime(st[stat.ST_MTIME])
```

```
os.stat => 0.371000051498
statcache.stat => 0.0199999809265
mode => 0666
size => 305
last modified => Sun Oct 10 18:39:37 1999
```

The grep module

This module provides different ways to search for text in text files.

Example: Using the grep module

```
# File:grep-example-1.py

import grep
import glob

grep.grep("\<rather\>", glob.glob("samples/*.txt"))

# 4: indentation, rather than delimiters, might become
```

The dircache module

(Obsolete). This module contains a function to get a list of files in a directory. It's an extension of the **os.listdir** function, in that it keeps a cache to avoid rereading directory that hasn't been modified.

Example: Using the dircache module

```
# File:dircache-example-1.py

import dircache

import os, time

#
# test cached version

t0 = time.clock()

for i in range(100):
    dircache.listdir(os.sep)

print "cached", time.clock() - t0

#
# test standard version

t0 = time.clock()

for i in range(100):
    os.listdir(os.sep)

print "standard", time.clock() - t0
```

```
cached 0.0664509964968
standard 0.5560845807
```

The dircmp module

(Obsolete, only in 1.5.2). This module provides a class that can be used to compare the contents of two disk directories.

Example: Using the dircmp module

```
# File:dircmp-example-1.py

import dircmp

d = dircmp.dircmp()
d.new("samples", "oldsamples")
d.run()
d.report()
```

```
diff samples oldsamples
Only in samples : ['sample.aiff', 'sample.au', 'sample.wav']
Identical files : ['sample.gif', 'sample.gz', 'sample.jpg', ...]
```

In Python 2.0 and later, this module has been replaced by the **filecmp** module.

The cmp module

(Obsolete, only in 1.5.2). This module contains a function to compare two files.

Example: Using the cmp module

```
# File:cmp-example-1.py

import cmp

if cmp.cmp("samples/sample.au", "samples/sample.wav"):
    print "files are identical"
else:
    print "files differ!"
```

```
files differ!
```

In Python 2.0 and later, this module has been replaced by the **filecmp** module.

The cmpcache module

(Obsolete, only in 1.5.2). This module contains a function to compare two files. It's an extension of the **cmp** module, in that it keeps a cache over recently made comparisons.

Example: Using the cmpcache module

```
# File:cmpcache-example-1.py

import cmpcache

if cmpcache.cmp("samples/sample.au", "samples/sample.wav"):
    print "files are identical"
else:
    print "files differ!"
```

```
files differ!
```

In Python 2.0 and later, this module has been replaced by the **filecmp** module.

The util module

(Obsolete, 1.5.2 only). This module is included for backwards compatibility only. New code should use the replacement constructs shown in the examples below.

remove(sequence, item) removes the given item, if found in the sequence.

Example: Emulating the util module's remove function

```
# File:util-example-1.py

def remove(sequence, item):
    if item in sequence:
        sequence.remove(item)
```

readfile(filename) => string reads the contents of a text file as a single string.

Example: Emulating the util module's readfile function

```
# File:util-example-2.py

def readfile(filename):
    file = open(filename, "r")
    return file.read()
```

readopenfile(file) => string returns the contents of an open file (or other file object).

Example: Emulating the util module's readopenfile function

```
# File:util-example-3.py

def readopenfile(file):
    return file.read()
```

The soundex module

(Optional, 1.5.2 only). This module implements a simple hash algorithm, which converts words to 6-character strings based on their English pronunciation.

As of version 2.0, this module is no longer included.

get_soundex(word) => string returns the soundex string for the given word. Words that sound similar should give the same soundex string.

sound_similar(word1, word2) => flag returns true if the two words has the same soundex.

Example: Using the soundex module

```
# File:soundex-example-1.py

import soundex

a = "fredrik"
b = "friedrich"

print soundex.get_soundex(a), soundex.get_soundex(b)
print soundex.sound_similar(a, b)
```

```
F63620 F63620
1
```

The timing module

(Obsolete, Unix-only). This module can be used to time the execution of a Python program.

Example: Using the timing module

```
# File:timing-example-1.py

import timing
import time

def procedure():
    time.sleep(1.234)

timing.start()
procedure()
timing.finish()

print "seconds:", timing.seconds()
print "milliseconds:", timing.milli()
print "microseconds:", timing.micro()
```

```
seconds: 1
milliseconds: 1239
microseconds: 1239999
```

The following script shows how you can emulate this module using functions in the standard **time** module.

Example: Emulating the timing module

```
# File:timing-example-2.py

import time

t0 = t1 = 0

def start():
    global t0
    t0 = time.time()

def finish():
    global t1
    t1 = time.time()

def seconds():
    return int(t1 - t0)

def milli():
    return int((t1 - t0) * 1000)

def micro():
    return int((t1 - t0) * 1000000)
```

You can use **time.clock()** instead of **time.time()** to get CPU time, where supported.

The posixfile module

(Obsolete, Unix only). This module provides a file-like object with support for file locking. New programs should use the **fcntl** module instead.

Example: Using the posixfile module

```
# File:posixfile-example-1.py

import posixfile
import string

filename = "counter.txt"

try:
    # open for update
    file = posixfile.open(filename, "r+")
    counter = int(file.read(6)) + 1
except IOError:
    # create it
    file = posixfile.open(filename, "w")
    counter = 0

file.lock("w|", 6)

file.seek(0) # rewind
file.write("%06d" % counter)

file.close() # releases lock
```

The bisect module

This module provides functions to insert items in sorted sequences.

insort(sequence, item) inserts item into the sequence, keeping it sorted. The sequence can be any mutable sequence object that implements **__getitem__** and **insert**.

Example: Using the bisect module to insert items in an ordered list

```
# File:bisect-example-1.py

import bisect

list = [10, 20, 30]

bisect.insort(list, 25)
bisect.insort(list, 15)

print list
```

```
[10, 15, 20, 25, 30]
```

bisect(sequence, item) => index returns the index where the item should be inserted. The sequence is not modified.

Example: Using the bisect module to find insertion points

```
# File:bisect-example-2.py

import bisect

list = [10, 20, 30]

print list
print bisect.bisect(list, 25)
print bisect.bisect(list, 15)
```

```
[10, 20, 30]
2
1
```

The knee module

This is a Python re-implementation of the package import mechanism that was introduced in Python 1.5. Since this is already supported by the standard interpreter, this module is mostly provided to show how things are done in there. It does work, though. Just import the module to enable it.

Example: Using the knee module

```
# File:knee-example-1.py

import knee

# that's all, folks!
```

The tzparse module

(Obsolete). This (highly incomplete) module contains a parser for timezone specifications. When you import this module, it parses the content of the TZ environment variable.

Example: Using the tzparse module

```
# File:tzparse-example-1.py

import os
if not os.environ.has_key("TZ"):
    # set it to something...
    os.environ["TZ"] = "EST+5EDT;100/2,300/2"

# importing this module will parse the TZ variable
import tzparse

print "tzparams", "=>", tzparse.tzparams
print "timezone", "=>", tzparse.timezone
print "altzone", "=>", tzparse.altzone
print "daylight", "=>", tzparse.daylight
print "tzname", "=>", tzparse.tzname
```

```
tzparams => ('EST', 5, 'EDT', 100, 2, 300, 2)
timezone => 18000
altzone => 14400
daylight => 1
tzname => ('EST', 'EDT')
```

In addition to the variables shown in this example, this module contains a number of time manipulation functions that use the defined time zone.

The regex module

(Obsolete) This is the old (pre-1.5) regular expression machinery. New code should use **re** where possible.

Note that **regex** is usually faster than the **re** module used in Python 1.5.2, but slower than the new version used in 1.6 and later.

Example: Using the regex module

```
# File:regex-example-1.py

import regex

text = "Man's crisis of identity in the latter half of the 20th century"

p = regex.compile("latter") # literal
print p.match(text)
print p.search(text), repr(p.group(0))

p = regex.compile("[0-9]+") # number
print p.search(text), repr(p.group(0))

p = regex.compile("\<\w\w\>") # two-letter word
print p.search(text), repr(p.group(0))

p = regex.compile("\w+$") # word at the end
print p.search(text), repr(p.group(0))
```

```
-1
32 'latter'
51 '20'
13 'of'
56 'century'
```

The regsub module

(Obsolete). This module provides string replacements, based on regular expressions. New code should use the **re** module's **replace** function instead.

Example: Using the regsub module

```
# File:regsub-example-1.py

import regsub

text = "Well, there's spam, egg, sausage and spam."

print regsub.sub("spam", "ham", text) # just the first
print regsub.gsub("spam", "bacon", text) # all of them
```

```
Well, there's ham, egg, sausage and spam.
Well, there's bacon, egg, sausage and bacon.
```

The reconvert module

(Obsolete). This module converts old-style regular expressions, as used by the **regex** module to the new style used by the **re** module. It can also be used as a command line tool.

Example: Using the reconvert module

```
# File:reconvert-example-1.py

import reconvert

for pattern in "abcd", "a\(b*c\)d", "\<\w+\>":
    print pattern, "=>", reconvert.convert(pattern)
```

```
abcd => abcd
a\(b*c\)d => a(b*c)d
\<\w+\> => \b\w+\b
```

The regex_syntax module

(Obsolete). This module contains a bunch of flags that can be used to change the behavior of the **regex** regular expression module.

Example: Using the regex_syntax module

```
# File:regex-syntax-example-1.py

import regex_syntax
import regex

def compile(pattern, syntax):
    syntax = regex.set_syntax(syntax)
    try:
        pattern = regex.compile(pattern)
    finally:
        # restore original syntax
        regex.set_syntax(syntax)
    return pattern

def compile_awk(pattern):
    return compile(pattern, regex_syntax.RE_SYNTAX_AWK)

def compile_grep(pattern):
    return compile(pattern, regex_syntax.RE_SYNTAX_GREP)

def compile_emacs(pattern):
    return compile(pattern, regex_syntax.RE_SYNTAX_EMACS)
```

The find module

(Obsolete, 1.5.2 only). The **find** module provides a single function, with the same name as the module:

find(pattern, directory) -> list scans a given directory and all its subdirectories for files matching a given pattern.

For more information on the pattern syntax, see the **fnmatch** module.

Example: Using the find module

```
# File:find-example-1.py

import find

# find all JPEG files in or beneath the current directory
for file in find.find("*.jpg", "."):
    print file
```

```
.\samples\sample.jpg
```