python™ 2 programming

The Python Standard Library

The Python Standard Library

- Contents

The Standard Library

- **Represents a large resource of code**
  - Core modules
  - Standard modules
  - Threads and Processes
  - Data Representation
  - File Formats
  - Mail and News Message Processing
  - Network protocols
  - Internationalization
  - Multimedia modules
  - Data Storage
  - Tools and Utilities
  - Platform-specific Modules
  - Many others….

Batteries included!

How many modules have we seen so far?

module	Chapter
abc	11
builtins	1
collections	5
copy	9,11
cProfile	10
distutils	10
doctest	10
fileinput	7
glob	3,5,7,9,10
gzip	7
multiprocessing	13
os	3,9,13

module	Chapter
pdb	10
pickle	7
pstats	10
re	4,6,7,8
shelve	7
sqlite3	7
subprocess	13
sys	1,3,7,8,10,12
time	8,12,13
threading	13
warnings	12

- **It is unusual to have a Python program which does not use at least one standard library module**

Pretty Printer - a useful utility

- **Standard library module `pprint`**
  - **Can be used on any Python structure**
  - **Dictionaries are output sorted by key**
  - **Some control over formatting, for example line width**

```
import pprint

myd={'UK':['London',('Wigan','Macclesfield','Bolton')],
     'US':['Washington',('Springfield','New York','Boston')],
     'FR':['Paris', ('Lyon', 'Bordeaux', 'Toulouse')]}

pprint.pprint(myd)
```

```
{'FR': ['Paris', ('Lyon', 'Bordeaux', 'Toulouse')],
 'UK': ['London', ('Wigan', 'Macclesfield', 'Bolton')],
 'US': ['Washington', ('Springfield', 'New York', 'Boston')]}
```

Operating System interfaces - os and friends

- **os - Operating System**
  - **The idea was that all os specific routines would go here**
  - **Many of the functions are based on UNIX C equivalents**
    - Process parameters (environment variables, uid, pid, etc.)
    - File descriptor level operations (open, fsync, lseek, etc.)
    - File and directory operations (mkfifo, listdir, remove/unlink,etc.)
    - Process management (abort, fork/exec, kill, etc.)
    - System information (OS type, path separator, etc.)

- **Other related modules**

`os.path`	**Filename processing**
`fileinput`	**Building UNIX style filter programs**
`tempfile`	**Creating temporary files and directories**
`shutil`	**Copying, deleting, and moving groups of files**

os.open example

- Offers operating specific features
  - For low-level tasks
  - File descriptor based on UNIX, file handle on Windows

```
import os
fd = os.open (filename, flags [, mode ])
bytes = os.read (fd, n)
os.write (fd, bytes)
os.close (fd)
```

UNIX style permissions

```
import os

fd = os.open ('a file', os.O_CREAT|os.O_WRONLY, 0o640)
buffer = 'This is some text\r\nanother line\r\n'
bytes = os.write (fd, buffer)
os.close (fd)
print bytes,"bytes written"
```

System specific attributes - sys

- **Most interfaces are portable**
  - **Information about the operating system**
    - Operating system version
    - Byte order
    - Character and floating point formats
  - **Information about the python interpreter**
    - Version information
    - Lists of builtins
    - Module load path
  - **Information about your program**
    - Tracing and Exception information
    - Reference counts
    - Streams - stdin, stdout, stderr

Signal handling - signal

- Not all features are portable
  - Signals are part of UNIX architecture, not Windows
- Signal handling is similar to other languages
  - The signals are defined with a `SIG` prefix
  - `SIGPIPE` is ignored by default, `SIGINT` generates an exception
  - Supports `alarm` on UNIX only
- Three possible actions:
  - `SIG_DFL` Take default action
  - `SIG_IGN` Ignore the signal
  - Create a signal handler
- Also supports interval timers
  - `setitimer` and `getitimer`: similar to standard UNIX

Converting a signal to an exception

```
import signal
import time

class MyError(Exception):
    pass

def handler(sig, frame):
    raise MyError('Received signal ' + str(sig) +
                  ' on line ' + str(frame.f_lineno) +
                  ' in ' + frame.f_code.co_filename)

signal.signal(signal.SIGINT, handler)

try:
    while 1:
        time.sleep(1)
except KeyboardInterrupt:
    print 'Keyboard interrupt caught'
except MyError as err:
    print "Hiccup:",err

print 'Clean exit'
```

User written signal handler

Hit CTRL+C here

Configuration files

- **Similar to old Windows .INI files**
  - **Used by many applications**
  - **Consist of [sections]**
    - Containing options and comments

```
opt=value
opt: value
opt2=%(opt)stext
```

"classic" syntax

```
; comment
# comment
```

```
; Alchemy 3ds Global UI Settings
[Alchemy3dsExporter]

;************************************************
;       Viewer controls group
;
;  Refresh the embedded viewport Alchemy viewer
Export_PC = 1
;  export the display to psx2
```

The ConfigParser module

```
from ConfigParser import *
config = ConfigParser()

config.add_section('GLOBALS')
config.set ('GLOBALS', 'TRACE', True)
config.add_section('FILENAMES')
config.set ('FILENAMES', 'DIR','C:\\myapp')
config.set ('FILENAMES', 'MASTER','%(dir)s\\master.qa')
config.set ('FILENAMES', 'SLAVE','%(dir)s\\slave.qa')

fh = open("config.ini", "w")
config.write(fh)
fh.close()
```

config.ini →

```
[GLOBALS]
trace = True

[FILENAMES]
slave = %(dir)s\slave.qa
master = %(dir)s\master.qa
dir = C:\myapp
```

```
config.read('config.ini')
print config.get ('FILENAMES','master', 0)
print config.getboolean('GLOBALS', 'TRACE')
```

```
C:\myapp\master.qa
True
```

The datetime module and friends

- **Date and time manipulation functions in datetime**
  - **datetime objects - for extracting date/time in different formats**
  - **timedelta objects - for calculating date/time differences**
  - **date, time, and tzinfo (time zone) objects**
  - **strftime - well known date/time formatting function**
    - Methods available for date, datetime, and time objects

```
print date.today().strftime("%A %d %B %Y")
```

```
Sunday 05 April 2009
```

- **Also related:**
  - **calendar module**
    - Includes calendar iterator
  - **time module**
    - Supports mktime(), localtime(), gmtime(), strftime(), sleep() etc.

datetime example

- **Calculate someone's age in years**

```
import sys
from datetime import *
from calendar import *

sBirth = raw_input("Enter birthday (dd/mm/yyyy):")
try:
    (day, month, year) = sBirth.split('/')
    dBirth = date(int(year), int(month), int(day))
except ValueError:
    print >> sys.stderr,"Invalid date:",sBirth
    exit(1)

dToday = date.today()
diff = dToday - dBirth

diff = diff.days - leapdays(int(year),dToday.year)
years = diff // 365
print "Client is",years,"years old"
```

Number of leap years between years (calendar)

The platform module

- **Used for identifying the platform we are running on**
  - **Mostly the operating system and the Java or C runtime library**

```
import sys
import platform

print sys.platform
print "Platform:",platform.platform()
print "Compiler:",platform.python_compiler()
print "Python  :",platform.python_version()
print "LibC :",platform.libc_ver()
```

```
linux2
Platform: Linux-2.6.24-22-generic-i686-with-debian-lenny-sid
Compiler: GCC 4.2.4 (Ubuntu 4.2.4-1ubuntu3)
Python  : 2.6.1
LibC : ('glibc', '2.4')
```

```
win32
Platform: Windows-XP-5.1.2600-SP3
Compiler: MSC v.1500 32 bit (Intel)
Python  : 2.6.2
LibC : ('', '')
```

External function interface - ctypes

- **Enables run-time dynamic linking to foreign libraries**
  - **DLLs on Windows, shared objects on UNIX/Linux**
    - Main interface is through cdll
  - **Windows interface includes:**
    - windll - stdcall calling convention interface
    - oledll - for HRESULT return codes (and stdcall)

```
from ctypes import *

msvcrt = cdll.msvcrt
text = "Hollow World!\n"
msvcrt.printf("%s", text)

mydll = cdll.LoadLibrary("C:\QA\Win32Dlls\DllModule7")
mydll.DllFunc7()
```

C RTL call
Use libc.so on UNIX/Linux

Windows _cdecl DLL call

Win32 ctypes example

- **Most base APIs are in kernel32.dll**
- **Many Microsoft specific types are already defined**

```
from ctypes import *
from ctypes.wintypes import *

kernel = windll.kernel32


Startup  = STARTUPINFO(0)
ProcInfo = PROCESS_INFORMATION(0)

retn = kernel.CreateProcessA(None, "myprog.exe",
                             None, None,
                             False, 0, None, None,
                             byref(Startup),
                             byref(ProcInfo))

print WinError()
```

Define Structures here (see notes)

The socket module

- Part of the standard library
  - Based on BSD 4.3
  - Supported by most operating systems
  - Highly portable
- Supports IPv4 and IPv6
  - Some methods only work on IPv4
- If you have used sockets in C:
  - The principle, and many of the functions, is the same
  - The Python socket interface is *much* easier to use
  - There are a few Python specific methods to abstract operations
- Python also supports secure sockets layer - `ssl`

Socket server example

- **Connection oriented stream socket (TCP/IP)**

```
from socket import *
nPortID = 600

sock = socket(AF_INET,SOCK_STREAM,0)
sock.bind(("",nPortID) )
sock.listen(5)

(newsock,addr) = sock.accept()
sock.close()

while True:
    bMessage = newsock.recv(1024, 0)
    print "Recieved: ", bMessage

newsock.close();
```

Hardcoded port number

Create a socket
Bind to local address
Set max. pending requests

Wait for client to connect
Original socket no longer required in this case

Wait for data from client

Close connected socket

__future__

- **A pseudo module for enabling new language features**
  - **Includes all future features from previous releases**
    - Even when it is implemented in the current release
    - Version 3 __future__ includes all the 2.6 "futures"

```
import __future__

print __future__.all_feature_names
```

```
['nested_scopes','generators','division','absolute_import',
'with_statement','print_function','unicode_literals']
```

```
print __future__.absolute_import
```

```
_Feature((2, 5, 0, 'alpha', 1), (2, 7, 0, 'alpha', 0), 16384)
```

optional release mandatory release compiler flag

Other modules

- **There are many other in the Standard Library**
  - **http://docs.python.org/library/index.html/**
  - **See also http://www.doughellmann.com/PyMOTW**
  - **In book form:** The Python Standard Library By Example
- **Other modules are available**
  - **For example, Win32 interfaces in win32-py on sourceforge**
- **PyPi - Python Package Index**
  - **http://pypi.python.org/pypi**
  - **Also known as "The cheese shop"**
  - **Over 10,000 packages available for Python 2**
- **Easy Install**
  - **http://peak.telecommunity.com/DevCenter/EasyInstall**

Summary

- The Standard Library is always available
  - Batteries included
  - os supplies interfaces to the operating system
  - sys gives information about the Python environment
  - ConfigParser enables access to .INI style configuration files
  - datetime, calendar, and time modules give comprehensive date and time functions
  - platform supplies detailed information about the platform
  - __future__ enables new language features for testing
- There are many others in the Standard Library, and elsewhere

Example - converting Python 2 scripts to Py3

```
import sys
import os
import glob
from subprocess import *

if len(sys.argv) > 1:
    dir = sys.argv[1]
else:
    dir = '.'

script = os.path.join(sys.prefix,'Tools','Scripts','2to3.py')

procs = []
pattern = os.path.join(dir, '*.py')
for name in glob.iglob(pattern):
    procs.append(Popen([sys.executable, script, '-w', name]))

while len(procs) > 0:
    proc = procs.pop(0)
    proc.wait()
```

Typical Python program

Note how many standard library routines are used