Advance Python Programming

Mr. Nay Linn Than
Myanmar Computer Professionals Association (MCPA)
Sixth AOSS workshop, 7-11 Mar 2010 @ UniSIM Singapore

Overview

- Goal is to highlight many of Python's capabilities.
- Generally speaking, most programmers don't have trouble picking up Python
- Advanced Programming in Python
  - String processing
  - Working with the files systems, file system objects
  - Database Interface
  - Operation system interfaces
  - Programming with Threads
  - Network programming

Python Library

- Python is packaged with a large library of standard modules
  - String processing
  - Operating systems interfaces
  - Networking
  - Threads
  - GUI
  - Database
  - Language services, (Jython, Cython)
  - Security
- And there are many third party modules
  - XML
  - Numeric Processing
  - Plotting/Graphics
  - Etc.

String Processing

Various string processing functions

```
string.atof(s)              # Convert to float
string.atoi(s)              # Convert to integer
string.atol(s)              # Convert to long
string.count(s,pattern)     # Count occurrences of pattern in s
string.find(s,pattern)      # Find pattern in s
string.split(s, sep)        # String a string
string.join(strlist, sep)   # Join a list of string
string.replace(s,old,new)   # Replace occurrences of old with new
```

Examples

```
s = "Hello World"
a = string.split(s)                    # a = ['Hello','World']
b = string.replace(s,"Hello","Goodbye")
c = string.join(["foo","bar"],":")   # c = "foo:bar"
```

Regular Expressions

- **Background**
  - Regular expressions are patterns that specify a matching rule.
  - Generally contain a mix of text and special characters

    foo.* # Matches any string starting with foo

    \d* # Match any number decimal digits

    [a-zA-Z]+ # Match a sequence of one or more letters
- **The re module**
  - Provides regular expression pattern matching and replacement.
- **General idea**
  - Regular expressions are specified using syntax described.
  - Compiled into a regular expression "object".
  - This is used to perform matching and replacement operations.

The re Module

- **Example**

```
import re
pat = r'(\d+)\.(\d*)' # My pattern
r = re.compile(pat) # Compile it
m = r.match(s) # See if string s matches
if m:
# Yep, it matched
...
else:
# Nope
```

- **A more complex example**

```
# Replace URL such as http://www.python.org with a hyperlink
pat = r'(http://[\w-]+(\.[\w-]+)*((/[\w-~]*)?))'
r = re.compile(pat)
r.sub('<a href="\\1">\\1</a>',s) # Replace in string
```

Working with the files systems

- File Objects
- Standard Input, Output, and Error
- File and Path Manipulation
- Globbing
- Low-Level File I/O operations
- Low-level File and Directory Manipulation
- The StringIO and cStringIO modules

DBM-Style Databases

- **Python provides a number of DBM-style database interfaces**
  - Key-based databases that store arbitrary strings.
  - Similar to shelve, but can't store arbitrary objects (strings only)

  Examples: dbm, gdbm, bsddb, anydbm (for generic access)
- **Example:**

```
import dbm
d = dbm.open("database","r")
d["foo"] = "bar" # Store a value
s = d["spam"] # Retrieve a value
del d["name"] # Delete a value
d.close() # Close the database
```

- **Comments**
  - The availability of DBM modules depends on optional libraries and may vary.
  - Don't use these if you should really be using a relational database (e.g., MySQL).

Databases (Working with MySQL)

- The beauty of Python is that it can provide abstraction in data access and processing at various levels.
- A Python-MySQL combination is a good option for a data-bound application.
- Python has provided a specification for API for database access.
- This specification is in its second version and is known as DB-API 2.0.
- Any database access module has to conform to the DB-API 2.0 specification.
- Almost all the modules for RDBMSs conform to the specification.

MySQLdb module

You can download it from
http://sourceforge.net/projects/mysql-python/

There are four basic type of process to access database;

1. Connecting with the database
2. Creation of the cursor
3. Execution of the SQL statement
4. Fetching the result set and iterating over the result set

1. Connecting with the database

- A connection to the database must be established.
- Done by using the connect() method.
- Then connect() is the constructor of the MySQLdb.

```
db= MySQLdb.connect(host='Demo', user='root' ,
   passwd='admın', db='test')
```

2. Creation of the cursor

- The cursor is that area in the memory where the data fetched from the data tables are kept once the query is executed.
- MySQL does not support cursors.
- But it is easy to emulate the functionality of cursors by using the MySQLdb .

```
cursor= db.cursor()
```

3. Execution of the SQL statement

- Any SQL statement supported by MySQL can be executed using the execute() method of the Cursor class.
- The SQL statement is passed as a string to it.
- Once the statement is executed successfully, the Cursor object will contain the result set of the retrieved values.

```
cursor.execute("select * from employees")
```

4. Fetching the resultset

- In the real world, fetching all the rows at once may not be feasible.
- MySQLdb answers this situation by providing different versions of the fetch() function of Cursor class.
- **fetchone()**: This fetches one row in the form of a Python tuple. (increments the cursor position by one)
- **fetchall()**: This fetches all the rows as tuple of tuples.

4. Fetching the resultset (cont)

- To fetch one row at a time and display the result

```
#get the count of total rows in the #resultset
numrows = int(cursor.rowcount)
# get and display one row at a time
for x in range(0,numrows):
        row = cursor.fetchone()
        print row[0], "-->", row[1]
```

4. Fetching the resultset (cont)

- By using **fetchall()**

```
result = cursor.fetchall()
# iterate through resultset
for record in result:
        print record[0] , "-->", record[1]
```

- The iteration is through the core Python APIs only.

Operating System Services

- **Python provides a wide variety of operating system interfaces**
  - Basic system calls
  - Operating environment
  - Processes
  - Timers
  - Signal handling
  - Error reporting
  - Users and passwords
- **Implementation**
  - A large portion of this functionality is contained in the os module.
  - The interface is based on POSIX.
  - Not all functions are available on all platforms (especially Windows/Mac).

Windows and Macintosh

- **Comment**
  - Most of Python's OS interfaces are Unix-centric.
  - However, much of this functionality is emulated on non-Unix platforms.
  - With a number of omissions (especially in process and user management).
- **The msvcrt module**
  - Provides access to a number of functions in the Microsoft Visual C++ runtime.
  - Functions to read and write characters.
  - Some additional file handling (locking, modes, etc...).
  - But not a substitute for PythonWin.
- **The macfs, macostools, and findertools modules**
  - Manipulation of files and applications on the Macintosh.

Threads

- **Background**
  - A running program is called a "process"
  - Each process has memory, list of open files, stack, program counter, etc…
  - Normally, a process executes statements in a single sequence of control-flow.
- **Process creation with fork(),system(), popen(), etc…**
  - These commands create an entirely new process.
  - Child process runs independently of the parent.
  - Has own set of resources.
  - There is minimal sharing of information between parent and child.
  - Think about using the Unix shell.
- **Threads**
  - A thread is kind of like a process (it's a sequence of control-flow).
  - Except that it exists entirely inside a process and shares resources.
  - A single process may have multiple threads of execution.
  - Useful when an application wants to perform many concurrent tasks on shared data.
  - Think about a browser (loading pages, animations, etc.)

Python Threads

- **Python supports threads on the following platforms**
  - Solaris
  - Windows
  - Systems that support the POSIX threads library (pthreads)
- **Thread scheduling**
  - Tightly controlled by a global interpreter lock and scheduler.
  - Only a single thread is allowed to be executing in the Python interpreter at once.
  - Thread switching only occurs between the execution of individual byte-codes.
  - Long-running calculations in C/C++ can block execution of all other threads.
  - However, most I/O operations do not block.
- **Comments**
  - Python threads are somewhat more restrictive than in C.
  - Effectiveness may be limited on multiple CPUs (due to interpreter lock).
  - Threads can interact strangely with other Python modules (especially signal handling).
  - Not all extension modules are thread-safe.

Network Programming

- **Python provides a wide assortment of network support**
  - Low-level programming with sockets (if you want to create a protocol).
  - Support for existing network protocols (HTTP, FTP, SMTP, etc…)
  - Web programming (CGI scripting and HTTP servers)
  - Data encoding
- **Recommended Reference**
  - *Unix Network Programming by W. Richard Stevens.*

Network Programming, Socket

- **Python's networking modules primarily support TCP/IP**
  - TCP - A reliable connection-oriented protocol (streams).
  - UDP - An unreliable packet-oriented protocol (datagrams).
  - Of these, TCP is the most common (HTTP, FTP, SMTP, etc...).
- **Both protocols are supported using "sockets"**
  - A socket is a file-like object.
  - Allows data to be sent and received across the network like a file.
  - But it also includes functions to accept and establish connections.
  - Before two machines can establish a connection, both must create a socket object.
- **Socket programming in a nutshell**
  - Server creates a socket, binds it to some well-known port number, and starts listening.
  - Client creates a socket and tries to connect it to the server (through the above port).
  - Server-client exchange some data.
  - Close the connection (of course the server continues to listen for more clients).

Conclusion

- Python is a great language for experimentation.
- Again, most programmers don't have trouble picking up Python
- This is a great way to learn about the various modules
- **For more information:**
  - Python Essential Reference (By David M. Beazley)
  - Online documentation (www.python.org)
- **My References:**
  - O'Reilly Open Source Conference (David M. Beazley)
  - Apress, Beginning Python From Novice to Professional (2005)
  - http://www.devshed.com/c/a/Python/Database-Programming-in-Python-Accessing-MySQL

Thank You!