Beginner's Guide

James Gardner

January 31, 2005

http://www.pythonweb.org
docs at pythonweb.org

CONTENTS

CHAPTER

ONE

Introduction

1.1 Purpose of This Document

The purpose of this document is to give someone who is new to the Web Modules an overview of the design philosophy and software components and to get them up and running with their first scripts with a tutorial.

There is also a full Module Reference and plenty of examples. If you are still stuck after all that please contact the authour for help.

1.2 Documentation Overview

The documentation consists of a number of major sections.

Introduction Contains basic information on the Python Web Modules, sets out the aims and purpose of the modules and gives an overview of the development status and core components of the code.

Tutorials This section is a series of tutorials on web programming. They start off with the absolute basics and work up to a tutorial on an SQL database driven contact form. If you are looking for a quick start to using the Python Web Modules then this is the place to look.

Modules Contains module documentation and examples for each of the modules.

Developer Guide Contains information about how to go about developing code for the Python Web Modules themselves.

1.3 Components

The Python Web Modules contain code written by various authours and released under various licences. Please see the Licence section for full information.

The Python Web Modules provide modules for all the common needs of web developers where components can be used together or independently. It is designed to run on any platform or web server supporitng Python and does not require administrator priveldges to install.

The modules contain a pure Python database engine and webserver so that everything you need to run a web site is available in the download. This means that if you decide not to use existing technologies such as a Apache or MySQL you can still run full web services with only Python and the Python Web Modules.

The suite uses well known design patterns and includes popular modules such as:

auth Easy to use authorisation and user management system with support for multiple access levels, storage drivers, and templates.

datetime Compatibility code providing date and time classes for Python 2.2 users.

database Implementation of the Python Datbase Connectivity specification. SQL abstraction layer, automatic type conversions and multiple result formats allowing tuple, dictionary or attribute access. Currently support for MySQL, SQLite and SnakeSQL.

database.object An object relation mapper built on the `database` module. Allows database objects to be manipulated in Python code without any knowledge of SQL or the storage mechanism. Supports one-to-many and many-to-many mappings automatically. Data structures are defined in code so no precompilation is required. The code also interfaces with the `form` module to provide customisable HTML form views of the data so that the database can be manipulated using a website without significant coding effort.

error Enhanced error handling based on the `cgitb` module. Automatically log errors or email error reports. Provide customiseable error views.

form Construction of persistant forms/wizards for HTML interfaces.

image Create and manipulate graphics including JPG, PNG, PDF, PS.

mail Simple function to send email in HTML or plain text with a variety of options.

session Peristant storage of session data via an SQL database. Automatic cookie handling and full API.

template For the easy display of data as HTML/XML includes Cheetah, a poweful templating module.

util Useful utility functions

wsgi Support for the new Web Server Gateway Interface specification PEP 333.

xml Including DOM, SAX and XSL Transforms

1.4 Development Status

Development of the modules is continuing at a rapid pace. The modules could currently be described as ”beta code”. Although the modules are currently being used in a number of commercial products they have not had enough exposure to the wider community to guarantee that all issues have been ironed out.

1.5 Getting the Latest Version

The latest version of the Python Web Modules is always available at http://www.pythonweb.org/.

1.6 Philosophy

Developing web applications can be difficult. Many programmers have developed complicated systems to massively simplify the process. These systems work very well if your indended purpose closely matches the intention of the authour. The cost of using these systems is the steep learning curve required to understand the system properly and the lack of flexibility if the system doesn’t quite match your intended purpose.

The Web Modules take the opposite approach. By making the really simple functions available easily in a modular form you can quickly build your own system without having to reinvent the wheel or learn a new framework every time you want to write a new system. The Web Modules are desinged to be an extremely flexible toolkit to allow you to program any way you want as easily as possible whilst still giving you the functionality you require to build complex systems quickly.

1.7 Design Principles

The Web Modules are built on a number of popular design patterns. They address the following problems typically encountered in the development of complex sites:

Use Anywhere The Web Modules are designed to work with any Python system. They can be used with Zope and Webware but also can be used on a shared hosting account without needing administrator priveldges. This makes the modules especially useful as they can be used to write CGI scripts on Apache servers or WSGI applications using their own server.

Separating design from logic The Web Modules include a number of templating systems including Cheetah, XYAPTU and others, which allow for a clean divide between the HTML page design and the program data.

Extreme Programming The Web Modules are built on the methadology of Extreme Programming. By writing simple clean modules we avoid unncecesary features and offer a very flexible toolkit for the development of advanced applications.

Form and Field Mangement The Web Modules provide a `form` module to allow the simple display of data through an HTML form. Complex fields can be created with typed return values.

Managing users and their access permissions The Web Modules include an `auth` module which can use md5 encryption to store users' usernames and passwords in a database. It also offers a very simple interface for user management and permision checking as well as a persistance mechanism.

Database independence The web modules come with a database abstraction layer supporting ODBC, MySQL and SQLite. The database layer can be used to program database independant SQL

Object Relation Mapper The Modules come with an advanced ORM called `object` which makes database programming simple with no knowledge of SQL. It supports a query builder, one-to-many and many-to-many mappings and integrates with the `form` module to provide an HTML interface to the data structures.

1.8 Alternatives

If you don't want to program in Python you should have a good look at PHP and the PEAR modules distributed with PHP 4.

If you do want to program in Python you might want to look at the following:

Zope Complete web server and application framework

Twisted Complete set of web modules implementing everything web-related.

Webware Suite for developing web-based applications

Alternatively search Google

CHAPTER

TWO

Getting Started

2.1 What Is Python?

Here is a description taken from the Python page for newbies, http://www.python.org/topics/learn

Python is a clear and powerful object-oriented programming language, comparable to Perl, Tcl, Scheme, or Java.

Some of Python's notable features:

- Python uses an elegant syntax for readable programs
- Python is an agile language that makes it easy to get your program working. This makes Python an ideal language for prototype development and other ad-hoc programming tasks, without compromising maintainability.
- A variety of basic data types are available: numbers (integers, floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, dictionaries.
- A variety of basic data types are available: numbers (integers, floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, dictionaries.
- Python supports object-oriented programming with classes and multiple inheritance.
- Code can be grouped into modules and packages.
- The language supports raising and catching exceptions, resulting in cleaner error handling.
- Data types are strongly but dynamically typed. Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised.
- Python contains advanced programming features such as generators and list comprehensions.
- Automatic garbage collection frees you from the hassles of memory management.
- The large standard library supports many common programming tasks such as connecting to web servers, regular expressions, and file handling.
- Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
- The Python interpreter is easily extended by adding new modules implemented in a compiled language such as C or C++.
- The interpreter can also be embedded into an application to provide a programmable interface.
- Python runs on many different computers and operating systems: Windows, MacOS, many brands of Unix, OS/2, ...

- Python is open source software, copyrighted but freely usable and distributable, even for commercial use.

To begin learning the Python language, you should download the Python interpreter and read the tutorial.

2.2 How are PythonWeb Applications Written?

PythonWeb is designed to let you program in whichever way is most appropriate for your platform and setup. The aim of creating the modules was not to tie users into a particular way of doing things.

There are however two main ways in which PythonWeb applications are commonly written. These are:

- CGI Scripts
- Web Server Gateway Interface applications.

CGI scripts are slighlty simpler to understand and so the first part of this guide focuses on the principles you need to understand in order to use PythonWeb to write Python CGI scripts.

The Web Server Gateway Interface or WSGI as it is often referred to is a recent specification designed to be a standard in which servers, middleware and applications from different Python web frameworks can be used together.

See Also:

PEP 333 - Python Web Server Gateway Interface
(http://www.python.org/peps/pep-0333.html)
: This document specifies a proposed standard interface between web servers and Python web applications or frameworks, to promote web application portability across a variety of web servers.

CHAPTER

THREE

Getting Ready For PythonWeb

3.1 What Do I Need?

You will need to download and install Python as described in the Installation guide.

You will need a web server on which to run your application. Any web server which can run Python CGI scripts can be used. For example Apache or the PythonWeb web server available in the 'scripts/webserver.py' directory.

Note: If you intend to run WSGI applications you will also need a WSGI server such as 'scripts/WSGIServer.py' (although WSGI aplications can also be run as CGI applications with a consequent performance loss)

The next sections explain the options available. Once you have a web server capable of running Python CGI scripts you can continue with this guide.

3.1.1 Apache

Apache is the reference web server for the Web Modules and it is available for many platforms including Windows.

Apache can be configured to support Python. You may need to edit you 'httpd.conf' file adding something similar to `addhandler py /usr/bin/python`. `/usr/bin/python` should be relpaced with the correct path to the Python interpreter on your system. This can usually be found by typing the following at a command prompt:

```
> whereis python
```

Alternatively you could start python and type the following to obtain the `sys.prefix`. This is often the directory in which the python interpreter is.

```
>>> import sys
>>> print sys.prefix
/usr/local/bin/python
```

You should consult the Apache documentation at http://httpd.apache.org for full information.

3.1.2 webserver.py

If you do not have a web server you can use the one that comes with the web modules. You can find it in the 'scripts' directory.

Start the server with the following command:

```
> python webserver.py
```

Once the server is running you should be able to access the examples at http://localhost:8080/doc/html_multipage/web/examples.html and run scripts in the 'doc/src/lib' directory at http://localhost:8080/doc/src/lib.

By default the only directory you can access scripts in is the 'doc/src/lib' directory and these scripts must be Python files and end in '.py'. You change which directory should be treated as the 'cgi-bin' directory by changing the options you start the webserver with.

To see a list of options use:

```
> python webserver.py -h
```

CHAPTER

FOUR

CGI Tutorial

4.1 What is CGI?

CGI stands for Common Gateway Interface and is a simple specification which defines some common variables and allows computer programs to run on web servers.

CGI scripts are simply computer programs which run on a web server and output HTTP headers and HTML in a standard format back to the web browser. CGI scripts can be written in virtually any language including Python. CGI scripts can be used for creating dynamic web pages, responding to user actions, accessing databases, creating shopping carts, etc.

CGI programs have to have special permission to run on a server. This is to prevent people uploading files to a server and then running potentially harmful programs. Many ISP's do not let people run CGI scripts on their web space so you will have to find a server that does or use the webserver provided with PythonWeb (described in the previous section).

Rather than printing to the screen, CGI programs print to the web browser so that a standard `print` command produces text which will be seen on a web browser.

4.2 Hello world!

The best way to demonstrate the basics is to create a "Hello world!" program. This tutorial uses Python and we will assume you are using the Apache Web Server. In a text editor type the following into a file called 'hello.py':

```
#!/usr/bin/env python

print "Content-type: text/html\n\n"
print "<html>Hello world!</html>"
```

The first line tells the web server where the Python program is on the server. It is usually in /usr/bin/python or /usr/local/bin/python so you must change the first line to suit your server but you must remember the #! at the start, the slashs are forward slashes, and that you must have a least one completely blank line before you start anything else.

Warning: One issue which sometimes casues problems is line endings. UNIX uses LF as a line feed, windows uses CR+LF. To confuse matters further some FTP clients try to automatically convert the line endings for you. My tip is to use a text editor such as SciTE which supports both types of line endings. Set `Options->Line Ending Characters->LF` and then `Options->Convert Line Ending Characters` to make sure you are using LF line feeds. Then you can upload your file in Binary mode on the FTP client and you can guarantee your line endings are correct.

Back to our Hello World! program. The line containing `text/html` is an HTTP header which tells the web browser to expect HTML to be sent to it. If your CGI script sent a GIF image, for example, you would use `image/gif`. Note that in this case we used two line ending characters `\n\n` after the HTTP header the second of these leaves a completely blank line which tells the server or browser the HTTP headers have finished and to expect some content. If we had many HTTP headers only the last would be sent with two blank lines. HTTP headers must be sent to the web browser before the main content of a page otherwise you are likely to get a message saying `"Server Error 500"`.

The last line prints the HTML body to the web browser and is what would be seen if you were to use your browser's View Source option.

Once you have written your script you need to upload it to a directory which supports execution of scripts using FTP software. This directory will normally be called `cgi-bin` or `cgi`. **Note:** In the case of the webserver which comes with PythonWeb the cgi directory is `doc/src/lib` so you should put your application in there.

Next you will need to change the file permissions of your script using the UNIX command `chmod 755 hello.py` or using your FTP software to change the file permissions so that everyone can read and execute your file but only you, the owner, can write it. Now, using your web browser visit the hello.cgi script by typing http://www.yourservername.com/cgi-bin/hello.py or something equivalent for where your file is on the internet.

If all goes well you should see the text:

```
Hello world
```

If not, check you have followed the instructions correctly. If you get an error message saying something like `"Error 500 there was an error in the script"` it is likely you have mis-typed something or the server cannot find the Python program. Error 404 means that you have typed the wrong address and that is not where the 'hello.py' script is. Error 403 means you do not have permission to execute the file and you need to change the file permissions. If you simply see the source code it means the server is treating the file as a plain text file and not as a computer program. Be sure you have put the put the script in an executable directory.

If you have problems you can access your script using telnet. Load telnet and type `open yourserver.com` enter your username and password and navigate to your cgi script using the following commands:

`ls` to give a list of directories `cd cgi-bin` to change directory to cgi-bin `cd ..` to go up a level.

You can then type `python hello.py` and if there are errors in your script, the Python interpreter will find them. Correct the errors and then try to execute `./hello.py` If the script will not execute check the permisssions and check the first line correctly points to the python interpreter. You may also want to check you are using the correct line endings as mentioned earlier. Once this works the webserver should also be able to execute your script. Occasionally you may also need to change the user of the script but this is very rare.

As a final resort if you cannot get Python to work you could try and write a CGI script in perl or rename the file 'hello.cgi' rather than 'hello.py'. If you are still having problems it would be worth contacting your ISP and asking for some help.

It can be a complicated business to start with but once you have done it once you should be able to do it again with ease!

4.3 Using PythonWeb

Hopefully by now you have a functioning web server that is producing output to your web browser. You are now ready to start learning PythonWeb.

If you haven't already installed the web modules you should do so now. There are instructions in the *Getting Started* guide. Alternatively you can use `sys.path` to tell Python where the modules are installed.

Here is the same `Hello world!` example written with PythonWeb.

```
#!/usr/bin/env python

import sys; sys.path.append('../dir')
import web
print web.header()
print "<html>Hello world!</html>"
```

Agian you may have to change the first line to point to the Python interpreter and you should change `'../dir'` to point to the directory containing the 'web' directory as described in the installation section.

The next example prints a list of all the CGI variables your server is making available to CGI scripts.

```
#!/usr/bin/env python

import sys; sys.path.append('../dir')
import web
print web.header('text/plain')
for key, value in web.cgi.items():
    print "%s: %s\n"%(key, value)
```

Note how this time we are sending the information as plain text using an HTTP header `web.header('text/plain')`.

Once you have this working you are ready to begin reading the *Module Reference* and looking at the examples in the next section.

CHAPTER

FIVE

Examples

As you read through the *Module Reference* section of the documentation you will come across a number of examples. All the exmples from the Module Reference can be found in the 'doc/src/lib' directory of the distribution.

The examples with a filename starting with `command-` are designed to be run from the command line. Those starting with `webserver-` are designed to be run by the test webserver. If you are using the test webserver you do not need to install the modules to test the examples. They will run directly from the source distribution.

The web modules distribution comes with a simple Python webserver, similar to the one described earlier in this document. The test webserver is designed to allow you to test the examples in this documentation on your own machine. To use it run the 'webserver.py' file in the 'scripts' directory of the source distribution. If you run the webserver and visit http://localhost:8080/doc/html/web/examples.html on your local machine you should see the examples index page. If this is the case you can test the examples on your own computer.

Warning: For the webserver to work properly, the path to the webserver.py file should contain no spaces. One important point to note about the test webserver is that it treats all paths relative to the place where the webserver is running and not relative to the script it is executing. This means that if you want to run the examples on a different webserver you will have to change the paths to certain files.

Please note the first line in each of the examples is the following:

```
#!/usr/bin/env python
```

You should change this line to whatever you use on your system to run Python. This doesn't apply if you are running Windows but on UNIX systems you may need to change it. Common values are:

```
#!/usr/bin/python
```

or

```
#!/usr/local/bin/python
```

and there should be two blank lines after this line.

The examples use UNIX linefeeds. This means that if you use something like notepad.exe on Windows all the lines will look like they run into one. Instead use IDLE (included with Python) or WordPad to display the examples correctly.

Below is an index of examples from the module reference:

web.auth

- *Simple Auth Example*

- *Full Auth Example*

web.database
- *Database Test Example*

web.database.object
- *Simple Example*
- *One-to-Many Mappings*
- *Many-to-Many Mappings*
- *Building Queries*
- *Creating Forms*
- *Other Features*

web.error
- *Error Handling Example*

web.form
- *Basic Fields Example*
- *Typed Fields Example*

web.image.graph
- *Graph Example*

web.mail
- *Email Example*

web.session
- *Simple Example*
- *Full Example*

web.template
- *Cheetah Template Example*
- *XYAPTU Template Example*
- *Dreamweaver MX Template Example*

web.util
- *Table Output Example*

web.util
- *XSL Transform Example*

CHAPTER

SIX

Web Server Gateway Interface Programming

Full information about WSGI progamming can be found in the *`web.wsgi` module documentation*.

CHAPTER

SEVEN

Appendicies

7.1 Future Things To Do

This is an adhoc list of future improvements whcih could be made.

`web.database`

- Write a Python PEP using DB as a recommendation
- Write a Postgresl drivers
- Write a proper unit test

`datetime`

- Fix the issues with mx modules

`web.auth`

- Auth Encryption type: password etc

`web.template`

- Implement a PHP style version ¡%python %¿