CHAPTER 15

Python and the Web

This chapter tackles some aspects of web programming with Python. This is a really vast area, but I've selected three main topics for your amusement: screen scraping, CGI, and mod_python. In addition, I give you some pointers for finding the proper toolkits for more advanced web application and web service development. For extended examples using CGI, see Chapters 25 and 26. For an example of using the specific web service protocol XML-RPC, see Chapter 27.

Screen Scraping

Screen scraping is a process whereby your program downloads web pages and extracts information from them. This is a useful technique that pops up every time there is a page online that has information you want to use in your program. It is especially useful, of course, if the web page in question is dynamic; that is, if it changes over time. Otherwise, you could just download it once and extract the information manually. (The ideal situation is, of course, one where the information is available through web services, as discussed later in this chapter.)

Conceptually, the technique is very simple. You download the data and analyze it. You could, for example, simply use urllib, get the web page's HTML source, and then use regular expressions (see Chapter 10) or another technique to extract the information. Let's say, for example, that you wanted to extract the various employer names and web sites from the Python Job Board, at http://python.org/community/jobs. You browse the source and see that the names and URLs can be found as links in h3 elements, like this (except on one, unbroken line):

```html
<h3><a name="google-mountain-view-ca-usa"><a class="reference" href="http://www.google.com">Google</a> ...</h3>
```

Listing 15-1 shows a sample program that uses urllib and re to extract the required information.

Listing 15-1. A Simple Screen-Scraping Program

```python
from urllib import urlopen
import re
p = re.compile('<h3><a .*?><a .*? href="(.*?)">(.*?)</a>\n
for url, name in p.findall(text):
    print '%s (%s)' % (name, url)
```

The code could certainly be improved (for example, by filtering out duplicates), but it does its job pretty well. There are, however, at least three weaknesses with this approach:

- The regular expression isn’t exactly readable. For more complex HTML code and more complex queries, the expressions can become even more hairy and unmaintainable.

- It doesn’t deal with HTML peculiarities like CDATA sections and character entities (such as `&amp;`). If you encounter such beasts, the program will, most likely, fail.

- The regular expression is tied to details in the HTML source code, rather than some more abstract structure. This means that small changes in how the web page is structured can break the program. (By the time you’re reading this, it may already be broken.)

The following sections deal with two possible solutions for the problems posed by the regular expression-based approach. The first is to use a program called Tidy (as a Python library) together with XHTML parsing. The second is to use a library called Beautiful Soup, specifically designed for screen scraping.

**Note** There are other tools for screen scraping with Python. You might, for example, want to check out Ka-Ping Yee’s scrape.py (found at http://zesty.ca/python).

---

**Tidy and XHTML Parsing**

The Python standard library has plenty of support for parsing structured formats such as HTML and XML (see the Python Library Reference, Section 8, “Structured Markup Processing Tools,” at http://python.org/doc/lib/markup.html). I discuss XML and XML parsing in more depth in Chapter 22. In this section, I just give you the tools needed to deal with XHTML, the most up-to-date dialect of HTML, which just happens to be a form of XML.

If every web page consisted of correct and valid XHTML, the job of parsing it would be quite simple. The problem is that older HTML dialects are a bit more sloppy, and some people don’t even care about the strictures of those sloppier dialects. The reason for this is, probably, that most web browsers are quite forgiving, and will try to render even the most jumbled and meaningless HTML as best they can. If this happens to look acceptable to the page authors, they may be satisfied. This does make the job of screen scraping quite a bit harder, though.

The general approach for parsing HTML in the standard library is event-based; you write event handlers that are called as the parser moves along the data. The standard library modules `sgmllib` and `htmllib` will let you parse really sloppy HTML in this manner, but if you want to extract data based on document structure (such as the first item after the second level-two heading), you’ll need to do some heavy guessing if there are missing tags, for example. You are certainly welcome to do this, if you like, but there is another way: Tidy.

**What’s Tidy?**

Tidy (http://tidy.sf.net) is a tool for fixing ill-formed and sloppy HTML. It can fix a range of common errors in a rather intelligent manner, doing a lot of work that you would probably rather not do yourself. It’s also quite configurable, letting you turn various corrections on or off.