CHAPTER 1

Introducing Pylons

This book is about Pylons, an exciting, modern web development framework that puts the developer firmly in control and makes building complex web applications as easy as possible.

Pylons has grown hugely in popularity in recent years because of its careful balance of powerful development features and its modular internal architecture, which help developers to quickly create sophisticated web applications without hiding what is really going on behind the scenes. Pylons gives you the power you need to efficiently create web sites and web applications while also being flexible enough to allow you to do things differently when you really need to. Best of all, Pylons is an open source project with a great community behind it to offer help and support when you need it.

The first part of this book will give you all the knowledge you need to start using Pylons' default configuration to build high-quality production web sites. In Part 2, you'll learn about some of Pylons' more advanced features, such as Unicode and internationalization support, Ajax, and URL routing, before moving on to Part 3 to learn about expert topics such as the Web Server Gateway Interface, authentication and authorization, deployment, and logging.

Each chapter will serve as a complete guide to each of the topics covered and will contain links to areas you can go for further information. Throughout the book, I will also be taking you through how to develop a simple web site application called SimpleSite so that you can see how the principles described in each of the chapters apply in a real Pylons application.

The Old Way: CGI Scripts

In the past, developers typically wrote web applications as a series of simple scripts, each of which would be responsible for accessing the database for the data it needed and generating HTML to produce the pages it output. Each individual script was quick to write and easy for an experienced developer viewing the code for the first time to understand, because everything relevant to the generation of a particular page would be in one script. Developers had direct access via SQL to the database they were using and had the power and flexibility to write their code in whichever way was appropriate for their needs.

Here's a simple example of the way CGI scripts used to be written:

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

# Get the configuration for the script
import ConfigParser
config = ConfigParser.ConfigParser()
config.read('/path/to/config.ini')
```

```
# If debugging is enabled, set up the cgitb module
if config.get('general', 'debug') == 'on':
    import cgitb; cgitb.enable()

# Begin the non-configuration-dependent imports
import cgi
import MySQLdb
import os

# Output the HTTP headers
print "Content-type: text/html\n\n"

# Output the head of the HTML page
print "<html><head><title>Example</title></head>"
print "<body><h1>Example</h1><p>Here are the comments:</p>"

# Get the ID from the URL based on the QUERY_STRING environment
# variable using the cgi module
fields = cgi.FieldStorage()
page = int(fields['page'].value)

# Fetch data from the database
connection = MySQLdb.connect(
    db=config.get('database', 'database'),
    user=config.get('database', 'user'),
    passwd=config.get('database', 'password'),
    host=config.get('database', 'host')
)
cursor = connection.cursor()
cursor.execute("SELECT id, data FROM comment WHERE page=%s", (page,))
results = cursor.fetchall()
cursor.close()
connection.close()

# Output the comments
for id, data in results:
    print "<p>Commment #%s: %s</p>"%(id, cgi.escape(data))

# Output the rest of the HTML page
print "</body></html>"
```

This script would display a list of comments when a URL such as `/cgi-bin/test.cgi?page=1` was entered. You would also need a config file, which looked something like this:

```
[general]
debug = off

[database]
database = dbname
host = localhost
user = james
password = somepassword
```

The CGI script code isn't as elegant as it could be, but it does have some benefits. Let's look at the pros and cons. First, here are the pros:

- Someone who understands HTTP and SQL will probably be able to understand most of this code because it uses standard web development techniques.
- Coding in this manner gives the developer a huge amount of power because they have control over every aspect of the HTTP response and can write SQL queries that are as complex as they like.

Now, here are some of the cons:

- Every script in the site needs the same code to load the config file and to handle the errors.
- Writing database access code is very repetitive, and the data structures from the database don't necessarily represent the objects your application wants to deal with.
- CGI scripts can be slow because the whole Python interpreter as well as the modules the script uses need to be loaded into memory on each request.
- Designers will find it difficult to change the theme of the site because the HTML-generating code is interspersed with Python code.

There are also some more subtle problems with creating a whole application as a series of scripts:

- Code is frequently duplicated in multiple scripts so over time the code can become difficult to maintain as developers change the database or the code in certain files but aren't aware that other scripts also rely on the way the database or code used to work.
- It can be difficult to understand how the whole application is structured because each script can behave fairly autonomously.
- URLs in the form `/cgi-bin/path/to/script.cgi?controller=page&action=view&id=3` do not readily reflect the structure of your web application and are not as natural to a user as a URL such as `/page/view/3`.

The Pylons Way

To address these problems, Pylons (as well as other popular frameworks such as Django, TurboGears, and Ruby on Rails) use two main techniques:

- A Model View Controller (MVC) architecture
- Convention over configuration

Pylons also puts particular emphasis on loose coupling and clean separation. You'll learn about each of these ideas in the following sections.

Model View Controller Architecture

The *Model View Controller* architecture is a result of the recognition that, at their heart, most web applications:

- Store and retrieve data in a way that is natural to the programming language involved (the model)
- Represent the data in various ways, most commonly as HTML pages (the view)
- Execute logic code to manipulate the data and control how it is interacted with (the controllers)

In Pylons, each of these components is kept separate. Requests are dispatched to a *controller*, which is an ordinary Python class with methods called *actions* that handle the application logic. The controller then interacts with the model classes to fetch data from the database. Once all the necessary information has been gathered, the controller passes the key information to a *view template* where an HTML representation of the data is generated and returned to the user's browser. The user then interacts with the view to create a new request, and the process starts again. The model and controller don't contain code for generating HTML, and the view templates shouldn't interact directly with the model.

This architecture is useful because it not only reflects what happens in most web applications, but it also keeps your application easy to maintain because you always know where the code handling a particular aspect of your application can be found. For example, Pylons uses a templating language called Mako to help you generate HTML and recommends the object-relational mapper SQLAlchemy to help you with your model. You'll learn much more about models, controllers, and view templates in the following chapters.

■ **Caution** Those of you coming to Pylons from Django might be accustomed to Django's approach to the MVC architecture, which it refers to as MTV (which stands for model/template/view). Although conceptually quite similar to Pylons' traditional approach to MVC, there are two key terminology differences:

Django's template is equivalent to Pylons' view.

Django's view is equivalent to Pylons' controller.

The model is treated similarly in Django and Pylons. You can find a discussion of Django's reasons for its terminology on the Django web site at `http://tinyurl.com/3mwwhf`.

Convention Over Configuration

A lot of the complexity of web development can be removed by assuming that the developer wants to do the most obvious thing. For example, almost every time a user requests a page from your site, you will want to return a simple HTML page. Sometimes you may want to return an image or perhaps stream some custom binary data, but most of the time, a simple HTML page is all that's required.

With this in mind, the Pylons developers designed Pylons to automatically *assume* data you return is HTML *unless* you specify otherwise. This means that for the common cases you don't have to *configure* the `Content-type` because the *convention* is that it will be `text/html` unless you want to do things differently.

Loose Coupling and Clean Separation

Web frameworks such as Django and Ruby on Rails have become extremely popular in recent years because they provide a structure that allows you to quickly create good-looking web sites by defining the way the data is structured. The tools they provide then work on that data either to automatically generate code (scaffold in the case of Ruby on Rails) or to create form interfaces at runtime (as is the case with Django).

Although these frameworks maintain a clean separation between the model, view, and controller layers of code, they aren't loosely coupled in the way that Pylons is because the ability of the application as a whole to work relies heavily on the glue code found in the framework itself. Although it can be easy to write a simple application with these frameworks, it can also be harder to customize their behavior later in a project because doing so frequently involves understanding how the code provided by the framework itself works before you can change its behavior. To make the framework itself easy to use, the framework code sometimes has to be quite complex, and as a result, customization can sometimes be rather difficult.

Pylons, on the other hand, is much more loosely coupled. Because it doesn't provide tools to automatically generate a nearly finished site for you from the definition of the model, it doesn't need complicated glue code holding everything together. Instead, it provides sensible low-level APIs and methodologies that allow you to quickly and easily glue together the component parts you choose to use for yourself.

This loosely coupled approach doesn't mean you have to code absolutely everything for yourself either. Pylons uses convention over configuration and assumes you will want to use a standard setup when you create a new project. If you don't want a standard setup, it is easy to customize the way Pylons works. For example, by default Pylons uses a templating framework called Mako to help you generate the templates for your views, but you are under no obligation to use it. By customizing your Pylons project, you can easily use any of the other major Python templating languages including Genshi, Jinja, or even TAL or Breve.

Other Features

In addition to handling a model, views, and controllers, modern web frameworks also have to provide tools to facilitate each of the following processes:

- Mapping the URL a user visits to the code to be run
- Reading data such as form posts that are sent via HTTP
- Validating and repopulating forms
- Dealing with user accounts
- Storing session information, perhaps using a cookie

They also often provide the following:

- A server component to run the application
- Automatic documentation generation tools
- Systems for packaging, distribution, and deployment
- Testing tools
- Internationalization tools and Unicode support

- Interactive debugging tools
- Logging facilities

Pylons is no exception and provides tools and methodologies for handling all of these things. You'll learn about each of them during the course of the book.

The Python Language

Python is a fantastic language for a huge range of programming problems. It is easy to learn and yet powerful and expressive enough to handle all manner of tasks. It is also great for web programming (there are good reasons it is one of the three official languages at Google). Perhaps its key benefit is that Python has a great deal of support across all popular platforms including BSD, Linux, Mac OS X, and Windows, as well as a huge range of software libraries already available for it, so the majority of the time you will be able to find a suitable tool for the task you are trying to achieve without having to write it yourself.

If you haven't yet learned Python, now would be a good time to read the Python Tutorial at `http://docs.python.org/tut/tut.html`, which will give you all the knowledge you need to get started developing applications with Pylons.

Another good source of information for learning Python is Mark Pilgrim's book *Dive Into Python* published in 2004. It is freely available online at `http://www.diveintopython.org/toc/index.html` and goes into a bit more detail about Python.

Python 3.0

Python 3.0 is a new version of the Python programming language that is currently in alpha release (as of this writing). Unlike recent upgrades to the Python programming language, Python 3.0 will not be fully backward compatible with previous versions, so it is likely that code written for Pylons at the moment will not automatically work with Python 3.0.

Luckily, this isn't a problem you need to be too worried about for three reasons:

- The changes in Python 3.0 are fairly small, so it is going to be fairly easy to upgrade your code.
- The Python language team will continue to release 2.*x* versions of Python after the 3.0 release.
- A tool will be available to automatically translate Python source files from 2.*x* to 3.*x*, and it will handle the vast majority of the conversions necessary. If you are interested, the current development version is at `http://svn.python.org/projects/sandbox/trunk/2to3/`.

Once there is enough demand for a Python 3.0 version of Pylons and enough of the Pylons dependencies have been updated, the Pylons team plans to release a pair of feature-identical Pylons versions, one for Python 2.6 and one for Python 3.0, in line with the Python community's current recommendation.

Python 2.6 includes a feature to print warning messages about any code that is not compatible with Python 3.0, so you will be able to run your code on Pylons for Python 2.6 to find out where any problems might be and then, after making changes or running the 2to3 refactor tool, you will be able to run your code on Pylons for Python 3.0.

The Pylons Community

A key benefit of choosing Pylons is that there is a thriving and helpful community built around it. Getting active in the Pylons community is easy, and we're always looking to increase community participation.

Besides the official documentation, there is also a Pylons Cookbook that contains user-contributed documentation as part of the Pylons wiki. The community always welcomes new contributions to the Cookbook or comments on existing articles, and if you register on the wiki, you can even export to PDF by clicking the PDF icon in the top right of the pages to take them on the road with you.

Pylons wiki: `http://wiki.pylonshq.com/`

Official documentation: `http://wiki.pylonshq.com/display/pylonsdocs/`

Cookbook: `http://wiki.pylonshq.com/display/pylonscookbook/`

For more direct help, the Pylons Discuss mailing list on Google Groups is always active, and usually quite a few people on the Pylons freenode channel on IRC are happy to help too.

IRC: `#pylons` on `irc.freenode.net`

Mailing list: `http://groups.google.com/pylons-discuss`

Pylons Components

Unlike other web frameworks where each component has been custom built for the framework and then tightly integrated into its other components. Pylons is more like a collection of very carefully chosen third-party software. Rather than starting from scratch on each of the components making up Pylons, the developers instead worked with existing software teams to develop standards and APIs that would allow their software to work with Pylons.

This approach turns out to be hugely useful and has three core benefits:

- The APIs and methodologies that allow Pylons to work with one class of component—say, templating languages—also mean that when newer and better software comes along, it is a simple task to use the same APIs and methodologies to allow Pylons to work with the new software.
- From the individual software project's point of view, the APIs that have been developed to allow the software to work with Pylons also mean the software is much easier to integrate into other frameworks since the required APIs already exist.
- Because many of the components weren't initially designed just to be used in Pylons, it also means you are much more likely to be able to use them in ways you wouldn't normally expect web framework components to be used. For example, FormEncode is also an excellent general-purpose conversion library, and SQLAlchemy is used in many projects entirely unrelated to the Web. This means that as your applications grow or if your requirements change, Pylons is much more likely to be able to keep pace.

One of the problems for newcomers to Pylons is that it can be difficult to know how all the components fit together, particularly because the individual projects' documentation isn't necessarily web-focused. That's where this book comes in.

What's Coming Up

Over the next 20 chapters, you'll learn everything you need to know to create a simple web site with a navigation hierarchy, editable sections and pages, a comment system, and tags support. The application will serve as a very good starting point for any Pylons-based web site you create.

The book is divided into three parts: "Getting Started," "Advanced Pylons," and "Expert Pylons." In the first part, you'll learn the following:

- How to install Pylons on Linux, Mac, or Windows in a way that doesn't interfere with other software on your system
- How to use project templates to create customizable project skeletons to get you up and running quickly
- The basics of HTTP and how Pylons' request and response objects make working with it much easier
- The basics of the Pylons architecture and what each of the Pylons globals is for
- How to use Pylons' industry-leading interactive debugger, as well as email reporting tools
- How to create view templates with Mako including how to take advantage of features such as inheritance to apply consistent themes across multiple pages as well as how to use components to generate common structures such as navigational elements
- How to create forms using the Pylons helpers and how to validate and repopulate them as necessary using FormEncode and HTML Fill
- How to deal with file uploads and repeating validation structures involving one-to-many mappings in your model
- The various software options for your model, whether it be an XML database, an Amazon S3 store, or a traditional relational database management system
- How you can use SQLAlchemy to model your database, saving you time and effort

At the end of Part 1, you'll also get started with the example application called SimpleSite so that you see how the techniques you've learned apply in a real application.

Once you've mastered the basics, you'll move on to Part 2 to take a look at some of the more advanced features and techniques that can be used in Pylons applications. These include the following:

- How to use Routes to allow complex mappings between the URLs your application handles and the code that powers them as well as best practices for URL design and how URLs can be used as simple state stores
- What Unicode is and how to use it throughout your Pylons application
- How to write a Pylons application that supports multiple languages and displays non-Western characters such as Japanese or Arabic
- The YUI library and how it can be used to simplify your client-side CSS and layouts
- The basics of the JavaScript language including the areas where it differs from Python
- How to use Ajax and animation to improve your web applications
- How to write effective unit and functional tests
- How to use docstrings and reStructuredText to quickly and easily write good documentation for your Pylons project

You'll then return to the SimpleSite example and add a navigation hierarchy, CSS, and JavaScript to the example application as well use some advanced SQLAlchemy features such as inheritance.

Once you've mastered these techniques, you'll learn all about Pylons' internal structure and how you can easily use a specification called the Web Server Gateway Interface to extend or change the way Pylons itself works. in Part 3, you'll cover the following:

- The Web Server Gateway Interface specification and the details of how to program various types of Web Server Gateway Interface components
- A bit about the history of Pylons and how it influences the design methodology
- How Pylons uses the PasteDeploy package and egg entry points to allow easy customization of middleware and applications in configuration files
- How Pylons uses the egg format and `setuptools` to allow easy packaging and distribution of Pylons applications and dependencies on the Python Package Index
- How to use AuthKit to implement authentication and authorization appropriate to your application's needs
- The principles of various ways web applications can be written including multithreaded, multiprocess, and asynchronous web applications and why most deployment options for Pylons are multithreaded
- How to deploy a Pylons application using Apache or Nginx proxying to a Paster server, how to use `mod_wsgi` to embed a Pylons applications in an Apache server, and where to learn about the many other ways to deploy Pylons applications
- How to use Python's powerful logging system with a Pylons application

You'll then take a final look at the SimpleSite application and see how to turn the project back into a project template so that other people can use it as a starting point for their own applications.

By the end of the book you should have a thorough understanding of how to use Pylons as well as a good knowledge of the technologies Pylons uses and the underlying reasons for their inclusion in the Pylons framework so that you are empowered to make your own choices about which components to use in your own Pylons applications.

Pylons is a framework that is designed to work with you and not to enforce its view of the world on your project. This book will give you the skills you need to use Pylons' default options but also to know when to break the rules.

Summary

This chapter gave you a broad understating of some of the design philosophies of Pylons and what makes it slightly different from other frameworks you might have used.

There's clearly a lot to learn, so let's get started!