Introduction to Django

This book is about Django, a Web development framework that saves you time and makes Web development a joy. Using Django, you can build and maintain high-quality Web applications with minimal fuss.

At its best, Web development is an exciting, creative act; at its worst, it can be a repetitive, frustrating nuisance. Django lets you focus on the fun stuff—the crux of your Web application—while easing the pain of the repetitive bits. In doing so, it provides high-level abstractions of common Web development patterns, shortcuts for frequent programming tasks, and clear conventions for how to solve problems. At the same time, Django tries to stay out of your way, letting you work outside the scope of the framework as needed.

The goal of this book is to make you a Django expert. The focus is twofold. First, we explain, in depth, what Django does and how to build Web applications with it. Second, we discuss higher-level concepts where appropriate, answering the question “How can I apply these tools effectively in my own projects?” By reading this book, you’ll learn the skills needed to develop powerful Web sites quickly, with code that is clean and easy to maintain.

In this chapter, we provide a high-level overview of Django.

What Is a Web Framework?

Django is a prominent member of a new generation of Web frameworks. So, what exactly does that term mean?

To answer that question, let’s consider the design of a Web application written using the Common Gateway Interface (CGI) standard, a popular way to write Web applications circa 1998. In those days, when you wrote a CGI application, you did everything yourself—the equivalent of baking a cake from scratch. For example, here’s a simple CGI script, written in Python, that displays the ten most recently published books from a database:

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

import MySQLdb
```
print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>"
connection = MySQLdb.connect(user='me', passwd='letmein', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC LIMIT 10")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]
print "</ul>"
print "</body></html>"
connection.close()

This code is straightforward. First, it prints a “Content-Type” line, followed by a blank line, as required by CGI. It prints some introductory HTML, connects to a database, and executes a query that retrieves the latest ten books. Looping over those books, it generates an HTML unordered list. Finally, it prints the closing HTML and closes the database connection.

With a one-off dynamic page such as this one, the write-it-from-scratch approach isn't necessarily bad. For one thing, this code is simple to comprehend—even a novice developer can read these 16 lines of Python and understand all it does, from start to finish. There's nothing else to learn; no other code to read. It's also simple to deploy: just save this code in a file called latestbooks.cgi, upload that file to a Web server, and visit that page with a browser.

But as a Web application grows beyond the trivial, this approach breaks down, and you face a number of problems:

• What happens when multiple pages need to connect to the database? Surely that database-connecting code shouldn't be duplicated in each individual CGI script, so the pragmatic thing to do would be to refactor it into a shared function.

• Should a developer really have to worry about printing the “Content-Type” line and remembering to close the database connection? This sort of boilerplate reduces programmer productivity and introduces opportunities for mistakes. These setup- and teardown-related tasks would best be handled by some common infrastructure.

• What happens when this code is reused in multiple environments, each with a separate database and password? At this point, some environment-specific configuration becomes essential.

• What happens when a Web designer who has no experience coding Python wishes to redesign the page? Ideally, the logic of the page—the retrieval of books from the database—would be separate from the HTML display of the page, so that a designer could edit the latter without affecting the former.

These problems are precisely what a Web framework intends to solve. A Web framework provides a programming infrastructure for your applications, so that you can focus on writing clean, maintainable code without having to reinvent the wheel. In a nutshell, that's what Django does.