to my family
CONTENTS

Preface xv

1 Introduction 1
   1.1 Principles 3
   1.2 Web Frameworks 4
   1.3 Model-View-Controller 5
   1.4 Why web2py 8
   1.5 Security 9
   1.6 In the box 12
   1.7 License 13
   1.8 License Commercial Exception 14
   1.9 Acknowledgments 15
   1.10 About this Book 16
   1.11 Elements of Style 18
2 The Python Language 21
2.1 About Python 21
2.2 Starting up 22
2.3 help, dir 23
2.4 Types 24
2.5 About Indentation 28
2.6 for...in 28
2.7 while 29
2.8 def...return 29
2.9 if...elif...else 31
2.10 try... except...else...finally 31
2.11 class 33
2.12 Special Attributes, Methods and Operators 34
2.13 File Input/Output 34
2.14 lambda 35
2.15 exec, eval 36
2.16 import 37

3 Overview 41
3.1 Startup 41
3.2 Say Hello 45
3.3 Let’s Count 50
3.4 Say My Name 51
3.5 Form self-submission 53
3.6 An Image Blog 56
3.7 Adding CRUD 69
3.8 Adding Authentication 70
3.9 A Wiki 71
3.10 More on admin 81
   [site] 81
   [about] 84
   [EDIT] 85
   [errors] 87
   [mercurial] 91
3.11 More on appadmin 91

4 The Core 93
4.1 Command Line Options 93
## CONTENTS

4.2 URL Mapping 96  
4.3 Libraries 99  
4.4 Applications 103  
4.5 API 104  
4.6 request 105  
4.7 response 107  
4.8 session 110  
4.9 cache 111  
4.10 URL 113  
4.11 HTTP and redirect 115  
4.12 T and Internationalization 116  
4.13 Cookies 117  
4.14 init Application 118  
4.15 URL Rewrite 118  
4.16 Routes on Error 120  
4.17 Cron 121  
4.18 Import Other Modules 124  
4.19 Execution Environment 124  
4.20 Cooperation 126  

5 The Views 127  
5.1 Basic Syntax 129  
      for...in 129  
      while 130  
      if...elif...else 130  
      try...except...else...finally 131  
      def...return 131  
5.2 HTML Helpers 132  
      XML 133  
      Built-in Helpers 134  
      Custom Helpers 142  
5.3 BEAUTIFY 143  
5.4 Page Layout 143  
5.5 Using the Template System to Generate Emails 146  
5.6 Layout Builder 147  

6 The Database Abstraction Layer 149  
6.1 Dependencies 149
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2 Connection Strings</td>
<td>151</td>
</tr>
<tr>
<td>Connection Pooling</td>
<td>152</td>
</tr>
<tr>
<td>6.3 DAL, Table, Field</td>
<td>153</td>
</tr>
<tr>
<td>6.4 Migrations</td>
<td>154</td>
</tr>
<tr>
<td>insert</td>
<td>158</td>
</tr>
<tr>
<td>commit and rollback</td>
<td>159</td>
</tr>
<tr>
<td>executesql</td>
<td>160</td>
</tr>
<tr>
<td>_lastsql</td>
<td>160</td>
</tr>
<tr>
<td>drop</td>
<td>160</td>
</tr>
<tr>
<td>Indexes</td>
<td>160</td>
</tr>
<tr>
<td>Legacy Databases</td>
<td>161</td>
</tr>
<tr>
<td>Distributed Transaction</td>
<td>161</td>
</tr>
<tr>
<td>6.5 Query, Set, Rows</td>
<td>162</td>
</tr>
<tr>
<td>select</td>
<td>162</td>
</tr>
<tr>
<td>Serializing Rows in Views</td>
<td>164</td>
</tr>
<tr>
<td>orderby, groupby, limitby, distinct</td>
<td>164</td>
</tr>
<tr>
<td>Logical Operators</td>
<td>165</td>
</tr>
<tr>
<td>count, delete, update</td>
<td>166</td>
</tr>
<tr>
<td>Expressions</td>
<td>166</td>
</tr>
<tr>
<td>update_record</td>
<td>166</td>
</tr>
<tr>
<td>6.6 One to Many Relation</td>
<td>167</td>
</tr>
<tr>
<td>Inner Joins</td>
<td>168</td>
</tr>
<tr>
<td>Left Outer Join</td>
<td>168</td>
</tr>
<tr>
<td>Grouping and Counting</td>
<td>169</td>
</tr>
<tr>
<td>6.7 How to see SQL</td>
<td>169</td>
</tr>
<tr>
<td>6.8 Exporting and Importing Data</td>
<td>170</td>
</tr>
<tr>
<td>CSV (one table at a time)</td>
<td>170</td>
</tr>
<tr>
<td>CSV (all tables at once)</td>
<td>170</td>
</tr>
<tr>
<td>CSV and remote Database Synchronization</td>
<td>171</td>
</tr>
<tr>
<td>HTML/XML (one table at a time)</td>
<td>173</td>
</tr>
<tr>
<td>6.9 Many to Many</td>
<td>173</td>
</tr>
<tr>
<td>6.10 Other Operators</td>
<td>175</td>
</tr>
<tr>
<td>like, upper, lower</td>
<td>175</td>
</tr>
<tr>
<td>year, month, day, hour, minutes, seconds</td>
<td>175</td>
</tr>
<tr>
<td>belongs</td>
<td>176</td>
</tr>
<tr>
<td>6.11 Caching Selects</td>
<td>176</td>
</tr>
<tr>
<td>6.12 Shortcuts</td>
<td>177</td>
</tr>
<tr>
<td>6.13 Self-Reference and Aliases</td>
<td>177</td>
</tr>
</tbody>
</table>
7 Forms and Validators

7.1 FORM
Hidden fields 185
keepvalues 186
onvalidation 186
Forms and redirection 187
Multiple forms per page 188
No self-submission 189

7.2 SQLFORM
Insert/Update/Delete SQLFORM 193
SQLFORM in HTML 194
SQLFORM and uploads 195
Storing the original filename 197
Removing the action file 198
Links to referencing records 198
Prepopulating the form 200
SQLFORM without database IO 200

7.3 SQLFORM.factory 201

7.4 Validators
Basic Validators 202
Database Validators 210
Custom Validators 211
Validators with Dependencies 212

7.5 Widgets 213

7.6 CRUD
Attributes 214
Messages 216
Methods 217

7.7 Custom form
CSS Conventions 220
Switch off errors 220

8 Access Control

8.1 Authentication 225
Email verification 227
Restrictions on registration 228
CONTENTS

CAPTCHA and reCAPTCHA 228
Customizing Auth 229
Renaming Auth tables 230
Alternate Login Methods 230

8.2 Authorization 233
Decorators 234
Combining requirements 235
Authorization and CRUD 235
Authorization and Downloads 236
Access control and Basic authentication 237
Settings and Messages 237

8.3 Central Authentication Service 241

9 Services 245

9.1 Rendering a dictionary 246
  HTML, XML, and JSON 246
  How it works 246
  Rendering Rows 247
  Custom Formats 248
  RSS 248
  CSV 250

9.2 Remote Procedure Calls 251
  XMLRPC 253
  JSONRPC 253
  AMFRPC 257

9.3 Low Level API and Other Recipes 259
  simplejson 259
  PyRTF 260
  ReportLab and PDF 260

9.4 Services and Authentication 261

10 Ajax Recipes 263

10.1 web2pyajax.html 263
10.2 jQuery Effects 268
  Conditional Fields in Forms 271
  Confirmation on Delete 272
10.3 The ajax Function 274
  Eval target 274
Auto-completion 275
Form Submission 277
Voting and Rating 278

11 Deployment Recipes 281
11.1 Setup Apache on Linux 284
11.2 Setup mod_wsgi on Linux 285
mod_wsgi and SSL 287
11.3 Setup mod_proxy on Linux 288
11.4 Start as Linux Daemon 290
11.5 Setup Apache and mod_wsgi on Windows 291
11.6 Start as Windows Service 293
11.7 Setup Lighttpd 294
11.8 Apache2 and mod_python in a shared hosting environment 295
11.9 Setup Cherokee with FastGGI 296
11.10 Setup PostgreSQL 297
11.11 Security Issues 298
11.12 Scalability Issues 299
   Sessions in Database 300
   Pound, a High Availability Load Balancer 301
   Cleanup Sessions 301
   Upload Files in Database 302
   Collecting Tickets 303
   Memcache 304
   Sessions in Memcache 305
   Removing Applications 305
11.13 Google App Engine 305

12 Other Recipes 309
12.1 Upgrading WEB2PY 309
12.2 Fetching a URL 310
12.3 Geocoding 310
12.4 Pagination 310
12.5 Streaming Virtual Files 311
12.6 httpserver.log and the log file format 312
12.7 Send an SMS 313
12.8 Twitter API 314
12.9 Jython 314
References 317
I am guilty! After publicly complaining about the existence of too many Python based web frameworks, after praising the merits of Django, Pylons, TurboGears, CherryPy, and web.py, after having used them professionally and taught them in University level courses, I could not resist and created one more: web2py.

Why did I commit such a crime? I did it because I felt trapped by existing choices and tempted by the beautiful features of the Python language. It all started with the need to convince my father to move away from Visual Basic and embrace Python as a development language for the Web. At the same time I was teaching a course on Python and Django at DePaul University. These two experiences made me realize how the beautiful features of those systems were hidden behind a steep learning curve. At the University for example we teach introductory programming using languages like Java and C++ but we do not get into networking issues until later courses. In many Universities students can graduate in Computer Science without ever seeing a Unix Bash Shell or editing an Apache configuration file. And yet these days to be an effective web developer you must know shell scripting, Apache, SQL, HTML, CSS, JavaScript, and Ajax. Knowing how to program in one
language is not enough to understand the intricacy and subtleties of the APIs exposed by the existing frameworks. Not to mention security.

_web2py_ started with the goal to drastically reduce the learning curve, incorporating everything needed into a single tool that is accessible via the web browser, collapsing the API to a minimum (only 12 core objects and functions), delegating all the security issues to the framework, and forcing developers to follow modern software engineering practices.

Most of the development work was done in the summer of 2007 while I was on vacation. Since _web2py_ was released many people have contributed by submitting patches to fix bugs and to add features. _web2py_ has evolved steadily since and yet it never broke backward compatibility. In fact, _web2py_ has a top-down design vs the bottom-up design of other frameworks. It is not built by adding layer upon layer. It is built from the user perspective and it has been constantly optimized inside in order to become faster and leaner, while always keeping backward compatibility. I am happy to say that today _web2py_ is one of the fastest web frameworks and also one of the smallest (the core libraries including the Database Abstraction Layer, the template language, and all the helpers amounts to about 300KB, the entire source code including sample applications and images amounts to less than 2.0MB).

Yes, I am guilty, but so are the growing number of users and contributors. Nevertheless, I feel, I am no more guilty than the creators of the other frameworks I have mentioned.

Finally, I would like to point out, I have already paid a price for my crime, since I have been condemned to spend my 2008 summer vacation writing this book and my 2009 summer vacations revising it.

This second edition describes many features added after the release of the first edition, including CRUD, Access Control, and Services.

I hope you, dear reader, understand I have done it for you: to free you from current web programming difficulties, and to allow you to express yourself more and better on the Web.
CHAPTER 1

INTRODUCTION

WEB2PY [1] is a free, open-source web framework for agile development of secure database-driven web applications; it is written in Python[2] and programmable in Python. WEB2PY is a full-stack framework, meaning that it contains all the components you need to build fully functional web applications.

WEB2PY is designed to guide a web developer to follow good software engineering practices, such as using the Model View Controller (MVC) pattern. WEB2PY separates the data representation (the model) from the data presentation (the view) and also from the application logic and workflow (the controller). WEB2PY provides libraries to help the developer design, implement, and test each of these three parts separately, and makes them work together.

WEB2PY is built for security. This means that it automatically addresses many of the issues that can lead to security vulnerabilities, by following well established practices. For example, it validates all input (to prevent injections), escapes all output (to prevent cross-site scripting), renames uploaded files (to prevent directory traversal attacks), and stores all session information
server side. **WEB2PY** leaves little choice to application developers in matters related to security.

**WEB2PY** includes a Database Abstraction Layer (DAL) that writes SQL [3] dynamically so that the developer does not have to. The DAL knows how to generate SQL transparently for SQLite [4], MySQL [6], PostgreSQL [5], MSSQL [7], FireBird [8], Oracle [9], IBM DB2 [10] and Informix [11]. The DAL can also generate function calls for Google BigTable when running on the Google App Engine (GAE) [12]. Once one or more database tables are defined, **WEB2PY** also generates a fully functional web-based database administration interface to access the database and the tables.

**WEB2PY** differs from other web frameworks in that it is the only framework to fully embrace the Web 2.0 paradigm, where the web is the computer. In fact, **WEB2PY** does not require installation or configuration; it runs on any architecture that can run Python (Windows, Windows CE, Mac OS X, iPhone, and Unix/Linux), and the development, deployment, and maintenance phases for the applications can be done via a local or remote web interface. **WEB2PY** runs with CPython (the C implementation) and/or Jython (the Java implementation), versions 2.4, 2.5 and 2.6 although "officially" only support 2.5 else we cannot guarantee backward compatibility for applications.

**WEB2PY** provides a ticketing system. If an error occurs, a ticket is issued to the user, and the error is logged for the administrator. **WEB2PY** is open source and released under the GPL2.0 license, but **WEB2PY** developed applications are not subject to any license constraint. As long as applications do not explicitly contain **WEB2PY** source code, they are not considered "derivative works". **WEB2PY** also allows the developer to bytecode-compile applications and distribute them as closed source, although they will require **WEB2PY** to run. The **WEB2PY** license includes an exception that allows web developers to ship their products with original pre-compiled **WEB2PY** binaries, without the accompanying source code.

Another feature of **WEB2PY**, is that we, its developers, commit to maintain backward compatibility in future versions. We have done so since the first release of **WEB2PY** in October, 2007. New features have been added and bugs have been fixed, but if a program worked with **WEB2PY** 1.0, that program will still work today.

Here are some examples of **WEB2PY** statements that illustrate its power and simplicity. The following code:

```python
1. db.define_table('person',
2.    Field('name', 'string'),
3.    Field('image', 'upload'))
```
creates a database table called "person" with two fields: "name", a string; and "image", something that needs to be uploaded (the actual image). If the table already exists but does not match this definition, it is altered appropriately.

Given the table defined above, the following code:

```python
form = SQLFORM(db.person)
```

creates an insert form for this table that allows users to upload images.

The following statement:

```python
if form.accepts(request.vars, session):
    pass
```

validates a submitted form, renames the uploaded image in a secure way, stores the image in a file, inserts the corresponding record in the database, prevents double submission, and eventually modifies the form itself by adding error messages if the data submitted by the user does not pass validation.

1.1 Principles

Python programming typically follows these basic principles:

- Don’t repeat yourself (DRY).
- There should be only one way of doing things.
- Explicit is better than implicit.

**Web2Py** fully embraces the first two principles by forcing the developer to use sound software engineering practices that discourage repetition of code. **Web2Py** guides the developer through almost all the tasks common in web application development (creating and processing forms, managing sessions, cookies, errors, etc.).

**Web2Py** differs from other frameworks with regard to the third principle, which sometimes conflicts with the other two. In particular, **Web2Py** automatically imports its own modules and instantiates its global objects (request, response, session, cache, T) and this is done "under the hood". To some this may appear as magic, but it should not. **Web2Py** is trying to avoid the annoying characteristic of other frameworks that force the developer to import the same modules at the top of every model and controller.

**Web2Py**, by importing its own modules, saves time and prevents mistakes, thus following the spirit of "don’t repeat yourself" and "there should be only one way of doing things".

If the developer wishes to use other Python modules or third-party modules, those modules must be imported explicitly, as in any other Python program.
1.2 Web Frameworks

At its most fundamental level, a web application consists of a set of programs (or functions) that are executed when a URL is visited. The output of the program is returned to the visitor and rendered by the browser.

The two classic approaches for developing web applications are:

- Generating HTML [13, 14] programmatically and embedding HTML as strings into computer code.

- Embedding pieces of code into HTML pages.

The first model is the one followed, for example, by early CGI scripts. The second model is followed, for example, by PHP [15] (where the code is in PHP, a C-like language), ASP (where the code is in Visual Basic), and JSP (where the code is in Java).

Here we present an example of a PHP program that, when executed, retrieves data from a database and returns an HTML page showing the selected records:

```php
<html><body><h1>Records</h1><?
mysql_connect(localhost,username,password);
@mysql_select_db(database) or die( "Unable to select database");
$query="SELECT * FROM contacts";
$result=mysql_query($query);
mysql_close();
$i=0;
while ($i < mysql_numrows($result)) {
  $name=mysql_result($result,$i,"name");
  $phone=mysql_result($result,$i,"phone");
  echo "<b>$name</b><br>Phone:$phone<br /><br /><hr /><br ">
  $i++;
}
?></body></html>
```

The problem with this approach is that code is embedded into HTML, but this very same code also needs to generate additional HTML and to generate SQL statements to query the database, entangling multiple layers of the application and making it difficult to read and maintain. The situation is even worse for Ajax applications, and the complexity grows with the number of pages (files) that make up the application.

The functionality of the above example can be expressed in web2py with two lines of Python code:

```python
def index():
    return HTML(BODY(H1('Records'), db().select(db.contacts.ALL)))
```