

# Pro Django



Marty Alchin

## **Pro Django**

**Copyright © 2009 by Marty Alchin**

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-1047-4

ISBN-13 (electronic): 978-1-4302-1048-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Java™ and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the US and other countries. Apress, Inc., is not affiliated with Sun Microsystems, Inc., and this book was written without endorsement from Sun Microsystems, Inc.

Lead Editor: Steve Anglin

Technical Reviewers: Jacob Kaplan-Moss, George Vilches

Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Project Manager: Richard Dal Porto

Copy Editors: Liz Welch, Ami Knox

Associate Production Director: Kari Brooks-Copony

Production Editor: Laura Cheu

Compositor: Kinetic Publishing Services, LLC

Proofreader: April Eddy

Indexer: Julie Grady

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit <http://www.apress.com>.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at <http://www.apress.com/info/bulksales>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com>. You will need to answer questions pertaining to this book in order to successfully download the code.

# Contents at a Glance

About the Author.....	xiii
Acknowledgments .....	xv
Preface.....	xvii
Introduction.....	xix
<b>CHAPTER 1</b> Understanding Django.....	1
<b>CHAPTER 2</b> Django Is Python .....	13
<b>CHAPTER 3</b> Models .....	45
<b>CHAPTER 4</b> URLs and Views.....	91
<b>CHAPTER 5</b> Forms .....	113
<b>CHAPTER 6</b> Templates .....	133
<b>CHAPTER 7</b> Handling HTTP .....	163
<b>CHAPTER 8</b> Backend Protocols .....	183
<b>CHAPTER 9</b> Common Tools .....	213
<b>CHAPTER 10</b> Coordinating Applications .....	231
<b>CHAPTER 11</b> Enhancing Applications.....	253
<b>APPENDIX</b> Contributing to Django.....	279
<b>INDEX</b> .....	285



# Contents

About the Author.....	xiii
Acknowledgments .....	xv
Preface.....	xvii
Introduction.....	xix
<b>CHAPTER 1    Understanding Django .....</b>	<b>1</b>
Philosophy.....	1
Django's Interpretation of the MVC Pattern.....	2
Loose Coupling.....	5
Don't Repeat Yourself (DRY).....	5
A Focus on Readability .....	6
Failing Loudly .....	6
Community.....	8
Management of the Framework .....	9
News and Resources.....	10
Reusable Applications.....	10
Getting Help .....	11
Now What?.....	12
<b>CHAPTER 2    Django Is Python .....</b>	<b>13</b>
How Python Builds Classes .....	13
Building a Class Programmatically .....	14
Metaclasses Change It Up .....	15
Using a Base Class with a Metaclass.....	16
Declarative Syntax.....	17
Common Duck-Typing Protocols .....	19
Callables .....	20
Dictionaries.....	21
Files .....	22
Iterables.....	23
Sequences.....	25

Augmenting Functions .....	25
Excess Arguments .....	26
Decorators .....	28
Descriptors .....	34
__get__(self, instance, owner) .....	35
__set__(self, instance, value) .....	35
Keeping Track of Instance Data .....	36
Introspection .....	36
Common Class and Function Attributes .....	37
Identifying Object Types .....	37
Function Signatures .....	39
Docstrings .....	40
Applied Techniques .....	41
Tracking Subclasses .....	41
A Simple Plugin Architecture .....	42
Now What? .....	44

<b>CHAPTER 3</b>	<b>Models .....</b>	<b>45</b>
	How Django Processes Model Classes .....	46
	Setting Attributes on Models .....	46
	Getting Information About Models .....	47
	Class Information .....	47
	Field Definitions .....	48
	Primary Key Fields .....	49
	Configuration Options .....	50
	Accessing the Model Cache .....	52
	Using Model Fields .....	57
	Common Field Attributes .....	57
	Common Field Methods .....	60
	Subclassing Fields .....	62
	Deciding Whether to Invent or Extend .....	62
	Performing Actions During Model Registration .....	63
	Altering Data Behavior .....	64
	Controlling Database Behavior .....	68
	Dealing with Files .....	71
	get_directory_name(self) .....	71
	get_filename(self, filename) .....	72
	generate_filename(self, instance, filename) .....	72

save_form_data(self, instance, data) .....	73
delete_file(self, instance, sender) .....	73
attr_class .....	74
Customizing the File Class .....	74
Signals .....	76
class_prepared .....	77
pre_init and post_init .....	78
pre_save and post_save .....	79
pre_delete and post_delete .....	80
post_syncdb .....	80
Applied Techniques .....	82
Loading Attributes on Demand .....	82
Creating Models Dynamically at Runtime .....	87
Now What? .....	90

## CHAPTER 4    **URLs and Views** .....

URLs .....	91
Standard URL Configuration .....	92
Resolving URLs to Views .....	94
Resolving Views to URLs .....	95
Views .....	97
Templates Break It Up a Bit .....	98
Anatomy of a View .....	98
Writing Views to Be Generic .....	99
View Decorators .....	101
Using an Object As a View .....	107
Applied Techniques .....	109
Dual-Format Decorator .....	109
Now What? .....	111

## CHAPTER 5    **Forms** .....

Declaring and Identifying Fields .....	113
Binding to User Input .....	114
Validating Input .....	115
Custom Fields .....	117
Validation .....	117
Controlling Widgets .....	118

Defining HTML Behavior .....	119
Custom Widgets .....	119
Customizing Form Markup .....	123
Accessing Individual Fields .....	124
Customizing the Display of Errors .....	124
Applied Techniques .....	125
Pending and Resuming Forms .....	125
Now What? .....	132

<b>CHAPTER 6</b>	<b>Templates</b> .....	133
	What Makes a Template .....	133
	Exceptions .....	134
	The Process at Large .....	135
	Content Tokens .....	135
	Parsing Tokens into Nodes .....	136
	Template Nodes .....	137
	Rendering Templates .....	138
	Context .....	138
	Simple Variable Resolution .....	139
	Complex Variable Lookup .....	140
	Including Aspects of the Request .....	141
	Retrieving Templates .....	141
	django.template.loader.get_template(template_name) .....	141
	django.template.loader.select_template(template_name_list) .....	142
	Shortcuts to Load and Render Templates .....	142
	Adding Features for Templates .....	143
	Setting Up the Package .....	143
	Variable Filters .....	144
	Template Tags .....	146
	Adding Features to All Templates .....	148
	Applied Techniques .....	148
	Embedding Another Template Engine .....	148
	Enabling User-Submitted Themes .....	152
	Now What? .....	162

<b>CHAPTER 7</b>	<b>Handling HTTP</b> .....	163
	Requests and Responses .....	163
	HttpRequest .....	163
	HttpResponse .....	169



Writing HTTP Middleware	174
MiddlewareClass.process_request(self, request)	174
MiddlewareClass.process_view(self, request, view, args, kwargs)	174
MiddlewareClass.process_response(self, request, response)	175
MiddlewareClass.process_exception(self, request, exception)	175
Deciding Between Middleware and View Decorators	176
HTTP-Related Signals	178
django.core.signals.request_started	178
django.core.signals.request_finished	178
django.core.signals.got_request_exception	178
Applied Techniques	178
Signing and Validating Cookies	179
Now What?	181

<b>CHAPTER 8 Backend Protocols</b>	183
Database Access	183
django.db.backends	184
Creation of New Structures	189
Introspection of Existing Structures	191
DatabaseClient	192
DatabaseError and IntegrityError	193
Authentication	193
get_user(user_id)	193
authenticate(**credentials)	193
Storing User Information	194
Files	194
The Base File Class	194
Handling Uploads	196
Storing Files	198
Session Management	199
Caching	201
Specifying a Backend	201
Using the Cache Manually	202
Template Loading	203
load_template_source(template_name, template_dirs=None)	203
load_template_source.is_usable	204
Context Processors	204

Applied Techniques .....	205
Loading Templates Using a Different Engine .....	205
Scanning Incoming Files for Viruses .....	210
Now What? .....	212

## **CHAPTER 9 Common Tools** .....

Core Exceptions .....	213
django.core.exceptions.ImproperlyConfigured .....	213
django.core.exceptions.MiddlewareNotUsed .....	214
django.core.exceptions.MultipleObjectsReturned .....	214
django.core.exceptions.ObjectDoesNotExist .....	215
django.core.exceptions.PermissionDenied .....	215
django.core.exceptions.SuspiciousOperation .....	216
django.core.exceptions.ViewDoesNotExist .....	216
Text Modification .....	217
get_text_list(items, last_word='or') .....	217
javascript_quote(s, quote_double_quotes=False) .....	217
normalize_newlines(text) .....	217
phone2numeric(phone) .....	218
recapitalize(text) .....	218
smart_split(text) .....	218
truncate_words(s, num) .....	219
truncate_html_words(s, num) .....	219
wrap(text, width) .....	219
Data Structures .....	220
django.utils.datastructures.MergeDict .....	220
django.utils.datastructures.MultiValueDict .....	221
django.utils.datastructures.SortedDict .....	222
Functional Utilities .....	222
django.utils.functional.curry .....	222
django.utils.functional.memoize .....	223
django.utils.functional.wraps .....	225
Signals .....	226
How It Works .....	226
Defining a Signal .....	226
Sending a Signal .....	227
Capturing Return Values .....	227
Defining a Listener .....	227

Registering Listeners . . . . .	228
Forcing Strong References . . . . .	228
Now What? . . . . .	229
<b>CHAPTER 10 Coordinating Applications . . . . .</b>	<b>231</b>
Contacts . . . . .	231
contacts.models.Contact . . . . .	232
contacts.forms.UserEditorForm . . . . .	233
contacts.forms.ContactEditorForm . . . . .	234
contacts.views.edit_contact . . . . .	235
Admin Configuration . . . . .	238
URL Configuration . . . . .	238
Real Estate Properties . . . . .	241
properties.models.Property . . . . .	241
properties.models.Feature . . . . .	246
properties.models.PropertyFeature . . . . .	246
properties.models.InterestedParty . . . . .	246
Admin Configuration . . . . .	247
URL Configuration . . . . .	251
Now What? . . . . .	252
<b>CHAPTER 11 Enhancing Applications . . . . .</b>	<b>253</b>
Recording the Current User . . . . .	253
The Thread-Local Approach—Useful but Dangerous . . . . .	254
The Admin Approach . . . . .	256
Introducing the CurrentUserField . . . . .	257
Keeping Track of CurrentUserField Instances . . . . .	257
The CurrentUserMiddleware . . . . .	259
Performance Considerations . . . . .	261
Keeping Historical Records . . . . .	263
Intended Usage . . . . .	264
Overview of the Process . . . . .	266
Step 1: Copy the Model . . . . .	267
Step 2: Register Signal Handlers . . . . .	272
Step 3: Assign a Manager . . . . .	272
Now What? . . . . .	278

APPENDIX

Contributing to Django.....279

Reporting a Ticket .....279

Supplying a Patch .....279

Writing Tests.....280

Writing Documentation.....280

Development Sprints.....281

Publishing Code.....281

Releasing an Application .....282

INDEX .....

285

# About the Author

■ **MARTY ALCHIN** is a professional programmer with a passion for the Web. Over the past two and a half years, he has developed and released a few Django applications and a significant improvement to Django's file storage handling.

Raised in the wild by a pack of mechanical engineers, Marty learned at a young age the importance of knowing how things work and how to improve them. When not coding for work, he goes by the name Gulopine and codes for fun. He keeps a blog at <http://martyalchin.com/>, where much of this code is announced and described.



# Acknowledgments

I can't imagine anyone taking on a project like this alone. In the year and a half since I first considered putting my thoughts on paper, no one has been more supportive than my beautiful wife, Angel. Without her, I'd be lost and confused, mumbling incoherently about declarative metaclass implementations. There are no words to express how much help she's been throughout the process.

I'd also like to thank George Vilches for stepping up to take on a book he hadn't been involved with from the beginning. He's been an amazing asset, going well beyond what was required of him to make sure this book is as good as we could possibly make it.

Of course, the *Lawrence Journal-World* and its Internet division are to thank for the Django Web framework's existence and for its release to the public, which made all of this possible in the first place. I don't expect they had any idea how far it would go when designing and releasing it. I have a feeling this is far from the end.

In fact, the entire community that surrounds Django has fueled me in more ways than I can explain. It's because of people like you that I chose to take on this challenge, and it's the thought of a greater community that keeps me going. Thank you.





# Preface

**P**rogramming has always been equal parts art and science. It's easy to see the science in teaching computers how to do things, but once that's out of the way, we often try to embrace the artistic side. We spend our first few years learning to make code functional and the rest of our careers trying to make it beautiful.

Django started its life in much the same way, serving the day-to-day needs of a local news organization. In the years since its first public release, Django itself has grown more elegant and has helped its adopters to write more elegant code for their own applications.

This focus on beauty isn't unique to Django. Most Python applications strive for a notion of being "Pythonic"—an unwritten ideal that embodies the nature and spirit of the Python language itself. Having a vague goal like that may seem problematic; after all, how do you know when you've succeeded? Ironically, that's the point: there is no finish line. There's not even a measuring stick to tell you how close you are to achieving your goal.

The true goal is the journey itself, the lessons learned along the way, the discoveries that open your eyes to new ideas. Python includes a number of tools that make this process quite interesting, especially for those programmers coming from other languages. Django builds on that toolset, adding its own techniques for easing the burden on other programmers, making it easy to produce more beautiful code all around.

I first got started with Django shortly after it completed its "magic removal" phase, which was a long process of making the framework more Pythonic overall. I was new to Python at the time, and reading about the process and the ideals that encouraged it caused me to dig deeper into what made Django work. I was fascinated by the richness of the toolset at my disposal and quickly began my own journey of discovery.

What fascinated me most was how few people knew about some of the tricks that can be used to encourage Pythonic code for programmers using the framework. Every time I showed a new trick to someone, I joked that I could write a book about what I've learned so far. After several months of doing so—and several people encouraging me to drop the joke and do it for real—I finally took the plunge and contacted Apress.

I'm not interested in making a fortune with this book. My goal has always been to help more people understand the many tools available with Python and Django, in hopes that they too can have enriching journeys of their own. I hope this book will help bring Django to new people and new places, where it might have been previously considered inappropriate.

Those of us working with Django are often called Djangonauts with good reason. The "-naut" suffix has been used historically to represent sailors and is the same concept as in the word "nautical." More generally, it often refers to those who sail into the unknown, such as astronauts and cosmonauts. It represents explorers and adventurers, those people brave enough to challenge what they knew before and dare to discover new things and new places.

I am a Djangonaut. What follows is my journey thus far.



# Introduction

**P***ro Django* represents two and a half years of accumulated knowledge in Python and Django, designed to educate readers who are already familiar with both topics and would like to take them further than they had previously done. You will learn a wide range of advanced techniques available in both Python and Django, along with tips on how to use them to achieve advanced functionality.

This book is designed to be both a narrative to be read from start to finish and a general reference to be searched for specific information. Since you may not know what to look for or where to find it yet, feel free to read through the book first, then keep it handy for refreshing your memory as necessary.

## What This Book Is Not

There are plenty of resources available for learning Python and Django, so this book does not strive to teach the basics. For readers new to Python, I highly recommend *Dive Into Python* by Mark Pilgrim (Apress, 2004). For learning Django, I'd recommend *The Definitive Guide to Django: Web Development Done Right* by Adrian Holovaty and Jacob Kaplan-Moss (Apress, 2006). Additionally, *Practical Django Projects* by James Bennett (Apress, 2008) is an excellent resource for general application development.

## Who This Book Is For

Because *Pro Django* doesn't dwell on introductory details, readers will be expected to have experience with both Python and Django. If you're new to either subject, please consider one of the books mentioned in the previous section before trying to tackle this book.

Even if you've only experimented on your own without launching a full site yet, a basic familiarity should be sufficient. You don't need to be an expert to start reading *Pro Django*, but you might be by the time you finish.

## Interpreting Code Samples

*Pro Django* uses a simple format, interleaving explanations of Python's and Django's available features with code that demonstrates their use in the real world. There are two types of code samples used, which differ in how they should be executed.

Python's interactive interpreter is a great way to test out small pieces of code and see how it works in a variety of situations. Lines of code intended for use in that environment will always be prefixed with three characters: three greater-than signs (`>>>`) or three periods (`...`). Lines with greater-than signs are the outermost block of code, while the period-prefixed lines are indented at least one level. The three initial characters are also followed by a space. These

first four characters are not typed into the interactive interpreter directly; they simply mimic what the interpreter itself looks like by reproducing its output.

A line started with three periods but containing no other text indicates that you should simply press Enter on a blank line in the interpreter. This completes any open code blocks, bringing you back to the `>>>` prompt. Any lines that don't begin with either `>>>` or `...` represent the output of the code or the result of the previous expression.

```
>>> import django
>>> print django.get_version()
u'1.0-final'
```

The first line of an interactive example will always begin with `>>>`; everything else is code that should be written in a file and executed as part of a running Django application. The surrounding text will indicate what file the code should be placed in and how it will execute.

## Prerequisites

*Pro Django* is written for Django 1.0, which was released on September 3, 2008. That release or a more recent checkout from the Django code repository is required for the code samples to work properly. Since Django in turn relies on Python, these examples also assume a working Python environment of version 2.3 or higher.