Pro Django

Marty Alchin

Apress®

Pro Django

Copyright © 2009 by Marty Alchin

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

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`, 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, 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 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

Contents

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

Programming 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

Pro 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.