
Table of Contents

Preface.....	xv
Prerequisites and Assumptions.....	xxi
Acknowledgments.....	xxvii

Part I. The Basics of TDD and Django

1. Getting Django Set Up Using a Functional Test.....	3
Obey the Testing Goat! Do Nothing Until You Have a Test	3
Getting Django Up and Running	6
Starting a Git Repository	8
2. Extending Our Functional Test Using the unittest Module.....	13
Using a Functional Test to Scope Out a Minimum Viable App	13
The Python Standard Library's unittest Module	16
Implicit waits	18
Commit	18
3. Testing a Simple Home Page with Unit Tests.....	21
Our First Django App, and Our First Unit Test	22
Unit Tests, and How They Differ from Functional Tests	22
Unit Testing in Django	23
Django's MVC, URLs, and View Functions	24
At Last! We Actually Write Some Application Code!	26
urls.py	27
Unit Testing a View	30
The Unit-Test/Code Cycle	31
4. What Are We Doing with All These Tests?.....	35
Programming Is like Pulling a Bucket of Water up from a Well	36

Using Selenium to Test User Interactions	37
The “Don’t Test Constants” Rule, and Templates to the Rescue	40
Refactoring to Use a Template	40
On Refactoring	44
A Little More of Our Front Page	45
Recap: The TDD Process	47
5. Saving User Input.....	51
Wiring Up Our Form to Send a POST Request	51
Processing a POST Request on the Server	54
Passing Python Variables to Be Rendered in the Template	55
Three Strikes and Refactor	59
The Django ORM and Our First Model	60
Our First Database Migration	62
The Test Gets Surprisingly Far	63
A New Field Means a New Migration	64
Saving the POST to the Database	65
Redirect After a POST	68
Better Unit Testing Practice: Each Test Should Test One Thing	68
Rendering Items in the Template	69
Creating Our Production Database with migrate	71
6. Getting to the Minimum Viable Site.....	77
Ensuring Test Isolation in Functional Tests	77
Running Just the Unit Tests	80
Small Design When Necessary	81
YAGNI!	82
REST	82
Implementing the New Design Using TDD	83
Iterating Towards the New Design	86
Testing Views, Templates, and URLs Together with the Django Test Client	87
A New Test Class	88
A New URL	88
A New View Function	89
A Separate Template for Viewing Lists	90
Another URL and View for Adding List Items	92
A Test Class for New List Creation	93
A URL and View for New List Creation	94
Removing Now-Redundant Code and Tests	95
Pointing Our Forms at the New URL	96
Adjusting Our Models	97
A Foreign Key Relationship	99

Adjusting the Rest of the World to Our New Models	100
Each List Should Have Its Own URL	102
Capturing Parameters from URLs	103
Adjusting new_list to the New World	104
One More View to Handle Adding Items to an Existing List	105
Beware of Greedy Regular Expressions!	106
The Last New URL	106
The Last New View	107
But How to Use That URL in the Form?	108
A Final Refactor Using URL includes	110

Part II. Web Development Sine Qua Nons

7. Prettification: Layout and Styling, and What to Test About It.	115
What to Functionally Test About Layout and Style	115
Prettification: Using a CSS Framework	118
Django Template Inheritance	120
Integrating Bootstrap	121
Rows and Columns	122
Static Files in Django	123
Switching to StaticLiveServerCase	124
Using Bootstrap Components to Improve the Look of the Site	125
Jumbotron!	125
Large Inputs	125
Table Styling	126
Using Our Own CSS	126
What We Glossed Over: collectstatic and Other Static Directories	127
A Few Things That Didn't Make It	130
8. Testing Deployment Using a Staging Site.	131
TDD and the Danger Areas of Deployment	132
As Always, Start with a Test	133
Getting a Domain Name	135
Manually Provisioning a Server to Host Our Site	136
Choosing Where to Host Our Site	136
Spinning Up a Server	137
User Accounts, SSH, and Privileges	137
Installing Nginx	138
Configuring Domains for Staging and Live	139
Using the FT to Confirm the Domain Works and Nginx Is Running	139
Deploying Our Code Manually	140

Adjusting the Database Location	141
Creating a Virtualenv	142
Simple Nginx Configuration	144
Creating the Database with migrate	147
Getting to a Production-Ready Deployment	148
Switching to Gunicorn	148
Getting Nginx to Serve Static Files	149
Switching to Using Unix Sockets	150
Switching DEBUG to False and Setting ALLOWED_HOSTS	151
Using Upstart to Make Sure Gunicorn Starts on Boot	151
Saving Our Changes: Adding Gunicorn to Our requirements.txt	152
Automating	152
“Saving Your Progress”	156
9. Automating Deployment with Fabric.....	157
Breakdown of a Fabric Script for Our Deployment	158
Trying It Out	162
Deploying to Live	163
Nginx and Gunicorn Config Using sed	165
Git Tag the Release	166
Further Reading	166
10. Input Validation and Test Organisation.....	169
Validation FT: Preventing Blank Items	169
Skipping a Test	170
Splitting Functional Tests out into Many Files	171
Running a Single Test File	174
Fleshing Out the FT	174
Using Model-Layer Validation	175
Refactoring Unit Tests into Several Files	175
Unit Testing Model Validation and the self.assertRaises Context Manager	177
A Django Quirk: Model Save Doesn’t Run Validation	178
Surfacing Model Validation Errors in the View	178
Checking Invalid Input Isn’t Saved to the Database	181
Django Pattern: Processing POST Requests in the Same View as Renders the Form	183
Refactor: Transferring the new_item Functionality into view_list	184
Enforcing Model Validation in view_list	186
Refactor: Removing Hardcoded URLs	187
The {% url %} Template Tag	188

Using <code>get_absolute_url</code> for Redirects	188
11. A Simple Form.	193
Moving Validation Logic into a Form	193
Exploring the Forms API with a Unit Test	194
Switching to a Django <code>ModelForm</code>	195
Testing and Customising Form Validation	196
Using the Form in Our Views	198
Using the Form in a View with a GET Request	198
A Big Find and Replace	201
Using the Form in a View That Takes POST Requests	203
Adapting the Unit Tests for the <code>new_list</code> View	203
Using the Form in the View	204
Using the Form to Display Errors in the Template	205
Using the Form in the Other View	205
A Helper Method for Several Short Tests	206
Using the Form's Own Save Method	208
12. More Advanced Forms.	211
Another FT for Duplicate Items	211
Preventing Duplicates at the Model Layer	212
A Little Digression on Queryset Ordering and String Representations	214
Rewriting the Old Model Test	216
Some Integrity Errors Do Show Up on Save	217
Experimenting with Duplicate Item Validation at the Views Layer	218
A More Complex Form to Handle Uniqueness Validation	219
Using the Existing List Item Form in the List View	221
13. Dipping Our Toes, Very Tentatively, into JavaScript.	225
Starting with an FT	225
Setting Up a Basic JavaScript Test Runner	226
Using jQuery and the Fixtures Div	229
Building a JavaScript Unit Test for Our Desired Functionality	232
Javascript Testing in the TDD Cycle	234
Columbo Says: Onload Boilerplate and Namespacing	234
A Few Things That Didn't Make It	235
14. Deploying Our New Code.	237
Staging Deploy	237
Live Deploy	237
What to Do If You See a Database Error	238

Part III. More Advanced Topics

15. User Authentication, Integrating Third-Party Plugins, and Mocking with JavaScript.	241
Mozilla Persona (BrowserID)	242
Exploratory Coding, aka “Spiking”	242
Starting a Branch for the Spike	243
Frontend and JavaScript Code	243
The Browser-ID Protocol	244
The Server Side: Custom Authentication	245
De-spiking	251
A Common Selenium Technique: Explicit Waits	253
Reverting Our Spiked Code	255
JavaScript Unit Tests Involving External Components: Our First Mocks!	256
Housekeeping: A Site-Wide Static Files Folder	256
Mocking: Who, Why, What?	257
Namespacing	258
A Simple Mock to Unit Tests Our initialize Function	258
More Advanced Mocking	264
Checking Call Arguments	267
QUnit setup and teardown, Testing Ajax	268
More Nested Callbacks! Testing Asynchronous Code	272
16. Server-Side Authentication and Mocking in Python.....	277
A Look at Our Spiked Login View	277
Mocking in Python	278
Testing Our View by Mocking Out authenticate	278
Checking the View Actually Logs the User In	281
De-spiking Our Custom Authentication Backend: Mocking Out an Internet Request	285
1 if = 1 More Test	286
Patching at the Class Level	287
Beware of Mocks in Boolean Comparisons	290
Creating a User if Necessary	291
The get_user Method	291
A Minimal Custom User Model	293
A Slight Disappointment	295
Tests as Documentation	296
Users Are Authenticated	297
The Moment of Truth: Will the FT Pass?	298

Finishing Off Our FT, Testing Logout	299
17. Test Fixtures, Logging, and Server-Side Debugging.....	303
Skipping the Login Process by Pre-creating a Session	303
Checking It Works	305
The Proof Is in the Pudding: Using Staging to Catch Final Bugs	306
Setting Up Logging	307
Fixing the Persona Bug	309
Managing the Test Database on Staging	311
A Django Management Command to Create Sessions	311
Getting the FT to Run the Management Command on the Server	312
An Additional Hop via subprocess	314
Baking In Our Logging Code	317
Using Hierarchical Logging Config	318
Wrap-Up	320
18. Finishing “My Lists”: Outside-In TDD.....	323
The Alternative: “Inside Out”	323
Why Prefer “Outside-In”?	323
The FT for “My Lists”	324
The Outside Layer: Presentation and Templates	325
Moving Down One Layer to View Functions (the Controller)	326
Another Pass, Outside-In	327
A Quick Restructure of the Template Inheritance Hierarchy	327
Designing Our API Using the Template	328
Moving Down to the Next Layer: What the View Passes to the Template	329
The Next “Requirement” from the Views Layer: New Lists Should Record	
Owner	330
A Decision Point: Whether to Proceed to the Next Layer with a Failing Test	331
Moving Down to the Model Layer	331
Final Step: Feeding Through the .name API from the Template	333
19. Test Isolation, and “Listening to Your Tests”.....	337
Revisiting Our Decision Point: The Views Layer Depends on Unwritten	
Models Code	337
A First Attempt at Using Mocks for Isolation	338
Using Mock side_effects to Check the Sequence of Events	339
Listen to Your Tests: Ugly Tests Signal a Need to Refactor	341
Rewriting Our Tests for the View to Be Fully Isolated	342
Keep the Old Integrated Test Suite Around as a Sanity Check	342
A New Test Suite with Full Isolation	343
Thinking in Terms of Collaborators	343

Moving Down to the Forms Layer	347
Keep Listening to Your Tests: Removing ORM Code from Our Application	348
Finally, Moving Down to the Models Layer	351
Back to Views	353
The Moment of Truth (and the Risks of Mocking)	354
Thinking of Interactions Between Layers as “Contracts”	355
Identifying Implicit Contracts	356
Fixing the Oversight	357
One More Test	358
Tidy Up: What to Keep from Our Integrated Test Suite	359
Removing Redundant Code at the Forms Layer	359
Removing the Old Implementation of the View	360
Removing Redundant Code at the Forms Layer	361
Conclusions: When to Write Isolated Versus Integrated Tests	362
Let Complexity Be Your Guide	363
Should You Do Both?	363
Onwards!	363
20. Continuous Integration (CI).....	365
Installing Jenkins	365
Configuring Jenkins Security	367
Adding Required Plugins	368
Setting Up Our Project	369
First Build!	371
Setting Up a Virtual Display so the FTs Can Run Headless	372
Taking Screenshots	374
A Common Selenium Problem: Race Conditions	378
Running Our QUnit JavaScript Tests in Jenkins with PhantomJS	381
Installing node	382
Adding the Build Steps to Jenkins	383
More Things to Do with a CI Server	384
21. The Token Social Bit, the Page Pattern, and an Exercise for the Reader.....	387
An FT with Multiple Users, and addCleanup	387
Implementing the Selenium Interact/Wait Pattern	389
The Page Pattern	390
Extend the FT to a Second User, and the “My Lists” Page	393
An Exercise for the Reader	395
22. Fast Tests, Slow Tests, and Hot Lava.....	397
Thesis: Unit Tests Are Superfast and Good Besides That	398
Faster Tests Mean Faster Development	398

The Holy Flow State	399
Slow Tests Don't Get Run as Often, Which Causes Bad Code	399
We're Fine Now, but Integrated Tests Get Slower Over Time	399
Don't Take It from Me	399
And Unit Tests Drive Good Design	400
The Problems with "Pure" Unit Tests	400
Isolated Tests Can Be Harder to Read and Write	400
Isolated Tests Don't Automatically Test Integration	400
Unit Tests Seldom Catch Unexpected Bugs	400
Mocky Tests Can Become Closely Tied to Implementation	400
But All These Problems Can Be Overcome	401
Synthesis: What Do We Want from Our Tests, Anyway?	401
Correctness	401
Clean, Maintainable Code	401
Productive Workflow	402
Evaluate Your Tests Against the Benefits You Want from Them	402
Architectural Solutions	402
Ports and Adapters/Hexagonal/Clean Architecture	403
Functional Core, Imperative Shell	403
Conclusion	404
Obey the Testing Goat!.....	407
A. PythonAnywhere.....	409
B. Django Class-Based Views.....	413
C. Provisioning with Ansible.....	423
D. Testing Database Migrations.....	427
E. What to Do Next.....	433
F. Cheat Sheet.....	437
G. Bibliography.....	441
Index.....	443