Index

A
assertLessEqual method, 31
assertNotIsInstance method, 30
assertRaises method, 20–22, 24–25, 27, 32
assertTrue method, 27–28
AttributeError, 24
automated builds, 168–174, 176
automated smoke tests, 183, 195–200
automation (of testing process)
  adding Coverage and PyLint reports, 175–176, 177
  automating nearly everything, 206–207
  creating Paver task, 164–165
  defining default build, 166–167
  executing Paver tasks, 165–166
  installation of Jenkins, 169–174
  installation of Paver, 164
  making build status highly visible, 176, 178–180
  overview, 163–164
  setting up automated builds, 168–169

B
b (break) (PDB key command), 37
bank account example
  analyzing cProfile output, 136–143
  creating class for, 73
  creating Paver task, 164–165
  customizing PyLint’s output, 150–154
  defining default build, 166–168
  defining Paver tasks, 185–187
  defining steps, 92–94
  delivering finished article, 98–102
  developing more of features, 95–98
  driving more features with tests, 75–77
  enabling archiving on Jenkins, 188–189
  example application stack, 196–197
  executing Paver tasks, 165–166
  fail fast feature, 112–114
  ignoring Coverage option, 160–161
  implementing code, 94–95
  implementing keywords with Robot, 117–119
  implementing smoke tests, 198–200
  incorporating packaging into build, 187
  making your test pass, 75
  Nosetest integration, 114–115
  producing HTML/XML report using Coverage, 157–159
  restricting coverage to specific package option, 159
  running cProfile session, 136–141
  running Robot tests, 119–123
  setting minimum coverage threshold using Coverage, 159
  step files, 88–89
  tagging, 108–112
  telling PyLint to ignore errors, 153–154
  using Coverage, 155–161
  using PyLint, 146–149
  wrapping up the task, 77–82
  writing failing acceptance test, 90–92
  writing failing test for, 74–75
bank_app.py file, 94, 95, 96, 98, 99, 146, 149, 150
bash language, 11, 207
bash plug-in, 11
bash settings, 173
.bashrc, 164, 173
batch development process, 5
behavior driven development (BDD), 1–2, 6, 86, 91, 102, 107, 108. See also Lettuce
BBD Testing Framework
blank line standard, 23
blocking I/O calls, 126
Brew (package manager), 127, 128, 169
brittle test suites, 17
BSkyB, 67
bugs. See debugging
Build a Free-style Software Project (Jenkins), 170, 171
Build Monitor Plugin (Jenkins), 178
build process
adding code coverage statistics as part of, 145, 154, 155
building packaging of app into, 187
as key feature of many teams, 2
running doctests as part of, 62
running entire test suite as part of, 107
running Lettuce and unit tests together using Noisetest as useful for creating, 114
setting up to ignore certain tests, 111
specifying default jobs for, 206
using machine with decent specification to handle, 169
build status visibility, 176, 178–180

C
Calculate class, 16, 18, 29, 57, 61, 63
calculate_test.py test file, 18, 20, 41, 42, 44
calculator program example
  add method for, 70
  adding doctests to method, 57–58
  advanced doctest usage, 61–62
  checking exception handling with assertRaises, 20–22
  checking your coverage, 38–39
  getting more detail with Verbose, 35–36
  handling error cases, 59–61
  improving doctests with Nose integration, 62–64
  running doctests, 58–59
  running specific test files, 35
  use of to demonstrate unit testing, 18
  using PyTest, 41–46
  using setUp, 26–27
Cassettes, 204
ccheckstyle, 175
CI (continuous integration), 2, 10, 33, 115, 157, 158, 168, 172, 175, 181, 206, 210
class name standard, 23
Clean Code: A Handbook of Agile Software Craftsmanship (Martin), 25
ccloud-based deployment, 185, 191–195
Cobertura, 176, 177, 181
code, making it public/open sourced, 207–208
code completion, 11
code coverage, 2, 38–39, 145, 154, 157, 159, 166
code defects, consequences of, 7
code evaluation score (PyLint report), 150
code modification cycles, 5
code profiling tools, 125, 135
code quality, 10
code review systems, 10
Coghlan, Nick (creator of Python language), 22
collaboration, between developer and QA, 90, 108
colorized output, 62
combined graph view (test plan), 134
Composite element (test plan), 133, 134
ConnectionError, 49
Console Output option (Jenkins), 173, 174
continuous deployment, 163
continuous integration (CI), 2, 10, 33, 115, 157, 158, 168, 172, 175, 181, 206, 210
Copy from Existing Job (Jenkins), 170
Coverage 3.7.1, 155
coverage reporting, 10, 176
Coverage tool
  advanced options, 157–161
  generating report, 176
  ignoring Coverage option, 160–161
  installation of, 155
  overview, 154
  producing HTML/XML report, 157–158
  purpose of, 82
  restricting Coverage to specific package, 159
Coverage tool (continued)
setting minimum coverage threshold, 159
using, 155–157
website, 162, 181
coverage.xml file, 157, 175, 176
cProfile
analyzing output from, 142–143
overview, 135–136
purpose of, 125
running session with, 136–142
website, 144
CPU intensive operations, 126
Cucumber
as acceptance test standard, 107–108
for accessing collaborative step
definitions, 208
described, 87
Gherkin syntax, 87–88
website, 123
Cucumber Pro, 208–209, 210

D
d (down) (PDB key command), 37
dashboards, 178
data, tables of in scenarios, 103–104
data submission forms, 7
database or other 3rd party systems
( example application stack), 196
data_interface class/layer/object, 47,
48, 49, 51
debugging, 7, 26, 33, 35, 36–38, 40, 43–44,
52, 62, 98, 108, 122
default build, 166–168, 188
deployment
cloud-based, 185, 191–195
continuous, 163
to Heroku, 192
making it natural, 205–206
to production environment, 183,
184–189, 205
to QA environment, 189–190
smoke tests as built into processes
of, 200
source control as integration point
for, 10
to stage environment, 183, 184,
190–191
status pages for displaying information
about, 198
deployment pipeline, 183, 184, 185, 201
development (stage of agile process), 69
development environment
creating perfect one, 206
deploying to, 183
interactive development environment
(IDE), 11–12
running Paver to check setup of, 164
distutils library, 187
Django web framework, 9, 11, 185
doc strings, 57
doctest documentation, 65
doctest library, 55
doctest module, 56, 61
doctest nose integration, 62–64, 65
doctests
adding of to methods, 57–58
advanced doctest usage, 61–62
handling error cases, 59–61
improvement in with nose integration,
62–64
Python shell, 56
running, 58–59
writing your first, 56–62
documentation
adding doctests to method, 57–58
advanced doctest usage, 61–62
BDD and acceptance testing as ways of
providing, 2
Coverage, 155
cProfile, 142
doctest, 65
Flask, 94
GitHub, 115, 204, 207–208
handling error cases, 59–61
human-readable, 116
improving doctests with nose integration, 62–64
JMeter, 126, 127
nose test, 46
PDB, 37
PyLint, 152
PyTest, 46
Python, 9, 28, 32, 36
Python 3.0, 8
Python shell, 56
Robot, 116
running doctests, 58–59
testing in, 1
unittest, 32
VCR.py GitHub offering, 204
writing first doctest, 56–62
writing testable documentation, 55–64, 86
Documentation string, 117
Don’t Repeat Yourself (D.R.Y.), 25
dummies (mocks), 51
dyno, 192

G
ending_test.py files, 34
EngineYard, 185, 201
error cases, 27, 48, 59–61, 160
example application stack, 196–197
exception handling, checking of with assertRaises, 20–22
Execute Shell option (Jenkins), 173, 174
eXtreme Feedback Panel (Jenkins), 178, 179, 180

H
happy paths, 6, 32, 71
Heroku, 185, 192–195, 201
Homebrew, 8
HTML output (Coverage), 157–158
HTTP Request element (test plan), 128, 129, 130, 132
HTTP Requests
  recording and replaying of using Stub Servers, 203
  in smoke test results, 198
  transactions as, 132, 133
HTTP Restful service, 205
human readable feature file, 2, 6
human-readable documentation, 116
### I

IDE (interactive development environment), 11–12

Ignore errors (PyLint customized error output), 153

Ignoring Coverage option (Coverage), 160–161

Import statement standard, 23

Indentation, 23, 145, 152

IndexError, 32

index.html page, 96, 97, 158

_init_.py files, 24, 46

Installation
- of Coverage tool, 155
- of dependencies, 173
- of Jenkins, 169–174
- of JMeter, 127
- of Mock library, 47
- of Nose, 34
- of nose-cov package, 38
- of nose-lettuce package, 114
- of Paver, 164
- of PyCallGraph library, 143
- of PyLint, 146
- of PyTest, 40
- of pytest-cov plug-in, 45
- of Python, 8–9
- of Rednose, 39–40
- of requests, 50
- of Robot test framework, 116
- of WebTest, 93

Instance properties, 145

Integration tests, 183

Intellectual property protection, 207

IntelliJ, 11

Interactive development environment (IDE), 11–12

I/O calls, blocking, 126

### J

Java, 126, 169

Jenkins
- Add Build Step drop-down menu, 173
- Add Post-Build Action option, 188
- adding integration to, 175
- benefits of, 163
- Build a Free-style Software Project, 170, 171
- Build Monitor Plugin, 178
- Console Output option, 173, 174
- Copy from Existing Job, 170
- creating custom dashboard for, 178
described, 168–169
- enabling archiving on, 188–189
- Execute Shell option, 173, 174
- eXtreme Feedback Panel, 178, 179, 180
- incorporating packaging into build, 187
- installation of, 169–174
- plug-ins, 122, 167, 172, 175, 176, 178, 181
- Poll SCM option, 172
- Radiator View Plugin, 178
- and smoke tests, 200
- Source Code Management section, 170, 172
- as tool to manage and run CI, 2, 115
- use of to automate testing, 206
- website, 181
- XML option of Coverage as use to, 157

JIRA, 206

jjp@gc (Throughput Shaping Timer element), 131

JMeter, 125, 126–128, 131–133, 135, 144

JMeterPluginsStandard, 128

Journeys (routes through application), 85, 196, 200, 201

jp@gc (Composite Graph element), 133–134
jp@gc (Response Latencies Over Time element), 132–133
jp@gc (Transactions per Second element), 132
jslint, 175
JSON, 103, 204

K
key commands (PDB), 37
keyword argument, 100
keywords
  for Robot test case, 116, 117–119
  for using Gherkin syntax, 87, 88

L
Lettuce BBD Testing Framework
  adding BDD folder alongside, 91
  in detail, 108–115
  as example of Python community support for testing, 203
  fail fast feature, 112–114
  implementation for Python known as, 87
  for implementing smoke tests, 198
  nosetest integration with, 114–115
  output, 92
  Robot test framework as alternative to, 123
  running test, 93–94, 95, 97, 98, 166
Scenario;, 88
  as sharing features offered by
    Cucumber, 107
    step files, 88–89
    step . hashes call, 104
    tagging, 108–112
    website, 105, 123
  Lettuce Nose Plug-in, 123
  lineno argument, 37
Linux
  Jenkins installation, 169
  JMeter installation, 127
  JMeter plug-ins, 128
  Python installation, 8
  Python shell, 56
  Listener element (test plan), 129, 132, 133
  lock contention, 126
  loose typing, 5

M
Mac, software installation, 8, 128, 169
MacPorts, 8
Martin, Robert Cecil (author)
  Clean Code: A Handbook of Agile Software Craftsman ship, 25
Mavericks, Python installation, 8
maximum line length standard, 23
messages by category section (PyLint report), 149–150
messages section (PyLint report), 150
method name standard, 23
minimum threshold option (Coverage), 159
Mock class, 47, 51
Mock library, 47
mock_data_interface, 48
mock_example_test.py file, 47
mocking, 46–49, 50, 52
mock_request.get call, 52
mock_response object, 52
mocks, 1, 51
module block (PyLint report), 149

N
n (next) (PDB key command), 37
naming conventions, 16, 17, 18, 23, 34, 55, 145
Netflix, 67, 205, 206, 208, 210
NetTuts, 83, 135
New Relic, 135, 144, 190
No Long Capture (-s flag), 35–36, 140
Nokia, 115
non-zero exit code, 165
non-zero return code, 167
Nose
best features of, 35
checking your coverage, 38–39
choosing between Nose and PyTest, 46
coloring tests with Rednose, 39–40
debugging support with PDB, 36–37
getting more detail with Verbose, 35–36
improving doctests with Nose integration, 62–64
installation of, 34
PyTest compared to, 40, 41, 43
running specific test files, 35
as test runner, 22
using, 33–40
Nose 1.3.0, 34
nose-lettuce package, 114
Nosetest
as commonplace tool, 33
described, 35, 62–64
as good first Paver task, 164
help command, 38
Nosetest integration, 114–115, 123, 154
output, 165, 166
PyTest compared to, 41
tags as feature of, 108
using assertions of, 122
using Coverage tool through integration in, 155
nosetests command, 34, 157
nose.tools, 93

O
offline legs, 190, 191, 196, 200
online legs, 190, 191, 200
open-sourcing code, 207–208
orchestration layer (example application stack), 196
OS X, 8, 56, 127, 169, 170
output.xml file, 122

P
PaaS (Platform as a Service), 185, 200
packaging, incorporation of into build, 187–188
pair programming, 1, 72, 73
patching, 1, 46, 50–52
@patch, 50, 51
pavement.py file, 164, 186, 187
Paver
building Paver tasks, 164–168
creating Paver task, 164–165
defining default build, 166–168
defining Paver tasks, 185–187
executing Paver tasks, 165–166
installation of, 164
reports generated from Paver tasks, 175–176
website, 181
Paver 1.2.2, 164
PCI compliance, 207
PDB, 36–38, 43–44
PEP-8 standards, 2, 22–23, 33, 145, 146, 149, 150, 152, 153, 162
PEPs (Python Enhancement Proposals), 22, 145
performance testing
importance of, 2, 126
as part of extensive testing suite, 201
to see how well application will cope with type of load it is expected to encounter in production, 189
testing in isolation as useful in, 49
using cProfile, 135–143
using JMeter, 125, 126–135
using Stub Server, 203
ping-pong programming, 72–73
Pip
to install Coverage, 155
to install dependencies, 173
to install Mock library, 47
to install Nose, 34
to install nose-cov package, 38
  to install nose-lettuce package, 114
  to install Paver, 164
  to install PyCallGraph library, 143
  to install PyLint, 146
  to install PyTest, 40
  to install pytest-cov plug-in, 45
  to install Rednose, 39–40
  to install requests, 50
  to install Robot framework, 116
  to install WebTest, 93
  installation of, 9
  to manage dependencies, 169
  as package manager for Python, 9
Platform as a Service (PaaS), 185, 200
plug-ins
  Cobertura, 176, 177, 181
  Git, 172
  Jenkins, 122, 167, 172, 175, 176, 178, 181
  JMeter, 128
  nose-lettuce package, 114, 115
  pytest-cov, 45
  Rednose, 39–40
  repository of, 12, 34
  Violations, 175, 181
Poll SCM option (Jenkins), 172
Post Build Action (PyLint), 175, 176
Postgres database, 192
pre-production environment, 135
Procfile, 192, 194
product owner, 6, 85, 210
production deployment, 184–189, 190
production environment
  cautions with leaving performance
    problems to be discovered only in, 126
    deploying to, 183, 184–189, 205
    implementation of, 190–191
    smoke tests on, 197
public domain, making code available to, 207–208
pybot test runner, 119, 122
PyCallGraph library, 143, 144
PyCharm, 11, 12
PyLint
  customizing PyLint’s output, 150–154
    in defining default build, 166
    errors, 167, 176
    installation of, 146
    overview, 145
    reports. See PyLint reports
      as returning names of failing tests, 43
      using, 146–149
      website, 162, 181
PyLint reports
  code evaluation score, 150
  generating, 175–176
  messages by category section, 149–150
  messages section, 150
  module block, 149
  raw metrics section, 150
  statistics by type section, 150
  understanding, 149–150
PyLintrc file, 150, 151, 152
PyLint .txt file, 175
PyPi repository, 9, 34, 114, 162
PyTest, 33, 40–46
PyTest 2.5.1, 40
pytest-cov plug in, 45
Python 2.7, 8
Python 2.7.5, 8
Python 3.0, 8
Python documentation, 9, 28, 32, 36
Python Enhancement Proposals (PEPs), 22, 145
Python installation, 8–9
Python Jinja 2 template, 97
Python official website, 9
Python shell, 56
python_testing.py file, 193
Q
QA environment, 189–190
quality assurance (QA), 6, 69, 85, 90, 108, 189

R
Radiator View Plugin (Jenkins), 178
raw metrics section (PyLint report), 150
readable feature files, 2, 6, 116
real-life conditions/real-world conditions, response to, 2, 125, 144
Rednose, 39–40
release process, 184, 190
render_template function, 96
report.html file, 122
Request Per Second (RPS) Schedule table (test plan), 131
requests library, 50, 51, 52
Requests Library GitHub, 207, 210
requests.get call, 50
resource.txt file, 118, 119
Response Latencies Over Time element (test plan), 132, 133
REST services, 50
restricting coverage to specific package option (Coverage), 159
retrospective (stage of agile process), 70
Robot 2.8.3, 115
Robot 2.8.4, 116
Robot test framework
  implementing keywords, 117–119
  installation of, 116
  overview, 107, 115–116
  running tests, 119–123
  website, 123
  writing test case, 116–117
robotLibrary.py file, 118, 119
Ruby programming language/community, 87, 107, 203

S
Scenario: (Gherkin syntax), 87, 88
Scenario Outline: (Gherkin syntax), 87
scenario outlines, 102–103
scrum master, 6, 161
sdist task, 187, 188
setUp method, 19, 26–27
setup script, 185, 206
setup.py file, 187
setuptools library, 187
setuputils library, 186
-s flag (No Long Capture), 35–36, 140
shaping timer element (test plan), 131
shebangs, 12
shell syntax, 164
showcase (stage of agile process), 69
smoke testing, 2–3, 183, 184, 190, 195–200, 209
sort argument, 138
Source Code Management section (Jenkins), 170, 172
Source Control, 10–11
Spotify, 67
sprint length, 68
sprint planning/planning games (stage of agile process), 69
stage deployment environment, 183, 184, 190–191
standards
  Cucumber as acceptance test standard, 107–108
  JMeterPluginsStandard, 128
  PEP-8. See PEP-8 standards
statistics by type section (PyLint report), 150
status codes, 52, 93, 96, 118, 129, 130, 197, 198, 199, 200
status monitor, 190
step definitions, 92, 94, 116, 119, 208
step files, 6, 85, 87, 88–89, 119
step.hashes call, 104
story management tools, 206
Stub Servers, 203–204, 210
style guidelines, 145
Subversion (SVN), 10–11
syntax highlighting, 11

tags/tagging, 108–112
--tags argument, 108
test cases
global variables/practices available to, 27, 89, 118
setting up imports of steps in, 118
using Robot, 119, 121, 123
writing, 116–117
test driven development (TDD)
advantages of, 16, 72
agile development processes, 67–70
ethos of, 70–71
example, 73–82
life cycle, 71
Test Plan element (test plan), 131
test plans
combined graph view, 134
Composite element, 133, 134
configuration of, 128–134
HTTP Request element, 128, 129, 130, 132
Listener element, 129, 132, 133
Request Per Second (RPS) Schedule table, 131
Response Latencies Over Time element, 132, 133
shaping timer element, 131
Test Plan element, 131
Thread Group element, 129
Throughput Shaping Timer element, 131
Timer element, 131
Transactions per Second view, 132
utilizing effectively, 135
View Results Tree element, 129
test runners
Nose. See Nose
PyTest. See PyTest
testing. See also acceptance tests/acceptance testing; automation (of testing process); Lettuce BBD Testing Framework; performance testing; smoke testing; unit testing
benefits of, 7
fundamentals and best practices, 7–12
future of, 203–210
history of, 5–12
what should be tested, 17
test/unit directory, 23
Then (Gherkin syntax), 87
Thread Group element (test plan), 129
Throughput Shaping Timer element (test plan), 131
Timer element (test plan), 131
tracebacks, 59–60
Transactions per Second view (test plan), 132
truth values, 28, 29
TypeError, 22, 60

U
u (up) (PDB key command), 37
Ubuntu, 8, 127
unhappy paths, 32, 71
unit testing
additional examples, 24–27
covering all code with, 154–161
described, 15–17
unit test structure, 23–24
useful methods, 26–32
writing your first unit test, 17–23
unittest documentation, 32
unittest package/module, 18, 19, 33
urllib2, 50
V
values
  checking of with `assertEqual`
  method, 18–20
  truth values, 28, 29
van, Rossum Guido (creator of Python), 22, 145
VCR.py library, 204, 210
Verbose (-v flag), 35, 42, 58, 59, 60, 63
version control, 10, 11
View Results Tree element (test plan), 129
Violations, 175, 176, 181
Virtualenv, 9–10, 11, 34, 39, 74, 173, 174

W
w (where) (PDB key command), 37
Warsaw, Barry (creator of Python language), 22
waterfall development process, 6, 67, 68
web application frontend (example application stack), 196
WebTest Browser testing library, 92–93, 105, 116, 122
WebTest Browser testing library 2.0.15, 93
When (Gherkin syntax), 87
Windows, Python installation, 8–9
--with-coverage argument, 155

X
XML output, 157, 206

Y
YAML files, 204