Morepath by Example Documentation

Release 0.1

Michael R. Bernstein

April 10, 2017

Contents

You want to develop an application with Python and Morepath? Here you have the chance to learn that by example. In this tutorial we will create a simple microblog application. It only supports one user that can create text-only entries and there are no feeds or comments, but it still features everything you need to get started. We will use Morepath as our framework and SQLite as our database which comes out of the box with Python, so there is nothing else you need.

If you want the full source code in advance or for comparison, check out the example source.

Contents:

CHAPTER 1

Introducing MoreBlog

We will call our blogging application MoreBlog here, feel free to choose a less web-2.0-ish name ;) Basically we want it to do the following things:

1. let the user sign in and out with credentials specified in the configuration. Only one user is supported.
2. when the user is logged in they can add new entries to the page consisting of a text-only title and some HTML for the text. This HTML is not sanitized because we trust the user here.
3. the page shows all entries so far in reverse order (newest on top) and the user can add new ones from there if logged in.

We will be using SQLite3 directly for that application because it's good enough for an application of that size. For larger applications however it makes a lot of sense to use SQLAlchemy that handles database connections in a more intelligent way, allows you to target different relational databases at once and more. You might also want to consider one of the popular NoSQL databases if your data is more suited for those.

Here a screenshot from the final application:

Continue with *Step 1: Creating The Folder*.

CHAPTER 2

Step 0: Setup Your Machine!

Goal: All learners will have a consistent computing environment for the rest of the workshop.

Learners: For each technology below, install and setup on your machine.

Guides: Verify the learner has completed install for each technology below.

Setup Linux

Your Computer

What is Your Computer?

Your computer is the thing you are reading and typing on :) You should know a few things about it before we go much further.

Why do I need my Computer?

Starting development work requires at least a passing familiarity with what is happening inside your machine, what software is installed, and where to look next for help!

Get information about your computer!

1. Figure out your OS and version
2. sudo privilege is pretty helpful
3. Check to see that you have at least 1 Gb of disk space left.
4. Start up a terminal. You can find the Terminal application at *Applications/Accessories/Terminal*, or it may already be on your menu bar. In your terminal:

   ```
   $ uname -a
   ```

Verify It Works!

Know your OS and version number? Good!

Command Line Interface

What is a Command Line Interface (CLI)?

A command line interface (CLI) is way of interacting with a computer by typing commands. DOS is an example of a command line interface.

Why do I need a Command Line Interface (CLI)?

Many development tools don't have graphic user interfaces–they only have command line interfaces.

Get a Command Line Interface (CLI)!

1. Linux comes with a command line interface included!
2. The program used to access the CLI is often called a "terminal".

Verify It Works!

Start up a terminal. You can find the Terminal application at *Applications/Accessories/Terminal*, or it may already be on your menu bar. In your terminal:

```
$ bash --version
```

gedit

What is `gedit`?

`gedit` is a cross-platform, syntax-highlighting text editor.

Why do I need `gedit`?

To write your code in! Word is a fine program, but it is not a text editor.

Get `gedit`!

Gedit is probably installed on your machine already.

- Ubuntu/Debian: https://help.ubuntu.com/community/gedit and follow the instructions
- Redhat: `yum install gedit`
- Build form source: http://ftp.gnome.org/pub/GNOME/sources/gedit/2.30/gedit-2.30.2.tar.gz

python

What is `python`?

Python is a general purpose, dynamically-typed, strongly-typed, interpreted computer programming language.

Why do I need `python`?

Well, this is a Python progammming workshop!

Get `python`!

Linux comes with Python installed!

Verify It Works!

1. Start up a terminal. You can find the Terminal application at *Applications/Accessories/Terminal*, or it may already be on your menu bar.
2. Test your Python install at the command prompt. Type `python` and hit enter. You should see something like:

```
Python 2.5.2 (r252:60911, Jan 24 2010, 17:44:40)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit the Python prompt, type `exit()` and press Enter. This will take you back to the Terminal command prompt.

Setup Mac

Your Computer

What is Your Computer?

Your computer is the thing you are reading and typing on :) You should know a few things about it before we go much further.

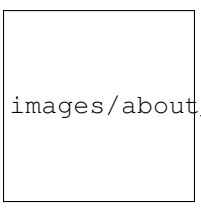
Why do I need my Computer?

Starting development work requires at least a passing familiarity with what is happening inside your machine, what software is installed, and where to look next for help!

Get information about your computer!

1. Figure out your OS and version
2. An Admin type account is pretty helpful ("Allow user to administer this computer")
3. Check to see that you have at least 1 Gb of disk space left.

```
Apple menu (upper left) > About This Mac
```



Verify It Works!

Know your OS and version number? Good!

Command Line Interface

What is a Command Line Interface (CLI)?

A command line interface (CLI) is way of interacting with a computer by typing commands.

Why do I need a Command Line Interface (CLI)?

Many development tools don't have graphic user interfaces–they only have command line interfaces.

Get a Command Line Interface (CLI)!

Macs come with a command line interface included!

Verify It Works!

1. Start up a Terminal. You can find the Terminal application through Spotlight, or navigate to *Applications/Utilities/Terminal*. In your terminal, type:

   ```
   bash --version
   ```

   Output should look similar to:

   ```
   $ bash --version
   GNU bash, version 3.2.48(1)-release (x86_64-apple-darwin10.0)
   Copyright (C) 2007 Free Software Foundation, Inc.
   ```

Terminal

While `Terminal` is a full-featured utility, its default configuration is a bit poor for programming usage. Let's configure `Terminal` to be a bit more friendly.

1. In your Terminal, copy and paste this command into your terminal and then hit enter. All this yarnbarf is to enable a colored terminal prompt.

   ```
   echo "export PS1='\[\e[0;33m\]\u\[\e[0;37m\]\[\e[0;36m\](\W)\[\e[0;0m\]\$ '" >> .bash_profile
   ```

2. Click `Terminal` in the upper-left hand corner, and go to Preferences.
3. Click on the Settings Section (right of Startup and left of Window Groups)
1. Highlight the Pro choice in the list on the left, and then click the Default button underneath
4. Click the Window tab (left of Text and right of Shell)
1. Click the black block left of "Color" and drag the Opacity slider to the right (100%)
2. Set the number of Rows to 40, under the Window Size section. Feel free to adjust taller or shorter to taste.
5. Close your Preferences window and terminal, and then go to Shell > New Window

Done properly, your new terminal should look like this:

gedit

What is `gedit`?

`gedit` is a cross-platform, syntax-highlighting text editor.

Why do I need `gedit`?

To write your code in! Word is a fine program, but it is not a text editor.

Get `gedit`!

- http://ftp.gnome.org/pub/GNOME/binaries/mac/gedit/2.30/gedit-2.30.2.dmg
- Install using the usual OSX 'DMG' style drag and drop installer.
- Add the application to your dock if you are so inclined.

python

What is `python`?

Python is a general purpose, dynamically-typed, strongly-typed, interpreted computer programming language.

Why do I need `python`?

Well, this is a Python programming workshop!

Get `python`!

OS X comes with Python installed!

Verify It Works!

1. Start up a Terminal. You can find the Terminal application through Spotlight, or navigate to */Applications/Utilities/Terminal*
2. Test your Python install at the command prompt. Type python and hit enter. You should see something like:

```
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to return to your terminal/shell. Don't worry if your version is different than the one shown here. Any 2.x series python 2.5 or higher (i.e., 2.5, 2.6, 2.7) should be fine!

Setup Windows

Your Computer

What is Your Computer?

Your computer is the thing you are reading and typing on :) You should know a few things about it before we go much further.

Why do I need my Computer?

Starting development work requires at least a passing familiarity with what is happening inside your machine, what software is installed, and where to look next for help!

Get information about your computer!

1. Figure out your OS and version
2. An Administrator type account is pretty helpful
3. Check to see that you have at least 1 Gb of disk space left.

```
Control Panel > System
```

Verify It Works!

Know your OS and version number? Good!

Command Line Interface

What is a Command Line Interface (CLI)?

A command line interface (CLI) is way of interacting with a computer by typing commands.

Why do I need a Command Line Interface (CLI)?

Many development tools don't have graphic user interfaces–they only have command line interfaces. Windows comes with a command line interface called a command prompt. Unfortunately, it does not support all of the tools we want to use.

Get a Command Line Interface (CLI)!

1. One of the easiest ways to get all the tools at once is from mysysGit
2. Download and install **msysGit-fullinstall** from http://msysgit.github.io/
3. This will install MSYS/Mingw/GitBash (it has many names!).

Note: If you are not an administrator on your machine, you might have to choose an alternate path for install rather than `C:\Git`, such as `C:\Documents and Settings\yourname\Git\`.

You will also want to make the MSYS/MINGW bash window behave in a saner way. After opening it, 'right-click' on the title bar, and select 'properties'. Under 'Edit Options', enable 'Quick Edit Mode'.

Verify It Works!

In your command window:

```
$ bash --version
```

gedit

What is `gedit`?

`gedit` is a cross-platform, syntax-highlighting text editor.

Why do I need `gedit`?

To write your code in! Word is a fine program, but it is not a text editor.

Get `gedit`!

- http://ftp.gnome.org/pub/GNOME/binaries/win32/gedit/2.30/gedit-setup-2.30.1-1.exe

python

What is `python`?

Python is a general purpose, dynamically-typed, strongly-typed, interpreted computer programming language.

Why do I need `python`?

Well, this is a Python progammming workshop!

Get python!

1. Go to http://python.org/download/ and download the latest version of Python 2.7 (2.7.1 at the time of writing). Unless you know otherwise, get the "Windows Installer" version, and not the "Windows X86-64 Installer" version.
2. Start up a command prompt by clicking on the Start menu, clicking the `Run...` option, typing `cmd`, and hitting enter. If you are using Windows Vista, you should click on the Start menu, type `cmd` into the Search field directly above the Start menu button, and click on `cmd` in the search results above the Search field.

Test your Python install by typing `\Python27\python.exe` and hitting enter. You should see something like:

```
Python 2.7.1 (r271:86832, ...) on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit the Python prompt, type `exit()` and press Enter. This will take you back to the Windows command prompt.

We also would like to make sure our python is available from the msys/mgit/mingw cli. After starting up msys/mgit:

```
$ echo '#!/bin/sh' > /bin/python
$ echo 'C:/Python27/python.exe $*' >> /bin/python
$ echo 'export PATH=$PATH:/c/Python27/Scripts' > ~/.profile
$ source ~/.profile
```

From here on, if you are in the msys/mgit/mingw cli, you can type `python` to get a python prompt, should you need one!

Verify It Works!

After install, show your guide that use can start up `python`. You should see something like:

```
Python 2.7.1 (r271:86832, ...) on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to return to your terminal/shell. Don't worry if your version is different than the one shown here. Any 2.x series python 2.5 or higher (i.e., 2.5, 2.6, 2.7) should be fine!

python

What is python?

Python is a general purpose, dynamically-typed, strongly-typed, interpreted computer programming language.

Why do I need python?

Well, this is a Python progammming workshop!

Get python!

windows

1. Go to http://python.org/download/ and download the latest version of Python 2.7 (2.7.1 at the time of writing). Unless you know otherwise, get the "Windows Installer" version, and not the "Windows X86-64 Installer" version.
2. Start up a command prompt by clicking on the Start menu, clicking the `Run...` option, typing `cmd`, and hitting enter. If you are using Windows Vista, you should click on the Start menu, type `cmd` into the Search field directly above the Start menu button, and click on `cmd` in the search results above the Search field.

Test your Python install by typing `\Python27\python.exe` and hitting enter. You should see something like:

```
Python 2.7.1 (r271:86832, ...) on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit the Python prompt, type `exit()` and press Enter. This will take you back to the Windows command prompt.

We also would like to make sure our python is available from the msys/mgit/mingw cli. After starting up msys/mgit:

```
$ echo '#!/bin/sh' > /bin/python
$ echo 'C:/Python27/python.exe $*' >> /bin/python
$ echo 'export PATH=$PATH:/c/Python27/Scripts' > ~/.profile
$ source ~/.profile
```

From here on, if you are in the msys/mgit/mingw cli, you can type `python` to get a python prompt, should you need one!

Mac OSX

OS X ships with Python installed, so the goal of this page is to make sure you can start a Terminal and run Python from the command line.

1. Start up a Terminal. You can find the Terminal application through Spotlight, or navigate to Applications/Utilities/Terminal
2. Test your Python install at the command prompt. Type python and hit enter. You should see something like:

   ```
   Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
   [GCC 4.2.1 (Apple Inc. build 5646)] on darwin
   Type "help", "copyright", "credits" or "license" for more information.
   >>>
   ```

Linux

Linux ships with Python installed, so the goal of this page is to make sure you can start a terminal and run Python from the command line.

1. Start up a terminal. You can find the Terminal application at *Applications/Accessories/Terminal*, or it may already be on your menu bar.
2. Test your Python install at the command prompt. Type `python` and hit enter. You should see something like:

   ```
   Python 2.5.2 (r252:60911, Jan 24 2010, 17:44:40)
   [GCC 4.3.2] on linux2
   Type "help", "copyright", "credits" or "license" for more information.
   >>>
   ```

To exit the Python prompt, type `exit()` and press Enter. This will take you back to the Terminal command prompt.

Verify It Works!

After install, show your guide that use can start up `python`. You should see something like:

```
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to return to your terminal/shell. Don't worry if your version is different than the one shown here. Any 2.x series python 2.5 or higher (i.e., 2.5, 2.6, 2.7) should be fine!

gedit

What is `gedit`?

`gedit` is a cross-platform, syntax-highlighting text editor.

Why do I need `gedit`?

To write your code in! Word is a fine program, but it is not a text editor.

Get `gedit`!

- Gedit's main site: http://projects.gnome.org/gedit/ . Download links are in the upper right corner.
- (alternate) copy the installer from the Group Guide memory stick.

windows

http://ftp.gnome.org/pub/GNOME/binaries/win32/gedit/2.30/gedit-setup-2.30.1-1.exe

Mac OSX

- http://ftp.gnome.org/pub/GNOME/binaries/mac/gedit/2.30/gedit-2.30.2.dmg
- install using the usual OSX 'DMG' style drag and drop installer.
- add the application to your dock if you are so inclined.

Linux

Gedit is probably installed on your machine already.

- ubuntu/debian: https://help.ubuntu.com/community/gedit and follow the instructions
- redhat: `yum install gedit`
- build form source: http://ftp.gnome.org/pub/GNOME/sources/gedit/2.30/gedit-2.30.2.tar.gz

Verify It Works!

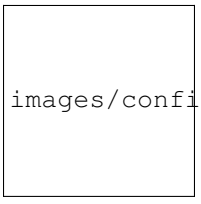
After installing, start `gedit` up by clicking on its icon, typing `gedit` from your cli or the like.

Suggested configuration:

In the preferences window

- **view**, turn on:
  - display line numbers
  - display right margin
  - highlight matching bracket
  - highlight current line
  - syntax-highlighting: Highlight Mode -> Scripts -> Python
- **editor**:
  - change tab with to 4
  - check the box to insert spaces instead of tabs
- **font & colors**:
  - choose a good monospace editor font
  - choose a color scheme (Gregg likes Cobalt)
- **plugins**
  - checkupdate
  - code comment
  - drawspaces
  - filebrowser
  - indentlines
  - python console

Once you are satified, restart `gedit`. If it looks similar to the screenshot below, you are done!



git, sshkeys, github

What is `git`?

`git` is a distributed verion control system ([D]VCS) . Others includee `mercurial (hg)` and `bazaar (bzr)`.

Why do I need `git`?

To download project sources, keep track of progress, and save your work.

Get `git`!

Note:

1. During the git install, choose all default settings
2. Skip the "take a crash course on git" at GitHub.
3. Skip other optional lessons (on terminal, etc.), unless you feel so inclined!
4. where it says 'text editor' you can use Gedit, if you like.
5. Set up a github account before setting up your keys!

---

windows

1. follow the instructions at http://help.github.com/win-set-up-git/ .
2. This will install MSYS/Mingw/GitBash (it has many names!).

Note: If you are not an administrator on your machine, you might have to choose an alternate path for install rather than `C:\Git`, such as `C:\Documents and Settings\yourname\Git\`.

You will also want to make the MSYS/MINGW bash window behave in a saner way. After opening it, 'right-click' on the title bar, and select 'properties'. Under 'Edit Options', enable 'Quick Edit Mode'.

Mac OSX

Note: When you get to the git download, you probably want the 386 version (vs. x64) of the 'newest' (highest version number) git install .dmg file.

---

follow the instructions at http://help.github.com/mac-set-up-git/ .

(*alternate path* : `brew install git` if homebrew is installed)

Linux

follow the instructions at http://help.github.com/linux-set-up-git/

Verify It Works!

- at your command-line, type `git`. Show your guide your ssh-key.
- show your guide your github account and userid.

virtualenv

What is `virtualenv`?

`virtualenv` is a tool that creates "new" Python installations that are copies of your installed Python. These "virtual environments" are clean and consistent.

Why do I need `virtualenv`?

To keep work areas clean and consistent, and minimize assumptions about what tools are installed where. This makes it easier to have repeatable, consistent builds.

Get `virtualenv`!

```
$ pip install virtualenv
```

Verify It Works!

```
# assume you have created a directory D and moved to it!
$ virtualenv --no-site-packages testenv
$ . testenv/bin/activate  # windows -> . testenv\Scripts\activate.bat
# your prompt should change to say (testenv)
$ deactivate
# remove the testenv directory
rm -rf testenv
```

- http://pypi.python.org/pypi/virtualenv

pip

What is `pip`?

`pip` is a utility for install and uninstalling Python packages. Packages are bundles of code that give additional functionality to an existing Python installation.

Why do I need `pip`?

To install and manage packages in a consistent way in Python.

Get `pip`!

First, install `ez_setup` to get `easy_install` (the *old* python package managers).

1. Open a terminal and run these commands:

   ```
   $ curl http://peak.telecommunity.com/dist/ez_setup.py | python
   $ easy_install pip
   ```

Verify It Works!

Run `pip` . You should see something like:

```
$ pip
Usage: pip COMMAND [OPTIONS]

pip: error: You must give a command (use "pip help" to see a list of commands)
```

morepath

What is `Morepath`?

Morepath is a new web-development framework written in Python.

Why do I need `morepath`?

For building our blog application!

Get `Morepath`!

We don't want to install Morepath system wide, but rather in our local virtualenv. We're also going to install `SQLAlchemy`, a library which will make it easier for us to use a database from our Morepath app later.

```
$ pip install morepath
$ pip install sqlalchemy
```

Verify It Works!

```
$ python
>>> import morepath
```

To quit the python prompt do:

```
exit()
```

Verify you can create a new Morepath app

Create a `hello.py` file with the following code:

```
import morepath

class App(morepath.App):
    pass

@App.path(path='')
class Root(object):
    pass

@App.view(model=Root)
```

```
def hello_world(self, request):
    return "Hello World!"

if __name__ == '__main__':
    config = morepath.setup()
    config.scan()
    config.commit()
    morepath.run(App())
```

And run it:

```
$ python hello.py
```

Output should look like this:

```
Running <__main__.App object at 0x7fda6fe86990> with wsgiref.simple_server on http://127.0.0.1:5000
```

In your browser, go to http://127.0.0.1:5000/

You should see in the browser the text 'Hello World', and in your terminal window you should see something like:

```
127.0.0.1 - - [30/Nov/2014 09:03:20] "GET / HTTP/1.1" 200 12
127.0.0.1 - - [30/Nov/2014 09:03:20] "GET /favicon.ico HTTP/1.1" 404 154
```

In the Terminal window where you ran python main.py, type control-c to kill the server.

Congratulations, you have run your first Morepath app!

How does this work?

We'll go into more detail as we develop our blogging app, but here is a quick overview of what this code is doing:

```
import morepath

class App(morepath.App):
    pass
```

We import `morepath`, and then we create a subclass of `morepath.App`. This class contains our application's configuration: what models and views are available. It can also be instantiated into a WSGI application object.

```
@App.path(path='')
class Root(object):
    pass
```

We then set up a `Root` class. Morepath is model-driven and in order to create any views, we first need at least one model, in this case the empty `Root` class.

We set up the model as the root of the website (the empty string `''` indicates the root, but `'/'` works too) using the `morepath.App.path()` decorator.

```
@App.view(model=Root)
def hello_world(self, request):
    return "Hello World!"
```

Now we can create the "Hello world" view. It's just a function that takes `self` and `request` as arguments (we don't need to use either in this case), and returns the string `"Hello World!"`. The `self` argument of a view function is the instance of the `model` class that is being viewed.

We then need to hook up this view with the `morepath.App.view()` decorator. We say it's associated with the `Root` model. Since we supply no explicit `name` to the decorator, the function is the default view for the `Root` model on `/`.

```
if __name__ == '__main__':
    config = morepath.setup()
    config.scan()
    config.commit()
    morepath.run(App())
```

The `if __name__ == '__main__'` section is a way in Python to make the code only run if the `hello.py` module is started directly with Python as discussed above.

`morepath.setup()` sets up Morepath's default behavior, and returns a Morepath config object.

We then `scan()` this module (or package) for configuration decorators (such as `morepath.App.path()` and `morepath.App.view()`) and cause them to be registered using `morepath.Config.commit()`.

This step ensures your configuration (model routes, views, etc) is loaded exactly once in a way that's reusable and extensible.

Continue with *Step 1: Creating The Folder*.

CHAPTER 3

Step 1: Creating The Folder

Before we get started, let's create the folders and files needed for this application:

```
/moreblog
    setup.py
    /moreblog
        __init__.py
        main.py
        model.py
        path.py
        view.py
```

The outer *moreblog* folder will not be a python package, but just somewhere we drop our files. We will then put our actual package containing all of our code into the inner *moreblog* folderfolder

As a first step, we need to list our dependencies and how to start the app in `setup.py`:

```
from setuptools import setup, find_packages

setup(name='moreblog',
    packages=find_packages(),
    install_requires=[
        'setuptools',
        'morepath',
        'transaction',
        'more.transaction',
        'zope.sqlalchemy >= 0.7.4',
        'sqlalchemy >= 0.9',
        'werkzeug',
        ],
    entry_points={
        'console_scripts': [
        'moreblog-start = moreblog.main:main'
        ]
    })
```

A few notes on the dependency packages: `morepath` specifies the latest version (currently 0.9 at the time of this writing), `transaction`, `more.transaction`, `zope.sqllchemy`, and `sqlalchemy` are all concerned with database persistence, and `werkzeug` is the web server we'll be using to run the app, chosen because it will auto refresh when files are edited without requiring a restart.

`entry_points` is defining a command that will start the app, in this case, `moreblog-start`, when the package is installed.

Continue with *Step 2: Adding the DB Schema*.

CHAPTER 4

Step 2: Adding the DB Schema

In `model.py` add the following code:

```
from sqlalchemy import (
    Column,
    Integer,
    Text,
    DateTime,
    ForeignKey,
    )
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Post(Base):
    __tablename__ = 'post'

    id = Column(Integer, primary_key=True)
    title = Column(Text)
    content = Column(Text)
```

In this code, we are importing from SQLAlchemy, a popular Python ORM, and then creating a table-based model class (As well see later though, Morepath models need not be based on a database, and can be almost any sort of object).

In our app, blog posts are very simple, and have an id, a title, and some content.

Next, in `main.py`, add the following .. code:

```
import morepath
import sqlalchemy
from more.transaction import transaction_app
from sqlalchemy.orm import scoped_session, sessionmaker
from zope.sqlalchemy import register

from werkzeug.serving import run_simple

from . import model

Session = scoped_session(sessionmaker())
register(Session)

class App(morepath.App):
    pass

def main():
```

```
engine = sqlalchemy.create_engine('sqlite:///morepath_sqlalchemy.db')
Session.configure(bind=engine)
model.Base.metadata.create_all(engine)
model.Base.metadata.bind = engine

morepath.autosetup()
run_simple('localhost', 8080, App(), use_reloader=True)
```

With this code, we are setting up the app to be run from the console script, making sure that the app has a database session, and also instantiatiating the database and table for storing the blog posts.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*