|  |
| --- |
| **http://www.saltycrane.com/blog/2008/05/django-new-blog-project/** |
| **CREATING A PROJECT:** From the command line, cd into a directory where you’d like to store your code, then run the following command:  **django-admin.py startproject mysite** This will create a mysite directory in your current directory. The following files are created: |
| The outer **mysite/ directory**: a container for your project. Its name doesn't matter to Django; give any name  **manage.py**: A command-line utility that lets you interact with this Django project in various ways  **The inner mysite/ directory**: the actual Python package for your project. Python will use to that name to import anything inside (e.g. import mysite.settings)  **mysite/\_\_init\_\_.py**: An empty file that tells Python that this directory should be considered a Python package  **mysite/settings.py**: Settings/configuration for this Django project  **mysite/urls.py**: The URL declarations for this Django project; a "table of contents" of your Django-powered site  **mysite/wsgi.py**: An entry-point for WSGI-compatible webservers to serve your project |
| **THE DEVELOPMENT SERVER**: cd to outer mysite directory and run the command: **python manage.py runserver**  You've started the Django development server  Now that the server's running, visit <http://127.0.0.1:8000/> with your Web browser. It worked!  By default, the [runserver](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-runserver) command starts the development server on the internal IP at port 8000.  If you want to change the server's port to port 8080, type in terminal: **python manage.py runserver 8080**  If you want to change the server's IP, pass it along with the port: **python manage.py runserver 0.0.0.0:8000** |
| **DATABASE SET UP: Edit mysite/settings.py**: change the following keys in the [**DATABASES**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-DATABASES) 'default' item to match your database connection settings.  [**ENGINE**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-DATABASE-ENGINE) -- Either 'django.db.backends.postgresql\_psycopg2', 'django.db.backends.mysql', 'django.db.backends.sqlite3' or 'django.db.backends.oracle'  [**NAME**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-NAME) -- The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, [NAME](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-NAME) should be the full absolute path, including filename, of that file. If the file doesn't exist, it will automatically be created when you synchronize the database for the first time (see below).  When specifying the path, always use forward slashes, even on Windows (e.g. C:/homes/user/mysite/sqlite3.db).  [**USER**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-USER) -- Your database username (not used for SQLite).  [**PASSWORD**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-PASSWORD) -- Your database password (not used for SQLite).  [**HOST**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-HOST) -- The host your database is on. Leave this as an empty string if your database server is on the same physical machine (not used for SQLite).  If new to databases, use SQLite by setting ENGINE to 'django.db.backends.sqlite3' and [NAME](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-NAME) to the place where you'd like to store the database  Set [**TIME\_ZONE**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-TIME_ZONE) to your time zone  [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-INSTALLED_APPS) setting toward the bottom of the file. That holds the names of all Django applications that are activated in this Django instance. Applications make use of at least one database table. We need to create the tables in the database, run the following command: python manage.py syncdb  The [syncdb](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-syncdb) command looks at the [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-INSTALLED_APPS) setting and creates any necessary database tables according to the database settings in your settings.py  To display the tables Django created:.run command-line client for your database and type \dt (PostgreSQL), SHOW TABLES; (MySQL), or .schema (SQLite)  *If you use PostgreSQL/MySQL, create a database. Do that with "CREATE DATABASE database\_name;" within your database's interactive prompt.*  *If you're using SQLite, you don't need to create anything beforehand - the database file will be created automatically when it is needed.* |
| **CREATING MODELS:** we create our poll app right next to your manage.py file so that it can be imported as its own top-level module, rather than a submodule of mysite. To create your app, make sure you're in the same directory as manage.py and type this command: **python manage.py startapp polls** That'll create a directory polls. |
| In our simple poll app, we'll create two models: polls and choices. A poll has a question and a publication date. A choice has two fields: the text of the choice and a vote tally. Each choice is associated with a poll. These concepts are represented by simple Python classes. Edit the polls/models.py:  **from django.db import models**  **class Poll(models.Model):**  **question = models.CharField(max\_length=200)**  **pub\_date = models.DateTimeField('date published')**  **class Choice(models.Model):**  **poll = models.ForeignKey(Poll)**  **choice = models.CharField(max\_length=200)**  **votes = models.IntegerField()**  Each model is represented by a class that subclasses[**django.db.models.Model**](https://docs.djangoproject.com/en/1.4/ref/models/instances/#django.db.models.Model). Each field is represented by an instance of a [**Field**](https://docs.djangoproject.com/en/1.4/howto/custom-model-fields/#django.db.models.Field)class which tells Django what type of data each field holds.A [**Field**](https://docs.djangoproject.com/en/1.4/howto/custom-model-fields/#django.db.models.Field) to designates a human-readable name. If this field isn't provided, Django will use the machine-readable name. In this example, we've only defined a human-readable name for Poll.pub\_date. A relationship is defined with [**ForeignKey**](https://docs.djangoproject.com/en/1.4/ref/models/fields/#django.db.models.ForeignKey) which tells Django each Choice is related to a single Poll |
| **ACTIVATING MODELS:** Edit the settings.py file again, and change the[**INSTALLED\_APPS**](https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-INSTALLED_APPS) setting to include the string 'polls', so django knows to include the polls app:  **INSTALLED\_APPS = (**  **'django.contrib.auth',**  **'django.contrib.contenttypes',**  **'django.contrib.sessions',**  **'django.contrib.sites',**  **'django.contrib.messages',**  **'django.contrib.staticfiles',**  **# Uncomment the next line to enable the admin:**  **# 'django.contrib.admin',**  **# Uncomment the next line to enable admin documentation:**  **# 'django.contrib.admindocs',**  **'polls',**  **)** |
| Run another command: **python manage.py sql polls**  [**python manage.py validate**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-validate) -Checks for any errors in the construction of your models.  [**python manage.py sqlcustom polls**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlcustom) -Outputs any [**custom SQL statements**](https://docs.djangoproject.com/en/1.4/howto/initial-data/#initial-sql)(such as table modifications or constraints) that are defined for the application.  [**python manage.py sqlclear polls**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlclear) -Outputs the necessary DROP TABLE statements for this app, according to which tables already exist in your database.  [**python manage.py sqlindexes polls**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlindexes) - Outputs the CREATE INDEX statements for this app.  [**python manage.py sqlall polls**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlall) -- A combination of all the SQL from the [**sql**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sql)**,** [**sqlcustom**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlcustom), and [**sqlindexes**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-sqlindexes) commands. |
| Run [**syncdb**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-syncdb) again to create those model tables in your database: **python manage.py syncdb** This creates all the tables, initial data and indexes for any apps you've added to your project since the last time you ran syncdb.[**syncdb**](https://docs.djangoproject.com/en/1.4/ref/django-admin/#django-admin-syncdb) can be called as often as you like, and it will only ever create the tables that don't exist |
| **PLAYING WITH THE API:** To invoke the Python shell, use this command: **python manage.py shell** manage.py sets the DJANGO\_SETTINGS\_MODULE environment variable, which gives Django the Python import path to your settings.py file. |
| >>> **from polls.models import Poll, Choice** # Import the model classes we just wrote. # No polls are in the system yet.  **>>>** **Poll.objects.all()**  []  # Create a new Poll.Support for time zones is enabled in the default settings file, so Django expects a datetime with tzinfo for pub\_date.  # Use timezone.now() instead of datetime.datetime.now().  **>>> from django.utils import timezone**  **>>> p = Poll(question="What's new?", pub\_date=timezone.now())**  **>>> p.save()** # Save the object into the database. You have to call save() explicitly.  # Now it has an ID. Note that this might say "1L" instead of "1", depending on which database you're using. That's no biggie; it just means your database backend prefers to return integers as Python long integer objects.  **>>> p.id**  1  **>>> p.question** # Access database columns via Python attributes.  "What's new?"  **>>> p.pub\_date**  datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)  **>>> p.question = "What's up?"**  # Change values by changing the attributes, then calling save().  **>>> p.save()**  **>>> Poll.objects.all()** # objects.all() displays all the polls in the database.  [<Poll: Poll object>] |
| <Poll: Poll object> is, utterly, an unhelpful representation of this object. Let's fix that by editing the polls model (in the polls/models.py file) and adding a [**\_\_unicode\_\_()**](https://docs.djangoproject.com/en/1.4/ref/models/instances/#django.db.models.Model.__unicode__) method to both Poll and Choice because objects' representations are used throughout Django's automatically-generated admin  **class Poll(models.Model):**  **# ...**  **def \_\_unicode\_\_(self):**  **return self.question**  **class Choice(models.Model):**  **# ...**  **def \_\_unicode\_\_(self):**  **return self.choice** |