Django

- **by Adrian Holovaty and Jacob Kaplan-Moss (released July 2005)**

Django Reinhart, 1910-1953

- **a collection of Python scripts to**

- **create a new project / site**
  - generates Python scripts for settings, etc.
  - configuration info stored as Python lists
- **creat a new application within a project**
  - generates scaffolding/framework for models, views
- **run a development web server for local testing**

- **generate a database or build interface to an existing database**
- **provide a command-line interface to application**
- **create an administrative interface for the database**
- **...**

Conventional approach to building a web site

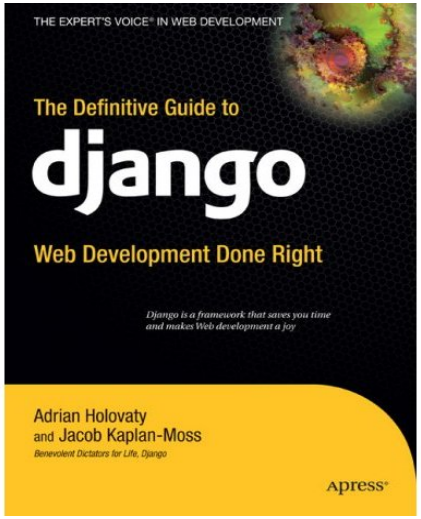
- **user interface, logic, database access are all mixed together**

```
import MySQLdb
print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>"
connection = MySQLdb.connect(user='me', passwd='x', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]
print "</ul>"
print "</body></html>"
connection.close()
```

Model-View-Controller (MVC) pattern

- **an example of a design pattern**
- **model: the structure of the data**
  - how data is defined and accessed
- **view: the user interface**
  - what it looks like on the screen
  - can have multiple views for one model
- **controller: how information is moved around**
  - processing events, gathering and processing data, generating HTML, ...
- **separate model from view from processing so that when one changes, the others need not**
- **used with varying fidelity in**
  - Django, App Engine, Ruby on Rails, XCode Interface Builder, ...
- **not always clear where to draw the lines**
  - but trying to separate concerns is good

Django web framework



djangobook.com

- **write client code in HTML, CSS, Javascript, ...**
  - Django template language helps
    separate form from content
- **write server code in Python**
  - some of this is generated for you
- **write database access with Python library calls**
  - they are translated to SQL database commands

- **URLs on web page map mechanically to Python function calls**
  - regular expressions specify classes of URLs
  - URL received by server is matched against regular expressions
  - if a match is found, that identifies function to be called
    and arguments to be provided to the function

Django automatically-generated files

- **generate framework/skeleton of code by program**
- **three basic files:**

  models.py: database tables, etc.

  views.py: business logic, formatting of output

  urls.py: linkage between web requests and view functions
- **plus others for special purposes:**

  settings.py: db type, names of modules, ...

  tests.py: test files

  templates: for generating and filling HTML info

Database linkage

in settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/Users/bwk/dj1/mysite/sql3.db', ...
```

in models.py

```
from django.db import models
class Books(models.Model):
    isbn = models.CharField(max_length=15)
    title = models.CharField(max_length=35)
    author = models.CharField(max_length=35)
    price = models.FloatField()
```

generated by Django

```
BEGIN;
CREATE TABLE "db1_books" (
    "id" integer NOT NULL PRIMARY KEY,
    "isbn" varchar(15) NOT NULL,
    "title" varchar(35) NOT NULL,
    "author" varchar(35) NOT NULL,
    "price" real NOT NULL
);
```

URL patterns

- **regular expressions used to recognize parameters and pass them to Python functions**
- **provides linkage between web page and what functions are called for semantic actions**

```
urlpatterns = patterns('',
  (r'^time/$', current_datetime),
  (r'^time/plus/(\d{1,2})/$', hours_ahead),
)
```

- **a reference to web page …/`time/` calls the function**

```
current_datetime()
```

- **tagged regular expressions for parameters: url …/`time/plus/12` calls the function**

```
hours_ahead(12)
```

Templates for generating HTML

- **try to separate page design from code that generates it**
- **Django has a specialized language for including HTML within code**
  - loosely analogous to PHP mechanism

```
# latest_books.html (the template)

<html><head><title>Books</title></head>
<body>
<h1>Books</h1>
<ul>
{% for book in book_list %}
    <li>{{ book.name }}</li>
{% endfor %}
</ul>
</body></html>
```

Administrative interface

- **most systems need a way to modify the database even if initially created from bulk data**
  - add / remove users, set passwords, ...
  - add / remove records
  - fix contents of records
  - ...
- **often requires special code**

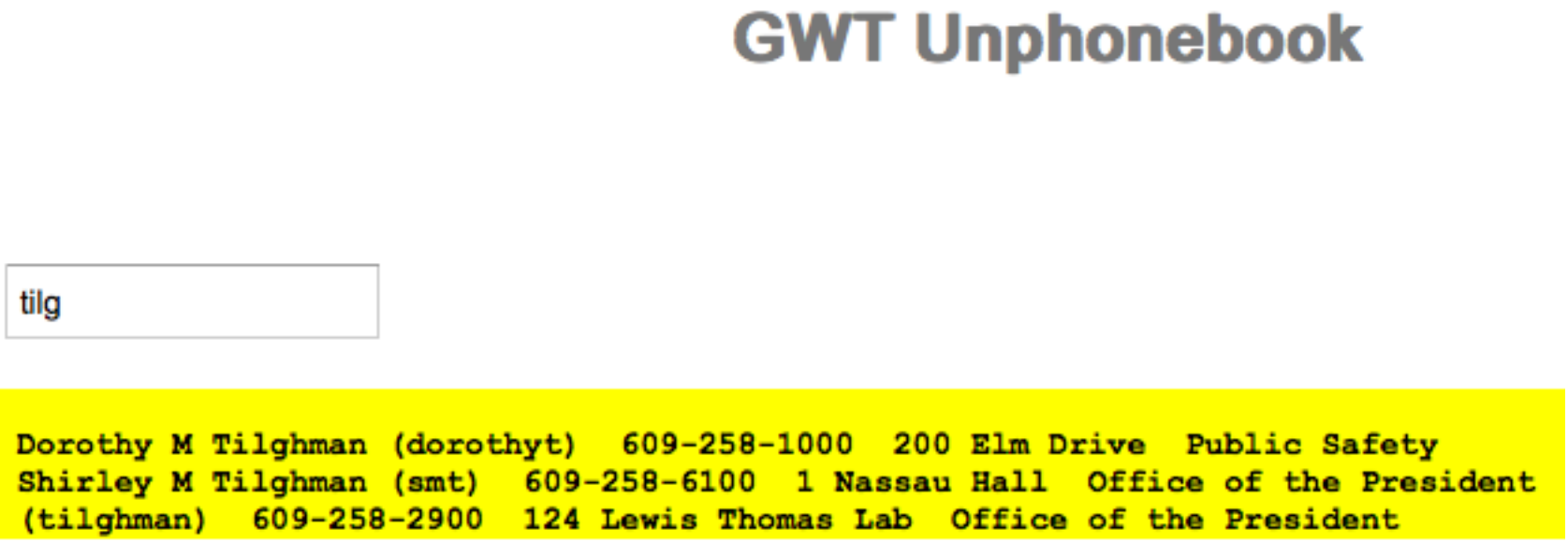
- **Django generates an administrative interface automatically**
  - loosely equivalent to MyPhpAdmin

```
urlpatterns = patterns('',
  ...
  # Uncomment this for admin:
  # (r'^admin/', include('django.contrib.admin.urls')),
```

Google Web Toolkit (GWT) (first available May 2006)

- **write client (browser) code in Java**
  - widgets, events, layout loosely similar to Swing
- **test client code on server side**
  - test browser, or plugin for testing with real browser on local system
- **compile Java to Javascript and HTML/CSS**
  - [once it works]
- **use generated code as part of a web page**
  - generated code is browser independent (diff versions for diff browsers)
- **can use development environments like Eclipse**
  - can use JUnit for testing
- **strong type checking on source**
  - detect typos, etc., at compile time (unlike Javascript)
- **doesn't handle all Java runtime libraries**
  - ?
- **no explicit support for database access on server**
  - use whatever package is available

Unphonebook in GWT



GWT example (client side, excerpt 1)

```
public void onModuleLoad() {
      final TextBox nameField = new TextBox();
      final Label outputArea = new Label(); // was TextArea

      RootPanel.get("nameFieldContainer").add(nameField);
      RootPanel.get("outputAreaContainer").add(outputArea);
      nameField.setFocus(true);

      final Label textToServerLabel = new Label();
      final HTML serverResponseLabel = new HTML();

      // Create a handler for the sendButton and nameField
      class MyHandler implements KeyUpHandler {

         public void onKeyUp(KeyUpEvent event) {
            if (nameField.getText().length() > 1) {
               sendNameToServer();
            }
         }
```

GWT example (client side, excerpt 2)

```
private void sendNameToServer() {
      String textToServer = nameField.getText();
      textToServerLabel.setText(textToServer);
      serverResponseLabel.setText("");
      greetingService.greetServer(textToServer,
            new AsyncCallback<String>() {
               public void onFailure(Throwable caught) {
               }
               public void onSuccess(String result) {
                  outputArea.setText(result);
               }
            });
   }
}

// Add a handler to send the name to the server
MyHandler handler = new MyHandler();
nameField.addKeyUpHandler(handler);
```

GWT example (client side, excerpt 3)

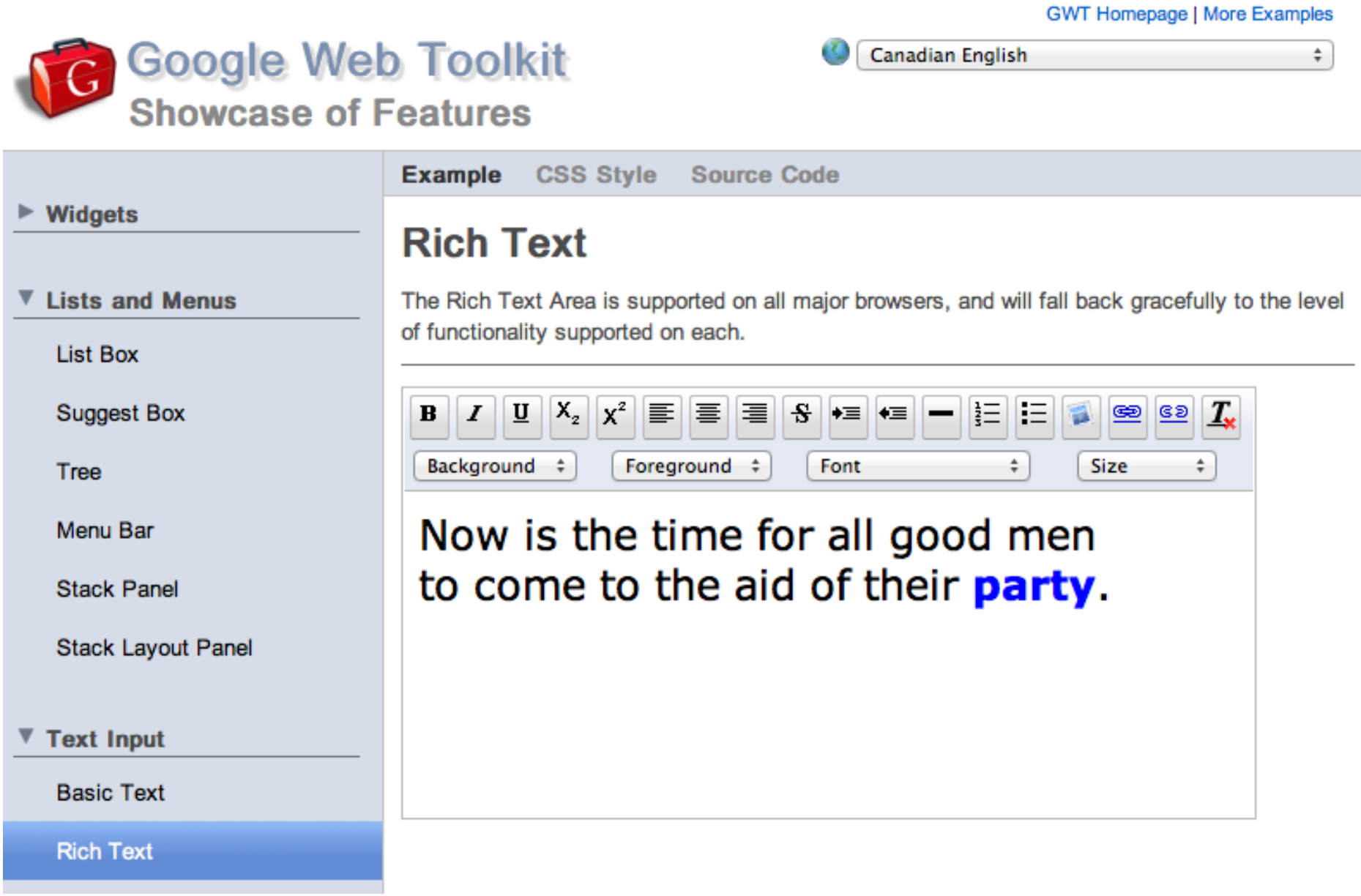
```
<h1 align="left">GWT Unphonebook</h1>

  <table >
    <tr>
      <td id="nameFieldContainer"></td>
    </tr>
    <tr>
      <td colspan="2" style="color:red;"
         id="errorLabelContainer"></td>
    </tr>
  </table>
  <pre id="outputAreaContainer"
     style="backgroundColor:#FFFF00; fontWeight:bold;"></pre>
```

GWT example (server side)

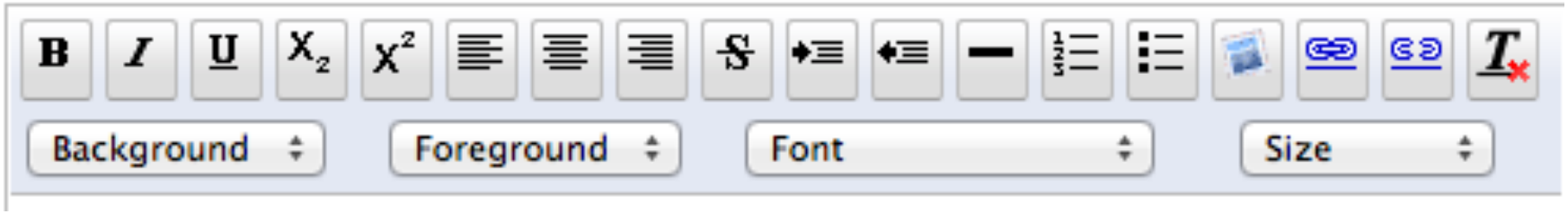
```
public class GreetingServiceImpl extends RemoteServiceServlet
                                 implements GreetingService {
   public String greetServer(String input)
                                 throws IllegalArgumentException {
   String result = "";
   Runtime rt = Runtime.getRuntime();
   Process p;
   try {
       input = escapeHtml(input);
       p = rt.exec("grep -i " + input + " phone.txt");
       BufferedReader pin = new BufferedReader(
               new InputStreamReader(p.getInputStream()));
       String s;
       while ((s = pin.readLine()) != null)
          result += ”\n" + s;
       pin.close();
   } catch (IOException e) { result = "exec error”; }
   return result;
}
```

GWT Widgets


GWT Homepage | More Examples
Google Web Toolkit
Showcase of Features
Canadian English
Example
CSS Style
Source Code
Widgets
Lists and Menus
List Box
Suggest Box
Tree
Menu Bar
Stack Panel
Stack Layout Panel
Text Input
Basic Text
Rich Text
Rich Text
The Rich Text Area is supported on all major browsers, and will fall back gracefully to the level of functionality supported on each.
Background
Foreground
Font
Size
Now is the time for all good men
to come to the aid of their party.

Browser independence, almost

- **Firefox**



- **Chrome**

"Same Origin Policy"

- **"The same origin policy prevents a document or script loaded from one origin from getting or setting properties of a document from another origin. This policy dates all the way back to Netscape Navigator 2.0." (Mozilla)**

- **"The SOP states that JavaScript code running on a web page may not interact with any resource not originating from the same web site." (Google)**

- **basically Javascript can only reference information from the site that provided the original code**

- **BUT: if a page loads Javascript from more than one site (e.g., as with cookies from third-party sites), then that JS code can interact with that third-party site**

GWT assessment

- problem: Javascript is irregular, unsafe, not portable, easily abused

- solution: use Java, which is type-safe, standard, portable
-
- translate Java to Javascript to either be browser independent
  or tailored to specific browser as appropriate
- can take advantage of browser quirks, make compact code,
  discourage reverse engineering
- can provide standardized mechanisms for widgets, events,
  DOM access, server access, AJAX, RE's and other libraries,
  ...

- in effect, treat each browser as a somewhat irregular machine and compile optimized code for it specifically

GWT vs Django

- **focusing on different parts of the overall problem**

- **GWT provides**
  - reliable, efficient, browser-independent Javascript (from Java)
  - extensive widget set
  - no help with database access, generating HTML, …

- **Django provides**
  - no Javascript help
  - no widgets
  - easy database access; template language for generating HTML, …
  - easy linkage from URLs on web page to Python functions

- **is GWT + App Engine a good combination?**

Assessment of Web Frameworks

- **advantages**
  - takes care of repetitive parts
    more efficient in programmer time
  - automatically generated code is likely to be more reliable, have more uniformity of structure
  - "DRY" (don't repeat yourself) is encouraged
  - "single point of truth"
    information is in only one place so it's easier to change things
  - ...
- **potential negatives**
  - automatically generated code
    can be hard to figure out what's going on
    can be hard to change if you don't want to do it their way
  - systems are large and can be slow
  - ...
- **read Joel Spolsky's "Why I hate frameworks"**

  `http://discuss.joelonsoftware.com/default.asp?joel.3.219431.12`

Assessment of Ajax-based systems

- **potential advantages**
  - can be much more responsive (cf Google maps)
  - can off-load work from server to client
  - code on server is not exposed
  - continuous update of services
- **potential negatives**
  - browsers are not standardized
  - Javascript code is exposed to client
  - Javascript code can be bulky and slow
  - asynchronous code can be tricky
  - DOM is very awkward
  - browser history not maintained without effort
- **what next? (changing fast)**
  - more and better libraries
  - better tools and languages for programming
  - better standardization?
  - will the browser ever replace the OS?