Advanced Programming Techniques

Web Frameworks

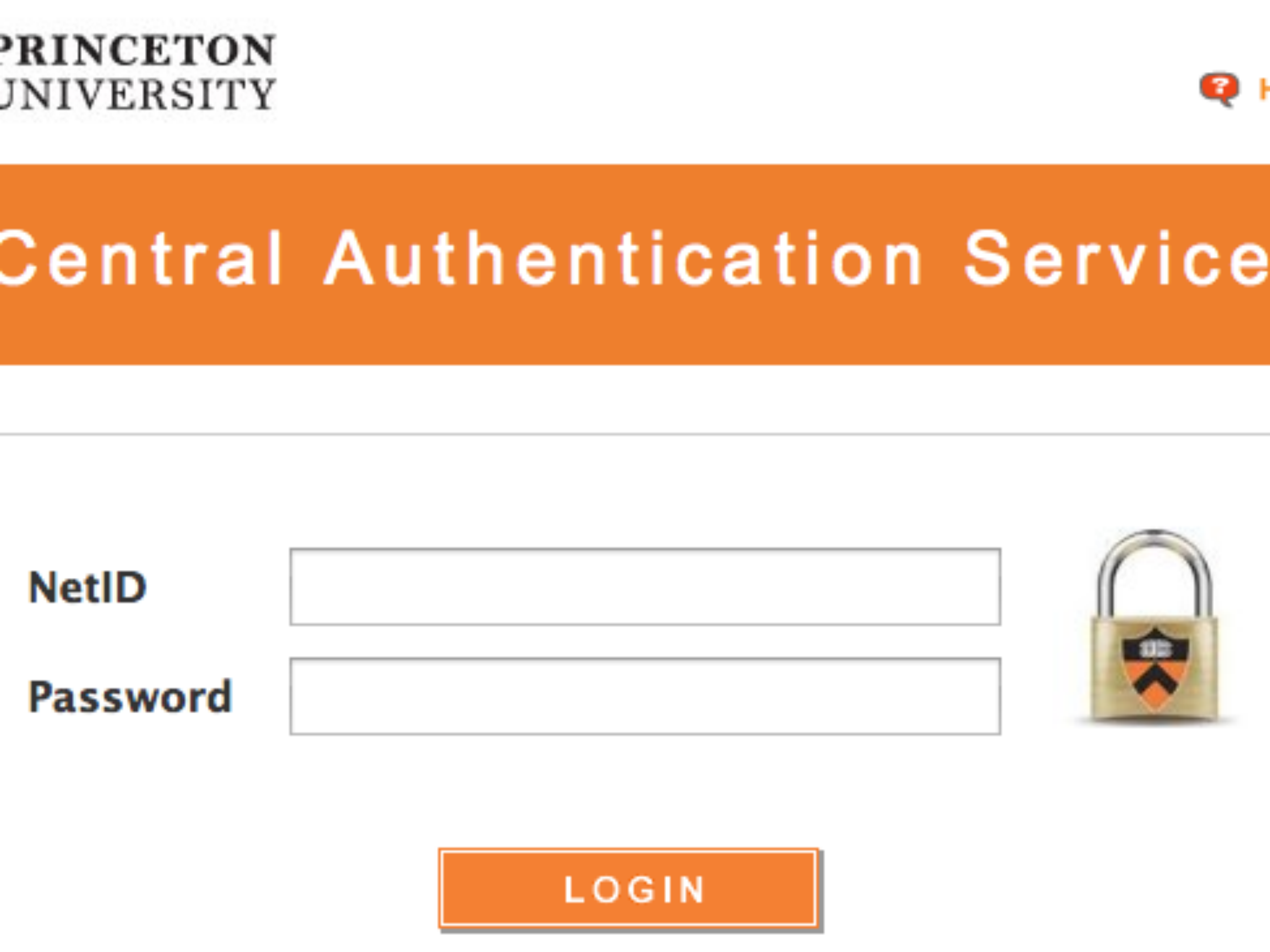
Christopher Moretti

But first, a word from our Cupertino overlords …

iOS Development Legal Stuff

- For an iOS app, you need a developer certificate to load your app on a device. P and Apple have an arrangement that lets you do these experiments for free
- http://goo.gl/forms/Q5YjwJ1Pkj
- This will not let you put your app on the App Store, but it's all you need for local development and testing with friends.
- Thanks to McGraw's Janet Temos!

Continuing with PU services to make our projects work …


PRINCETON
UNIVERSITY
Central Authentication Service
NetID
Password
LOGIN

CAS Overview

- OIT-provided central authentication service
  - User visits your homepage
  - You kick them to CAS to log in (via libraries)
  - If successful, CAS returns to you with "`yes netid`"
- This is **REQUIRED** if you are using Princeton accounts
  - in lieu of or in addition to your own accounts, to verify status for non-public resources, etc.

CAS Details

```php
<?php
require 'CASClient.php';
$C = new CASClient();
$netid = $C->Authenticate();

echo "Hello from the other side, $netid.";
echo "<P>Think of this as the main page ";
echo "after $netid has been authenticated.";
?>
```

```python
#!/usr/bin/python

import CASClient
C = CASClient.CASClient()
netid = C.Authenticate()
print "Content-Type: text/html"
print ""
print "Hello from the other side, %s\n" % netid
print "<p>Think of this as the main page after %s has been authenticated." % netid
```

Sample usage in several languages: http://www.cs.princeton.edu/~cmoretti/cos333/CAS/

(Cryptic) OIT Details: https://sp.princeton.edu/oit/sdp/CAS/Wiki%20Pages/Home.aspx

Before we start on actual content …

Surprise!

Web Projects: A problem

- Conventional web development:
  - Write ad-hoc client code in HTML, CSS, js, etc. by hand
  - Write ad-hoc server code in (anything) by hand
  - Write ad-hoc access to database
  - Iterate
    - (every page, feature, app, etc. is new from scratch)

Worst Case

UI, business logic, DB interface are all jumbled in one place!

```
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()
```

C'mon, I'd **never** do that!

```
import SocketServer
import SimpleHTTPServer

class Reply(SimpleHTTPServer.SimpleHTTPRequestHandler):
  def do_GET(self):
    output=filter_by_query(self.path) #business logic
    self.wfile.write("query was %s\n" % self.path)
    self.wfile.write(output) #maybe marked up and styled?

def main():
  f = open("courses.json") #"database" access
  courses = eval(f.read())
  if len(sys.argv > 1): PORT = int(sys.argv[1])
  else: PORT = 8080
  SocketServer.ForkingTCPServer('', PORT),
                           Reply).serve_forever()

def filter_by_query(path):
  ...
# and lots more functions!
main()
```

Model-View-Controller Pattern

- A design pattern we've already seen (iOS)
- Model: structure of data (how is it defined, accessed?)
- View: the UI (what is on-screen?)
  - View:Model can be a many-to-one relationship
- Controller: business logic (how does data flow?)
  - event- or logic-based, gather input, prepare output
- Where to draw the lines, how to define interfaces?

Web Application Frameworks

- A response to the “ad hoc” problem above!
  - Structure everything with MVC design, templates
  - Pick the right pieces and the mechanics are (near-) automatic, rather than laborious and error-prone
  - You pick your data model, you design your standard views (as templates), and you write your core logic
    - But not interface, “glue”, “plumbing”, data flow

Web Application Frameworks

- Typically Client/Server: client sends request from form; server parses request, retrieves info from DB in a structured and safe way, formats result (maybe with templates) and returns response.
    - Often REST-like (server parses URL to call function)
        - e.g. `/add/data_to_add` or `/login/username`
- Possible downsides: heavyweight, not always a model fit, hard to track/debug auto-generated content

Flask

- Python-based micro-framework
- Simple, low barrier-to-entry for initial tests
    - Good example of a template engine
    - Good example of app “paths” (“routes”)

Flask Hello World

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Flask Survey Handler

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['POST','GET'])
def hello():
    s = ""
    for (k,v) in request.form.iteritems():
        s = "%s %s=%s<br>" % (s, k, v)
    return ‘Entry:<br>' + s

app.run()
```

Flaskr Miniblog Example

- Comes default as part of Flask documentation
  - Created by Flask developer Armin Ronacher
- Examples of:
  - URL routing
  - Integrated CSS styling
  - Templates with variables integrated into layout
  - SQLite3 database stored locally

Memoizing Fibonacci Server

- Running example for several services
- Takes argument n, returns nth Fib number, memoizing result (pseudocode):

```
def get_n(n):
    if n < 0:
        return 0
    result = db.query "SELECT val from entries where num == n”
    if result != None:
     return result
    else:
        nm2 = get_n(n-2)
        nm1 = get_n(n-1)
        new = nm2+nm1
        db.query("insert into entries (num, val) values (?, ?)", (n, new))
        return new
```

Memoizing Fibonacci Server (Flask)

```python
from sqlite3 import dbapi2 as sqlite3
from flask import Flask, render_template, request, _app_ctx_stack

@app.route(‘/add_numbers')
def add_numbers():
    num = request.args.get('num', 0, type=int)
    val = get_n(get_db(),num)
    return render_template('index.html',result=val)

@app.route('/clear')
def clear():
    init_db()
    return render_template('index.html')

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    init_db()
    app.run()
```

```python
def init_db():
    with app.app_context():
        db = get_db()
        script = """
        drop table if exists entries;
        create table entries (
        num integer primary key,
        val text
        );
        insert into entries (num,val) values (0,0);
        insert into entries (num,val) values (1,1);
        """
        db.cursor().executescript(script)
        db.commit()
```

Fibonacci Template (Flask)

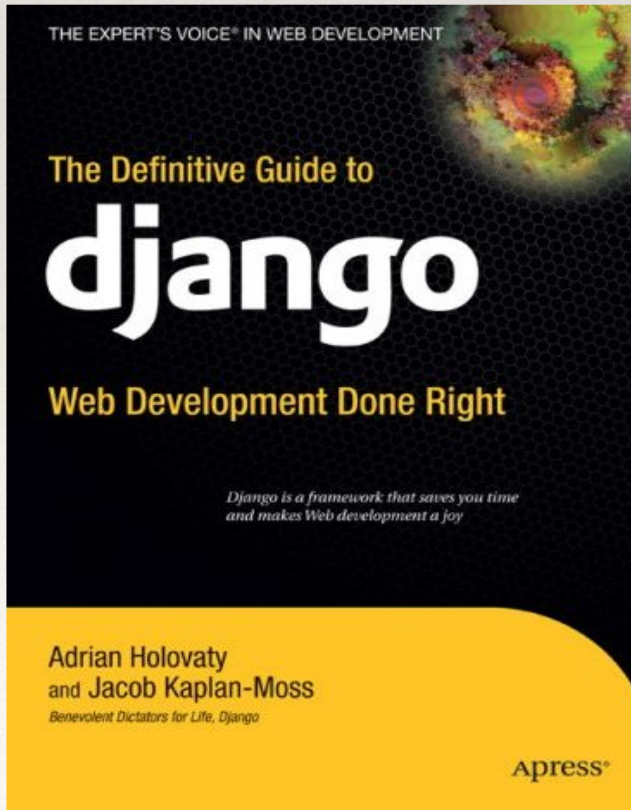
```
{% extends "layout.html" %}
{% block body %}

<h1>Fibonacci Example</h1>
<p>
  <form action="http://localhost:5000/add_numbers" method=GET>
    <input type=text size=5 name=num>
    <br />
    <div id=result style="width:55% ; word-wrap:break-word"> {{ result }} </div>
    <br />
    <button type=submit value=“fib me!”>
  </form>
</p>
{% endblock %}
```

Django

- A more heavyweight Python MVC framework.
- Adrian Holovaty, Jacob Kaplan-Moss, Wilson Miner Simon Willison ~2005
- Now used by Pinterest, Instagram, Bitbucket, and ~50% of COS333 projects
- Named for Django Reinhardt >

Django Overview

- Client code is still customary HTML, CSS, js
    - Write a small number of templates to form all pages
    - Template classes help separate form from content
- Server code is garden-variety Python
    - Much is generated for you or “pluggable”
- Use restricted set of Python DB library calls
    - Automatically escaped, safety-checked, and translated into SQL queries


THE EXPERT'S VOICE® IN WEB DEVELOPMENT

The Definitive Guide to

django

Web Development Done Right

Django is a framework that saves you time and makes Web development a joy

Adrian Holovaty
and Jacob Kaplan-Moss
Benevolent Dictators for Life, Django

Apress®

Django Auto-generated Files

- `models.py` (database tables, information architecture)
- `views.py` (business logic, output generation)
- `urls.py` (mapping from web path to code function)
- `settings.py` (config info - DB type, module IDs, &c.)
- `tests.py` (testing framework *(you can't escape it!)*)
- `admin.py` (administrative console info)
- templates (presentation, filled with generated output)

Django DB Linking Example

settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ‘/tmp/fib.db', ...
```

models.py

```
from django.db import models
class Fib(models.Model):
    num = models.IntegerField(primary_key=True)
    val = models.TextField()
```

(auto-generated)

```
BEGIN;
CREATE TABLE “fib” (
“num” integer NOT NULL PRIMARY KEY
“val” text NOT NULL
);
```

Fibonacci Template (Django)

```
# ”layout.html"

<html>
<body>

<h1>All-terms Fibonacci Example</h1>
<p>
    <ol>
     {% for num in fib_list %}
     <li style="width:55% ; word-wrap:break-word"> {{ num }} </li>
     {% endfor %}
    </ol>
</p>

</body>
</html>
```

URL Patterns

- Regexp to map URLs into functions and parameters
  - Implemented as regexp, Python decorator, etc.
  - Example:
    - `${BASE_URL}/time/` should give current time
    - `${BASE_URL}/time/plus/n` should add `n` hours

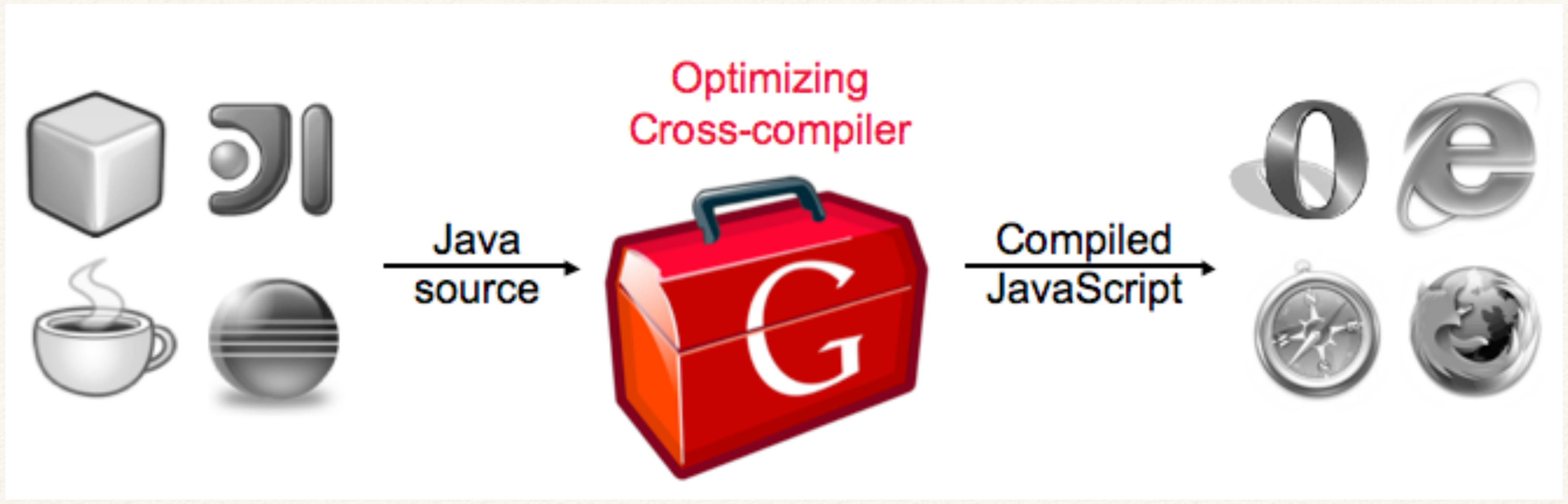
```
urlpatterns = patterns('',
        (r'^time/$', current_datetime),
        (r'^time/plus/(\d{1,2})/$', hours_ahead),
    )
```

Other Frameworks

- Ruby on Rails
- Google App Engine
- Node.js + Express
- Google Web Toolkit
- … (an ever-expanding list of others)

Google App Engine

- Web framework analogous to Django
- Server-side support for wide variety of languages
  - Python, Php, Go, Java, …
- Can run locally for test/debug, deploy to appspot.com
- Google limits some server-side functionality
  - DB limited to GQL or BigTable
  - No system calls, threads, network access …



- Java-based framework (both server and front-end), compiles to JavaScript for presentation
- Works using RPC that serializes to JSON
- + single language, code re-use (similar to Node.js)
- + browser compatible presentation for free
- + lots of widgets and plugins available free from Google

Caveat

"Although we were all most familiar with Java and appreciated its abilities to integrate with various libraries and the GWT framework, we would definitely be willing to find ways to accomplish our goals in another language in order to avoid having to deal with GWT."

– Feedback from a group in a previous year

Frameworks Summary

- Advantages
  - Take care of repetitive parts easily; encourage “DRY”
  - Automated code is less likely to have human error
  - Every piece has its place: only need to change once
- Disadvantages
  - You’re less likely to know what the #$&* is going on
  - You’re out of luck if you don’t want things “how it is”

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