Announcements

- Checkpoint on Proj3 due Wednesday night (pushed back one day)
- Set up user ID Django database, Web sites for user ID entry

Pedometer data entry – use it!

pc110.cs.ucdavis.edu:10002/hw2/index.html

Steps so far - checklist

- Start app (python manage.py startapp newpolls)
- Edit newpolls/models.py, add database classes
- Edit mysite/settings.py to connect models to Django
- python manage.py syncdb
- Put some data in with shell (not necessary in HW)
- Edit mysite/urls.py
- Add and edit newpolls/urls.py
- Add and edit newpolls/views.py
- Put templates into newpolls/templates/newpolls

Today

- Getting a form onto a Django Web page
- Getting data out of URL and using it
- Familiar data transfer strategy: pass variables in a little dictionary

Template for voting page

```html
<h1>{{ question }}</h1>

{% if message %}<p><strong>{{ message }}</strong></p>{% endif %}

<form action="/django/newpolls/vote" method="get">
  {% for choice in choices %}
  <label><input type="radio" name="choice" value="{{ choice.id }}" />
  {{ choice.choice_text }}</label>
  <br />
  <br />
  {% endfor %}
  <input type="submit" value="Vote" />
</form>
```
## Django templates

- A variable is inside `{{  }}`
  ```
  {{ message }}
  ```
- Attributes of objects via the usual dot notation,
  e.g. `choice.choice_text` or `choice.votes`

## More templates

- Programming constructs inside `{%  %}`
  ```
  {% if message %} – this means if message is not empty.
  ```
- Block ends with `{% endif %}`
- Can have `{% if...%}{% else%}{% endif %}`
- For loop
  ```
  {% for choice in choices %}...
  {% endfor %}
  ```

## Fill in data for template in views.py

```python
def detail(request):
    p = Poll.objects.get(id=1)
    context = { 'question': p.question,
                'choices': p.choice_set.all(),
                'message': '' }
    return render(request, 'newpolls/detail.html', context)
```

- context is a dictionary where keys are template variable names and whose values can be constants or items from database

## GET vs POST HTTP request

- `<form action="/django/newpolls/vote" method="get">`
- Recall these are two ways to send form data to the server. GET puts it into the URL; POST puts it in the body of the HTTP request.
- Tutorial uses POST, but GET is visible.
- Produces URL such as:
  ```
  pc110.cs.ucdavis.edu:10000/django/newpolls/vote?choice=1
  ```

## Template for reporting votes

```html
<h1> {{ question }} </h1>

<ul>
  {% for choice in choices %}
  <li> {{ choice.choice_text }} got {{choice.votes}} votes. </li>
  {% endfor %}
</ul>
<a href="/django/newpolls/detail">Return to poll</a>
```

## Finding the vote in views.py

```python
def votes(request):
    p = Poll.objects.get(id=1)
    try:
        selected_choice = p.choice_set.get(id=request.GET['choice'])
        request is an HttpRequest object
    except Choice.DoesNotExist:
        return render(request, 'newpolls/vote.html', {'error': True})
    request.GET is a method returning a dictionary of variable names and values, from the URL, eg.
    ..../votes?choice=1&poll=1
    Will give the dictionary:
    {"choice": 1, "poll":1}
```
def votes(request):
    p = Poll.objects.get(id=1)
    try:
        selected_choice = p.choice_set.get(id=request.GET['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the poll voting form.
        context = {
            'question': p.question,
            'choices': p.choice_set.all(),
            'message': "You didn't select a choice"
        }
        return render(request, 'newpolls/detail.html', context)
    else:
        selected_choice.votes += 1
        selected_choice.save()
        context = {
            'question': p.question,
            'choices': p.choice_set.all(),
        }
        return render(request, 'newpolls/vote.html', context)