# Software Testing Methodology

Lecture 8

Gregory S. DeLozier, Ph.D.

# Agenda

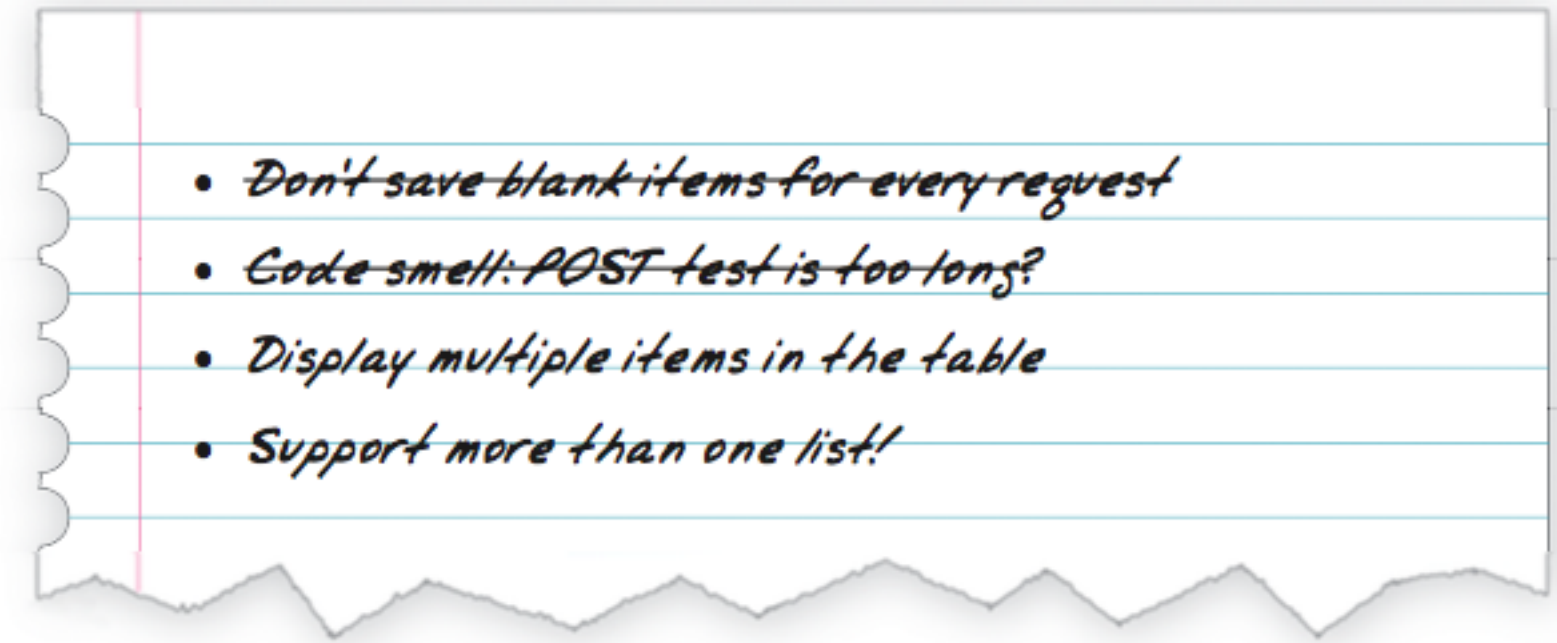
- Review homework assignments
- Catch up on highlights from the book Chapter 5 & 6
  - A lot of this is web programming technique. Interesting, but not here.
- Discuss content from Chapter 7 regarding Layout and Styling
  - This is hard to test
  - Gets relegated to manual testing very frequently
- Review future topics

# Separation of Concerns

```
def test_home_page_can_save_a_POST_request(self):
    request = HttpRequest()
    request.method = 'POST'
    request.POST['item_text'] = 'A new list item'

    response = home_page(request)

    self.assertEqual(Item.objects.count(), 1)
    new_item = Item.objects.first()
    self.assertEqual(new_item.text, 'A new list item')


def test_home_page_redirects_after_POST(self):
    request = HttpRequest()
    request.method = 'POST'
    request.POST['item_text'] = 'A new list item'

    response = home_page(request)

    self.assertEqual(response.status_code, 302)
    self.assertEqual(response['location'], '/')
```

# Keep a ToDo List

- Don’t let these things interrupt flow…but don’t lose them, either.



# Review TDD Concepts

- Red/Green/Refactor
  - Tests don’t work
  - Tests do work
  - Reorganize / improve code while constantly testing
- Triangulation
  - Adding a test to cause a failure to clarify a requirement.
  - This is often to justify an improvement you want to add to the code.
- Three Strikes rule
  - The third case really helps identify commonality.

# Test Isolation

- Separate tests from app code
- Notice that unit tests clean up
  - Test database create/teardown
  - “python3 manage.py test”
- Create functional_tests app
  - App is separated in django
  - Use “LiveServerTestCase”
  - Call with
    - “python3 manage.py test functional_tests”

```
.
├── db.sqlite3
├── functional_tests
│   ├── __init__.py
│   └── tests.py
├── lists
│   ├── admin.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   ├── 0002_item_text.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   ├── models.py
│   ├── __pycache__
│   ├── templates
│   │   └── home.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── superlists
    ├── __init__.py
    ├── __pycache__
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

# Code Changes for LiveServerTestCase

```
from django.test import LiveServerTestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

class NewVisitorTest(LiveServerTestCase):

    def setUp(self):
        [...]


    def test_can_start_a_list_and_retrieve_it_later(self):
        # Edith has heard about a cool new online to-do app. She goes
        # to check out its homepage
        self.browser.get(self.live_server_url)
```

# Running Tests

- Tests are now in more than one app.

- “python3 manage.py test”
  - Runs all tests in all apps.
- “python3 manage.py test <app>”
  - Runs the tests in “app”

- “python3 manage.py test functional_tests”
  - Runs the tests in “functional_tests” which is an app containing only tests

# YAGNI

- A bit about agile design
- Tendency to overdesign
  - “I might need to <….> someday, so I should…”

- Agile Response is:
  - “You Ain’t Gonna Need It”

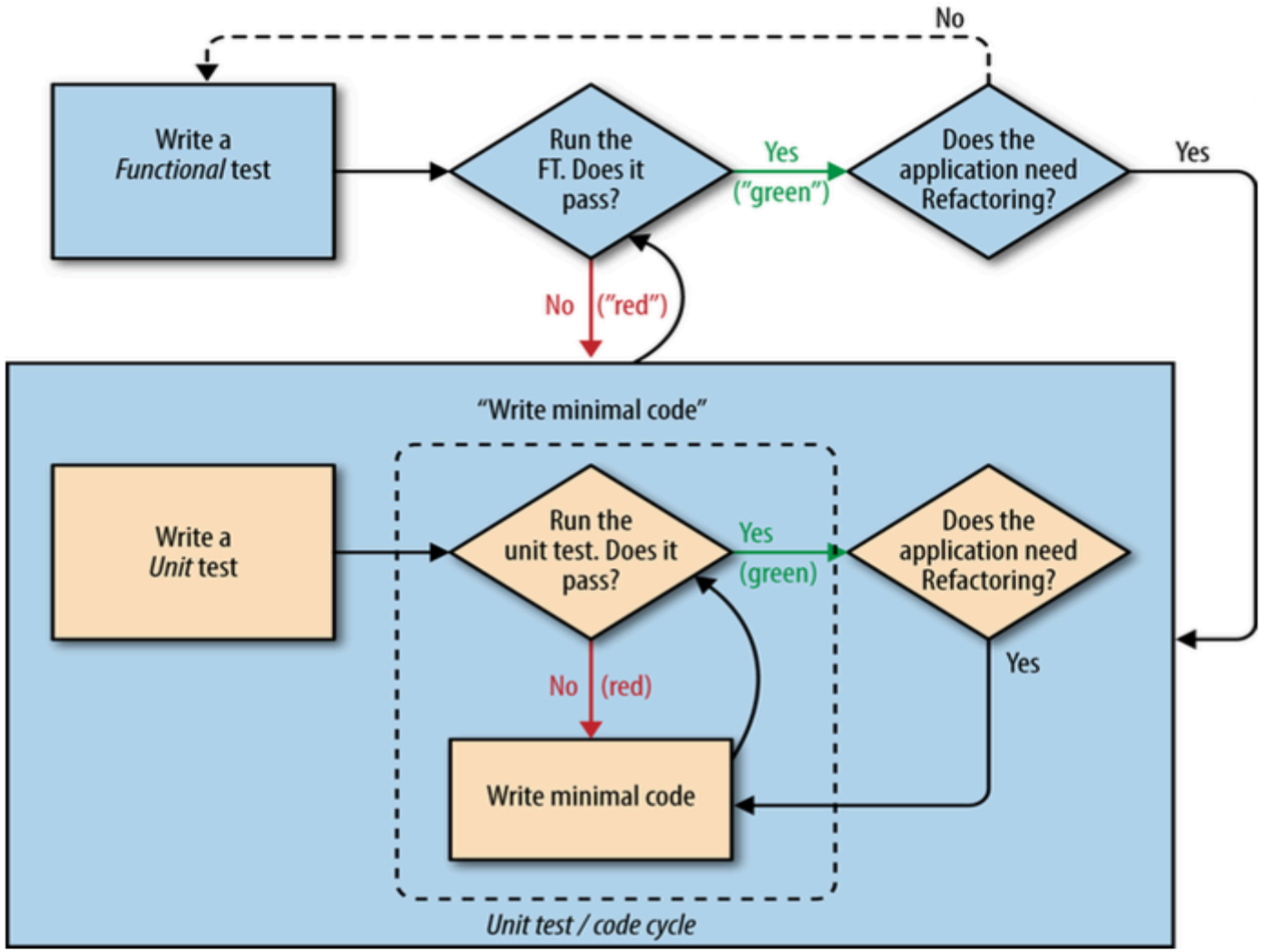
- If you really need it, the tests will eventually show it.
  - (Which is one of the reasons we test, of course.)

# Reviewing the TDD Cycle

- This is worth looking at again:

Last week:

- 1 week
- 100 LOC
- 500 LOTC
- 0 known bugs



# The Test Client

- The test client sets up objects and calls…

```
class ListViewTest(TestCase):

    def test_displays_all_items(self):
        Item.objects.create(text='itemey 1')
        Item.objects.create(text='itemey 2')

        response = self.client.get('/lists/the-only-list-in-the-world/') #❶

        self.assertContains(response, 'itemey 1') #❷
        self.assertContains(response, 'itemey 2') #❸
```

…of course, this fails…

# Passing the Test…

- Add the route…

```
urlpatterns = patterns('',
    url(r'^$', 'lists.views.home_page', name='home'),
    url(r'^lists/the-only-list-in-the-world/$', 'lists.views.view_list',
        name='view_list'
    ),
    # url(r'^admin/', include(admin.site.urls)),
)
```

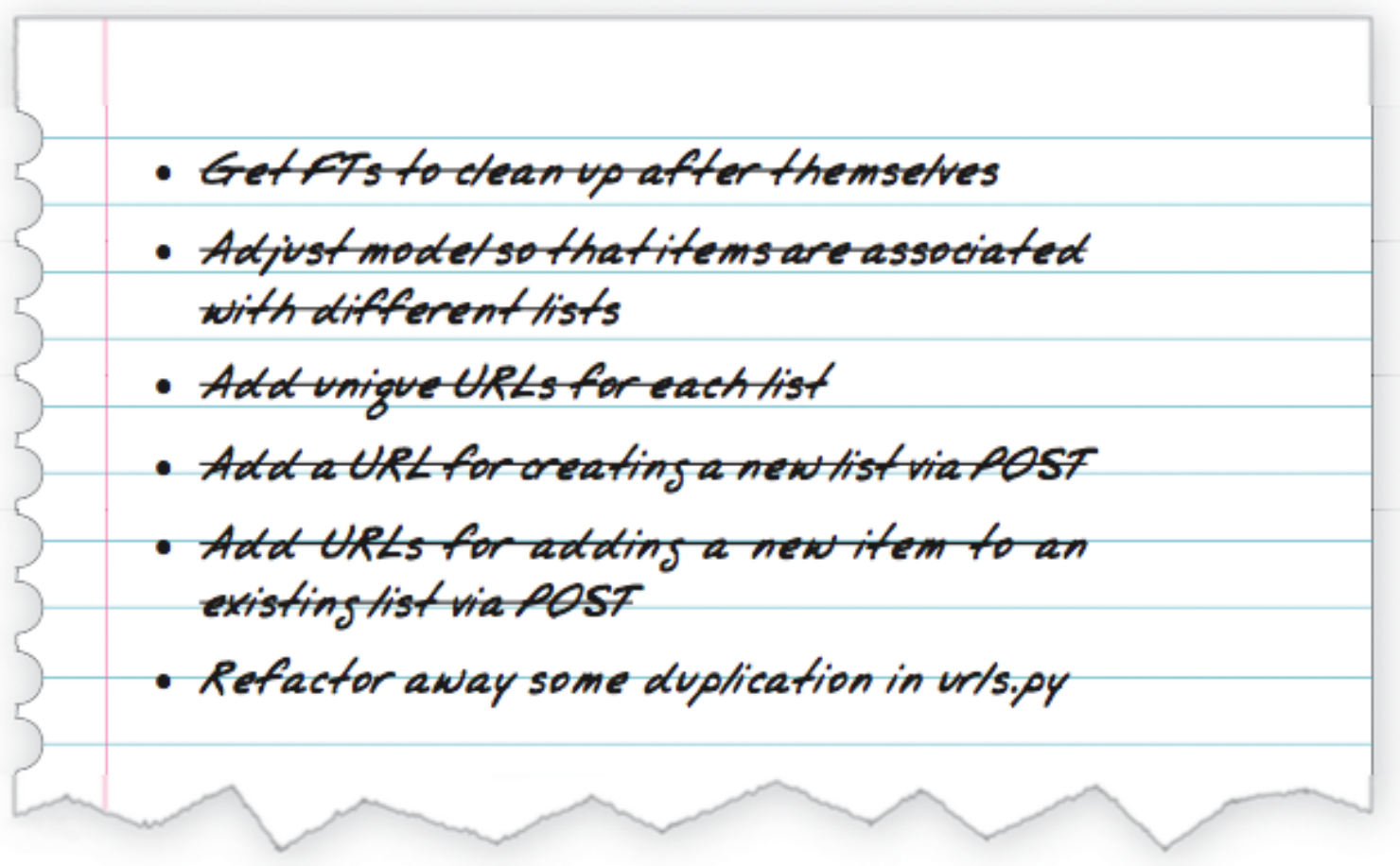
- Add the code…

```
def view_list(request):
    items = Item.objects.all()
    return render(request, 'home.html', {'items': items})
```

- …and so on until test passes.

# Things to Review at Home

- A lot of TDD:



# Getting the Code

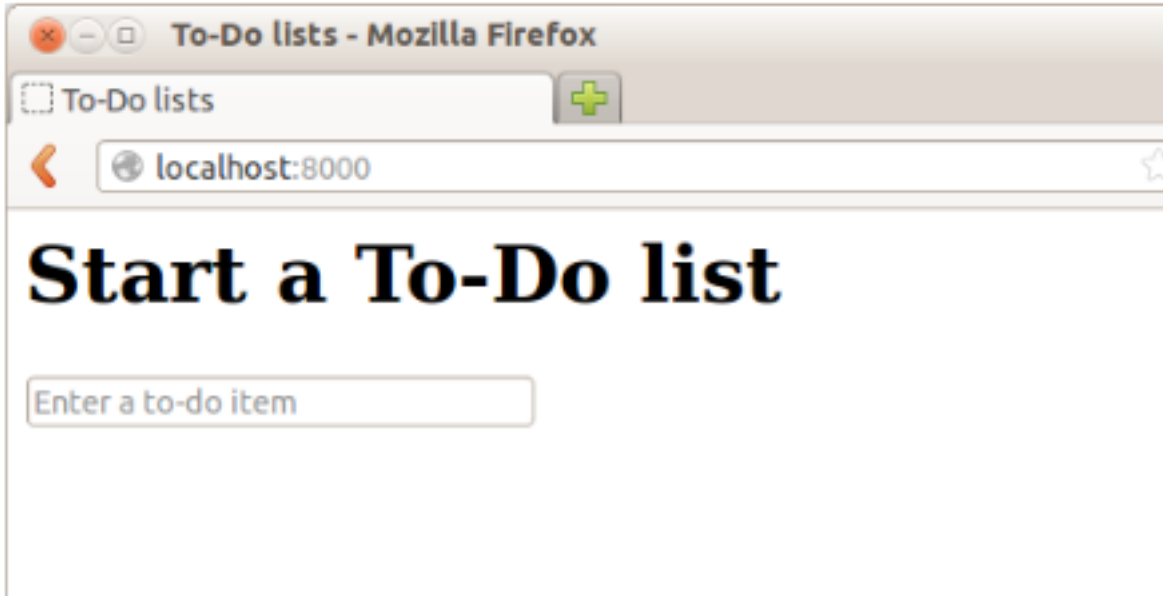
- *$ git clone* [*git@github.com:hjwp/book-example.git*](git@github.com:hjwp/book-example.git)
  - Or *$ git clone* [*https://github.com/hjwp/book-example.git*](https://github.com/hjwp/book-example.git)
- *$ git checkout chapter_06  (or chapter_07, etc)*
- That leaves you looking at Chapter 6 (or Chapter 7, etc) files.

```
Gregs-MacBook-Air:book-example greg$ ls -l
total 8
drwxr-xr-x   4 greg  staff  136 Oct 28 17:28 functional_tests
drwxr-xr-x  10 greg  staff  340 Oct 28 17:28 lists
-rw-r--r--   1 greg  staff  253 Oct 28 17:26 manage.py
drwxr-xr-x   6 greg  staff  204 Oct 28 17:28 superlists
```

- *$ python3 manage.py migrate*
- *$ python3 manage.py runserver*

# Testing Appearances

- Application isn't very pretty




- Test things that *cause* appearances.
- In this case, test the CSS and DOM information

# Testing DOM positioning

- Force a size
- Test an element's location
- This can be extended
  - Vertical alignment
  - Size relative to content
  - Responsive design

```python
def test_layout_and_styling(self):
    # Edith goes to the home page
    self.browser.get(self.live_server_url)
    self.browser.set_window_size(1024, 768)

    # She notices the input box is nicely centered
    inputbox = self.browser.find_element_by_id('id_new_item')
    self.assertAlmostEqual(
        inputbox.location['x'] + inputbox.size['width'] / 2,
        512,
        delta=5
    )
```

# CSS Frameworks

- Adding a CSS framework saves a lot of manual work
- Design work done in advance gives us *themes*
- CSS done in advance helps with reponsive design
- http://getbootstrap.com or on GitHub

```
$ wget -O bootstrap.zip https://github.com/twbs/bootstrap/releases/download/\
v3.1.0/bootstrap-3.1.0-dist.zip
$ unzip bootstrap.zip
$ mkdir lists/static
$ mv dist lists/static/bootstrap
$ rm bootstrap.zip
```

# Bootstrap as part of Django App

```
$ tree lists
lists
├── __init__.py
├── __pycache__
│   └── [...]
├── admin.py
├── models.py
├── static
│   └── bootstrap
│       ├── css
│       │   ├── bootstrap.css
│       │   ├── bootstrap.css.map
│       │   ├── bootstrap.min.css
│       │   ├── bootstrap-theme.css
│       │   ├── bootstrap-theme.css.map
│       │   └── bootstrap-theme.min.css
│       ├── fonts
│       │   ├── glyphicons-halflings-regular.eot
│       │   ├── glyphicons-halflings-regular.svg
│       │   ├── glyphicons-halflings-regular.ttf
│       │   └── glyphicons-halflings-regular.woff
│       └── js
│           ├── bootstrap.js
│           └── bootstrap.min.js
├── templates
│   ├── home.html
│   └── list.html
├── tests.py
├── urls.py
└── views.py
```

# Including Bootstrap in HTML

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="http://code.jquery.com/jquery.js"></script>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

# Create a Base Template

```
<html>
<head>
    <title>To-Do lists</title>
</head>

<body>
    <h1>{% block header_text %}{% endblock %}</h1>
    <form method="POST" action="{% block form_action %}{% endblock %}">
        <input name="item_text" id="id_new_item" placeholder="Enter a to-do item" />
        {% csrf_token %}
    </form>
    {% block table %}
    {% endblock %}
</body>
</html>
```

# Using the Base Template

*lists/templates/home.html.*

```
{% extends 'base.html' %}

{% block header_text %}Start a new To-Do list{% endblock %}

{% block form_action %}/lists/new{% endblock %}
```

# Using the Base Template, Again

*lists/templates/list.html.*

```
{% extends 'base.html' %}

{% block header_text %}Your To-Do list{% endblock %}

{% block form_action %}/lists/{{ list.id }}/add_item{% endblock %}

{% block table %}
    <table id="id_list_table">
        {% for item in list.item_set.all %}
            <tr><td>{{ forloop.counter }}: {{ item.text }}</td></tr>
        {% endfor %}
    </table>
{% endblock %}
```

# Integrating Bootstrap Header

- Put changes in <head> section of base.html

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
```

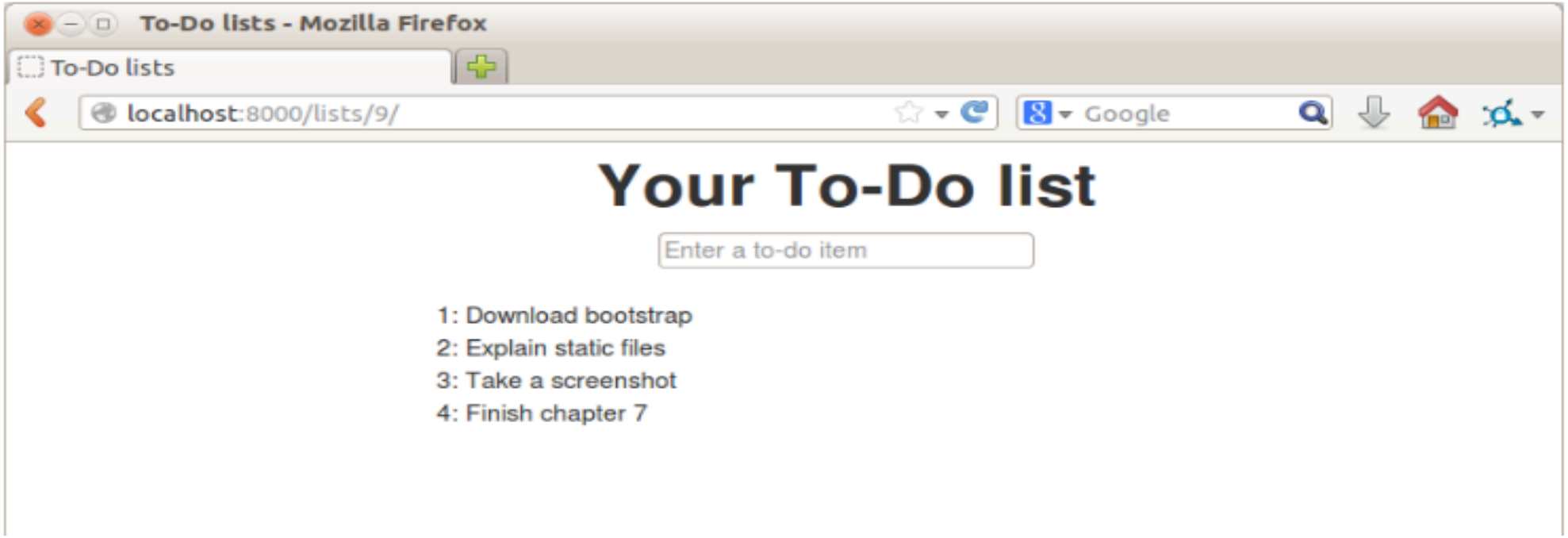
- Modify to use our static file location

```
<link href="/static/bootstrap/css/bootstrap.min.css" rel="stylesheet" media="screen">
```

# Using Bootstrap Classes

```
<body>
<div class="container">

    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <div class="text-center">
                <h1>{% block header_text %}{% endblock %}</h1>
                <form method="POST" action="{% block form_action %}{% endblock %}">
                    <input name="item_text" id="id_new_item"
                           placeholder="Enter a to-do item"
                    />
                    {% csrf_token %}
                </form>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            {% block table %}
            {% endblock %}
        </div>
    </div>

</div>
</body>
```

# With Bootstrap




- Still doesn't pass tests…
- Switch to StaticLiveServerTestCase, tests pass.

# Check out Bootstrap Components

- Use classes…

```
<div class="col-md-6 col-md-offset-3 jumbotron">
    <div class="text-center">
        <h1>{% block header_text %}{% endblock %}</h1>
        <form method="POST" action="{% block form_action %}{% endblock %}">
```

```
<input name="item_text" id="id_new_item"
       class="form-control input-lg"
       placeholder="Enter a to-do item"
/>
```

```
<table id="id_list_table" class="table">
```

# Using Custom CSS

- Add reference to the header

```
<head>
    <title>To-Do lists</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="/static/bootstrap/css/bootstrap.min.css" rel="stylesheet" media="screen">
    <link href="/static/base.css" rel="stylesheet" media="screen">
</head>
```

- Add some item modifications

```
#id_new_item {
    margin-top: 2ex;
}
```

# Demo Time…

- Switch to Chapter 7 branch
- Check out some of this stuff

# Future Topics

- We will cover the following additional chapters:
  - 8 – Deployment
  - 11 – Forms
  - 13 – Javascript
  - 15,16 – Mocking
  - 17 – Logging
  - 18 – Outside-In
  - 19 – Test Isolation
  - 22 – Test Architecture and Topics

# Regarding Software Engineering (Spring 16)

- We will be using this book for some topics in SWE
  - Continuous Integration
  - Deployment preparation
  - Deployment management and DevOps
  - Environment validation and testing
  - Authentication and Security
  - Database migration
  - A few other things here and there. ☺