{ Welcome, PyLadiesSF! }

Goals:

>>> have fun!

>>> learn about web development

>>> create something!

layout-ish:

1. walk through concepts & code
2. you try it out
3. repeat

Ruby on Rails vs Python and Django

Ruby vs Python

MVC = Model View Controller

err...

MTV = Model Template View

First: Installation packages

1. Open a Terminal/Command Line
2. sudo easy_install pip
3. pip install virtualenv
4. pip install virtualenvwrapper
5. pip install django
6: download git

Optional**: setup a GitHub account

Next: Setup Django Blog environment

In your command line/terminal, type:

```
django-admin.py startproject myblog
```

In your command line/terminal, type:

```
mkvirtalenv myblog
```

Con't: Setup Django Blog environment

In your command line/terminal, type:

```
$ mkvirtalenv myblog
```

In your command line/terminal, type:

```
$ cd myblog
```

```
C:\> dir myblog
```

Next: Setup Django Blog environment

In your command line/terminal, type:

```
$ python manage.py runserver
```

Navigate in your browser to:

http://localhost:8000

Configuration of our App

In the Text Editor of your choice, open up settings.py

Configure the database to sqlite3

Prepare the App

In your command line/terminal, type:

```
$ python manage.py syncdb
```

Again in your terminal:

```
$ python manage.py startapp blog
```

{ WHOA Hold up. }

Indentation is UBER important.

```
if tab is not tab:
    return 4 spaces
else:
    return IndentationError
```

Once more:

Indentation is UBER important.

Define your models

{ Open up models.py in your text editor. }

Define your models

```
from django.db import models

class BlogPost(models.Model):
    # how do we want to define a blog post?


class Comment(models.Model):
    # how do we want to define a comment?
```

Activate Admin site

open up settings.py and
uncomment three lines:

```
from django.contrib import admin
admin.autodiscover()
url(r'^admin/', include(admin.site.urls)),
```

add ‘blog’, to INSTALLED_APPS

Prepare the App

In your command line/terminal, type:

```
$ python manage.py sql blog
```

Again in your terminal:

```
$ python manage.py syncdb
```

Activate Admin site

open up settings.py and
uncomment one line under
INSTALLED_APPS:

```
'django.contrib.admin'
```

Pro-tip: a commented line
starts with a hashtag: #

Activate Admin site

open up blog/urls.py and
uncomment two lines:

```
from django.contrib import admin
admin.autodiscover()
```

Activate Admin site

navigate to localhost:8000/admin

login with the user/password you set up with earlier

BAM

Activate Admin site

create a file called admin.py and write the following:

```
from blog.models import BlogPost, Comment
from django.contrib import admin

admin.site.register(BlogPost)
admin.site.register(Comment)
```

Explore Admin site

{

Navigate to the Admin site and notice the changes.

It will be changing again :-D

}

Customize Admin site

```
class CommentAdmin(admin.ModelAdmin):
        list_display = ["post", "author", "created"]

class BlogPostAdmin(admin.ModelAdmin):
        def preview(self, obj):
            return obj.body[0:100]
        list_display = ["title", "author", "preview"]
        search_fields = ["title"]


admin.site.register(Comment, CommentAdmin)
admin.site.register(BlogPost, BlogPostAdmin)
```

Customize Admin site

{ Revisit the Admin site.
Write a blog post x3. }

Don't forget to pimp us out: @PyLadiesSF #hackingPyLadiesSF

Customizing URLs

URLs allow the front-end to talk to the back-end. If you want to see a detailed post, clicking on it will send info to the SinglePost view.

Customizing Views

We've outlined two views so far:

BlogView for a list of posts

&

SinglePost for viewing a single post with comments

Customizing Views

In my code, I also have a few other experimental things like AddComment. This is so I can make a form for commenting on posts.

Unfortunately I could not fight the sand man any longer and had to go to bed before I could get it to work.

Comment Form

Comment form is just based off our model, nearly verbatim.

Again, it isn't rendered anywhere [yet].

Templates

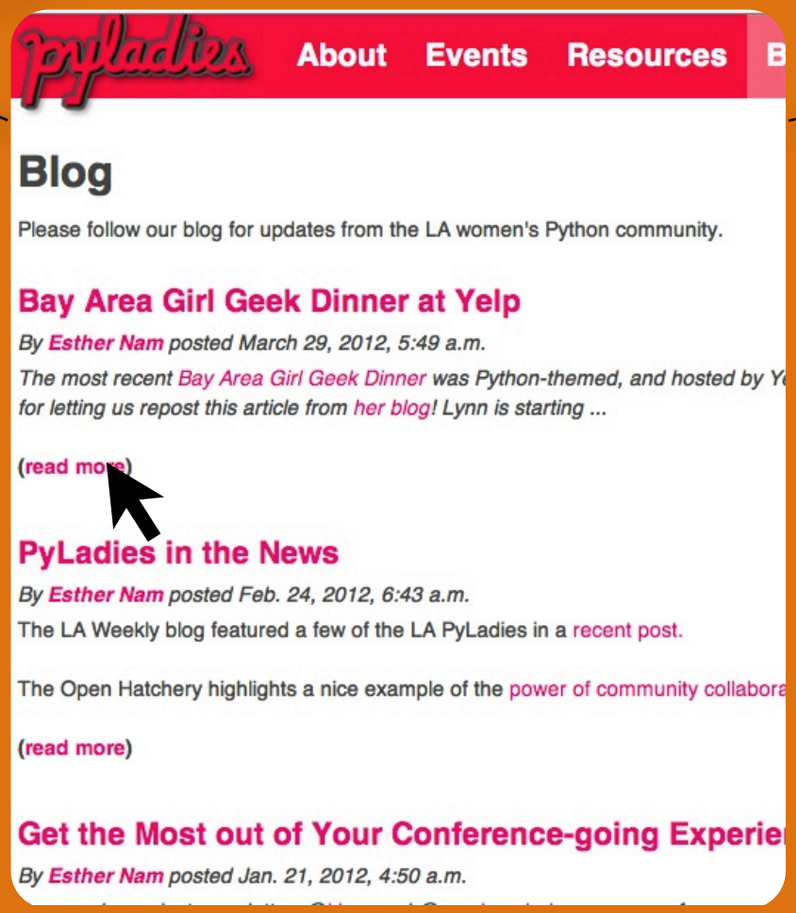
Templates need to be in their own folder under /blog/.
Templates connects the data with the user, populating blog post info with the view they requested via the URL. Here you also incorporate design with HTML & CSS.

models.py

database

views.py

urls.py



template
.html

models.py

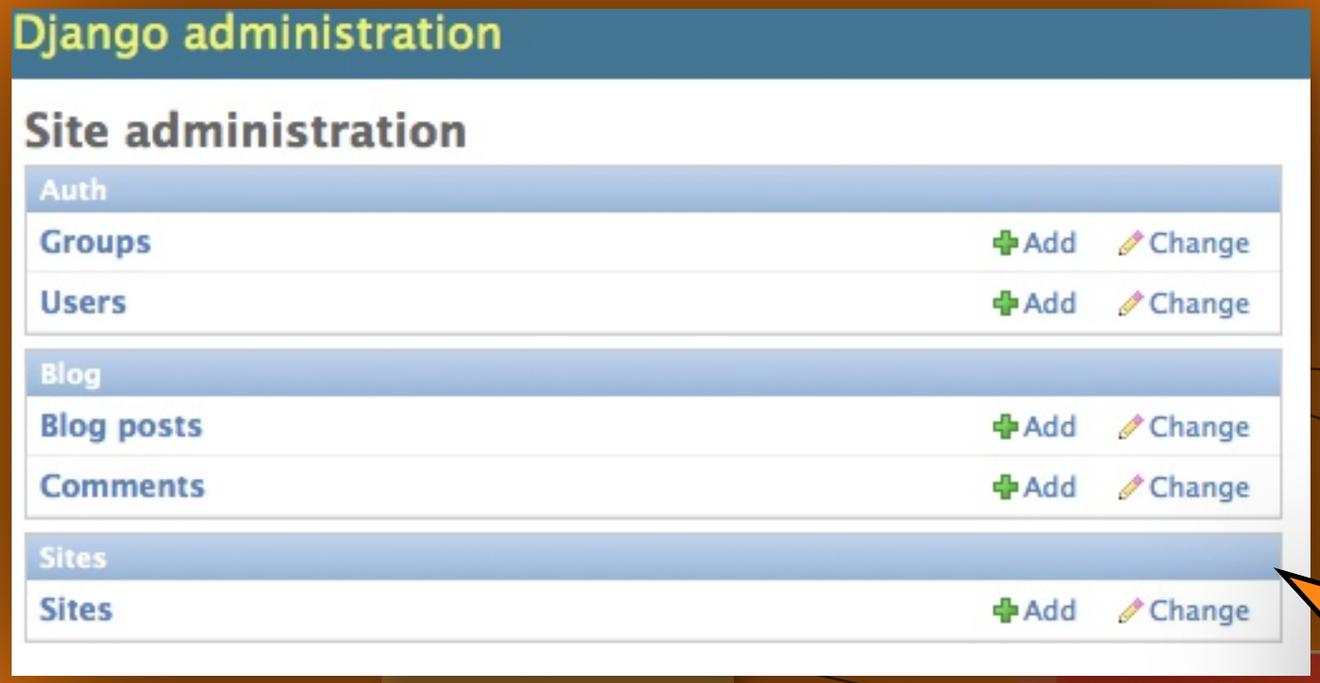
database

urls.py

We first define what data we want our database to house by outlining our models. We outline our models based on the type of site we want to build, e.g. BlogPosts and Comments, with attributes such as title, author, date_created, etc.

models.py

database



template.html

urls.py

"Behind the scenes" you have your admin page, where you can populate your database with data, aka write your blog posts.

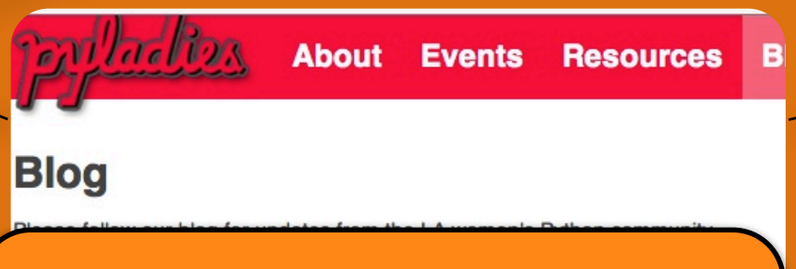
You can also make yourself seem awesome by writing favorable comments :)

models.py

database

views.py

urls.py



template
.html

The user clicks on a link, which is initially handled by the urls.py file. If there is no url defined in urls.py, or the url search mechanism can not return a match for the requested information, it hands this error issue to views.py, which in turns gives the user a 404 error: page not found.

models.py

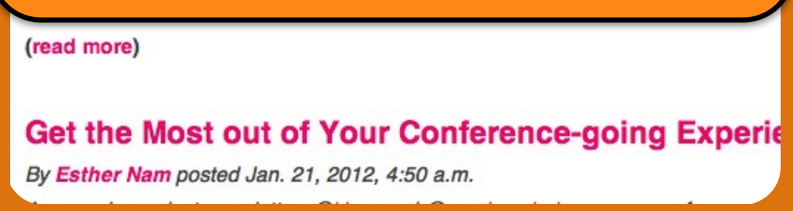
database

views.py

urls.py

If the url requested is in the urls.py file, and the actual data request is in the database, then success! The url will link to a view defined in views.py. The view then processes how the data requested will be viewed, e.g. the most recent blog posts first, or a detailed-view of one single post.

template .html

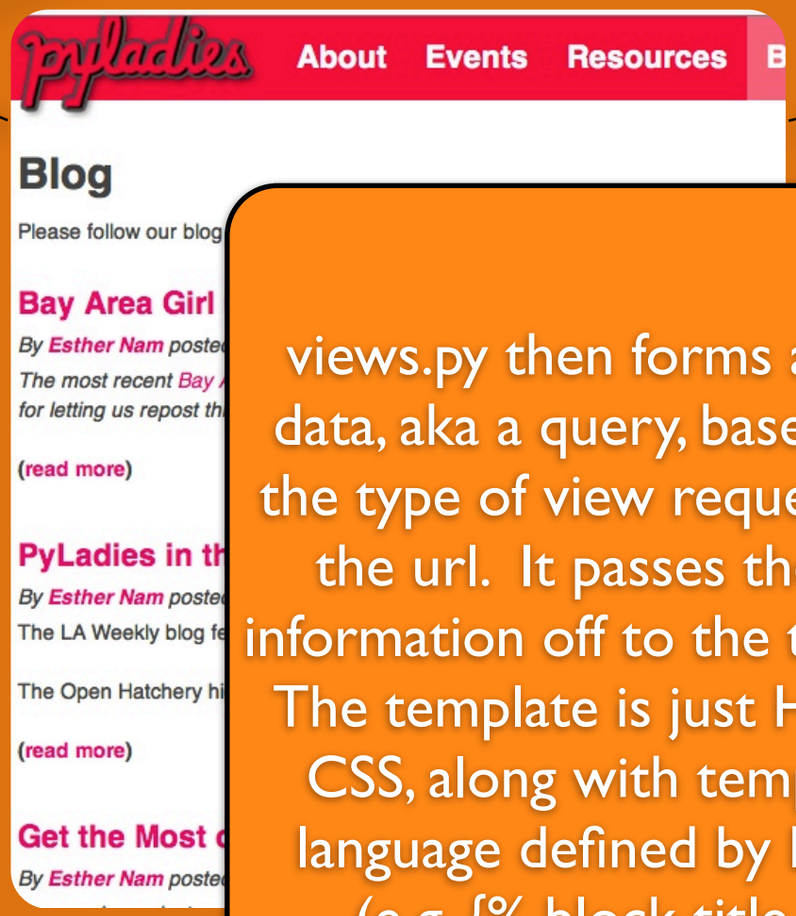


models.py

database

views.py

urls.py



template
.html

views.py then forms a set of data, aka a query, based off of the type of view requested via the url. It passes the data information off to the template. The template is just HTML & CSS, along with templating language defined by Django (e.g. {% block title %})

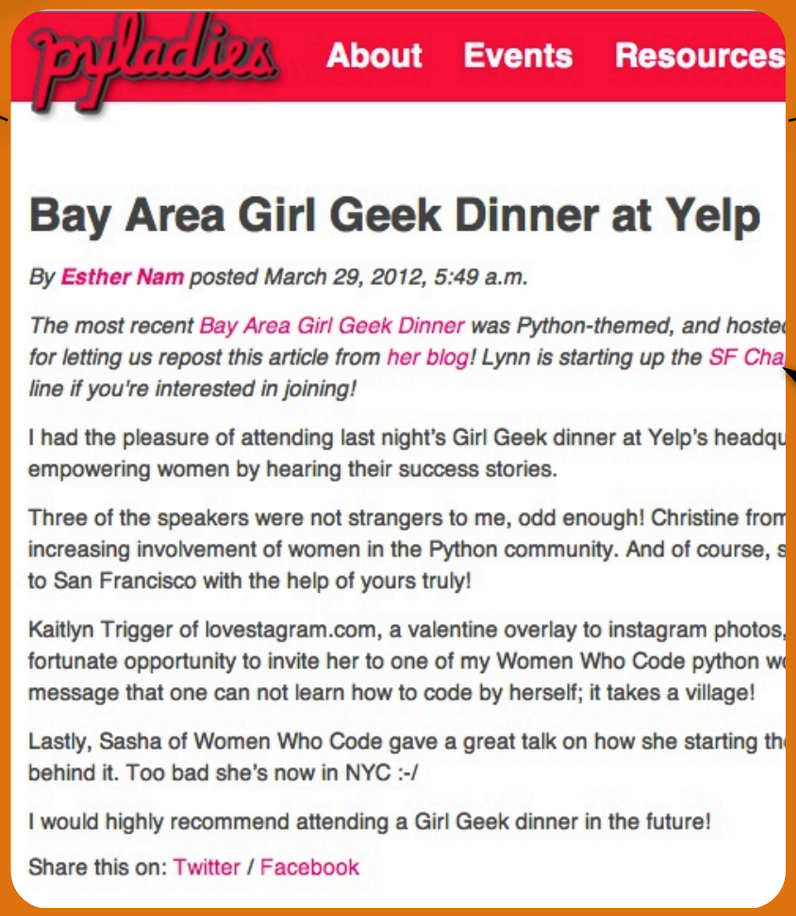
models.py

database

views.py

urls.py

template
.html



The template populates the page as dictated by the view with the data requested via the url.