PYLADIES PRESENTS:

Build your own blog with Django!

{{ GOALS }}

Have fun

Learn about web development

Create something

{{ LAYOUT }}

Quick intro to Python

Overview of Django Framework

Code together

{{ VOCAB }}

shell / terminal / console

python

django

text editor

Sunday, November 25, 12

I use these words interchangeably: shell, terminal, and console. They are one in the same, essentially, and provide the ability to execute code, programs, and commands just from typing (as opposed to double clicking on an icon to launch a program).

Python is a programming language that can be used to write websites, simple scripts, native computer programs, networks, etc. A computer must have an interpreter on its machine to understand Python. That's why it is required to 'have python on your computer' when it really is just downloading the ability to understand the Python language. Most *nix computers already have the ability to interpret Python, but others may require downloading.

Django is a web framework built using Python. A web framework provides the scaffolding to making a website so we don't have to reinvent the wheel.

A text editor provides the ability to write the actual code that you can execute through the shell, or as a website/program/etc itself. It's like Microsoft Word for your code. There are many text editors out there; popular ones include Vim, Emacs, gedit, Sublime Text 2, Notepad++. I switch back and forth between Sublime Text 2 and Vim.

{{ VOCAB }}

Data types / Variables

Function / Method

Class

Modules / Packages

Sunday, November 25, 12

To note: in python: everything is an object!
Data types:
- Numbers (integers & float/decimal numbers)
- Strings (bits of text; chars, words, numbers, whitespace, punctuation)
- Booleans (True/False)
- Lists (store data where order matters; indexed at 0)
- Tuples (like a list, but can not change it - immutable)
- Dictionaries (key/value pairs, order doesn't matter)
- A list can hold dictionaries, lists, strings, numbers, etc
- A dictionary can hold dictionaries, lists, strings, etc

Functions
- like math!
- A function can take no arguments (data types or another function) or as many as one wants
- Returns nothing, a value, True/False, another function, anything

Classes
- Group like objects together
- Inherit from another class
- "mixin" to alter behavior from inheritance
- Multiple inheritance

Modules / Packages
- A module is one file; a package is a collection of files
- Django is a package
- Import modules from a package when you need them (rather than reinventing the wheel!)
- Python standard library has a great set of modules

{{ PYTHON }}

Let's try in our Terminal/Console.

Type `python` into the shell.

{{ PYTHON }}

```
$ python
Python 2.7.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>>
```


When in the `python` shell, have some fun:

```
>>> x = 1
>>> y = 2
>>> x + y
3
>>> my_list = [‘red’, ‘blue’, ‘green’]
>>> my_list
[‘red’, ‘blue’, ‘green’]
>>> my_list.append(‘yellow’)
>>> my_list
[‘red’, ‘blue’, ‘green’, ‘yellow’]
>>> my_dict = {‘lynn’: ‘red’, ‘ana’: ‘green’, ‘stacy’: ‘blue’}
>>> my_dict
{'lynn': 'red', 'ana': 'green', 'stacy': 'blue'}
>>> another_dict = { 1: ‘red’, 5: ‘purple’, 3: ‘yellow’}
>>> another_dict
{1: 'red', 3: 'yellow', 5: 'purple'}
>>> another_dict.keys()
[1 ,3, 5]
>>> another_dict.values()
[‘red’, ‘yellow’, ‘blue’]
```

CONTROL+D will get you out of the shell. (or COMMAND+D if you use a Mac)

{{ DJANGO }}

Django is to Python

as

Rails is to Ruby

Sunday, November 25, 12

This doesn't have to make since if you're not familiar with Ruby.

It's just: Python & Ruby are separate, different, but similar languages.

Django is a framework that uses Python to make a website (the server side/backend of a website).
Rails is a framework that uses Ruby to make a website (again, the server side/backend of a website).

Django & Rails have different communities behind it, and there are differences in how these frameworks are built. But you can't easily tell if a website is built using Django or Rails.

{{ DJANGO }}

MVC =

Model - View - Controller

Sunday, November 25, 12

If you go to the end of this presentation, with all the orange slides, the graphic explains an overview of MVC. Just simply know that Django employs a MVC framework, and this type of framework is very popular not in just websites but elsewhere.

The idea is that you separate your data from the user. Simple as that.

{{ LET'S GET STARTED }}

Quick note:

do NOT copy text on these slides into your text editor. Write it out yourself.

Sunday, November 25, 12

If you go to the end of this presentation, with all the orange slides, the graphic explains an overview of MVC. Just simply know that Django employs a MVC framework, and this type of framework is very popular not in just websites but elsewhere.

The idea is that you separate your data from the user. Simple as that.

{{ LET'S GET STARTED }}

Start a Python shell

```
$ python
```

And type 'django'

```
>>> django
```


This means django is installed!

To exit out of the Python shell, press CONTROL+D (at the same time) (or COMMAND+D if you use a Mac)

{{ LET'S GET STARTED }}

Back in the terminal (NOT in the python shell), type:

```
$ django-admin.py startproject RuPyWorkshop
```

Sunday, November 25, 12

This starts a website in Django.

django-admin.py is a python file.
startproject is a command that the django-admin.py file will understand.

'RuPyWorkshop' is your creative name to name your site/project. Follow what I wrote to avoid errors, but when you're ready to go off these training wheels, this is where you name your site.

Notice after you run this command, you have a bunch of files that were just created.

Related tutorial portion: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-a-project

{{ RUNSERVER }}

On the terminal/console:

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

In a web browser, go to:

```
http://localhost:8000
```

BAM it works.

Sunday, November 25, 12

Django uses Python's ability to run a server RIGHT ON YOUR COMPUTER. (How awesome is that?!) Because it's only on your computer, only you can see it. If someone else went to http://localhost:8000 on their computer, they would not see what you're seeing.

You will need to stop the server from running. In your terminal, press CONTROL+C (at the same time). (or COMMAND+C if you use a Mac)

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#the-development-server

{{ SETTINGS.PY }}

Open up settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'rupyblog.db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

Sunday, November 25, 12

The `django-admin.py startproject RuPyWorkshop` command created a settings.py file. This is where you configure your database. In our workshop, we elected to use sqlite3 (sqlite is often pre-installed on computers, and it's also lightweight).

Just fill out your database portions like I have on the slide. Database setup is continued on next slide.

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#database-setup

{{ SYNCDB }}

Back on the terminal/console:

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

and follow prompts.
BAM your DB is set up.

Sunday, November 25, 12

In order for Django to recognize the code that you just wrote, you will have to invoke `$ python manage.py syncdb` from your terminal.
IMPORTANT: make sure you are *inside* the top-level folder of your project. You may have to do this: `$ cd {{ mysite }}`

Note: remember the details you input for username & password.

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#database-setup

The tutorial link says for you to change `TIMEZONE` - go for it. In Brno, we're considered `Europe/Prague` time zone.

{{ STARTAPP }}

Back on the terminal/console:

```
$ python manage.py startapp MyBlog
```

```
$ cd MyBlog
```

```
$ ls
```

BAM more files!

Sunday, November 25, 12

This also needs to be done within the top-level directory of your project.

This command, `$ python manage.py startapp MyBlog` will start the actual blog project. You can have multiple apps within your website. For instance, later, you can make a second app within the same {{ mysite }} project, like a Polls app, or an app to share your images. But that's for someone who is ready to take the next step.

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-models

{{ MODELS.PY }}

Open up models.py

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=60)
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)

    def __unicode__(self):
        return self.title
```

Sunday, November 25, 12

Now we need to edit our models.py file. This will map to our database that we set up, and tells our database how exactly we want to structure our data (which in this application, is our blog post).

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-models

HOLD UP

[next slide]

{{ PSA }}

Notice that indentation.

Indentation is key in Python

a.k.a. that lovely whitespace.

{{ PSA }}

No tabs.

Use spaces.

4 spaces per indentation.

Sunday, November 25, 12

You must keep multiples of 4 spaces when you have multiple indentation levels.

(You may see some code that is written with multiples of 2, which is fine, but less common. The idea is to be consistent).

RETURNING TO YOUR REGULARLY SCHEDULED PROGRAMMING.

{{ SETTINGS.PY }}

Return to settings.py file.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'MyBlog',
)
```


Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#activating-models

{{ SYNCDB }}

Back on the terminal/console:

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

again. Then do:

```
$ python manage.py shell
```

Let's play around a little.

Sunday, November 25, 12

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/#activating-models

Ok so let's play around in the Python shell. Comments are in red, code within shell is black.

```
>>> from MyBlog.models import Post
# NOTE: if you get an error here, make sure you are consistent with naming. Refer to how you ran $ python manage.py startapp MyBlog
>>> Post.objects.all()
[]
>>> from datetime import datetime
>>> p = Post(title="Write your own title here!", body="Write the body of your post here, anything you'd like! How about tell how awesome it is to create your own Django application!", created=datetime.now())
# NOTE: syntax is really important here. Mind your quotations, your equal signs, and your open/close parethesis.
>>> p.save()
>>> p.id
1
>>> p.title
"Write your own title here!"
>>>p.created
datetime.datetime(2012, 11, 25, 13, 59, 59, 520594) # This will show correctly on your website. This is how Python reads time when using datetime.now(), as Year, Month, Day, Hour, Minute, Second, Millisecond)
>>>Post.objects.all()
[<Post: Write your own title here!>]
```

now CONTROL+D will close out of the shell. (or COMMAND+D if you use a Mac)

{{ SETTINGS.PY }}

Return to settings.py file.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'MyBlog',
)
```

Sunday, November 25, 12

Now, we want to see the Admin portion that Django gives us. Let's uncomment 'django.contrib.admin' line from the INSTALLED_APPS in settings.py (do not uncomment 'django.contrib.admindocs', that is the wrong one!).

NOTE: Commas are SUPER important. The INSTALLED_APPS variable is a python tuple (I briefly went over that in class).

{{ ADMIN }}

Something is missing.

Create MyBlog/admin.py

```
from django.contrib import admin

from MyBlog.models import Post

class PostAdmin(admin.ModelAdmin):
    search_fields = ['title']

admin.site.register(Post, PostAdmin)
```


This is a total of 5 lines of code, and it tells the Admin portion of Django to display the Blog application so *you* can do your blogging from the admin site.

{{ ADMIN }}

Back on the terminal/console:

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

In a web browser, go to:

`http://localhost:8000/admin`

woah.

Again, remember to shut down the server. CONTROL+C at the same time (or COMMAND+D if you use a Mac)

{{ URLS }}

Reopen urls.py
And edit for the following:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
)
```


You will have to uncomment three lines:

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

This is so when you type in 'http://localhost:8000/admin' the URL will be recognized, and directed to your newly created admin.py file from the previous slide.

{{ ADMIN }}

Back on the terminal/console:

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

In a web browser, go to:

`http://localhost:8000/admin`

woah.

Sunday, November 25, 12

The login is the username & password you created when you ran `$ python manage.py syncdb` for the first time.

You should see Site Administration, that includes Groups, Users, Sites, and Posts - for your blog post! If you click on Post, you should even see your first entry we wrote a few slides back from`$ python manage.py shell`. Feel free to add more posts! This is where you will be blogging from. This is also where you can grant access to other users so they can blog under your website too.

To terminate the server, press CONTROL+C at the same time. (or COMMAND+D if you use a Mac)

{{ URLS }}

Now go to:

`http://localhost:8000`

Missing something...

We can’t really *see* our post. Yet.

{{ URLS }}

Reopen urls.py
And edit for the following:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',
    url(r'^$', ‘MyBlog.views.home’, name='home'),
    url(r'^(\d+)/$', ‘MyBlog.views.post’, name='post'),
    url(r'^admin/', include(admin.site.urls)),
)
```

Sunday, November 25, 12

You’ll notice that there are two lines about the admin URL.

The first URL the basic ‘site’: http://localhost:8000/. If you were to deploy, or ‘make live’, your blog website where other people can access and read your blog, it would look like http://www.my-awesome-rupy-blog.com/

This first URL maps the main part of the site to a view called ‘MyBlog.views.home’. We have not created the view yet, but we will.
the `name=‘home’` portion is 1) for you to easily remember what the URL does, and 2) can be a variable that is passed on to our Templates (the HTML code), in the future.

The second URL maps a specific post to be displayed. It will look like http://localhost:8000/1 or http://localhost:8000/15 (or however many posts you have. When deployed, using the same example as above, it would look like http://www.my-awesome-rupy-blog.com/1 and http://www.my-awesome-rupy-blog.com/15.

Again, here, the second URL maps a single post to ‘MyBlog.views.post’. Again, we haven’t created the view yet, and we will. Same with the `name=‘post’` variable.

URLs defined or NOT defined here will tell Django how to handle requests. If someone on your website looks for post #21, and you only have 19 posts, it would give an error (specifically, a 404 Error, Page Not Found).

{{ ADMIN }}

Back on the terminal/console:

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

In a web browser, go to:

```
http://localhost:8000/
```

Hrmph. No Views.

Well, duh. We haven't written any yet!

{{ VIEWS }}

Open views.py...

And let's write two views together.

Sunday, November 25, 12

Here, it will be easier to look at code rather than have the code be on the slide.

Please navigate here: https://github.com/econchick/PyLadiesBYOBlog/blob/master/RuPyWorkshop/MyBlog/views.py

Again, rather than copy & paste, I suggest you type it in yourself.

{{ TEMPLATES }}

Make a directory under the Parent Directory (the 1st RuPyWorkshop):

`templates`

And under *that* directory, make:

`blog`

/RuPyWorkshop/templates/blog
The next slide shows the overall directory structure.

{{ DIRECTORIES }}

```
└── RuPyWorkshop/
    ├── MyBlog/
    |   ├── __init__.py
    |   ├── admin.py
    |   ├── models.py
    |   ├── tests.py
    |   ├── views.py
    ├── RuPyWorkshop/
    |   ├── __init__.py
    |   ├── settings.py
    |   ├── urls.py
    |   ├── wsgi.py
    ├── manage.py
    ├── rupyblog.db
    └── templates/
        └── blog/
```


This is what your directory structure should look like. Note that the lines that finish with a '/' after them mean they are a directory

{{ TEMPLATES }}

Download some templates

https://gist.github.com/4085195

https://gist.github.com/4085203

https://gist.github.com/4085210

Just copy each into it's own file, and:
- Save the first one under RuPyWorkshop/templates/blog/ as `base.html`
- Save the second one under RuPyWorkshop/templates/blog as `list.html`
- Save the third one under RuPyWorkshop/templates/blog as `post.html`

Related portion of the Django tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial03/#write-views-that-actually-do-something
NOTE: I am skipping over the templating language of Django, but I highly encourage you to read the above link to notice how the data is populated dynamically into HTML.

OPTIONAL: You may elect to get a disqus (http://disqus.com) account, which allows you to add a module on your post.html template for comments. In the post.html, copy the JavaScript code that was given to you when you created an account and a project, and paste it under <!-- Comments -->. It's that easy.

{{ TEMPLATES }}

Make a directory under the Website Directory (not Parent):

`static`

And under that directory, make:

`css`

`images`

RuPyWorkshop/RuPyWorkshop/static/css
RuPyWorkshop/RuPyWorkshop/static/images
The next slide shows the overall directory structure.

{{ DIRECTORIES }}

```
└── RuPyWorkshop/
    ├── MyBlog/
    |   ├── __init__.py
    |   ├── admin.py
    |   ├── models.py
    |   ├── tests.py
    |   ├── views.py
    ├── RuPyWorkshop/
    |   ├── __init__.py
    |   ├── settings.py
    |   ├── static/
    |   |   ├── css/
    |   |   ├── images/
    |   ├── urls.py
    |   ├── wsgi.py
    ├── manage.py
    ├── rupyblog.db
    └── templates/
        └── blog/
            ├── base.html
            ├── list.html
            └── post.html
```


This is what your directory structure should look like. Notice the addition of three folders, ‘static’ that lives under ‘RuPyWorkshop’, and ‘css’ and ‘images’ that live under static.

{{ TEMPLATES }}

Save some pictures & css

`2px.png` - https://gist.github.com/4085215

`bullet.png` - https://gist.github.com/4085219

`pages.png` - https://gist.github.com/4085230

`main.css` - https://gist.github.com/4085231

Sunday, November 25, 12

Save the first three files in your /static/images folder as the following:

/RuPyWorkshop/RuPyWorkshop/static/images/2px.png
/RuPyWorkshop/RuPyWorkshop/static/images/bullet.png
/RuPyWorkshop/RuPyWorkshop/static/images/pages.png

Save the last file (main.css) in:
/RupyWorkshop/RuPyWorkshop/static/css/main.css

FYI: you just copied the 'code' of images (the png files). When translating pixels, there are different ways to represent them in code. Here, we have hexidecimal 'dump' of what makes up these photos. They will render something more visually appealing then numbers and letters, I promise!

FYI: CSS is Cascading Style Sheets - it allows the web designer to describe the look and format of a website (rather than repeating herself through every HTML file).

{{ SETTINGS.PY }}

Reopen settings.py
and add these few bits.

Refer to notes for specific line numbers.

https://gist.github.com/4144268

Sunday, November 25, 12

Last few tweaks to settings.py file.

02 - add the new import statement.
04 - add the defining of the PROJECT_DIR for later use, using what we imported.
30 - notice I did change the TIME_ZONE, it's not required here
51 - edit this line for MEDIA_ROOT
62 - edit this line for STATIC_ROOT
66 - edit this line for STATIC_URL
73 - edit this line for STATICFILES_DIR
113 & 114 - add the two directories we created for templates: 'templates' and 'templates/blog'

{{ COLLECTSTATIC }}

```
$ python manage.py collectstatic
```


This command helps Django organize the static files (as pointed to in the settings.py file) and organize them to the STATIC_ROOT file.

It may prompt you, asking “ARE YOU SURE?” just say yes.

You will also see a lot more files produced. This is because Django stuffs in some javascript etc for the Admin site. Don’t delete these files.

{{ TA-DA }}

One last time

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

You should see your blog!

{{ M-V-C }}

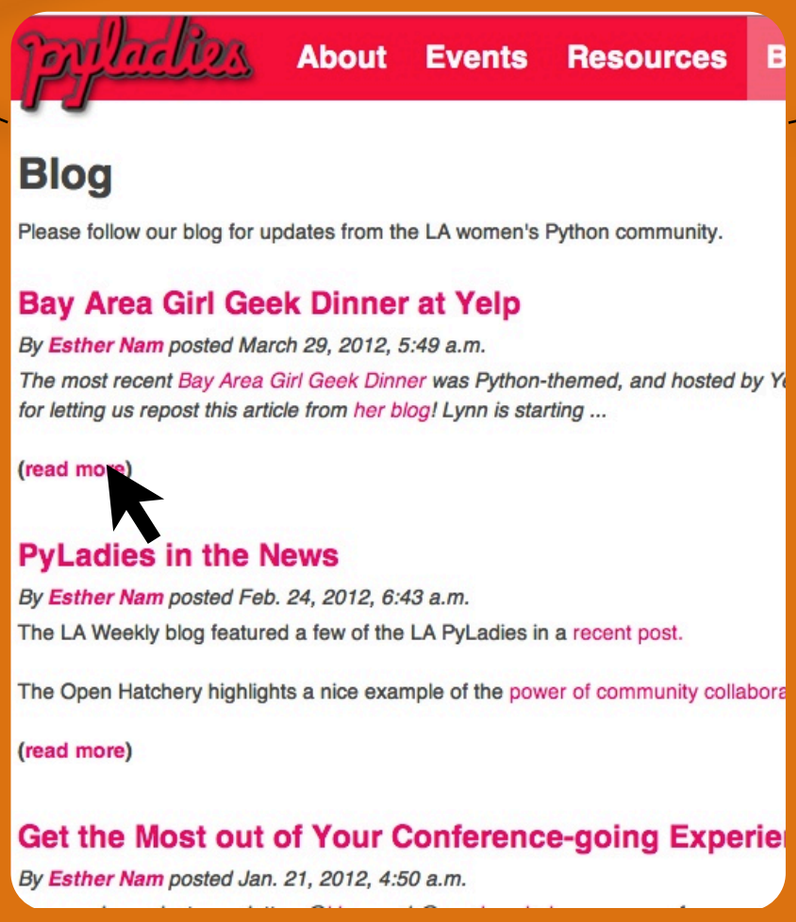
The next slides are a general overview of how Model-View-Controller works in Django

models.py

database

views.py

urls.py



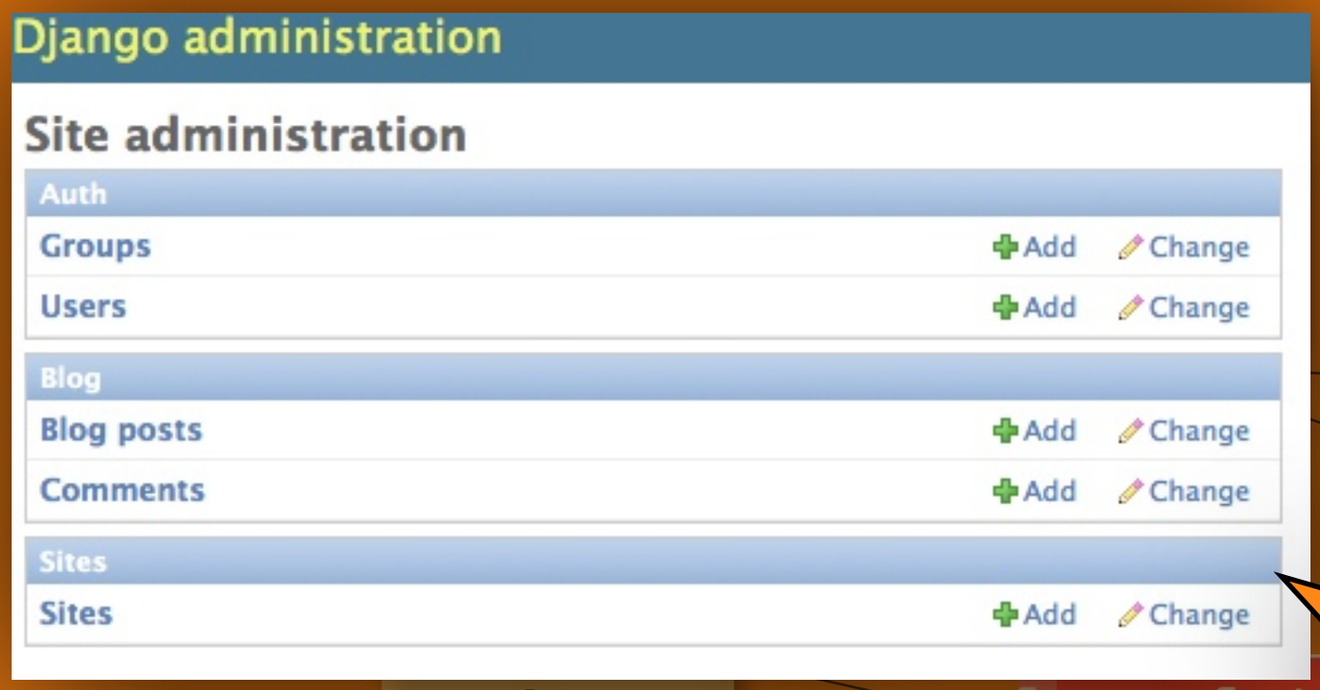
template
.html

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.

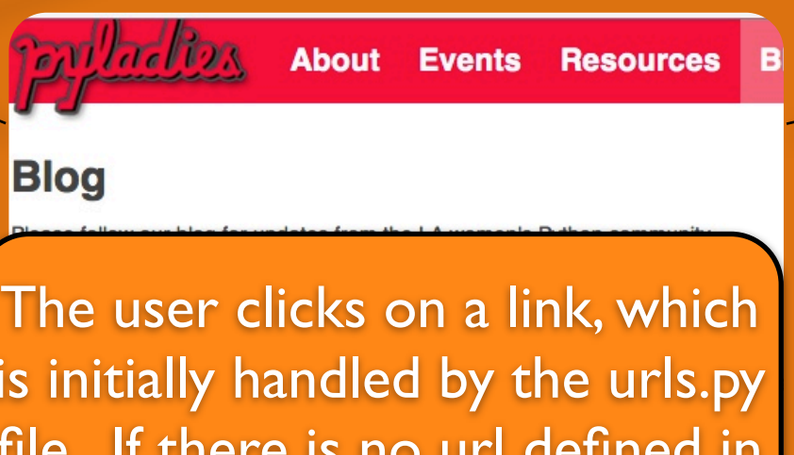
You can also make yourself seem awesome by writing favorable comments :)

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

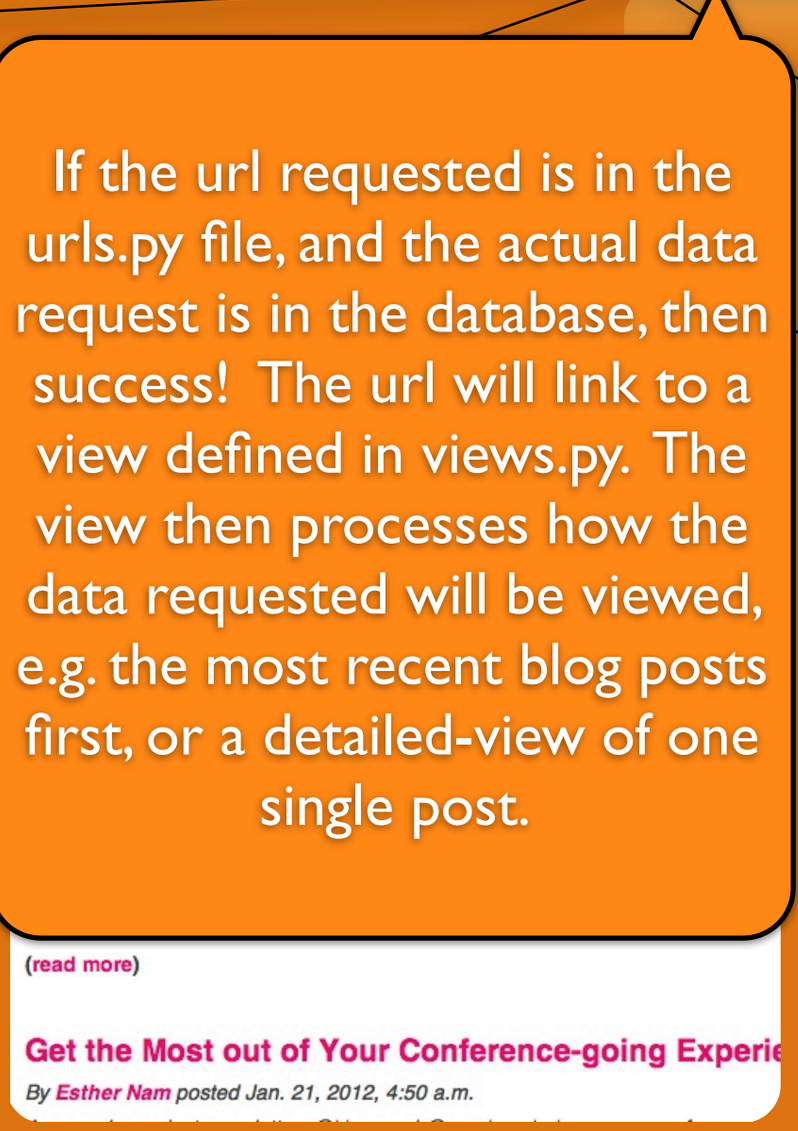
database

views.py

urls.py

template
.html

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.



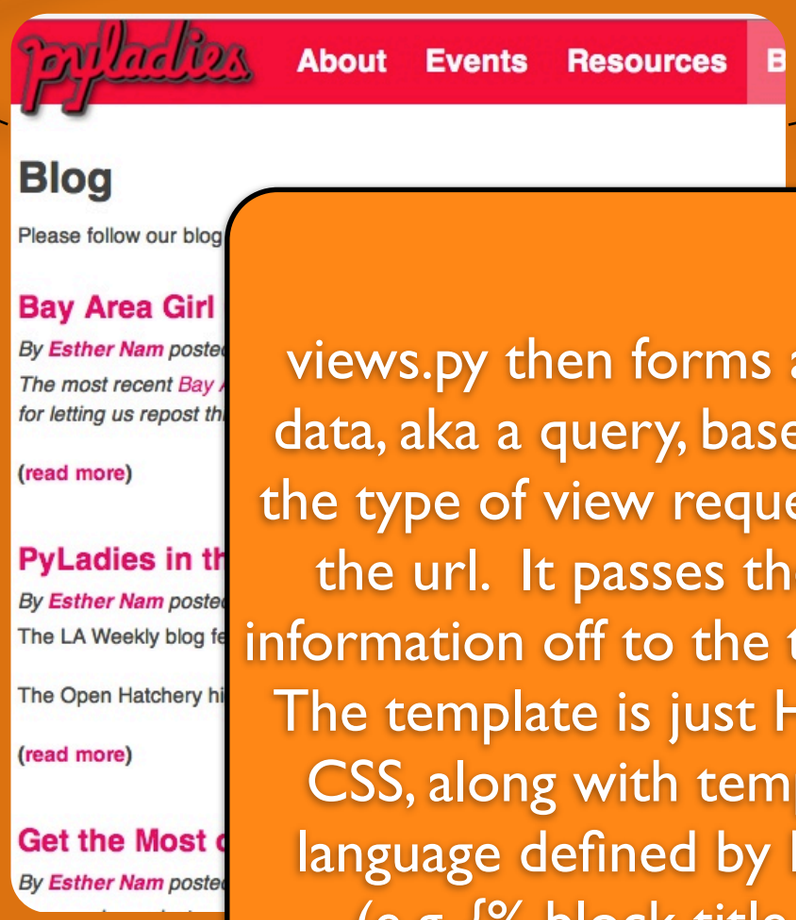
models.py

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 %})

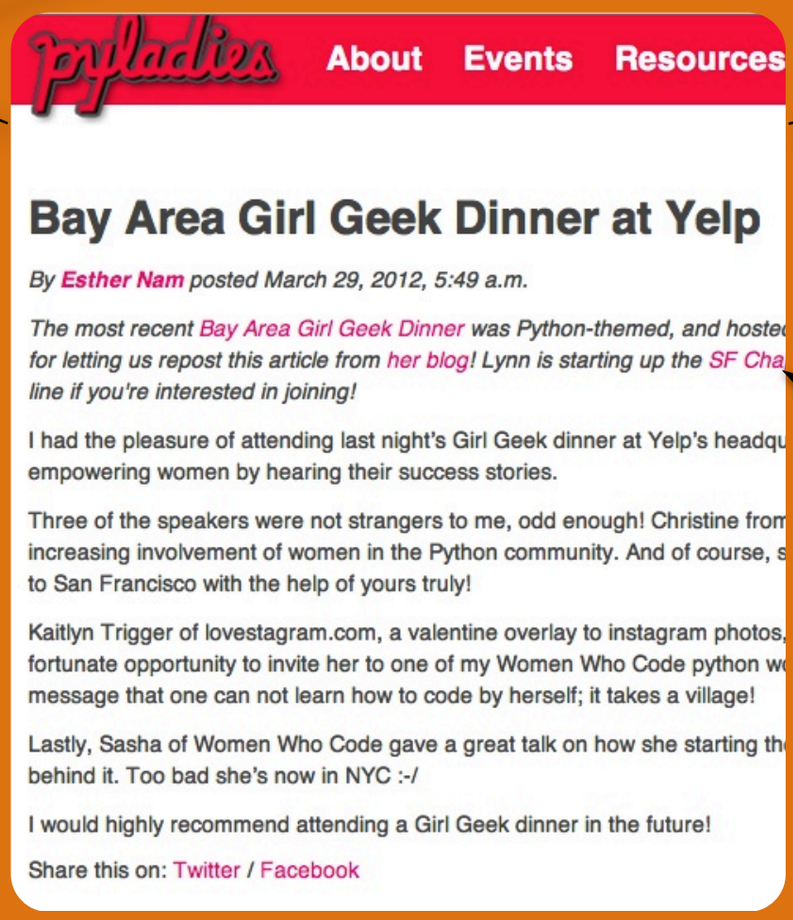
models.py

database

views.py

urls.py

template
.html



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