**Managing**

Starting: > python manage.py startapp

Starting shell > python manage.py shell

Updating db

>python manage.py makemigrations

>python manage.py migrate

**Administration**

*Basics*

Create Superuser to use admin tools

>python manage.py createsuperuser

Import and register models in admin.py to have control over them

Use admin tools from /admin

*Logins*

Pretty much just a login form with username and password

**Settings**

Register each app in settings.py’s INSTALLED\_APPS

**URL Dispatcher**

*Basic Usage*

imports: mainapp / views, django.conf.urls / url, django.contrib / admin

urlpatterns = […]

url( r’^<regex>’, views.<view function>)

]

*Passing Arguments*



*Routing to main\_app*

route from the base directory URLs to the project by inserting

from django.conf.urls import include, url

…

url(r’^’, include(‘main\_app.urls’)):

**Tempaltes**

*Basic Usage*

Mostly HTML with a template language <https://docs.djangoproject.com/en/1.11/ref/templates/builtins/>

Called by render in Views, which usually gives it a context dictionary

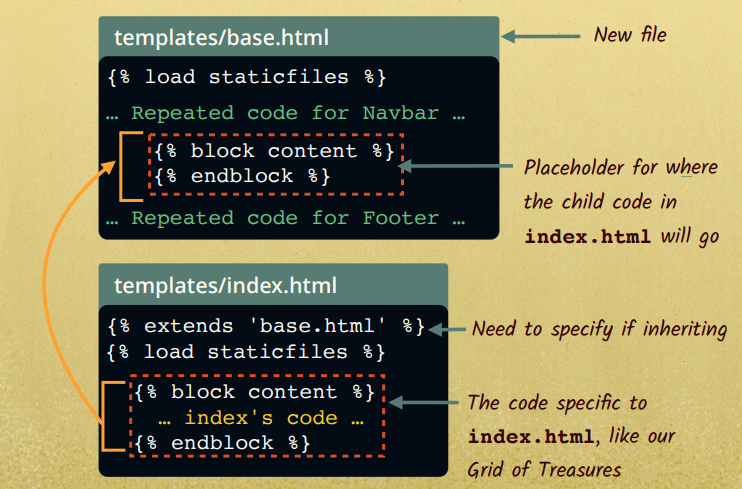
Useful stuff:

cycle: alternates between arguments, inserted anywhere

evalutate dictionary: {{ key }}

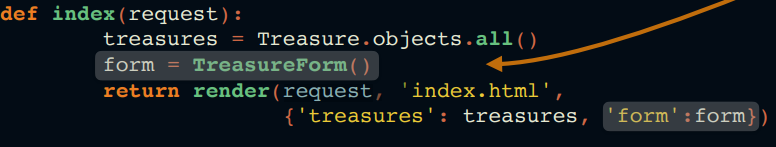
*Inheritance*

Use a base for everything not in the block, then define the block in another file



**Views**

*Basic Example Usage*

needs import of .models / \*

Has template to render, context, call model-defined Treasure, and form (which is is forms.py)

Can also receive parameters from URLs (see URLs)

*Parameters*

Always takes a request as a parameter, which may contain information?

*HTTPResponse*

Using django.http / HttpResponse, send HTTP

> return HttpResponse(…)

*Template Renders*

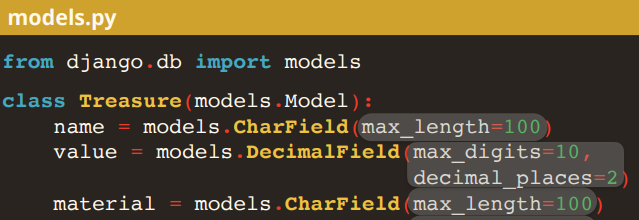
Can return renders of Templates

*Post Requests*

Usually for use in conjunction with AJAX

**Models**

*Simple models*



Fields can be Char-, Integer-, Float-, Decimal-

More defined at <https://docs.djangoproject.com/en/1.11/ref/models/fields/>

Nice to have a toString

> def \_ \_Str\_ \_(self): return self.name

All will have primary key (“id”) with an integer when it was created

ANY CHANGES TO MODEL YOU HAVE TO MIGRATE so it goes thru the Python layer to the SQL

>python manage.py makemigrations

>python manage.py migrate

*Default data*

Just have an object further down that calls the constructors or uses other objects!

**Static Resources**

*Basics*

Put a static dir in your main\_app dir for CSS, Javascript, and Static Images

A subdirectory for images

Load in templates with {% load staticfiles %} at very top

Download libraries like bootstrap and jquery

**Forms**

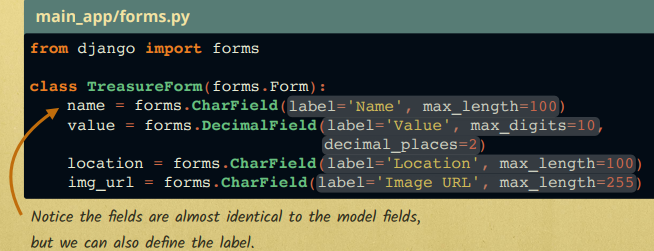
*Basics*

Define a format of data

Give it a url and receiving view so it can be inputted implicitly

Then in the front end, give it to a template and its view render function so the user can use it

-Defining the data: The form sets the parameters for the whole process



-The input end: Receiving and writing the data

Assign it a URL and a receiving function in the view



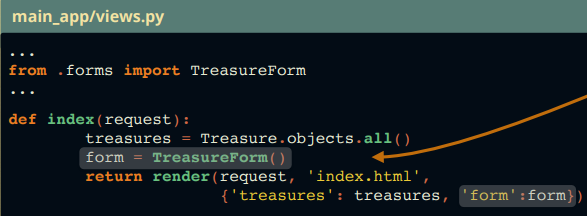


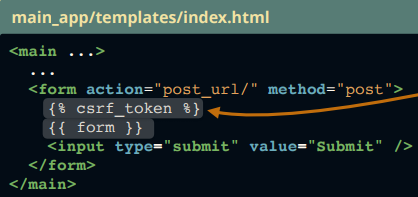
-The front end:

Pass the format to a template via the view that renders that page

Need a csrf\_token for security purposes

Can do {{form.as\_p}} for paragraph format or {{form}} for inline





*Widgets*

For more complex types of data input (date/password/etc) use a widget

forms.py

> forms.

**Tests**

**Javascript**

*AJAX*

Use a views function in conjunction with your AJAX function

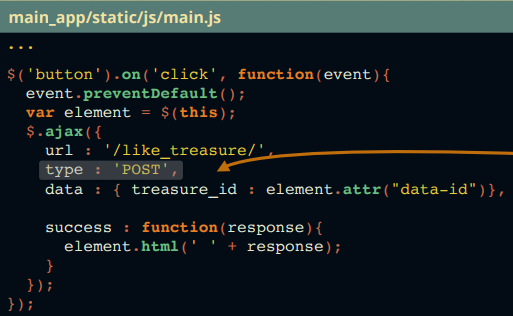
Best practice is to put js in main\_app/static/js, along with jquery-3.0.0.min.js

Link them in templates

<script src="{% static 'js/jquery-3.0.0.min.js' %}"></script>

IMPORTANT: for post, we need to copy/past ajax code getCookie and csrfSafeMethod

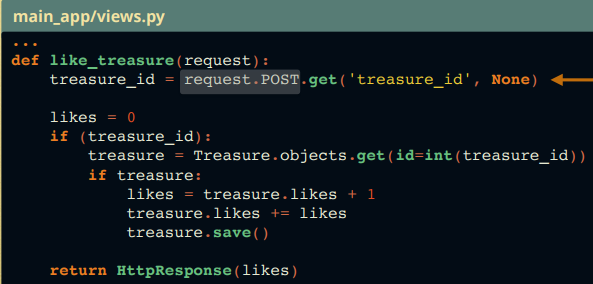
AJAX: add a link to the object when it’s clicked, getting the ID from the local part of the DOM



Register the URL, with this object ID passed via the AJAX function



Views: Receive the data and save

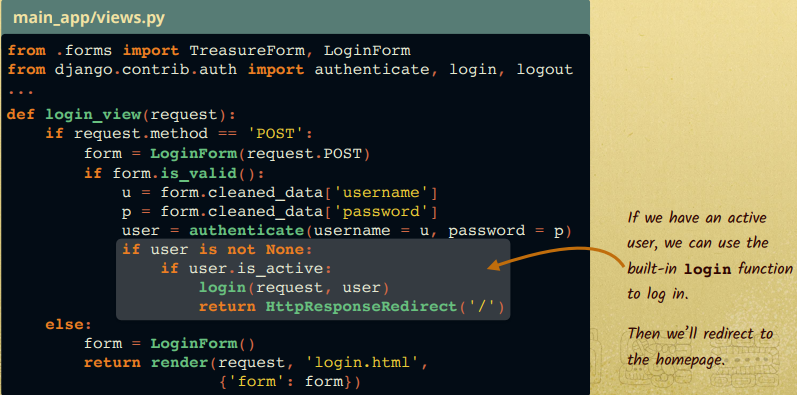


**User Authentication**

*Logging In*

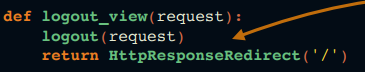
Pretty much just a post form (must define LoginForm)

But we can combine the POST and render view functions with some special code



… and logging out





Template language has a handy “ {% if user.is\_authenticated %}

**Python Language Features**

*Queries of Objects defined in the Model*

Get all instances of object

> <Object Name>.objects.all()

Subset matching field

> <Object Name>.objects.filter(<field> = <value>)

Get by unique value