LEARNING THE BASICS OF DJANGO

Akhil Boddu

Adopted by Caleb Smiths Learning Python and Django on Lynda.com

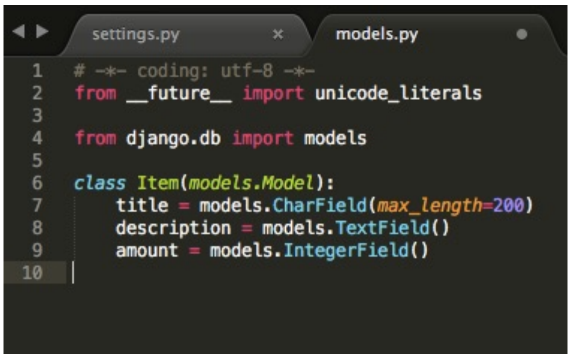
LEARNING DJANGO

Table of Contents

All the steps

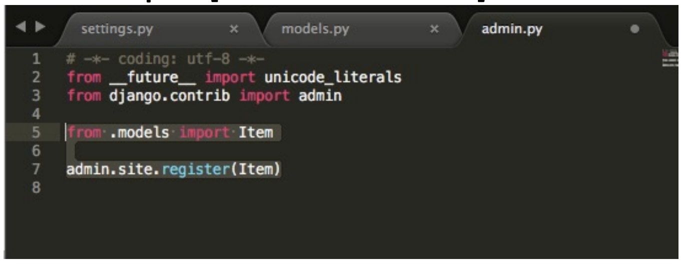
Setting up

1) **django-admin startproject projectName**
2) **cd projectName**
3) **python manage.py runserver**
4) **python manage.py** – *gives a list of the available commands*
5) *Creating an app*: **python manage.py startapp appName**
6) *Edit settings.py in projectName: under installed apps, add:* **'appName,'**
7) *Create model for Inventory app: title, description, amount become the new columns for DB*



```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models

class Item(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    amount = models.IntegerField()

```

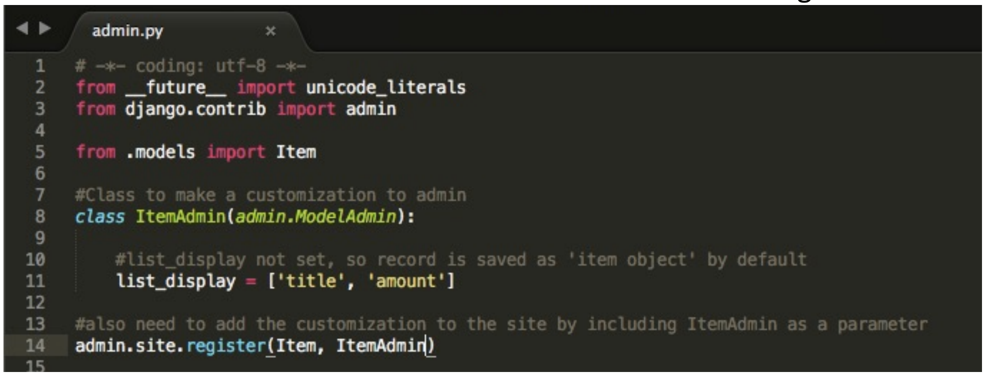
8) *Register the model.py of the inventory app with the admin to control database as admin.*
admin.site.register(Item) completes the registration of model with admin. Also do not forget **from .models import [CreatedClassModel]**



```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.contrib import admin

from .models import Item

admin.site.register(Item)

```

9) *Need Superuser to login to the admin interface to make changes to inventory:* **python manage.py createsuperuser**

10) Customize admin class to make it more flexible in terms of viewing list:



```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.contrib import admin

from .models import Item

#Class to make a customization to admin
class ItemAdmin(admin.ModelAdmin):

    #list_display not set, so record is saved as 'item object' by default
    list_display = ['title', 'amount']

#also need to add the customization to the site by including ItemAdmin as a parameter
admin.site.register(Item, ItemAdmin)
```

11) **python manage.py shell :** *Opens shell in Django* *FOR QUERYING*

- Querying item using **Item.objects.all()**

```
>>> from inventory.models import Item
>>> Item.objects.all()
<QuerySet [<Item: Item object>, <Item: Item object>, <Item:
Item object>, <Item: Item object>, <Item: Item object>]>
>>> items = Item.objects.all()
>>> item = items[0]
>>> item.title
u'MR39'
>>> item = items[3]
>>> item.title
u'polo'
>>> item.id
4
>>> item.amount
10
```

- Querying using **Item.objects.get([attribute]=[somevalue]).title(or amount)**

```
>>> #Querying for items using id
>>> Item.objects.get(id=2).title
u'PGXT'
>>> Item.objects.get(amount=10)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/Library/Python/2.7/site-
packages/4jango/db/models/manager.py", line 85, in
manager_method
    return getattr(self.get_queryset(), name)(*args,
**kwargs)
  File "/Library/Python/2.7/site-
packages/4jango/db/models/query.py", line 384, in get
    (self.model._meta.object_name, num)
MultipleObjectsReturned: get() returned more than one Item –
it returned 2!
```

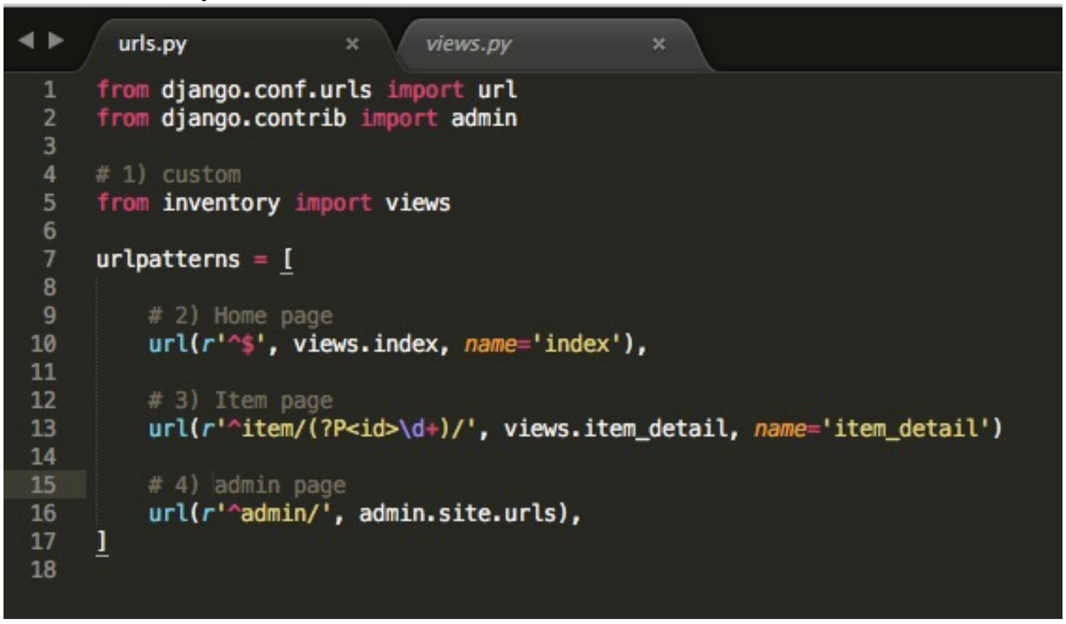
This error is generated because two tuples have the same amount.

- Querying using **Item.objects.filter([attribute]=[value]).[row]title** & using **Item.objects.exclude([attribute]=[value]).title)**

```
>>> Item.objects.filter(amount=0)
<QuerySet [<Item: Item object>]>
>>> Item.objects.filter(amount=0)[0].title
u'PGXT'
>>> Item.objects.exclude(amount=0)[1].title
u'raybans'
>>> Item.objects.exclude(amount=0)[0].title
u'MR39'
>>> Item.objects.exclude(amount=0)[3].title
u'oakley'
>>> Item.objects.filter(amount=0)[0].title
u'PGXT'
```

Configuring the URL patterns

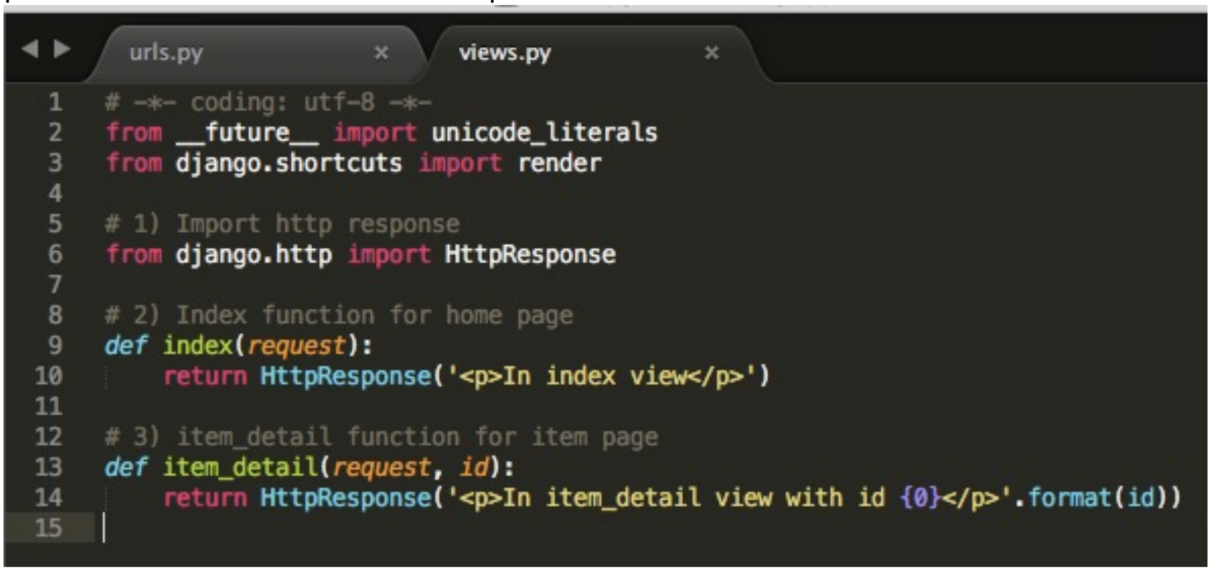
12) Configuring URL patterns in firstProject/urls.py: Note that a named group is used. *Do not forget to put a comma after url as this is a list, below there is no comma and **an error was produced!!***



urls.py × views.py ×

```python
from django.conf.urls import url
from django.contrib import admin

# 1) custom
from inventory import views

urlpatterns = [

    # 2) Home page
    url(r'^$', views.index, name='index'),

    # 3) Item page
    url(r'^item/(?P<id>\d+)/', views.item_detail, name='item_detail')

    # 4) admin page
    url(r'^admin/', admin.site.urls),
]
```

13) Configuring URL patterns in inventory/views.py. All views take in a request parameter. Look at comments for steps.



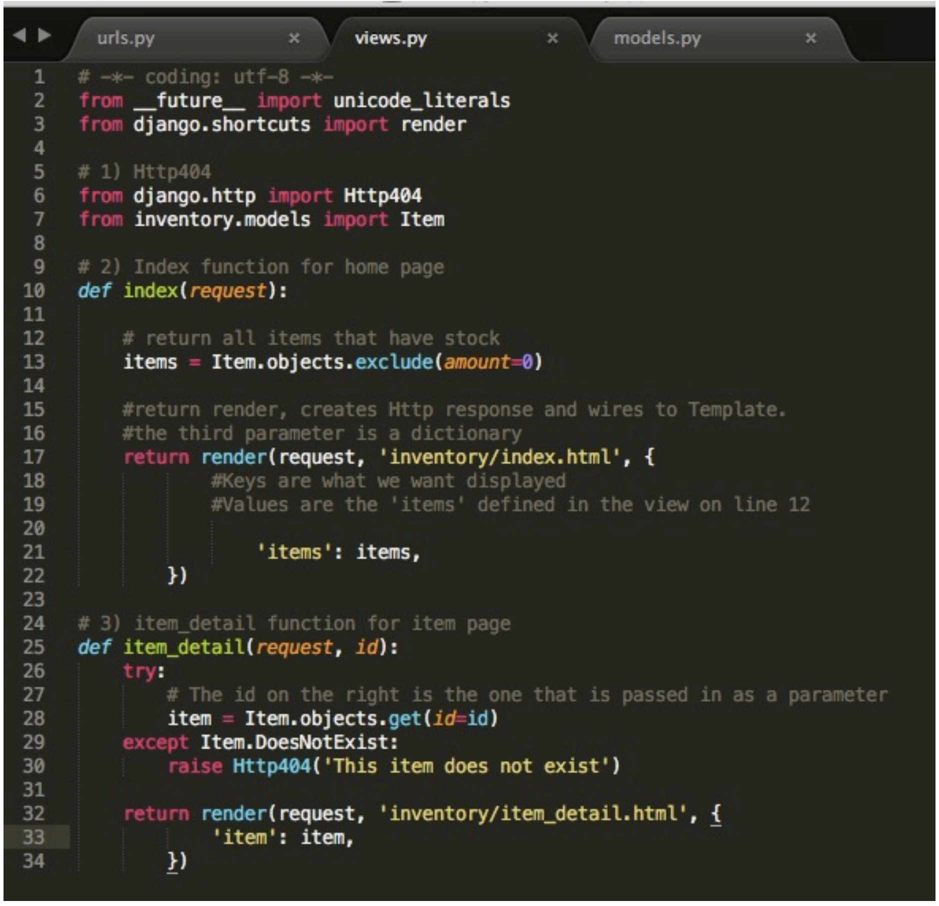
urls.py × views.py ×

```python
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.shortcuts import render

# 1) Import http response
from django.http import HttpResponse

# 2) Index function for home page
def index(request):
    return HttpResponse('<p>In index view</p>')

# 3) item_detail function for item page
def item_detail(request, id):
    return HttpResponse('<p>In item_detail view with id {0}</p>'.format(id))
```

Note that the views.py above is only to check if it works. The views.py is supposed to fulfill the relevant tasks.

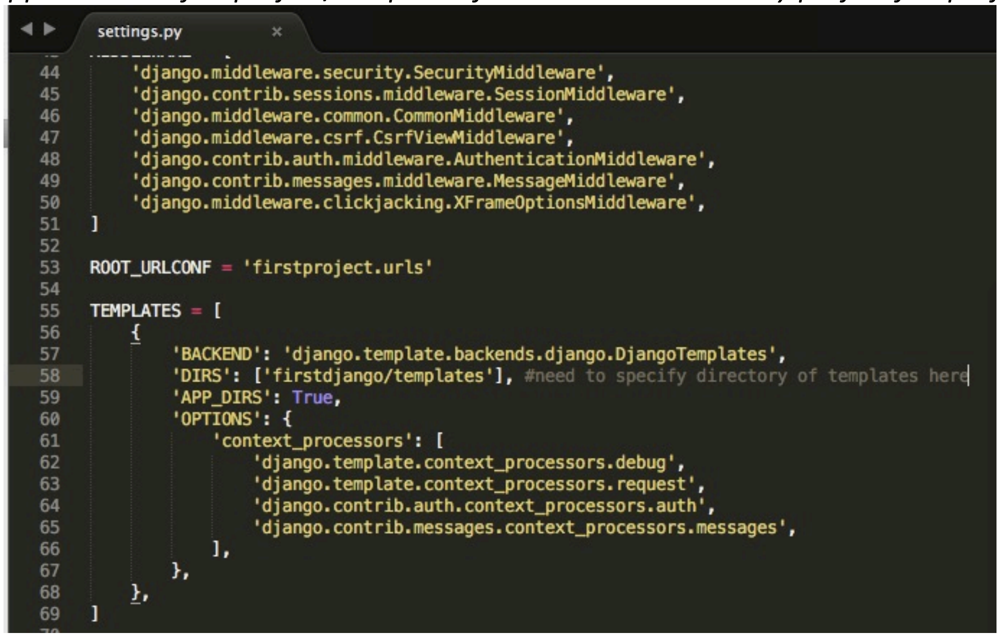
Implementing the Views

14) Implementing views in the inventory/views.py file. Follow steps in the comments.
Index.html and item_detail.html do not exist yet. So that's the following step.



```python
1   # -*- coding: utf-8 -*-
2   from __future__ import unicode_literals
3   from django.shortcuts import render
4
5   # 1) Http404
6   from django.http import Http404
7   from inventory.models import Item
8
9   # 2) Index function for home page
10  def index(request):
11
12      # return all items that have stock
13      items = Item.objects.exclude(amount=0)
14
15      #return render, creates Http response and wires to Template.
16      #the third parameter is a dictionary
17      return render(request, 'inventory/index.html', {
18              #Keys are what we want displayed
19              #Values are the 'items' defined in the view on line 12
20
21                  'items': items,
22          })
23
24  # 3) item_detail function for item page
25  def item_detail(request, id):
26      try:
27          # The id on the right is the one that is passed in as a parameter
28          item = Item.objects.get(id=id)
29      except Item.DoesNotExist:
30          raise Http404('This item does not exist')
31
32      return render(request, 'inventory/item_detail.html', {
33              'item': item,
34          })
```

15) Go to firstProject/settings.py – specify directory (line 58)
(*supposed to be firstproject/templates for me since I named my project firstproject*)

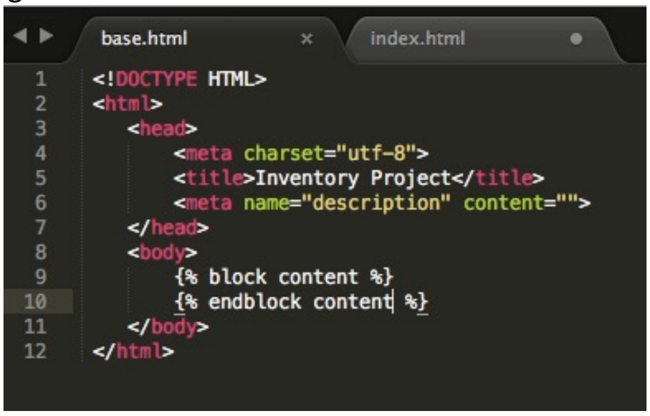


```python
44      'django.middleware.security.SecurityMiddleware',
45      'django.contrib.sessions.middleware.SessionMiddleware',
46      'django.middleware.common.CommonMiddleware',
47      'django.middleware.csrf.CsrfViewMiddleware',
48      'django.contrib.auth.middleware.AuthenticationMiddleware',
49      'django.contrib.messages.middleware.MessageMiddleware',
50      'django.middleware.clickjacking.XFrameOptionsMiddleware',
51  ]
52
53  ROOT_URLCONF = 'firstproject.urls'
54
55  TEMPLATES = [
56      {
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',
58          'DIRS': ['firstdjango/templates'], #need to specify directory of templates here
59          'APP_DIRS': True,
60          'OPTIONS': {
61              'context_processors': [
62                  'django.template.context_processors.debug',
63                  'django.template.context_processors.request',
64                  'django.contrib.auth.context_processors.auth',
65                  'django.contrib.messages.context_processors.messages',
66              ],
67          },
68      },
69  ]
```

16) Create templates directory in firstproject that contains settings.py etc.
17) Create inventory directory within firstproject.
18) Create html files (index.html and item_detail.html) within the inventory directory.

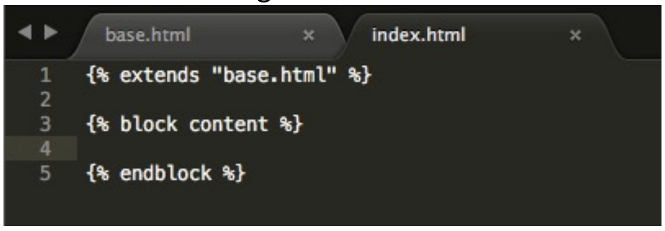
Implementing the Templates

19) Create base.html file inside the templates folder but not inside the templates/inventory folder.
20) Make the following additions to relevant base.html file:



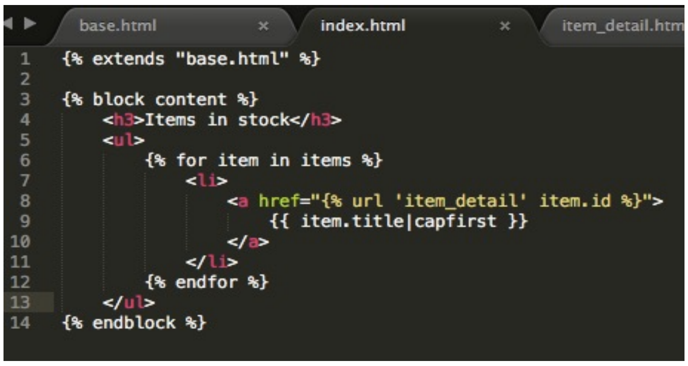
```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8">
        <title>Inventory Project</title>
        <meta name="description" content="">
    </head>
    <body>
        {% block content %}
        {% endblock content %}
    </body>
</html>
```

21) Open index.html and do the following first:



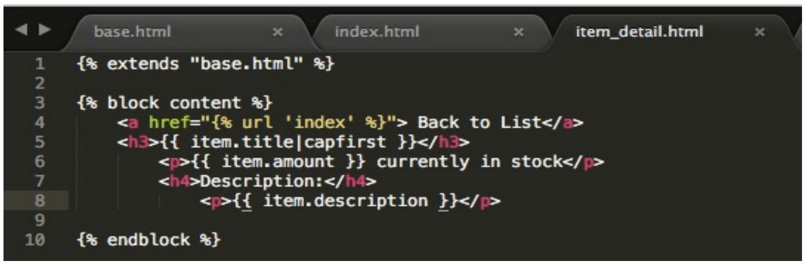
```
{% extends "base.html" %}

{% block content %}

{% endblock %}
```

22) Then go on and add relevant html.



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

{% block content %}
    <h3>Items in stock</h3>
    <ul>
        {% for item in items %}
            <li>
                <a href="{% url 'item_detail' item.id %}">
                    {{ item.title|capfirst }}
                </a>
            </li>
        {% endfor %}
    </ul>
{% endblock %}
```

23) Then do the same with the other html files.

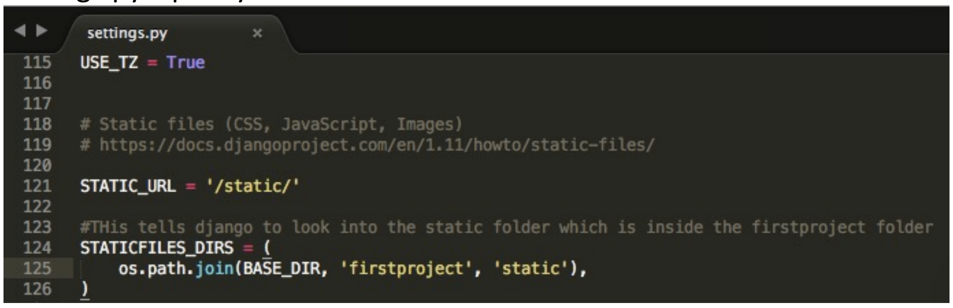


```
{% extends "base.html" %}

{% block content %}
    <a href="{% url 'index' %}"> Back to List</a>
    <h3>{{ item.title|capfirst }}</h3>
        <p>{{ item.amount }} currently in stock</p>
        <h4>Description:</h4>
            <p>{{ item.description }}</p>

{% endblock %}
```

24) Then runserver

Integrating CSS and JavaScript (Static assets)

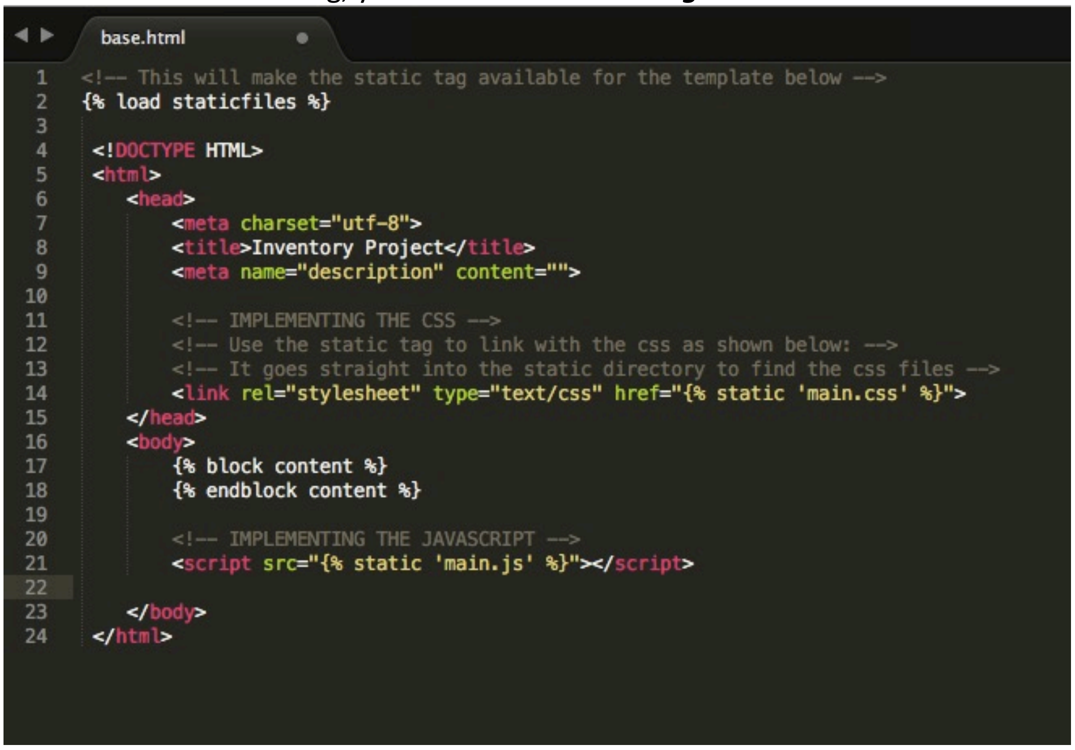
25) Go to settings.py: specify where to look for static files.



settings.py

```python
USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.11/howto/static-files/

STATIC_URL = '/static/'

#THis tells django to look into the static folder which is inside the firstproject folder
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'firstproject', 'static'),
)
```

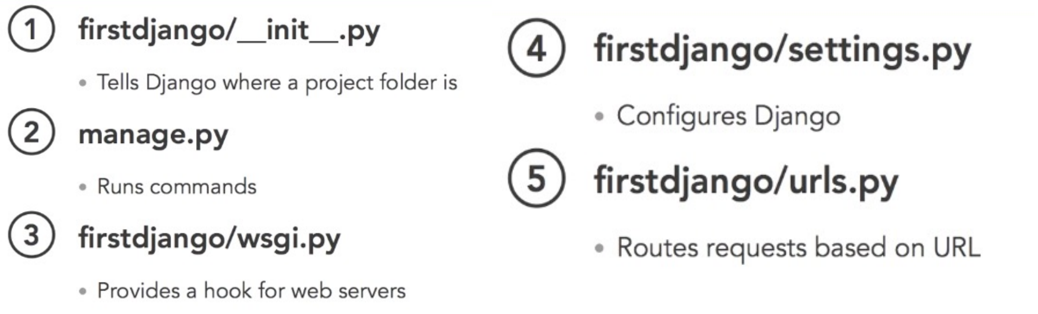
26) Create static directory inside firsproject with relevant CSS and JavaScript files.
27) Need to change base.html file
   a. To include CSS and JavaScript, using the ***static tag.***
   b. To use the static tag, you must use the ***load tag*** to load the static files.



base.html

```html
<!-- This will make the static tag available for the template below -->
{% load staticfiles %}

 <!DOCTYPE HTML>
 <html>
     <head>
         <meta charset="utf-8">
         <title>Inventory Project</title>
         <meta name="description" content="">

         <!-- IMPLEMENTING THE CSS -->
         <!-- Use the static tag to link with the css as shown below: -->
         <!-- It goes straight into the static directory to find the css files -->
         <link rel="stylesheet" type="text/css" href="{% static 'main.css' %}">
     </head>
     <body>
         {% block content %}
         {% endblock content %}

         <!-- IMPLEMENTING THE JAVASCRIPT -->
         <script src="{% static 'main.js' %}"></script>

     </body>
 </html>
```

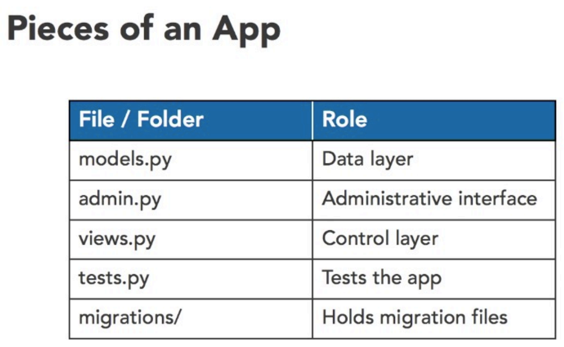
Roles of each of the files when projectName is created



1. **firstdjango/__init__.py**
   - Tells Django where a project folder is
2. **manage.py**
   - Runs commands
3. **firstdjango/wsgi.py**
   - Provides a hook for web servers
4. **firstdjango/settings.py**
   - Configures Django
5. **firstdjango/urls.py**
   - Routes requests based on URL

1,2,3 are not changed manually.

Each Django project can have one or more apps



Pieces of an App

File / Folder	Role
models.py	Data layer
admin.py	Administrative interface
views.py	Control layer
tests.py	Tests the app
migrations/	Holds migration files

Models

- These create the database structure.
- Allows querrying db.
- Inherited class from “models.Model”
- Like a spreadsheet

Migrations

- Generate scripts each time model is created (initial migration), field is added, field is removed, or when attributes of a field have changed.

Commands:

- python manage.py makemigrations – *generates migration files by comparing current fields with existing fields.*
- python manage.py migrate – *runs all migrations that haven’t been run*
- migrate --list

Field Types: main examples

Field Types—Numeric Data

Field	Example Values
IntegerField	-1, 0, 1, 20
DecimalField	0.5, 3.14

Field Types—Textual Data

Field	Example Values
CharField	"Product Name"
TextField	"To elaborate on my point…"
EmailField	george@email.com
URLField	www.example.com

Field Types—File Data

Field	Example Values
FileField	user_uploaded.docx
ImageField	best_avatar.jpg

Field Types—Miscellaneous Data

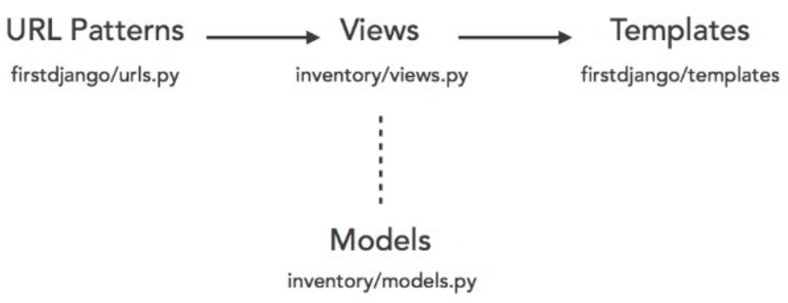
Field	Example Values
BooleanField	True, False
DateTimeField	datetime(1960, 1, 1, 8, 0, 0)

Field Attribute Options

- max_length
- null
- blank
- default
- choices

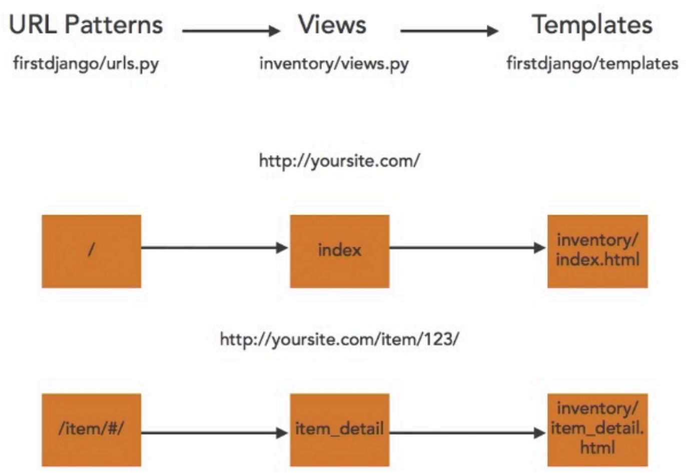
High level view of the “MVC”

Model view controller:



- URL patterns
  - *pass control to views*
  - *projectName/**urls.py***
- Views
  - *logic layer*
  - *python functions take requests and return HTTPS response*
  - *inventory/**views.py***
  - *Each view can use inventory/**models.py** to query DB.*
  - *Relies on Templates for presentation layer (HTML)*
- Models
  - *inventory/**models.py***
  - *DB queries*
- Templates
  - Contain HTML files with extra template syntax.
  - *projectName/templates*

Web domain determines what is displayed

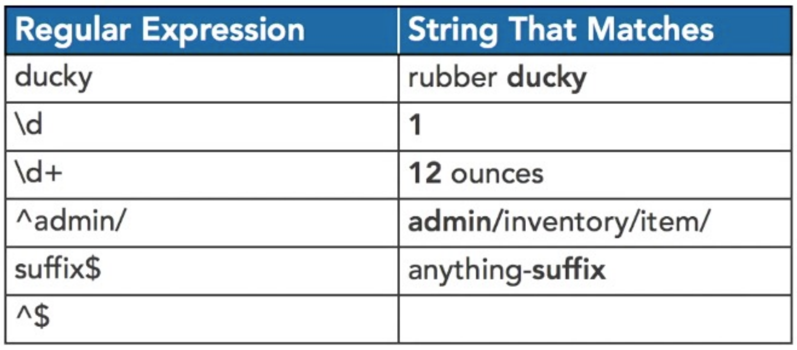


- index is a function that uses index.html to display page
- item_detail is a function that queries DB, finds relevant item using # (item number), then displays the item_detail.html

URL Patterns

Regular Expressions:

- ❖ Interpret URLs for the site.
- ❖ Determines search patterns for strings as shown below:

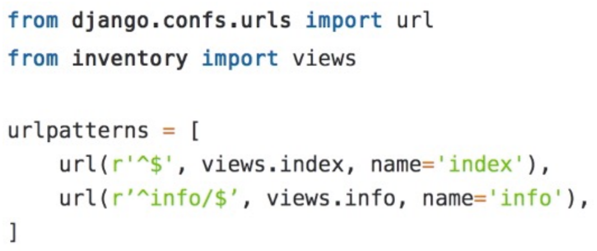


Regular Expression	String That Matches
ducky	rubber **ducky**
\d	**1**
\d+	**12** ounces
^admin/	**admin/**inventory/item/
suffix$	anything-**suffix**
^$	

- \d means exactly one digit
- \d+ means one or more digits
- ^admin/ matches any string that begins with admin/
- suffix$ matches to any string that ends with suffix
- ^$ matches to empty string. Starts and ends with nothing in between. View to domain with nothing after it.

Click here to play around with Regular expressions.

Example:



```
from django.confs.urls import url
from inventory import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^info/$', views.info, name='info'),
]
```

If patterns are not matched Error404 is returned.

Named group:

`r'^item/(?P<id>\d+)/`

Specifies that the digit that is passed is an ID showing that id is a Parameter that will be used in the function in views.py

Views

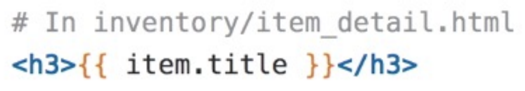
- This is where the main logic is stored.
- Look through steps from 12-16 to understand how views work (by connecting to templates and by determining when function is called from firstproject/urls.py)
- View calls render which passes data into the template to be viewed.

Templates

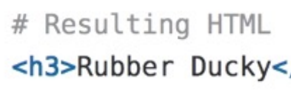
Syntax for Django Templates

{{ variable }}

This shows the variable if used with curly braces as shown.



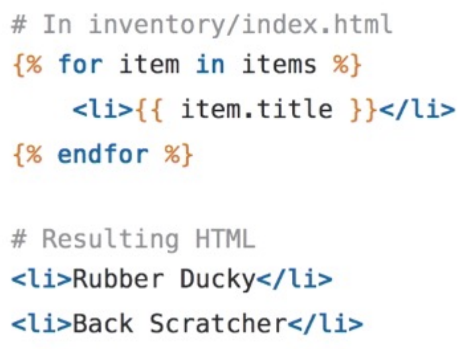
```
# In inventory/item_detail.html
<h3>{{ item.title }}</h3>
```



```
# Resulting HTML
<h3>Rubber Ducky</h3>
```

{% tag %}

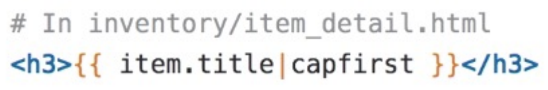
Used for loops, if/else, and other structural elements



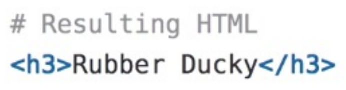
```
# In inventory/index.html
{% for item in items %}
    <li>{{ item.title }}</li>
{% endfor %}

# Resulting HTML
<li>Rubber Ducky</li>
<li>Back Scratcher</li>
```

{{ variable|filter }}

- Take string as input and return string as output
- Filters the variable by piping ("|")
- Controls the way output is formatted
- R and D is capitalized using the capfirst filter



```
# In inventory/item_detail.html
<h3>{{ item.title|capfirst }}</h3>
```



```
# Resulting HTML
<h3>Rubber Ducky</h3>
```

Can be used in html file to return the url path as follows (No corresponding end tag, and render a string instead):

URL tag {% url ‘index’ %}

- This tag takes in the name of url as a parameter

```
# In firstdjango/urls.py
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^item/(?P<id>\d+)/$', views.item_detail,
        name='item_detail'),
]
{% url 'index' %}

# Result
/
```

Observe the {% url ‘index’ %}

- The tag needs other parameters if a named group is used:

```
# In firstdjango/urls.py
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^item/(?P<id>\d+)/$', views.item_detail,
        name='item_detail'),
]
{% url 'item_detail' item.id %}

# Result
/item/1/
```

Observe the {% url ‘item_detail’ item.id %}

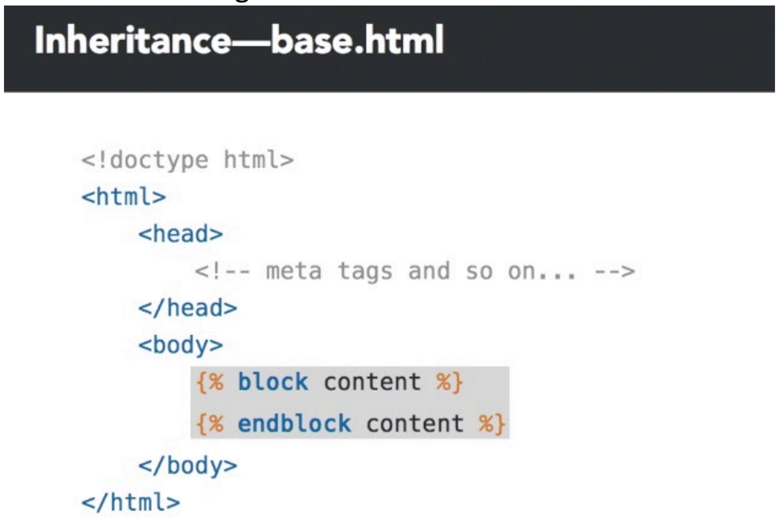
Example of a template in html file:

```
# In inventory/index.html
<ul>
  {% for item in items %}
    <li>
      <a href="{% url 'item_detail' item.id %}">
        {{ item.title|capfirst }}
      </a>
    </li>
  {% endfor %}
</ul>
```

```
# Resulting HTML
<ul>
    <li><a href="/item/1">Rubber Ducky</a></li>
    <li><a href="/item/1">Back Scratcher</a></li>
</ul>
```

Template Inheritance

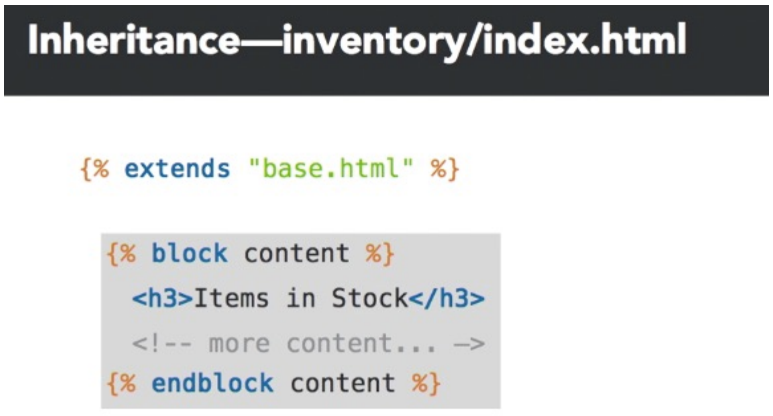
- Using the extends and block tags



Inheritance—base.html

```
<!doctype html>
<html>
    <head>
        <!-- meta tags and so on... -->
    </head>
    <body>
        {% block content %}
        {% endblock content %}
    </body>
</html>
```

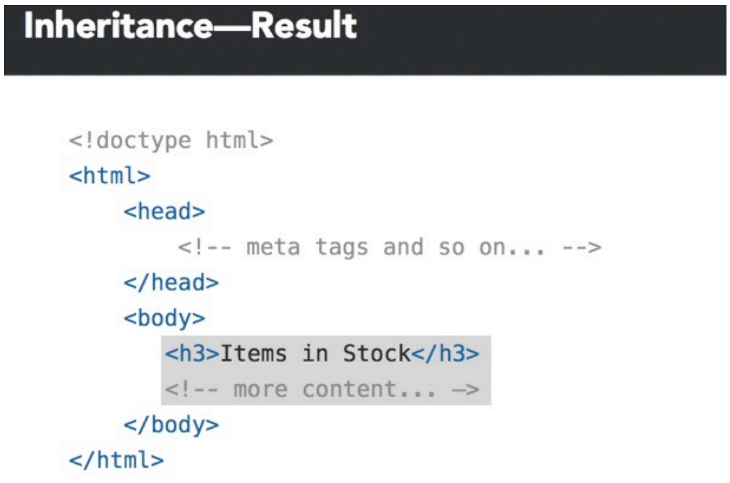
In the base.html file



Inheritance—inventory/index.html

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

  {% block content %}
    <h3>Items in Stock</h3>
    <!-- more content... ->
  {% endblock content %}
```

the extends tag has to be the first line of the template



Inheritance—Result

```
<!doctype html>
<html>
    <head>
        <!-- meta tags and so on... -->
    </head>
    <body>
        <h3>Items in Stock</h3>
        <!-- more content... ->
    </body>
</html>
```