**Django Cheat Sheet**

**🐍 Initializing on Visual Studio Code**

* Make main folder with $ mkdir <main\_folder>, enter and open with Visual Studio Code
* Open Bash Terminal inside VS-Code and type $ python -m venv ./venv
* Enter virtual environment with $ source venv/scripts/activate
* Install django with  $ pip install django
* Install psycopy2 with $ pip install psycopg2
* Install psycopy2 with $ pip install psycopg2-binary
* Install pillow with $ pip install pillow
* Click [F1] and type ‘Python: Select Interpretator’ and choose ‘.\venv\scripts\python.exe’
* Install ‘Linter Pylint’ within the virtual environment

On Ubuntu

* sudo apt-get upgrade
* sudo apt-get install
* sudo apt-get install python3-pip
* apt-get install python3-venv OR
* sudo pip3 install virtualenv

Create a Model:

Writing Data Into Model

Displaying Data Into Website:

**📘 Creating a project**

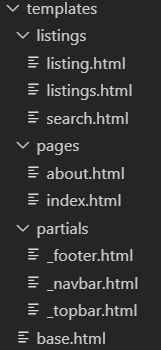
* Create project with $ django-admin startproject <project\_name> **.**

**📃 Creating apps – pages and listings**

* Create apps with $ python manage.py startapp <app\_name>
* Inside each app folder, create a file called urls.py.
* To link app to project, create a URL-path at main’s urls.py: path('', include('pages.urls')),
* To link app to project, create a URL-path at main’s urls.py: path('', include('listings.urls')),

**🎨 Creating a HTML directory named ‘templates’**

* Create a ‘templates’ directory under the main folder and create directories ‘pages’ and ‘listings’ directories in it. The 2 directories will house HTML files from ‘pages’ and ‘listings’ end-points.
* Create a ‘partials’ directory inside templates which houses \_navbar, \_footer and \_topbar html
* Create a base.html inside the templates directory which references bootstrap files
* See samples base.html and \_navbar.html at Appendix



**📃 Make Changes in Settings.py**

1. **Create Secret Key**

If you want your SECRET\_KEY to be more secure, you can set it to reference an environment variable.

SECRET\_KEY = os.environ.get('SECRET\_KEY')

* To quickly generate a random hex for your secret key:

>>> import secrets

>>> secrets.token\_hex()

* You can set this environment variable in your shell with export SECRET\_KEY=<secret\_key>

1. **Create Templates Directory Path**

'DIRS': [os.path.join(BASE\_DIR,'templates')],

1. **Create Static File Directory Path**

STATIC\_ROOT= os.path.join(BASE\_DIR, 'static')

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, '<project\_name>/static')]

Create a ‘static’ directory within the <startproject> directory.

When run ‘collectstatic’ command, a static directory will be created at <root> which takes files from <startproject>/static

1. **Create Media File Directory Path**

# Media Folder Settings

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

Add the following code inside <main\_project> urls.py

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('', include('pages.urls')),

path('listings/', include('listings.urls')),

path('accounts/', include('accounts.urls')),

path('contacts/', include('contacts.urls')),

path('admin/', admin.site.urls),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

1. **Include App Into Project**

* To include this app in your project, add your app to the project's settings.py file by adding its name to the INSTALLED\_APPS list:

INSTALLED\_APPS = [

'pages.apps.PagesConfig',

'listings.apps.ListingsConfig',

# ...

]

1. **Make Change to Database**

A create a new database:

c:\'program files'\postgresql\11\scripts\runpsq

postgres=# CREATE DATABASE btre\_prod;

postgres=# CREATE USER dbadmin WITH PASSWORD 'abc123!';

postgres=# ALTER ROLE dbadmin SET client\_encoding TO 'utf8';

postgres=# ALTER ROLE dbadmin SET default\_transaction\_isolation TO 'read committed';

postgres=# ALTER ROLE dbadmin SET timezone TO 'UTC';

postgres=# GRANT ALL PRIVILEGES ON DATABASE btre\_prod TO dbadmin;

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': '<database name ie. btre\_prod>',

'USER': '<username ie. dbadmin>',

'PASSWORD': '<password ie. abc123!>',

'HOST': 'localhost'

}

}

**📺 Creating a View.PY**

Views.Py is the file which takes the data from the database (Models.Py) and then renders it into the HTML file.

[database]->[models.py]->[views.py]->html

The views file imports the Class in the model and creates a dictionary and passes the dictionary to the HTML

from django.shortcuts import render

from listings.models import Listing

def index(request):

    listings = Listing.order\_by('-list\_date').filter(is\_published=True)[:3]

    context = {

        'listings' : listings

    }

    return render(request, 'pages/index.html', context)

Data can also be passed as a dictionary from view.py to the html via the following:

from django.http import HttpResponse

from django.shortcuts import render

from .models import Listing

def index(request):

return render(request, "Listings/listing.html", {‘name‘: ‘Brad’ } )

At Listings/listing.html

<h1 class=”display-4”> Browse {{ name }} </h1> would display ‘Browse Brad’

listings = Listing.objects.all()

context = {

    'listings01': listings

}

def index(request):

    return render(request, 'listings/listings.html', context)

{%if listings01%}

{% for listing02 in listings01%}

<!-- Display Each Listing -->

<h4 class="text-primary">{{ listing02.title }}</h4>

<img class="card-img-top" src="{{ listing02.photo\_main.url }}" alt="">

<i class="fas fa-user"></i> {{ listing02.realtor }} </div>

<i class="fas fa-clock"></i> {{ listing02.list\_date | timesince }}</div>

<a href="{% url 'listing' listing02.id %}" class="btn btn-primary btn-block">More Info</a>

{% endfor%}

{% else %}

<div class="col-md-12"><p>No listings available..</p></div>

{% endif%}

**🐍 Initializing Urls.PY file to Link with Views.PY**

Edit urls.py to include the following

from django.contrib import admin

from django.urls import include, path

from . import views

urlpatterns = [

path('app/', include('app.urls')),

path('admin/', admin.site.urls),

path('', views.index, name='index'),

]

* To create a url pattern to the index of the site, use the following urlpattern:

urlpatterns = [

path("", include('app.urls')),

]

* Remember: there are multiple files named urls.py
* The urls.py file within app directories are organized by the urls.py found in the project folder.

**🎨 Creating a HTML directory named ‘templates’**

* Create a ‘templates’ directory under the main, CSS, and JavaScript directories are located within the following locations:

<main\_project>/

templates/

pages/

static/

css\_dir/

js\_dir/

* To add a template to views, open views.py within the app directory and include the following:

from django.shortcuts import render

def index(request):

return render(request,'index.html')

* To include context to the template:

def index(request):

context = {"context\_variable": context\_variable}

return render(request,'index.html', context)

* Within the HTML file, you can reference static files by adding the following:

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="{% static 'styles.css' %}">

</head>

</html>

* To make sure to include the following in your settings,py:

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, "static")

]

* To add an extends:

{% extends 'base.html' %}

{% block content %}

Hello, World!

{% endblock %}

* And then in base.html add:

<body>

{% block content %}{% endblock %}

</body>

**🎫 Creating a model**

* Within the app's models.py file, an example of a simple model can be added with the following:

from django.db import models

class Person(models.Model):

first\_name = models.CharField(max\_length=30)

last\_name = models.CharField(max\_length=30)

*Note that you don't need to create a primary key, Django automatically adds an IntegerField.*

* To inact changes in your models, use the following commands in your shell:

$ python manage.py makemigrations <app\_name>

$ python manage.py migrate

*Note: including <app\_name> is optional.*

* A one-to-many relationship can be made with a ForeignKey:

class Musician(models.Model):

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

instrument = models.CharField(max\_length=100)

class Album(models.Model):

artist = models.ForeignKey(Musician, on\_delete=models.CASCADE)

name = models.CharField(max\_length=100)

release\_date = models.DateField()

num\_stars = models.IntegerField()

* In this example, to query for the set of albums of a musician:

>>> m = Musician.objects.get(pk=1)

>>> a = m.album\_set.get()

* A many-to-many relationship can be made with a ManyToManyField:

class Topping(models.Model):

# ...

pass

class Pizza(models.Model):

# ...

toppings = models.ManyToManyField(Topping)

*Note that the ManyToManyField is****only defined in one model****. It doesn't matter which model has the field, but if in doubt, it should be in the model that will be interacted with in a form.*

* Although Django provides a OneToOneField relation, a one-to-one relationship can also be defined by adding the kwarg of unique = True to a model's ForeignKey:

ForeignKey(SomeModel, unique=True)

* For more detail, the [official documentation for database models](https://docs.djangoproject.com/en/2.0/topics/db/models/) provides a lot of useful information and examples.

**📮 Creating model objects and queries**

* Example models.py file:

from django.db import models

class Blog(models.Model):

name = models.CharField(max\_length=100)

tagline = models.TextField()

def \_\_str\_\_(self):

return self.name

class Author(models.Model):

name = models.CharField(max\_length=200)

email = models.EmailField()

def \_\_str\_\_(self):

return self.name

class Entry(models.Model):

blog = models.ForeignKey(Blog, on\_delete=models.CASCADE)

headline = models.CharField(max\_length=255)

body\_text = models.TextField()

pub\_date = models.DateField()

mod\_date = models.DateField()

authors = models.ManyToManyField(Author)

n\_comments = models.IntegerField()

n\_pingbacks = models.IntegerField()

rating = models.IntegerField()

def \_\_str\_\_(self):

return self.headline

* To create an object within the shell:

$ python manage.py shell

>>> from blog.models import Blog

>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')

>>> b.save()

* To save a change in an object:

>>> b.name = 'The Best Beatles Blog'

>>> b.save()

* To retrieve objects:

>>> all\_entries = Entry.objects.all()

>>> indexed\_entry = Entry.objects.get(pk=1)

>>> find\_entry = Entry.objects.filter(name='Beatles Blog')

**👨 Using the Admin page**

* To create a superuser:

$ python manage.py createsuperuser

* To add a model to the Admin page include the following in admin.py:

from django.contrib import admin

from .models import Listing

admin.site.register(Listing)

To displar more fields in the admin panel, add a new class and have the class as part of the parameters in admin.site.register:

class ListingAdmin(admin.ModelAdmin):

list\_display = ('id', 'title', 'is\_published', 'price', 'list\_date', 'realtor')

list\_display\_links = ('id', 'title')

list\_filter = ('realtor',)

list\_editable = ('is\_published',)

search\_fields = ('title', 'description', 'address', 'city', 'state', 'zipcode', 'price')

list\_per\_page = 25

admin.site.register(Listing, ListingAdmin)

**Appendix**

Link: <https://github.com/lucrae/django-cheat-sheet#man-using-the-admin-page>

**\_navbar.html**

|  |
| --- |
| {% load static %}  <!-- Navbar -->  <nav class="navbar navbar-expand-lg navbar-dark bg-primary sticky-top">  <div class="container">  <a class="navbar-brand" href="index.html">  <img src="{% static 'img/logo.png' %}" class="logo" alt="">  </a>  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup">  <span class="navbar-toggler-icon"></span>  </button>  <div class="collapse navbar-collapse" id="navbarNavAltMarkup">  <ul class="navbar-nav">  <li  {% if '/' == request.path %}  class="nav-item active mr-3"  {% else %}  class="nav-item mr-3"  {% endif %}  >  <a class="nav-link" href="{% url 'index' %}">Home</a>  </li>  <li  {% if 'about' in request.path %}  class="nav-item active mr-3"  {% else %}  class="nav-item mr-3"  {% endif %}  >  <a class="nav-link" href="{% url 'about' %}">About</a>  </li>  <li  {% if 'listings' in request.path %}  class="nav-item active mr-3"  {% else %}  class="nav-item mr-3"  {% endif %}  >  <a class="nav-link" href="{% url 'listings' %}">Listings</a>  </li>  </ul>  <ul class="navbar-nav ml-auto">  <li class="nav-item mr-3">  <a class="nav-link" href="register.html">  <i class="fas fa-user-plus"></i> Register</a>  </li>  <li class="nav-item mr-3">  <a class="nav-link" href="login.html">  <i class="fas fa-sign-in-alt"></i>  Login</a>  </li>  </ul>  </div>  </div>  </nav> |

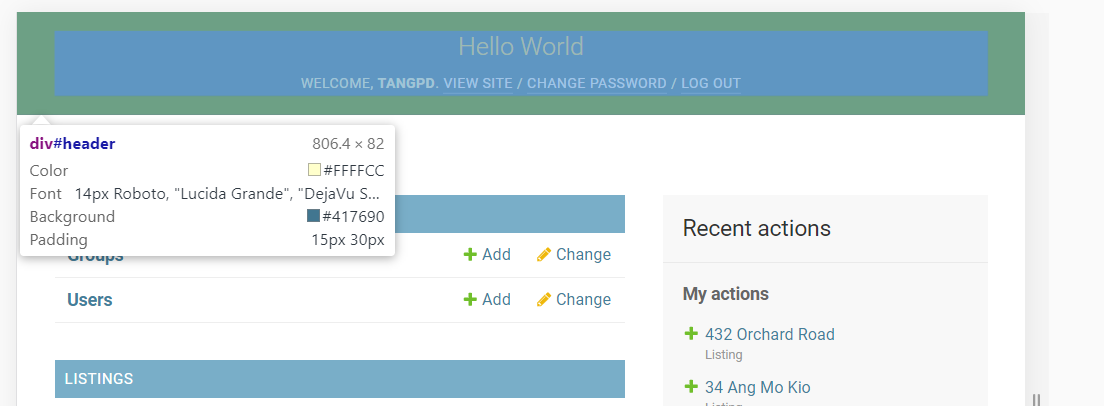
**base.html**

|  |
| --- |
| {% load static %}  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <meta http-equiv="X-UA-Compatible" content="ie=edge">  **<!-- All -->**  **<link rel="stylesheet" href="{% static 'css/all.css' %}">**  **<!-- Bootstrap -->**  **<link rel="stylesheet" href="{% static 'css/bootstrap.css' %}">**  **<!-- Lightbox -->**  **<link rel="stylesheet" href="{% static 'css/lightbox.min.css' %}">**  **<!-- Style -->**  **<link rel="stylesheet" href="{% static 'css/style.css' %}">**  <title>BTRE-TITLE</title>  </head>  <body>  {% include 'partials/\_topbar.html'%}  {% include 'partials/\_navbar.html'%}  {%block content %}{% endblock %}  {% include 'partials/\_footer.html'%}    **<script src="{% static 'js/bootstrap.bundle.min.js' %}"></script>**  **<script src="{% static 'js/jquery-3.3.1.min.js' %}"></script>**  **<script src="{% static 'js/lightbox.min.js' %}"></script>**  **<script src="{% static 'js/main.js' %}"></script>**  </body>  </html> |

**To change current CSS value settings in Bootstrap**

1. Create a new css file ie. admin.css and place it inside <main\_project>/static/css directory

2. Find the id of the CSS element to be changed from Google Chrome:



Admin.css:

#header {

height: 50px;

background: #10284e;

color: #fff;

}

#branding h1 {

color: #fff;

}

Inside the html file, type the following:

{% load static %}

{% block extrastyle %}

<link rel=”stylesheet” href=”{% static ‘css/admin.css’ %}”>

{% endblock %}

**🐍 Initializing pipenv (optional)**

* Make main folder with $ mkdir <folder> and navigate to it with $ cd <folder>
* Initialize pipenv with $ pipenv install
* Enter pipenv shell with $ pipenv shell
* Install django with $ pipenv install django
* Install other package dependencies with $ pipenv install <package\_name>

Cleaning Up Databases:

Get-ChildItem \* -Include 0001\_initial.py -Recurse | Remove-Item

Get-ChildItem \* -Include 0001\_initial.py -Recurse | Remove-Item

del /s 0001\_initial.py

**🐍 Linter Setup**

Install the following:$pip install pylint-django

Add the following to .vscode -> user settings:

"python.linting.pylintArgs": ["--load-plugins=pylint\_django", “—errors-only” ],

* Within the app directory, open views.py and add the following to bring the models data into views: