:scroll: Django Cheat Sheet

A cheat-sheet for creating web apps with the Django framework using the Python language. Most of the summaries and examples are based off the official documentation for Django v2.0.

Sections

:snake: 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>`

:blue_book: Creating a project

- Navigate to main folder with `$ cd <folder>`
- Create project with `$ django-admin startproject <project_name>`

The project directory should look like this: `project/ manage.py project/ __init__.py settings.py urls.py wsgi.py` - Run the development server with `$ python manage.py runserver` within the project directory - If you want your `SECRET_KEY` to be more secure, you can set it to reference an environment variable - In the `settings.py` file within the project directory change the `SECRET_KEY` line to the following: `python SECRET_KEY = os.environ.get('SECRET_KEY')` - To quickly generate a random hex for your secret key: ```python

> | | | import secrets secrets.token*hex()* `` `- You can set this environment variable in your shell with` *export SECRET*KEY=`

:page*with*curl: Creating an app

- Navigate to the outer project folder `$ cd <outer_project_folder>`
- Create app with `$ python manage.py startapp <app_name>`
- Inside the `app` folder, create a file called `urls.py`

The project directory should now look like this:
`project/ manage.py db.sqlite3 project/ __init__.py settings.py urls.py wsgi.py app/ migrations/ __init__.py __init__.py admin.py apps.py`
- 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:
`python INSTALLED_APPS = [ 'app', # ... ]` - To migrate changes over: `bash $ python manage.py migrate`

:tv: Creating a view

- Within the app directory, open `views.py` and add the following: ```python from django.http import HttpResponse

def index(request): return HttpResponse("Hello, World!") `- Still within the app directory, open (or create) `urls.py`` python from django.urls import path from . import views

urlpatterns = [path('', views.index, name='index'),] `- Now within the project directory, edit `urls.py` to include the following` python from django.contrib import admin from django.urls import include, path

urlpatterns = [path('app/', include('app.urls')), path('admin/', admin.site.urls),]
`- To create a url pattern to the index of the site, use the following urlpattern:` python 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.

:art: Creating a template

- Within the app directory, HTML, CSS, and JavaScript files are located within the following locations: `app/ templates/ index.html static/ style.css script.js`
- To add a template to views, open `views.py` within the app directory and include the following: ```python from django.shortcuts import render

def index(request): return render(request,'index.html') `- To include context to the template:` python 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:` html {% load static %}

<!DOCTYPE html>

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

`- To make sure to include the following in your `settings,py`:` python STATIC*URL = '/static/' STATICFILES*DIRS = [os.path.join(BASE_DIR, "static")]
`- To add an `extends`:` html {% extends 'base.html'% }

{% block content %}

Hello, World!

{% endblock %} `- And then in `base.html` add:` html {% block content %}{% endblock %} ```

:ticket: Creating a model

- Within the app's `models.py` file, an example of a simple model can be added with the following: ```python 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 is optional.*
- A one-to-many relationship can be made with a `ForeignKey`: ```python 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:` python

> > > m = Musician.objects.get(pk=1) a = m.album_set.get() ```

- A many-to-many relationship can be made with a `ManyToManyField`: ```python 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`: `python ForeignKey(SomeModel, unique=True)`

- For more detail, the official documentation for database models provides a lot of useful information and examples.

:postbox: Creating model objects and queries

- Example `models.py` file: ```python 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 ` ` python

> > > 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:` python b.name = 'The Best Beatles Blog' b.save() `- To retrieve objects:` python all*entries = Entry.objects.all() indexed*entry = Entry.objects.get(pk=1) find_entry = Entry.objects.filter(name='Beatles Blog') ```

:man: Using the Admin page

- To create a `superuser`: `bash $ python manage.py createsuperuser`
- To add a model to the Admin page include the following in `admin.py`: ```python from django.contrib import admin from .models import Author, Book

admin.site.register(Author) admin.site.register(Book) ```