**Django Cheat Sheet**

|  |  |
| --- | --- |
| **Start a project:**  Example Project Name: Python\_Django\_Blog | *django-admin startproject Python\_Django\_Blog* |
| **Create an application inside the project:**  Example App Name: blog | *cd Python\_Django\_Blog*  *python manage.py startapp blog*  The urls.py in Python\_Django\_Blog handles the main route before it passes the matched route to other applications.  urls.py will not be created by default in blog app, you need to add this manually.  The views.py in blog app handles what the routes in urls.py should return to the user (render). It is always tied together.  General phase:   * Create route from urls.py main project to app project. * Create views.py content from the project. * Connect urls.py to views.py in that project. |
| **Run the Web Server:** | *python manage.py runserver* |
| **Templates**  Templates (html files) are used so that an html file can be directly displayed. Templates should be created in a way where it is not repeatable (ex: having a base template.) | You need to add first the created app in the INSTALLED\_APPS in the main project (settings.py).  Add *blog.apps.BlogConfig*  Create a folder named templates/project\_name in the project’s folder and add the html files there. In our case we should have:  *templates/blog* in the blog folder.  To display it, we edit the views.py and use the render function pointing to that file. To point to that file, add the return like: |
| **Passing static date from templates:**  Sample how to a static data from templates to be displayed in the web page. This data normally comes from a database, this is just a guide how to pass.  Jinja templating is used. | The render function accepts a third argument, a dictionary containing a list of dictionaries.    You must pass this to the render as:  *return render(request, "blog/home.html", {"posts": posts})*  Then in html you can loop through the posts using jinja as:  {% for post in posts %}  <h1{{post.title}}</h1>  <p>{{post.author}}</p>  <p>{{post.date\_posted}}</p>  <p>{{post.content}}</p>  {% endfor %} |
| **Using a base html template:**  Having a base template removes repetitions in html files. So when making changes, you don’t need to edit each every file. |  |
| **Adding static files bootstrap template:**  Setting up how static files should be created and accessed so you can add files like CSS and JS.  Then just simply adding the bootstrap template. | Just like in how you create the templates folder, same in static, you should have a folder like:  *static/blogs*  Then you can place inside there the files/folder you need. Example if you have css/main.css, then you can access it in the html files as: |
| **Referencing to links by name:**  Putting links in the html should not be hardcoded because when you change it at one place, you need to change it throughout the whole program. The best way is to reference it via links, that is why the urls.py have name in it. |  |
| **Migrations and Admin Page:**  Django comes with an admin page where you can basically do CRUD operations as an admin.  In order for you to create a super user or database, you have to create migrations and migrate first. | To create an admin page, you need first to create a superuser.  Commands:  *python manage.py makemigrations*  *python manage.py migrate*  The makemigrations command will detect the changes you have made in the database.  The migrate command applys those changes.  When you first create the project, no database has been applied yet but it is already created, you just need to run one. The database contains data for users which is build-in in Django where you can do things such as authentication, forgot password, login, logout.  To create a super user:  python manage.py createsuperuser  Then it’ll ask for username, email and password.  You can access the admin page by:  Localhost:8000/admin |