**Chapter 1**

1. Create virtual environment
2. pip install flask
   1. Check Flask installed correctly by going into the interactive shell of Python and then -> import Flask
   2. This should not give an error
3. Create a new folder called Flask\_Blog and inside that folder create a file called flaskblog.py
4. The flaskblog.py file would contain the below:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

@app.route("/home")

def home():

    return "<h1>Home Page</h1>"

@app.route("/about")

def about():

    return "<h1>About Page</h1>"

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

1. Next in the terminal, cd to Flask\_Blog folder and then export the Flask app to an environment variable by below:
   1. SET FLASK\_APP = flaskblog.py
2. Next enter the following in the console:
   1. flask run
   2. This would run the flask “production” server at <http://127.0.0.1:5000/>
3. To run in the debug mode, we need to do the following:
   1. SET FLASK\_DEBUG = 1
   2. Note: The above is not to be confused with development as debug and development are two different things. The above just runs the server in the debug mode.
   3. In the debug mode, if we make any changes to the app, then we don’t have to restart the server to see the changes in the browser.
4. If we don’t want to run the app with the environment variable (i.e. flask run) then there is another way of running our application directly with Python. To do this we need to add the following line of code in the above flaskblog.py file.

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

Then in the console we can run the same app by python flaskblog.py. Essentially while running directly with Python, we want to run our application in the debug mode. While running directly with Python, we cannot use the environment variable, so we have to set the debug mode here in the code itself.

* 1. Note: to understand the \_\_name\_\_ concept, refer to this article: <https://stackoverflow.com/questions/419163/what-does-if-name-main-do#:~:text=It%20checks%20if%20the%20__,case%20the%20main()%20function).>
  2. Essentially, if we use python to run a module say for example in the terminal, **python flaskblog.py** then the \_\_name\_\_ variable is given a special hard coded value of ‘\_\_main\_\_’.
  3. If the module flaskblog.py is imported in another module by the **import statement** then the \_\_name\_\_ variable is given the name ‘flaskblog’

**Chapter 2**

If we want to return html templates, instead of the raw html then we create a folder called templates, and put our html templates within than folder.

Create html file and then use the emmet shortcut ! and hit enter to put blank html in the file.

In flask static files like css and js need to be put in a static folder, so we need to create that folder and keep our css and js files in that.

In Django we used the url to find the exact locations of our routes. For example, we used the below in the footer to access the about page.

href="{% url 'about' %}"

In Django this works differently. We define a name for the route in the urls.py file, while defining the routes. Then the url in the curly braces works out the url for the route using this alias name.

path('about/', blog\_views.about, name = 'about')

In flask we need to import the **url\_for** module in the main app module i.e. flaskblog.py.

from flask import Flask, render\_template, url\_for

Then in any of the templates we can use the url\_for within curly braces.

href="{{ url\_for('static', filename='main.css') }}">

Note that although the url\_for within curly braces look similar, the approach is totally different. Also, unlike in Django where to access any python function within a template, we have to use a context processor, we can directly use some of the module functions in the templates of Flask. This is because many functions in the Flask templates in the jinja templating language use the exact same function names as in the modules, as in the example above.

<https://stackoverflow.com/questions/58880873/run-a-python-module-in-a-flask-html-template>

**Chapter 3 - Forms and Validation**

Install wtf forms. pip install flask-wtf

Create forms.py, where we define the forms class model(s).

We config a SECRET\_KEY for our app by using the app.config in the main app py file (flaskblog.py). The value for the secret key is generated by using the python secrets module.

*import secrets*

*secrets.token\_hex(16)*

*9e3437defc928a23906c0852b352cf17*

Create routes for login and registration

We are using {{ form.hidden\_tag() }} to do the csrf token as we did in Django (we used {% csrf\_token %} over there).

Also note that when a user presses the submit button on a form, the front-end application itself has an action item after the submit button is pressed. However, we want the redirect to be performed from the back-end i.e. the flask application. That is why, the action item is blank in the html form, because we want the front-end to not cause the page to redirect but the back-end to cause this.

**Chapter 4 – Database**

Sql lite in development and Postgres in Production

pip install flask-sqlalchemy

To create sql database using alchemy type this in the interactive python console :

>>> from flaskblog import db

>>> db.create\_all()

this would create the site.db database

Next some experiments as below:

>>> from flaskblog import User, Post

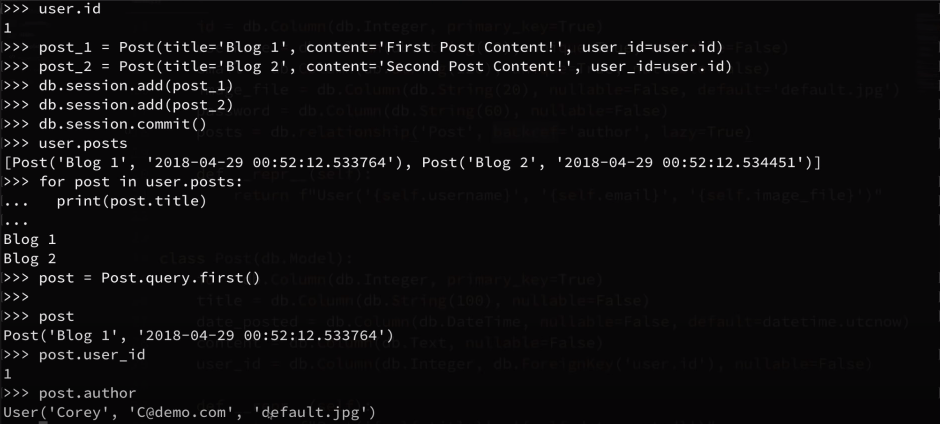
>>> user1 = User(username = 'Mausam', email = 'm.g@gmail.com', password = 'password')

>>> db.session.add(user1)

>>> db.session.commit()

>>> User.query.all()

This is what the author field does. Basically the entire user instance for a post can be retrieved with the author field.



To remove all data from the database, perform the below:

db.drop\_all()