Flask with Python

Our first web app

What's Flask?

- Flask is what's known as a WSGI framework. Mercifully pronounced "whiskey,"
- This stands for Web Server Gateway Interface.
- Essentially, this is a way for web servers to pass requests to web applications or frameworks

Preparations Needed:

1. Installing Flask Packages

   - **`Pip install flask`**
   - **`pip install flask-login`** → Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering your users' sessions over extended periods of time.

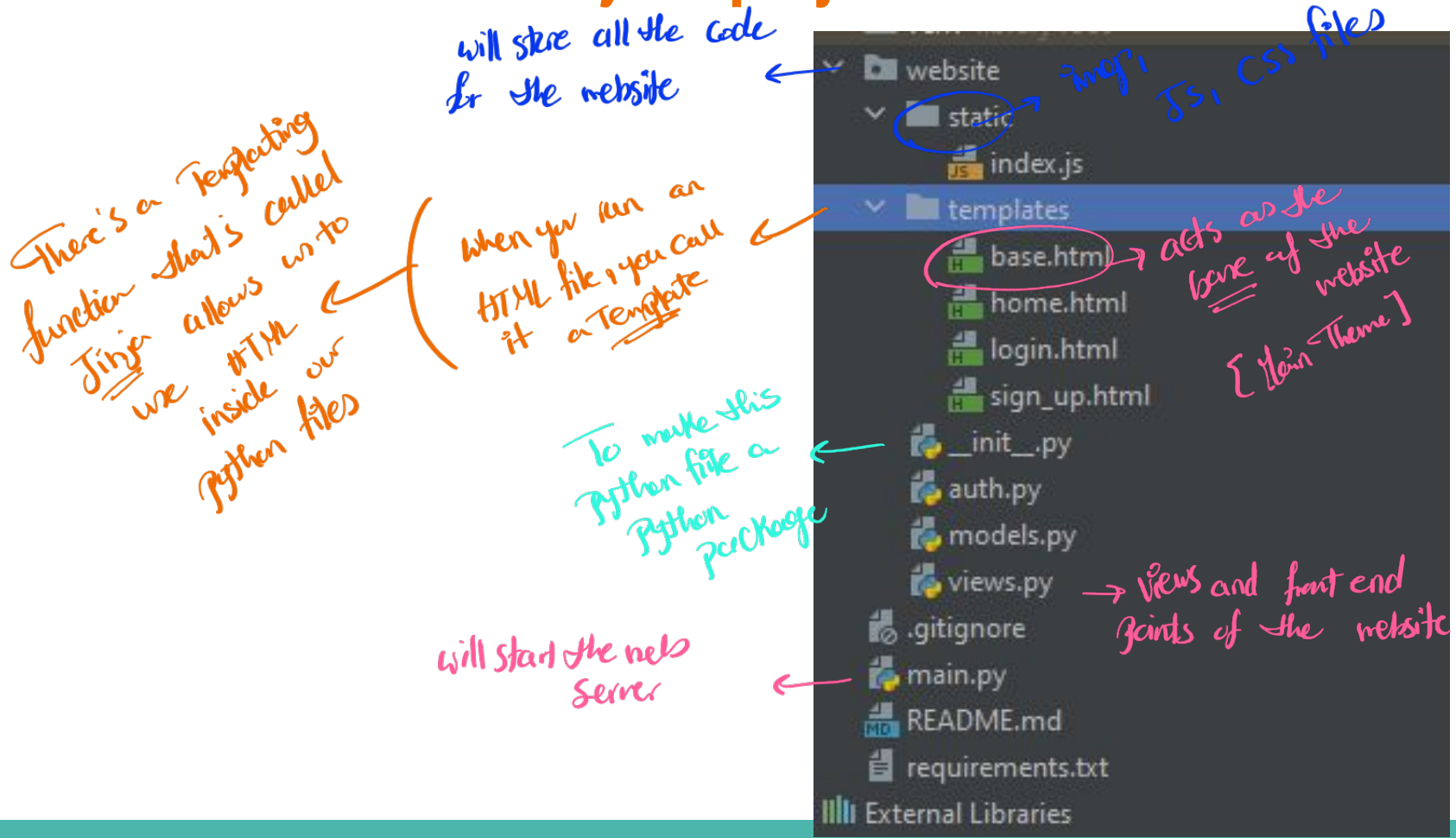
   - **`Pip install flask-sqlalchemy`** → Flask-SQLAlchemy is an extension for Flask that adds support for **SQLAlchemy** to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

SQLAlchemy

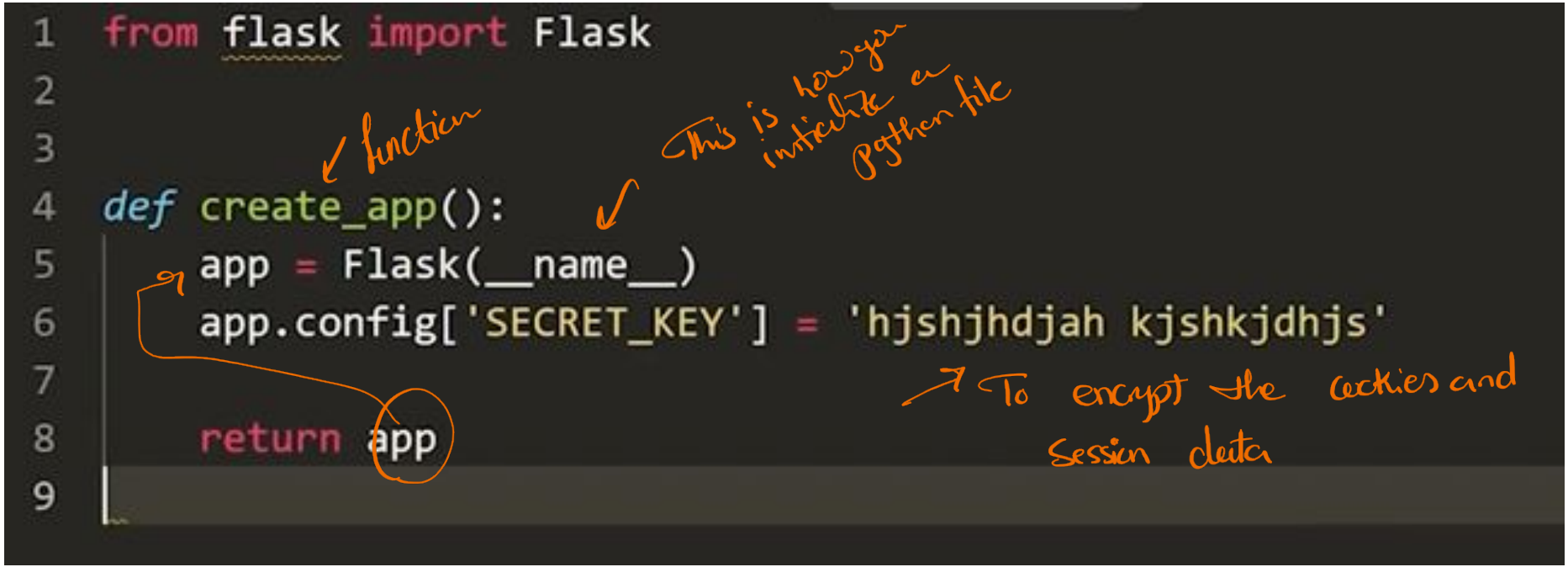
- SQLAlchemy is a library that **facilitates the communication between Python programs and databases**.

- Most of the times, this library is used as an Object Relational Mapper (ORM) tool that translates **Python classes** to **tables** on relational databases and automatically converts **function calls to SQL statements.**

Files to be created within your project:



__init__.py


1 from flask import Flask
2
3
4 def create_app():
5 app = Flask(__name__)
6 app.config['SECRET_KEY'] = 'hjshjhdjah kjshkjdhjs'
7
8 return app
9
function
This is how you initialize a python file
To encrypt the cookies and session data

Main.py

```
1  from website import create_app
2
3  app = create_app()
4
5  if __name__ == '__main__':
6      app.run(debug=True)
7
```

Python package

Only if we run this file, we will run the server not if we just imported it.

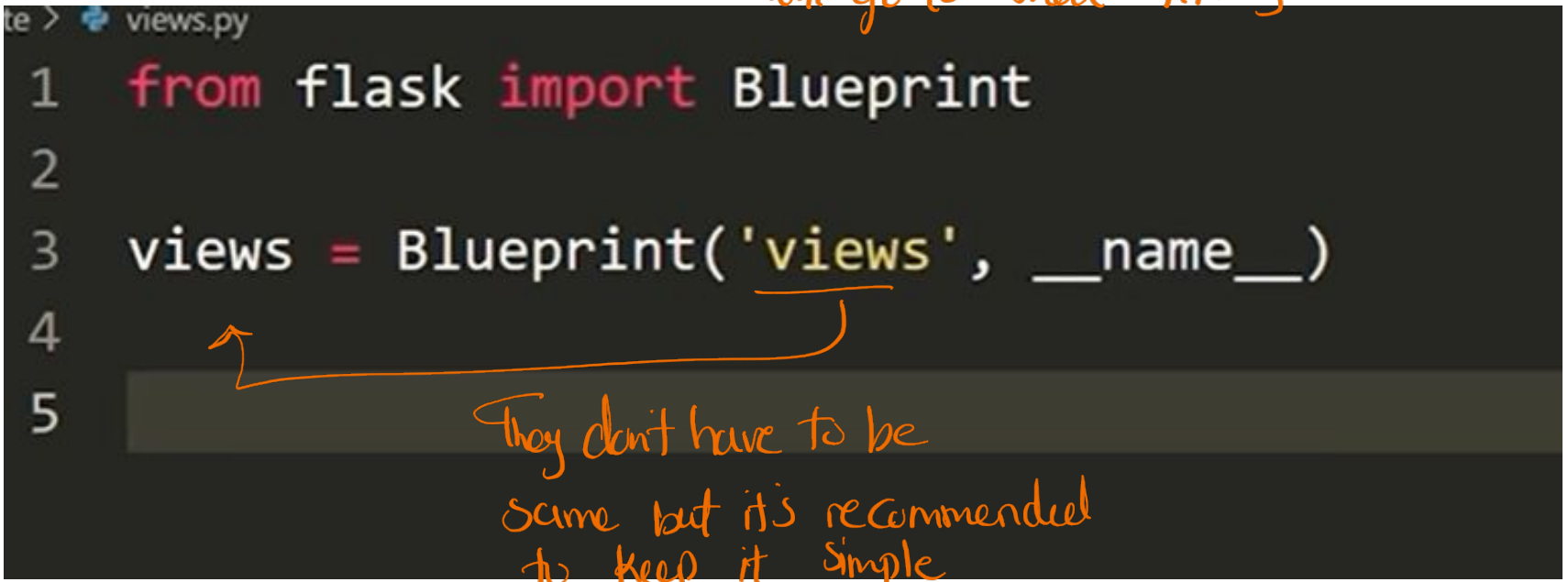
Every time we change the Python code, it will automatically run the server

What is a blueprint?

A blueprint defines a collection of views, templates, static files and other elements that can be applied to an application. For example, let's imagine that we have a blueprint for an admin panel. This blueprint would define the views for routes like */admin/login* and */admin/dashboard*. It may also include the templates and static files that will be served on those routes. We can then use this blueprint to add an admin panel to our app, be it a social network for astronauts or a CRM for rocket salesmen.

Views.py

where we gonna save the standard routes of our web server [anything that's not authentication will go to that file]



Auth.py

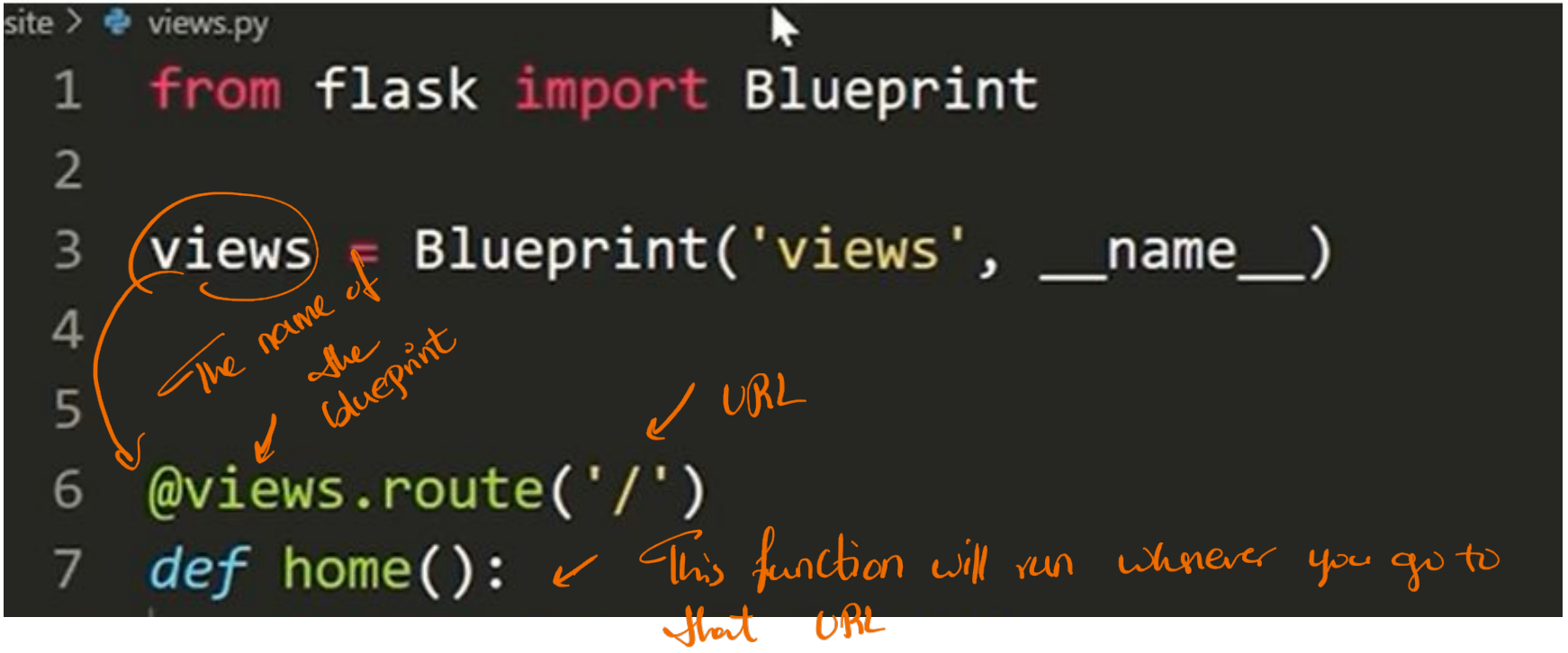
website > auth.py

```
1  from flask import Blueprint
2
3  auth = Blueprint('auth', __name__)
4
```

same thing

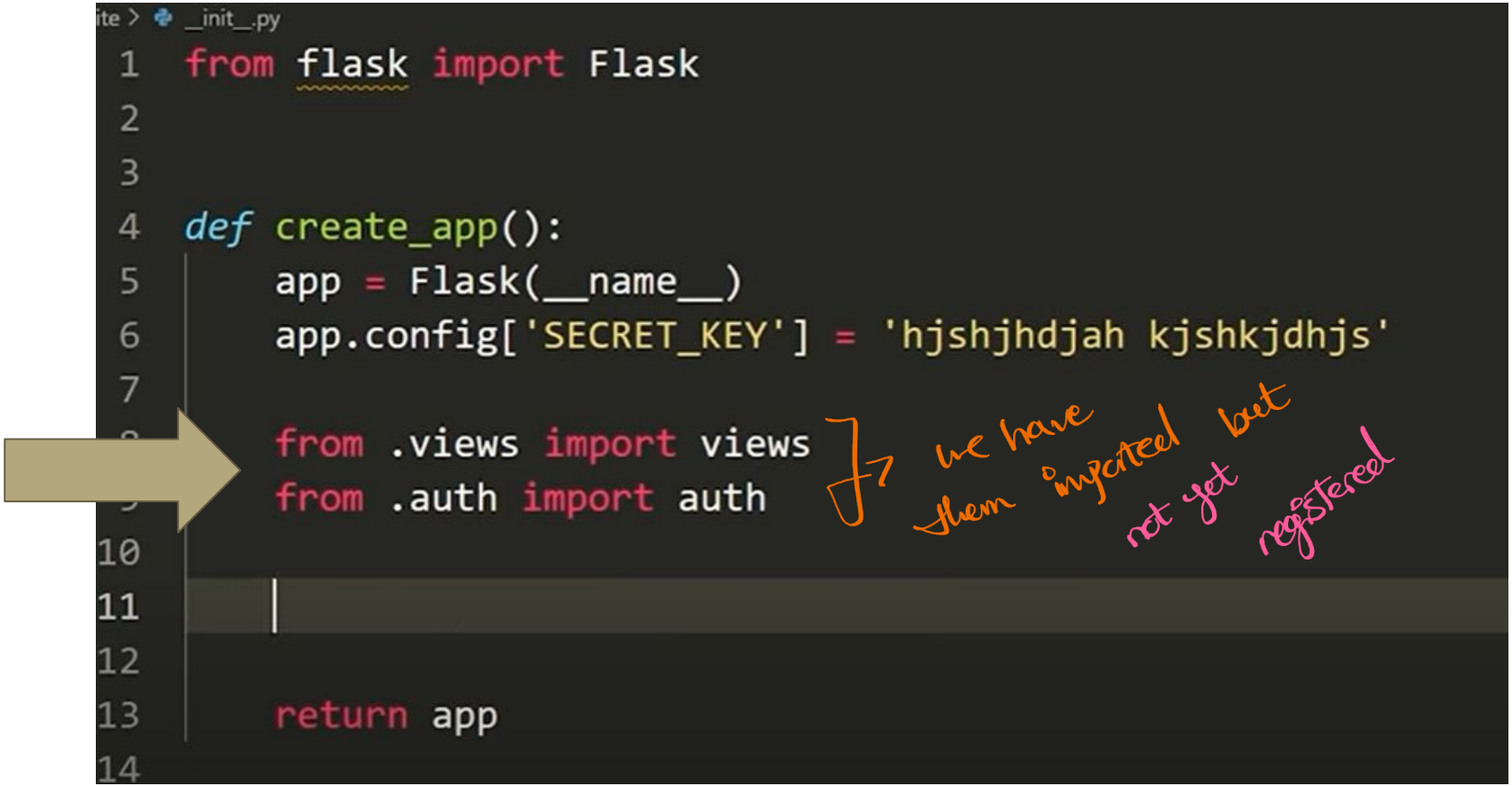
↳ Now I say that i have a different URLs

Views.Py


site > views.py
1 from flask import Blueprint
2
3 views = Blueprint('views', __name__)
4
5
6 @views.route('/')
7 def home():
The name of the blueprint
URL
This function will run whenever you go to that URL

__init__.py

Now, we have some blueprints, we need to define them into our initialization file to include them inside our packages

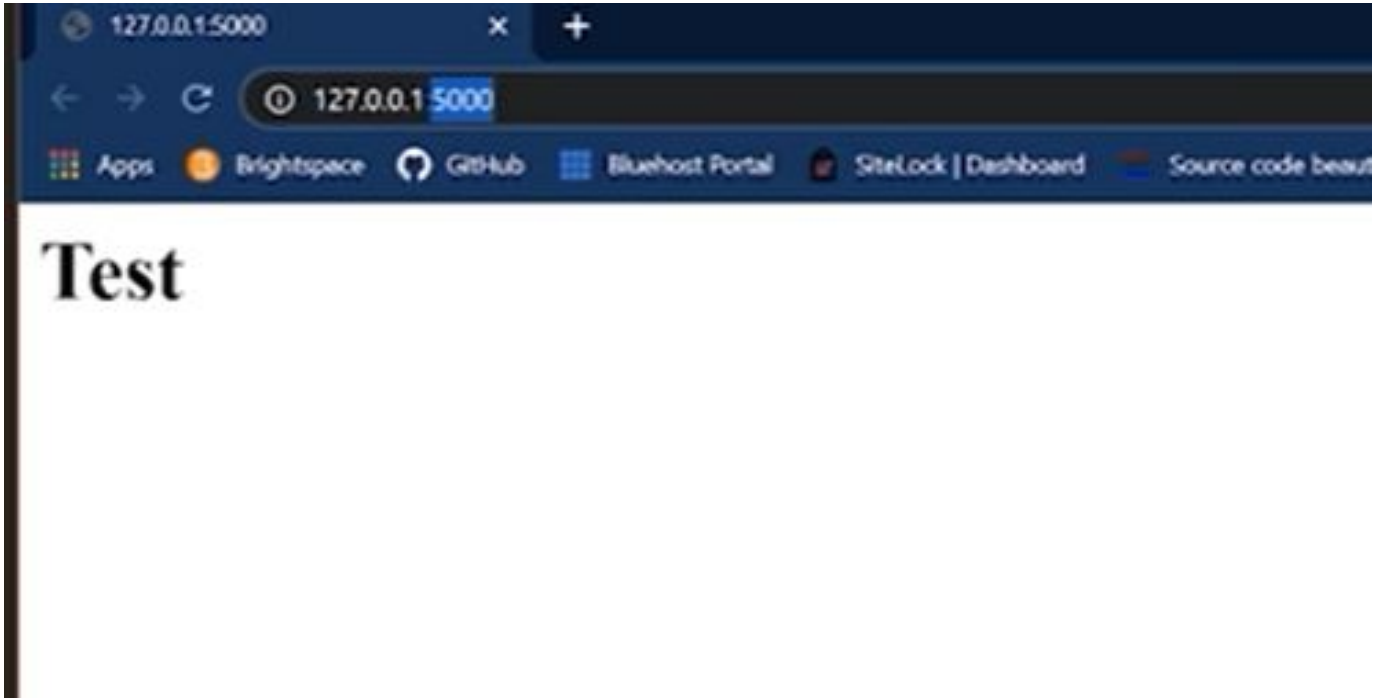


Registering the blueprints on __init__.py file

```python
def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = 'hjshjhdjah kjshkjdhjs'

    from .views import views
    from .auth import auth

    app.register_blueprint(views, url_prefix='/')
    app.register_blueprint(auth, url_prefix='/')

    return app
```

The start of the URL

Run and Test your app

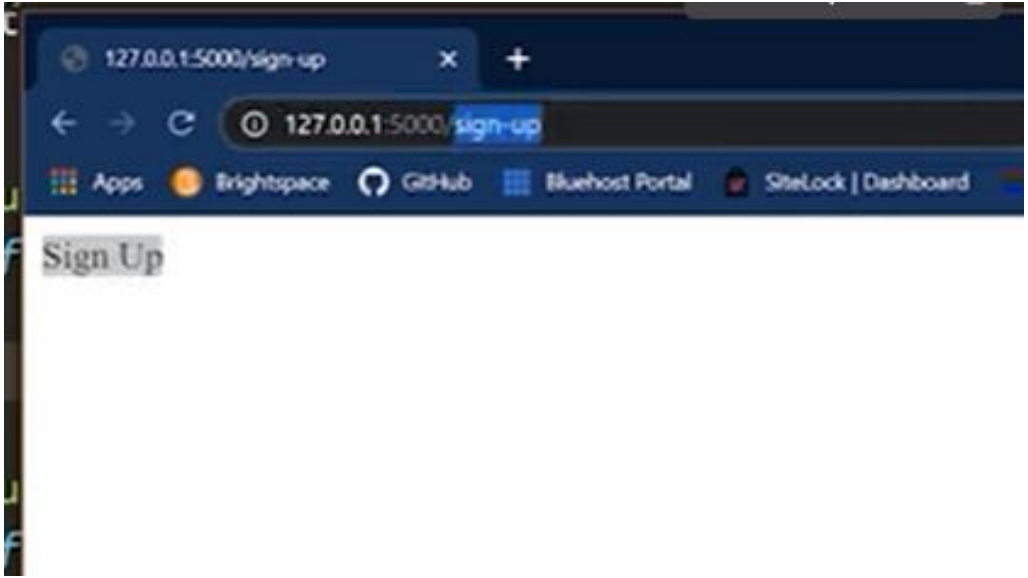


Adding your sign in and up functions on auth.py

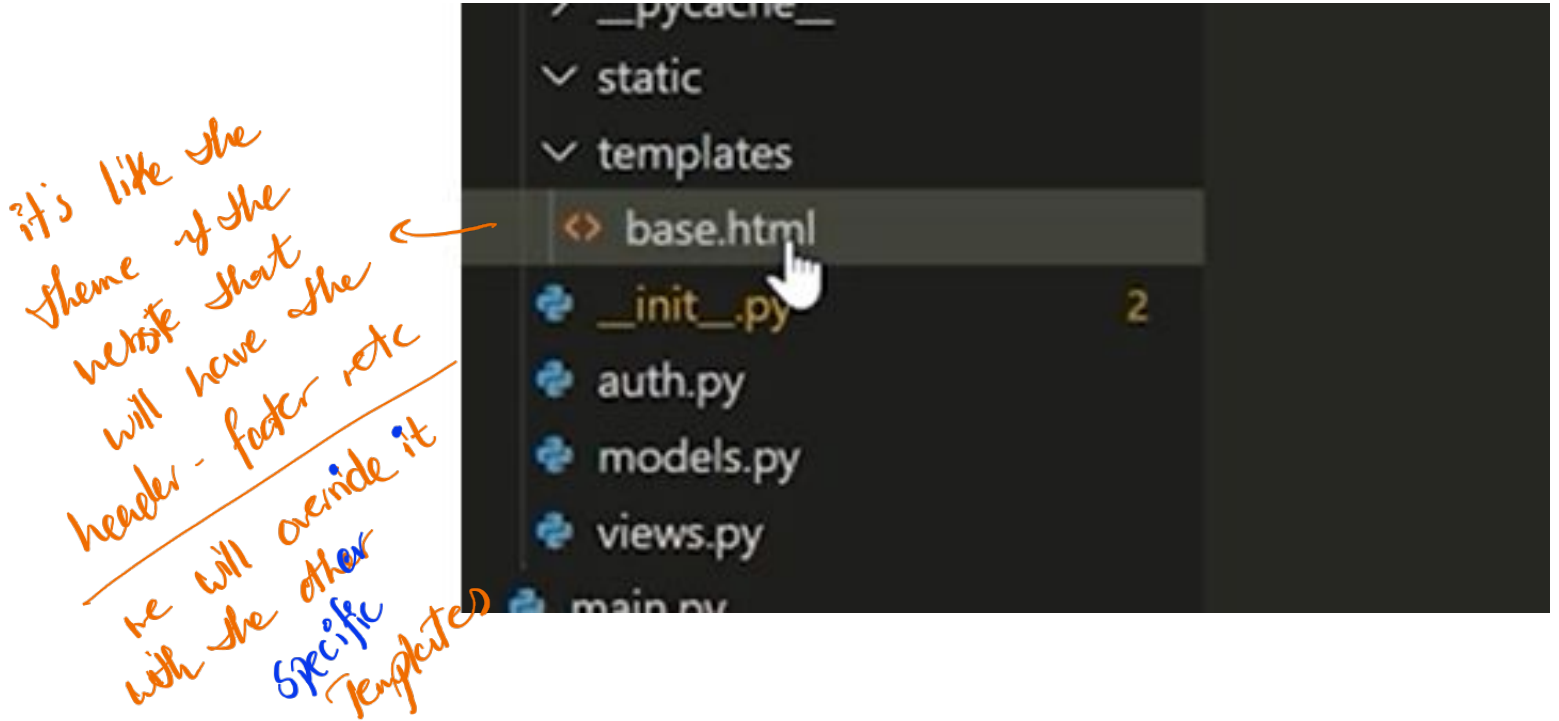
```python
@auth.route('/login')
def login():
    return "<p>Login</p>"


@auth.route('/logout')
def logout():
    return "<p>logout</p>"


@auth.route('/sign-up')
def sign_up():
    return "<p>Sign Up</p>"
```

Just for Testing Purposes

Test your three routes



Create an HTML file inside your templates folder

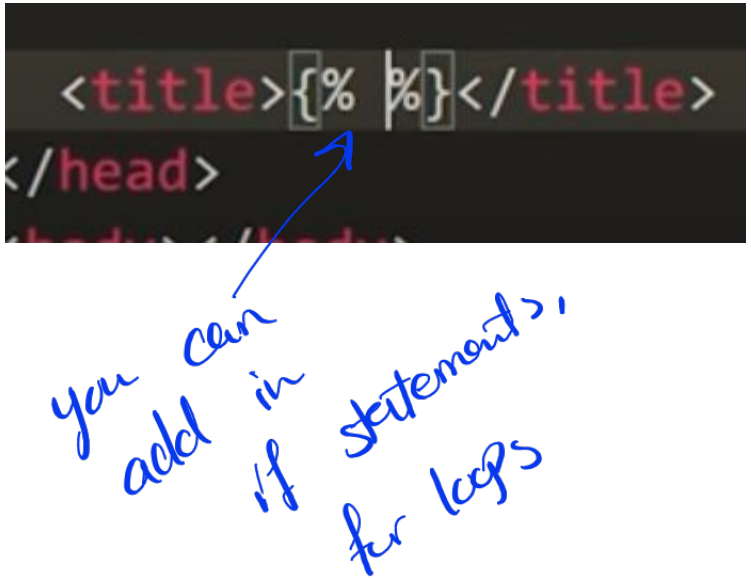


Scan this code for the HTML Files

- These are the files that we gonna use, we imported some bootstrap packages that will help us with some animations to make our app prettier

With Jinja,

The curly brackets and the percentage sign are used to surround any **python** code within your HTML File



```
    />

    <title>{% block title %}Home{% endblock %}</title>
</head>
```

Children will inherit this block but they can override it To change the title

Add in your Nav Bar inside base.html file

```
  <title>{% block title %}Home{% endblock %}</title>
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    </nav>
```

Bootstrap classes ! Check out the Bootstrap documentation if you want

Add in a button on your nav bar in order to expand the menu items if the screen is too small:

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbar"
  >
```

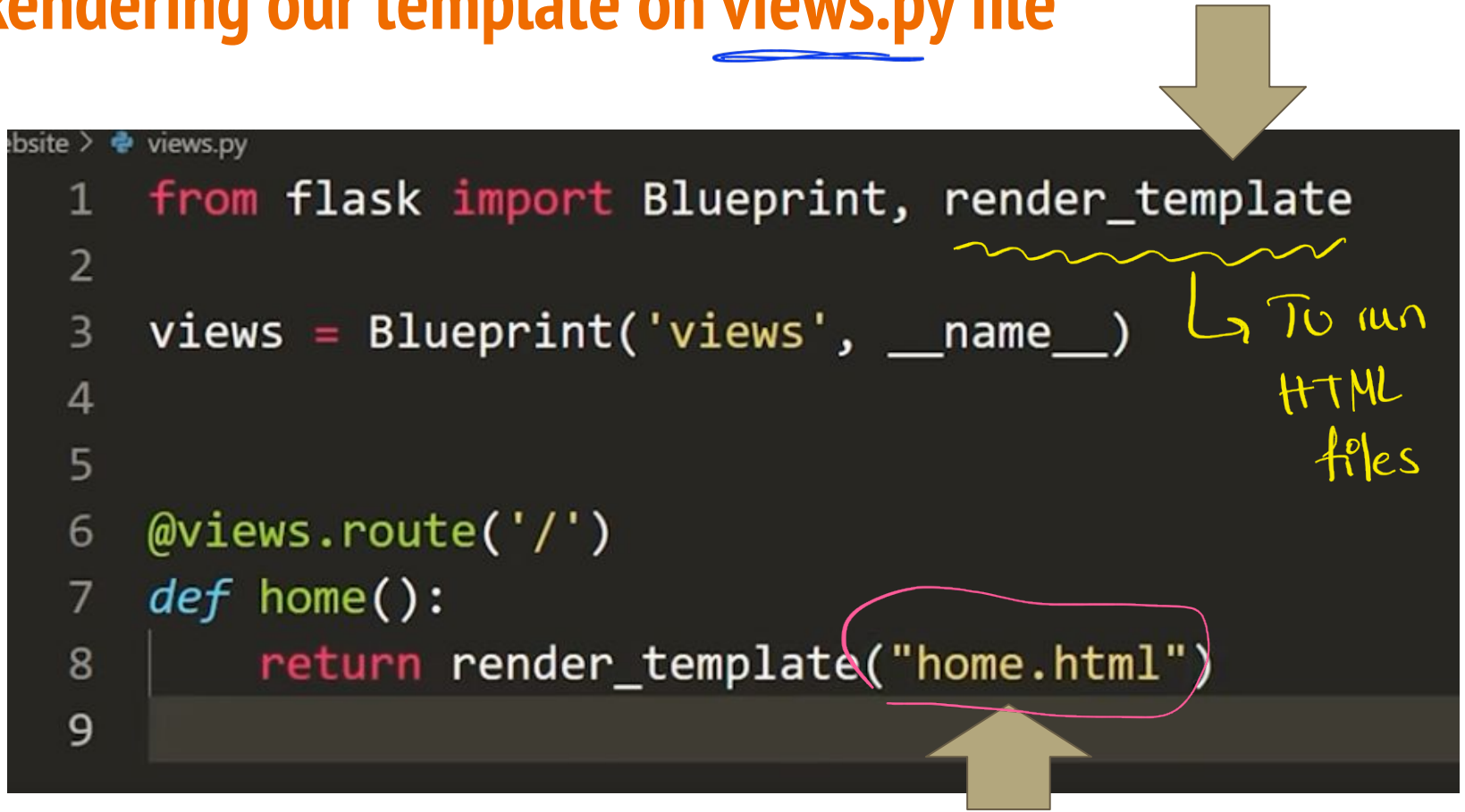
Adding our buttons inside our nav bar:

```
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbar">
    <div class="navbar-nav">
      <a class="nav-item nav-link" id="login" href="/login">Login</
      <a class="nav-item nav-link" id="signUp" href="/sign-up">Sig
      <a class="nav-item nav-link" id="logout" href="/logout">Logo
      <a class="nav-item nav-link" id="home" href="/">Home</a>
```

Extending the base temp into the home.html file

So that home.HTML inherit base.HTML

Rendering our template on views.py file



Rendering our templates on auth.py

Add to your auth.py the route to your templates:

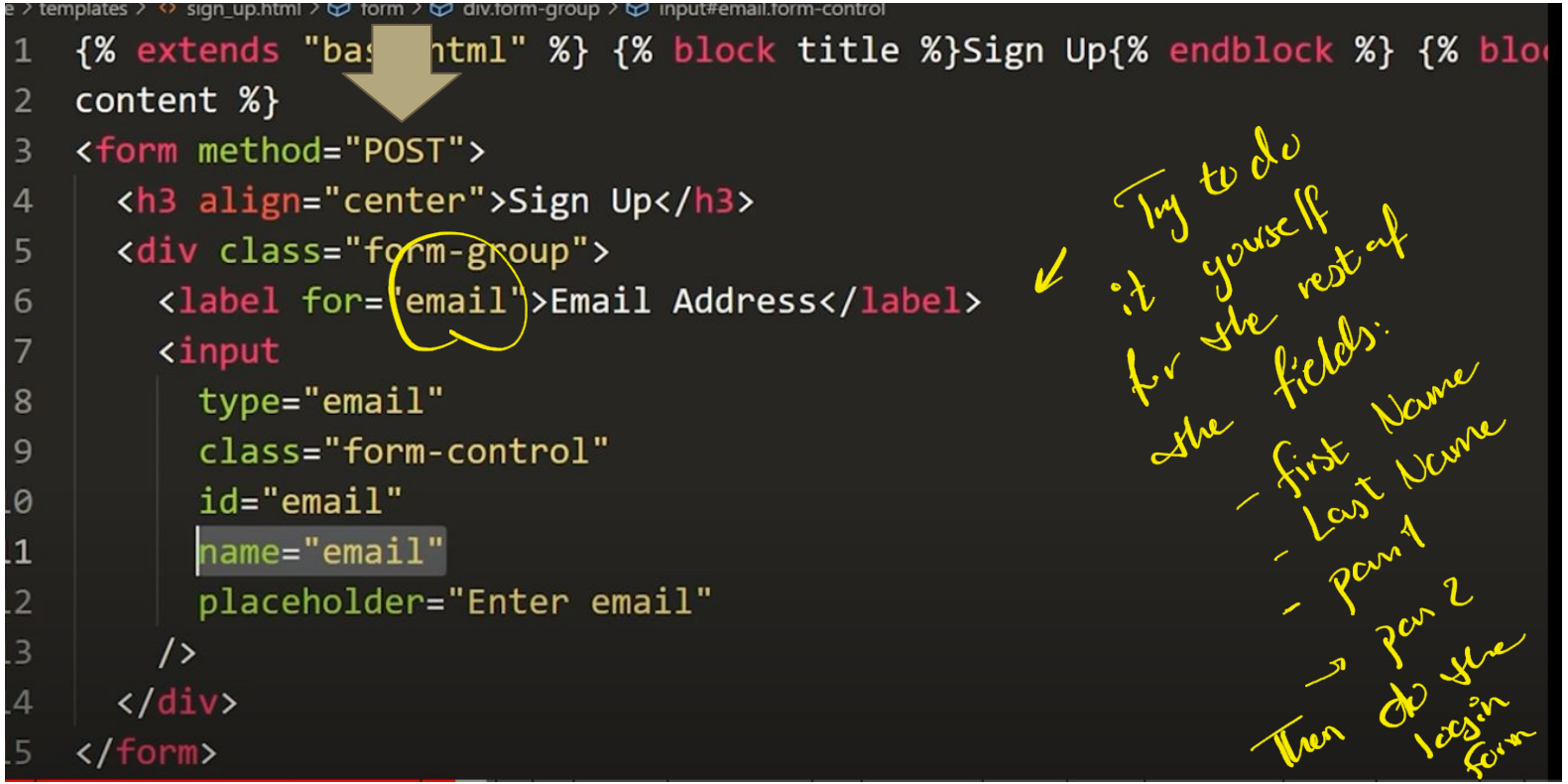
```
@auth.route('/login')
def login():
    return render_template("login.html")
```

Try to add it for your sign up

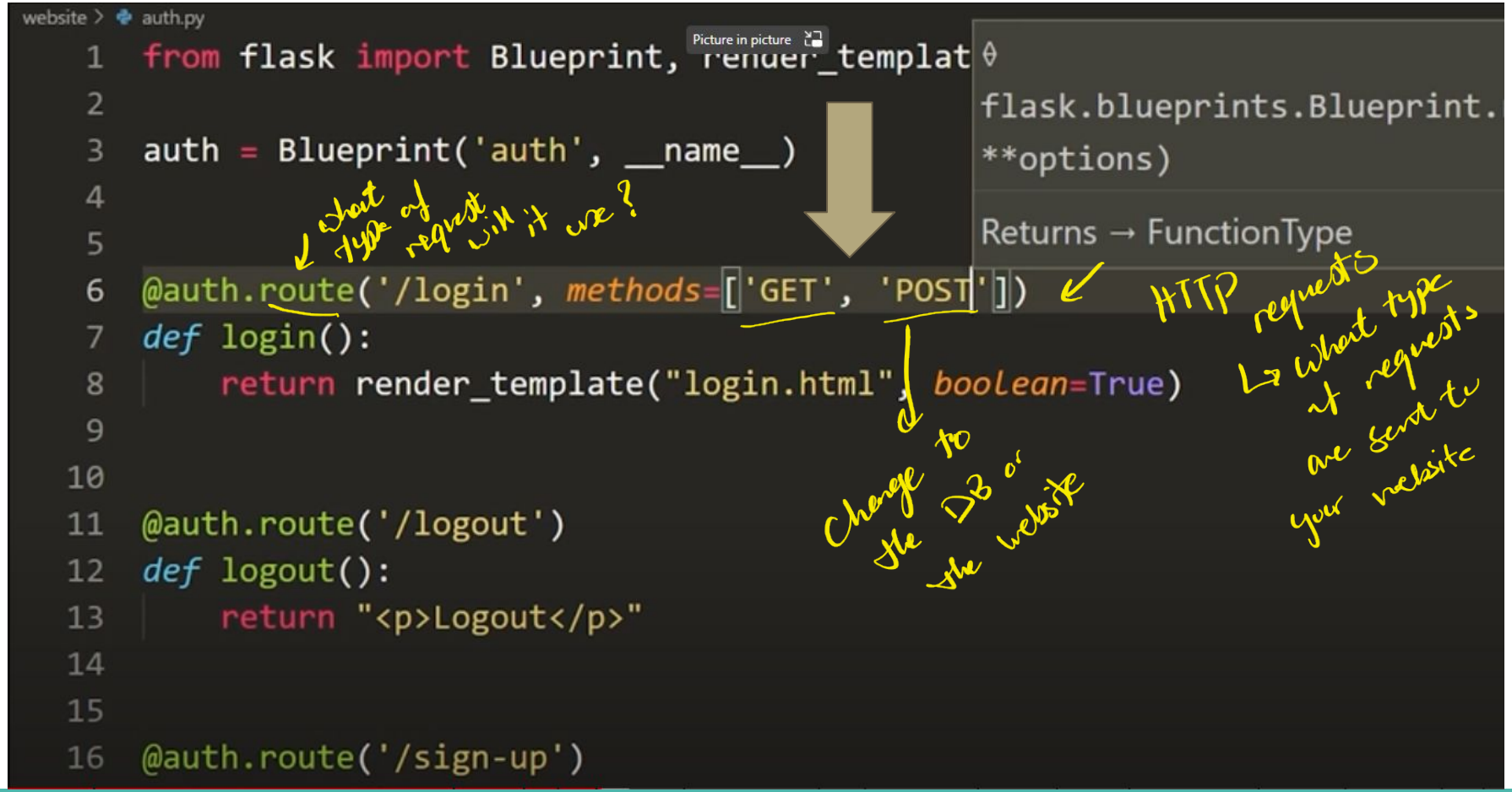
website > auth.py

```
3  auth = Blueprint('auth', __name__)
4
5
6  @auth.route('/login')
7  def login():
8      return render_template("login.html")
9
10
11 @auth.route('/logout')
12 def logout():
13     return "<p>Logout</p>"
14
15
16 @auth.route('/sign-up')
17 def sign_up():
18     return render_template("sign_up.html")
```

Working on your sign up.html



Add in the methods that our functions may accept inside our auth.py file:

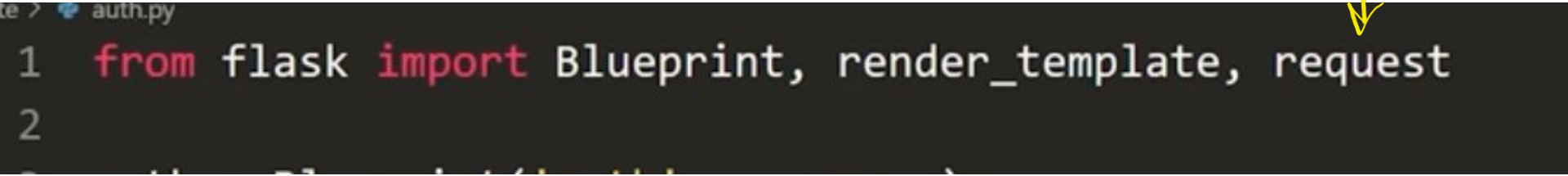


Add it to your sign up method as well:

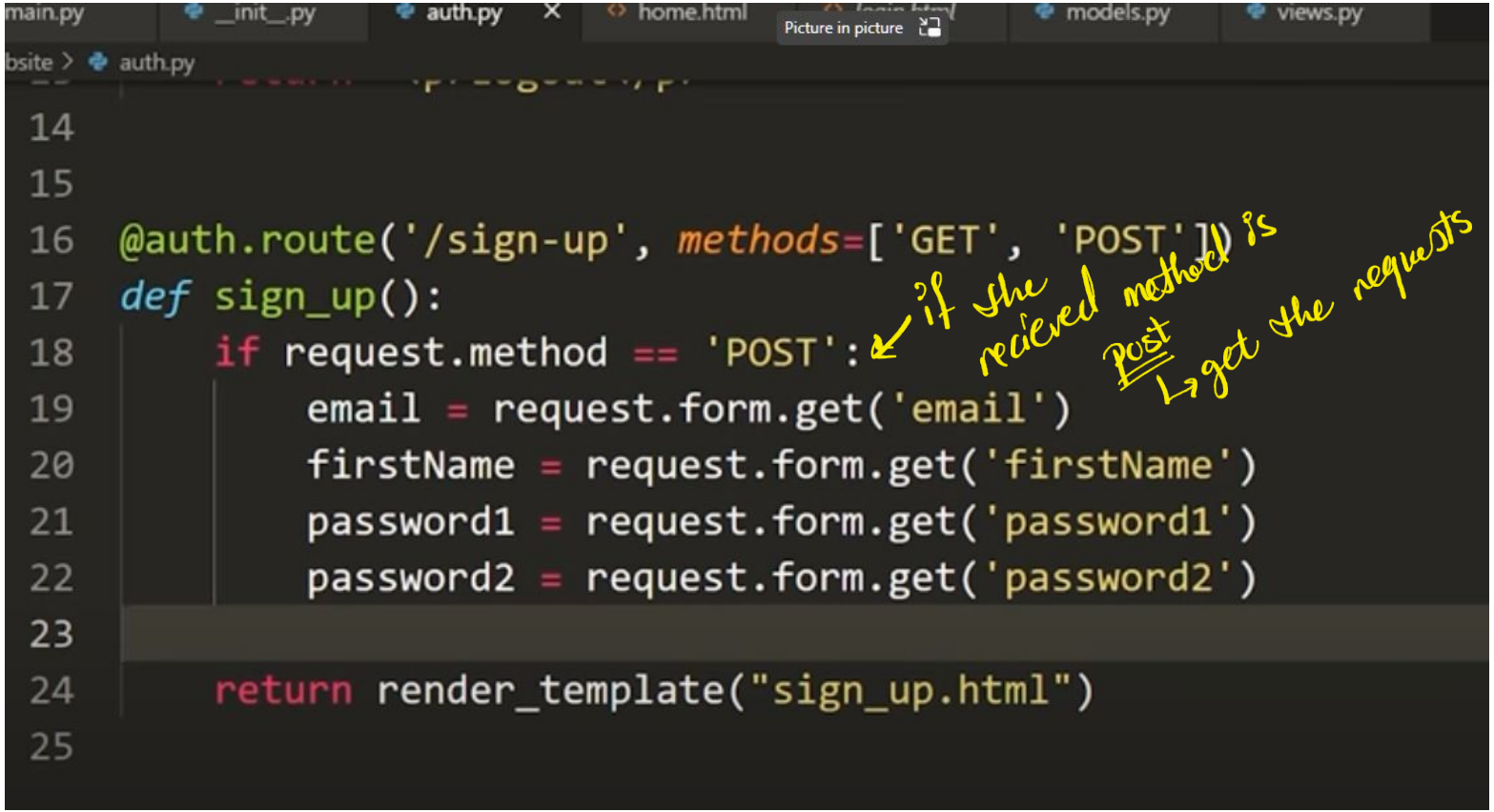
```
4
5
6   @auth.route('/login', methods=['GET', 'POST'])
7   def login():
8       return render_template("login.html", boolean=True)
9
10
11  @auth.route('/logout')
12  def logout():
13      return "<p>Logout</p>"
14
15
16  @auth.route('/sign-up', methods=['GET', 'POST'])
17  def sign_up():
18      return render_template("sign_up.html")
```

```
flask.blueprints.Bluepr
**options)

Returns → FunctionType
```

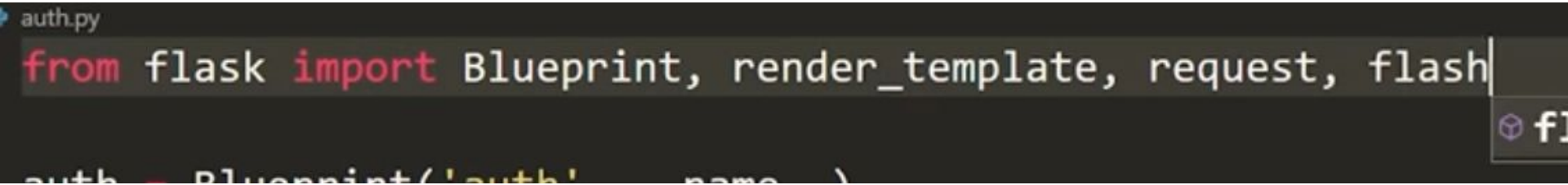
On our auth.py file, we need to import the request



Sign up function on your auth. Py file



What if the user entered a wrong pass or didn't add in an email? You have to have a warning msg [Flashing]



```python

@auth.route('/sign-up', methods=['GET', 'POST'])
def sign_up():
    if request.method == 'POST':
        email = request.form.get('email')
        firstName = request.form.get('firstName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')

        if len(email) < 4:
            flash('Email must be greater than 3 characters.', category='error')
        elif len(firstName) < 2:
            flash('First name must be greater than 1 characters.', category='error')
        elif password1 != password2:
            flash('Passwords don\'t match.', category='error')
        elif len(password1) < 7:
            flash('Password must be at least 7 characters.', category='error')
        else:
            flash('Account created!', category='success')
            pass

```

Now your msgs are flashed but they’re not displayed on the front-end of your app. So we need to go to our base.html file to display them:

main.py | __init__.py | auth.py | base.html | home.html | models.py | views.py

website > templates > base.html > html > body

```
28          <span class="navbar-toggler-icon"></span>
29        </button>
30        <div class="collapse navbar-collapse" id="navbar">
31          <div class="navbar-nav">
32            <a class="nav-item nav-link" id="login" href="/login">Login</
33            <a class="nav-item nav-link" id="signUp" href="/sign-up">Sign
34            <a class="nav-item nav-link" i                          ogou
35            <a class="nav-item nav-link" i
36          </div>
37        </div>
38      </nav>
39
40      {% with messages = get_flashed_messages(with_categories=true) %}
41
42      {% endwith %}
```

That is done below the nav-bar, the with messages is a function that's built in in flask that allows us to receive those msgs

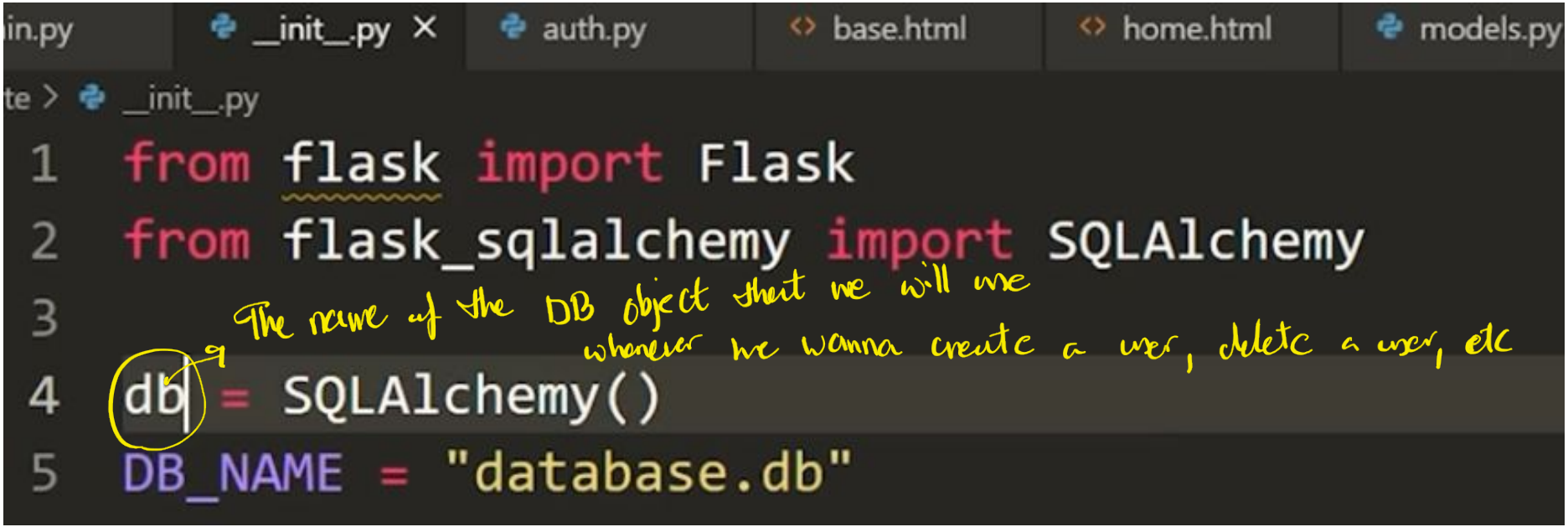
Eg error / success

The msgs are now received but they're not yet displayed.

```
{% with messages = get_flashed_messages(with_categories=true) %} {% if
messages %} {% for category, message in messages %} {% if category ==
'error' %}
<div class="alert alert-danger alter-dismissable fade show" role="alert">
  {{ message }}
  <button type="button" class="close" data-dismiss="alert">
    <span aria-hidden="true">×</span>
  </button>
</div>
{% else %}
<div class="alert alert-success alter-dismissable fade show" role="alert">
  {{ message }}
  <button type="button" class="close" data-dismiss="alert">
    <span aria-hidden="true">×</span>
  </button>
</div>
{% endif %} {% endfor %} {% endif %} {% endwith %}
```

Python

Initializing our DB inside our __init__.py



Setting up our database

main.py | __init__.py | auth.py | base.html | home.html | models.py | views.py

website > __init__.py

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
DB_NAME = "database.db"


def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = 'hjshjhdjah kjshkjdhjs'
    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
    db.init_app(app)

    from .views import views
    from .auth import auth
```

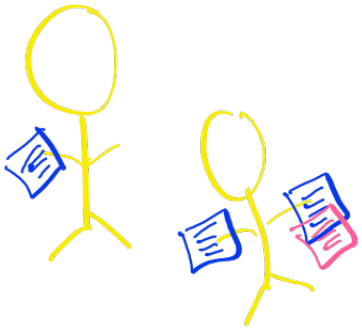
The main website folder

↳ initialize the DB

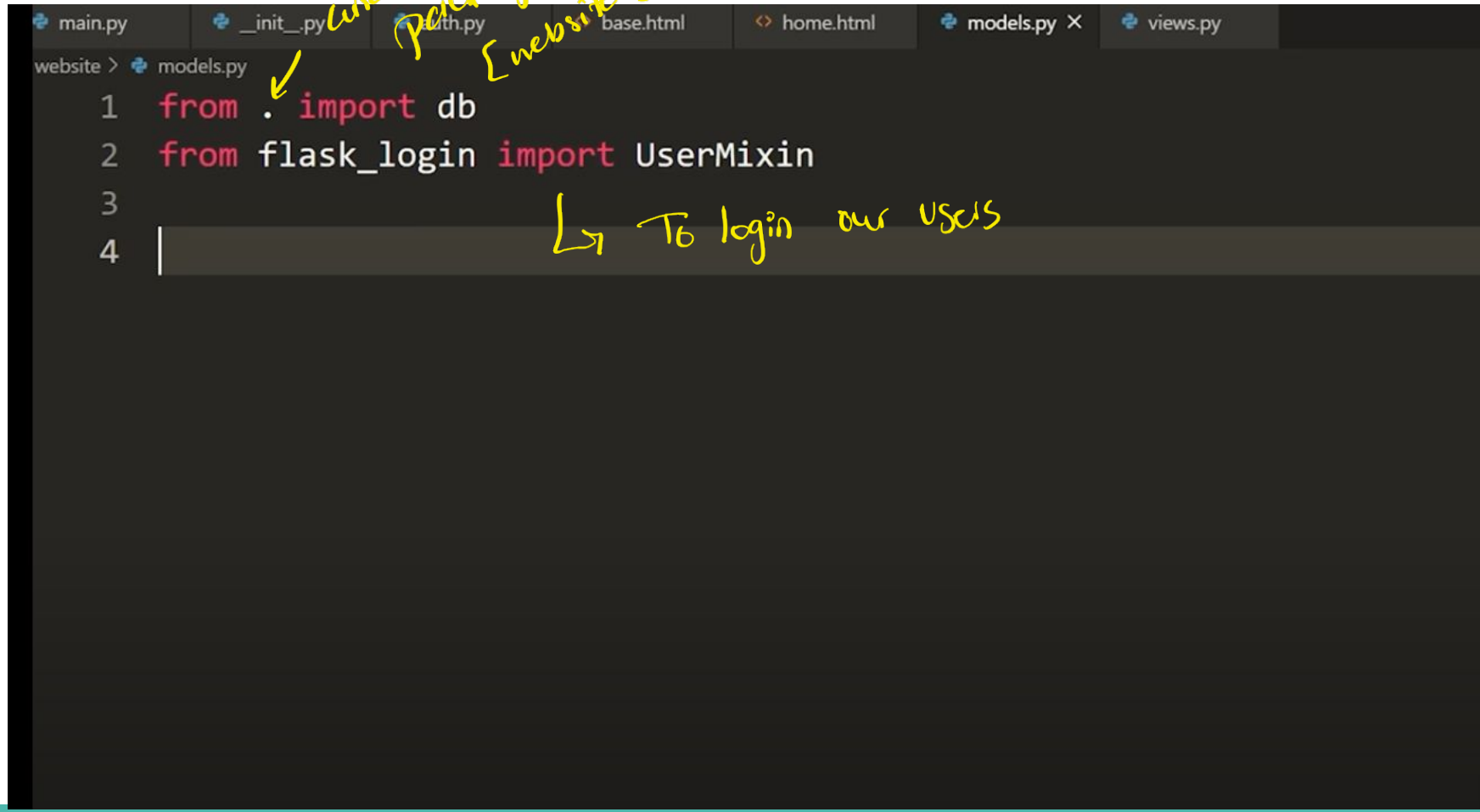
↳ The location of the DB

Let’s head to our models.py file to draft out the schema of our database

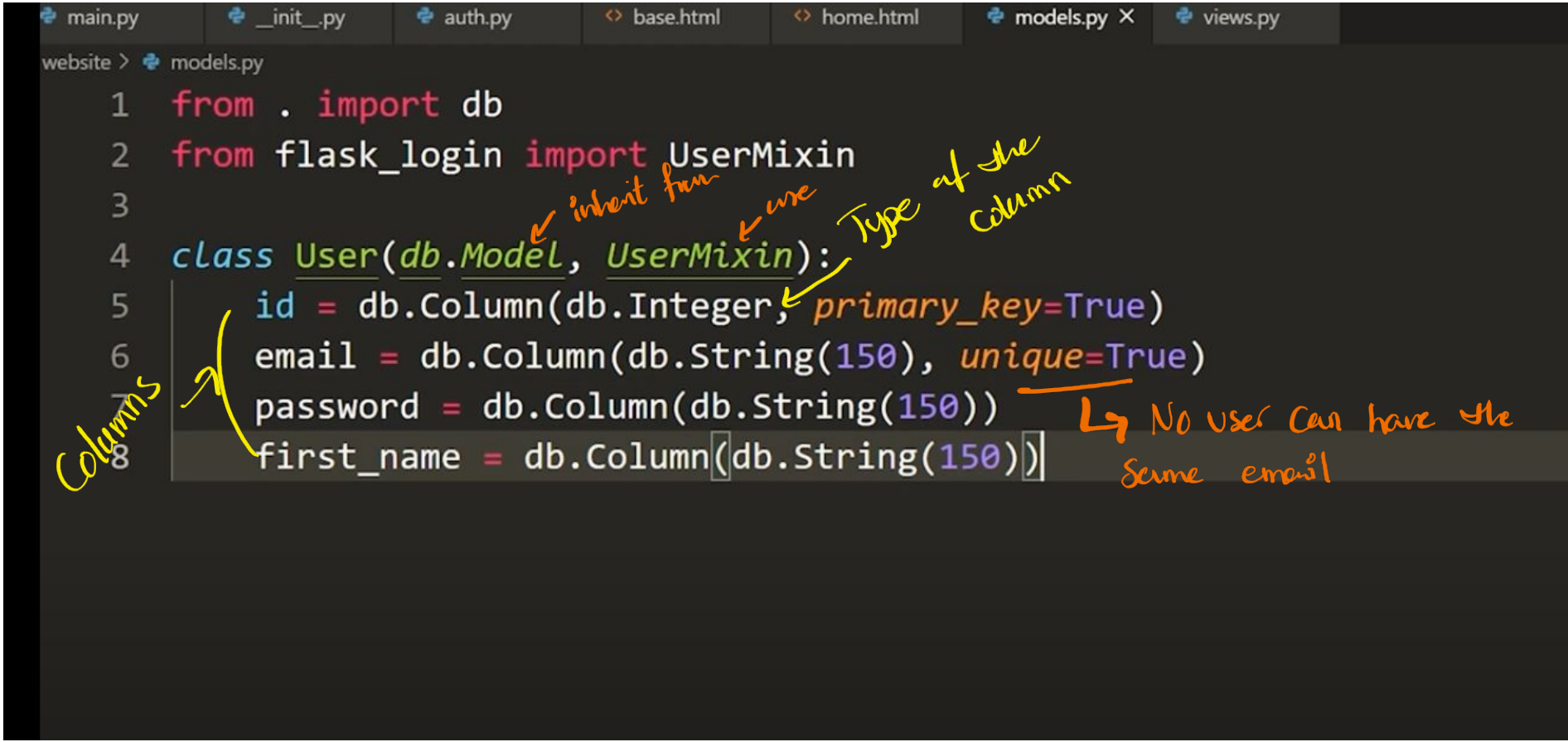
- We gonna have a db for our **Users** and another for our **Notes**



Models.py


main.py
__init__.py
auth.py
base.html
home.html
models.py
views.py
website > models.py
1 from . import db
2 from flask_login import UserMixin
3
4
current package [website]
To login our users

Defining the users table:



Defining the notes table:

main.py | __init__.py | auth.py | base.html | home.html | models.py | views.py

website > models.py

```python
from . import db
from flask_login import UserMixin
from sqlalchemy.sql import func


class Note(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    data = db.Column(db.String(10000))
    date = db.Column(db.DateTime(timezone=True), default=func.now())
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True)
```

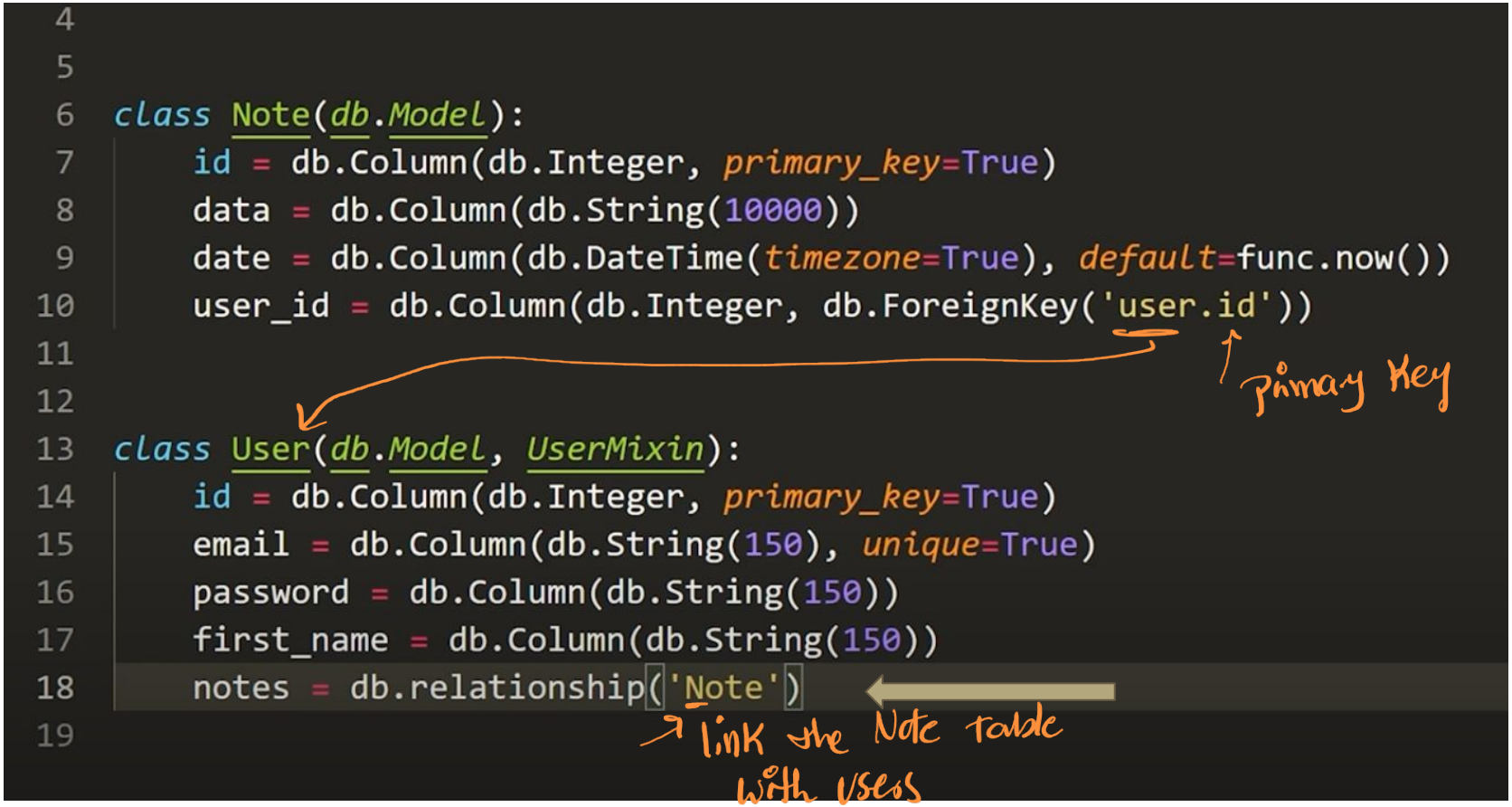
for saving the date

ids are always unique and automatically incremented

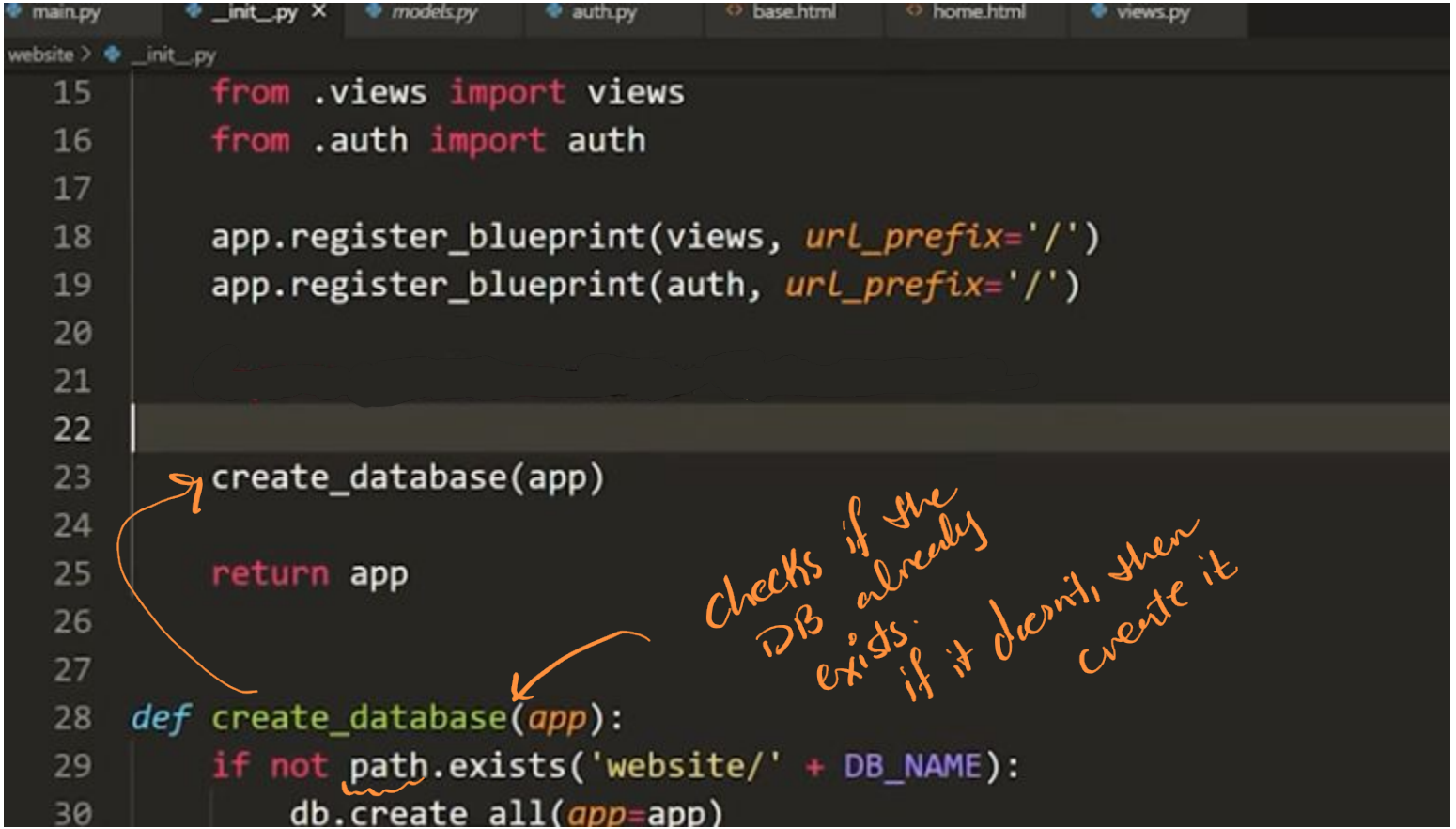
↳ association with the users table

↳ database column that refrences to another Column in a diff table

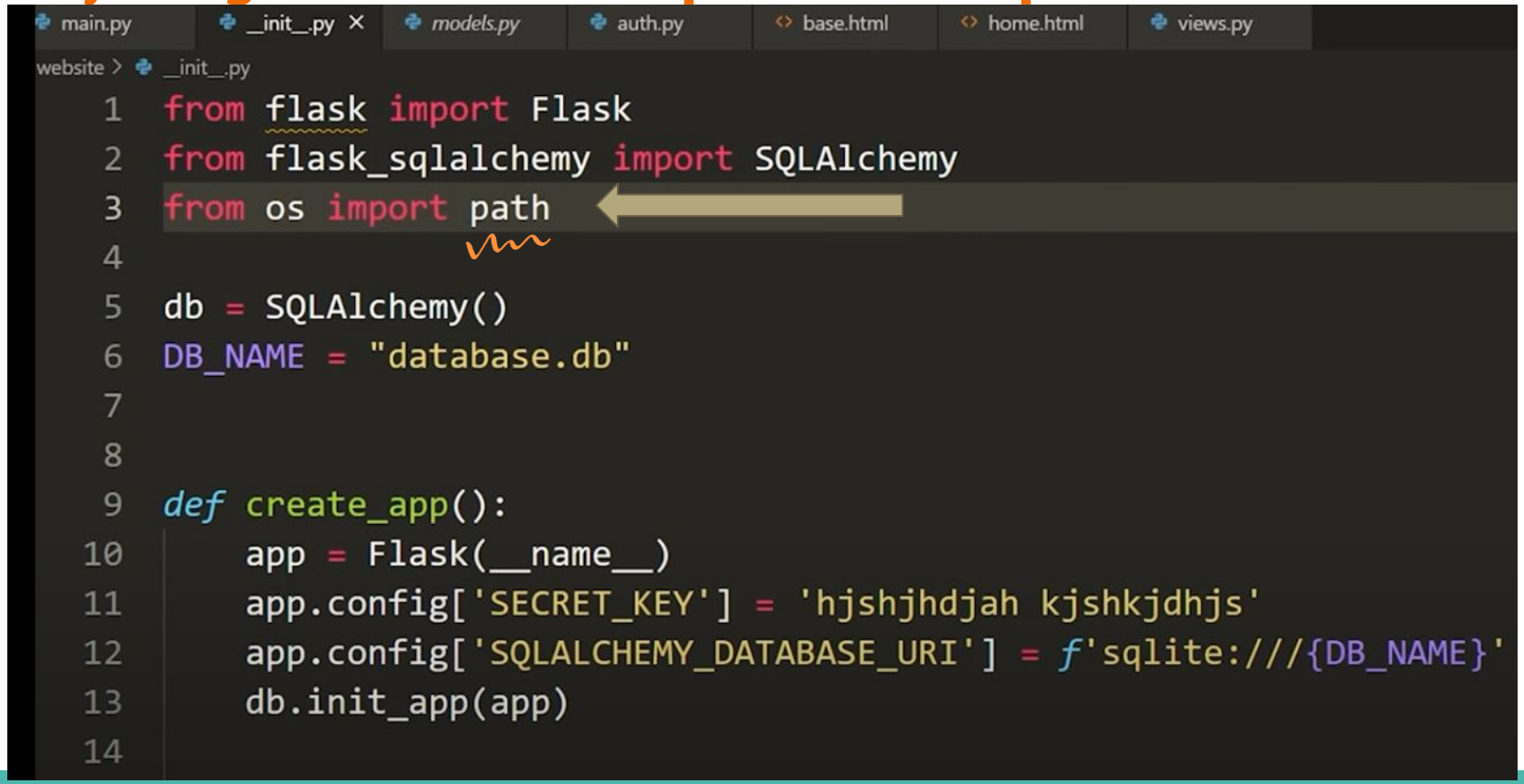
Linking the 2 tables together:



Importing the models into our __init__.py file



To check whether or not the DB was created before anything, we need to import the OS path:



Importing the tables from the DB models:

site > __init__.py

```
15      from .views import views
16      from .auth import auth
17
18      app.register_blueprint(views, url_prefix='/')
19      app.register_blueprint(auth, url_prefix='/')
20
21      from .models import User, Note
22
23      create_database(app)
24
25      return app
26
```

Now, our DB is created and linked, let's work on signing up the users.

On our auth.py file, we need to configure:

website > auth.py

```
1  from flask import Blueprint, render_template, request, flash
2  from .models import User
3
4  auth = Blueprint('auth', __name__)
5
6
7  @auth.route('/login', methods=['GET', 'POST'])
8  def login():
9      return render_template("login.html", boolean=True)
10
11
12 @auth.route('/logout')
13 def logout():
14     return "<p>Logout</p>"
15
16
```

We need to import a few things that will help us hash our password:

```
from werkzeug.security import generate_password_hash, check_password_hash
```

hashing is a way of encryption that makes the pass has no inverse

Adding our new users to our auth.py file

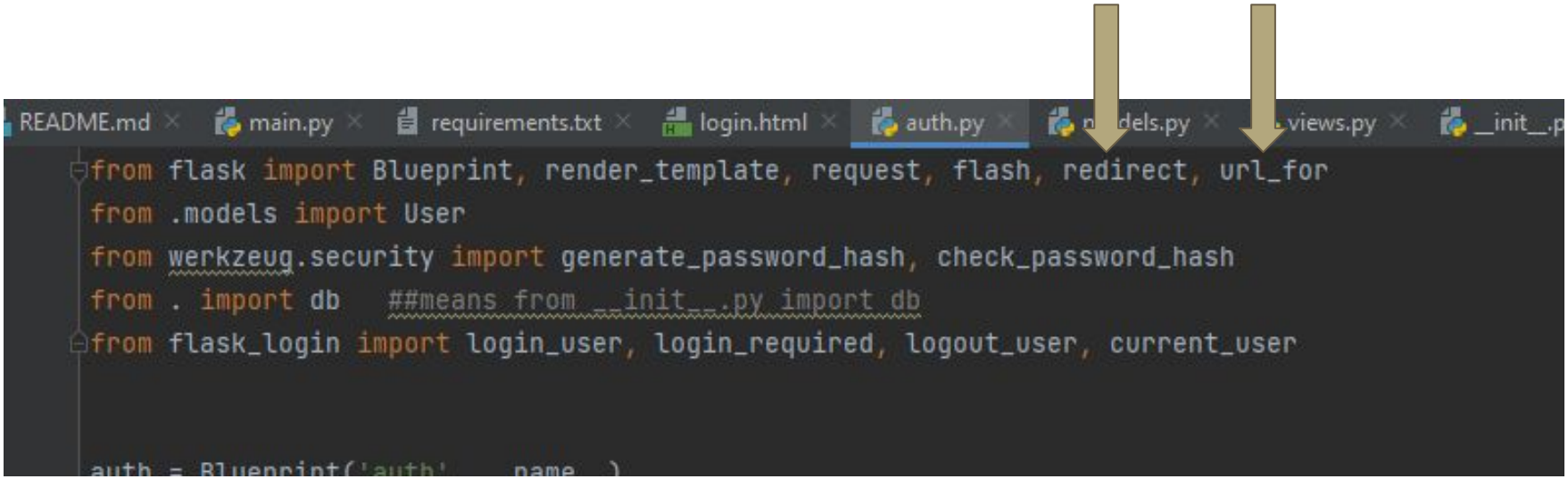
```
    if user:
        flash('Email already exists.', category='error')
    elif len(email) < 4:
        flash('Email must be greater than 3 characters.', category='error')
    elif len(first_name) < 2:
        flash('First name must be greater than 1 character.', category='error')
    elif password1 != password2:
        flash('Passwords don\'t match.', category='error')
    elif len(password1) < 7:
        flash('Password must be at least 7 characters.', category='error')
    else:
        new_user = User(email=email, first_name=first_name, password=generate_password_hash(
            password1, method='sha256'))
        db.session.add(new_user)
        db.session.commit()
        login_user(new_user, remember=True)
        flash('Account created!', category='success')
        return redirect(url_for('views.home'))
```

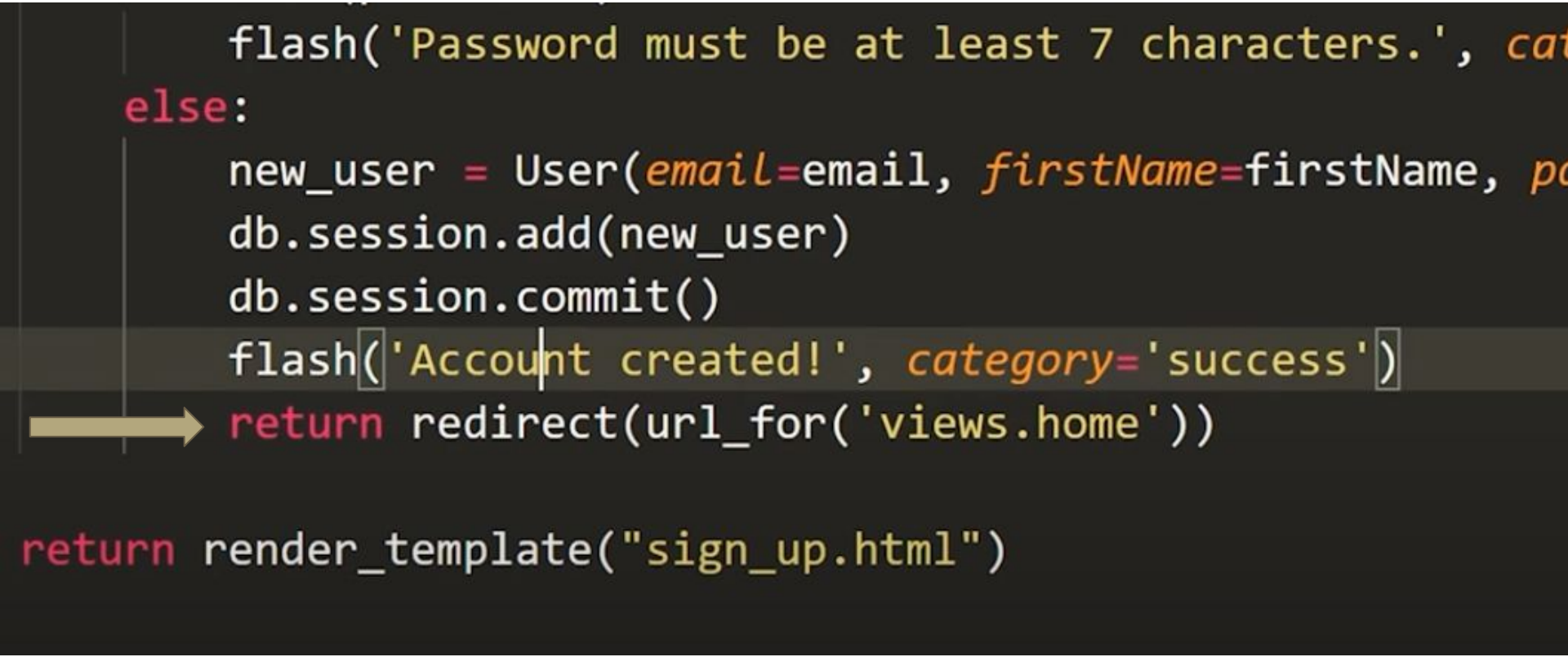
hashing algorithm

update DB

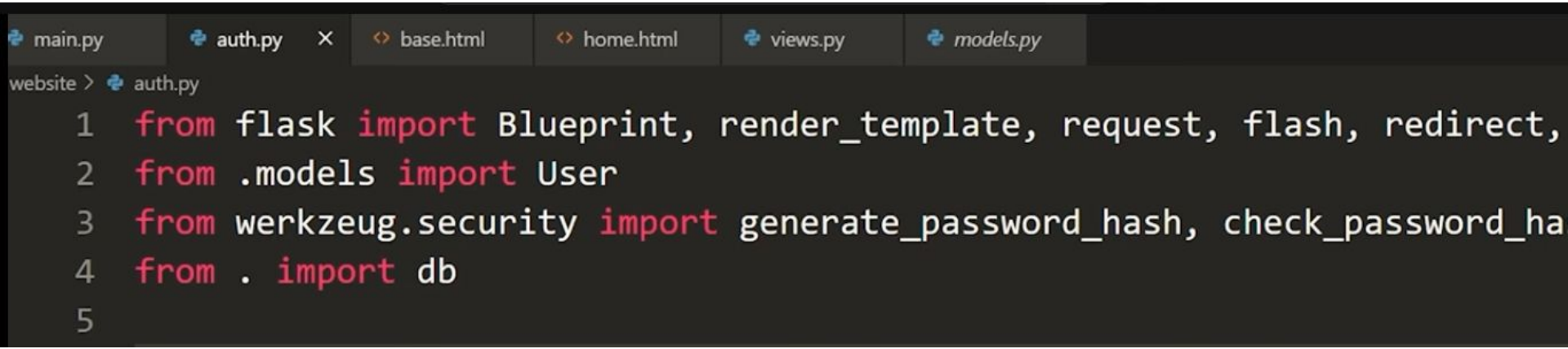
Redirecting the user into the homepage after they sign up

After a new user is added, we should redirect them to the to home page:




flash('Password must be at least 7 characters.', cat
else:
new_user = User(email=email, firstName=firstName, pa
db.session.add(new_user)
db.session.commit()
flash('Account created!', category='success')
return redirect(url_for('views.home'))
return render_template("sign_up.html")

Don’t forget about importing the db to your auth.py file



Handling the sign-in inside our auth.py file

↳ Try it yourself!

Def login():
If request.method=='post':
email =request.form.get('email')
password =request.form.get('password')

```
user = User.query.filter_by(email=email).first()
if user:
    if check_password_hash(user.password, password):
        flash('Logged in successfully!', category='success')
        login_user(user, remember=True)
        return redirect(url_for('views.home'))
    else:
        flash('Incorrect password, try again.', category='error')
else:
    flash('Email does not exist.', category='error')
```

The hashed pass

return only the 1st result if we have many [we should not]

Browser will remember the user until they clear their browser.

Libraries that will help us safe the user sessions:

README.md | main.py | requirements.txt | login.html | auth.py | models.py | views.py | __init__.py | base.html

```
1  from flask import Blueprint, render_template, request, flash, redirect, url_for
2  from .models import User
3  from werkzeug.security import generate_password_hash, check_password_hash
4  from . import db   ##means from __init__.py import db
5  from flask_login import login_user, login_required, logout_user, current_user
6
```

To ensure that no user can access the homepage unless they're logged in.

Redirecting the users after they log out:

```
30
31  @auth.route('/logout')
32  @login_required
33  def logout():
34      logout_user()
35      return redirect(url_for('auth.login'))
36
```

→ makes sure that the user logged out only if they're logged in.

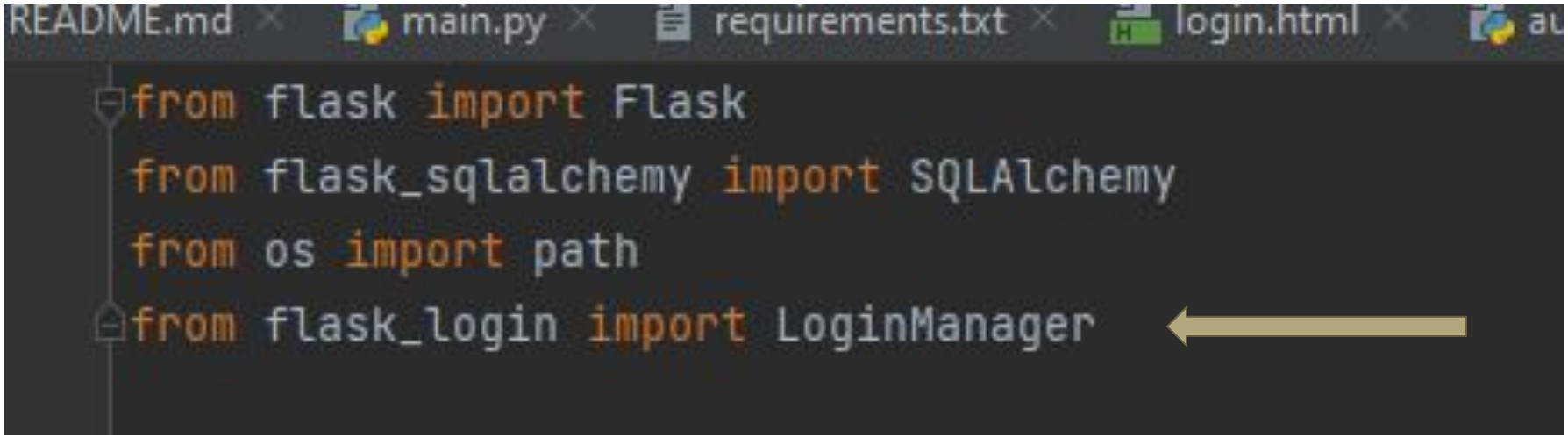
Require the users to be logged in before they access the homepage: (views.py)

website > views.py

```python
from flask import Blueprint, render_template
from flask_login import login_required, current_user

views = Blueprint('views', __name__)


@views.route('/')
@login_required
def home():
    return render_template("home.html")

```

You cannot go to the homepage until you're logged in

Flask Login Manager: (__ init.py __)



Where should the flask take us if we're not logged in?

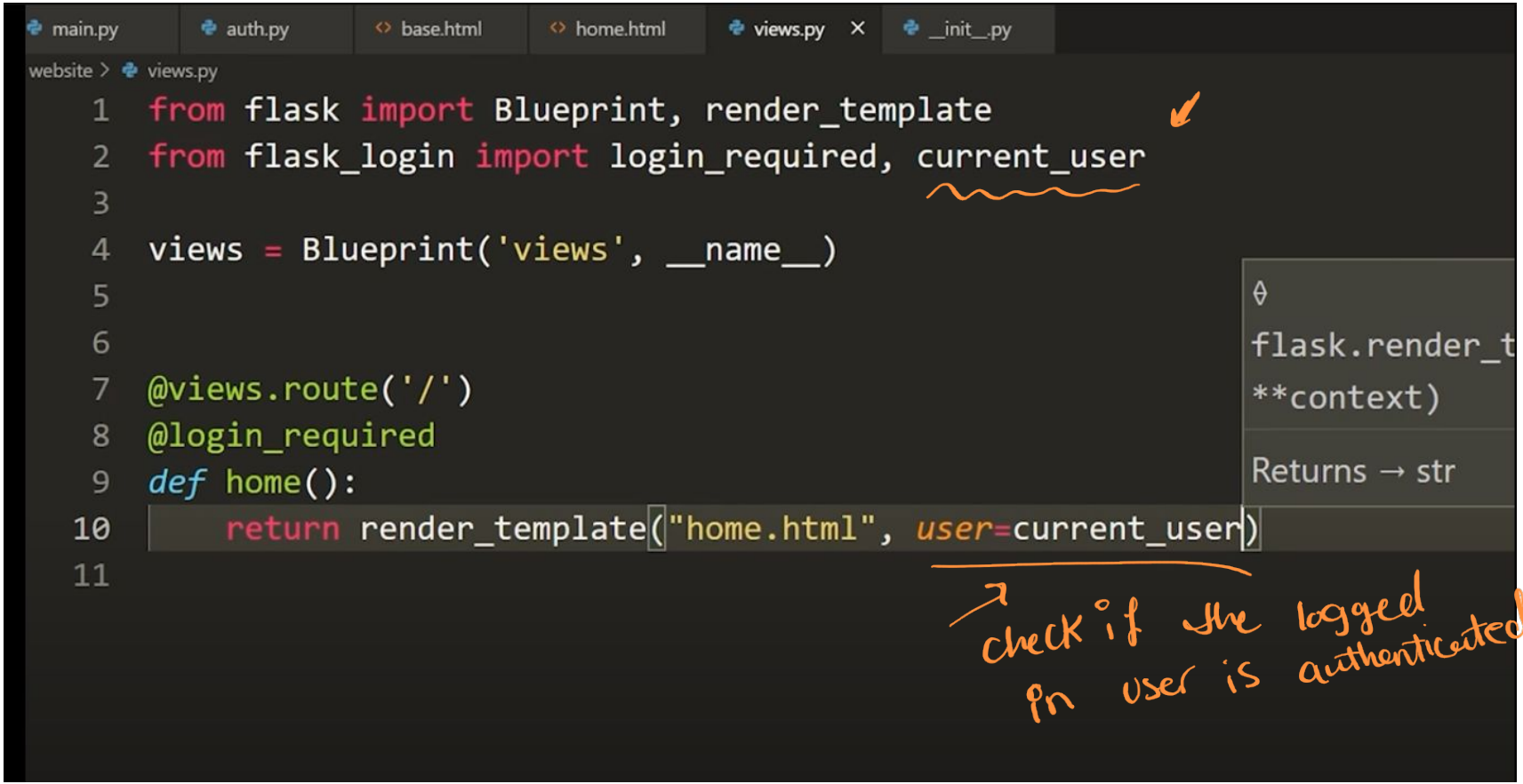
Back to the login screen: (still on __init__.py)

How it Works

You will need to provide a **user_loader** callback. This callback is used to reload the user object from the user ID stored in the session. It should take the **str** ID of a user, and return the corresponding user object. For example:

```
@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)
```

On your views.py, check if there's a logged in user



We need to ensure that even the html nav-bar is only shown to the logged in users: [back to base.html]

```
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbar">
    <div class="navbar-nav">
      {% if user.is_authenticated %}
      <a class="nav-item nav-link" id="home" href="/">Home</a>
      <a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
      {% else %}
      <a class="nav-item nav-link" id="login" href="/login">Login</a>
      <a class="nav-item nav-link" id="signUp" href="/sign-up">Sign Up</a>
      {% endif %}
    </div>
  </div>
</nav>
```

In our auth.py file, we need to modify the login method to check the current user:

```
                flash('Logged in successfully!', category='success')
                login_user(user, remember=True)
                return redirect(url_for('views.home'))
            else:
                flash('Incorrect password, try again.', category='error
        else:
            flash('Email does not exist.', category='error')

    return render_template("login.html", user=current_user)
```

Same for signup:

```
    return render_template("sign_up.html", user=current_user)
```

Adding & Displaying notes: [home.html]

```
{% extends "base.html" %} {% block title %}Home{% endblock %} {% block content
%}
<h1 align="center">Notes</h1>
<ul class="list-group list-group-flush" id="notes">
  {% for note in user.notes %}
  <li class="list-group-item">{{ note.data }}</li>
  {% endfor %}
</ul>
<form method="POST">
  <textarea name="note" id="note" class="form-control"></textarea>
  <div align="center">
    <button type="submit" class="btn btn-primary">Add Note</button>
  </div>
</form>
{% endblock %}
```

Some bootstrap code

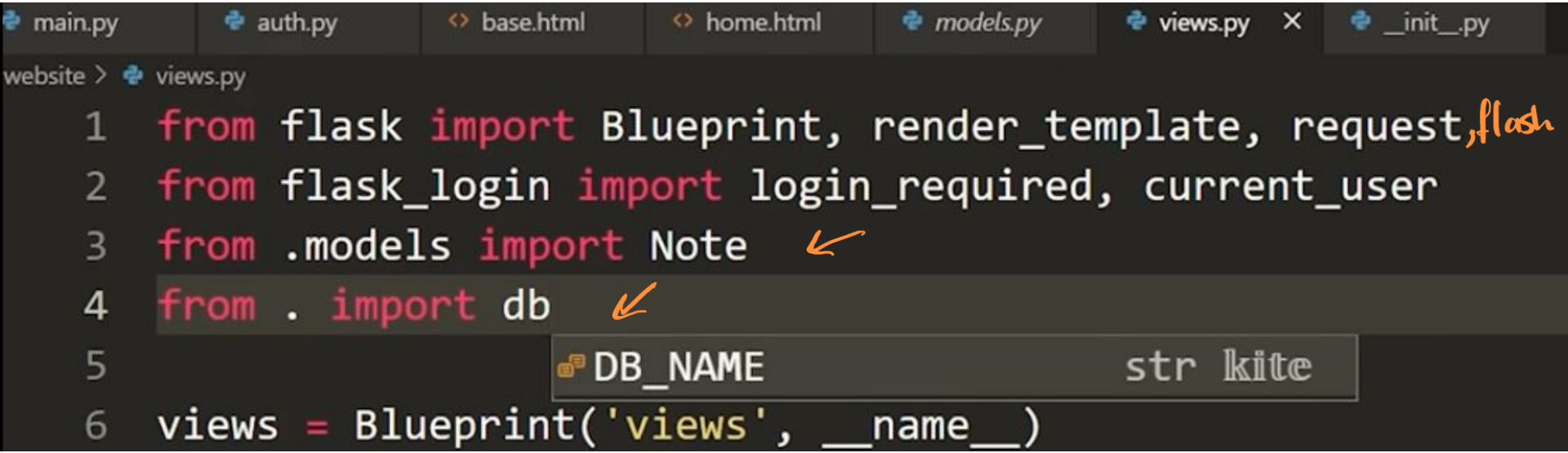
Ensuring that the notes are related to the user

To get the notes from the user.

Views.py

```python
@views.route('/', methods=['GET', 'POST'])
@login_required
def home():
    if request.method == 'POST':
        note = request.form.get('note')#Gets the note from the HTML

        if len(note) < 1:
            flash('Note is too short!', category='error')
        else:
            new_note = Note(data=note, user_id=current_user.id)  #providing the schema for the note
            db.session.add(new_note) #adding the note to the database
            db.session.commit()      # update
            flash('Note added!', category='success')

    return render_template("home.html", user=current_user)
```

Don’t forget about importing the DB into the views.py to link both the front and back end of the app:



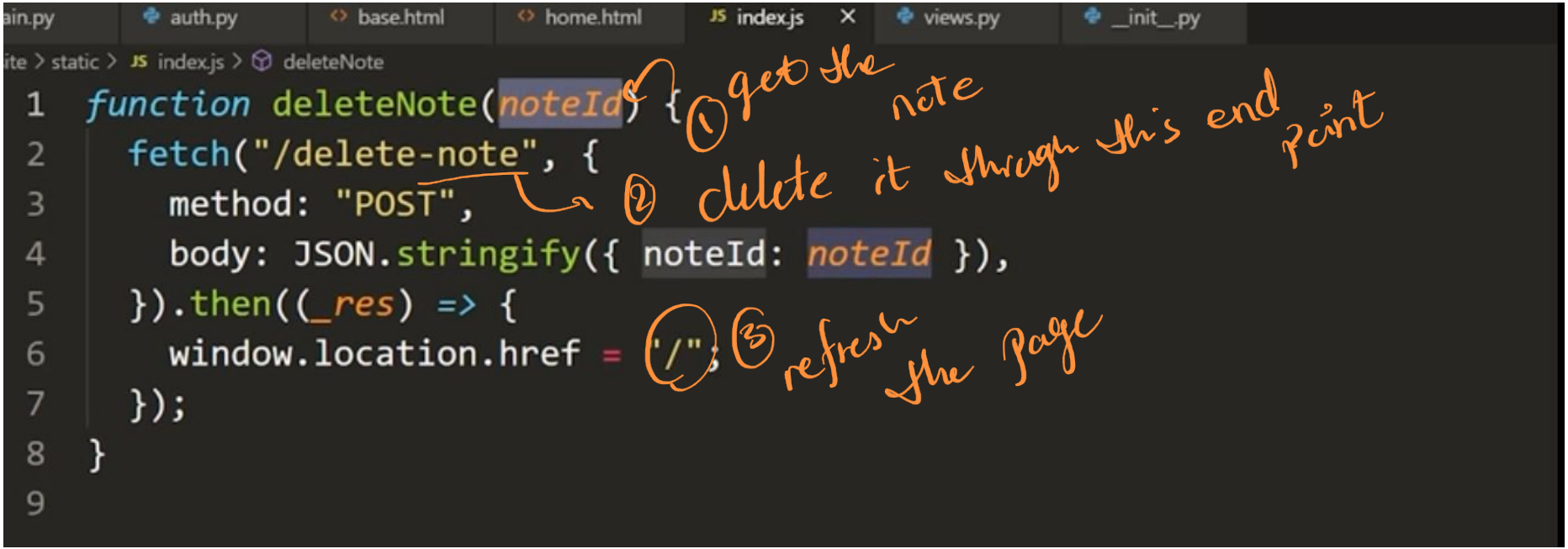
Deleting the note from home.html

```
<h1 align="center">Notes</h1>
<ul class="list-group list-group-flush" id="notes">
  {% for note in user.notes %}
  <li class="list-group-item">
    {{ note.data }}
    <button type="button" class="close" onClick="deleteNote({{ note.id }})">
      <span aria-hidden="true">×</span>
    </button>
  </li>
  {% endfor %}
</ul>
```

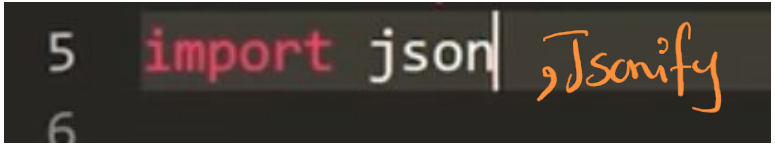
Js

function that will send out the data to the back end

We'll use our index.js file to handle the deletion function:



Deletion inside our views.py file:




```python
@views.route('/delete-note', methods=['POST'])
def delete_note():
    note = json.loads(request.data) # this function expects a JSON from the INDEX.js file
    noteId = note['noteId']
    note = Note.query.get(noteId)
    if note:
        if note.user_id == current_user.id:
            db.session.delete(note)
            db.session.commit()

    return jsonify({})
```

→ if the note does exist