Overview of Web Applications with Flask

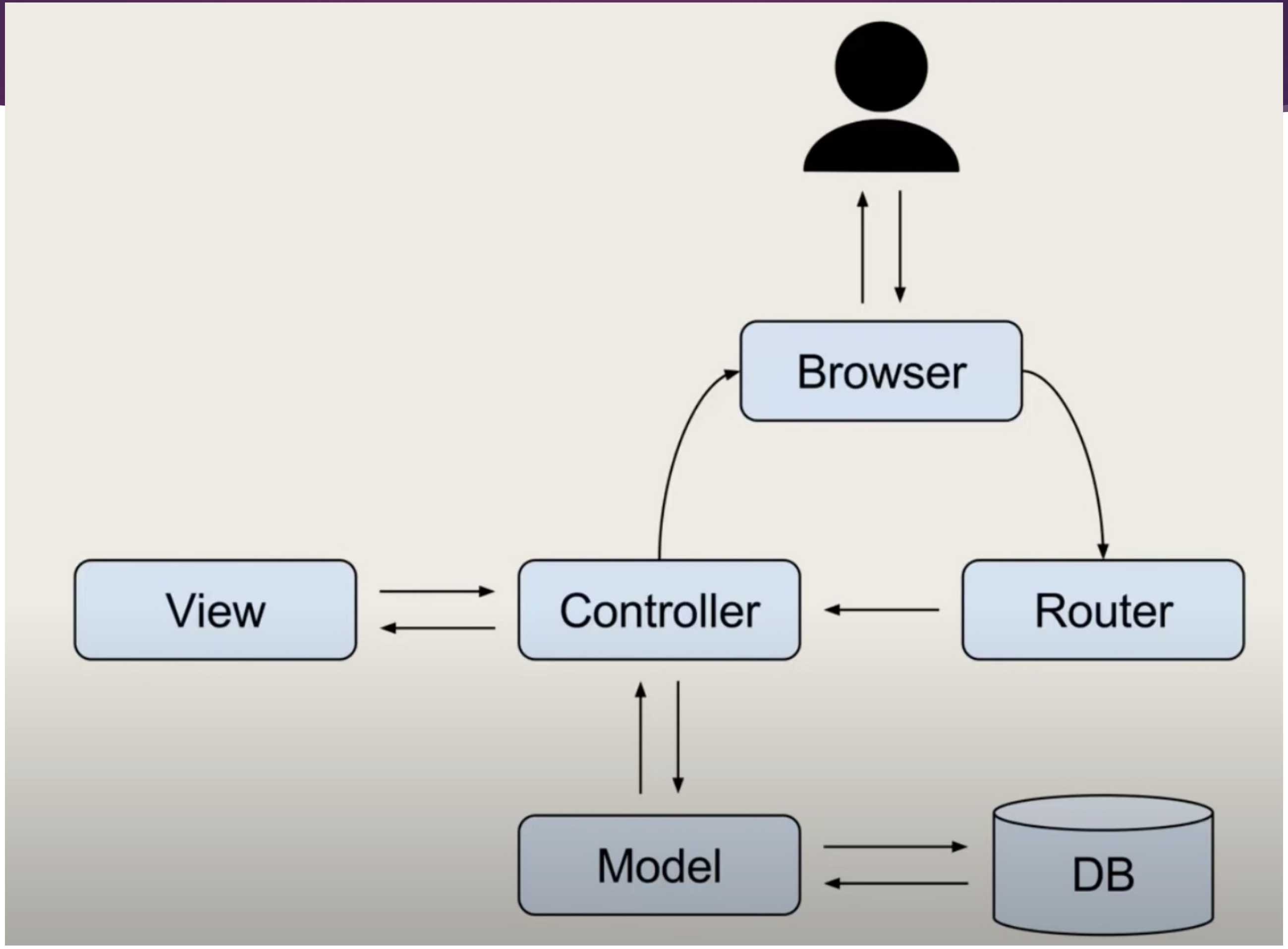
Quick guide on how to install Flask

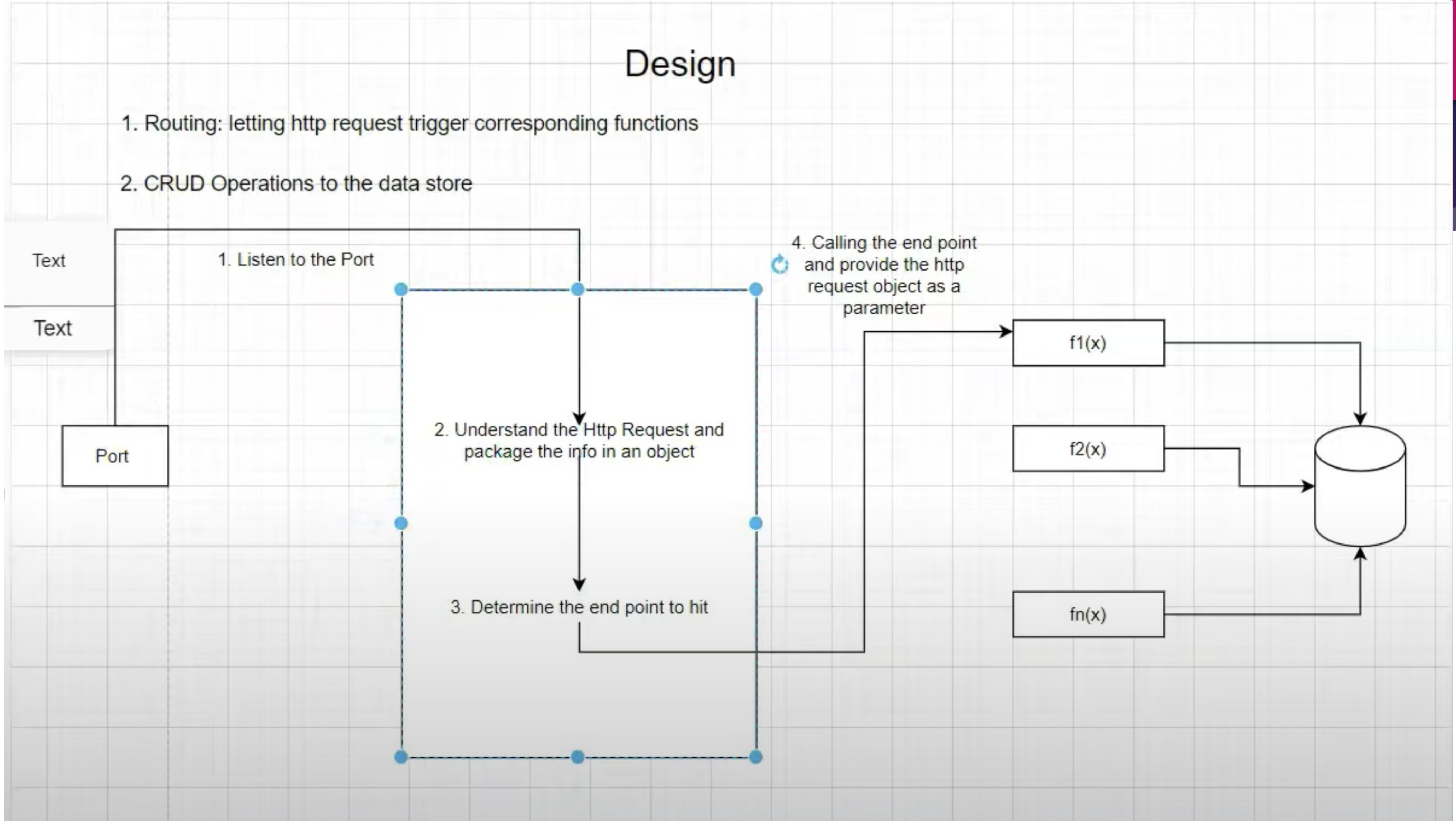
- Make sure you have python and MySQL installed
- Go to the terminal or command prompt (for windows, make sure python is added to your path environment variables) and run:
  - pip install flask
  - pip install pymysql (used to connect to database)
- If you don’t have pip you can get it in the following link:
  - https://pypi.python.org/pypi/pip
  - Run: python get-pip.py
- More information:
  - http://flask.pocoo.org/

Flask

- Flask is a micro-framework for python
- Easy to code
- Flexible
- Good documentation
- RESTful

Web Applications





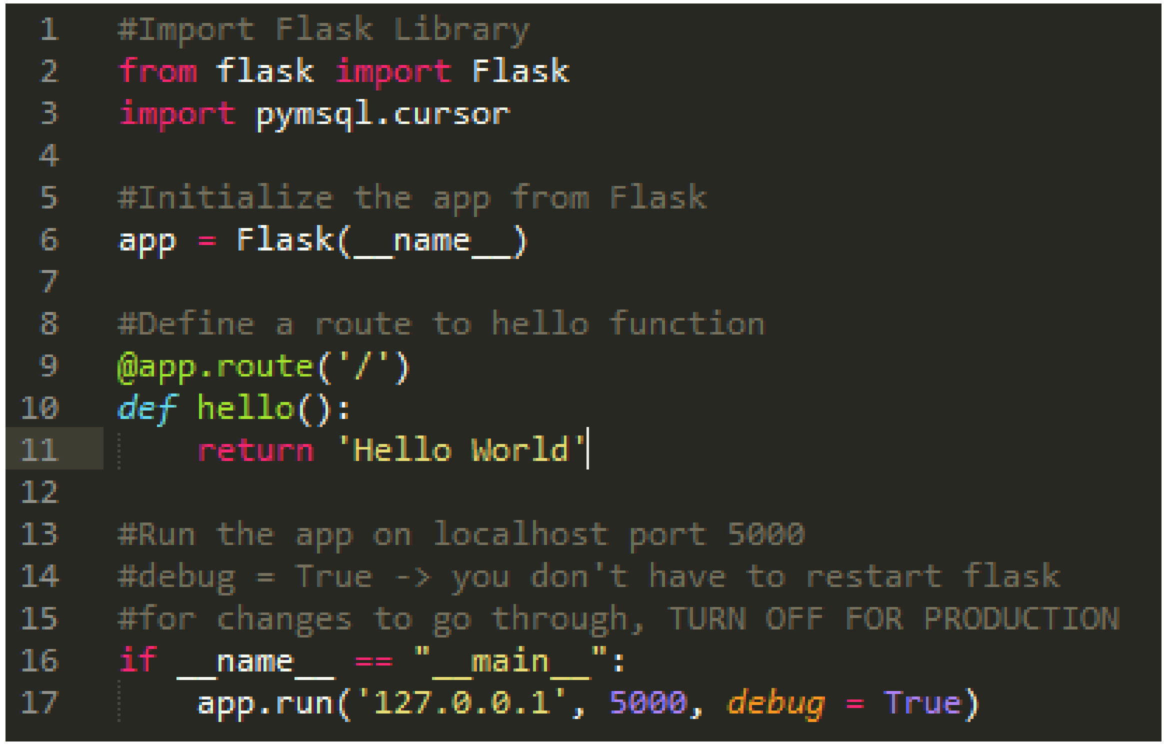
Design

1. Routing: letting http request trigger corresponding functions

2. CRUD Operations to the data store

Text
Text
Port
1. Listen to the Port
2. Understand the Http Request and package the info in an object
3. Determine the end point to hit
4. Calling the end point and provide the http request object as a parameter
f1(x)
f2(x)
fn(x)

Basic Flask App

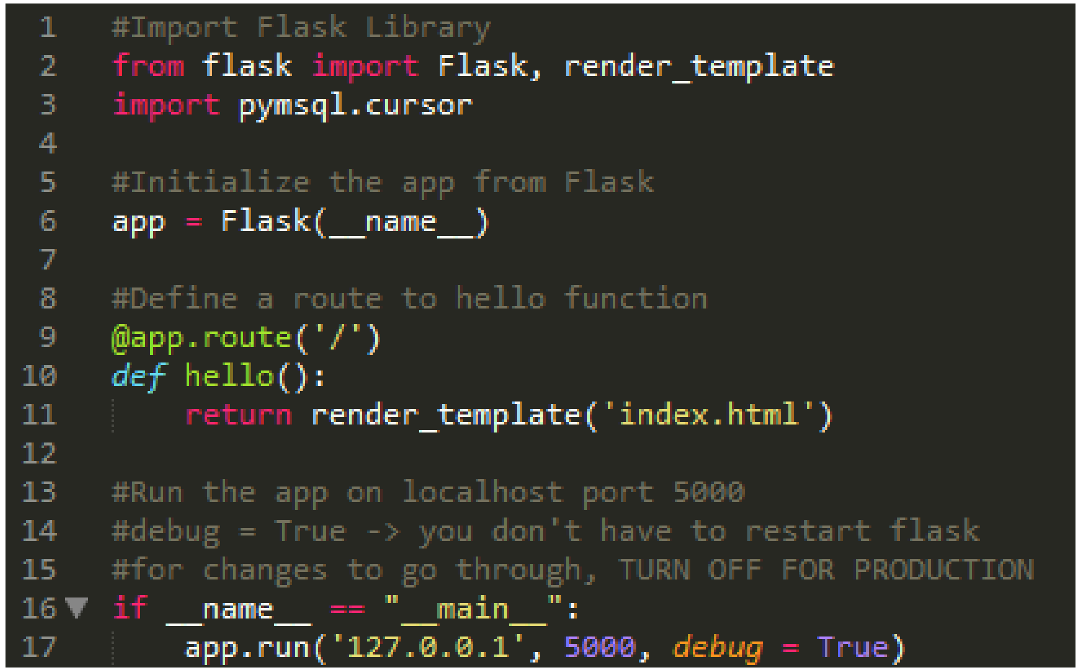


```python
#Import Flask Library
from flask import Flask
import pymsql.cursor

#Initialize the app from Flask
app = Flask(__name__)

#Define a route to hello function
@app.route('/')
def hello():
    return 'Hello World'

#Run the app on localhost port 5000
#debug = True -> you don't have to restart flask
#for changes to go through, TURN OFF FOR PRODUCTION
if __name__ == "__main__":
    app.run('127.0.0.1', 5000, debug = True)
```

@app.route('/') is saying that if the user goes to the root level of your application (e.g 127.0.0.1:5000/) the hello function will be called

If you go to localhost:5000
You should see Hello World being printed out

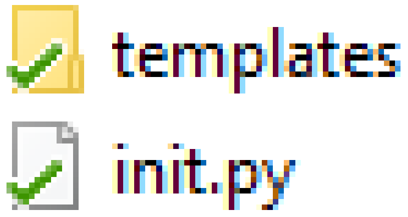
Rendering html files



```
#Import Flask Library
from flask import Flask, render_template
import pymsql.cursor

#Initialize the app from Flask
app = Flask(__name__)

#Define a route to hello function
@app.route('/')
def hello():
    return render_template('index.html')

#Run the app on localhost port 5000
#debug = True -> you don't have to restart flask
#for changes to go through, TURN OFF FOR PRODUCTION
if __name__ == "__main__":
    app.run('127.0.0.1', 5000, debug = True)
```

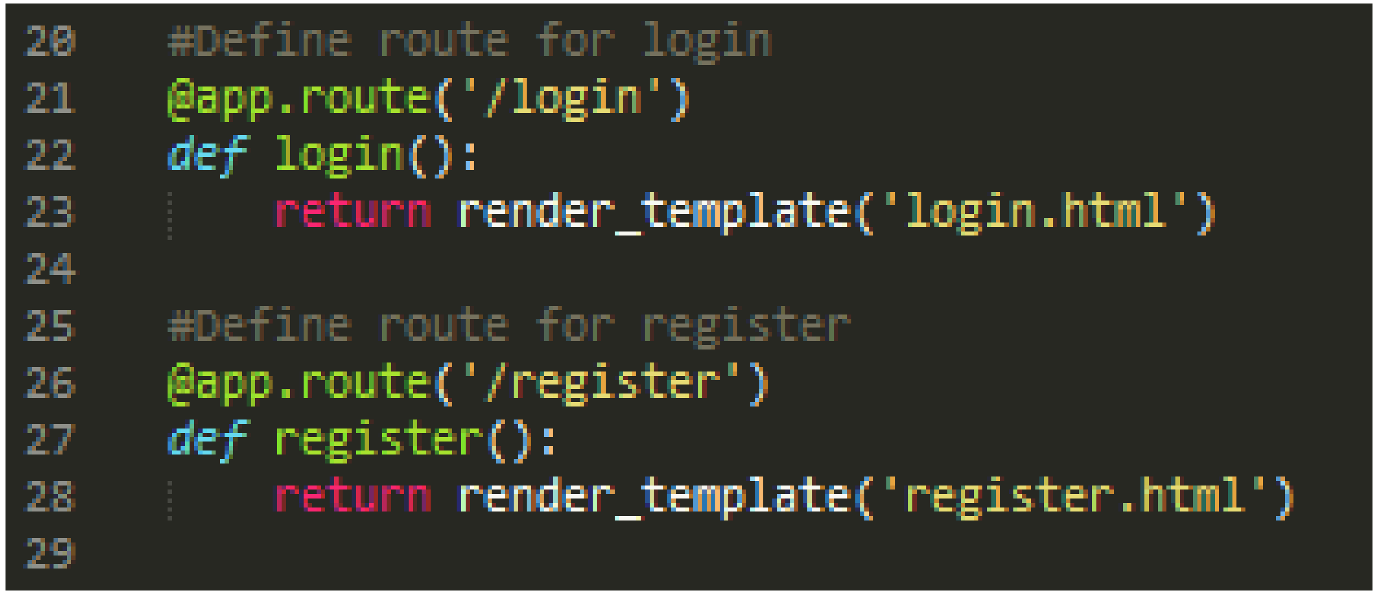
If you want to send back hml files, import render_template and use it to send back html files.

***The html files should be in a directory called templates



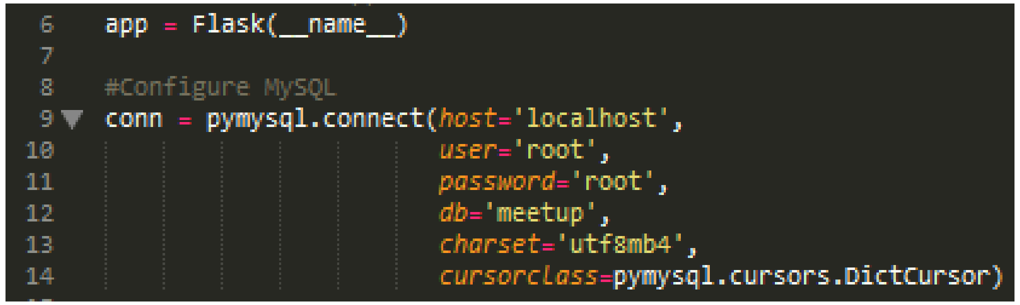
templates
init.py

Creating flask code to serve the files



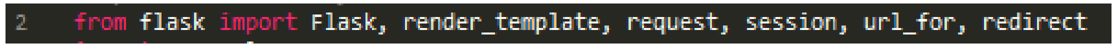
```
#Define route for login
@app.route('/login')
def login():
    return render_template('login.html')

#Define route for register
@app.route('/register')
def register():
    return render_template('register.html')

```

Configuring to connect to MySQL



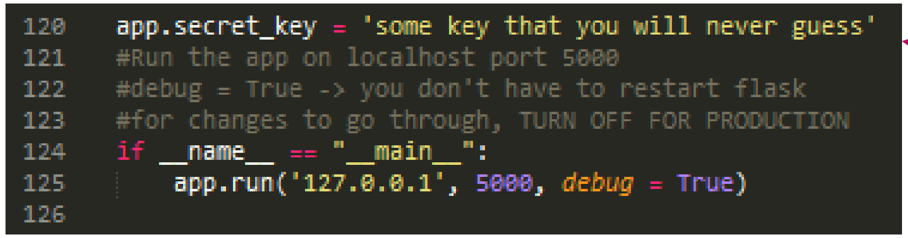
```
app = Flask(__name__)

#Configure MySQL
conn = pymysql.connect(host='localhost',
                       user='root',
                       password='root',
                       db='meetup',
                       charset='utf8mb4',
                       cursorclass=pymysql.cursors.DictCursor)
```

This sets the configuration to connect to your MySQL database



```
from flask import Flask, render_template, request, session, url_for, redirect
```

Import session, requests, url_for, and redirect for the next part



```
app.secret_key = 'some key that you will never guess'
#Run the app on localhost port 5000
#debug = True -> you don't have to restart flask
#for changes to go through, TURN OFF FOR PRODUCTION
if __name__ == "__main__":
    app.run('127.0.0.1', 5000, debug = True)
```

In order to use session, you must create a secret key, which is a random key used to encrypt your cookies.

Cookies are pieces of information that are saved in the user's browser. Every time that a permitted user navigates to a website, website will create and save data inside this file, which is unique per URL

Session

- A session object is a dictionary object:

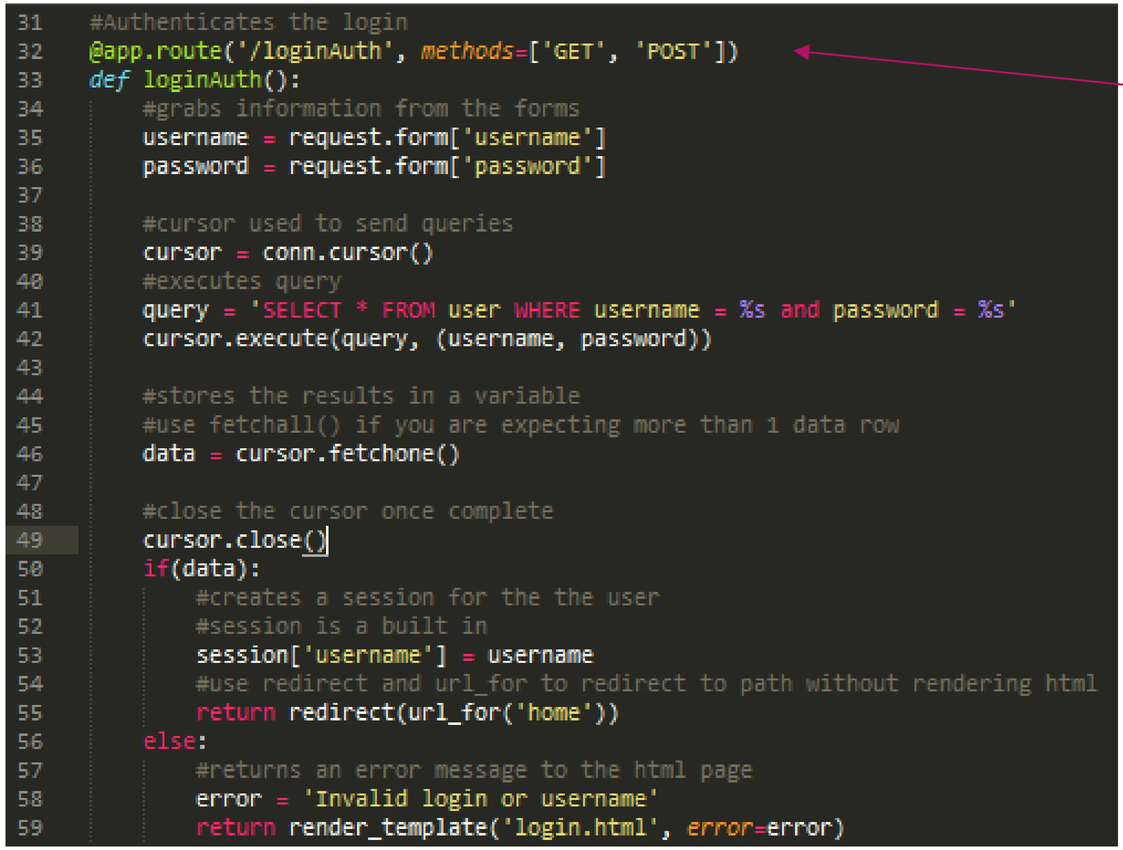
```
session['username'] = 'admin'
```

- To release a session variable, use the pop() method.

```
session.pop('username', None)
```

```
@app.route('/')
def index():
    if 'username' in session:
        username = session['username']
            return redirect(url_for('home'))
    return render_template('login.html')
```

Authenticating the login



```python
#Authenticates the login
@app.route('/loginAuth', methods=['GET', 'POST'])
def loginAuth():
    #grabs information from the forms
    username = request.form['username']
    password = request.form['password']

    #cursor used to send queries
    cursor = conn.cursor()
    #executes query
    query = 'SELECT * FROM user WHERE username = %s and password = %s'
    cursor.execute(query, (username, password))

    #stores the results in a variable
    #use fetchall() if you are expecting more than 1 data row
    data = cursor.fetchone()

    #close the cursor once complete
    cursor.close()
    if(data):
        #creates a session for the the user
        #session is a built in
        session['username'] = username
        #use redirect and url_for to redirect to path without rendering html
        return redirect(url_for('home'))
    else:
        #returns an error message to the html page
        error = 'Invalid login or username'
        return render_template('login.html', error=error)
```

Need to add this for GET and POST requests to work

We can send a query to the database by calling the execute method of cursor.

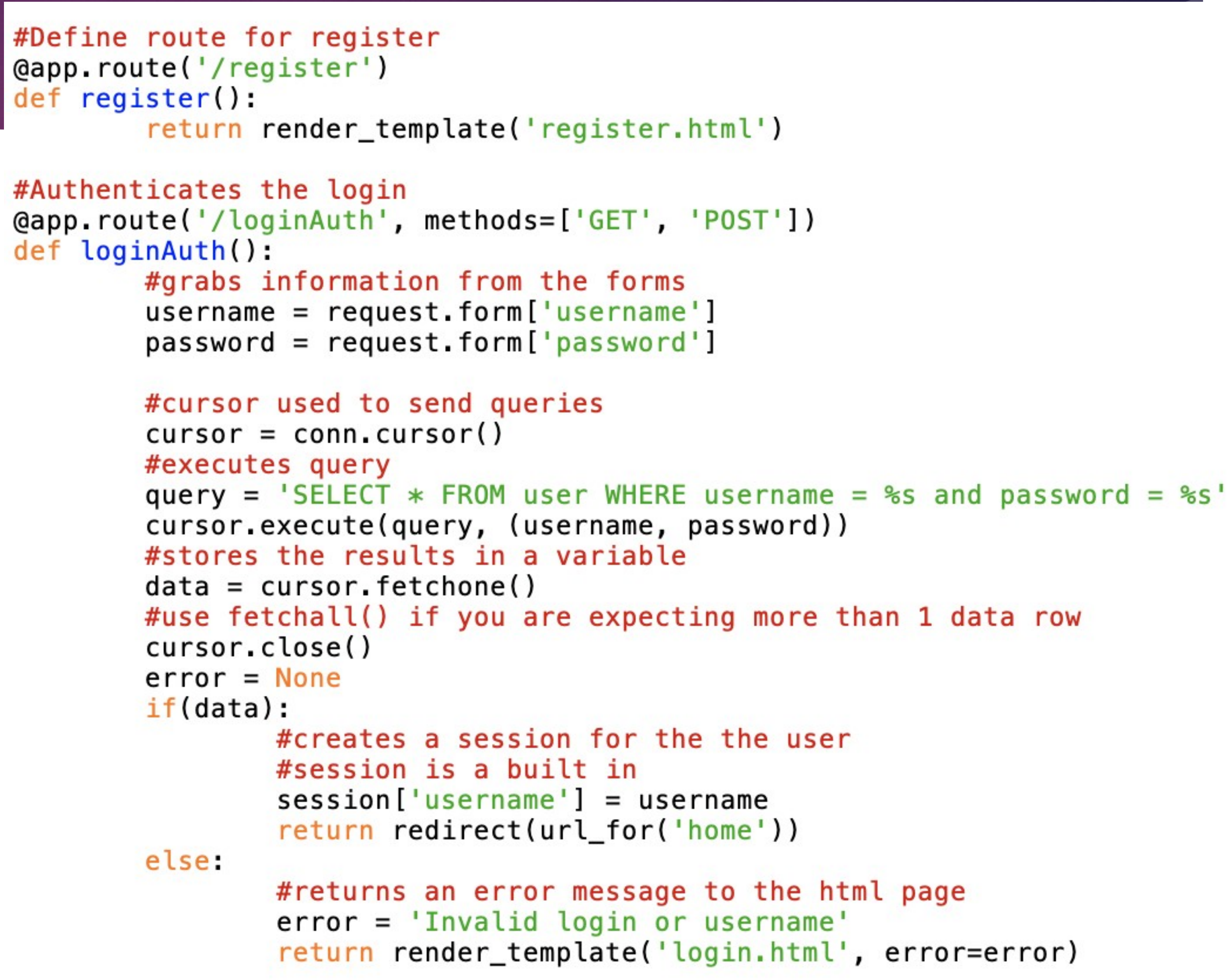
- Cursor is just an object that is used to interface with the database.
- If the query is successful, call fetchone() to get a single data row
- or fetchall() to get multiple rows.

If the login is successful, it will redirect to the home page

Flask HTTP Methods

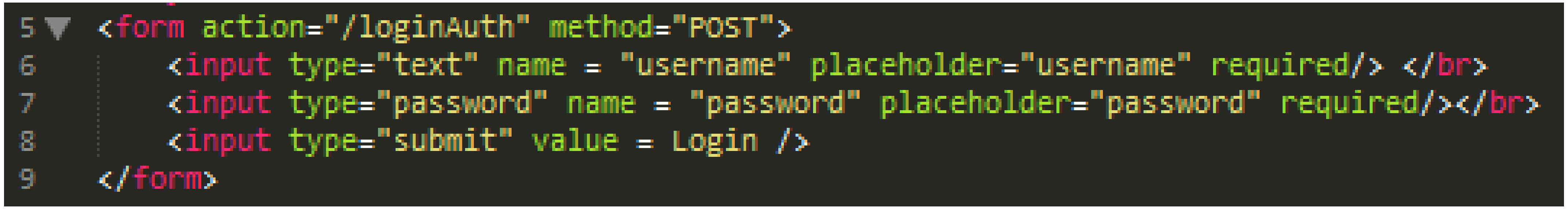
- GET: a GET message is send and the server returns data
  - By default, the Flask route responds to GET requests.

- POST: used to send HTML form data to the server



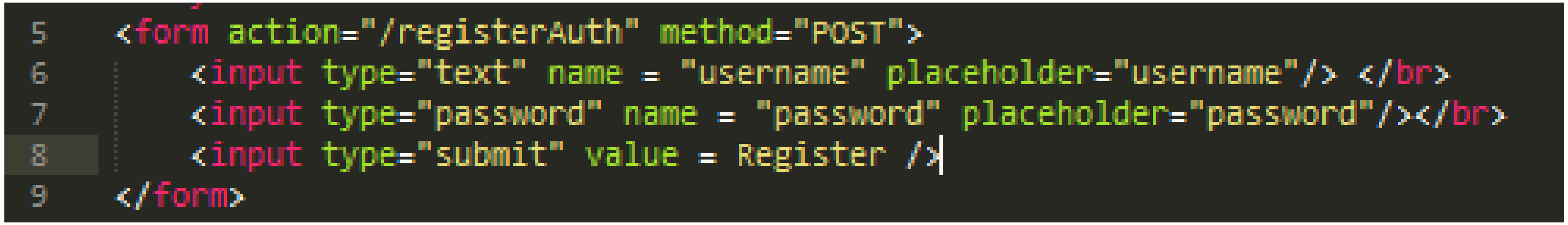
```
#Define route for register
@app.route('/register')
def register():
        return render_template('register.html')

#Authenticates the login
@app.route('/loginAuth', methods=['GET', 'POST'])
def loginAuth():
        #grabs information from the forms
        username = request.form['username']
        password = request.form['password']

        #cursor used to send queries
        cursor = conn.cursor()
        #executes query
        query = 'SELECT * FROM user WHERE username = %s and password = %s'
        cursor.execute(query, (username, password))
        #stores the results in a variable
        data = cursor.fetchone()
        #use fetchall() if you are expecting more than 1 data row
        cursor.close()
        error = None
        if(data):
                #creates a session for the the user
                #session is a built in
                session['username'] = username
                return redirect(url_for('home'))
        else:
                #returns an error message to the html page
                error = 'Invalid login or username'
                return render_template('login.html', error=error)
```

Creating a simple login/register form



```
<form action="/loginAuth" method="POST">
    <input type="text" name = "username" placeholder="username" required/> </br>
    <input type="password" name = "password" placeholder="password" required/></br>
    <input type="submit" value = Login />
</form>
```

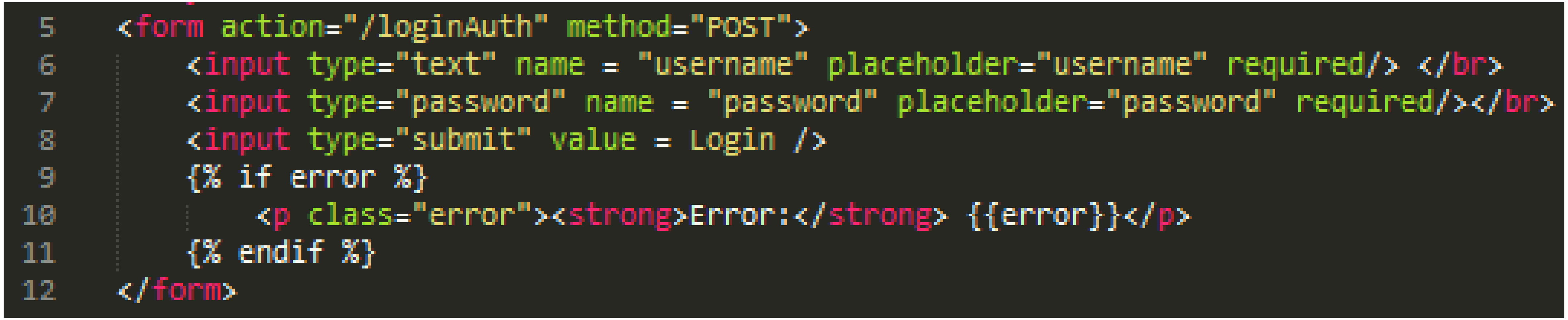
This would be in a login.html file, this makes a POST request to /loginAuth, we’ll talk more about /loginAuth later



```
<form action="/registerAuth" method="POST">
    <input type="text" name = "username" placeholder="username"/> </br>
    <input type="password" name = "password" placeholder="password"/></br>
    <input type="submit" value = Register />
</form>
```

This would be in a register.html file, this makes a POST request to /registerAuth, we’ll talk more about /registerAuth later

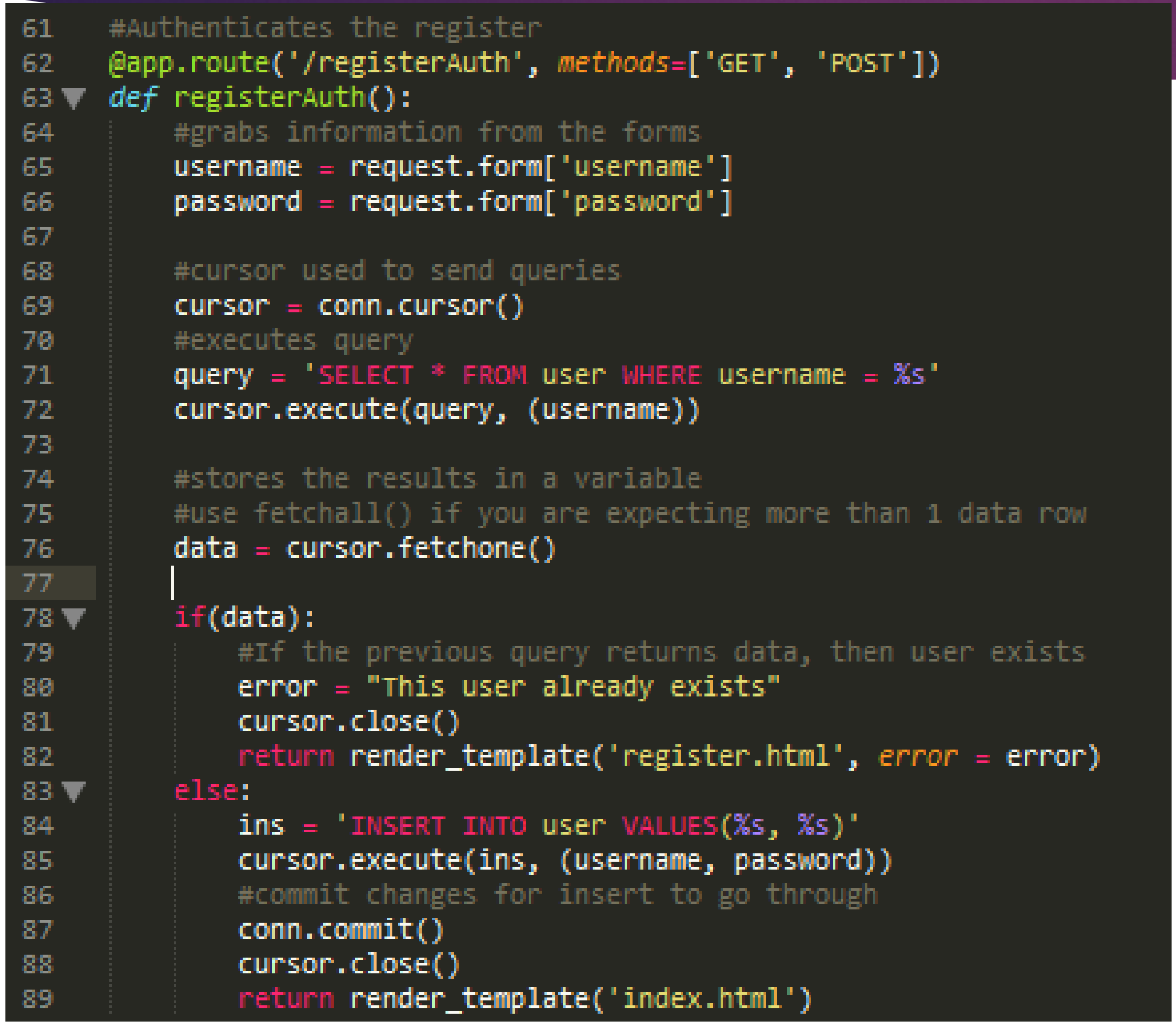
Updating the login form



```
<form action="/loginAuth" method="POST">
    <input type="text" name = "username" placeholder="username" required/> </br>
    <input type="password" name = "password" placeholder="password" required/></br>
    <input type="submit" value = Login />
    {% if error %}
        <p class="error"><strong>Error:</strong> {{error}}</p>
    {% endif %}
</form>
```

In the previous slide, we passed in an extra argument to render_template:
error = error. Error corresponds to the error that was passed in by the render_template call.
Flask uses jinja templating and we can pass variables from flask to the html page using this.
If there was an error message, we passed it in and the message is displayed. {{}} denotes a variable.

Authenticating register



```
61 #Authenticates the register
62 @app.route('/registerAuth', methods=['GET', 'POST'])
63 def registerAuth():
64     #grabs information from the forms
65     username = request.form['username']
66     password = request.form['password']
67 
68     #cursor used to send queries
69     cursor = conn.cursor()
70     #executes query
71     query = 'SELECT * FROM user WHERE username = %s'
72     cursor.execute(query, (username))
73 
74     #stores the results in a variable
75     #use fetchall() if you are expecting more than 1 data row
76     data = cursor.fetchone()
77 
78     if(data):
79         #If the previous query returns data, then user exists
80         error = "This user already exists"
81         cursor.close()
82         return render_template('register.html', error = error)
83     else:
84         ins = 'INSERT INTO user VALUES(%s, %s)'
85         cursor.execute(ins, (username, password))
86         #commit changes for insert to go through
87         conn.commit()
88         cursor.close()
89         return render_template('index.html')
```

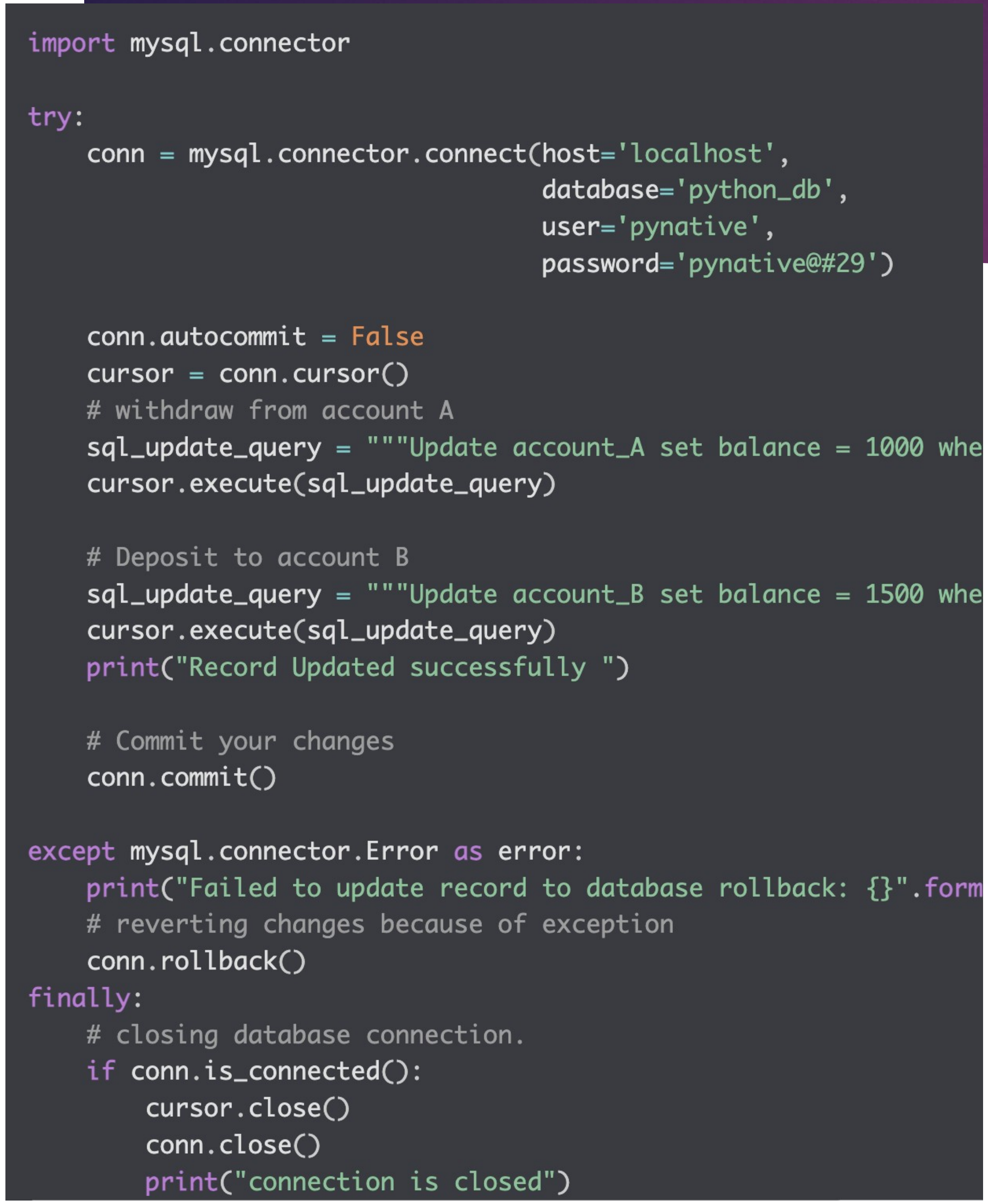
NOTE: You don’t want to store your passwords in your database as plain text, you probably want to hash it

Notice in line 87, we have to commit if the query modifies the database

- By default, Connector/Python does not autocommit, it is important to call mysqlconnection.commit() after every transaction that modifies data for tables

(https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-commit.html)

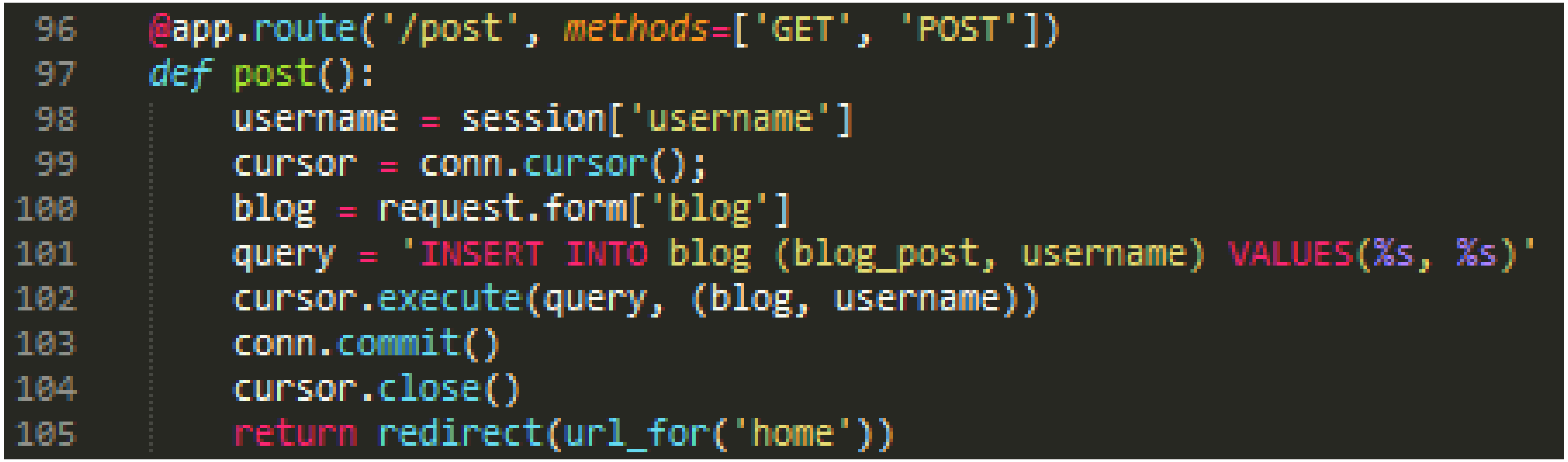
More on mysql.connector



```
import mysql.connector

try:
    conn = mysql.connector.connect(host='localhost',
                                   database='python_db',
                                   user='pynative',
                                   password='pynative@#29')

    conn.autocommit = False
    cursor = conn.cursor()
    # withdraw from account A
    sql_update_query = """Update account_A set balance = 1000 whe
    cursor.execute(sql_update_query)

    # Deposit to account B
    sql_update_query = """Update account_B set balance = 1500 whe
    cursor.execute(sql_update_query)
    print("Record Updated successfully ")

    # Commit your changes
    conn.commit()

except mysql.connector.Error as error:
    print("Failed to update record to database rollback: {}".form
    # reverting changes because of exception
    conn.rollback()
finally:
    # closing database connection.
    if conn.is_connected():
        cursor.close()
        conn.close()
        print("connection is closed")
```

1. After a successful MySQL connection, we set auto-commit to False.
2. Prepare two update SQL queries as a part of a single transaction to deposit money to account B from account A.
3. We execute both the queries one by one using a cursor.execute() method
4. After successful execution of both the queires, we committee our changes to the database using a con.commit()
5. In case of an exception or failure of one of the queries, we can revert our changes using a conn.rollback()
6. We placed all our code in the try-except block to catch the database exceptions that may occur during the process

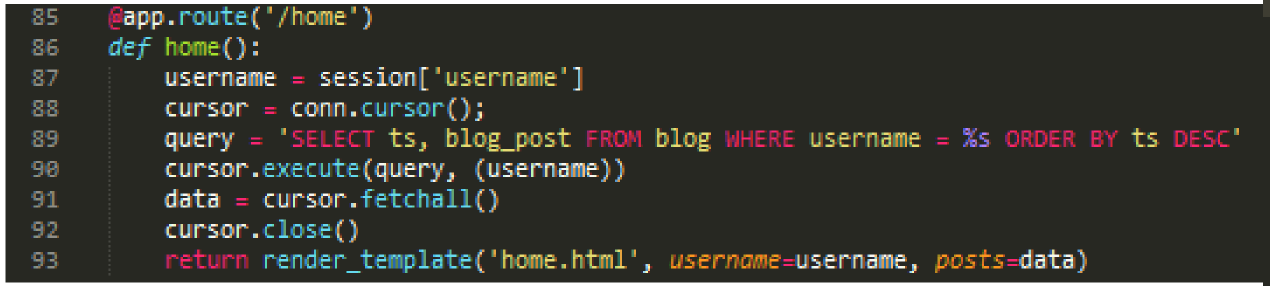
Posting



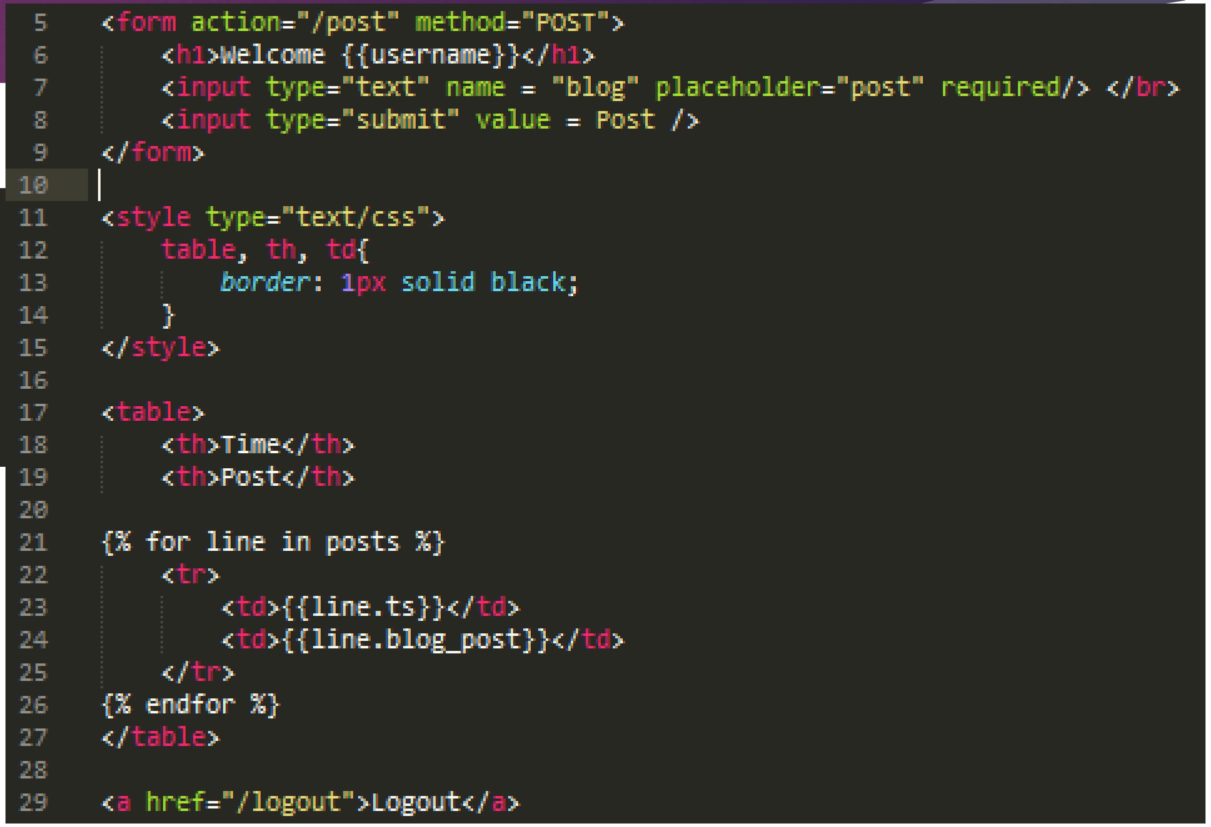
```
@app.route('/post', methods=['GET', 'POST'])
def post():
    username = session['username']
    cursor = conn.cursor();
    blog = request.form['blog']
    query = 'INSERT INTO blog (blog_post, username) VALUES(%s, %s)'
    cursor.execute(query, (blog, username))
    conn.commit()
    cursor.close()
    return redirect(url_for('home'))
```

Notice in the insert that we only insert into blog_post, and username.
We don’t want to insert into the timestamp because MySQL automatically updates it for us.

Home page



```python
@app.route('/home')
def home():
    username = session['username']
    cursor = conn.cursor();
    query = 'SELECT ts, blog_post FROM blog WHERE username = %s ORDER BY ts DESC'
    cursor.execute(query, (username))
    data = cursor.fetchall()
    cursor.close()
    return render_template('home.html', username=username, posts=data)
```

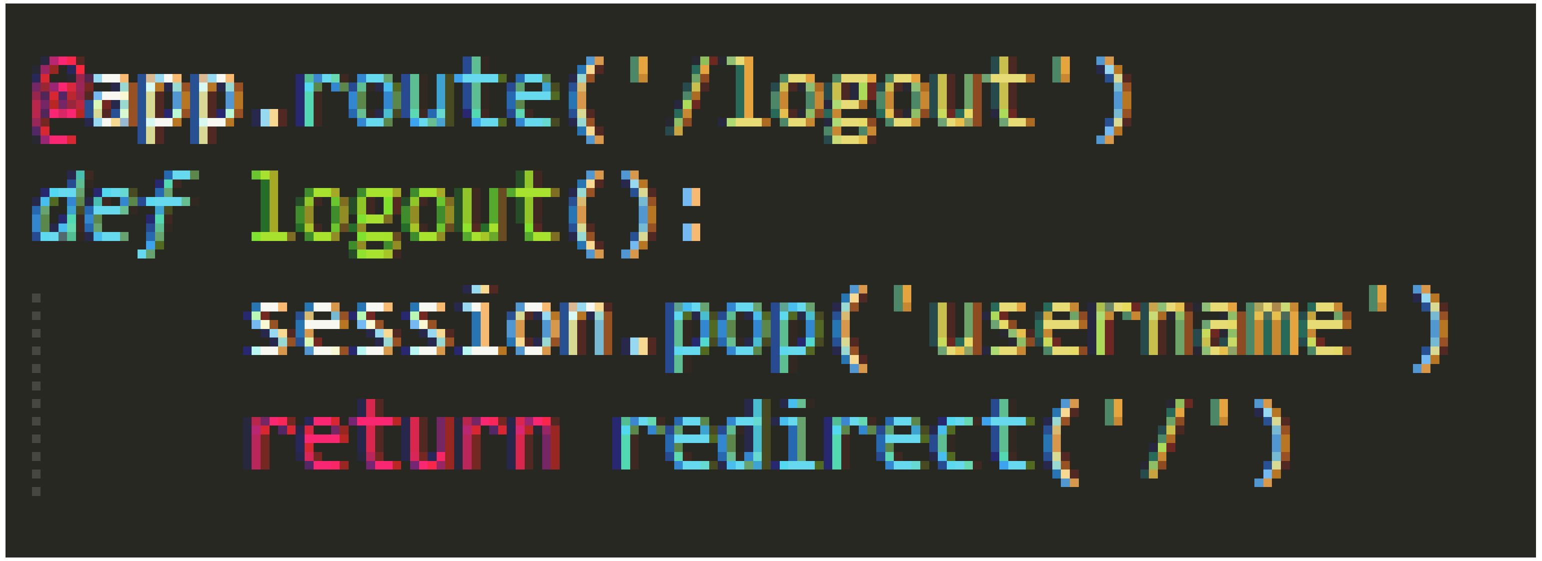


```html
<form action="/post" method="POST">
    <h1>Welcome {{username}}</h1>
    <input type="text" name = "blog" placeholder="post" required/> </br>
    <input type="submit" value = Post />
</form>

<style type="text/css">
    table, th, td{
        border: 1px solid black;
    }
</style>

<table>
    <th>Time</th>
    <th>Post</th>

{% for line in posts %}
    <tr>
        <td>{{line.ts}}</td>
        <td>{{line.blog_post}}</td>
    </tr>
{% endfor %}
</table>

<a href="/logout">Logout</a>
```

We want to allow the user to be able to post and see their posts on the front page. We call fetchall() and pass it into the home.html page.

Update the home.html page and use jinja’s for loop to iterate through the data given and display it

Logout



```
@app.route('/logout')
def logout():
    session.pop('username')
    return redirect('/')
```

To log out of the application, simply pop ‘username’ from the session store.
Note that if the user presses the back button on the browser or manually types in a path that requires the user to be logged in, bad things will happen. In all the routes add a check to see if ‘username’ is in session before doing any other operations.