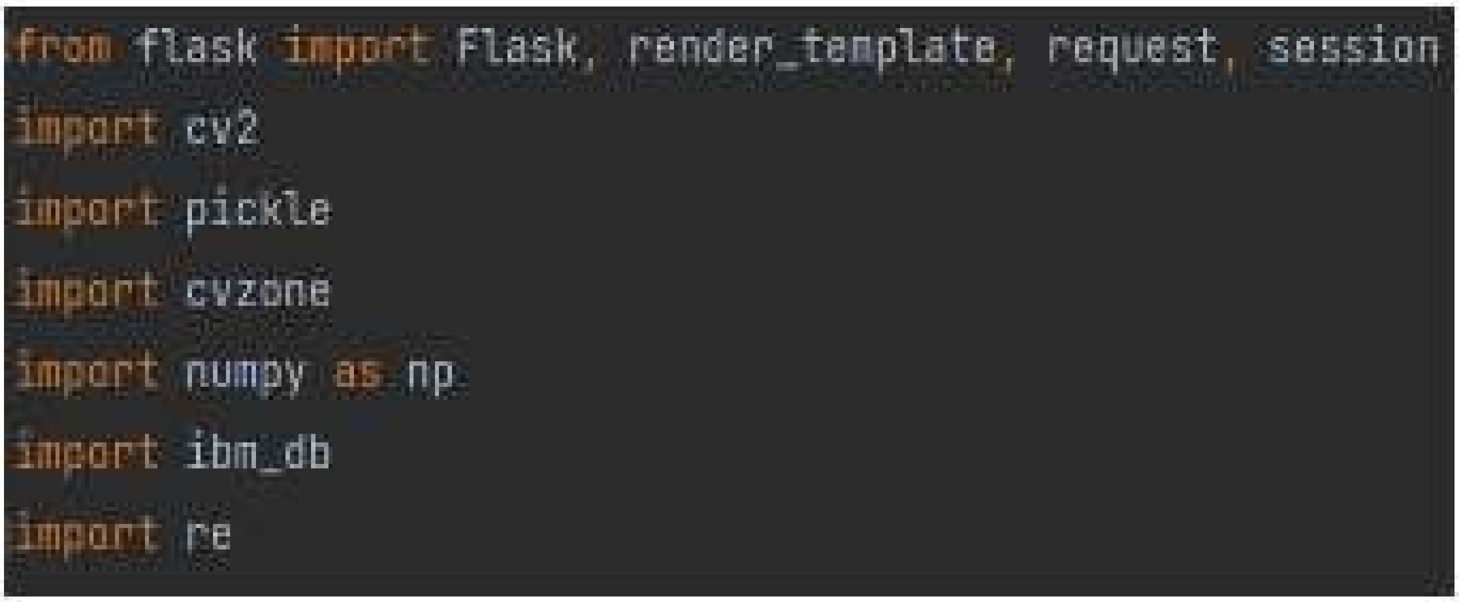
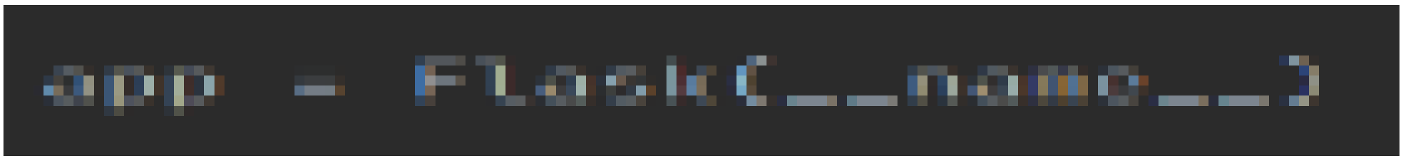
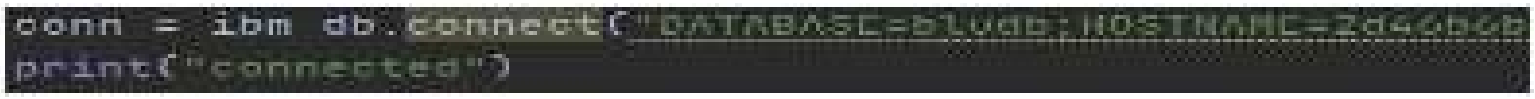


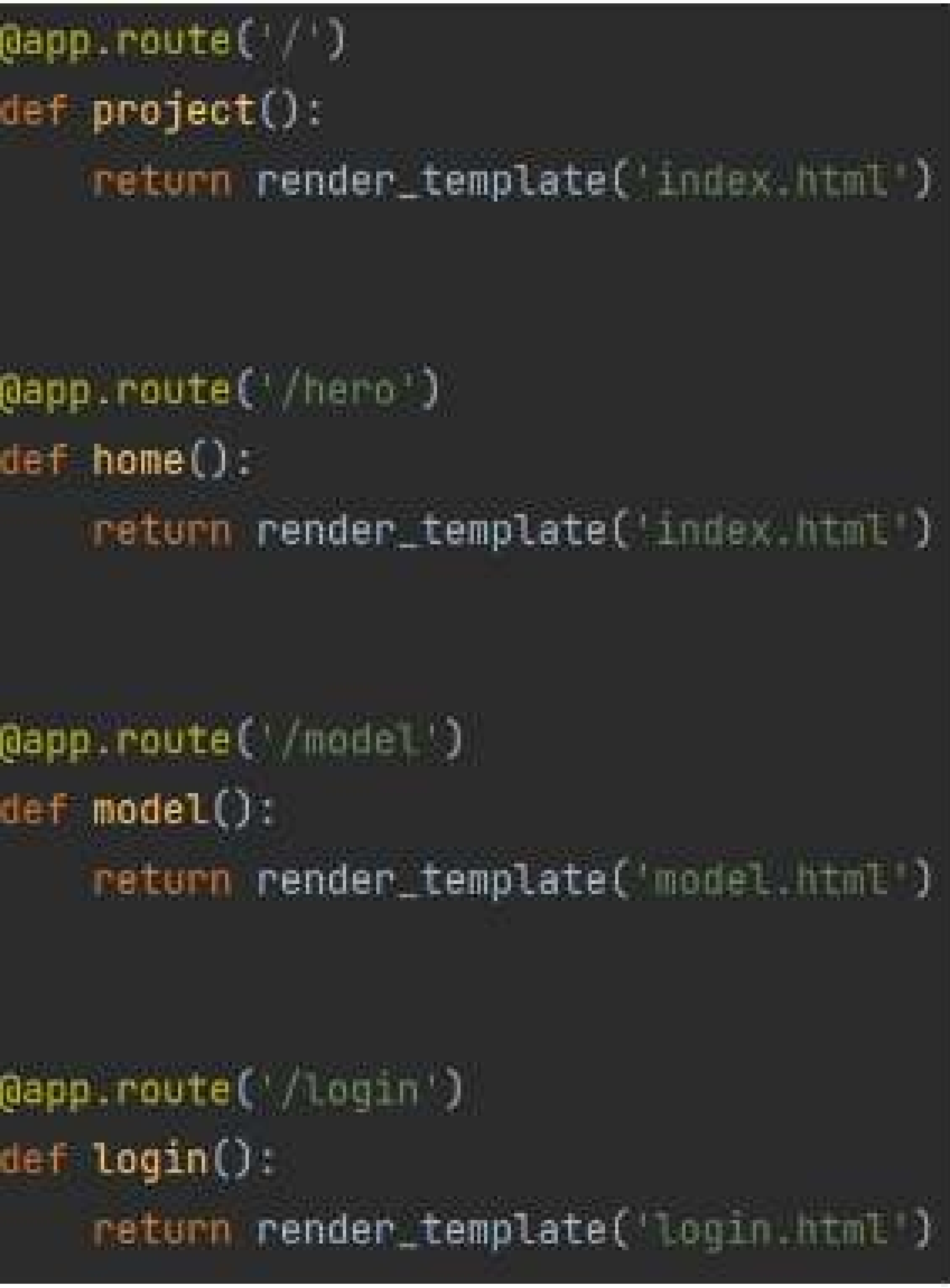
```python
from flask import Flask, render_template, request, session
import cv2
import pickle
import cvzone
import numpy as np
import ibm_db
import re
```



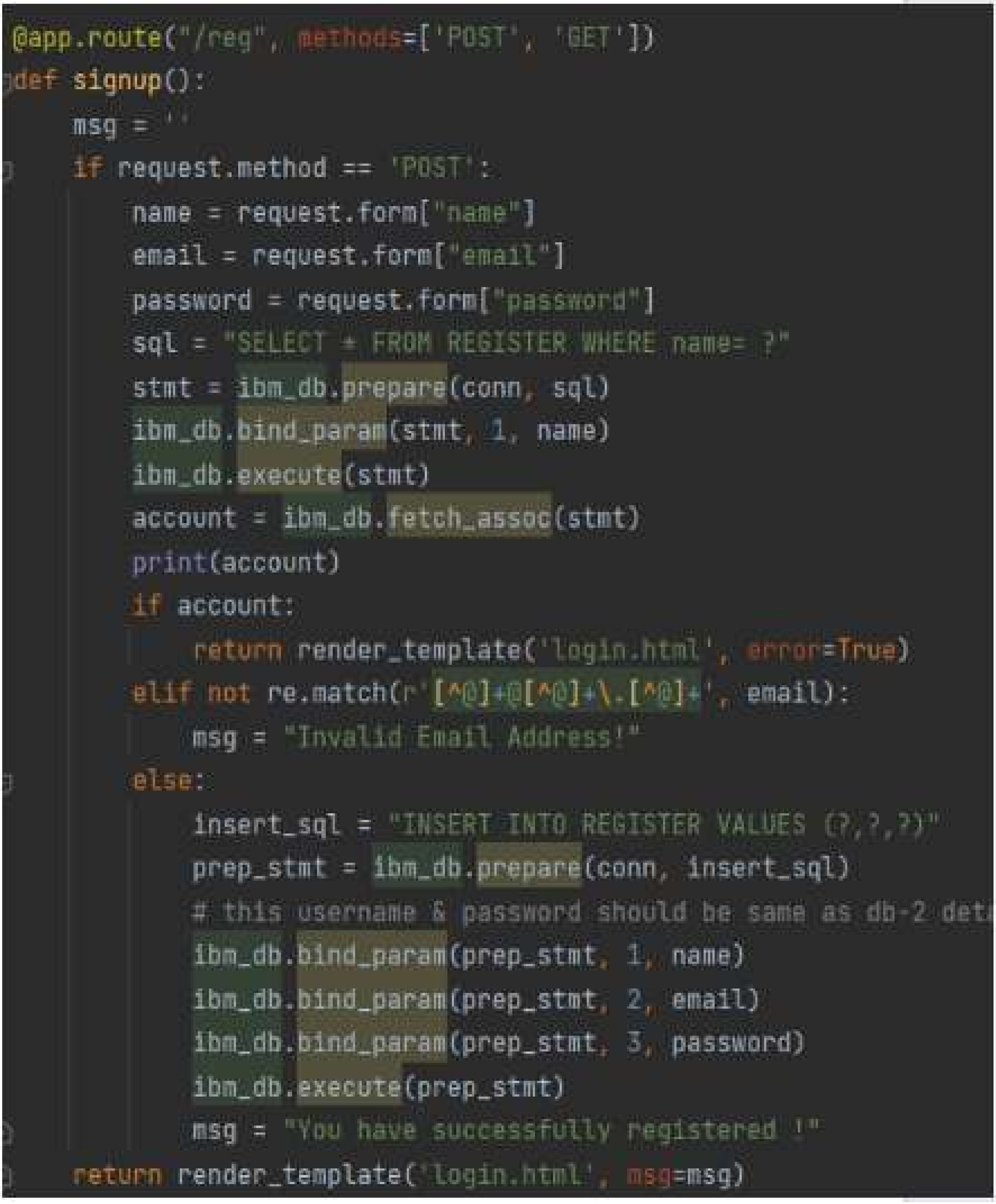
```
app = Flask(__name__)
```



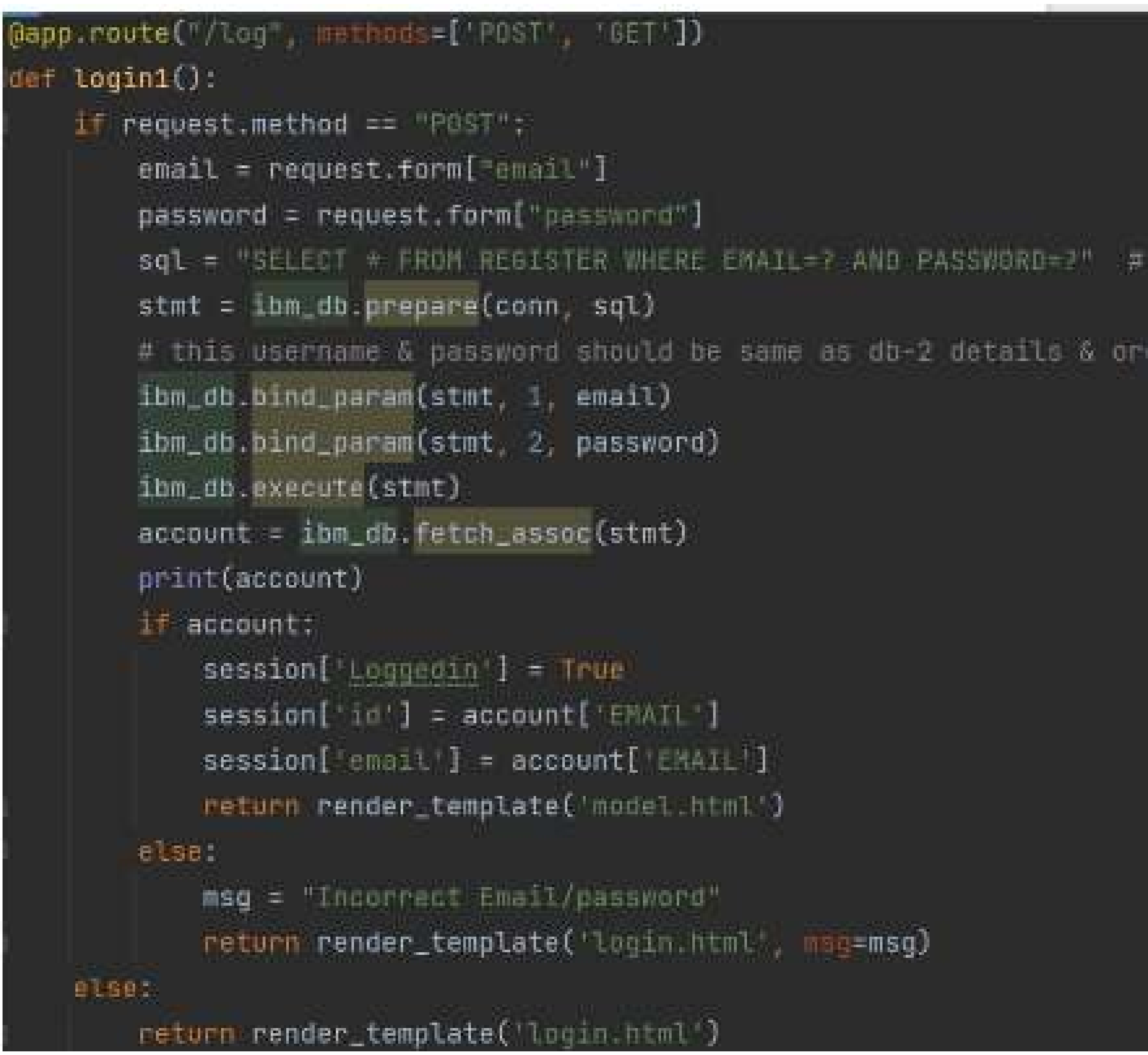
```
conn = ibm db.connect("DATABASE=bludb;HOSTNAME=2d46b6b
print("connected")
```



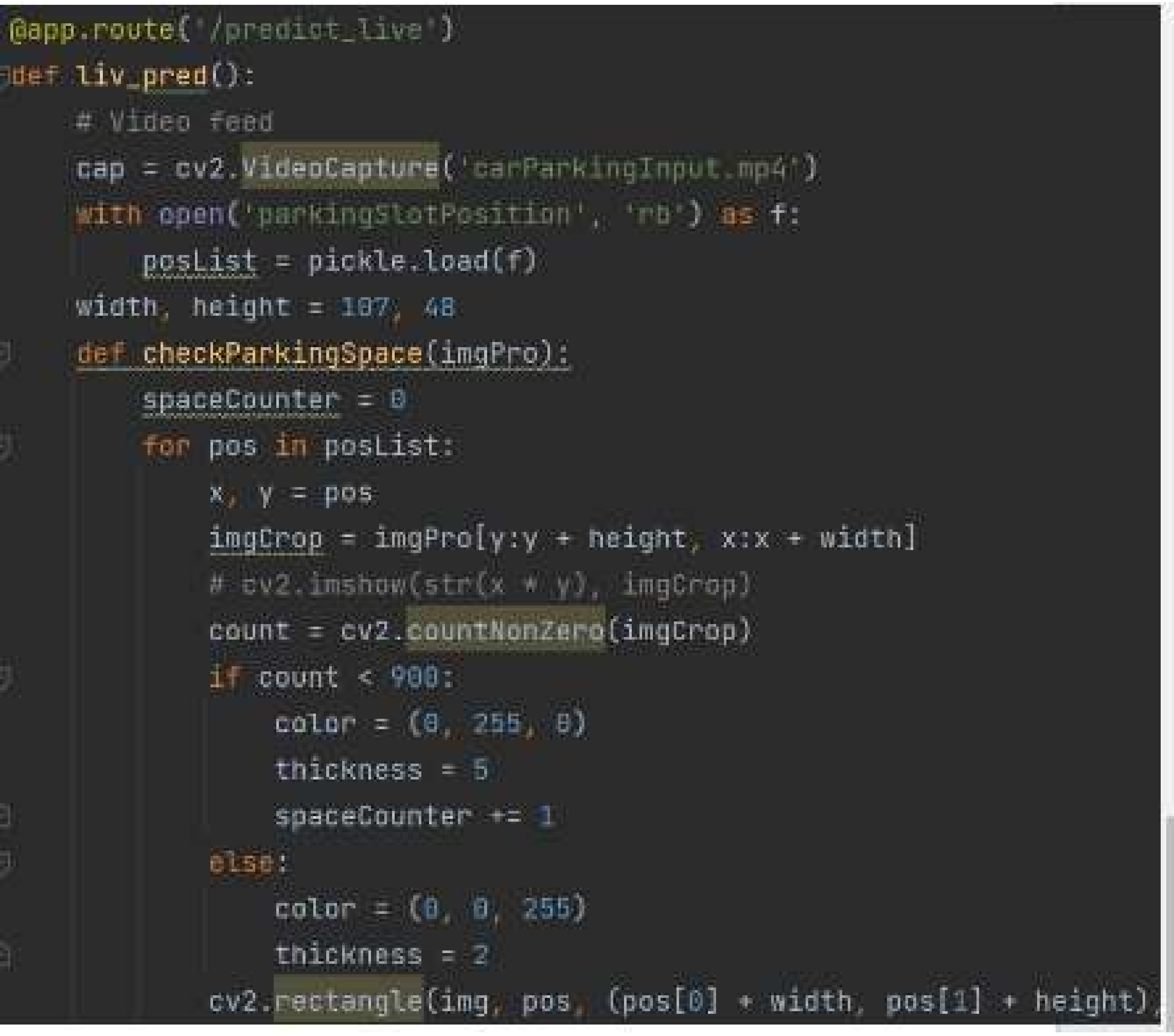
```
@app.route('/')
def project():
    return render_template('index.html')



@app.route('/hero')
def home():
    return render_template('index.html')



@app.route('/model')
def model():
    return render_template('model.html')



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



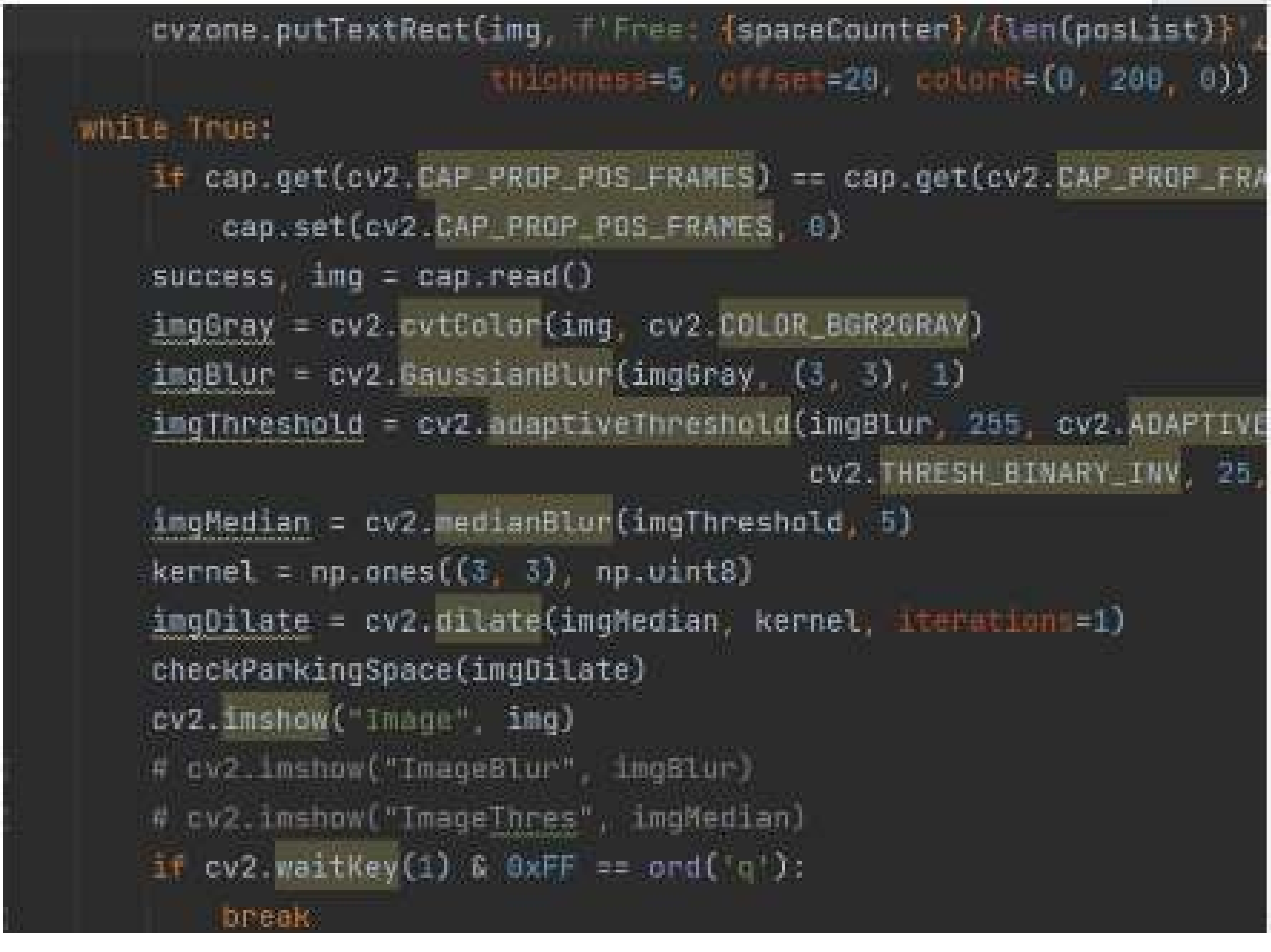
```python
@app.route("/reg", methods=['POST', 'GET'])
def signup():
    msg = ''
    if request.method == 'POST':
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE name= ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            return render_template('login.html', error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address!"
        else:
            insert_sql = "INSERT INTO REGISTER VALUES (?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            # this username & password should be same as db-2 deta
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg = "You have successfully registered !"
    return render_template('login.html', msg=msg)
```



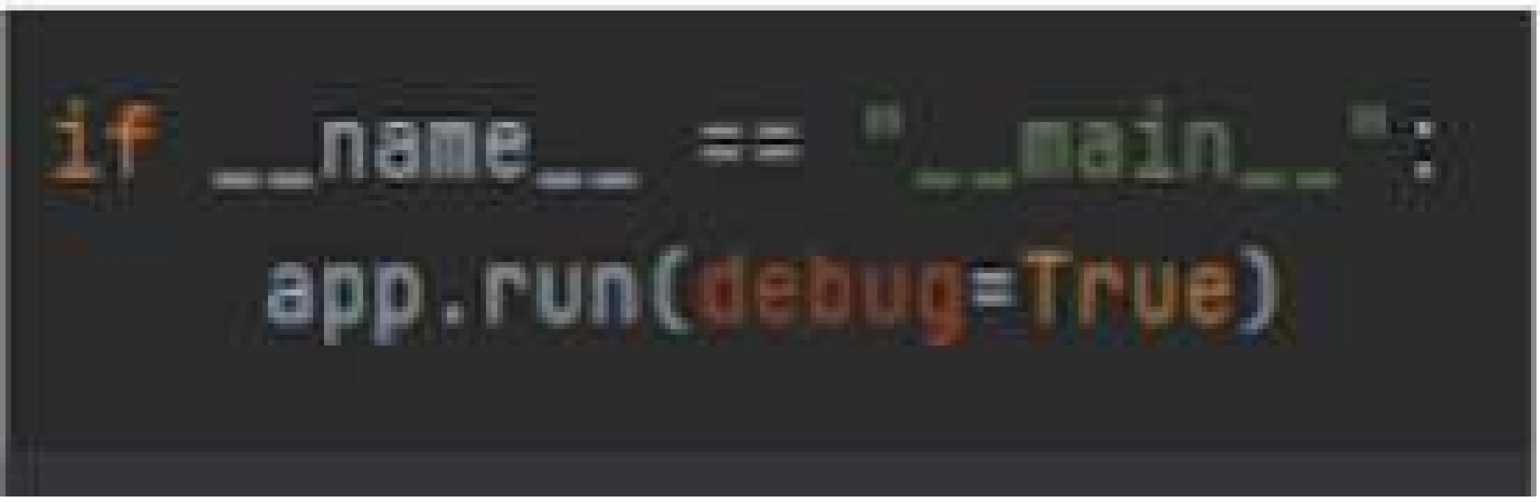
```
@app.route("/log", methods=['POST', 'GET'])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND PASSWORD=?"  #
        stmt = ibm_db.prepare(conn, sql)
        # this username & password should be same as db-2 details & or
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            session['email'] = account['EMAIL']
            return render_template('model.html')
        else:
            msg = "Incorrect Email/password"
            return render_template('login.html', msg=msg)
    else:
        return render_template('login.html')
```



```
@app.route('/predict_live')
def liv_pred():
    # Video feed
    cap = cv2.VideoCapture('carParkingInput.mp4')
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
    width, height = 107, 48
    def checkParkingSpace(imgPro):
        spaceCounter = 0
        for pos in posList:
            x, y = pos
            imgCrop = imgPro[y:y + height, x:x + width]
            # cv2.imshow(str(x * y), imgCrop)
            count = cv2.countNonZero(imgCrop)
            if count < 900:
                color = (0, 255, 0)
                thickness = 5
                spaceCounter += 1
            else:
                color = (0, 0, 255)
                thickness = 2
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height),
```



```
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}',
                       thickness=5, offset=20, colorR=(0, 200, 0))
while True:
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRA
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    success, img = cap.read()
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE
                                         cv2.THRESH_BINARY_INV, 25,
    imgMedian = cv2.medianBlur(imgThreshold, 5)
    kernel = np.ones((3, 3), np.uint8)
    imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
    checkParkingSpace(imgDilate)
    cv2.imshow("Image", img)
    # cv2.imshow("ImageBlur", imgBlur)
    # cv2.imshow("ImageThres", imgMedian)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```



```
if __name__ == "__main__":
    app.run(debug=True)
```