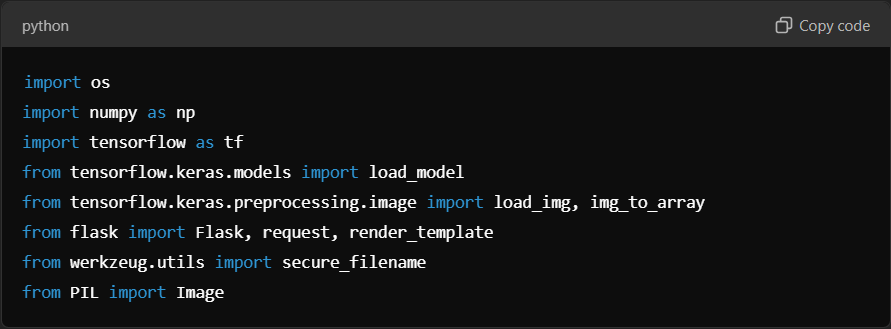
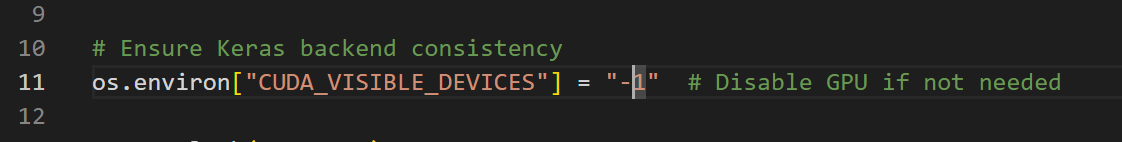
App.py explanation –



1. **os**: Provides functions to interact with the operating system (e.g., managing directories, file paths).
2. **numpy as np**: Used for numerical operations, such as manipulating image data.
3. **tensorflow as tf**: TensorFlow is used to load and interact with the machine learning model.
4. **load\_model**: A function from TensorFlow/Keras to load pre-trained models saved in the .h5 format.
5. **load\_img, img\_to\_array**: Functions for loading images and converting them into arrays for the model to process.
6. **Flask**: The web framework that powers this application.
7. **request**: Handles HTTP requests in Flask.
8. **render\_template**: Used to render HTML templates (e.g., index.html).
9. **secure\_filename**: Ensures filenames of uploaded files are safe to avoid security risks.
10. **Image (from PIL)**: Handles image-related tasks like opening and processing images.



**What is CUDA\_VISIBLE\_DEVICES?**

* **CUDA (Compute Unified Device Architecture)**: A parallel computing platform created by NVIDIA that enables developers to use the power of GPUs for tasks like deep learning, image processing, and more.
* **CUDA\_VISIBLE\_DEVICES**: An environment variable in TensorFlow (and other machine learning frameworks) that controls which GPU(s) the program can access.

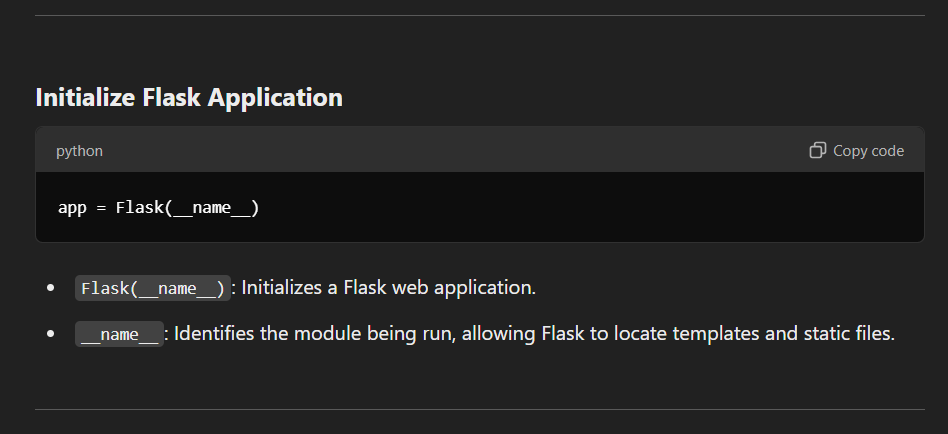
**What does -1 mean?**

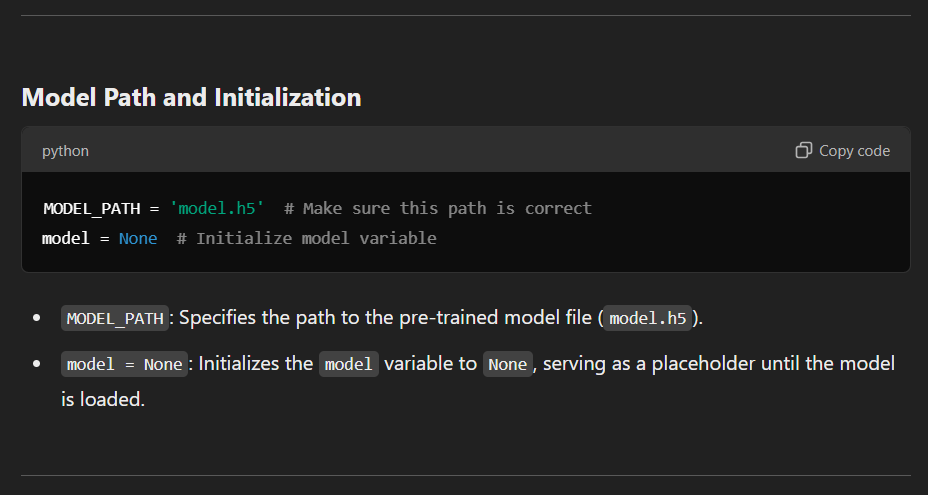
* When you set **CUDA\_VISIBLE\_DEVICES = "-1"**, it tells TensorFlow:
  + "Do not use any GPUs."
  + "Force all computations to run on the CPU instead."

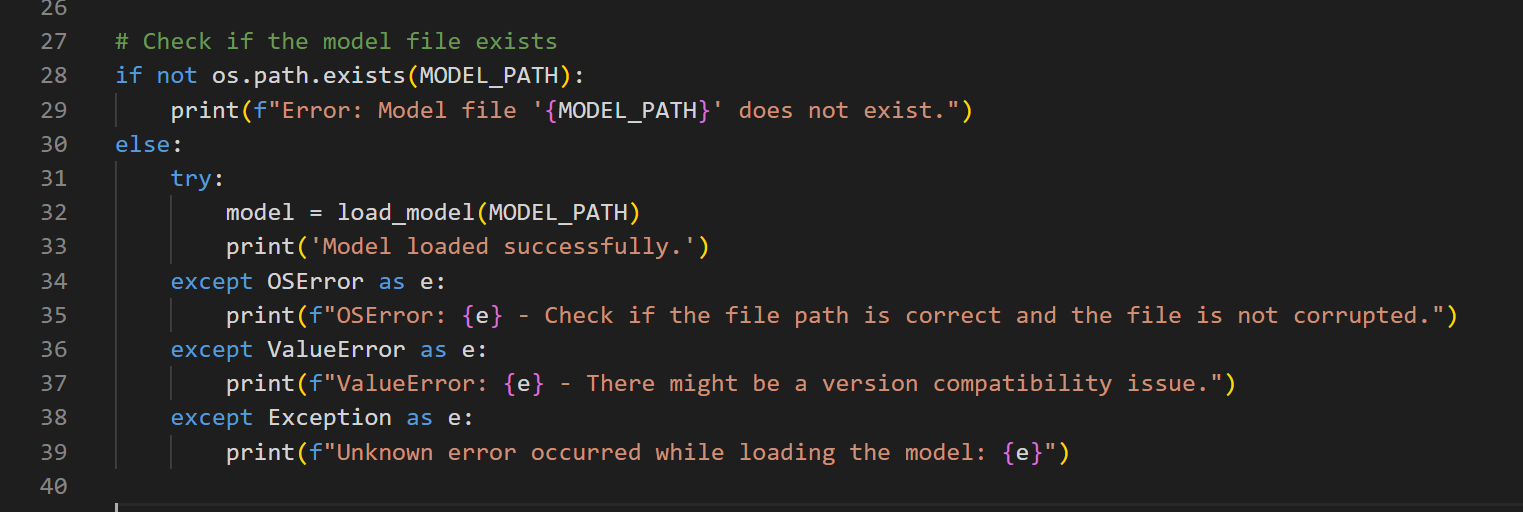
**How Does This Line Work?**

os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1"

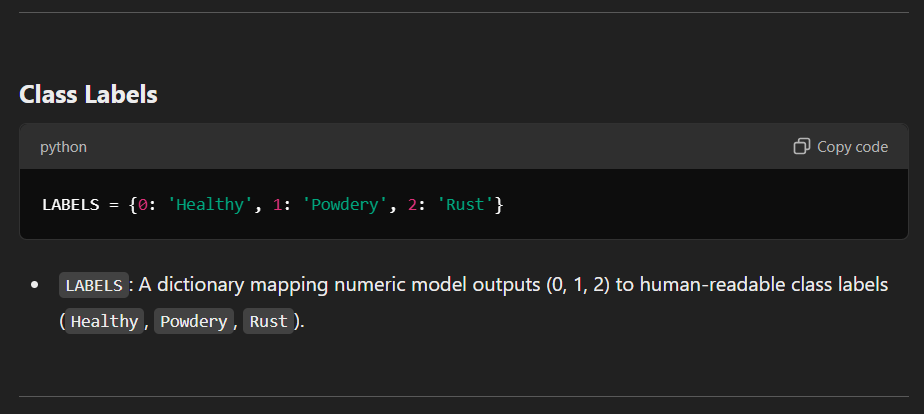
1. **os.environ**:
   * A Python dictionary that holds environment variables.
   * You use it to set or modify variables visible to your program.
2. **"CUDA\_VISIBLE\_DEVICES"**:
   * The name of the environment variable being modified.
   * It specifies which GPUs TensorFlow can use.
     + Example: "CUDA\_VISIBLE\_DEVICES=0,1" would allow TensorFlow to use **GPU 0 and GPU 1**.
     + Setting it to -1 makes **no GPUs** visible to TensorFlow.
3. **Value "-1"**:
   * Disables GPU usage by making the GPU "invisible" to TensorFlow.







* **os.path.exists(MODEL\_PATH)**: Checks if the file model.h5 exists.
* **If file doesn't exist**: Prints an error message.
* **load\_model(MODEL\_PATH)**: Loads the pre-trained model.
* **Exception handling**:
  + **OSError**: Handles file-related errors (e.g., file not found, corrupted file).
  + **ValueError**: Handles compatibility issues between the model and TensorFlow version.
  + **Generic Exception**: Catches any unexpected errors.



**Prediction Function**

**def get\_prediction(image\_path):**

**if model is None:**

**return "Error: Model is not loaded."**

**try:**

**# Load and preprocess the image**

**img = load\_img(image\_path, target\_size=(225, 225))**

**x = img\_to\_array(img)**

**x = x.astype('float32') / 255.0**

**x = np.expand\_dims(x, axis=0)**

**# Predict the class probabilities**

**predictions = model.predict(x)**

**if predictions.size == 0:**

**return "Error: No prediction returned."**

**# Get the label with the highest probability**

**predicted\_label = LABELS[np.argmax(predictions[0])]**

**return predicted\_label**

**except Exception as e:**

**return f"Prediction error: {e}"**

1. **Check if model is loaded**: Returns an error if the model is not loaded.
2. **Image preprocessing**:
   * **load\_img**: Loads the image and resizes it to 225x225 pixels.
   * **img\_to\_array**: Converts the image into a NumPy array.
   * **Scaling**: Divides pixel values by 255 to normalize them to [0, 1].
   * **Expand dimensions**: Adds an extra dimension to match the model’s expected input shape.
3. **Model prediction**:
   * **model.predict(x)**: Predicts class probabilities for the image.
   * **np.argmax(predictions[0])**: Identifies the class with the highest probability.
   * **LABELS**: Maps the predicted class index to its label.
4. **Error handling**: Catches any errors during prediction.



**Flask Route: Predict**

**@app.route('/predict', methods=['POST'])**

**def predict():**

**if 'file' not in request.files:**

**return "No file part in the request."**

**file = request.files['file']**

**if file.filename == '':**

**return "No file selected."**

**try:**

**basepath = os.path.dirname(\_\_file\_\_)**

**upload\_folder = os.path.join(basepath, 'uploads')**

**os.makedirs(upload\_folder, exist\_ok=True)**

**file\_path = os.path.join(upload\_folder, secure\_filename(file.filename))**

**file.save(file\_path)**

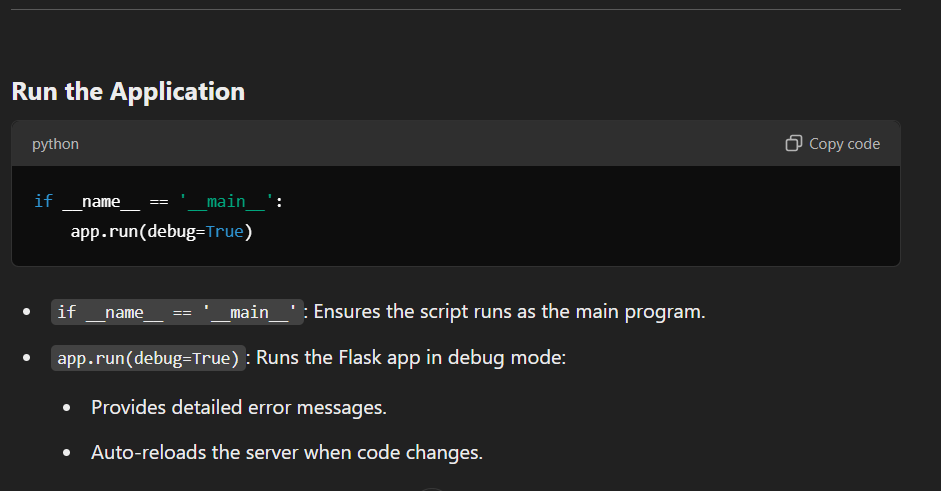
**prediction = get\_prediction(file\_path)**

**return prediction**

**except Exception as e:**

**return f"File processing error: {e}"**

1. **Check file in request**:
   * Ensures the user uploaded a file.
   * Returns an error if no file is provided or if the filename is empty.
2. **Save uploaded file**:
   * **os.path.dirname(\_\_file\_\_)**: Gets the script’s directory.
   * **os.path.join**: Joins paths to create the uploads folder.
   * **os.makedirs**: Creates the uploads folder if it doesn’t exist.
   * **secure\_filename**: Ensures the file name is safe.
   * **file.save**: Saves the uploaded file.
3. **Make a prediction**: Calls get\_prediction() on the saved file and returns the result.
4. **Error handling**: Catches any issues during file processing.



**Summary**

This script is a **Flask-based image classification web application**. It:

* Accepts image uploads through a web interface.
* Loads a pre-trained Keras model (model.h5).
* Predicts the class of the uploaded image (Healthy, Powdery, Rust).
* Returns the prediction result to the user.