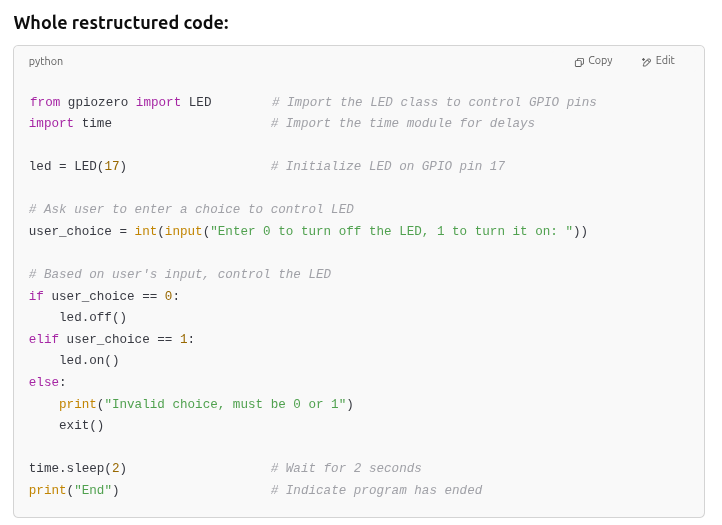
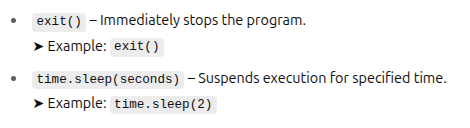
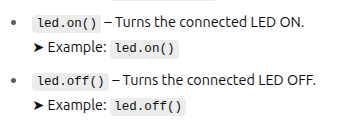
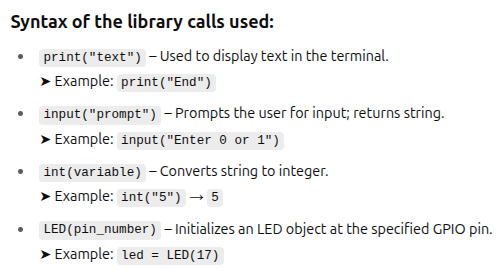
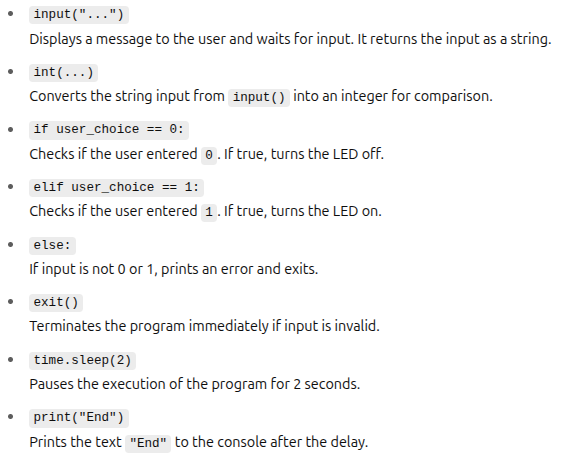
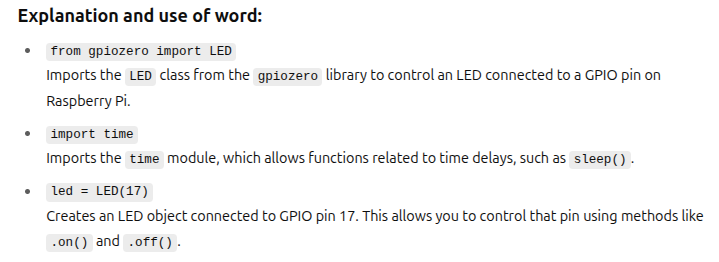
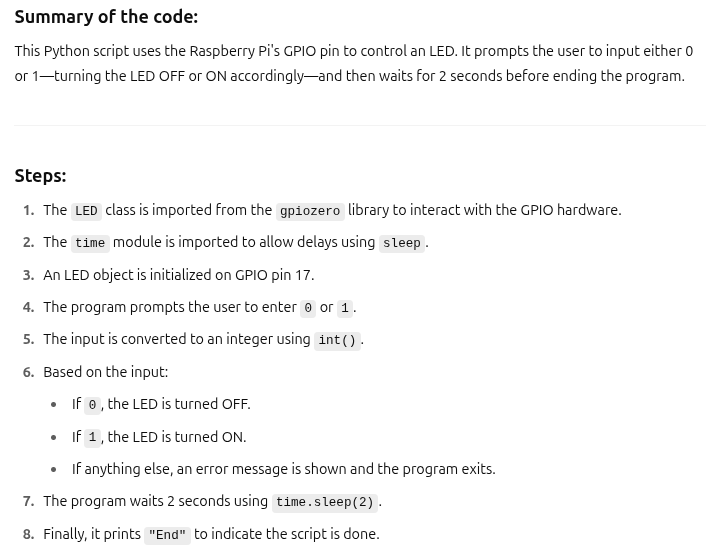
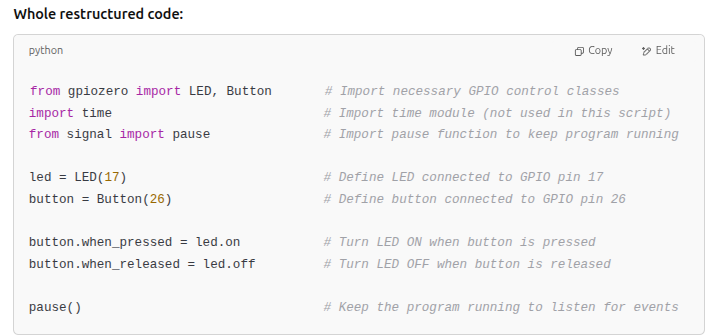
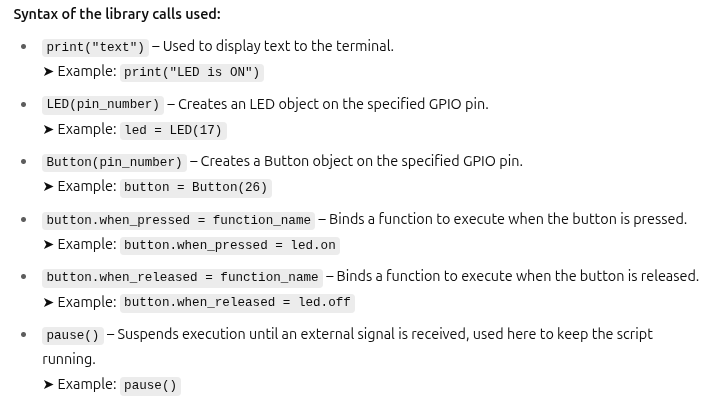
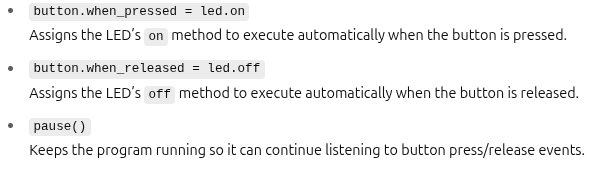
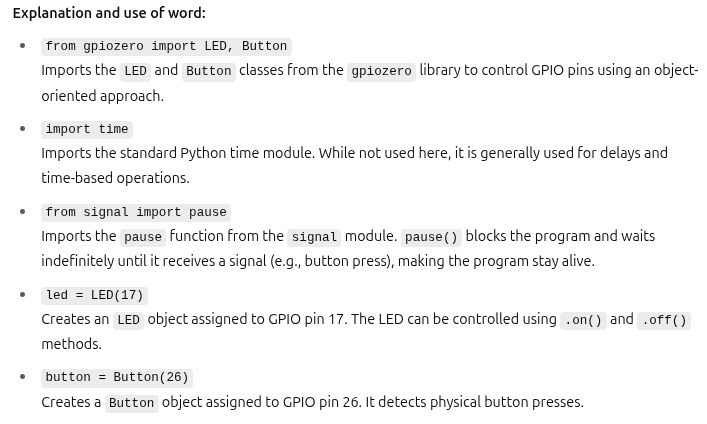
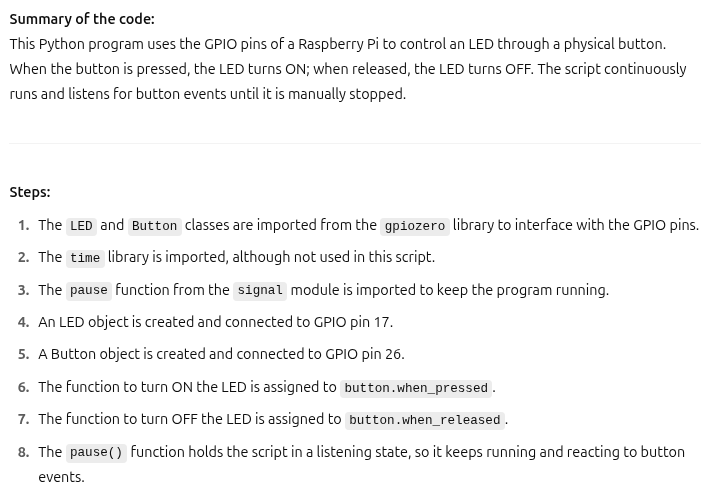
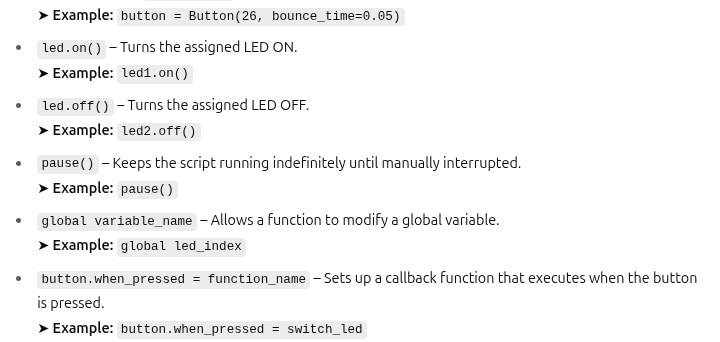
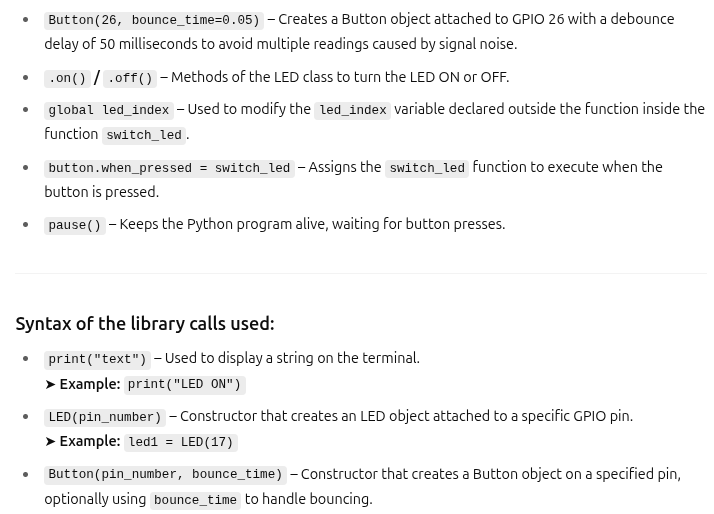
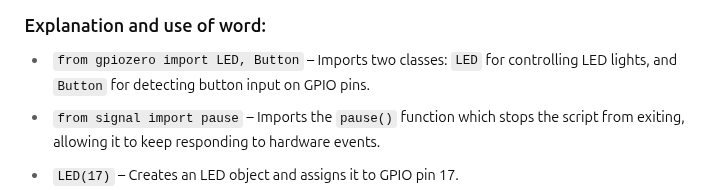
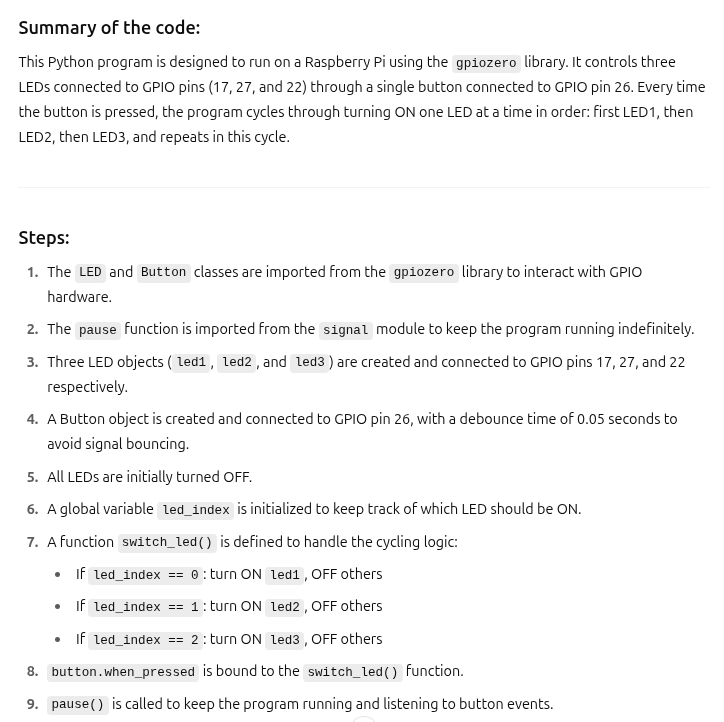
**LED-TURN OFF-ON**



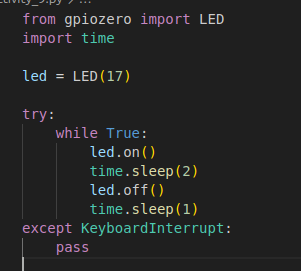
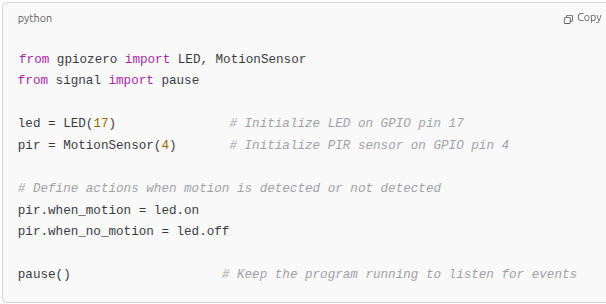
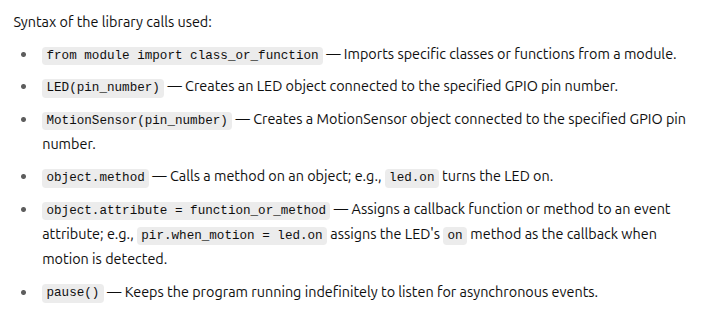
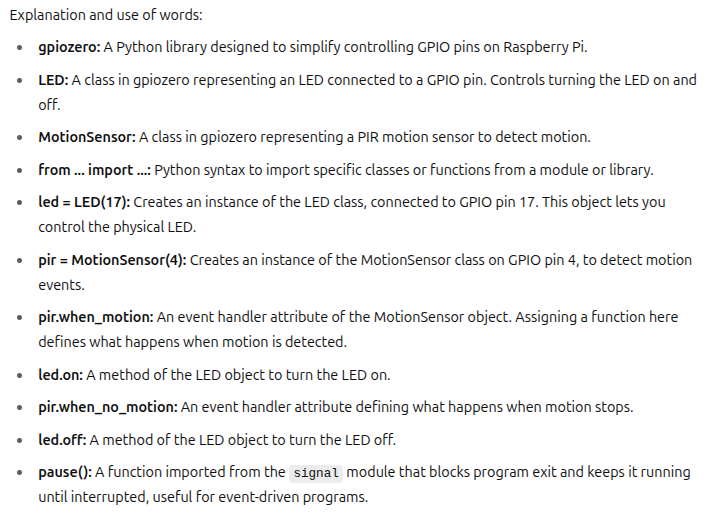
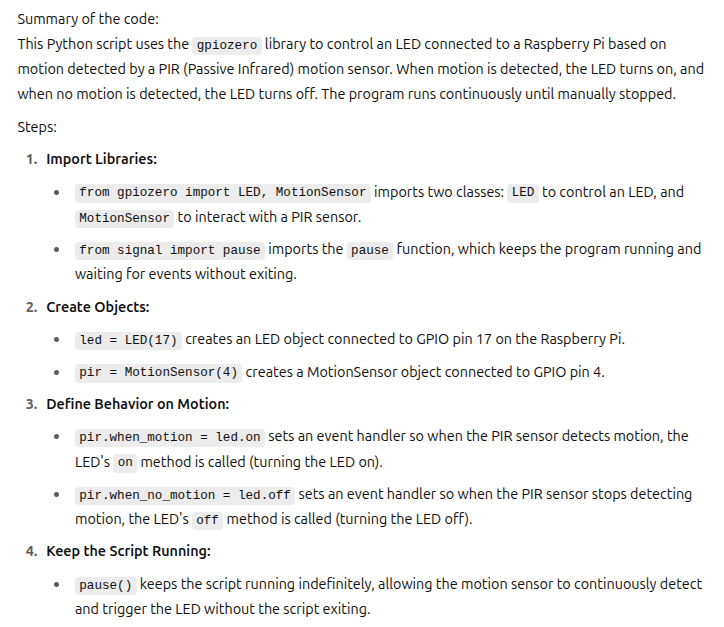
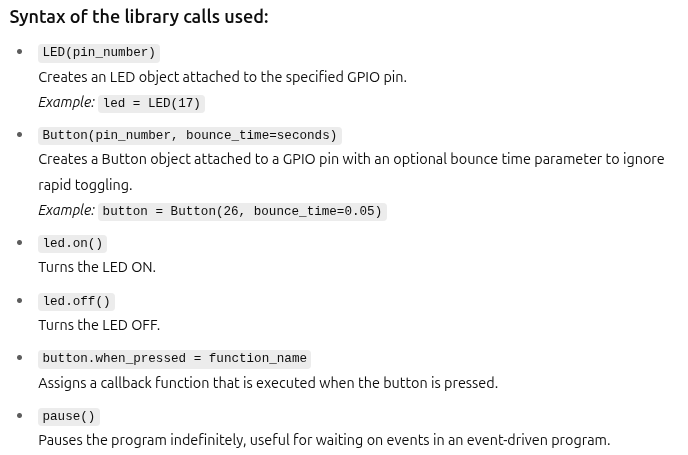
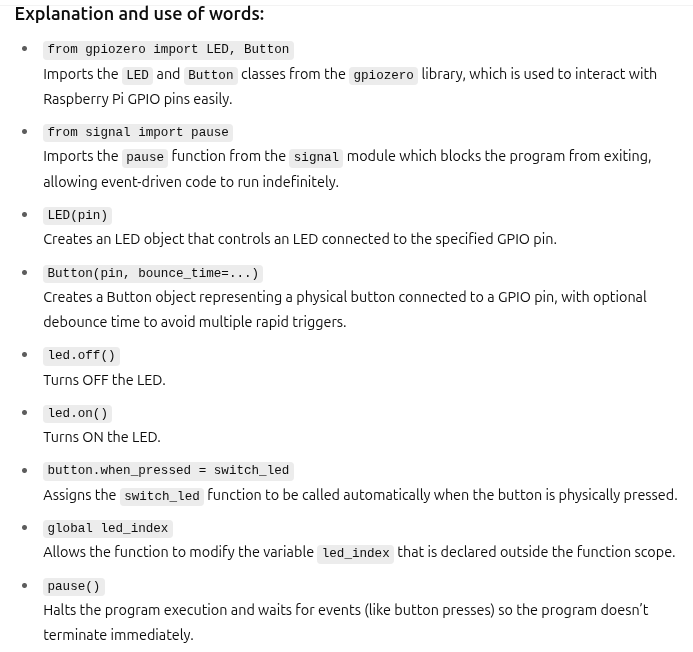
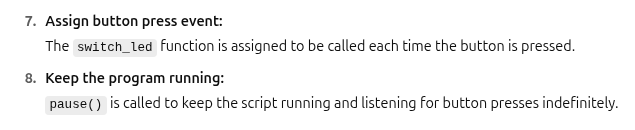
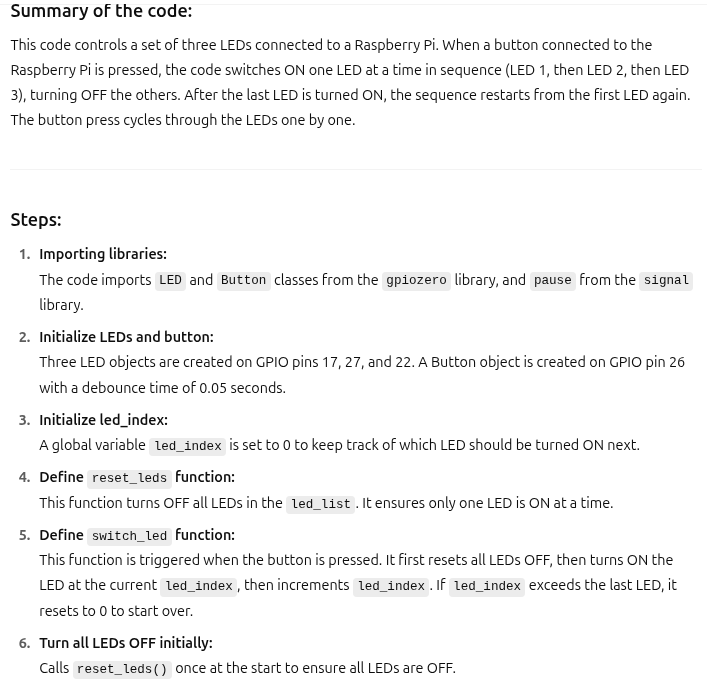
**SIGNAL-BUTTON**



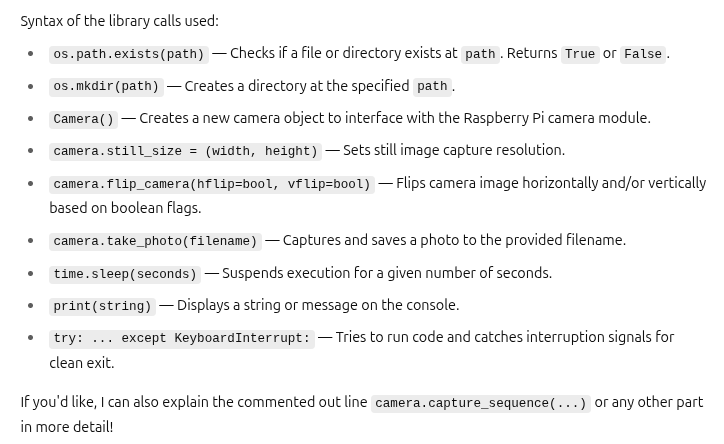
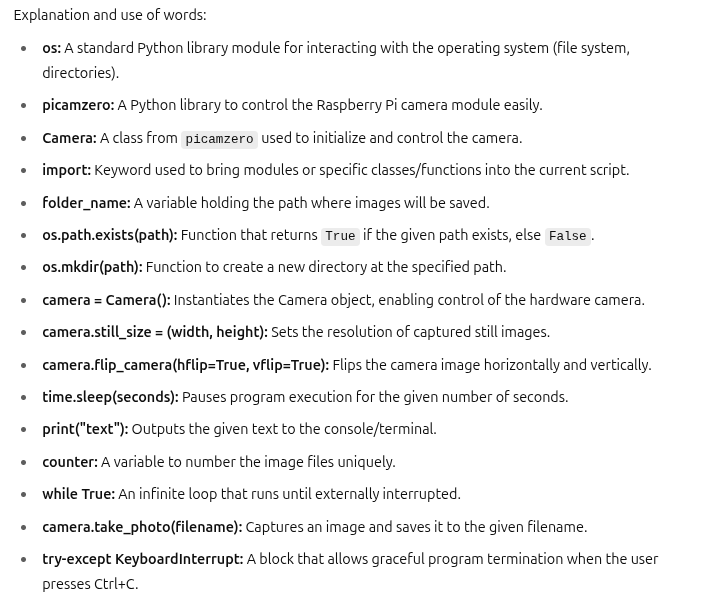
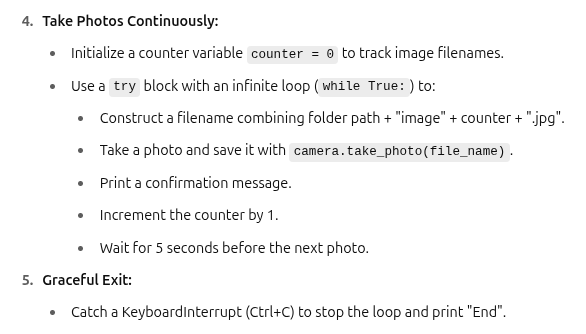
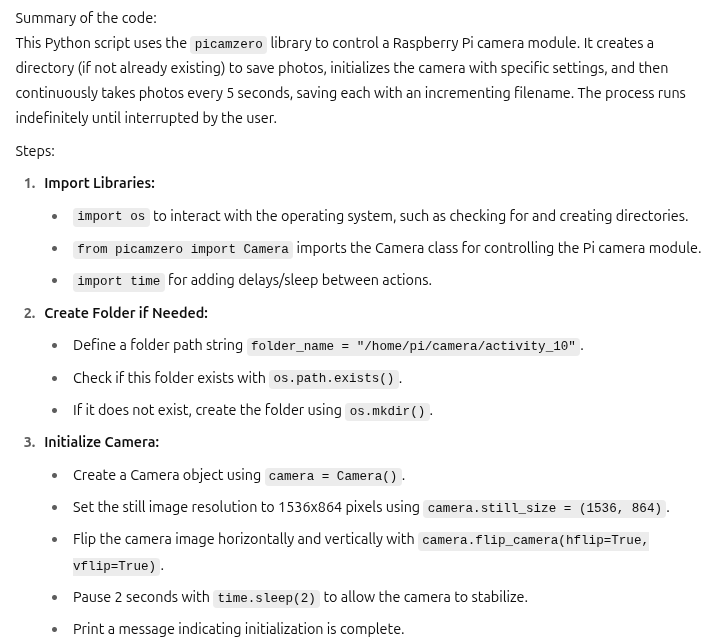
**MULTIPLE LED’S**



**PROGRAM -4**



**PYTHON - CAMERA**



**FLASK**

**Summary of the code**:

This Python script uses the Flask web framework to create a web server that interacts with GPIO components (a button and three LEDs) on a Raspberry Pi. The user can access the server through a browser to check the button state or control the LEDs by sending specific HTTP requests.

**Steps**:

1) Import required libraries: `Flask` for web application creation, `Button` and `LED` from `gpiozero` for GPIO hardware control.

2) Initialize a `Button` object on GPIO pin 26 with a debounce time of 0.05 seconds to prevent false triggers.

3) Initialize three `LED` objects connected to GPIO pins 17, 27, and 22 and store them in a list.

4) Turn off all LEDs initially using a `for` loop.

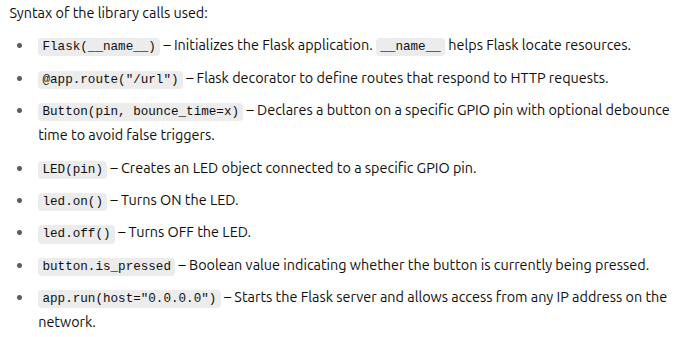
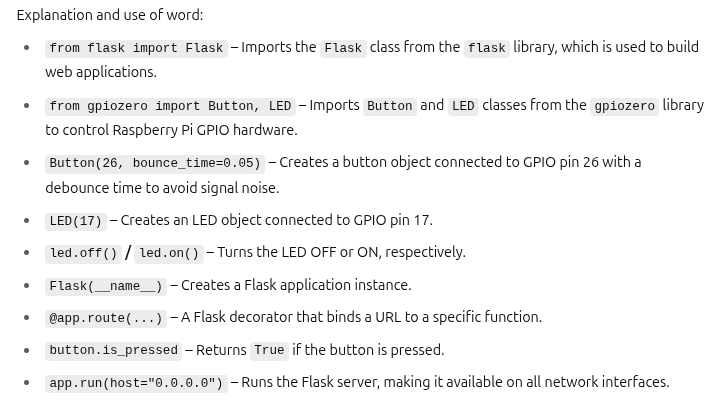
5) Create a Flask web application instance.

6) Define a route at `/` which returns a welcome message when accessed.

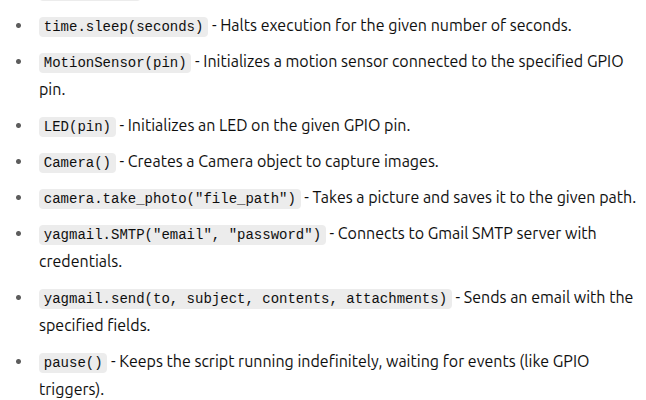
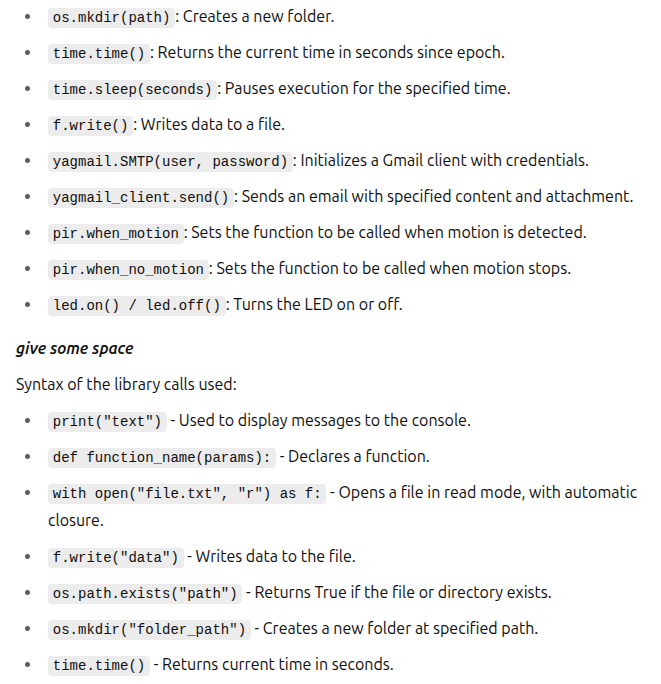
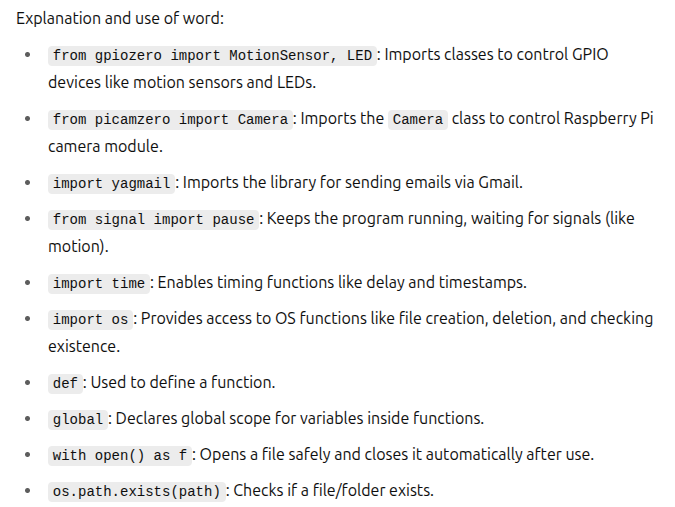
7) Define a route `/push-button` that checks if the physical button is pressed and returns the result.

8) Define a route `/led/<int:led\_number>/state/<int:state>` to turn ON or OFF a specific LED based on the values passed in the URL.

9) Start the Flask application, making it accessible from any IP on the network.



**RASPBERRY PI –Project Setup**



**Summary of the code:**

This Python script runs on a Raspberry Pi and automates the process of detecting motion using a PIR (Passive Infrared) sensor. When motion is detected and it lasts longer than a threshold time, the Raspberry Pi camera takes a picture. This picture is saved in a folder, logged into a file, and then sent via email to a predefined recipient. The system uses LEDs for visual feedback and ensures that photos are not taken too frequently.

**Steps**:

1) Import required libraries to handle GPIO (motion sensor, LED), camera control, email sending, time control, and file operations.

2) Define utility functions:

- `take\_photo()` to capture an image using the camera.

- `update\_photo\_log\_file()` to log photo file paths.

- `send\_photo\_by\_email()` to email the captured image.

3) Initialize global variables for motion tracking, timing, paths, and thresholds.

4) Setup email credentials securely from a hidden password file.

5) Setup and configure the camera resolution and orientation.

6) Ensure the destination folder and log file are ready (create/delete as needed).

7) Initialize GPIO components: PIR motion sensor and LED.

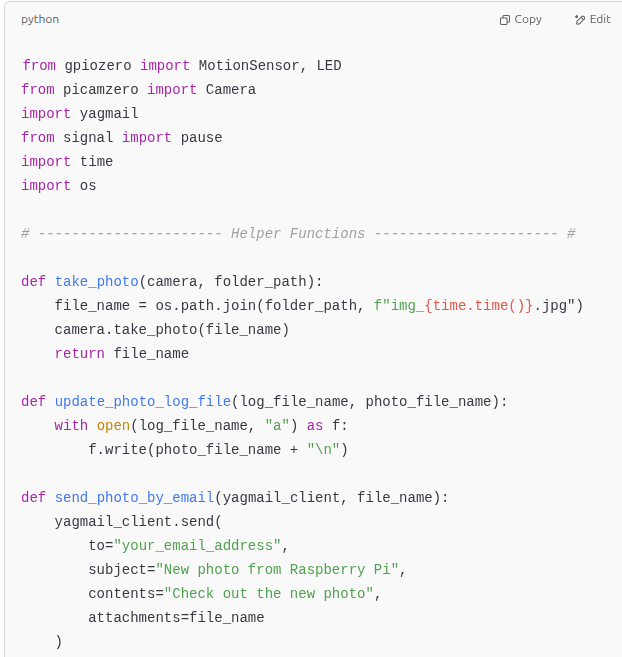
8) Define callbacks:

- `motion\_detected()` to record the start time and turn the LED on.

- `motion\_finished()` to check motion duration, take a photo if valid, and send it via email.

9) Bind the callbacks to the PIR sensor events.

10) Run the program continuously with `pause()` to keep it active.



def motion\_finished():

global last\_time\_photo\_taken

led.off()

motion\_duration = time.time() - time\_motion\_started

if motion\_duration > MOVEMENT\_DETECTED\_THRESHOLD:

if time.time() - last\_time\_photo\_taken > MIN\_DURATION\_BETWEEN\_PHOTOS:

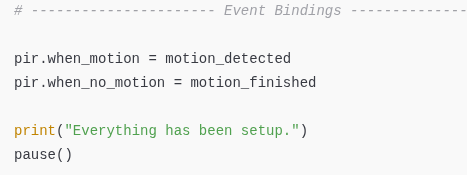
last\_time\_photo\_taken = time.time()

print("Taking a photo and sending it by email")

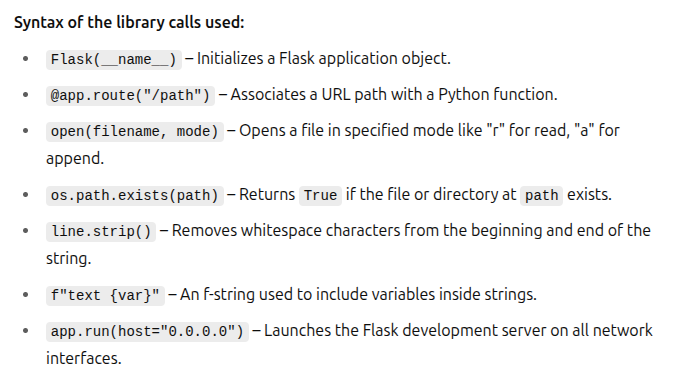
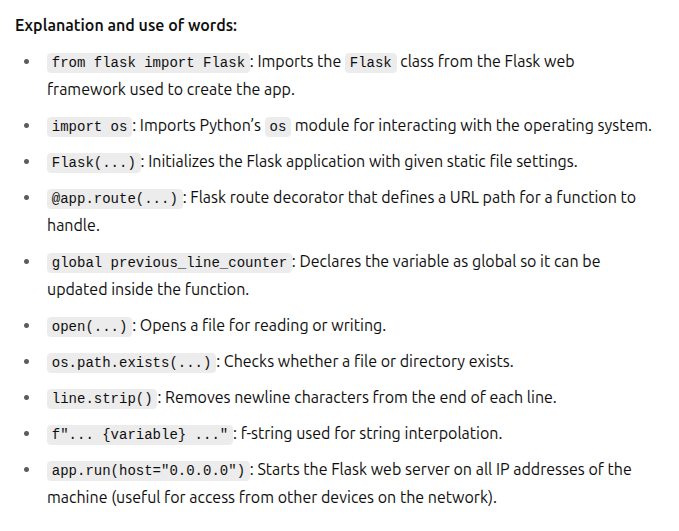
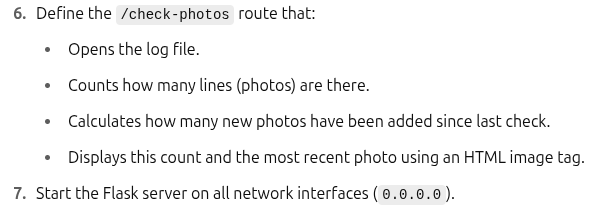
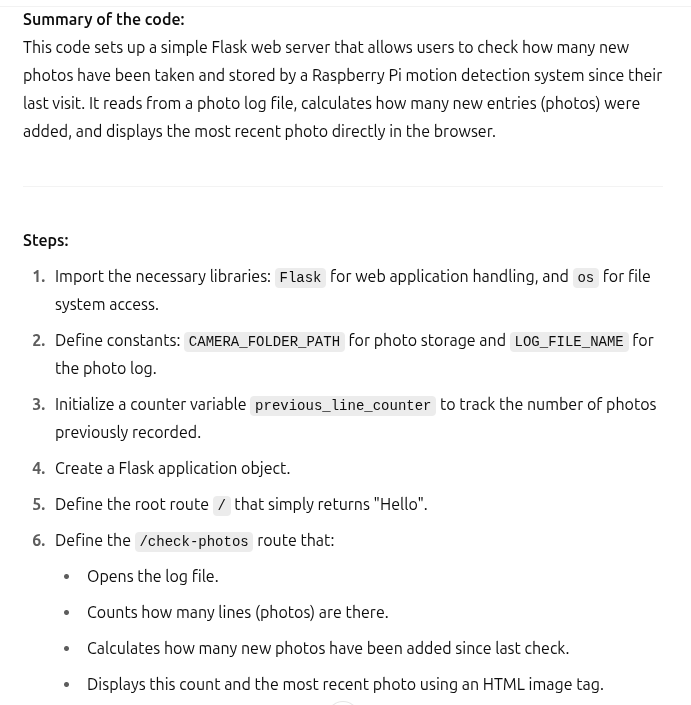
photo\_file\_name = take\_photo(camera, CAMERA\_FOLDER\_PATH)

update\_photo\_log\_file(LOG\_FILE\_NAME, photo\_file\_name)

send\_photo\_by\_email(yag, photo\_file\_name)



**FINAL CODE**



from flask import Flask

import os

# ---------------- Constants ----------------

CAMERA\_FOLDER\_PATH = "/home/pi/photos\_final\_project"

LOG\_FILE\_NAME = os.path.join(CAMERA\_FOLDER\_PATH, "photo\_logs.txt")

previous\_line\_counter = 0

# ---------------- Flask App Initialization ----------------

app = Flask(\_\_name\_\_, static\_url\_path=CAMERA\_FOLDER\_PATH, static\_folder=CAMERA\_FOLDER\_PATH)

# ---------------- Routes ----------------

@app.route("/")

def index():

return "Hello"

@app.route("/check-photos")

def check\_photos():

global previous\_line\_counter

line\_counter = 0

if os.path.exists(LOG\_FILE\_NAME):

last\_photo\_file\_name = ""

with open(LOG\_FILE\_NAME, "r") as f:

for line in f:

line\_counter += 1

last\_photo\_file\_name = line.strip()

difference = line\_counter - previous\_line\_counter

previous\_line\_counter = line\_counter

message = f"{difference} new photos since you last checked <br/>"

message += f"Last photo: {last\_photo\_file\_name} <br/>"

message += f"<img src=\"{last\_photo\_file\_name}\">"

return message

else:

return "No photo available"

# ---------------- Run Server ----------------

app.run(host="0.0.0.0")