LED-TURN OFF-ON

Summary of the code:

This Python script uses the Raspberry Pi's GPIO pin to control an LED. It prompts the user to input either 0 or 1—turning the LED OFF or ON accordingly—and then waits for 2 seconds before ending the program.

Steps:

1. The `LED` class is imported from the `gpiozero` library to interact with the GPIO hardware.
2. The `time` module is imported to allow delays using `sleep`.
3. An LED object is initialized on GPIO pin 17.
4. The program prompts the user to enter `0` or `1`.
5. The input is converted to an integer using `int()`.
6. Based on the input:
   - If `0`, the LED is turned OFF.
   - If `1`, the LED is turned ON.
   - If anything else, an error message is shown and the program exits.
7. The program waits 2 seconds using `time.sleep(2)`.
8. Finally, it prints `"End"` to indicate the script is done.

Explanation and use of word:

- `from gpiozero import LED`
  Imports the `LED` class from the `gpiozero` library to control an LED connected to a GPIO pin on Raspberry Pi.
- `import time`
  Imports the `time` module, which allows functions related to time delays, such as `sleep()`.
- `led = LED(17)`
  Creates an LED object connected to GPIO pin 17. This allows you to control that pin using methods like `.on()` and `.off()`.

- `input("...")`
  Displays a message to the user and waits for input. It returns the input as a string.
- `int(...)`
  Converts the string input from `input()` into an integer for comparison.
- `if user_choice == 0:`
  Checks if the user entered `0`. If true, turns the LED off.
- `elif user_choice == 1:`
  Checks if the user entered `1`. If true, turns the LED on.
- `else:`
  If input is not 0 or 1, prints an error and exits.
- `exit()`
  Terminates the program immediately if input is invalid.
- `time.sleep(2)`
  Pauses the execution of the program for 2 seconds.
- `print("End")`
  Prints the text `"End"` to the console after the delay.

Syntax of the library calls used:

- `print("text")` – Used to display text in the terminal.
  ➤ Example: `print("End")`
- `input("prompt")` – Prompts the user for input; returns string.
  ➤ Example: `input("Enter 0 or 1")`
- `int(variable)` – Converts string to integer.
  ➤ Example: `int("5")` → `5`
- `LED(pin_number)` – Initializes an LED object at the specified GPIO pin.
  ➤ Example: `led = LED(17)`

- `led.on()` – Turns the connected LED ON.
  ➤ Example: `led.on()`
- `led.off()` – Turns the connected LED OFF.
  ➤ Example: `led.off()`

- `exit()` – Immediately stops the program.
  ➤ Example: `exit()`
- `time.sleep(seconds)` – Suspends execution for specified time.
  ➤ Example: `time.sleep(2)`

Whole restructured code:

python Copy Edit

```python
from gpiozero import LED        # Import the LED class to control GPIO pins
import time                     # Import the time module for delays

led = LED(17)                   # Initialize LED on GPIO pin 17

# Ask user to enter a choice to control LED
user_choice = int(input("Enter 0 to turn off the LED, 1 to turn it on: "))

# Based on user's input, control the LED
if user_choice == 0:
    led.off()
elif user_choice == 1:
    led.on()
else:
    print("Invalid choice, must be 0 or 1")
    exit()

time.sleep(2)                   # Wait for 2 seconds
print("End")                    # Indicate program has ended
```

SIGNAL-BUTTON

Summary of the code:

This Python program uses the GPIO pins of a Raspberry Pi to control an LED through a physical button. When the button is pressed, the LED turns ON; when released, the LED turns OFF. The script continuously runs and listens for button events until it is manually stopped.

Steps:

1. The `LED` and `Button` classes are imported from the `gpiozero` library to interface with the GPIO pins.
2. The `time` library is imported, although not used in this script.
3. The `pause` function from the `signal` module is imported to keep the program running.
4. An LED object is created and connected to GPIO pin 17.
5. A Button object is created and connected to GPIO pin 26.
6. The function to turn ON the LED is assigned to `button.when_pressed`.
7. The function to turn OFF the LED is assigned to `button.when_released`.
8. The `pause()` function holds the script in a listening state, so it keeps running and reacting to button events.

Explanation and use of word:

- `from gpiozero import LED, Button`
  Imports the `LED` and `Button` classes from the `gpiozero` library to control GPIO pins using an object-oriented approach.
- `import time`
  Imports the standard Python time module. While not used here, it is generally used for delays and time-based operations.
- `from signal import pause`
  Imports the `pause` function from the `signal` module. `pause()` blocks the program and waits indefinitely until it receives a signal (e.g., button press), making the program stay alive.
- `led = LED(17)`
  Creates an `LED` object assigned to GPIO pin 17. The LED can be controlled using `.on()` and `.off()` methods.
- `button = Button(26)`
  Creates a `Button` object assigned to GPIO pin 26. It detects physical button presses.

- `button.when_pressed = led.on`
  Assigns the LED's `on` method to execute automatically when the button is pressed.
- `button.when_released = led.off`
  Assigns the LED's `off` method to execute automatically when the button is released.
- `pause()`
  Keeps the program running so it can continue listening to button press/release events.

Syntax of the library calls used:

- `print("text")` – Used to display text to the terminal.
  ➤ Example: `print("LED is ON")`
- `LED(pin_number)` – Creates an LED object on the specified GPIO pin.
  ➤ Example: `led = LED(17)`
- `Button(pin_number)` – Creates a Button object on the specified GPIO pin.
  ➤ Example: `button = Button(26)`
- `button.when_pressed = function_name` – Binds a function to execute when the button is pressed.
  ➤ Example: `button.when_pressed = led.on`
- `button.when_released = function_name` – Binds a function to execute when the button is released.
  ➤ Example: `button.when_released = led.off`
- `pause()` – Suspends execution until an external signal is received, used here to keep the script running.
  ➤ Example: `pause()`

Whole restructured code:

python Copy Edit

```python
from gpiozero import LED, Button       # Import necessary GPIO control classes
import time                            # Import time module (not used in this script)
from signal import pause               # Import pause function to keep program running

led = LED(17)                          # Define LED connected to GPIO pin 17
button = Button(26)                    # Define button connected to GPIO pin 26

button.when_pressed = led.on           # Turn LED ON when button is pressed
button.when_released = led.off         # Turn LED OFF when button is released

pause()                                # Keep the program running to listen for events
```

MULTIPLE LED'S

Summary of the code:

This Python program is designed to run on a Raspberry Pi using the `gpiozero` library. It controls three LEDs connected to GPIO pins (17, 27, and 22) through a single button connected to GPIO pin 26. Every time the button is pressed, the program cycles through turning ON one LED at a time in order: first LED1, then LED2, then LED3, and repeats in this cycle.

Steps:

1. The `LED` and `Button` classes are imported from the `gpiozero` library to interact with GPIO hardware.
2. The `pause` function is imported from the `signal` module to keep the program running indefinitely.
3. Three LED objects (`led1` , `led2` , and `led3`) are created and connected to GPIO pins 17, 27, and 22 respectively.
4. A Button object is created and connected to GPIO pin 26, with a debounce time of 0.05 seconds to avoid signal bouncing.
5. All LEDs are initially turned OFF.
6. A global variable `led_index` is initialized to keep track of which LED should be ON.
7. A function `switch_led()` is defined to handle the cycling logic:
   - If `led_index == 0` : turn ON `led1` , OFF others
   - If `led_index == 1` : turn ON `led2` , OFF others
   - If `led_index == 2` : turn ON `led3` , OFF others
8. `button.when_pressed` is bound to the `switch_led()` function.
9. `pause()` is called to keep the program running and listening to button events.

Explanation and use of word:

- `from gpiozero import LED, Button` – Imports two classes: `LED` for controlling LED lights, and `Button` for detecting button input on GPIO pins.
- `from signal import pause` – Imports the `pause()` function which stops the script from exiting, allowing it to keep responding to hardware events.
- `LED(17)` – Creates an LED object and assigns it to GPIO pin 17.

- `Button(26, bounce_time=0.05)` – Creates a Button object attached to GPIO 26 with a debounce delay of 50 milliseconds to avoid multiple readings caused by signal noise.
- `.on()` / `.off()` – Methods of the LED class to turn the LED ON or OFF.
- `global led_index` – Used to modify the `led_index` variable declared outside the function inside the function `switch_led`.
- `button.when_pressed = switch_led` – Assigns the `switch_led` function to execute when the button is pressed.
- `pause()` – Keeps the Python program alive, waiting for button presses.

Syntax of the library calls used:

- `print("text")` – Used to display a string on the terminal.
  ➤ **Example:** `print("LED ON")`
- `LED(pin_number)` – Constructor that creates an LED object attached to a specific GPIO pin.
  ➤ **Example:** `led1 = LED(17)`
- `Button(pin_number, bounce_time)` – Constructor that creates a Button object on a specified pin, optionally using `bounce_time` to handle bouncing.

  ➤ **Example:** `button = Button(26, bounce_time=0.05)`
- `led.on()` – Turns the assigned LED ON.
  ➤ **Example:** `led1.on()`
- `led.off()` – Turns the assigned LED OFF.
  ➤ **Example:** `led2.off()`
- `pause()` – Keeps the script running indefinitely until manually interrupted.
  ➤ **Example:** `pause()`
- `global variable_name` – Allows a function to modify a global variable.
  ➤ **Example:** `global led_index`
- `button.when_pressed = function_name` – Sets up a callback function that executes when the button is pressed.
  ➤ **Example:** `button.when_pressed = switch_led`

```
from gpiozero import LED, Button          # Import LED and Button classes from gpiozero
from signal import pause                  # Import pause function to keep script running

led1 = LED(17)                            # Create LED object for GPIO 17
led2 = LED(27)                            # Create LED object for GPIO 27
led3 = LED(22)                            # Create LED object for GPIO 22
button = Button(26, bounce_time=0.05)     # Create Button object with debounce time

led1.off()                                # Turn OFF all LEDs initially
led2.off()
led3.off()

led_index = 0                             # Start LED index counter

def switch_led():                         # Function to switch LEDs in sequence
    global led_index
    if led_index == 0:
        led1.on()
        led2.off()
        led3.off()
        led_index += 1
    elif led_index == 1:
        led1.off()
        led2.on()
        led3.off()
        led_index += 1
    else:
        led1.off()
        led2.off()
        led3.on()
        led_index = 0

button.when_pressed = switch_led          # Bind the function to button press
pause()                                   # Keep the program running
```

PROGRAM -4

Summary of the code:

This code controls a set of three LEDs connected to a Raspberry Pi. When a button connected to the Raspberry Pi is pressed, the code switches ON one LED at a time in sequence (LED 1, then LED 2, then LED 3), turning OFF the others. After the last LED is turned ON, the sequence restarts from the first LED again. The button press cycles through the LEDs one by one.

Steps:

1. **Importing libraries:**
   The code imports `LED` and `Button` classes from the `gpiozero` library, and `pause` from the `signal` library.
2. **Initialize LEDs and button:**
   Three LED objects are created on GPIO pins 17, 27, and 22. A Button object is created on GPIO pin 26 with a debounce time of 0.05 seconds.
3. **Initialize led_index:**
   A global variable `led_index` is set to 0 to keep track of which LED should be turned ON next.
4. **Define `reset_leds` function:**
   This function turns OFF all LEDs in the `led_list`. It ensures only one LED is ON at a time.
5. **Define `switch_led` function:**
   This function is triggered when the button is pressed. It first resets all LEDs OFF, then turns ON the LED at the current `led_index`, then increments `led_index`. If `led_index` exceeds the last LED, it resets to 0 to start over.
6. **Turn all LEDs OFF initially:**
   Calls `reset_leds()` once at the start to ensure all LEDs are OFF.
7. **Assign button press event:**
   The `switch_led` function is assigned to be called each time the button is pressed.
8. **Keep the program running:**
   `pause()` is called to keep the script running and listening for button presses indefinitely.

Explanation and use of words:

- `from gpiozero import LED, Button`
  Imports the `LED` and `Button` classes from the `gpiozero` library, which is used to interact with Raspberry Pi GPIO pins easily.
- `from signal import pause`
  Imports the `pause` function from the `signal` module which blocks the program from exiting, allowing event-driven code to run indefinitely.
- `LED(pin)`
  Creates an LED object that controls an LED connected to the specified GPIO pin.
- `Button(pin, bounce_time=...)`
  Creates a Button object representing a physical button connected to a GPIO pin, with optional debounce time to avoid multiple rapid triggers.
- `led.off()`
  Turns OFF the LED.
- `led.on()`
  Turns ON the LED.
- `button.when_pressed = switch_led`
  Assigns the `switch_led` function to be called automatically when the button is physically pressed.
- `global led_index`
  Allows the function to modify the variable `led_index` that is declared outside the function scope.
- `pause()`
  Halts the program execution and waits for events (like button presses) so the program doesn't terminate immediately.

Syntax of the library calls used:

- `LED(pin_number)`
  Creates an LED object attached to the specified GPIO pin.
  Example: `led = LED(17)`
- `Button(pin_number, bounce_time=seconds)`
  Creates a Button object attached to a GPIO pin with an optional bounce time parameter to ignore rapid toggling.
  Example: `button = Button(26, bounce_time=0.05)`
- `led.on()`
  Turns the LED ON.
- `led.off()`
  Turns the LED OFF.
- `button.when_pressed = function_name`
  Assigns a callback function that is executed when the button is pressed.
- `pause()`
  Pauses the program indefinitely, useful for waiting on events in an event-driven program.

Copy Edit

```
from gpiozero import LED, Button
from signal import pause

# Initialize three LEDs connected to GPIO pins 17, 27, and 22
led_list = [LED(17), LED(27), LED(22)]

# Initialize a button connected to GPIO pin 26, with a debounce time of 0.05 seconds
button = Button(26, bounce_time=0.05)

# Index to track which LED to switch on next
led_index = 0

def reset_leds():
    # Turn OFF all LEDs in the list
    for led in led_list:
        led.off()

def switch_led():
    global led_index  # Use the global variable to update its value
    reset_leds()      # Turn off all LEDs before turning one ON
    led_list[led_index].on()  # Turn ON the current LED
    led_index += 1
    # Reset the index to 0 if it reaches the end of the LED list
    if led_index >= len(led_list):
        led_index = 0

# Initially turn off all LEDs
reset_leds()

# Bind the switch_led function to be called when the button is pressed
button.when_pressed = switch_led

# Keep the script running to listen for button presses
pause()
```

Summary of the code:
This Python script uses the `gpiozero` library to control an LED connected to a Raspberry Pi based on motion detected by a PIR (Passive Infrared) motion sensor. When motion is detected, the LED turns on, and when no motion is detected, the LED turns off. The program runs continuously until manually stopped.

Steps:

1. **Import Libraries:**
   - `from gpiozero import LED, MotionSensor` imports two classes: `LED` to control an LED, and `MotionSensor` to interact with a PIR sensor.
   - `from signal import pause` imports the `pause` function, which keeps the program running and waiting for events without exiting.
2. **Create Objects:**
   - `led = LED(17)` creates an LED object connected to GPIO pin 17 on the Raspberry Pi.
   - `pir = MotionSensor(4)` creates a MotionSensor object connected to GPIO pin 4.
3. **Define Behavior on Motion:**
   - `pir.when_motion = led.on` sets an event handler so when the PIR sensor detects motion, the LED's `on` method is called (turning the LED on).
   - `pir.when_no_motion = led.off` sets an event handler so when the PIR sensor stops detecting motion, the LED's `off` method is called (turning the LED off).
4. **Keep the Script Running:**
   - `pause()` keeps the script running indefinitely, allowing the motion sensor to continuously detect and trigger the LED without the script exiting.

Explanation and use of words:

- **gpiozero:** A Python library designed to simplify controlling GPIO pins on Raspberry Pi.
- **LED:** A class in gpiozero representing an LED connected to a GPIO pin. Controls turning the LED on and off.
- **MotionSensor:** A class in gpiozero representing a PIR motion sensor to detect motion.
- **from ... import ...:** Python syntax to import specific classes or functions from a module or library.
- **led = LED(17):** Creates an instance of the LED class, connected to GPIO pin 17. This object lets you control the physical LED.
- **pir = MotionSensor(4):** Creates an instance of the MotionSensor class on GPIO pin 4, to detect motion events.
- **pir.when_motion:** An event handler attribute of the MotionSensor object. Assigning a function here defines what happens when motion is detected.
- **led.on:** A method of the LED object to turn the LED on.
- **pir.when_no_motion:** An event handler attribute defining what happens when motion stops.
- **led.off:** A method of the LED object to turn the LED off.
- **pause():** A function imported from the `signal` module that blocks program exit and keeps it running until interrupted, useful for event-driven programs.

Syntax of the library calls used:

- `from module import class_or_function` — Imports specific classes or functions from a module.
- `LED(pin_number)` — Creates an LED object connected to the specified GPIO pin number.
- `MotionSensor(pin_number)` — Creates a MotionSensor object connected to the specified GPIO pin number.
- `object.method` — Calls a method on an object; e.g., `led.on` turns the LED on.
- `object.attribute = function_or_method` — Assigns a callback function or method to an event attribute; e.g., `pir.when_motion = led.on` assigns the LED's `on` method as the callback when motion is detected.
- `pause()` — Keeps the program running indefinitely to listen for asynchronous events.

python

Copy

```python
from gpiozero import LED, MotionSensor
from signal import pause

led = LED(17)               # Initialize LED on GPIO pin 17
pir = MotionSensor(4)       # Initialize PIR sensor on GPIO pin 4

# Define actions when motion is detected or not detected
pir.when_motion = led.on
pir.when_no_motion = led.off

pause()                     # Keep the program running to listen for events
```

```python
from gpiozero import LED
import time

led = LED(17)

try:
    while True:
        led.on()
        time.sleep(2)
        led.off()
        time.sleep(1)
except KeyboardInterrupt:
    pass
```

PYTHON - CAMERA

Summary of the code:
This Python script uses the `picamzero` library to control a Raspberry Pi camera module. It creates a directory (if not already existing) to save photos, initializes the camera with specific settings, and then continuously takes photos every 5 seconds, saving each with an incrementing filename. The process runs indefinitely until interrupted by the user.

Steps:

1. **Import Libraries:**
   - `import os` to interact with the operating system, such as checking for and creating directories.
   - `from picamzero import Camera` imports the Camera class for controlling the Pi camera module.
   - `import time` for adding delays/sleep between actions.
2. **Create Folder if Needed:**
   - Define a folder path string `folder_name = "/home/pi/camera/activity_10"`.
   - Check if this folder exists with `os.path.exists()`.
   - If it does not exist, create the folder using `os.mkdir()`.
3. **Initialize Camera:**
   - Create a Camera object using `camera = Camera()`.
   - Set the still image resolution to 1536x864 pixels using `camera.still_size = (1536, 864)`.
   - Flip the camera image horizontally and vertically with `camera.flip_camera(hflip=True, vflip=True)`.
   - Pause 2 seconds with `time.sleep(2)` to allow the camera to stabilize.
   - Print a message indicating initialization is complete.

4. **Take Photos Continuously:**
    - Initialize a counter variable `counter = 0` to track image filenames.
    - Use a `try` block with an infinite loop (`while True:`) to:
        - Construct a filename combining folder path + "image" + counter + ".jpg".
        - Take a photo and save it with `camera.take_photo(file_name)`.
        - Print a confirmation message.
        - Increment the counter by 1.
        - Wait for 5 seconds before the next photo.
5. **Graceful Exit:**
    - Catch a KeyboardInterrupt (Ctrl+C) to stop the loop and print "End".

Explanation and use of words:

- **os:** A standard Python library module for interacting with the operating system (file system, directories).
- **picamzero:** A Python library to control the Raspberry Pi camera module easily.
- **Camera:** A class from `picamzero` used to initialize and control the camera.
- **import:** Keyword used to bring modules or specific classes/functions into the current script.
- **folder_name:** A variable holding the path where images will be saved.
- **os.path.exists(path):** Function that returns `True` if the given path exists, else `False`.
- **os.mkdir(path):** Function to create a new directory at the specified path.
- **camera = Camera():** Instantiates the Camera object, enabling control of the hardware camera.
- **camera.still_size = (width, height):** Sets the resolution of captured still images.
- **camera.flip_camera(hflip=True, vflip=True):** Flips the camera image horizontally and vertically.
- **time.sleep(seconds):** Pauses program execution for the given number of seconds.
- **print("text"):** Outputs the given text to the console/terminal.
- **counter:** A variable to number the image files uniquely.
- **while True:** An infinite loop that runs until externally interrupted.
- **camera.take_photo(filename):** Captures an image and saves it to the given filename.
- **try-except KeyboardInterrupt:** A block that allows graceful program termination when the user presses Ctrl+C.

Syntax of the library calls used:

- `os.path.exists(path)` — Checks if a file or directory exists at `path`. Returns `True` or `False`.
- `os.mkdir(path)` — Creates a directory at the specified `path`.
- `Camera()` — Creates a new camera object to interface with the Raspberry Pi camera module.
- `camera.still_size = (width, height)` — Sets still image capture resolution.
- `camera.flip_camera(hflip=bool, vflip=bool)` — Flips camera image horizontally and/or vertically based on boolean flags.
- `camera.take_photo(filename)` — Captures and saves a photo to the provided filename.
- `time.sleep(seconds)` — Suspends execution for a given number of seconds.
- `print(string)` — Displays a string or message on the console.
- `try: ... except KeyboardInterrupt:` — Tries to run code and catches interruption signals for clean exit.

If you'd like, I can also explain the commented out line `camera.capture_sequence(...)` or any other part in more detail!

```python
import os
from picamzero import Camera
import time

folder_name = "/home/pi/camera/activity_10"

# Check if folder exists; if not, create it
if not os.path.exists(folder_name):
    os.mkdir(folder_name)

# Initialize the camera and configure settings
camera = Camera()
camera.still_size = (1536, 864)
camera.flip_camera(hflip=True, vflip=True)
time.sleep(2)  # Wait for camera to initialize

print("Camera has been initialized")

counter = 0

try:
    while True:
        file_name = folder_name + "/image" + str(counter) + ".jpg"
        camera.take_photo(file_name)
        print("New photo has been taken")
        counter += 1
        time.sleep(5)  # Wait 5 seconds before next photo
except KeyboardInterrupt:
    print("End")
```

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.

Explanation and use of word:

- `from flask import Flask` – Imports the `Flask` class from the `flask` library, which is used to build web applications.
- `from gpiozero import Button, LED` – Imports `Button` and `LED` classes from the `gpiozero` library to control Raspberry Pi GPIO hardware.
- `Button(26, bounce_time=0.05)` – Creates a button object connected to GPIO pin 26 with a debounce time to avoid signal noise.
- `LED(17)` – Creates an LED object connected to GPIO pin 17.
- `led.off()` / `led.on()` – Turns the LED OFF or ON, respectively.
- `Flask(__name__)` – Creates a Flask application instance.
- `@app.route(...)` – A Flask decorator that binds a URL to a specific function.
- `button.is_pressed` – Returns `True` if the button is pressed.
- `app.run(host="0.0.0.0")` – Runs the Flask server, making it available on all network interfaces.

Syntax of the library calls used:

- `Flask(__name__)` – Initializes the Flask application. `__name__` helps Flask locate resources.
- `@app.route("/url")` – Flask decorator to define routes that respond to HTTP requests.
- `Button(pin, bounce_time=x)` – Declares a button on a specific GPIO pin with optional debounce time to avoid false triggers.
- `LED(pin)` – Creates an LED object connected to a specific GPIO pin.
- `led.on()` – Turns ON the LED.
- `led.off()` – Turns OFF the LED.
- `button.is_pressed` – Boolean value indicating whether the button is currently being pressed.
- `app.run(host="0.0.0.0")` – Starts the Flask server and allows access from any IP address on the network.

RASPBERRY PI –Project Setup

Explanation and use of word:

- `from gpiozero import MotionSensor, LED` : Imports classes to control GPIO devices like motion sensors and LEDs.
- `from picamzero import Camera` : Imports the `Camera` class to control Raspberry Pi camera module.
- `import yagmail` : Imports the library for sending emails via Gmail.
- `from signal import pause` : Keeps the program running, waiting for signals (like motion).
- `import time` : Enables timing functions like delay and timestamps.
- `import os` : Provides access to OS functions like file creation, deletion, and checking existence.
- `def` : Used to define a function.
- `global` : Declares global scope for variables inside functions.
- `with open() as f` : Opens a file safely and closes it automatically after use.
- `os.path.exists(path)` : Checks if a file/folder exists.

- `os.mkdir(path)` : Creates a new folder.
- `time.time()` : Returns the current time in seconds since epoch.
- `time.sleep(seconds)` : Pauses execution for the specified time.
- `f.write()` : Writes data to a file.
- `yagmail.SMTP(user, password)` : Initializes a Gmail client with credentials.
- `yagmail_client.send()` : Sends an email with specified content and attachment.
- `pir.when_motion` : Sets the function to be called when motion is detected.
- `pir.when_no_motion` : Sets the function to be called when motion stops.
- `led.on() / led.off()` : Turns the LED on or off.

give some space

Syntax of the library calls used:

- `print("text")` - Used to display messages to the console.
- `def function_name(params):` - Declares a function.
- `with open("file.txt", "r") as f:` - Opens a file in read mode, with automatic closure.
- `f.write("data")` - Writes data to the file.
- `os.path.exists("path")` - Returns True if the file or directory exists.
- `os.mkdir("folder_path")` - Creates a new folder at specified path.
- `time.time()` - Returns current time in seconds.

- `time.sleep(seconds)` - Halts execution for the given number of seconds.
- `MotionSensor(pin)` - Initializes a motion sensor connected to the specified GPIO pin.
- `LED(pin)` - Initializes an LED on the given GPIO pin.
- `Camera()` - Creates a Camera object to capture images.
- `camera.take_photo("file_path")` - Takes a picture and saves it to the given path.
- `yagmail.SMTP("email", "password")` - Connects to Gmail SMTP server with credentials.
- `yagmail.send(to, subject, contents, attachments)` - Sends an email with the specified fields.
- `pause()` - Keeps the script running indefinitely, waiting for events (like GPIO triggers).

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.

python Copy Edit

```python
from gpiozero import MotionSensor, LED
from picamzero import Camera
import yagmail
from signal import pause
import time
import os

# ---------------------- Helper Functions ---------------------- #

def take_photo(camera, folder_path):
    file_name = os.path.join(folder_path, f"img_{time.time()}.jpg")
    camera.take_photo(file_name)
    return file_name

def update_photo_log_file(log_file_name, photo_file_name):
    with open(log_file_name, "a") as f:
        f.write(photo_file_name + "\n")

def send_photo_by_email(yagmail_client, file_name):
    yagmail_client.send(
        to="your_email_address",
        subject="New photo from Raspberry Pi",
        contents="Check out the new photo",
        attachments=file_name
    )
```

```
MOVEMENT_DETECTED_THRESHOLD = 5.0
MIN_DURATION_BETWEEN_PHOTOS = 30.0
CAMERA_FOLDER_PATH = "/home/pi/photos_final_project"
LOG_FILE_NAME = os.path.join(CAMERA_FOLDER_PATH, "photo_logs.txt")

# ---------------------- State Variables ---------------------- #

time_motion_started = time.time()
last_time_photo_taken = 0

# ---------------------- Email Setup ---------------------- #

with open("/home/pi/.local/share/.email_password", "r") as f:
    password = f.read().strip()

yag = yagmail.SMTP("sender_email_address", password)
print("Email setup OK")

# ---------------------- Camera Setup ---------------------- #

camera = Camera()
camera.still_size = (1536, 864)
camera.flip_camera(vflip=True, hflip=True)
time.sleep(2)
```

```
if not os.path.exists(CAMERA_FOLDER_PATH):
    os.makedirs(CAMERA_FOLDER_PATH)

print("Camera setup OK")

# ---------------------- Log File Reset ----------

if os.path.exists(LOG_FILE_NAME):
    os.remove(LOG_FILE_NAME)
    print("Previous log file removed")

# ---------------------- GPIO Setup --------------

pir = MotionSensor(4)
led = LED(17)
print("GPIOs setup OK")

# ---------------------- Motion Handlers ---------

def motion_detected():
    global time_motion_started
    time_motion_started = time.time()
    led.on()
```

```
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)
```

```
# ---------------------- Event Bindings --------------

pir.when_motion = motion_detected
pir.when_no_motion = motion_finished

print("Everything has been setup.")
pause()
```

FINAL CODE

Summary of the code:

This code sets up a simple Flask web server that allows users to check how many new photos have been taken and stored by a Raspberry Pi motion detection system since their last visit. It reads from a photo log file, calculates how many new entries (photos) were added, and displays the most recent photo directly in the browser.

Steps:

1. Import the necessary libraries: `Flask` for web application handling, and `os` for file system access.
2. Define constants: `CAMERA_FOLDER_PATH` for photo storage and `LOG_FILE_NAME` for the photo log.
3. Initialize a counter variable `previous_line_counter` to track the number of photos previously recorded.
4. Create a Flask application object.
5. Define the root route `/` that simply returns "Hello".
6. Define the `/check-photos` route that:
    - Opens the log file.
    - Counts how many lines (photos) are there.
    - Calculates how many new photos have been added since last check.
    - Displays this count and the most recent photo using an HTML image tag.

6. Define the `/check-photos` route that:
    - Opens the log file.
    - Counts how many lines (photos) are there.
    - Calculates how many new photos have been added since last check.
    - Displays this count and the most recent photo using an HTML image tag.
7. Start the Flask server on all network interfaces (`0.0.0.0`).

Explanation and use of words:

- `from flask import Flask` : Imports the `Flask` class from the Flask web framework used to create the app.
- `import os` : Imports Python's `os` module for interacting with the operating system.
- `Flask(...)` : Initializes the Flask application with given static file settings.
- `@app.route(...)` : Flask route decorator that defines a URL path for a function to handle.
- `global previous_line_counter` : Declares the variable as global so it can be updated inside the function.
- `open(...)` : Opens a file for reading or writing.
- `os.path.exists(...)` : Checks whether a file or directory exists.
- `line.strip()` : Removes newline characters from the end of each line.
- `f"... {variable} ..."` : f-string used for string interpolation.
- `app.run(host="0.0.0.0")` : Starts the Flask web server on all IP addresses of the machine (useful for access from other devices on the network).

Syntax of the library calls used:

- `Flask(__name__)` – Initializes a Flask application object.
- `@app.route("/path")` – Associates a URL path with a Python function.
- `open(filename, mode)` – Opens a file in specified mode like "r" for read, "a" for append.
- `os.path.exists(path)` – Returns `True` if the file or directory at `path` exists.
- `line.strip()` – Removes whitespace characters from the beginning and end of the string.
- `f"text {var}"` – An f-string used to include variables inside strings.
- `app.run(host="0.0.0.0")` – Launches the Flask development server on all network interfaces.

```
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")
```