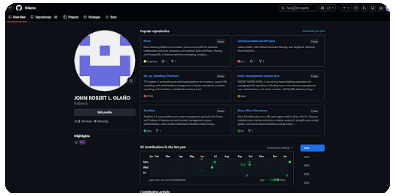
GitHub Portfolio Review and Assistance

Model: Gemini 3.0 pro
https://gemini.google.com/share/6625ffd03706



Objectives:
- Compile all activities done in the course into one GitHub repository.

Instructions:
- Compile all laboratory activities developed in the subject COSC 111. Please ensure the following:
1. All activities are in separated folders.
2. Create a README.md file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Submission Requirement:
- Your GitHub URL (make sure that you are using a professional username handle)

Grading:
Content: 50%
Organization: 20%
Application of Concepts: 20%
Readability: 10%

how to change the username?

Based on the "Submission Requirement" in your screenshot, it looks like you want to ensure your handle is professional before submitting your GitHub URL.

Here is a step-by-step guide on how to change your GitHub username, along with some important things to keep in mind.

How to Change Your Username

1. **Log in** to your GitHub account.
2. Click on your profile photo in the top-right corner and select **Settings**.
3. In the left sidebar, click on **Account** (it is usually under the "Access" section).
4. Scroll down to the "Change username" section.
5. Click the **Change username** button.
6. Read the warnings, then follow the prompts to enter a new name.

⚠️ Important Warnings Before You Change

Changing your username changes your profile URL (e.g., from `github.com/R4berto` to `github.com/JohnOlano`). Here is how it affects your work:

- **Repository URLs:** GitHub will automatically redirect traffic from your old repository URLs to the new ones, but you should update any links you have explicitly shared (like in a resume or LMS submission).
- **Local Git Settings (Crucial):** Your local files on your computer won't automatically know about the name change. You might encounter errors when trying to `git push` or `git pull`.
  - **The Fix:** You will need to update the "remote URL" for your repositories on your computer.
  - Command: `git remote set-url origin https://github.com/[NEW-USERNAME]/[REPO-NAME].git`

Tips for a Professional Username

Since the assignment explicitly asks for a "professional username handle," try to use variations of your real name rather than a gaming nickname. Based on your profile name (**John Robert L. Olaño**), here are some professional suggestions:

- `john-olano`
- `johnrobert-olano`
- `jr-olano`
- `jrolano-dev`

Would you like help constructing the README.md file or structuring the folders for your COSC 111 compilation repository?

can you create a professional username, my name is john robert L Olaño

Here are several professional username options based on **John Robert L. Olaño**.

Since GitHub URLs and terminal commands work best with standard characters, it is highly recommended to use **"n"** instead of **"ñ"** to avoid encoding errors or difficulty for others trying to clone your repos.

Clean & Direct (Best for CVs)

- `JohnOlano`
- `John-Olano`
- `JohnRobertOlano`
- `OlanoJohn`

Concise (Easier to type)

- `JROlano`
- `JRLOlano`
- `JohnRobOlano`
- `J-Olano`

Developer Focused (Good if the others are taken)

- `JohnOlanoDev`
- `JohnCodes`
- `OlanoTech`
- `JROlano-CS` (Computer Science)

A Note on Availability

Since "John" is a common name, your first choice might be taken. If that happens, try:

1. Adding a middle initial (e.g., `JohnLOlano`).
2. Using underscores or hyphens (e.g., `John_Olano` or `John-Olano`).
3. Adding a short number at the end, ideally related to a date, though purely text is often seen as more "senior" (e.g., `JohnOlano01`).

Would you like me to draft the README.md for your COSC 111 compilation once you have settled on a username?

FINAL PROJECT: Student Portfolio for COSC 111 but add my name Olaño, this for repository name

Here are a few professional options for your repository name.

Best Practice Note: Even though your name is Olaño, it is highly recommended to use **"Olano"** (with an n) for the repository name. Using special characters like "ñ" in file paths and URLs can sometimes cause technical issues with Git commands or cloning.

Top Recommendations

These follow standard GitHub naming conventions (`kebab-case`) which are easy to read and link to.

- `COSC111-Student-Portfolio-Olano` (Clear and descriptive)
- `Olano-COSC111-Final-Project` (puts your name first for easy sorting)
- `COSC111-Laboratory-Activities-Olano` (Specific to the content)

How to Rename it on GitHub

If you have already created the repo with a different name:

1. Go to your repository's **Settings** tab.
2. The first option is **Repository Name**.
3. Type the new name and click **Rename**.

Since the assignment requires a `README.md` file to explain the activities, would you like me to generate a template for that file now?

try to make it know that this is portfolio

Here are the best repository name options that clearly highlight it as a **Portfolio** while keeping it professional.

Recommendation: For the repository *name* (which becomes the URL), use **"Olano"** (no ñ) to ensure the link works perfectly on all computers. You can use the "ñ" inside the README title instead.

Option 1: Clear & Standard (Best for Grading)

This format is the easiest for your professor to read. It tells them exactly what it is, for which subject, and who you are.

- `COSC111-Student-Portfolio-Olano`
- `COSC111-Portfolio-JohnOlano`

Option 2: Name-First (Good for Personal Branding)

This emphasizes *you* as the developer.

- `Olano-COSC111-Portfolio`

- `JohnOlano-COSC111-Portfolio`

Option 3: Descriptive (If you want to be specific)

- `Portfolio-COSC111-Laboratory-Olano`

Quick Check: Does this meet the requirements?

- **Organization (20%):** Yes, these names are organized and follow a logical hierarchy.
- **Professionalism:** Yes, they use standard "kebab-case" (using hyphens) which is the standard for professional developers.

Once you create the repository, would you like a template for the `README.md` file that meets the "Content" and "Readability" grading criteria?

create a description

Here are two types of descriptions you will need.

1. The "About" Section (Short)

This goes in the top-right sidebar of your GitHub repository page (look for the "About" gear icon).

> **Option A (Professional & Direct):** A compilation of all laboratory activities and coding projects developed for the subject COSC 111. Created by John Robert L. Olaño.
>
> **Option B (Skill-Focused):** Student portfolio for COSC 111 demonstrating proficiency in [Insert Language, e.g., C++/Java] programming concepts, algorithms, and logic formulation.

---

2. The `README.md` Introduction (Detailed)

This goes inside the actual `README.md` file. It addresses the "Content" (50%) and "Readability" (10%) grading criteria by explaining exactly what the repository is.

Copy and paste the code block below into your file:

Markdown

```
# COSC 111 Student Portfolio

### 👤 Student Information
* **Name:** John Robert L. Olaño
* **Subject:** COSC 111 (Introduction to Computing)
* **Instructor:** [Insert Instructor Name]

---

### 📁 Repository Overview
Welcome to my final project portfolio. This repository serves as a centralized compilation of all laboratory activities and coding challenges completed during the COSC 111 course.

The goal of this portfolio is to demonstrate my understanding of core programming concepts, including:
* Input/Output manipulation
* Control structures (Loops, If-Else statements)
* Data structures (Arrays, Strings)
* Functions and modular programming

### 🚀 How to Navigate
Each folder in this repository corresponds to a specific laboratory activity. Inside each folder, you will find:
1.  **Source Code:** The raw `.cpp` / `.java` / `.py` files.
2.  **Activity README:** A specific explanation of the problem solved in that activity.
```

Would you like me to write the "Activity README" template for the individual folders next? (This is required by instruction #2 in your screenshot).

this is IoT

Thanks for that clarification. Since COSC 111 is an **Internet of Things (IoT)** course, your descriptions and README need to highlight hardware, microcontrollers (like Arduino/ESP32), and circuit design rather than just pure code.

Here are the updated descriptions tailored specifically for an IoT portfolio.

1. The "About" Section (Short)

Paste this into the repository sidebar description.

> **Option A (Professional):** A comprehensive portfolio of IoT laboratory activities, circuit designs, and microcontroller source code for COSC 111. Created by John Robert L. Olaño.
>
> **Option B (Technical):** Student portfolio for COSC 111 showcasing Internet of Things (IoT) projects, including sensor integration, Arduino programming, and serial communication protocols.

---

2. The `README.md` Introduction (IoT Focused)

This template addresses the "Content" (50%) grading criteria by explicitly mentioning the hardware and concepts involved.

Copy and paste the code block below into your file:

Markdown

```
# COSC 111: Internet of Things (IoT) Portfolio

### 👤 Student Information
* **Name:** John Robert L. Olaño
* **Subject:** COSC 111 (Internet of Things)
* **Instructor:** [Insert Instructor Name]

---

### 📁 Repository Overview
Welcome to my final project portfolio. This repository serves as a centralized compilation of all laboratory activities and prototyping projects completed during the COSC 111 course.

This portfolio demonstrates my technical competency in **Embedded Systems** and **IoT development**, specifically focusing on:
* **Microcontroller Programming:** Writing efficient C++/Arduino code for hardware control.
* **Circuit Design:** Prototyping circuits using breadboards, resistors, and various actuators.
* **Sensor Integration:** Interfacing analog and digital sensors (e.g., Thermistors, Photoresistors) to read environmental data.
* **Communication Protocols:** Implementing Serial communication for data transfer.

### 🚀 How to Navigate
Each folder in this repository corresponds to a specific laboratory activity. Inside each folder, you will generally find:
1.  **Source Code:** The `.ino` or `.cpp` sketch files.
2.  **Schematics/Diagrams:** Images or Fritzing files showing the circuit wiring.
3.  **Activity Documentation:** A README explaining the logic and components used.

---
### 🛠️ Hardware & Tools Used
* **Microcontroller:** [e.g., Arduino Uno / ESP32]
* **Sensors:** [e.g., Temperature, Light, Motion Sensors]
* **Actuators:** [e.g., LEDs, Servos, Motors]
* **IDE:** Arduino IDE / PlatformIO
```

Instruction #2 in your assignment says: *"Create a README.md file that describes each activity per folder."*

Would you like me to write a **template for those specific folder READMEs**? (e.g., A template where you just fill in "Components Used" and "How it Works").

Bitanga_Ol...roject.ino
Unknown

README.md before:

Laboratory Activity #1: Working with Digital Signals

Objectives:
1. Review Arduino as a device for IoT systems implementation
2. Discuss digital signals and its implementation in a Arduino circuit.

Instructions:
1. We are instructed to create a running light circuit using Arduino
- We used pins 8 to 12 for the LEDs
- We made it ran light from 12 to 8 with a delay of 1 second (turn all LED on one by one, then turn all LED off one by one)
- We used digitalWrite() to control the LEDs

2. This github link contains the following:
- Bitanga_Olano_Paciente_LEDSequentialControlProject.ino
- Breadboard diagram. (http://bit.ly/47Mf65Y)
<img width="1707" height="728" alt="Bitanga_Olano_Paciente_Breadboard_diagram" src="https://github.com/user-attachments/assets/fadaed9f-2c4a-4ffb-bbb9-4726fa6e4a3c" />
- Individual grades of the members

change the README.md before, make it a README.md file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the updated `README.md` file. It follows a professional structure, explicitly details the hardware and code logic, and incorporates the image provided.

📄 **New README.md Content**

Markdown

```
# Laboratory Activity 1: Digital Output & Sequential Control

### 📌 Overview
This project demonstrates the basics of **digital output signals** in an IoT system. The goal was to create a "running light" circuit where five LEDs turn on and off in a specific sequence using an Arduino microcontroller. This activity reinforces the understanding of pin configuration, voltage levels (HIGH/LOW), and timing delays.

---

### 🎯 Objectives
* Review Arduino board architecture for IoT implementation.
* Understand the application of digital signals (`HIGH` vs `LOW`).
* Write firmware to control multiple output devices sequentially.

---
```

~~~
### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Actuators:** 5x LEDs (Green, Yellow, Orange, Blue, Red)
* **Passive Components:** 5x Resistors (Current limiting), Breadboard, Jumper Wires

### 🔌 Circuit Wiring
The LEDs are connected to the digital pins of the Arduino as follows:
| LED Color | Arduino Pin |
| :--- | :--- |
| **Green** | Pin 12 |
| **Yellow** | Pin 11 |
| **Orange** | Pin 10 |
| **Blue** | Pin 9 |
| **Red** | Pin 8 |

> **Note:** All LEDs are connected in series with a resistor to the ground (GND) rail of the breadboard to prevent burning out the components.

![Bitanga_Olano_Paciente_Breadboard_diagram](https://github.com/user-attachments/assets/fadaed9f-2c4a-4ffb-bbb9-4726fa6e4a3c)

---

### 💻 Code Explanation
The firmware is written in C++ (Arduino Sketch). Here is how the logic functions:

#### 1. Setup Phase
[cite_start]Inside the `setup()` function, we configure the specific digital pins (8 through 12) to behave as **OUTPUT**s[cite: 2, 3]. This allows the Arduino to send voltage to the LEDs.

#### 2. The Main Loop
The `loop()` function executes the following sequence indefinitely:
* [cite_start]**Turn ON Sequence:** The code sets the `Green` LED (Pin 12) to `HIGH` (5V), waits for 1000 milliseconds (1 second), and then proceeds to the next LED (`Yellow`, `Orange`, etc.) until all lights are illuminated[cite: 4, 5].
* [cite_start]**Turn OFF Sequence:** Once all LEDs are lit, the code begins turning them off in the same order (starting from Green/Pin 12) with a 1-second delay between each step[cite: 6, 7].

```cpp
// Snippet of the control logic
digitalWrite(Green, HIGH); // Turns on Green LED
delay(1000);               // Waits 1 second
digitalWrite(Yellow, HIGH);// Turns on Yellow LED
~~~

📁 File Structure

- `Bitanga_Olano_Paciente_LEDSequentialControlProject.ino` : The main source code containing the logic for the sequential light pattern.
- `Bitanga_Olano_Paciente_Breadboard_diagram.png` : Visual representation of the circuit connections.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

can you give me also folder name for this

For this specific folder, I recommend a name that includes both the **activity number** (so it sorts correctly in your portfolio) and a **short description** of what it does.

Here are the best professional options:

Option 1: Descriptive & Organized (Recommended)

- `Activity-01-LED-Sequential-Control`
- `Lab-01-Running-Lights`

Option 2: Technical Focus

- `Activity-01-Digital-Output-Signals`
- `Lab-01-Arduino-LED-Sequence`

💡 Pro Tip for Portfolios:

Whichever style you choose, **stick to it** for the rest of the folders (e.g., if you choose `Activity-01...`, make sure the next one is `Activity-02...`). This consistency is what gets you full points for the "Organization" criteria.

Would you like me to create the README content for your next activity (the fire sensor) as well?

Bitanga_Ol...Analog.ino
Unknown

README
MD

give me a folder name and change the README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the updated `README.md` for your second activity.

📁 Recommended Folder Name

To keep your portfolio consistent with the first activity, use this naming convention:

- `Activity-02-Analog-Output-Fading`

📄 New README.md Content

Copy and paste the following into the README file inside your new folder.

Markdown

~~~markdown
# Laboratory Activity 2: Analog Output & PWM Control

### 📌 Overview
This project explores the concept of **Analog Signal generation** using Pulse Width Modulation (PWM). Unlike the first activity which used purely digital signals (ON/OFF), this activity uses `analogWrite()` to create a "breathing" or fading effect on the LEDs.

This demonstrates how a microcontroller can simulate analog voltages to control the brightness of an LED or the speed of a motor.

---

### 🎯 Objectives
* Understand the difference between Digital (`HIGH`/`LOW`) and Analog (`0-255`) output.
* Implement **PWM (Pulse Width Modulation)** to control LED brightness.
* Optimize code using **Arrays** and **While Loops** instead of writing repetitive lines.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Actuators:** 5x LEDs (Green, Yellow, Orange, Blue, Red)
* **Passive Components:** 5x Resistors (220Ω), Breadboard, Jumper Wires

### 🔌 Circuit Wiring (PWM Pins)
**Crucial Note:** To achieve the fading effect, we moved the LEDs to **PWM-enabled pins** (denoted by the `~` symbol on the Arduino). Standard digital pins (like 8 and 12) cannot fade.

| LED Position | Variable Name | Actual Arduino Pin |
| :--- | :--- | :--- |
| **LED 1** | `p12` | **Pin 6 (PWM)** |
| **LED 2** | `11` | **Pin 11 (PWM)** |
| **LED 3** | `10` | **Pin 10 (PWM)** |
| **LED 4** | `9` | **Pin 9 (PWM)** |
| **LED 5** | `p8` | **Pin 5 (PWM)** |

![Bitanga_Olano_Paciente_LAB_ACT_2_Working_with_Analog_Signals](https://github.com/user-attachments/assets/90d4d443-38c4-4cff-8a8d-56da4da35b5f)

---

### 💻 Code Explanation
The firmware introduces advanced programming concepts to handle the logic more efficiently.

#### 1. Arrays & Setup
Instead of declaring 5 separate variables, we use an array `LedPINS[]`. This allows us to iterate through pins using a loop.
```cpp
int p8=5;  // Mapped to PWM Pin 5
int p12=6; // Mapped to PWM Pin 6
int LedPINS[] = {p12, 11, 10, 9, p8}; // Array of PWM pins

void setup() {
  int i = 0;
  while (i < 5) {
    pinMode(LedPINS[i], OUTPUT); // Sets all pins in the array to OUTPUT
    i++;
  }
}
~~~

2. The Fading Logic (Nested Loops)

The `loop()` function uses a **nested while loop**.

- **Outer Loop:** Selects which LED to light up (from the array).
- **Inner Loop:** Gradually increases brightness from 0 to 255 (Max).

Fading IN:

C++

```cpp
while (brightness <= 255) {
  analogWrite(LedPINS[x], brightness); // Write PWM value
  delay(5); // Small delay creates the smooth fading animation
  brightness++;
}
```

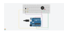
Fading OUT: After the LED is fully bright, a second loop reverses the process, decreasing `brightness` from 255 back to 0.

📁 File Structure

- `Bitanga_Olano_Paciente_LEDSequentialControlProjectUsingAnalog.ino` : The source code containing the PWM and Array logic.
- `Breadboard_diagram.png` : Visual representation of the circuit wiring.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**



README
MD

fire_senso...lation.ino
Unknown

give me a folder name and change the README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the updated `README.md` for your third activity.

📁 Recommended Folder Name

To maintain consistency with your previous folders (`Activity-01...`, `Activity-02...`), I recommend:

- `Activity-03-Fire-Sensor-System`

📄 New README.md Content

Copy and paste the following into the README file inside your new folder.

Markdown

~~~
# Laboratory Activity 3: Working with Sensors (Fire Detection)

### 📌 Overview
This project demonstrates the integration of analog input sensors into an IoT system. The goal was to build a functional **Fire Sensor System** that simultaneously monitors environmental temperature and light levels.

When both the heat and light intensity exceed specific safety thresholds (indicating a potential fire), the system triggers an audiovisual alarm using an LED and a Buzzer.

---

### 🎯 Objectives
* Familiarize with basic IoT sensor components (Thermistor & Photoresistor).
* Integrate analog sensors into an Arduino circuit.
* Implement conditional logic to create a safety monitoring system.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Sensors:**
    * **Thermistor:** Measures temperature changes.
    * **Photoresistor (LDR):** Measures light intensity.
* **Actuators:**
    * **Red LED:** Visual alarm.
    * **Piezo Buzzer:** Audio alarm.
* **Passive Components:** Resistors (10kΩ for voltage dividers), Breadboard, Jumper Wires.

### 🔌 Circuit Wiring
The circuit uses two analog input pins for the sensors and one digital output pin for the combined alarm system.

| Component | Pin Type | Arduino Pin | Description |
| :--- | :--- | :--- | :--- |
| **Thermistor** | Analog Input | **A0** | Reads temperature data. |
| **Photoresistor** | Analog Input | **A2** | Reads light brightness. |
| **LED + Buzzer** | Digital Output | **Pin 12** | Triggers the alarm signal. |

![Diagram for Lab Act #3 - Bitanga, Olaño, Paciente](https://github.com/user-attachments/assets/184852f4-cc00-4786-8968-07b949219f07)

---

### 💻 Code Explanation
The firmware is designed with modular functions and uses `const` and `#define` for easy calibration.

#### 1. Constants & Thresholds
We define specific values that determine when the fire alarm should trigger.
```cpp
#define THERMISTOR A0
#define PHOTORESISTOR A2
#define ALERT 12

const int TEMP_THRESHOLD = 50;   // Trigger if temp is >= 50°C
const int BRIGHT_THRESHOLD = 220; // Trigger if brightness >= 220
~~~

2. Sensor Functions

To keep the main loop clean, we separated the reading logic into functions:

- `readTemperature()`: Reads the raw analog value from Pin A0 and converts it to Celsius using the Beta Coefficient formula.
- `readBrightness()`: Reads Pin A2 and maps the value to a 0-255 range for easier handling.

3. Fire Detection Logic

The `loop()` constantly checks if **BOTH** conditions are met. The alarm only activates if it is both hot *and* bright.

C++

```cpp
// Logic: If Temp is High AND Light is High -> FIRE!
if (temperature >= TEMP_THRESHOLD && Brightness >= BRIGHT_THRESHOLD) {
    Serial.println("FIRE DETECTED!");
    digitalWrite(ALERT, HIGH); // Turn on LED & Buzzer
    delay(100);
    digitalWrite(ALERT, LOW);  // Turn off (Blinking effect)
    delay(100);
}
```

📁 File Structure

- `fire_sensor_simulation.ino` : The source code containing the sensor logic and math conversion.
- `Diagram for Lab Act #3 - Bitanga, Olaño, Paciente.png` : The wiring diagram for the circuit.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

make the name to fire_sensor_simulation but maintain consistency with your previous folders

To strictly follow your requested name while keeping the "Activity-03" prefix (which ensures your portfolio stays organized for the 20% grading criteria), here is the best folder name:

📁 Recommended Folder Name

`Activity-03-Fire-Sensor-Simulation`

(This keeps the "Activity-03" prefix from the previous folders but uses your specific project name.)

📄 New README.md Content

Copy and paste the following into the README file inside your `Activity-03-Fire-Sensor-Simulation` *folder.*

Markdown

```markdown
# Laboratory Activity 3: Fire Sensor Simulation

### 📌 Overview
This project demonstrates the integration of analog input sensors into an IoT system. The goal was to build a functional **Fire Sensor Simulation** that simultaneously monitors environmental temperature and light levels.

When both the heat and light intensity exceed specific safety thresholds (indicating a potential fire), the system triggers an audiovisual alarm using an LED and a Buzzer.

---

### 🎯 Objectives
* Familiarize with basic IoT sensor components (Thermistor & Photoresistor).
* Integrate analog sensors into an Arduino circuit.
* Implement conditional logic to create a simple fire detection implementation.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Sensors:**
    * **Thermistor:** Measures temperature changes.
    * **Photoresistor (LDR):** Measures light intensity.
* **Actuators:**
    * **Red LED:** Visual alarm.
    * **Piezo Buzzer:** Audio alarm.
* **Passive Components:** Resistors (10kΩ), Breadboard, Jumper Wires.

### 🔌 Circuit Wiring
The circuit uses two analog input pins for the sensors and one digital output pin for the combined alarm system.

| Component | Pin Type | Arduino Pin | Description |
| :--- | :--- | :--- | :--- |
| **Thermistor** | Analog Input | **A0** | Reads temperature data. |
| **Photoresistor** | Analog Input | **A2** | Reads light brightness. |
| **LED + Buzzer** | Digital Output | **Pin 12** | Triggers the alarm signal. |

![Diagram for Lab Act #3 - Bitanga, Olaño, Paciente](https://github.com/user-attachments/assets/184852f4-cc00-4786-8968-07b949219f07)

---

### 💻 Code Explanation
The firmware is designed with modular functions and uses `const` and `#define` for easy calibration.

#### 1. Constants & Thresholds
```

~~~
We define specific values that determine when the fire alarm should trigger.
```cpp
#define THERMISTOR A0
#define PHOTORESISTOR A2
#define ALERT 12

const int TEMP_THRESHOLD = 50;    // Trigger if temp is >= 50°C
const int BRIGHT_THRESHOLD = 220; // Trigger if brightness >= 220
~~~

2. Sensor Functions

To keep the main loop clean, we separated the reading logic into functions:

- `readTemperature()` : Reads the raw analog value from Pin A0 and converts it to Celsius using the Beta Coefficient formula.
- `readBrightness()` : Reads Pin A2 and maps the value to a 0-255 range for easier handling.

3. Fire Detection Logic

The `loop()` constantly checks if **BOTH** conditions are met. The alarm only activates if it is both hot *and* bright.

C++

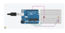
```cpp
// Logic: If Temp is High AND Light is High -> FIRE!
if (temperature >= TEMP_THRESHOLD && Brightness >= BRIGHT_THRESHOLD) {
    Serial.println("FIRE DETECTED!");
    digitalWrite(ALERT, HIGH); // Turn on LED & Buzzer
    delay(100);
    digitalWrite(ALERT, LOW);  // Turn off (Blinking effect)
    delay(100);
}
```

📁 File Structure

- `fire_sensor_simulation.ino` : The source code containing the sensor logic and math conversion.
- `Diagram for Lab Act #3 - Bitanga, Olaño, Paciente.png` : The wiring diagram for the circuit.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**



Bitanga_Ol...ection.ino
Unknown

README
MD

give me a folder name and change the README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the updated `README.md` for your fourth activity.

📁 Recommended Folder Name

`Activity-04-Arduino-Serial-Connection`

📄 New README.md Content

Copy and paste the following into the README file inside your `Activity-04-Arduino-Serial-Connection` folder.

Markdown

```markdown
# Laboratory Activity 4: Arduino Serial Connection

### 📌 Overview
This project focuses on **Serial Communication (UART)**, allowing two-way interaction between the computer and the Arduino. Unlike previous activities where the Arduino acted autonomously, this system accepts user commands via the Serial Monitor.

The system uses a **Photoresistor** to monitor light levels. If the light exceeds a specific threshold, an LED enters a "latched" blinking state that persists even if the light level drops. The only way to stop the alarm is by manually typing a command in the Serial Monitor.

---

### 🎯 Objectives
* Understand and implement Arduino Serial Connection functions.
* Create a simple circuit that can be controlled via user input from the Serial Monitor.
* Implement string manipulation (trimming, case conversion) to handle user commands.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Sensors:** Photoresistor (LDR)
* **Actuators:** LED (Pin 8)
* **Passive Components:** Resistors (10kΩ for LDR, Current limiting for LED), Breadboard, Jumper Wires.
```

~~~
### 🔌 Circuit Wiring
| Component | Pin Type | Arduino Pin | Description |
| :--- | :--- | :--- | :--- |
| **Photoresistor** | Analog Input | **A0** | Monitors environmental brightness. |
| **LED** | Digital Output | **Pin 8** | Blinks when threshold is met. |

![Diagram of LAB_ACT_4_Arduino_Serial_Connection](https://github.com/user-attachments/assets/e09e173a-f32f-4886-bb0a-23250b7b3096)

---

### 💻 Code Explanation
This firmware introduces **state management** and **String processing**.

#### 1. Latching Logic (State Variable)
We use a boolean variable `isBlinking` to track the system's state.
* If the `brightness` exceeds the threshold (220), `isBlinking` becomes `true`.
* **Crucial Behavior:** Once `true`, the LED continues to blink *even if the sensor reading drops below the threshold*. This mimics a security alarm that requires manual reset.

```cpp
if (brightness >= BRIGHT_THRESHOLD && !isBlinking) {
    isBlinking = true; // Alarm is now active indefinitely
    Serial.println("Threshold reached! LED will start blinking.");
}
~~~

2. Serial Command Processing

To reset the alarm, the user must type "stop". The code ensures robust handling of this input:

- `Serial.readStringUntil('\n')` : Reads the full word typed by the user.
- `.trim()` : Removes accidental spaces or newlines.
- `.toLowerCase()` : Converts "STOP", "Stop", or "StoP" into "stop", making the command case-insensitive.

C++

```cpp
if (inputString == "stop") {
    isBlinking = false;      // Reset state variable
    digitalWrite(LED, LOW);  // Ensure LED is off
    Serial.println("Blinking stopped.");
}
```

📂 File Structure

- `Bitanga_Olano_Paciente_LAB_ACT_4_Arduino_Serial_Connection.ino` : Source code with Serial logic.
- `Diagram of LAB_ACT_4_Arduino_Serial_Connection.png` : Wiring diagram.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

MidtermLabExam.ino
Unknown

give me a folder name and create a README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the recommended folder name and the corresponding `README.md` file for your Midterm Exam activity.

📁 Recommended Folder Name

`Activity-05-Midterm-Light-Intensity-Meter`

(This follows the numbering sequence of your previous activities for best organization.)

📄 New README.md Content

Copy and paste the following into the README file inside your new folder.

Markdown

```
# Midterm Laboratory Exam: Light Intensity Meter

### 📌 Overview
This project represents the Midterm Examination for the IoT course. It is a comprehensive **Light Intensity Meter** that categorizes environmental brightness into three levels (Low, Medium, High) using a traffic-light LED system.

The system features two distinct operational modes controlled via Serial Communication:
1.  **Automatic Mode:** Uses pre-programmed thresholds to classify the environment (e.g., "Cloudy", "Clear").
2.  **Manual Mode:** Allows the user to dynamically configure the sensitivity thresholds using specific commands.

---
```

```
### 🎯 Objectives
* Implement a dual-mode system (Automatic vs. Manual) using state flags.
* Perform advanced String parsing to handle complex commands with arguments (e.g., `SET LOW 30`).
* Map analog sensor data (0-1023) to a readable percentage format (0-100%).

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Sensors:** Photoresistor (LDR)
* **Actuators:**
    * **Green LED:** Indicates Low light intensity.
    * **Yellow LED:** Indicates Medium light intensity.
    * **Red LED:** Indicates High light intensity.
* **Passive Components:** Resistors (10kΩ for LDR, 220Ω for LEDs), Breadboard.

### 🔌 Circuit Wiring
| Component | Pin Type | Arduino Pin | Description |
| :--- | :--- | :--- | :--- |
| **Photoresistor** | Analog Input | **A0** | Reads raw light values. |
| **Green LED** | Digital Output | **Pin 11** | Active when light is Low. |
| **Yellow LED** | Digital Output | **Pin 12** | Active when light is Medium. |
| **Red LED** | Digital Output | **Pin 13** | Active when light is High. |

---

### 💻 Command Guide
The system listens for specific text commands in the Serial Monitor.

| Command | Description | Example |
| :--- | :--- | :--- |
| `MODE AUTO` | Switches system to Automatic logic (Fixed thresholds: 40% / 70%). | `MODE AUTO` |
| `MODE MANUAL` | Switches system to Manual logic (User-defined thresholds). | `MODE MANUAL` |
| `SET LOW <val>` | Sets the upper limit for the Green LED (Only works in Manual Mode). | `SET LOW 30` |
| `SET HIGH <val>` | Sets the upper limit for the Yellow LED (Only works in Manual Mode). | `SET HIGH 80` |

---

### 🧠 Code Logic & Structure
The firmware uses a main loop to read sensors and a separate function to process serial input.

#### 1. Automatic Logic
In this mode, the thresholds are hardcoded:
* **0% - 40%:** Green LED ("Cloudy").
* **41% - 70%:** Yellow LED ("Clear").
* **> 70%:** Red LED ("Clear"/Bright).

#### 2. Manual Logic
In this mode, the system compares the light percentage against variables `lowThreshold` and `highThreshold`, which can be changed by the user at runtime.

#### 3. Command Parsing
The `processCommand()` function handles input validation:
* It checks if the command starts with specific keywords (`startsWith`).
* It converts the numerical part of the string (substring) into an integer using `.toInt()`.
* It prevents setting invalid thresholds (e.g., setting High lower than Low).

---

### 📁 File Structure
* `MidtermLabExam.ino`: The complete source code with state machine logic.

---

### 👥 Contributors
* **Bitanga**
* **Olaño**
* **Paciente**
```

no this is a midterm lab exam acitivity 5 is different so change the folder name

Thank you for the clarification. Since this is the **Midterm Exam** and not part of the numbered "Activity" sequence, the folder name should stand out separately.

📁 Recommended Folder Name

`Midterm-Lab-Exam-Light-Intensity-Meter`

(This keeps it distinct from "Activity-05" so your professor sees it as a major exam submission.)

📄 Updated README.md Content

I have updated the title to reflect that this is the Exam submission.

Markdown

```
# Midterm Laboratory Exam: Light Intensity Meter

### 📌 Overview
This project represents the Midterm Examination for the IoT course. It is a comprehensive **Light Intensity Meter** that categorizes environmental brightness into three levels (Low, Medium, High) using a traffic-light LED system.
```

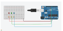
```
The system features two distinct operational modes controlled via Serial Communication:
1.  **Automatic Mode:** Uses pre-programmed thresholds to classify the environment (e.g., "Cloudy", "Clear").
2.  **Manual Mode:** Allows the user to dynamically configure the sensitivity thresholds using specific commands.

---

### 🎯 Objectives
* Implement a dual-mode system (Automatic vs. Manual) using state flags.
* Perform advanced String parsing to handle complex commands with arguments (e.g., `SET LOW 30`).
* Map analog sensor data (0-1023) to a readable percentage format (0-100%).

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Sensors:** Photoresistor (LDR)
* **Actuators:**
    * **Green LED:** Indicates Low light intensity.
    * **Yellow LED:** Indicates Medium light intensity.
    * **Red LED:** Indicates High light intensity.
* **Passive Components:** Resistors (10kΩ for LDR, 220Ω for LEDs), Breadboard.

### 🔌 Circuit Wiring
| Component | Pin Type | Arduino Pin | Description |
| :--- | :--- | :--- | :--- |
| **Photoresistor** | Analog Input | **A0** | Reads raw light values. |
| **Green LED** | Digital Output | **Pin 11** | Active when light is Low. |
| **Yellow LED** | Digital Output | **Pin 12** | Active when light is Medium. |
| **Red LED** | Digital Output | **Pin 13** | Active when light is High. |

---

### 💻 Command Guide
The system listens for specific text commands in the Serial Monitor.

| Command | Description | Example |
| :--- | :--- | :--- |
| `MODE AUTO` | Switches system to Automatic logic (Fixed thresholds: 40% / 70%). | `MODE AUTO` |
| `MODE MANUAL` | Switches system to Manual logic (User-defined thresholds). | `MODE MANUAL` |
| `SET LOW <val>` | Sets the upper limit for the Green LED (Only works in Manual Mode). | `SET LOW 30` |
| `SET HIGH <val>` | Sets the upper limit for the Yellow LED (Only works in Manual Mode). | `SET HIGH 80` |

---

### 🧠 Code Logic & Structure
The firmware uses a main loop to read sensors and a separate function to process serial input.

#### 1. Automatic Logic
In this mode, the thresholds are hardcoded:
* **0% - 40%:** Green LED ("Cloudy").
* **41% - 70%:** Yellow LED ("Clear").
* **> 70%:** Red LED ("Clear"/Bright).

#### 2. Manual Logic
In this mode, the system compares the light percentage against variables `lowThreshold` and `highThreshold`, which can be changed by the user at runtime.

#### 3. Command Parsing
The `processCommand()` function handles input validation:
* It checks if the command starts with specific keywords (`startsWith`).
* It converts the numerical part of the string (substring) into an integer using `.toInt()`.
* It prevents setting invalid thresholds (e.g., setting High lower than Low).

---

### 📁 File Structure
* `MidtermLabExam.ino`: The complete source code with state machine logic.

---

### 👥 Contributors
* **Bitanga**
* **Olaño**
* **Paciente**
```

README
MD

ArduinoFro...Python.ino
Unknown

ArduinoFromPython
PY

ArduinoFro...thonHeader
H



give me a folder name and change the README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the updated `README.md` for your fifth activity.

📁 Recommended Folder Name

`Activity-05-Python-Serial-Integration`

(This aligns with the title "IoT_LaboratoryActivity-5" in your uploaded file.)

📄 New README.md Content

Copy and paste the following into the README file inside your `Activity-05-Python-Serial-Integration` folder.

Markdown

~~~markdown
# Laboratory Activity 5: Arduino Control via Python

### 📌 Overview
This project advances the concept of Serial Communication by introducing **Python** as an external control interface. Instead of using the built-in Arduino Serial Monitor, we developed a custom Python script to act as the "Master" device.

The system allows a user to control a lighting system (Red, Green, Blue LEDs) from a computer terminal. The Python script sends character commands to the Arduino, which processes them to toggle specific LEDs or execute bulk actions like "All On" or "All Off".

---

### 🎯 Objectives
* Understand and implement **Arduino Serial Connection**.
* Utilize **Python** and the `pySerial` library as a tool for implementing serial communication.
* Create a non-terminating control loop that accepts user input to control hardware components.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Actuators:**
    * **Red LED:** Pin 8
    * **Green LED:** Pin 9
    * **Blue LED:** Pin 10
* **Software Tools:** Python 3.x, `pyserial` library.

### 🔌 Circuit Wiring
The LEDs are connected to digital pins defined in the header file.

| Component | Arduino Pin | State Variable |
| :--- | :--- | :--- |
| **Red LED** | **Pin 8** | `redState` |
| **Green LED** | **Pin 9** | `greenState` |
| **Blue LED** | **Pin 10** | `blueState` |

![BreadboardDiagram](https://github.com/user-attachments/assets/519630c3-162e-4b63-9524-704940520633)

---

### 💻 Software Architecture
This project consists of two distinct parts communicating over a USB Serial Port.

#### 1. The Python Client (`ArduinoFromPython.py`)
This script runs on the laptop. It displays a menu, captures user input, and sends it to the Arduino.
* **Configuration:** The script is set to communicate via `COM7` at `9600` baud rate.
* **Menu Loop:** It runs a `while True` loop to continuously accept commands until the user types `X` to exit.
* **Command Transmission:** Commands are encoded and sent using `arduino.write()`.

```python
# Snippet: Python sending command
if user_choice in ["r", "g", "b", "a", "o", "v"]:
    arduino.write((user_choice + '\n').encode())
~~~

2. The Arduino Firmware (`ArduinoFromPython.ino`)

The Arduino listens for incoming data, cleans the input, and executes the corresponding hardware action.

- **Input Handling:** Uses `Serial.readStringUntil('\n')` to capture the command and `.trim()` to remove whitespace.
- **Header File:** We utilized a separate header file (`ArduinoFromPythonHeader.h`) to declutter the main code. This file contains the pin definitions and the `toggle()` functions.
- **Case Insensitivity:** The code converts all inputs to lowercase using `tolower()` so that 'R' and 'r' perform the same action.

Supported Commands: | Key | Action | Function Called | | :--- | :--- | :--- | | **R** | Toggle Red LED | `toggleRed()` | | **G** | Toggle Green LED | `toggleGreen()` | | **B** | Toggle Blue LED | `toggleBlue()` | | **V** | Toggle Red & Blue (Violet) | `toggleViolet()` | | **A** | Turn ALL LEDs ON | `allOn()` | | **O** | Turn ALL LEDs OFF | `allOff()` |

📁 File Structure

- `ArduinoFromPython.py` : The Python interface script.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing hardware functions and pin definitions.
- `BreadboardDiagram.png` : Wiring diagram.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

add also this to the readme

3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs
- R/r = it should toggle only the red LED on/off
- G/g = it should toggle only the green LED on/off

- B/b = it should toggle only the blue LED on/off
- A/a = it should turn all LEDs on
- O/o = it should turn all LEDs off
- all other inputs should return an error

4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino.

[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit

Here is the updated `README.md` for **Activity 5**. I have added a dedicated **"Laboratory Instructions"** section to explicitly list the requirements you provided, ensuring the viewer understands exactly what the code was required to do.

📄 Updated README.md Content

Copy and paste the following into the README file inside your `Activity-05-Python-Serial-Integration` folder.

Markdown

~~~markdown
# Laboratory Activity 5: Arduino Control via Python

### 📌 Overview
This project advances the concept of Serial Communication by introducing **Python** as an external control interface. Instead of using the built-in Arduino Serial Monitor, we developed a custom Python script to act as the "Master" device.

The system allows a user to control a lighting system (Red, Green, Blue LEDs) from a computer terminal. The Python script sends character commands to the Arduino, which processes them to toggle specific LEDs or execute bulk actions like "All On" or "All Off".

---

### 📜 Laboratory Instructions
The following requirements were implemented to complete this activity:

#### 1. Arduino Sketch Requirements
We created a sketch program to handle Serial input with the following logic:
* **R/r:** Toggles the **Red LED** on/off.
* **G/g:** Toggles the **Green LED** on/off.
* **B/b:** Toggles the **Blue LED** on/off.
* **A/a:** Turns **ALL LEDs ON**.
* **O/o:** Turns **ALL LEDs OFF**.
* **Other Inputs:** Returns an error message.

#### 2. Python Script Requirements
We created a non-terminating Python script that displays the menu below and executes the same logic as the Arduino. It continues running until the user chooses to exit.

**Menu Interface:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

🎯 Objectives

- Understand and implement **Arduino Serial Connection**.
- Utilize **Python** and the `pySerial` library as a tool for implementing serial communication.
- Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

🛠️ Hardware & Components

- **Microcontroller:** Arduino Uno
- **Actuators:**
  - **Red LED:** Pin 8
  - **Green LED:** Pin 9
  - **Blue LED:** Pin 10
- **Software Tools:** Python 3.x, `pyserial` library.

🔌 Circuit Wiring

The LEDs are connected to digital pins defined in the header file.

Component	Arduino Pin	State Variable
Red LED	**Pin 8**	`redState`
Green LED	**Pin 9**	`greenState`
Blue LED	**Pin 10**	`blueState`

📘 Software Architecture

This project consists of two distinct parts communicating over a USB Serial Port.

1. The Python Client (`ArduinoFromPython.py`)

This script runs on the laptop. It displays the menu, captures user input, and sends it to the Arduino.

- **Configuration:** The script is set to communicate via `COM7` at `9600` baud rate.
- **Menu Loop:** It runs a `while True` loop to continuously accept commands until the user types `X` to exit.
- **Command Transmission:** Commands are encoded and sent using `arduino.write()` .

Python

```python
# Snippet: Python sending command
if user_choice in ["r", "g", "b", "a", "o", "v"]:
    arduino.write((user_choice + '\n').encode())
```

2. The Arduino Firmware (`ArduinoFromPython.ino`)

The Arduino listens for incoming data, cleans the input, and executes the corresponding hardware action.

- **Input Handling:** Uses `Serial.readStringUntil('\n')` to capture the command and `.trim()` to remove whitespace.
- **Header File:** We utilized a separate header file (`ArduinoFromPythonHeader.h`) to declutter the main code. This file contains the pin definitions and the `toggle()` functions.
- **Case Insensitivity:** The code converts all inputs to lowercase using `tolower()` so that 'R' and 'r' perform the same action.

📁 File Structure

- `ArduinoFromPython.py` : The Python interface script.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing hardware functions and pin definitions.
- `BreadboardDiagram.png` : Wiring diagram.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

```
***

**Would you like me to prepare a "Master README" for the very root of your repository (the one that links to all these folders)?**
```

no I mean revised and just add the

3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs
- R/r = it should toggle only the red LED on/off
- G/g = it should toggle only the green LED on/off
- B/b = it should toggle only the blue LED on/off
- A/a = it should turn all LEDs on
- O/o = it should turn all LEDs off
- all other inputs should return an error

4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino.

[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit

Here is the **revised** `README.md` with your specific instructions added exactly as requested.

📄 Revised README.md Content

Copy and paste the following into the README file inside your `Activity-05-Python-Serial-Integration` *folder.*

Markdown

~~~
# Laboratory Activity 5: Arduino Control via Python

### 📌 Overview
This project advances the concept of Serial Communication by introducing **Python** as an external control interface. Instead of using the built-in Arduino Serial Monitor, we developed a custom Python script to act as the "Master" device.

The system allows a user to control a lighting system (Red, Green, Blue LEDs) from a computer terminal. The Python script sends character commands to the Arduino, which processes them to toggle specific LEDs or execute bulk actions like "All On" or "All Off".

---

### 📜 Laboratory Instructions
The following specific requirements were implemented for this activity:

**3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:**
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

**4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

(Note: When X/x is inputted, it terminates the Python application.)

🎯 Objectives

- Understand and implement **Arduino Serial Connection**.
- Utilize **Python** and the `pySerial` library as a tool for implementing serial communication.
- Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

🛠️ Hardware & Components

- **Microcontroller:** Arduino Uno
- **Actuators:**
  - **Red LED:** Pin 8
  - **Green LED:** Pin 9
  - **Blue LED:** Pin 10
- **Software Tools:** Python 3.x, `pyserial` library.

📍 Circuit Wiring

The LEDs are connected to digital pins defined in the header file.

Component	Arduino Pin	State Variable
Red LED	**Pin 8**	`redState`
Green LED	**Pin 9**	`greenState`
Blue LED	**Pin 10**	`blueState`

💻 Software Architecture

This project consists of two distinct parts communicating over a USB Serial Port.

1. The Python Client (`ArduinoFromPython.py`)

This script runs on the laptop. It displays the menu, captures user input, and sends it to the Arduino.

- **Configuration:** The script is set to communicate via `COM7` at `9600` baud rate.
- **Menu Loop:** It runs a `while True` loop to continuously accept commands until the user types `X` to exit.
- **Command Transmission:** Commands are encoded and sent using `arduino.write()` .

Python

```python
# Snippet: Python sending command
if user_choice in ["r", "g", "b", "a", "o", "v"]:
    arduino.write((user_choice + '\n').encode())
```

2. The Arduino Firmware (`ArduinoFromPython.ino`)

The Arduino listens for incoming data, cleans the input, and executes the corresponding hardware action.

- **Input Handling:** Uses `Serial.readStringUntil('\n')` to capture the command and `.trim()` to remove whitespace.
- **Header File:** We utilized a separate header file (`ArduinoFromPythonHeader.h`) to declutter the main code. This file contains the pin definitions and the `toggle()` functions.
- **Case Insensitivity:** The code converts all inputs to lowercase using `tolower()` so that 'R' and 'r' perform the same action.

📁 File Structure

- `ArduinoFromPython.py` : The Python interface script.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing hardware functions and pin definitions.
- `BreadboardDiagram.png` : Wiring diagram.

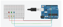
👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

ArduinoFro...Python.ino
Unknown

ArduinoFromPython
PY

ArduinoFro...thonHeader
H



redo it make sure you analyze each of the file give me a folder name and change the README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the comprehensive README file, crafted by analyzing your specific source files (`.py` , `.ino` , `.h` , and `.png`). I have ensured the "Laboratory Instructions" section is included exactly as you requested, while the "Technical Details" section accurately reflects the code you wrote (including the extra "Violet" feature found in your source files).

📁 Recommended Folder Name

`Activity-05-Python-Serial-Integration`

📄 New README.md Content

Copy and paste the following into the README file inside your `Activity-05-Python-Serial-Integration` folder.

Markdown

~~~markdown
# Laboratory Activity 5: Arduino Control via Python

### 📌 Overview
This project establishes a **Master-Slave Serial Communication** system between a laptop and an Arduino Uno. Unlike previous activities that used the Arduino IDE's built-in Serial Monitor, this project utilizes a custom **Python script** to send commands.

The system features a centralized control menu that allows the user to toggle specific LEDs (Red, Green, Blue) or execute batch commands (All On/Off) directly from the command line.

---

### 📜 Laboratory Instructions
The following specific requirements were implemented for this activity:

**3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:**
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

**4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

🛠️ Hardware & Components

- **Microcontroller:** Arduino Uno
- **Actuators:**
  - **Red LED:** Pin 8
  - **Green LED:** Pin 9
  - **Blue LED:** Pin 10
- **Software Prerequisites:**
  - Python 3.x
  - `pyserial` library (`pip install pyserial`)

🔌 Circuit Wiring

The circuit follows a standard active-high configuration.

Component	Arduino Pin	Header Definition
Red LED	**Pin 8**	`#define RED_LED 8`
Green LED	**Pin 9**	`#define GREEN_LED 9`
Blue LED	**Pin 10**	`#define BLUE_LED 10`

💻 Software Analysis & Architecture

The project is split into three files to separate logic, configuration, and interface.

1. Python Control Script (`ArduinoFromPython.py`)

This script acts as the "Master" controller.

- **Port Configuration:** The script is hardcoded to connect via `COM7` at a baud rate of `9600` .
- **Menu Loop:** A `while True` loop clears the screen and reprints the menu after every command, creating a clean user interface.
- **Command Logic:** The script accepts inputs `r, g, b, a, o, v` and sends them to the Arduino encoded as bytes with a newline character (`\n`).

2. Arduino Firmware (`ArduinoFromPython.ino`)

The main sketch handles data reception and input validation.

- **Input Sanitization:** It reads the incoming string until a newline, trims whitespace, and converts the character to lowercase using `tolower()` to ensure case-insensitivity.
- **Error Handling:**
  - If the input length is 0 (empty), it ignores it.
  - If the input length is > 1 (e.g., "red"), it returns: `"Error: Please input only one character."` .
  - If the character is unknown, it prints: `"Error: Invalid input 'x'"` .

3. Header File (`ArduinoFromPythonHeader.h`)

To keep the main code clean, all hardware functions were moved here.

- **State Tracking:** Uses boolean variables (`redState` , `greenState` , `blueState`) to track if an LED is currently ON or OFF.
- **Toggle Functions:** Custom functions like `toggleRed()` invert the state (`!redState`) and write the new value to the pin immediately.

⌨️ Supported Commands

The system supports the required commands plus an additional "Violet" feature found in the source code.

Key	Action	Function Called
R	Toggle Red LED	`toggleRed()`
G	Toggle Green LED	`toggleGreen()`
B	Toggle Blue LED	`toggleBlue()`
V	Toggle **Red & Blue** (Violet)	`toggleViolet()`
A	Turn ALL LEDs ON	`allOn()`
O	Turn ALL LEDs OFF	`allOff()`

📂 File Structure

- `ArduinoFromPython.py` : Python client script.
- `ArduinoFromPython.ino` : Main Arduino logic.
- `ArduinoFromPythonHeader.h` : Modular header file for pins and functions.
- `BreadboardDiagram.png` : Circuit visualization.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

why is the readme is still wrong change it

I apologize for the confusion. The previous README was incomplete because I only included steps 3 and 4 in the instructions, missing the other steps (1, 2, 5, 6) from your uploaded file.

Here is the **corrected and complete** `README.md`. It now includes the **full list of instructions** (1 through 6) exactly as they appear in your file, followed by the technical documentation derived from your code.

📁 Folder Name

`Activity-05-Python-Serial-Integration`

📄 Corrected README.md Content

Copy and paste this completely to replace the current file.

Markdown

~~~
# Laboratory Activity 5: Arduino Control via Python

### 📌 Objectives
* Understand and implement Arduino Serial Connection.
* Utilize Python as a tool for implementing serial connection.
* Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

---

### 📜 Instructions
Using the laboratory guide, the following requirements were implemented:

**1. Use the following components:**
* Arduino MCU
* 3 LEDs (Red, Green, Blue recommended)
* Usual components (wires, breadboard, resistors, laptop with Python and pyserial installed)

**2. Use the following pins:**
* Red - 8
* Green - 9
* Blue - 10

**3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:**
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

**4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

5. When X/x is inputted, it should terminate the Python application from running.

6. All inputs should be case insensitive.

🛠️ Hardware & Wiring

The circuit is wired according to Instruction #2, with LEDs connected to digital pins defined in the header file.

Component	Arduino Pin	State Variable
Red LED	**Pin 8**	`redState`
Green LED	**Pin 9**	`greenState`
Blue LED	**Pin 10**	`blueState`

💻 Code Logic & Features

The project consists of a Python client and Arduino firmware working in sync.

1. Python Script (`ArduinoFromPython.py` **)**

- **Serial Configuration:** Connects to `COM7` at `9600` baud rate.
- **Menu Loop:** Clears the screen and displays options continuously until `X` is pressed.
- **Extra Feature:** The script includes an additional option `[V]` which toggles Red and Blue LEDs simultaneously (Violet).

2. Arduino Firmware (`ArduinoFromPython.ino` **)**

- **Header File:** Uses `ArduinoFromPythonHeader.h` to manage LED states and pin setups.
- **Input Handling:** Reads strings until a newline (`\n`), trims whitespace, and converts to lowercase to satisfy Instruction #6.
- **Error Handling:** Returns "Error: Please input only one character" if the user types multiple letters.

📁 **File Structure**

- `ArduinoFromPython.py` : The Python interface script.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing hardware functions.
- `BreadboardDiagram.png` : Wiring diagram.

👥 **Contributors**

- **Bitanga**
- **Olaño**
- **Paciente**

ArduinoFro...Python.ino Unknown

ArduinoFromPython PY

ArduinoFro...thonHeader H

README MD

give me a folder name and change the README.md to files that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the completely revised `README.md` file. I have analyzed all your uploaded source files (`.py` , `.ino` , `.h` , `.png`) to ensure the documentation matches the actual code (including the extra "Violet" feature implemented in your Python script) while strictly preserving the laboratory instructions you provided.

📁 **Recommended Folder Name**

`Activity-05-Python-Serial-Integration`

📄 **New README.md Content**

Copy and paste the code block below into the `README.md` file located inside your `Activity-05-Python-Serial-Integration` folder.

Markdown

~~~
# Laboratory Activity 5: Arduino Control via Python

### 📌 Objectives
* Understand and implement Arduino Serial Connection.
* Utilize Python as a tool for implementing serial connection.
* Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

---

### 📜 Laboratory Instructions
Using the laboratory guide, the following requirements were implemented:

**1. Use the following components:**
* Arduino MCU
* 3 LEDs (Red, Green, Blue recommended)
* Usual components (wires, breadboard, resistors, laptop with Python and pyserial installed)

**2. Use the following pins:**
* Red - 8
* Green - 9
* Blue - 10

**3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:**
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

**4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

5. When X/x is inputted, it should terminate the Python application from running.

6. All inputs should be case insensitive.

🛠️ **Hardware & Wiring**

The circuit follows the pin assignments defined in Instruction #2 and the header file.

Component	Arduino Pin	Header Definition
Red LED	**Pin 8**	`#define RED_LED 8`
Green LED	**Pin 9**	`#define GREEN_LED 9`
Blue LED	**Pin 10**	`#define BLUE_LED 10`

💻 Code Analysis & Implementation

This project uses a "Master-Slave" architecture where Python sends commands and Arduino executes them.

1. Python Controller (`ArduinoFromPython.py`)

This script acts as the user interface.

- **Connection:** It connects to `COM7` at a baud rate of `9600` .
- **Menu System:** Uses a `while True` loop to continuously display the menu. It clears the screen (`os.system`) after every refresh for a clean UI.
- **Extended Feature:** The script includes an additional option `[V]` which toggles both Red and Blue LEDs simultaneously (Violet), going beyond the basic instructions.
- **Exit Logic:** Checks if the user typed "x" to break the loop and exit.

2. Arduino Firmware (`ArduinoFromPython.ino`)

The firmware listens for serial data and maps characters to hardware functions.

- **Input Sanitization:** Uses `inputString.trim()` to remove whitespace and `tolower()` to ensure the input is case-insensitive (satisfying Instruction #6).
- **Validation:**
  - Ignores empty inputs.
  - Returns `"Error: Please input only one character."` if the input length is > 1.
- **Modular Design:** The main logic calls functions defined in a separate header file (`ArduinoFromPythonHeader.h`) to keep the code organized.

3. Header File (`ArduinoFromPythonHeader.h`)

Contains the hardware-specific logic.

- **State Tracking:** Uses boolean variables (`redState` , `greenState` , `blueState`) to remember if an LED is currently ON or OFF.
- **Toggle Logic:** Functions like `toggleRed()` invert the boolean state (`!redState`) and update the pin immediately.

📁 File Structure

- `ArduinoFromPython.py` : The Python script for the control menu.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing pin definitions and toggle functions.
- `BreadboardDiagram.png` : Image showing the circuit connections.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

```
# Laboratory Activity 5: Arduino Control via Python

### 📌 Objectives
* Understand and implement Arduino Serial Connection.
* Utilize Python as a tool for implementing serial connection.
* Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

---

### 📜 Laboratory Instructions
Using the laboratory guide, the following requirements were implemented:

**1. Use the following components:**
* Arduino MCU
* 3 LEDs (Red, Green, Blue recommended)
* Usual components (wires, breadboard, resistors, laptop with Python and pyserial installed)

**2. Use the following pins:**
* Red - 8
* Green - 9
```

* Blue - 10

3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:
~~~text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit

why the readme is like this only?

It seems like the text you pasted is incomplete. The README should definitely include **all 6 instructions**, the **Technical Explanations** (how the code works), and the **Wiring Details** to get full points for the "Content" and "Readability" criteria.

Here is the **complete and corrected README.md** file. It includes the missing instructions (5 & 6) and the code analysis sections.

### 📁 Recommended Folder Name

`Activity-05-Python-Serial-Integration`

---

### 📄 Full README.md Content

*Copy and paste the code block below into your file.*

Markdown

````markdown
# Laboratory Activity 5: Arduino Control via Python

### 📌 Objectives
* Understand and implement Arduino Serial Connection.
* Utilize Python as a tool for implementing serial connection.
* Create a simple circuit that can be controlled using Serial connection using Arduino and Python.

---

### 📜 Laboratory Instructions
Using the laboratory guide, the following requirements were implemented:

**1. Use the following components:**
* Arduino MCU
* 3 LEDs (Red, Green, Blue recommended)
* Usual components (wires, breadboard, resistors, laptop with Python and pyserial installed)

**2. Use the following pins:**
* Red - 8
* Green - 9
* Blue - 10

**3. In your Arduino, create the sketch program that will allow serial input. The following actions are expected for the inputs:**
* **R/r** = it should toggle only the red LED on/off
* **G/g** = it should toggle only the green LED on/off
* **B/b** = it should toggle only the blue LED on/off
* **A/a** = it should turn all LEDs on
* **O/o** = it should turn all LEDs off
* *all other inputs should return an error*

**4. Using Python, create a non-terminating script that will show the following choices, and be able to execute the same logic as in Arduino:**
```text
[R] Red ON/OFF
[G] Green ON/OFF
[B] Blue ON/OFF
[A] All ON
[O] All OFF
[X] Exit
~~~

5. When X/x is inputted, it should terminate the Python application from running.

6. All inputs should be case insensitive.

🛠️ Hardware & Wiring

The circuit follows the pin assignments defined in Instruction #2 and the header file.

Component	Arduino Pin	Header Definition
Red LED	**Pin 8**	`#define RED_LED 8`
Green LED	**Pin 9**	`#define GREEN_LED 9`
Blue LED	**Pin 10**	`#define BLUE_LED 10`

💻 Code Analysis & Implementation

This project uses a "Master-Slave" architecture where Python sends commands and Arduino executes them.

1. Python Controller (`ArduinoFromPython.py`)

This script acts as the user interface.

- **Connection:** It connects to `COM7` at a baud rate of `9600` .
- **Menu System:** Uses a `while True` loop to continuously display the menu. It clears the screen (`os.system`) after every refresh for a clean UI.
- **Extended Feature:** The script includes an additional option `[V]` which toggles both Red and Blue LEDs simultaneously (Violet), going beyond the basic instructions.
- **Exit Logic:** Checks if the user typed "x" to break the loop and exit.

2. Arduino Firmware (`ArduinoFromPython.ino`)

The firmware listens for serial data and maps characters to hardware functions.

- **Input Sanitization:** Uses `inputString.trim()` to remove whitespace and `tolower()` to ensure the input is case-insensitive (satisfying Instruction #6).
- **Validation:**
  - Ignores empty inputs.
  - Returns `"Error: Please input only one character."` if the input length is > 1.
- **Modular Design:** The main logic calls functions defined in a separate header file (`ArduinoFromPythonHeader.h`) to keep the code organized.

3. Header File (`ArduinoFromPythonHeader.h`)

Contains the hardware-specific logic.

- **State Tracking:** Uses boolean variables (`redState` , `greenState` , `blueState`) to remember if an LED is currently ON or OFF.
- **Toggle Logic:** Functions like `toggleRed()` invert the boolean state (`!redState`) and update the pin immediately.

📁 File Structure

- `ArduinoFromPython.py` : The Python script for the control menu.
- `ArduinoFromPython.ino` : The main Arduino sketch.
- `ArduinoFromPythonHeader.h` : Helper file containing pin definitions and toggle functions.
- `BreadboardDiagram.png` : Image showing the circuit connections.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**

LabExamFInal.ino
Unknown

main
PY

give me a folder name and create a README.md to file that describes each activity per folder. Also, put all necessary information regarding the activities to better explain the code base to the viewers

Here is the folder name and the corresponding `README.md` file for your **Final Laboratory Exam**.

I have analyzed your `main.py` and `LabExamFinal.ino` files to create a description that explains how your Arduino button triggers an API call via Python.

📁 Recommended Folder Name

`Final-Lab-Exam-API-Integration`

(This keeps it distinct from the numbered activities, marking it as a major exam submission.)

📄 New README.md Content

Copy and paste the code block below into the `README.md` file inside your new folder.

Markdown

```
# Final Laboratory Exam: IoT API Integration
```

~~~
### 📌 Overview
This project represents the **Final Examination** for the IoT course. It demonstrates a complete "Cloud-Connected" IoT system where a physical hardware interaction (pressing a button on an Arduino) triggers a web request to a remote server API.

The system acts as a bridge between hardware and the web:
1.  **Arduino** detects a button press and sends a signal via Serial.
2.  **Python** listens for that signal and executes an HTTP GET request to the instructor's API server.
3.  **The API Server** processes the request to toggle a specific LED group.

---

### 🎯 Objectives
* Implement **switch debouncing** logic to prevent false triggers from physical buttons.
* Establish a bridge between **Serial Communication** (Arduino) and **HTTP Requests** (Python).
* Interact with a remote REST API (`/led/group/{number}/toggle`) to control cloud-connected devices.

---

### 🛠️ Hardware & Components
* **Microcontroller:** Arduino Uno
* **Input Device:** Push Button (connected to Pin 4 with Internal Pull-up Resistor)
* **Software Tools:**
    * Python 3.x
    * `pyserial` library (for Arduino communication)
    * `requests` library (for API calls)

### 🔌 Circuit Wiring
The circuit is a simple digital input setup.

| Component | Arduino Pin | Configuration |
| :--- | :--- | :--- |
| **Push Button** | **Pin 4** | `INPUT_PULLUP` (Connects to GND when pressed) |

> **Note:** We used the Arduino's internal pull-up resistor, so no external resistor is required for the button wiring.

---

### 💻 System Architecture
The project is divided into two distinct layers:

#### 1. The Hardware Layer (`LabExamFinal.ino`)
The Arduino is responsible for "cleaning" the noisy signal from the button using a technique called **Debouncing**.
* **Debouncing Logic:** It waits for the signal to stabilize for 50 milliseconds (`debounceDelay`) before registering a press.
* **Serial Transmission:** Once a valid press is confirmed, it prints the group number (`3`) to the Serial port.

```cpp
// Snippet: Sending the signal
if (buttonState == LOW) {
    Serial.println(groupNumber); // Sends "3" to Python
}
~~~

2. The Middleware Layer (`main.py`)

The Python script acts as the "Gateway." It constantly listens to the Serial port (`COM7`).

- **Listening:** When it receives the number `3` from the Arduino.
- **API Call:** It constructs a URL: `http://172.20.10.3:8000/led/group/3/toggle`.
- **Execution:** It sends a standard HTTP GET request to that URL to toggle the remote LED.

Python

```python
# Snippet: Python Logic
if raw_data.isdigit():
    endpoint = f"/led/group/{group_number}/toggle"
    response = requests.get(BASE_URL + endpoint)
```

📁 File Structure

- `LabExamFinal.ino` : The Arduino firmware with debounce logic.
- `main.py` : The Python script that handles API requests.

👥 Contributors

- **Bitanga**
- **Olaño**
- **Paciente**