David Bortz

CS-350

Module 3 Milestone 2

09-21-2025

1. Why do both the SerialTest-Write.py and SerialLightControl-Client.py scripts use the `encode()` method of the string datatype when writing data to the serial port?

- The `encode()` method converts a string into bytes, which is the format required for serial communication. Serial ports transmit data as bytes, so when sending a string, it must be encoded to ensure that the data is in the appropriate format for transmission over the serial connection. In most cases, UTF-8 encoding is used as it is a widely accepted character encoding.

2. Why does the SerialTest-Read.py script use the `decode()` method of the string datatype when reading the data from the serial port?

- The `decode()` method is used to convert the bytes received from the serial port back into a string. When data is read from the serial port, it comes as bytes, and decoding it allows the program to interpret those bytes as a human-readable string. Again, UTF-8 is commonly used for decoding since it corresponds to the encoding used when the data was sent.

3. What is the purpose of the try/except block in both the SerialLightControl-Client.py script and the SerialLightControl-Server.py script?

- The `try/except` block is used for error handling. It allows the program to attempt to execute code that may potentially raise an exception (such as I/O operations on the serial port). If an exception occurs (e.g., due to a disconnection or read/write error), the program can catch that exception in the `except` clause, allowing it to handle the error gracefully (e.g., by logging an error message or cleaning up resources) instead of crashing unexpectedly. This improves the robustness of the application.

4. Why is it necessary to make sure that the GPIO pins are always returned to their original state at the end of program run?

- Ensuring that GPIO pins are returned to their original state is crucial for several reasons:

- Safety: Leaving GPIO pins in an active state could lead to unintended behavior in the hardware, potentially damaging components or causing safety hazards (e.g., keeping an LED on continuously).

- Resource Management: Properly cleaning up GPIO pins frees up the resources for future applications or runs of the program, allowing other programs to use the GPIO pins without conflicts.

- Predictability: Returning GPIO pins to a known state makes the behavior of the hardware predictable and avoids issues when the program is restarted or when multiple programs are using the GPIO pins. This is particularly important in embedded systems where reliability is key.