**Day 6 Notes**

Lesson Reference

A circuit board with wires and wires

Description automatically generated

Mine (220 Ω Resistor)

A circuit board with wires

Description automatically generated

Yes, you can easily log Serial Monitor data to a file, which you can then import into a spreadsheet for analysis. Here are two methods to get this done:

### Method 1: Using the Arduino IDE's Built-in Serial Plotter (Arduino IDE 2.0 and Up)

If you’re using Arduino IDE 2.0+, you can export data directly from the Serial Monitor:

1. \*\*Open Serial Monitor\*\* and run your code to view the live data.

2. \*\*Click the “Log to file” button\*\* (a floppy disk icon) in the Serial Monitor. This will prompt you to choose a file location to save the incoming data.

3. \*\*Save the File\*\*: The data will automatically be logged to a `.txt` file, which can be opened in a spreadsheet program for analysis.

In the Arduino IDE 2.0, unfortunately, there isn’t a built-in “Log to File” button directly on the Serial Monitor. However, there are a couple of alternatives for capturing your data:

1. \*\*Copy and Paste\*\*: You can manually select and copy the visible output in the Serial Monitor. Although this approach is limited to what’s displayed on the screen, you can use `Ctrl + A` and `Ctrl + C` to copy all visible data, then paste it into a text editor for saving. Some users have noted issues with this approach when dealing with large amounts of data, as the Serial Monitor may lose selection control after scrolling.

2. \*\*Third-Party Serial Monitor Tools\*\*: Many Arduino users prefer using external tools like \*\*PuTTY\*\*, \*\*CoolTerm\*\*, or \*\*Tera Term\*\* for more advanced logging. These programs allow you to select your Arduino’s COM port and log all incoming data directly to a file without any display limits. This is often the easiest solution for long sessions or continuous data logging. In PuTTY, for example, you can go to \*\*Session > Logging\*\* to set up a log file.

These solutions should allow you to capture your data effectively for later analysis in a spreadsheet or other tools. If Arduino IDE updates in the future, they may address this limitation, but for now, these workarounds are widely used by the community.

For more details, you can check discussions about this issue on the Arduino forums and related documentation.

### Method 2: Using a Third-Party Serial Monitor like `PuTTY` or `CoolTerm`

For more advanced logging options, you can use tools like PuTTY or CoolTerm to capture serial data:

1. \*\*Download PuTTY\*\* (for Windows/Linux) or \*\*CoolTerm\*\* (cross-platform).

2. \*\*Set up the Serial Connection\*\*:

- Choose the correct \*\*COM Port\*\* and \*\*baud rate\*\* (e.g., 9600) to match your Arduino settings.

3. \*\*Log to File\*\*:

- In PuTTY, you can specify a log file by selecting \*\*Logging\*\* under the \*\*Session\*\* options before starting the session.

- In CoolTerm, you can click on \*\*Connection > Capture to Text/Binary File\*\* and select a location for saving.

4. \*\*Export to Spreadsheet\*\*: Open the `.txt` file in a spreadsheet program like Excel or Google Sheets, where you can analyze or graph your data.

Let me know if you try one of these or need any help setting it up!

**PuTTY**

Let’s get you set up in PuTTY to capture that Serial Monitor data! Here’s the step-by-step to configure and log your Arduino output:

### 1. \*\*Set Up the Serial Connection\*\*

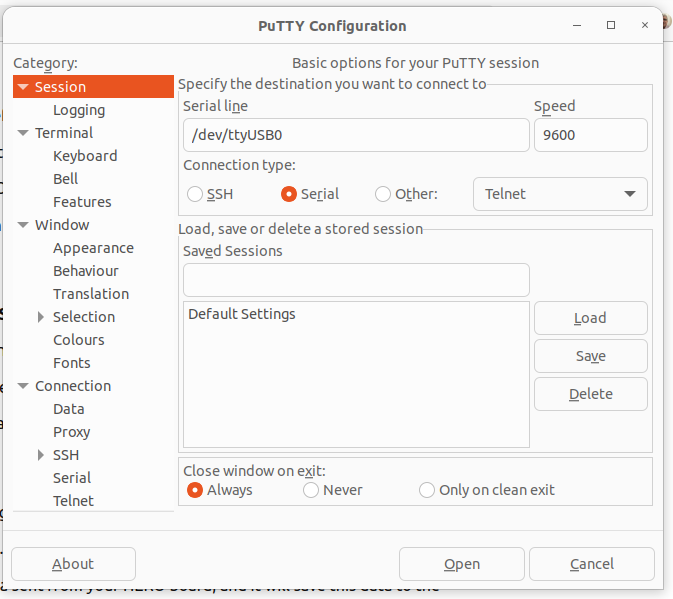
- Open PuTTY.

- In the \*\*Category\*\* pane on the left, select \*\*Session\*\*.

- Under \*\*Connection type\*\*, select \*\*Serial\*\*.

- Enter the \*\*COM Port\*\* your HERO board is using (e.g., `COM3` or `/dev/ttyUSB0` on Linux).

- Set the \*\*Speed (baud)\*\* to match your code setup (e.g., 9600).



### 2. \*\*Configure Logging\*\*

- In the \*\*Category\*\* pane, expand \*\*Session\*\* and go to \*\*Logging\*\*.

- Select \*\*All session output\*\* to ensure PuTTY logs everything.

- Choose \*\*Browse\*\* to select where you want to save the log file.

- Name the file and save it with a `.txt` extension.

### 3. \*\*Start the Session\*\*

- Go back to \*\*Session\*\* in the Category pane.

- Click \*\*Open\*\* to start the session.

- PuTTY will now display any data sent from your HERO board, and it will save this data to the file you specified.

### 4. \*\*End the Session\*\*

- When you’re done logging, close PuTTY to stop the session and save the data.

Now you can open the saved `.txt` file in any spreadsheet program or text editor to analyze your data. Let me know if this gets everything logging smoothly!

**Add Time Stamp**

Yes, we can add a timestamp to each output line in the Arduino code! Unfortunately, the Arduino itself doesn’t have a real-time clock, so we’ll rely on \*\*millis()\*\* to approximate time since the program started. If you’re running the code for extended periods or need precise date and time, an external real-time clock (RTC) module would be ideal.

Here’s how to add a relative timestamp in milliseconds to each line of output:

### Code Example with Timestamps

```cpp

void setup() {

Serial.begin(9600); // Start serial communication

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

// Example action: Blink the LED

digitalWrite(LED\_BUILTIN, HIGH);

delay(500);

digitalWrite(LED\_BUILTIN, LOW);

delay(500);

// Print timestamp and other data

Serial.print("Time: ");

Serial.print(millis()); // Prints milliseconds since program started

Serial.println(" ms | LED is blinking");

}

```

In the example above:

- `millis()` provides the time in milliseconds since the board was powered on or reset, allowing you to track elapsed time.

- Each line in the Serial Monitor or log file will have a timestamp in milliseconds, making it easier to analyze timing and patterns in your data.

If you want formatted timestamps (like HH:MM:SS), I can provide a bit more code to convert `millis()` into hours, minutes, and seconds. Let me know if that’s helpful!

To keep all the outputs in each loop iteration on a single line, you can replace each \*\*Serial.println()\*\* with \*\*Serial.print()\*\*, except for the final piece of information in each loop. This way, the data will print on the same line until the end of each loop, where \*\*Serial.println()\*\* will move to a new line for the next iteration.

Here’s an example using `Serial.print()` for everything except the final output:

```cpp

void setup() {

Serial.begin(9600); // Start serial communication

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

// Example action: Blink the LED

digitalWrite(LED\_BUILTIN, HIGH);

delay(500);

digitalWrite(LED\_BUILTIN, LOW);

delay(500);

// Print timestamp and all data on a single line

Serial.print("Time: ");

Serial.print(millis()); // Timestamp in milliseconds

Serial.print(" ms | LED State: ");

Serial.print(digitalRead(LED\_BUILTIN)); // LED state

Serial.print(" | Sensor Value: ");

Serial.print(analogRead(A0)); // Example sensor data

Serial.println(" | End of Data"); // Final piece ends the line

delay(1000); // Delay for readability

}

```

### Explanation:

- \*\*Serial.print()\*\* keeps each output on the same line.

- \*\*Serial.println()\*\* at the end finishes the line, making the next loop’s output start fresh on a new line.

This should give you a neat single-line output for each loop iteration! Let me know how it works or if you’d like more tweaks.

void setup() {

Serial.begin(9600); // Start serial communication

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

// Example action: Blink the LED

digitalWrite(LED\_BUILTIN, HIGH);

delay(500);

digitalWrite(LED\_BUILTIN, LOW);

delay(500);

// Print timestamp and all data on a single line

Serial.print("Time: ");

Serial.print(millis()); // Timestamp in milliseconds

Serial.print(" ms | LED State: ");

Serial.print(digitalRead(LED\_BUILTIN)); // LED state

Serial.print(" | Sensor Value: ");

Serial.print(analogRead(A0)); // Example sensor data

Serial.println(" | End of Data"); // Final piece ends the line

delay(1000); // Delay for readability

}