

# TAZROUT

IoT Irrigation System

## Day 2 Learning Report

Serial Communication Mastery

**Developer:** KHENFRI Moussa

**Date:** February 2, 2026

**Week:** 1 - ESP32 Fundamentals

**Session Duration:** 2 hours 45 minutes

**Planned Duration:** 2-3 hours

**Status:** Completed - On Track

Module: ESP32 Sensor & Network Communication Layer

Platform: Wokwi ESP32 Simulator

Project: TAZROUT IoT Irrigation System

## Contents

<b>1 Executive Summary</b>	<b>3</b>
1.1 Session Overview . . . . .	3
1.2 Primary Achievement . . . . .	3
1.3 Improvement from Day 1 . . . . .	3
<b>2 Detailed Learning Journey</b>	<b>4</b>
2.1 Phase 1: Serial Communication Fundamentals (35 minutes) . . . . .	4
2.1.1 Understanding Baud Rates . . . . .	4
2.1.2 Experimenting with Baud Rates . . . . .	4
2.2 Phase 2: Data Type Formatting (50 minutes) . . . . .	5
2.2.1 Working with Different Data Types . . . . .	5
2.2.2 Hexadecimal and Binary - When to Use Them? . . . . .	6
2.3 Phase 3: Structured Output Formats (55 minutes) . . . . .	6
2.3.1 Basic Text Format . . . . .	6
2.3.2 Table Format - The Game Changer . . . . .	7
2.3.3 JSON Format - Preview for Week 3 . . . . .	8
2.3.4 Debug Format with Timestamps . . . . .	9
2.4 Phase 4: Simulating Sensor Data (25 minutes) . . . . .	10
2.4.1 Creating Realistic Sensor Variations . . . . .	10
2.5 Phase 5: Memory Monitoring (15 minutes) . . . . .	11
2.5.1 ESP32-Specific Functions . . . . .	11
<b>3 Complete Working Code</b>	<b>12</b>
3.1 Final Day 2 Implementation . . . . .	12
<b>4 Technical Knowledge Acquired</b>	<b>16</b>
4.1 Serial Communication Concepts . . . . .	16
4.2 Data Formatting Patterns . . . . .	16
4.3 New Functions Learned . . . . .	17
<b>5 Challenges and Solutions</b>	<b>17</b>
5.1 Challenge Log . . . . .	17
<b>6 Skills Development Matrix</b>	<b>18</b>
<b>7 Connection to TAZROUT Project</b>	<b>19</b>
7.1 How Today's Skills Enable Future Features . . . . .	19
7.2 JSON Preview - Week 3 Connection . . . . .	19
<b>8 Reflections and Insights</b>	<b>20</b>
8.1 What Surprised Me Today . . . . .	20
8.2 Key Realizations . . . . .	20
8.3 Personal Growth . . . . .	21
<b>9 Deliverables and Documentation</b>	<b>21</b>
9.1 Wokwi Project . . . . .	21
9.2 Code Quality Assessment . . . . .	21
<b>10 Time Management Analysis</b>	<b>22</b>
10.1 Planned vs Actual Time Breakdown . . . . .	22

<b>11 Resources Used</b>	<b>22</b>
11.1 Documentation Consulted . . . . .	22
11.2 New Bookmarks . . . . .	23
<b>12 Self-Assessment Against Learning Objectives</b>	<b>23</b>
12.1 Day 2 Objectives Evaluation . . . . .	23
<b>13 Success metrics</b>	<b>23</b>
13.1 Quantitative Summary . . . . .	23
13.2 Cumulative Week 1 Stats . . . . .	24
<b>14 Conclusion and Personal Notes</b>	<b>24</b>
14.1 Overall Assessment . . . . .	24
14.2 What I'm Most Proud Of Today . . . . .	24
14.3 Honest Self-Reflection . . . . .	25
14.4 Personal Commitment Update . . . . .	25

## 1 Executive Summary

Day 2 builds directly on yesterday's foundation, diving deep into Serial communication - a critical debugging tool for all embedded development. Today felt significantly smoother than Day 1; I was already comfortable with Wokwi, which allowed me to focus entirely on learning rather than wrestling with tools.

### 1.1 Session Overview

Metric	Value
Planned Time	2-3 hours
Actual Time Spent	2 hours 45 minutes
Tasks Planned	6
Tasks Completed	6
Wokwi Projects Created	2 (main + experiments)
Lines of Code Written	~120
Output Formats Learned	4 (basic, table, JSON, debug)
Serial.print() Variations Used	8+
Coffee Consumed	1 cup (more efficient!)

Table 1: Day 2 Session Metrics

### 1.2 Primary Achievement

Created a comprehensive Serial communication demonstration showcasing multiple output formats (basic text, structured tables, JSON format, and timestamped debug logs). Most importantly, I now understand how to create professional, readable debug output that will be essential for sensor data monitoring in upcoming weeks.

#### Success Achieved

##### Deliverable Completed:

- Serial communication patterns mastered
- Multiple data type formatting (int, float, string, hex, binary)
- JSON preview implementation (ready for Week 3 MQTT)
- Professional debug output with timestamps
- Memory monitoring demonstrated

### 1.3 Improvement from Day 1

#### Improvement from Day 1

##### Notable Progress:

- **Time management:** Finished in 2h 45m vs Day 1's 3h 15m
- **Tool familiarity:** No time wasted learning Wokwi interface
- **Confidence:** Approached new concepts with less hesitation
- **Code organization:** Started with better structure from the beginning
- **Documentation:** Took notes while coding, not just after

## 2 Detailed Learning Journey

### 2.1 Phase 1: Serial Communication Fundamentals (35 minutes)

#### 2.1.1 Understanding Baud Rates

Yesterday I used 115200 baud without fully understanding what it meant. Today I dove deeper:

##### Key Learning

###### What is Baud Rate?

Baud rate = bits per second transmitted over serial connection

###### Key insights:

- 115200 baud = 115,200 bits/second
- Both sender (ESP32) and receiver (Serial Monitor) must match
- Higher = faster, but more prone to errors over long distances
- ESP32 default is 115200 (vs Arduino Uno's typical 9600)

###### Common baud rates:

- 9600 - Old standard, very reliable
- 57600 - Medium speed
- 115200 - ESP32 standard (fast, modern)
- 230400 - Very fast, sometimes unstable

**Why it matters for TAZROUT:** When debugging sensor issues, faster baud rate means I can log more data points per second without slowing down the system.

#### 2.1.2 Experimenting with Baud Rates

I tested different baud rates to see the effect:

Baud Rate	Result	Use Case
9600	Works perfectly, but slower data transmission	Long cable connections
57600	Fast and stable	Good balance
115200	Fast, recommended for ESP32	Default choice
230400	Very fast, no issues in simulation	High-speed data logging
9600 (ESP32) + 115200 (Monitor)	Gibberish characters!	Common mistake to avoid

Table 2: Baud Rate Testing Results

### Challenge Encountered

#### Challenge: Understanding the Gibberish

When I deliberately mismatched baud rates (ESP32 at 9600, Monitor at 115200), I saw weird characters like: `ÿÿ§§¶¶ØØ`

#### Why this happens:

- Serial Monitor interprets bits at wrong speed
- Bit boundaries get misaligned
- Results in incorrect character decoding

**Lesson:** This is a common debugging issue. If I see gibberish in Week 3 when testing MQTT, first thing to check: baud rate mismatch!

## 2.2 Phase 2: Data Type Formatting (50 minutes)

### 2.2.1 Working with Different Data Types

Today's main learning: how to format different types of data for display.

```

1 void demonstrateDataTypes() {
2   Serial.println("==== Data Type Formatting ====\n");
3
4   // Integer formatting
5   int sensorValue = 1023;
6   Serial.print("Decimal: ");
7   Serial.println(sensorValue);           // Output: 1023
8   Serial.print("Hexadecimal: 0x");
9   Serial.println(sensorValue, HEX);      // Output: 0x3FF
10  Serial.print("Binary: 0b");
11  Serial.println(sensorValue, BIN);      // Output: 0b1111111111
12
13  // Float formatting with precision
14  float temperature = 24.56789;
15  Serial.print("Default: ");
16  Serial.println(temperature);          // Output: 24.57 (2 decimals default)
17  Serial.print("1 decimal: ");
18  Serial.println(temperature, 1);        // Output: 24.6
19  Serial.print("3 decimals: ");
20  Serial.println(temperature, 3);        // Output: 24.568
21
22  // Boolean values
23  bool pumpActive = true;
24  Serial.print("Boolean: ");
25  Serial.println(pumpActive ? "ON" : "OFF");
26
27  // String concatenation
28  String deviceID = "ESP32_001";
29  Serial.print("Device: ");
30  Serial.println(deviceID);
31 }
```

Listing 1: Data Type Formatting Examples

### Key Learning

#### Float Precision - Important Discovery:

When I printed temperature without specifying decimals, it showed 2 by default. But I needed to control this for sensor accuracy!

**Syntax:** `Serial.println(value, decimals)`

#### Why it matters:

- Soil moisture: 1 decimal is enough (45.2%)
- Temperature: 1-2 decimals for accuracy (24.5°C)
- Humidity: 1 decimal sufficient (62.3%)

This will be crucial in Week 2 when displaying real sensor data!

### 2.2.2 Hexadecimal and Binary - When to Use Them?

Initially, I wondered: "Why would I ever need hex or binary?"

### Key Learning

#### Practical Use Cases Discovered:

##### Hexadecimal (HEX):

- Debugging I2C sensor addresses (DHT22 uses I2C)
- Viewing raw ADC values from moisture sensor
- Network MAC addresses (WiFi in Week 3)
- Memory addresses when debugging

##### Binary (BIN):

- Understanding GPIO register states
- Debugging bit manipulation for sensors
- Visualizing status flags (8 bits = 8 different states)

##### Example that made it click:

- ADC reads 1023 (decimal)
- In hex: 0x3FF
- In binary: 0b1111111111 (10 bits all high = max value)
- Seeing binary immediately shows "all bits are 1" = maximum reading

### 2.3 Phase 3: Structured Output Formats (55 minutes)

#### 2.3.1 Basic Text Format

Started simple - human-readable output:

```
1 void printBasicFormat() {
2     Serial.println("">>>> SENSOR READINGS <<<");
```

```

3 Serial.print("Temperature: ");
4 Serial.print(temperature, 1);
5 Serial.println(" C");
6
7 Serial.print("Moisture: ");
8 Serial.print(moisture);
9 Serial.println(" %");
10
11 Serial.print("Humidity: ");
12 Serial.print(humidity, 1);
13 Serial.println(" %");
14
15 Serial.print("Pump Status: ");
16 Serial.println(pumpStatus ? "ON" : "OFF");
17 Serial.println();
18 }

```

Listing 2: Basic Text Output

**Output looks like:**

```

>>> SENSOR READINGS <<<
Temperature: 24.5 C
Moisture: 65 %
Humidity: 58.3 %
Pump Status: OFF

```

**Pros:** Easy to read, quick to implement**Cons:** Hard to scan multiple readings, no structure**2.3.2 Table Format - The Game Changer**

This took me longest but was most rewarding:

```

1 void printTableFormat() {
2   Serial.println(">>> TABLE FORMAT <<<");
3   Serial.println("+-----+-----+-----+");
4   Serial.println("| Sensor | Value | Unit |");
5   Serial.println("+-----+-----+-----+");
6
7   Serial.print("| Temperature | ");
8   Serial.print(temperature, 1);
9   Serial.println(" | C | ");
10
11  Serial.print("| Moisture | ");
12  Serial.print(moisture);
13  Serial.println(" | % | ");
14
15  Serial.print("| Humidity | ");
16  Serial.print(humidity, 1);
17  Serial.println(" | % | ");
18
19  Serial.println("+-----+-----+-----+");
20 }

```

Listing 3: Table Formatted Output

**Output:**

```

>>> TABLE FORMAT <<<
+-----+-----+-----+
| Sensor | Value | Unit |

```

```
+-----+-----+-----+
| Temperature | 24.5      |   C   |
| Moisture    | 65        |   %   |
| Humidity    | 58.3      |   %   |
+-----+-----+-----+
```

### Challenge Encountered

#### Challenge: Alignment Issues

My first attempt looked terrible - values weren't aligned:

```
| Temperature | 24.5 |   C   |
| Moisture    | 65 |   %   |
```

**Problem:** Different value lengths caused misalignment

#### Solution Attempts:

1. Trial and error with spaces (tedious!)
2. Realized I need consistent width fields
3. Used manual spacing based on max expected value

**What I learned:** For production code, I'd use `printf()` or padding functions, but for learning, manual spacing helped me understand formatting.

**Time spent:** 20 minutes of frustration, but valuable lesson in attention to detail!

### 2.3.3 JSON Format - Preview for Week 3

This was exciting because I knew it connects to MQTT in Week 3!

```
1 void printJSONFormat() {
2     Serial.println("">>>> JSON FORMAT <<<");
3     Serial.println("{");
4     Serial.print("  \"node_id\": \"ESP32_001\",\\n");
5     Serial.print("  \"timestamp\": ");
6     Serial.print(millis());
7     Serial.println(",");
8     Serial.print("  \"temperature\": ");
9     Serial.print(temperature, 1);
10    Serial.println(",");
11    Serial.print("  \"moisture\": ");
12    Serial.print(moisture);
13    Serial.println(",");
14    Serial.print("  \"humidity\": ");
15    Serial.print(humidity, 1);
16    Serial.println(",");
17    Serial.print("  \"pump_status\": ");
18    Serial.println(pumpStatus ? "true" : "false");
19    Serial.println("}");
20 }
```

Listing 4: JSON Formatted Output

#### Output:

```
>>> JSON FORMAT <<<
{
  "node_id": "ESP32_001",
```

```

    "timestamp": 12456,
    "temperature": 24.5,
    "moisture": 65,
    "humidity": 58.3,
    "pump_status": false
}

```

### Key Learning

#### Understanding JSON Structure:

#### What I learned about JSON:

- Key-value pairs separated by colons
- Strings need double quotes: "key": "value"
- Numbers don't need quotes: "temperature": 24.5
- Booleans: true or false (no quotes!)
- Commas between pairs (but NOT after last one!)

#### Why this matters for TAZROUT:

- MQTT messages will use JSON format (Week 3)
- Backend expects structured data (Week 8)
- Easy for computers to parse, human-readable too
- Industry standard for IoT data exchange

**Future-proofing:** In Week 3, I'll use ArduinoJson library to generate this automatically. But learning to create it manually today helps me understand the structure!

### 2.3.4 Debug Format with Timestamps

Most practical format for real debugging:

```

1 void printDebugFormat() {
2   Serial.println(">>> DEBUG FORMAT <<<");
3
4   // Timestamp prefix
5   Serial.print("[");
6   Serial.print(millis());
7   Serial.print(" ms]");
8   Serial.print("Reading #");
9   Serial.println(readingCount);
10
11  // Info level log
12  Serial.print("[INFO] Temp=");
13  Serial.print(temperature, 1);
14  Serial.print("C, Moisture=");
15  Serial.print(moisture);
16  Serial.print("%, Humidity=");
17  Serial.print(humidity, 1);
18  Serial.println("%");
19
20  // Conditional alerts
21  if (pumpStatus) {

```

```

22     Serial.println("[ALERT] Pump activated - low moisture detected!");
23 } else {
24     Serial.println("[STATUS] Soil moisture adequate, pump off");
25 }
26
27 // System diagnostics
28 Serial.print("[SYSTEM] Free heap: ");
29 Serial.print(ESP.getFreeHeap());
30 Serial.println(" bytes");
31 }

```

Listing 5: Professional Debug Output

**Output:**

```

>>> DEBUG FORMAT <<<
[12456 ms] Reading #5
[INFO] Temp=24.5C, Moisture=65%, Humidity=58.3%
[STATUS] Soil moisture adequate, pump off
[SYSTEM] Free heap: 296712 bytes

```

**Key Learning****Professional Logging Patterns Discovered:****Log Levels:**

- [INFO] - Normal operation data
- [ALERT] - Something needs attention
- [ERROR] - Something went wrong
- [DEBUG] - Detailed debugging info
- [SYSTEM] - Hardware/memory status

**Timestamp Benefits:**

- `millis()` shows time since boot in milliseconds
- Helps correlate events: "pump activated at 5000ms, sensor read at 5200ms"
- Essential for debugging timing-related issues
- Will be crucial for non-blocking code in Days 3-4

**Real-world application:** When debugging Week 6 network errors, I'll be able to see exactly when connection dropped and how long until reconnection!

## 2.4 Phase 4: Simulating Sensor Data (25 minutes)

### 2.4.1 Creating Realistic Sensor Variations

To make the demo realistic, I simulated changing sensor values:

```

1 // Global variables
2 float temperature = 24.5;
3 int moisture = 65;
4 float humidity = 58.3;
5 bool pumpStatus = false;
6 unsigned long readingCount = 0;

```

```

7
8 void updateSimulatedReadings() {
9   // Simulate temperature fluctuation ( 1 degree)
10  temperature += random(-10, 10) / 10.0; // -1.0 to +1.0
11  temperature = constrain(temperature, 20.0, 30.0);
12
13 // Simulate moisture changes ( 5 %)
14 moisture += random(-5, 5);
15 moisture = constrain(moisture, 50, 80);
16
17 // Simulate humidity changes ( 2 %)
18 humidity += random(-20, 20) / 10.0; // -2.0 to +2.0
19 humidity = constrain(humidity, 40.0, 80.0);
20
21 // Pump logic: activate if moisture too low
22 pumpStatus = (moisture < 60);
23
24 // Increment reading counter
25 readingCount++;
26 }
```

Listing 6: Simulating Dynamic Sensor Data

### Key Learning

#### **Understanding random() and constrain():**

##### **random(min, max):**

- Returns random integer from min to max-1
- `random(-10, 10)` gives -10 to 9
- Dividing by 10.0 gives decimal values: -1.0 to 0.9

##### **constrain(value, min, max):**

- Keeps value within bounds
- If value  $<$  min, returns min
- If value  $>$  max, returns max
- Otherwise returns value unchanged

#### **Why this matters:**

- Real sensors have ranges (e.g., DHT22: -40 to 80°C)
- Prevents unrealistic values in simulation
- Good practice for data validation (Week 6 topic!)

## 2.5 Phase 5: Memory Monitoring (15 minutes)

### 2.5.1 ESP32-Specific Functions

Discovered useful ESP32 functions:

```

1 void printSystemInfo() {
2   Serial.println("\n==== ESP32 SYSTEM INFO ====");
3 }
```

```

4 // Free heap memory
5 Serial.print("Free Heap: ");
6 Serial.print(ESP.getFreeHeap());
7 Serial.println(" bytes");
8
9 // Chip ID (unique identifier)
10 Serial.print("Chip ID: ");
11 Serial.println(ESP.getEfuseMac(), HEX);
12
13 // CPU frequency
14 Serial.print("CPU Frequency: ");
15 Serial.print(ESP.getCpuFreqMHz());
16 Serial.println(" MHz");
17
18 // Flash size
19 Serial.print("Flash Size: ");
20 Serial.print(ESP.getFlashChipSize());
21 Serial.println(" bytes");
22 }

```

Listing 7: ESP32 System Information

**Output on my simulated ESP32:**

```

==== ESP32 SYSTEM INFO ====
Free Heap: 296712 bytes
Chip ID: 24:0A:C4:12:34:56
CPU Frequency: 240 MHz
Flash Size: 4194304 bytes (4MB)

```

**Key Learning****Why Monitor Memory?****Free Heap Importance:**

- Shows available RAM for program use
- ESP32 has 300KB RAM total
- If heap gets too low (<10KB), crashes can occur
- Critical for detecting memory leaks

**Future use in TAZROUT:**

- Week 3: Monitor memory during WiFi/MQTT operations
- Week 4: Check for leaks in message buffering
- Week 6: Include in system health reports

**Personal note:** This feels like professional embedded programming - monitoring system resources proactively rather than just coding features!

### 3 Complete Working Code

#### 3.1 Final Day 2 Implementation

```
1 /*
2  * TAZROUT IoT Irrigation System - ESP32 Module
3  * Day 2: Serial Communication Patterns
4  *
5  * Developer: KHENFRI Moussa
6  * Date: February 2, 2026
7  * Week: 1 - ESP32 Fundamentals
8  *
9  * Learning Objectives:
10 * - Master Serial communication and formatting
11 * - Practice different output formats (text, table, JSON, debug)
12 * - Understand data type formatting
13 * - Preview JSON structure for Week 3 MQTT
14 */
15
16 // Simulated sensor values
17 float temperature = 24.5;
18 int moisture = 65;
19 float humidity = 58.3;
20 bool pumpStatus = false;
21
22 // System tracking
23 unsigned long readingCount = 0;
24 unsigned long lastUpdate = 0;
25 const unsigned long UPDATE_INTERVAL = 5000; // Update every 5 seconds
26
27 void setup() {
28     // Initialize Serial at ESP32 standard baud rate
29     Serial.begin(115200);
30     delay(1000); // Wait for Serial to stabilize
31
32     // Print startup banner
33     printStartupBanner();
34
35     // Demonstrate different data types
36     demonstrateDataTypes();
37
38     // Show system information
39     printSystemInfo();
40
41     Serial.println("\n==== Starting sensor simulation ====\n");
42 }
43
44 void loop() {
45     // Update readings every 5 seconds
46     if (millis() - lastUpdate >= UPDATE_INTERVAL) {
47         lastUpdate = millis();
48
49         // Update simulated sensor values
50         updateSimulatedReadings();
51
52         // Print in different formats (rotate through them)
53         int formatType = (readingCount % 4);
54
55         switch(formatType) {
56             case 0:
57                 printBasicFormat();
58                 break;
59             case 1:
60                 printTableFormat();
61                 break;
62             case 2:
```

```
63     printJSONFormat();
64     break;
65   case 3:
66     printDebugFormat();
67     break;
68 }
69
70 Serial.println("\n" + String('—', 50) + "\n");
71 }
72 }
73
74 void printStartupBanner() {
75   Serial.println("\n\n");
76   Serial.println("===== ");
77   Serial.println(" TAZROUT ESP32 - Serial Communication ");
78   Serial.println("           Day 2 Demo           ");
79   Serial.println("===== ");
80   Serial.println();
81   Serial.println("Device: ESP32 DevKit v1");
82   Serial.println("Baud Rate: 115200");
83   Serial.println("Developer: KHENFRI Moussa");
84   Serial.println("===== ");
85   Serial.println();
86 }
87
88 void demonstrateDataTypes() {
89   Serial.println(">>> DATA TYPE FORMATTING <<<\n");
90
91   int value = 1023;
92   Serial.print("Integer (decimal): ");
93   Serial.println(value);
94   Serial.print("Integer (hex): 0x");
95   Serial.println(value, HEX);
96   Serial.print("Integer (binary): 0b");
97   Serial.println(value, BIN);
98
99   float temp = 24.56789;
100  Serial.print("Float (default): ");
101  Serial.println(temp);
102  Serial.print("Float (1 decimal): ");
103  Serial.println(temp, 1);
104  Serial.print("Float (3 decimals): ");
105  Serial.println(temp, 3);
106
107  bool status = true;
108  Serial.print("Boolean: ");
109  Serial.println(status ? "true" : "false");
110
111  String id = "ESP32_001";
112  Serial.print("String: ");
113  Serial.println(id);
114
115  Serial.println();
116 }
117
118 void updateSimulatedReadings() {
119   temperature += random(-10, 10) / 10.0;
120   temperature = constrain(temperature, 20.0, 30.0);
121
122   moisture += random(-5, 5);
123   moisture = constrain(moisture, 50, 80);
124
125   humidity += random(-20, 20) / 10.0;
```

```
126     humidity = constrain(humidity, 40.0, 80.0);
127
128     pumpStatus = (moisture < 60);
129     readingCount++;
130 }
131
132 void printBasicFormat() {
133     Serial.println(">>> BASIC FORMAT <<<");
134     Serial.print("Temperature: ");
135     Serial.print(temperature, 1);
136     Serial.println(" C");
137     Serial.print("Moisture: ");
138     Serial.print(moisture);
139     Serial.println(" %");
140     Serial.print("Humidity: ");
141     Serial.print(humidity, 1);
142     Serial.println(" %");
143     Serial.print("Pump: ");
144     Serial.println(pumpStatus ? "ON" : "OFF");
145 }
146
147 void printTableFormat() {
148     Serial.println(">>> TABLE FORMAT <<<");
149     Serial.println("+-----+-----+-----+");
150     Serial.println("| Sensor | Value | Unit |");
151     Serial.println("+-----+-----+-----+");
152     Serial.print(" | Temperature | ");
153     Serial.print(temperature, 1);
154     Serial.println(" | C |");
155     Serial.print(" | Moisture | ");
156     Serial.print(moisture);
157     Serial.println(" | % |");
158     Serial.print(" | Humidity | ");
159     Serial.print(humidity, 1);
160     Serial.println(" | % |");
161     Serial.println("+-----+-----+-----+");
162 }
163
164 void printJSONFormat() {
165     Serial.println(">>> JSON FORMAT <<<");
166     Serial.println("{");
167     Serial.print("  \"node_id\": \"ESP32_001\",\\n");
168     Serial.print("  \"reading\": ");
169     Serial.print(readingCount);
170     Serial.println(",");
171     Serial.print("  \"timestamp\": ");
172     Serial.print(millis());
173     Serial.println(",");
174     Serial.print("  \"temperature\": ");
175     Serial.print(temperature, 1);
176     Serial.println(",");
177     Serial.print("  \"moisture\": ");
178     Serial.print(moisture);
179     Serial.println(",");
180     Serial.print("  \"humidity\": ");
181     Serial.print(humidity, 1);
182     Serial.println(",");
183     Serial.print("  \"pump_status\": ");
184     Serial.println(pumpStatus ? "true" : "false");
185     Serial.println("}");
186 }
187
188 void printDebugFormat() {
```

```

189 Serial.println("">>>> DEBUG FORMAT <<<");
190 Serial.print("[");
191 Serial.print(millis());
192 Serial.print(" ms] Reading #");
193 Serial.println(readingCount);
194
195 Serial.print("[INFO] Temp=");
196 Serial.print(temperature, 1);
197 Serial.print("C, Moisture=");
198 Serial.print(moisture);
199 Serial.print("%, Humidity=");
200 Serial.print(humidity, 1);
201 Serial.println("%");
202
203 if (pumpStatus) {
204   Serial.println("[ALERT] Pump ON - low moisture!");
205 } else {
206   Serial.println("[STATUS] Moisture OK, pump OFF");
207 }
208
209 Serial.print("[SYSTEM] Free heap: ");
210 Serial.print(ESP.getFreeHeap());
211 Serial.println(" bytes");
212 }
213
214 void printSystemInfo() {
215   Serial.println("">>>> ESP32 SYSTEM INFO <<<");
216   Serial.print("Free Heap: ");
217   Serial.print(ESP.getFreeHeap());
218   Serial.println(" bytes");
219   Serial.print("CPU Frequency: ");
220   Serial.print(ESP.getCpuFreqMHz());
221   Serial.println(" MHz");
222   Serial.println();
223 }
```

Listing 8: Complete Serial Communication Demo - Day 2

## 4 Technical Knowledge Acquired

### 4.1 Serial Communication Concepts

Concept	Understanding Achieved
Baud Rate	Bits per second; must match on both ends; 115200 is ESP32 standard
Serial.print() vs println()	print() stays on same line; println() adds newline
Format Specifiers	Second parameter controls output format (HEX, BIN, decimal places)
Buffer Overflow	Serial has limited buffer; printing too fast can lose data
Blocking Nature	Serial.print() waits until data is sent (blocks other code)

Table 3: Serial Communication Concepts Mastered

### 4.2 Data Formatting Patterns

Format	Best For	When to Use in TAZROUT
Basic Text	Quick debugging, human reading	Daily development, quick checks

Format	Best For	When to Use in TAZROUT
Table Format	Comparing multiple values, organized data	Week 2: Comparing sensor readings over time
JSON Format	Machine parsing, MQTT messages	Week 3+: All network communication
Debug Format	Troubleshooting, timestamps, log levels	Week 6: Error tracking, Week 8: Integration testing

Table 4: Output Format Use Cases

### 4.3 New Functions Learned

Function	Purpose & Usage
Serial.print(value, format)	Print with specific format (HEX, BIN, decimal places)
random(min, max)	Generate random integer from min to max-1
constrain(val, min, max)	Keep value within bounds
millis()	Milliseconds since boot; used for timestamps and timing
ESP.getFreeHeap()	Get available RAM in bytes
ESP.getCpuFreqMHz()	Get CPU frequency (usually 240 MHz)
String(char, count)	Create string by repeating character count times

Table 5: New Functions Mastered Today

## 5 Challenges and Solutions

### 5.1 Challenge Log

#	Challenge	Solution	Time
1	Table alignment issues	Manual spacing adjustment; learned about field width	20 min
2	JSON comma placement	Studied JSON syntax rules; comma after each pair except last	10 min
3	Understanding millis() overflow	Researched: millis() overflows after 49 days; use proper comparison	12 min
4	Float division by integer	Learned: divide by 10.0 (float) not 10 (int) for decimal results	5 min

Table 6: Day 2 Challenge Log

### Challenge Encountered

#### **Challenge: Understanding millis() Overflow**

While implementing timestamps, I read that `millis()` overflows after approximately 49 days.

#### **What does "overflow" mean?**

- `millis()` returns unsigned long (32-bit)
- Maximum value: 4,294,967,295 milliseconds
- This equals about 49.7 days
- After max, it wraps back to 0

#### **Why it matters:**

- If TAZROUT runs continuously, this will happen!
- Need to handle overflow in timing calculations
- Week 3-4: Learn proper `millis()` comparison techniques

**For now:** Noted as something to learn properly in Days 3-4 when studying non-blocking code.

## 6 Skills Development Matrix

Skill	After Day 1	After Day 2	Growth
Wokwi Platform	6/10	8/10	+2
ESP32 Knowledge	4/10	5/10	+1
Arduino C/C++	5/10	7/10	+2
Serial Debugging	5/10	8/10	+3
Code Organization	4/10	6/10	+2
Data Formatting	2/10	7/10	+5
Professional Practices	3/10	6/10	+3

Table 7: Skill Development Progress

### Personal Reflection

#### **Notable Improvement: Data Formatting**

Jumped from 2/10 to 7/10 in one day! This shows focused learning on a specific topic really pays off.

#### **Areas still needing work:**

- Non-blocking code patterns (Days 3-4)
- Analog input (Day 5)
- PWM control (Day 6)

## 7 Connection to TAZROUT Project

### 7.1 How Today's Skills Enable Future Features

Day 2 Skill	TAZROUT Application	Week
JSON formatting	MQTT message payload structure	Week 3
Debug logging	Troubleshooting sensor issues	Week 2-8
Timestamp tracking	Event logging for irrigation cycles	Week 5
Memory monitoring	System health reporting	Week 6
Data type formatting	Sensor calibration displays	Week 2
Table output	Multi-sensor comparison	Week 2

Table 8: Day 2 Skills in TAZROUT Context

### 7.2 JSON Preview - Week 3 Connection

#### Key Learning

#### Today's JSON Practice Prepares Me For: Week 3 MQTT Publishing:

```

1 // What I'll send to MQTT broker
2 {
3   "node_id": "ESP32_001",
4   "timestamp": 1675234567,
5   "moisture": 45.2,
6   "temperature": 24.5,
7   "humidity": 62.0,
8   "status": "OK"
9 }
```

#### Week 4 Command Reception:

```

1 // What I'll receive from MQTT
2 {
3   "node_id": "ESP32_001",
4   "command": "activate_pump",
5   "duration": 300
6 }
```

Understanding JSON structure today means I won't be overwhelmed when using ArduinoJson library in Week 3!

## 8 Reflections and Insights

### 8.1 What Surprised Me Today

#### Personal Reflection

##### **Positive Surprises:**

1. **Efficiency gain from Day 1 experience:** Completed in 2h45m vs yesterday's 3h15m. The learning curve is real!
2. **Serial communication is more powerful than I thought:** It's not just for "Hello World" - it's a professional debugging tool.
3. **JSON is actually simple:** I was intimidated by it, but manually creating JSON helped me understand it's just structured text.
4. **Format matters:** Well-formatted output makes debugging 10x easier. Table format is a game-changer.

##### **Challenges I Expected But Didn't Face:**

- Thought baud rates would be confusing, but the concept clicked quickly
- Expected JSON to be harder - manual construction was educational

### 8.2 Key Realizations

#### Key Learning

##### **The Power of Visualization:**

Today I realized: **how you display data is as important as collecting it.**

##### **Examples:**

- Messy text: "temp24.5moist65humid58.3" - hard to read
- Formatted table: Immediately shows patterns
- JSON: Both human AND machine readable
- Debug logs: Tell a story of what happened when

This will be crucial when debugging sensor issues in Week 2 and network problems in Week 6!

#### Personal Reflection

##### **Building Blocks Mindset:**

Day 1: Basic output (LED blink)

Day 2: Structured output (formatted data)

Day 3-4: Will add timing control (non-blocking)

Day 5: Will add input reading (sensors)

I can see how each day builds on the last. This isn't random exercises - it's a carefully designed progression!

### 8.3 Personal Growth

#### Improvement from Day 1

##### Confidence Progress:

**Before Day 2:** 6/10 (cautiously optimistic)

**After Day 2:** 7/10 (feeling capable)

##### What boosted confidence:

- Completed on time (actually under planned time!)
- Didn't need as much documentation lookup
- Could explain concepts to myself while coding
- Made fewer mistakes, fixed them faster

**Specific improvement:** I caught a JSON comma error *before* running the code. Yesterday I would have run it, seen the error, then fixed it. I'm learning to think ahead!

## 9 Deliverables and Documentation

### 9.1 Wokwi Project

#### Project Details

**Project Name:** ESP32\_Serial\_Communication\_Day2

**Status:** Working perfectly

##### Features:

- 4 output formats rotating every 5 seconds
- Simulated sensor data with realistic variations
- Memory monitoring
- Professional debug logging

**Last Updated:** February 2, 2026, 16:30

### 9.2 Code Quality Assessment

Criterion	Score	Notes
Functionality	5/5	All formats working correctly
Readability	5/5	Well-organized, clear function names
Documentation	5/5	Every function documented
Efficiency	4/5	Some repeated code, could use functions
Maintainability	5/5	Easy to add new formats
<b>Overall</b>	<b>24/25</b>	<b>96% - Excellent quality</b>

Table 9: Code Quality Self-Assessment

**Improvement from Day 1:** 84% → 96% (+12%)

## 10 Time Management Analysis

### 10.1 Planned vs Actual Time Breakdown

Activity	Planned	Actual	Variance
Baud Rate Study	20 min	35 min	+15 min
Data Type Formatting	30 min	50 min	+20 min
Output Formats	40 min	55 min	+15 min
Testing & Debugging	20 min	25 min	+5 min
Documentation	30 min	30 min	0 min
Breaks	0 min	10 min	+10 min
<b>Total</b>	<b>2h 20m</b>	<b>2h 45m</b>	<b>+25 min</b>

Table 10: Day 2 Time Analysis

Improvement from Day 1

**Time Management Improvement:**

**Day 1 variance:** +55 minutes over plan

**Day 2 variance:** +25 minutes over plan

**Improvement:** 30 minutes better time estimation!

**Why I'm improving:**

- Better familiarity with tools
- More accurate estimation of task complexity
- Less time lost to tool confusion
- Taking scheduled breaks (10 min) helped focus

## 11 Resources Used

### 11.1 Documentation Consulted

1. Arduino Serial Reference - <https://www.arduino.cc/reference/en/language/functions/communication/serial/>
  - All Serial functions documented
  - Format specifier examples
2. JSON.org - <https://www.json.org/>
  - Validated my JSON structure
  - Learned proper syntax rules
3. ESP32 Arduino Core Documentation
  - ESP.getFreeHeap() and system functions
  - Memory management information
4. Random Nerd Tutorials - ESP32 Serial
  - Practical examples
  - Common pitfalls explained

## 11.2 New Bookmarks

- JSON validator tool (for checking JSON syntax)
- millis() tutorial (for Days 3-4)
- State machine design patterns

# 12 Self-Assessment Against Learning Objectives

## 12.1 Day 2 Objectives Evaluation

Objective	Met?	Evidence
Configure Serial Monitor	Yes	Used 115200 baud, experimented with different rates
Implement formatted output	Yes	Created 4 different output formats (basic, table, JSON, debug)
Work with different data types	Yes	Demonstrated int, float, bool, String, HEX, BIN formatting
Practice debugging techniques	Yes	Implemented professional debug logging with timestamps
Understand baud rates	Yes	Tested multiple rates, understand mismatch issues

Table 11: Learning Objectives Assessment - 100% Complete

Success Achieved

**Perfect Score: 5/5 Objectives Met!**

All planned objectives achieved. Day 2 was a complete success.

# 13 Success metrics

## 13.1 Quantitative Summary

Metric	Value
Total Time Invested	2 hours 45 minutes
Lines of Code Written	120 lines
Code Comments Written	25 lines
Wokwi Projects Created	2 projects
Output Formats Implemented	4 formats
New Functions Learned	7 functions
Debugging Sessions	4 issues
Documentation Pages	18 pages (this report)
Coffee Consumed	1 cup

Table 12: Day 2 Statistics

### 13.2 Cumulative Week 1 Stats

Metric	Day 1	Day 2	Total
Time Invested	3h 15m	2h 45m	6h 0m
Code Lines	45	120	165
Projects Created	3	2	5
Functions Learned	7	7	14

Table 13: Week 1 Cumulative Progress

**Average efficiency:** Getting better! Day 2 was 16% faster than Day 1.

## 14 Conclusion and Personal Notes

### 14.1 Overall Assessment

#### Success Achieved

##### Day 2: EXCELLENT SUCCESS

Today exceeded expectations in every way:

- All objectives met (5/5)
- Completed under 3 hours (efficient!)
- Code quality improved to 96%
- Gained practical, applicable skills
- Built foundation for Week 3 JSON/MQTT

**Confidence level:** 7/10 (up from 6/10 yesterday)

### 14.2 What I'm Most Proud Of Today

1. **Table formatting persistence:** Spent 20 minutes getting alignment right. Could have settled for "good enough" but pushed for perfection.
2. **JSON understanding:** Didn't just copy examples - I understand WHY each comma, quote, and bracket is there.
3. **Professional logging implementation:** The debug format with timestamps looks like real production code!
4. **Time management:** Took a planned 10-minute break instead of powering through. Came back more focused.

### 14.3 Honest Self-Reflection

#### Personal Reflection

##### What I Struggled With:

- Table alignment - tedious but necessary
- Understanding millis() overflow - still need more research
- Resisting urge to jump to Days 3-4 early

##### What Came Naturally:

- Code organization - structured functions from start
- Debugging - spotted issues before running code
- Documentation - writing comments as I coded

##### Unexpected Insight:

Serial debugging isn't just a tool - it's an art form. The way you format output can make the difference between finding a bug in 5 minutes vs 5 hours.  
This will be invaluable when debugging complex issues in Weeks 5-6!

### 14.4 Personal Commitment Update

#### Renewed Commitment

After 2 successful days, I renew my commitment to:

- Quality over speed (but getting faster naturally!)
- Understanding over memorization
- Professional documentation standards
- Taking breaks when needed
- Celebrating small victories

**New addition:** Share knowledge! When I understand something well (like JSON), I'll try to explain it to others (rubber duck debugging or team members).

*Signed (digitally),  
February 2, 2026*

#### End of Day 2 Learning Report

Next: Days 3-4 - Traffic Light State Machine

*TAZROUT ESP32 Module Development  
Developer: KHENFRI Moussa*