Project Task 1: Preparing the Embedded Devices

Gabriel Frank, Ryan Botwinick, Jacob Frankel

GitHub Repository

Part A

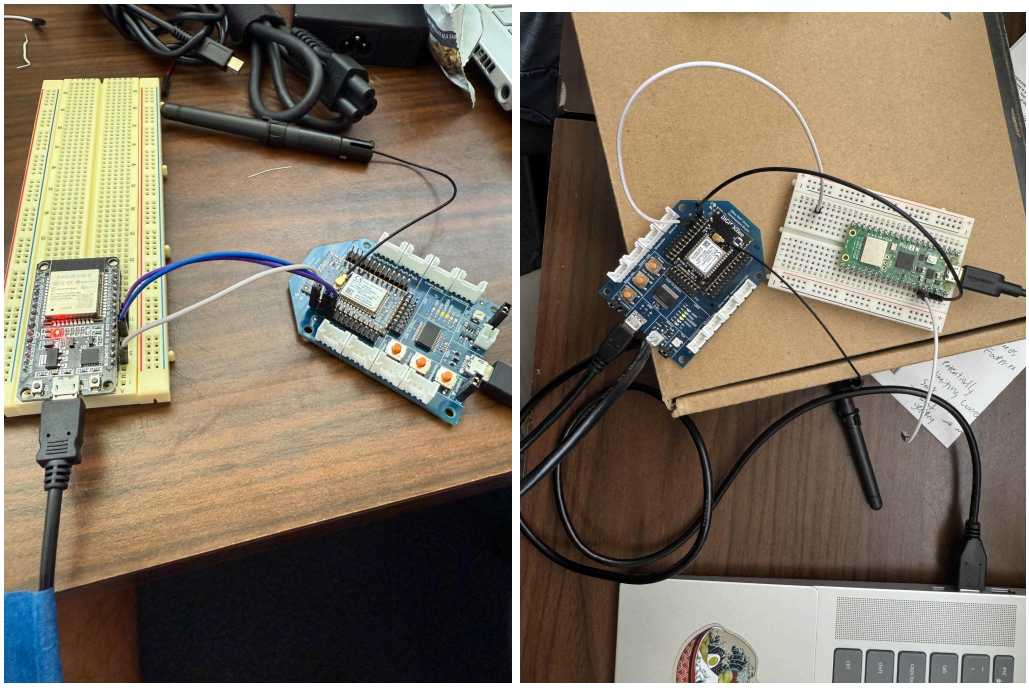

Figure 1: *Embedded devices and XBee radio connections*

Part B

task1.py

```python
from machine import UART, Pin
import time

# Initialize UART0 on GPIO 16 (TX) and 17 (RX)
uart = UART(0, baudrate=9600, tx=Pin(16), rx=Pin(17))

# Transmit to this XBee's 64-bit address (replace with your target)
DEST_ADDR_64 = bytes([0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])  # example address

# === Transparent Mode ===
def send_transparent(message: str):
    print("Sending transparent")
    uart.write(message + '\r\n')  # Append \r\n for readability in XCTU

# === API Mode (Transmit Request Frame) ===
def send_api_frame(payload: str, dest_addr: bytes):
    print("Sending API frame")
    frame_type = 0x10
    frame_id = 0x01
    dest16 = b'\xFF\xFE'
    broadcast_radius = 0x00
    options = 0x00
    rf_data = payload.encode('utf-8')

    # Compose frame data (not including 7E and length)
    frame_data = (
        bytes([frame_type, frame_id]) +
        dest_addr +
        dest16 +
        bytes([broadcast_radius, options]) +
        rf_data
    )

    # Compute frame length and checksum
    length = len(frame_data)
    length_bytes = length.to_bytes(2, 'big')
    checksum = 0xFF - (sum(frame_data) & 0xFF)

    # Full API frame
    full_frame = b'\x7E' + length_bytes + frame_data + bytes([checksum])

    #print(full_frame)

    uart.write(full_frame)

# === Main loop ===
while True:
    # Choose ONE:
    send_transparent("Hello World")  # for transparent mode
    #send_api_frame("Hello World", DEST_ADDR_64)  # for API mode

    time.sleep(1)
```

Figure 2: *Screenshot of the script for sending the "Hello World" text*

Part C

```
#define TXD1 17
#define RXD1 16
#define USE_API_MODE true  // change to change the mode sending in

uint8_t dest_addr[8] = { 0x00, 0x13, 0xA2, 0x00, 0x41, 0xD0, 0x0C, 0xF1 }; // MAC for sink node

// function to configure and send the API mode, used packet from Zigbee
void send_api_frame(const char* payload) {
  uint8_t frame_type = 0x10;
  uint8_t frame_id = 0x01;
  uint8_t dest16[2] = { 0xFF, 0xFE };
  uint8_t broadcast_radius = 0x00;
  uint8_t options = 0x00;

  size_t payload_len = strlen(payload);
  size_t frame_data_len = 14 + payload_len;

  uint8_t frame[100]; // large enough for a big frame

  // header information
  frame[0] = 0x7E;
  frame[1] = (frame_data_len >> 8) & 0xFF;
  frame[2] = frame_data_len & 0xFF;

  size_t i = 3;
  frame[i++] = frame_type;
  frame[i++] = frame_id;

  // address information
  for (int j = 0; j < 8; j++) frame[i++] = dest_addr[j];
  frame[i++] = dest16[0];
  frame[i++] = dest16[1];
  frame[i++] = broadcast_radius;
  frame[i++] = options;

  // writes the message to the frame
  for (int j = 0; j < payload_len; j++) frame[i++] = payload[j];

  // help with error control
  uint8_t checksum = 0;
  for (int j = 3; j < i; j++) checksum += frame[j];
  frame[i++] = 0xFF - checksum;

  // writes message
  Serial1.write(frame, i);
}

void send_transparent(const char* message) {
  // sends in transparent mode
  Serial1.println(message);
}


void setup() {
  // sets up baud rate and serial communication
  Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1); // 1 stop and start bit, 8 data, no parity, set Rx and Tx pins
  delay(1000);
}

// CHANGE MODE AT THE TOP
void loop() {
  if (USE_API_MODE) {
    send_api_frame("2025"); // case for api frame
  } else {
    send_transparent("2025"); // case for transparent mode
  }
  delay(1000); // delay for visability
}
```

Figure 3: *Screenshot of the script for sending the "2025" number*

Part D

To create the script to send text our team used a PiPico with MicroPython. The functions used to send in transparent mode used the string parameter. Transparent mode is the simpler of the two focusing on writing the raw data to the Xbee which allows for the direct writing of the string. When working in API mode we passed both the message as a string and the MAC address of the sink node to properly create a packet for the Xbee. When working with the ESP32 similar processes were applied for the transparent and API modes, but the data was sent in character format. This allowed for easier string manipulation in the Arduino C++ code. In both implementations UART was used to communicate data from the ESP32 and PiPico to the sink Xbee.

Part E

In transparent mode, the script sends raw data directly over UART without any additional formatting or structure. Whatever string is passed (e.g., "Hello World") is transmitted byte-for-byte as-is. This mode behaves like a basic serial passthrough, making it simple but limited in terms of control and addressing. In contrast, API mode requires the data to be wrapped in a well-defined frame structure that the XBee understands. This includes a start delimiter (`0x7E`), a two-byte length field, a frame type (`0x10` for Transmit Request), a frame ID, a 64-bit destination address, a 16-bit address (typically `0xFFFE` if unknown), broadcast radius, options byte, the payload itself, and a final checksum byte calculated as `0xFF` minus the 8-bit sum of all bytes from the frame type to the end of the payload. This structured approach allows for features like addressing, acknowledgments, and delivery status, which are not possible in transparent mode.

Part F

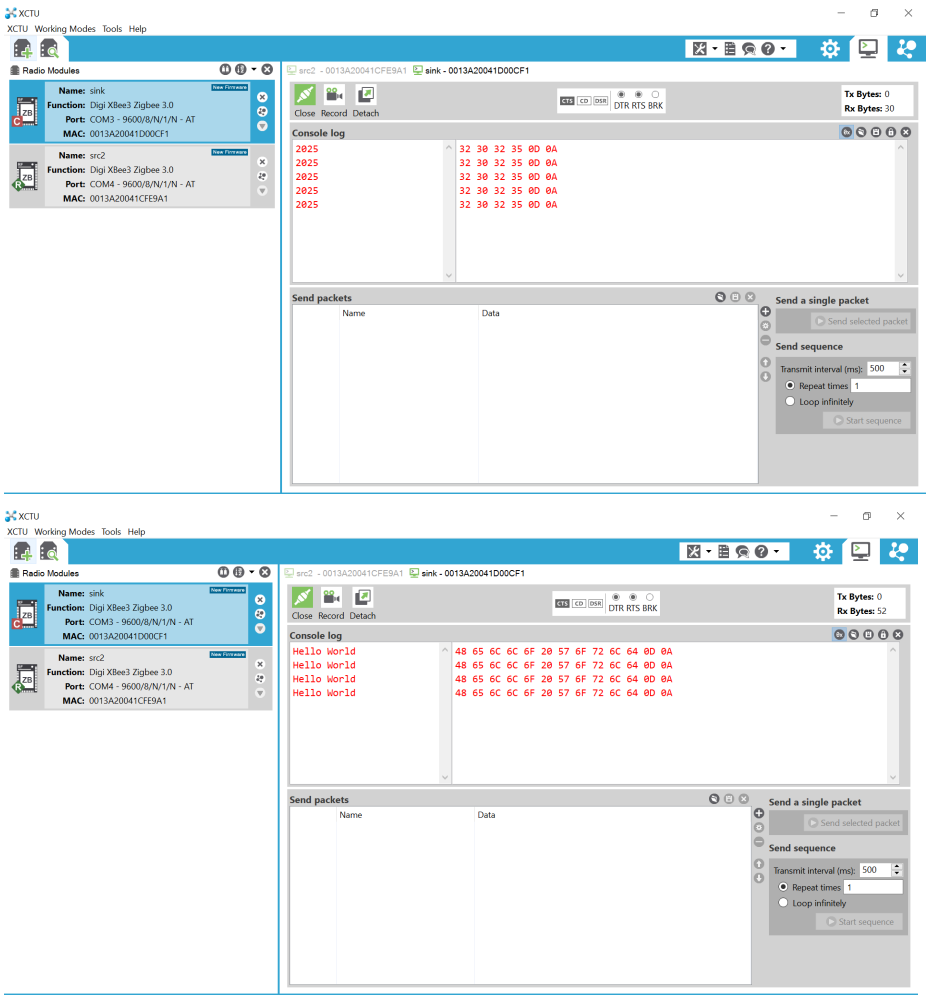

Figure 4: *Screenshots of XCTU software receiving data from the embedded devices in Transparent mode*

Part G

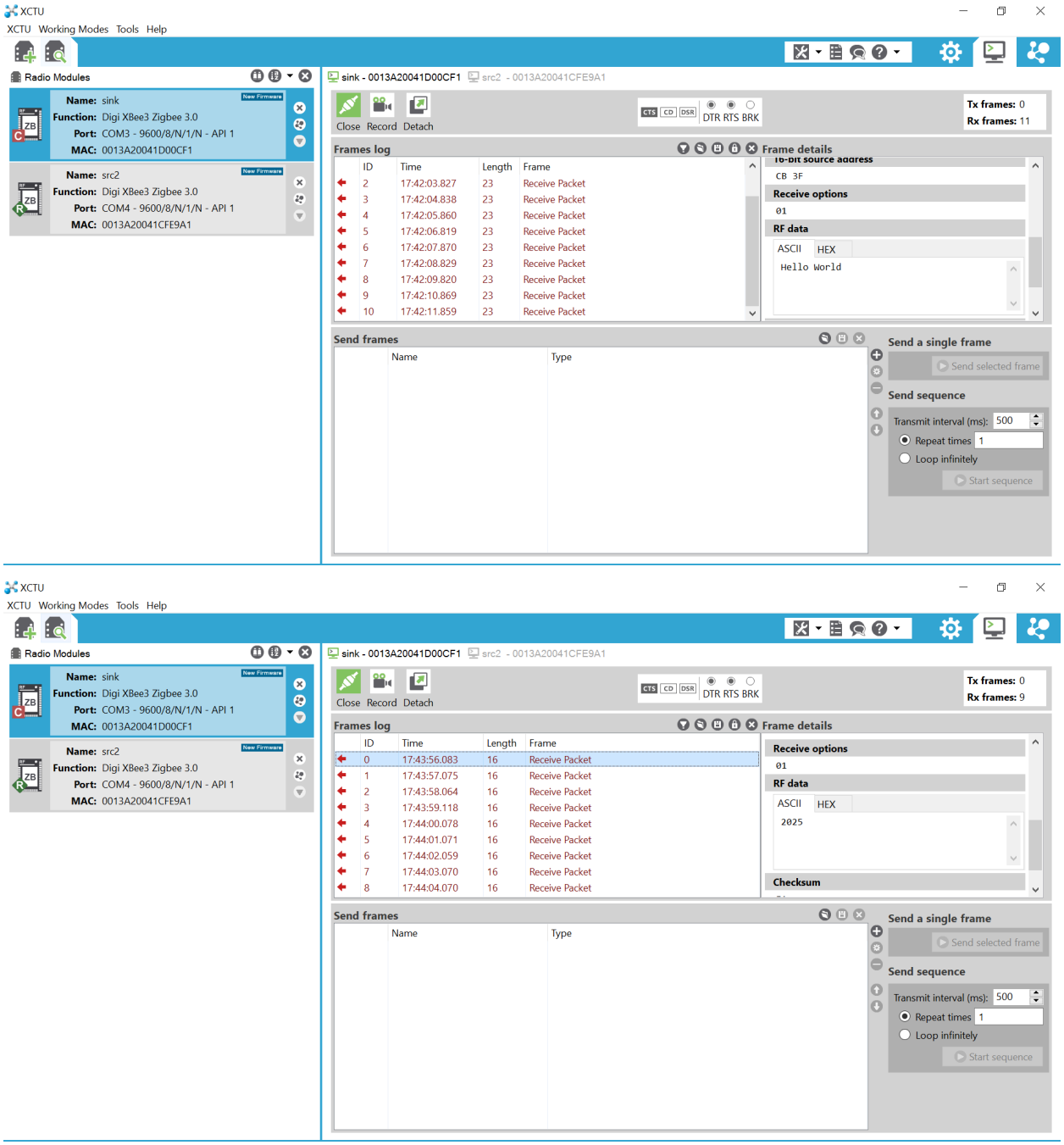

Figure 5: *Screenshots of XCTU software receiving data from the embedded devices in API mode*

Part H

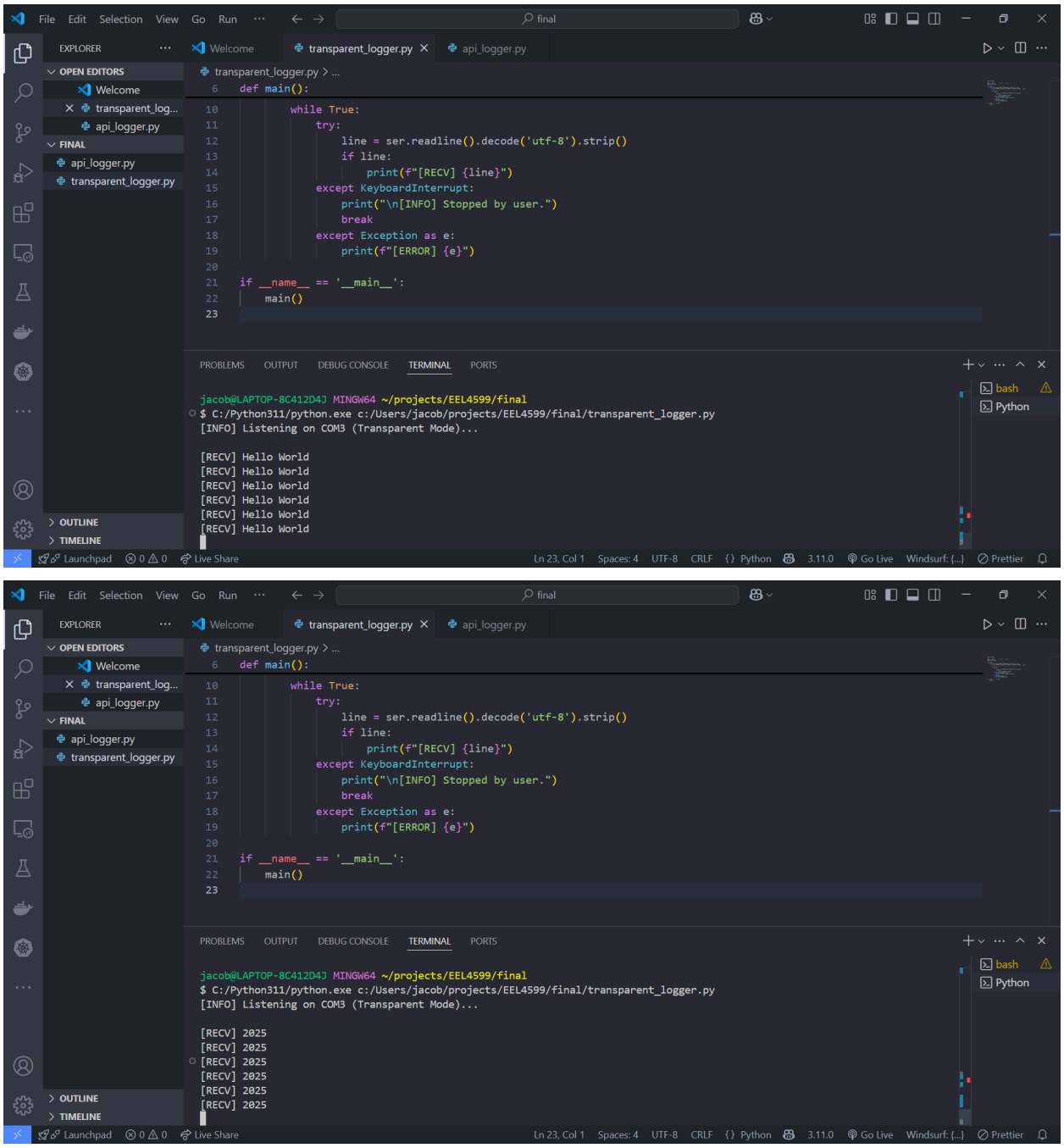

Figure 6: *Screenshots of the Python script receiving data from embedded devices in Transparent mode*

Part I

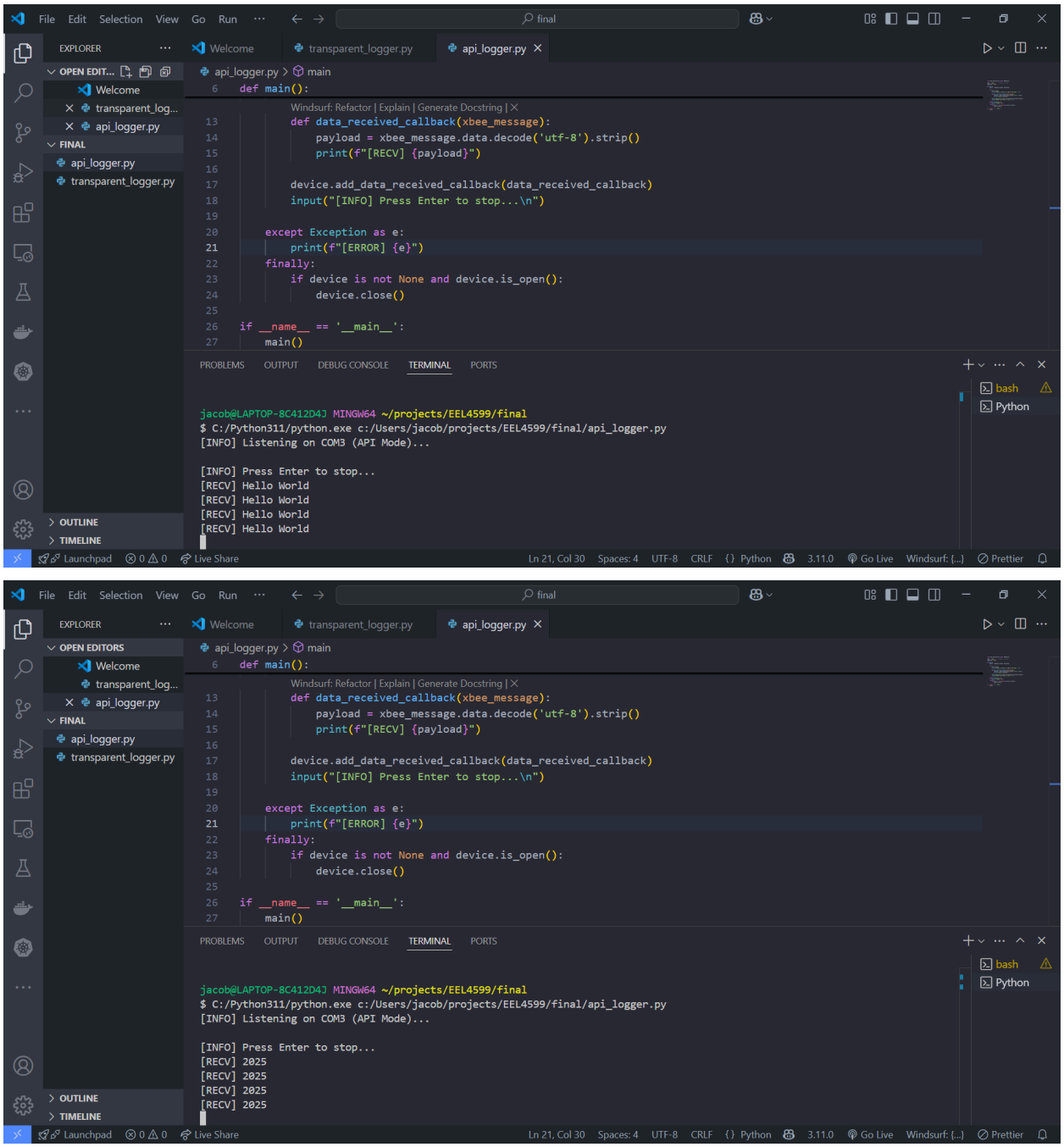

Figure 7: *Screenshots of the Python script receiving data from embedded devices in API mode*

Part J

We used only a single script for receiving text AND numbers. This is because the data transmitted from the embedded devices to the source nodes is done through UART, where raw serial data is sent to the Xbees. At this stage there is no difference between text and numbers. In essence, all text is numbers. Once the raw, non-decoded numerical data is sent to the sink, the Python script we use decodes that numerical data using UTF-8, which is a Unicode standard to convert these numbers into text. Every symbol has a UTF-8 counterpart, even numbers. Thus, regardless of whether text or numbers were sent, they get decoded all the same by the Python script.