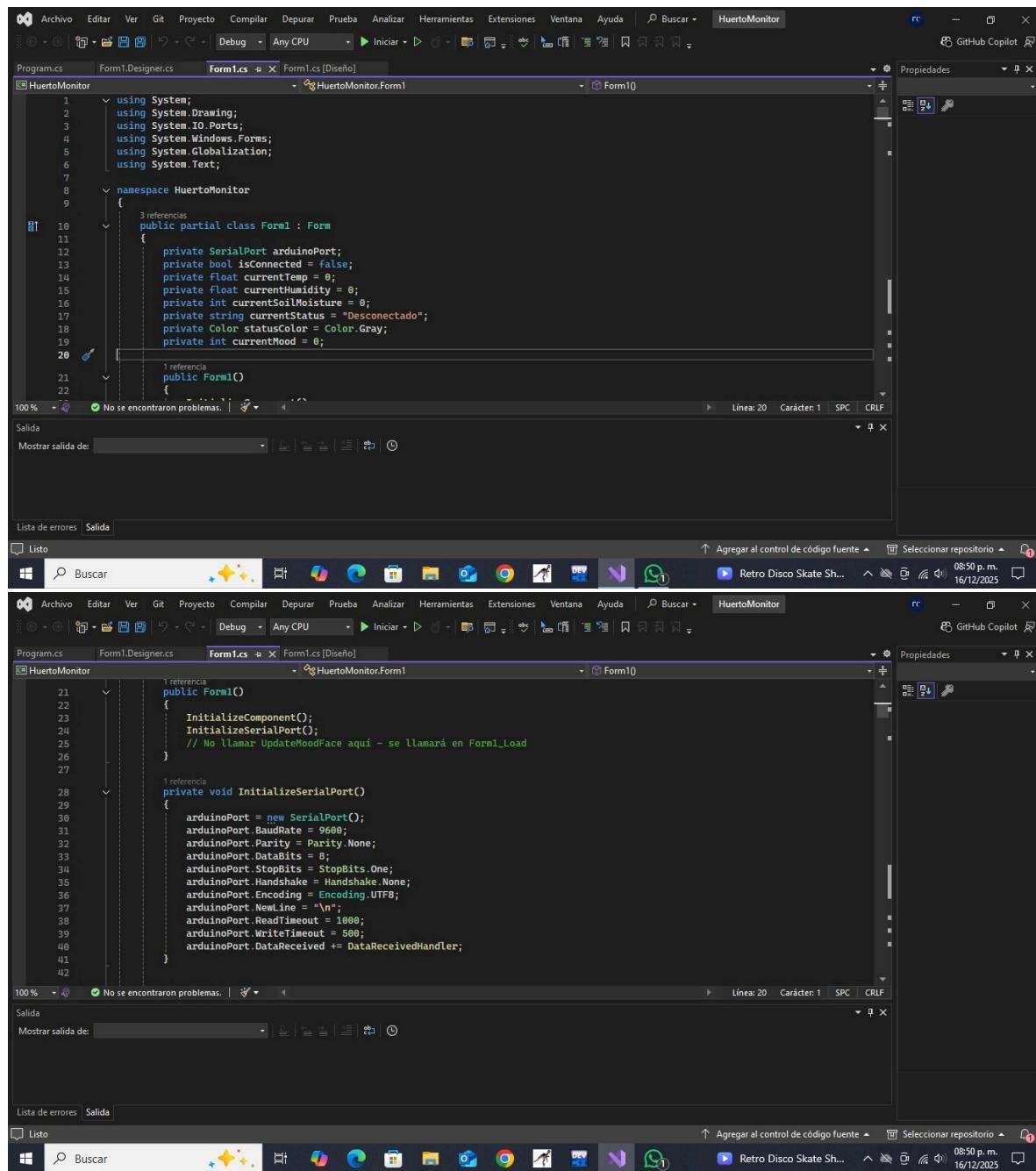


# Proyecto final de arduino: Capturas de las prácticas

## Visual



```
1  using System;
2  using System.Drawing;
3  using System.IO.Ports;
4  using System.Windows.Forms;
5  using System.Globalization;
6  using System.Text;
7
8  namespace HuertoMonitor
9  {
10     public partial class Form1 : Form
11     {
12         private SerialPort arduinoPort;
13         private bool isConnected = false;
14         private float currentTemp = 0;
15         private float currentHumidity = 0;
16         private int currentSoilMoisture = 0;
17         private string currentStatus = "Desconectado";
18         private Color statusColor = Color.Gray;
19         private int currentHood = 0;
20
21         public Form1()
22         {
23             InitializeComponent();
24             InitializeSerialPort();
25             // No llamar UpdateMoodFace aqui - se llamará en Form1_Load
26         }
27
28         private void InitializeSerialPort()
29         {
30             arduinoPort = new SerialPort();
31             arduinoPort.BaudRate = 9600;
32             arduinoPort.Parity = Parity.None;
33             arduinoPort.DataBits = 8;
34             arduinoPort.StopBits = StopBits.One;
35             arduinoPort.Handshake = Handshake.None;
36             arduinoPort.Encoding = Encoding.UTF8;
37             arduinoPort.NewLine = "\n";
38             arduinoPort.ReadTimeout = 1000;
39             arduinoPort.WriteTimeout = 500;
40             arduinoPort.DataReceived += DataReceivedHandler;
41         }
42     }
43 }
```

Este cuadro de diálogo muestra el código C# para el evento `Form1_Load`. El código se encarga de cargar los puertos COM disponibles y actualizar los controles correspondientes.

```
41 }  
42  
43 // Referencia  
44 private void Form1_Load(object sender, EventArgs e)  
45 {  
46     // Cargar puertos COM disponibles  
47     CargarPuertosCOM();  
48  
49     // Inicializar controles  
50     UpdateMoodFace();  
51     UpdateStatus();  
52  
53     // Referencia  
54     private void CargarPuertosCOM()  
55     {  
56         try  
57         {  
58             cmbPorts.Items.Clear();  
59             string[] ports = SerialPort.GetPortNames();  
60             cmbPorts.Items.AddRange(ports);  
61  
62             if (ports.Length > 0)  
63             {  
64                 cmbPorts.SelectedIndex = 0;  
65             }  
66             else  
67             {  
68                 cmbPorts.Text = "No hay puertos disponibles";  
69             }  
70         catch (Exception ex)  
71         {  
72             MessageBox.Show("Error al cargar puertos COM: " + ex.Message,  
73                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);  
74         }  
75     }  
76  
77     // Referencia  
78     private void btnConnect_Click(object sender, EventArgs e)  
79     {  
80         if (! isConnected)  
81         {  
82             ConectarArduino();  
83         }  
84     }  
85 }
```

Este cuadro de diálogo muestra el código C# para el evento `btnConnect_Click`. El código verifica si la conexión ya está establecida y, si no, llama al método `ConectarArduino()`.

```
60 }  
61  
62     if (ports.Length > 0)  
63     {  
64         cmbPorts.SelectedIndex = 0;  
65     }  
66     else  
67     {  
68         cmbPorts.Text = "No hay puertos disponibles";  
69     }  
70     catch (Exception ex)  
71     {  
72         MessageBox.Show("Error al cargar puertos COM: " + ex.Message,  
73                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);  
74     }  
75  
76     // Referencia  
77     private void btnConnect_Click(object sender, EventArgs e)  
78     {  
79         if (!isConnected)  
80         {  
81             ConectarArduino();  
82         }  
83     }  
84  
85 }
```

Imagen que muestra la parte inferior de la interfaz de Visual Studio, incluyendo la barra de status y la barra de tareas.

Este fragmento de código pertenece a la clase `HuertoMonitor` en el archivo `Form1.cs`. La función `ConectarArduino()` verifica si se ha seleccionado un puerto COM válido. Si es así, intenta abrirlo. Si falla, muestra un mensaje de error y retorna.

```
81     }
82     else
83     {
84         DesconectarArduino();
85     }
86 }
87
88 // referencia
89 private void ConectarArduino()
90 {
91     try
92     {
93         if (string.IsNullOrEmpty(cmbPorts.Text) ||
94             cmbPorts.Text.Contains("No hay puertos"))
95         {
96             MessageBox.Show("Selecciona un puerto COM válido",
97                             "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
98             return;
99         }
100
101         arduinoPort.PortName = cmbPorts.Text;
102         arduinoPort.Open();
103         isConnected = true;
104     }
105     catch (Exception ex)
106     {
107         btnConnect.Text = "Desconectar";
108         cmbPorts.Enabled = false;
109         currentStatus = "Conectado";
110         statusColor = Color.Blue;
111         UpdateStatus();
112     }
113 }
114
115 private void DesconectarArduino()
116 {
117     try
118     {
119         if (arduinoPort.IsOpen)
```

Este fragmento de código pertenece a la clase `HuertoMonitor` en el archivo `Form1.cs`. La función `DesconectarArduino()` intenta cerrar el puerto COM actual. Si falla, muestra un mensaje de error.

```
101     arduinoPort.PortName = cmbPorts.Text;
102     arduinoPort.Open();
103     isConnected = true;
104
105     // Actualizar interfaz
106     btnConnect.Text = "Desconectar";
107     cmbPorts.Enabled = false;
108     currentStatus = "Conectado";
109     statusColor = Color.Blue;
110     UpdateStatus();
111 }
112
113 catch (Exception ex)
114 {
115     MessageBox.Show("Error al conectar: " + ex.Message,
116                     "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
117 }
118
119 private void DesconectarArduino()
120 {
121     try
122     {
123         if (arduinoPort.IsOpen)
```

Este fragmento de código pertenece a la clase `HuertoMonitor` en el archivo `Form1.cs`. La función `DesconectarArduino()` intenta cerrar el puerto COM actual. Si falla, muestra un mensaje de error.

```
101     arduinoPort.PortName = cmbPorts.Text;
102     arduinoPort.Open();
103     isConnected = true;
104
105     // Actualizar interfaz
106     btnConnect.Text = "Desconectar";
107     cmbPorts.Enabled = false;
108     currentStatus = "Conectado";
109     statusColor = Color.Blue;
110     UpdateStatus();
111 }
112
113 catch (Exception ex)
114 {
115     MessageBox.Show("Error al conectar: " + ex.Message,
116                     "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
117 }
118
119 private void DesconectarArduino()
120 {
121     try
122     {
123         if (arduinoPort.IsOpen)
```

The screenshot shows the Visual Studio IDE interface with the code editor open. The file being edited is `Form1.cs`. The code implements logic to disconnect from an Arduino port. It first checks if the port is open, then sends commands to turn off LEDs ('R', 'Y', 'G') and sleeps for 100ms. Finally, it closes the port and updates the UI state.

```
121     try
122     {
123         if (arduinoPort.IsOpen)
124         {
125             // Apagar LEDs antes de desconectar
126             SendLEDCommand('R', 0);
127             SendLEDCommand('Y', 0);
128             SendLEDCommand('G', 0);
129             System.Threading.Thread.Sleep(100);
130
131             arduinoPort.Close();
132         }
133
134         isConnected = false;
135         btnConnect.Text = "Conectar";
136         cmbPorts.Enabled = true;
137         currentStatus = "Desconectado";
138         statusColor = Color.Gray;
139         UpdateStatus();
140     }
141     catch (Exception ex)
142     {
143         MessageBox.Show("Error al desconectar: " + ex.Message,
144                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
145     }
146 }
147
148 //referencia
149 private void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
150 {
151     try
152     {
153         string data = arduinoPort.ReadLine();
154         if (!string.IsNullOrWhiteSpace(data))
155         {
156             // Invocar en el hilo principal para actualizar UI
157         }
158     }
159 }
```

This screenshot shows the same Visual Studio session with the code editor still displaying the disconnect logic. Below it, the beginning of the `DataReceivedHandler` method is visible, which reads data from the serial port and attempts to invoke a UI update on the main thread.

```
133
134         isConnected = false;
135         btnConnect.Text = "Conectar";
136         cmbPorts.Enabled = true;
137         currentStatus = "Desconectado";
138         statusColor = Color.Gray;
139         UpdateStatus();
140     }
141     catch (Exception ex)
142     {
143         MessageBox.Show("Error al desconectar: " + ex.Message,
144                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
145     }
146 }
147
148 //referencia
149 private void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
150 {
151     try
152     {
153         string data = arduinoPort.ReadLine();
154         if (!string.IsNullOrWhiteSpace(data))
155         {
156             // Invocar en el hilo principal para actualizar UI
157         }
158     }
159 }
```

This final screenshot shows the completed implementation of the `DataReceivedHandler` method. It includes a check for null or whitespace data and a call to `UpdateUI()` to refresh the user interface.

```
133
134         isConnected = false;
135         btnConnect.Text = "Conectar";
136         cmbPorts.Enabled = true;
137         currentStatus = "Desconectado";
138         statusColor = Color.Gray;
139         UpdateStatus();
140     }
141     catch (Exception ex)
142     {
143         MessageBox.Show("Error al desconectar: " + ex.Message,
144                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
145     }
146 }
147
148 //referencia
149 private void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
150 {
151     try
152     {
153         string data = arduinoPort.ReadLine();
154         if (!string.IsNullOrWhiteSpace(data))
155         {
156             UpdateUI();
157         }
158     }
159 }
```

Screenshot of Visual Studio showing the code editor for Form1.cs. The code handles data processing from a serial port, ignoring null or whitespace data and catching exceptions. It then parses the data into arrays and updates UI elements.

```
153     if (!string.IsNullOrEmpty(data))
154     {
155         // Invocar en el hilo principal para actualizar UI
156         this.Invoke(new Action<string>(ProcessData), data.Trim());
157     }
158     catch (Exception)
159     {
160         // Ignorar errores de lectura cuando se cierra el puerto
161     }
162 }
163
164     private void ProcessData(string data)
165     {
166         try
167         {
168             if (data.StartsWith("DATA"))
169             {
170                 string[] values = data.Split(',');
171                 if (values.Length >= 4)
172                 {
173                     // Parsear datos con manejo de errores
174                     if (float.TryParse(values[1], NumberStyles.Float,
```

Screenshot of Visual Studio showing the code editor for Form1.cs. The code continues from the previous snippet, handling the parsed data to update sensor readings, evaluate conditions, and update mood faces. It also includes a catch block for exceptions.

```
175                     if (float.TryParse(values[1], NumberStyles.Float,
176                         CultureInfo.InvariantCulture, out float temp) &&
177                             float.TryParse(values[2], NumberStyles.Float,
178                             CultureInfo.InvariantCulture, out float humidity) &&
179                             int.TryParse(values[3], out int soil))
180                 {
181                     currentTemp = temp;
182                     currentHumidity = humidity;
183                     currentSoilMoisture = soil;
184
185                     // Actualizar interfaz
186                     UpdateSensorReadings();
187                     EvaluateConditions();
188                     UpdateMoodFace();
189                 }
190             }
191         }
192         catch (Exception ex)
193         {
194             Console.WriteLine("Error procesando datos: " + ex.Message);
195         }
196     }
197 }
```

Screenshot of Visual Studio showing the code editor for Form1.cs. The code concludes with a closing brace for the main method.

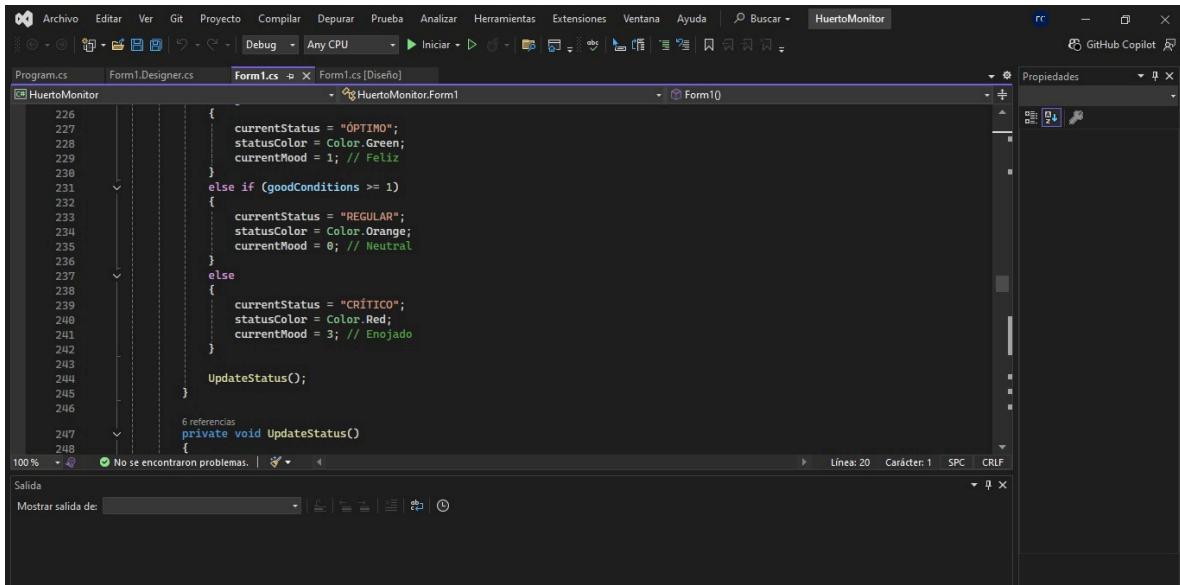
```
198 }
```

Code snippet for updating sensor readings:

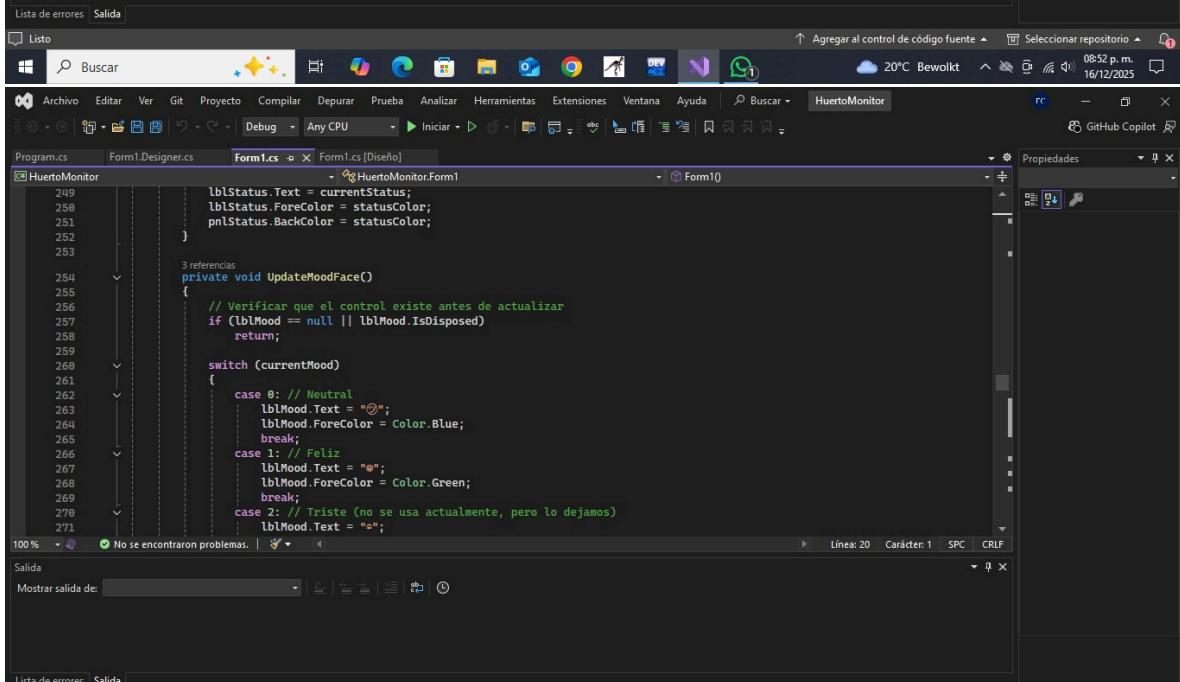
```
194     }  
195     }  
196     }  
197     }  
198     //referencia  
199     private void UpdateSensorReadings()  
200     {  
201         // Actualizar etiquetas de valores  
202         lblTempValue.Text = currentTemp.ToString("0.0") + "°C";  
203         lblHumidityValue.Text = currentHumidity.ToString("0.0") + "%";  
204         lblSoilValue.Text = currentSoilMoisture.ToString();  
205         // Actualizar barras de progreso (ajustando escalas)  
206         progressBarTemp.Value = (int)Math.Min(Math.Max(currentTemp, 0), 100);  
207         progressBarHumidity.Value = (int)Math.Min(Math.Max(currentHumidity, 0), 100);  
208         // Convertir humedad del suelo (0-1024) a porcentaje (0-100)  
209         int soilPercent = (int)Math.Min(Math.Max((currentSoilMoisture * 100) / 1024, 0), 100);  
210         progressBarSoil.Value = soilPercent;  
211     }  
212     //referencia  
213     private void EvaluateConditions()  
214     {  
215     }  
216 }
```

Code snippet for evaluating conditions:

```
215     }  
216     }  
217     }  
218     //referencia  
219     private void EvaluateConditions()  
220     {  
221         int goodConditions = 0;  
222         // Verificar condiciones óptimas  
223         if (currentTemp >= 15 && currentTemp <= 30) goodConditions++;  
224         if (currentHumidity >= 40 && currentHumidity <= 70) goodConditions++;  
225         if (currentSoilMoisture >= 400 && currentSoilMoisture <= 800) goodConditions++;  
226         // Determinar estado general  
227         if (goodConditions == 3)  
228         {  
229             currentStatus = "ÓPTIMO";  
230             statusColor = Color.Green;  
231             currentMood = 1; // Feliz  
232         }  
233         else if (goodConditions >= 1)  
234         {  
235             currentStatus = "REGULAR";  
236             statusColor = Color.Orange;  
237             currentMood = 0; // Neutral  
238         }  
239     }  
240 }
```



```
226     {
227         currentStatus = "ÓPTIMO";
228         statusColor = Color.Green;
229         currentMood = 1; // Feliz
230     }
231     else if (goodConditions >= 1)
232     {
233         currentStatus = "REGULAR";
234         statusColor = Color.Orange;
235         currentMood = 0; // Neutral
236     }
237     else
238     {
239         currentStatus = "CRÍTICO";
240         statusColor = Color.Red;
241         currentMood = 3; // Enojado
242     }
243
244     UpdateStatus();
245 }
246
247 6 referencias
248 private void UpdateStatus()
```



```
249     lblStatus.Text = currentStatus;
250     lblStatus.ForeColor = statusColor;
251     pnlStatus.BackColor = statusColor;
252 }
253
254 3 referencias
255 private void UpdateMoodFace()
256 {
257     // Verificar que el control existe antes de actualizar
258     if (lblMood == null || lblMood.IsDisposed)
259     {
260         return;
261     }
262     switch (currentMood)
263     {
264         case 0: // Neutral
265             lblMood.Text = "☺";
266             lblMood.ForeColor = Color.Blue;
267             break;
268         case 1: // Feliz
269             lblMood.Text = "☻";
270             lblMood.ForeColor = Color.Green;
271             break;
272         case 2: // Triste (no se usa actualmente, pero lo dejamos)
273             lblMood.Text = "☹";
274     }
275 }
```



```
100 % 6 referencias
No se encontraron problemas. | ↻ Salida
Mostrar salida de: | ↻ Salida
```

```
    LblMood.ForeColor = Color.Blue;
    break;
  case 3: // Enojado
    LblMood.Text = "憤";
    LblMood.ForeColor = Color.Red;
    break;
  default: // Por defecto neutral
    LblMood.Text = "utral";
    LblMood.ForeColor = Color.Blue;
    break;
  }

  #region Control de LEDs

  1 referencia
  private void btnRedOn_Click(object sender, EventArgs e)
  {
    SendLEDCommand('R', trackBarRed.Value);
  }

  1 referencia
  private void btnRedOff_Click(object sender, EventArgs e)
  {
    SendLEDCommand('R', 0);
  }
}
```

```
1 referencia
290     }
291 }
292 
293     private void btnRedOff_Click(object sender, EventArgs e)
294     {
295         SendLEDCommand('R', 0);
296     }
297 
298     private void btnYellowOn_Click(object sender, EventArgs e)
299     {
300         SendLEDCommand('Y', trackBarYellow.Value);
301     }
302 
303     private void btnYellowOff_Click(object sender, EventArgs e)
304     {
305         SendLEDCommand('Y', 0);
306     }
307 
308     private void btnGreenOn_Click(object sender, EventArgs e)
309     {
310         SendLEDCommand('G', trackBarGreen.Value);
311     }
```

```
private void btnGreen_Click(object sender, EventArgs e)
{
    SendLEDCommand('G', trackBarGreen.Value);
}

private void btnGreenOff_Click(object sender, EventArgs e)
{
    SendLEDCommand('G', 0);
}

private void trackBarRed_Scroll(object sender, EventArgs e)
{
    lblRedValue.Text = trackBarRed.Value.ToString();
    if (isConnected && chkAutoUpdate.Checked)
    {
        SendLEDCommand('R', trackBarRed.Value);
    }
}

private void trackBarYellow_Scroll(object sender, EventArgs e)
{
    lblYellowValue.Text = trackBarYellow.Value.ToString();
}
```

```
private void trackBarYellow_Scroll(object sender, EventArgs e)
{
    lblYellowValue.Text = trackBarYellow.Value.ToString();
    if (isConnected && chkAutoUpdate.Checked)
    {
        SendLEDCommand('Y', trackBarYellow.Value);
    }
}

private void trackBarGreen_Scroll(object sender, EventArgs e)
{
    lblGreenValue.Text = trackBarGreen.Value.ToString();
    if (isConnected && chkAutoUpdate.Checked)
    {
        SendLEDCommand('G', trackBarGreen.Value);
    }
}

12 referencias
private void SendLEDCommand(char color, int value)
{
    if (isConnected && arduinoPort != null && arduinoPort.IsOpen)
```

Code snippet for sending LED commands to an Arduino:

```
346 if (isConnected && arduinoPort != null && arduinoPort.IsOpen)
347 {
348     try
349     {
350         // Limitar valor entre 0-255
351         value = Math.Max(0, Math.Min(value, 255));
352
353         // Enviar comando al Arduino
354         string command = $"LED,{color},{value}";
355         arduinoPort.WriteLine(command);
356     }
357     catch (Exception ex)
358     {
359         MessageBox.Show("Error al enviar comando LED: " + ex.Message,
360                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
361     }
362 }
363
364 #endregion
365
366 #region Modos de operación
367
368
```

Code snippets for mode switching:

```
369 private void btnManualMode_Click(object sender, EventArgs e)
370 {
371     if (isConnected)
372     {
373         try
374         {
375             arduinoPort.WriteLine("MODE,1");
376             currentStatus = "MODO MANUAL";
377             statusColor = Color.Purple;
378             UpdateStatus();
379             currentMood = 0;
380             UpdateMoodFace();
381         }
382         catch (Exception ex)
383         {
384             MessageBox.Show("Error al cambiar a modo manual: " + ex.Message,
385                             "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
386         }
387     }
388 }
389
390 private void btnAutoMode_Click(object sender, EventArgs e)
391 {
392 }
```

Windows taskbar showing various pinned icons and system status.

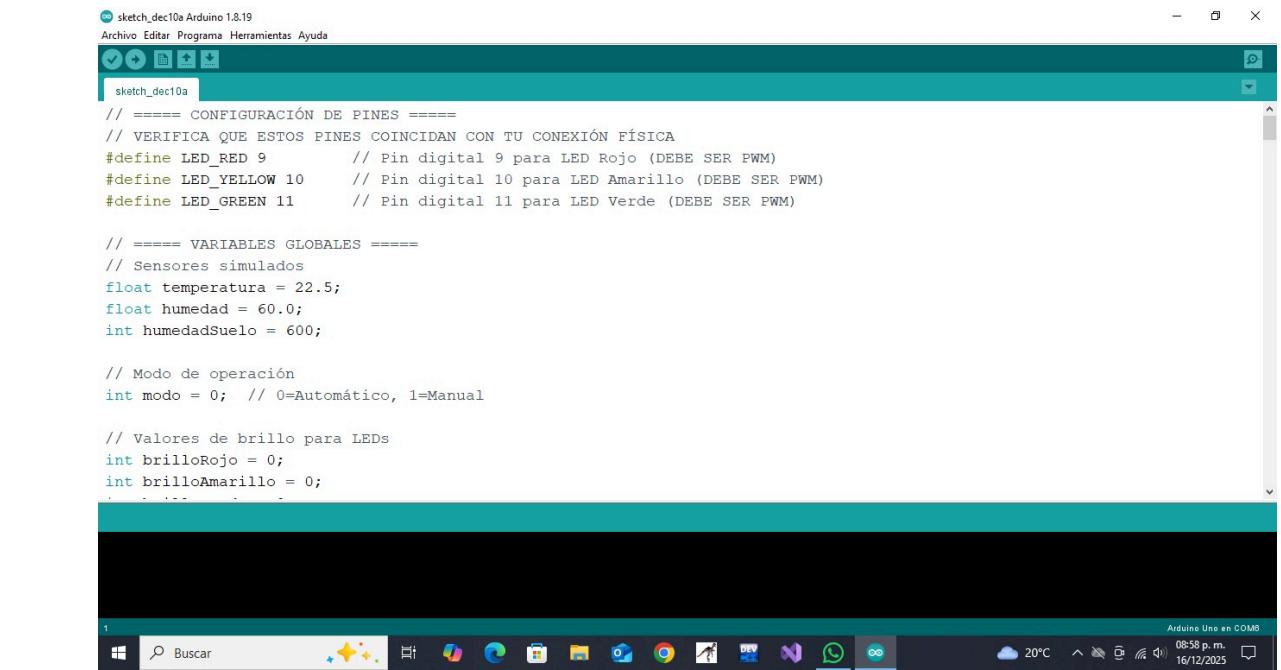
Code for `btnAutoMode_Click`:

```
390     if (isConnected)
391     {
392         try
393         {
394             arduinoPort.WriteLine("MODE,0");
395             currentStatus = "MODO AUTOMÁTICO";
396             statusColor = Color.Blue;
397             UpdateStatus();
398         }
399         catch (Exception ex)
400         {
401             MessageBox.Show("Error al cambiar a modo automático: " + ex.Message,
402                             "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
403         }
404     }
405 
```

Code for `btnRefreshPorts_Click` and `FormClosing`:

```
413     CargarPuertosCOM();
414 
415     // Asegurarse de cerrar el puerto serial al salir
416     if (isConnected && arduinoPort != null && arduinoPort.IsOpen)
417     {
418         try
419         {
420             DesconectarArduino();
421         }
422         catch
423         {
424             // Ignorar errores al cerrar
425         }
426     }
427 
```

# Arduino



```
// ===== CONFIGURACIÓN DE PINES =====
// VERIFICA QUE ESTOS PINES COINCIDAN CON TU CONEXIÓN FÍSICA
#define LED_RED 9           // Pin digital 9 para LED Rojo (DEBE SER PWM)
#define LED_YELLOW 10        // Pin digital 10 para LED Amarillo (DEBE SER PWM)
#define LED_GREEN 11          // Pin digital 11 para LED Verde (DEBE SER PWM)

// ===== VARIABLES GLOBALES =====
// Sensores simulados
float temperatura = 22.5;
float humedad = 60.0;
int humedadSuelo = 600;

// Modo de operación
int modo = 0; // 0=Automático, 1=Manual

// Valores de brillo para LEDs
int brilloRojo = 0;
int brilloAmarillo = 0;
int brilloVerde = 0;

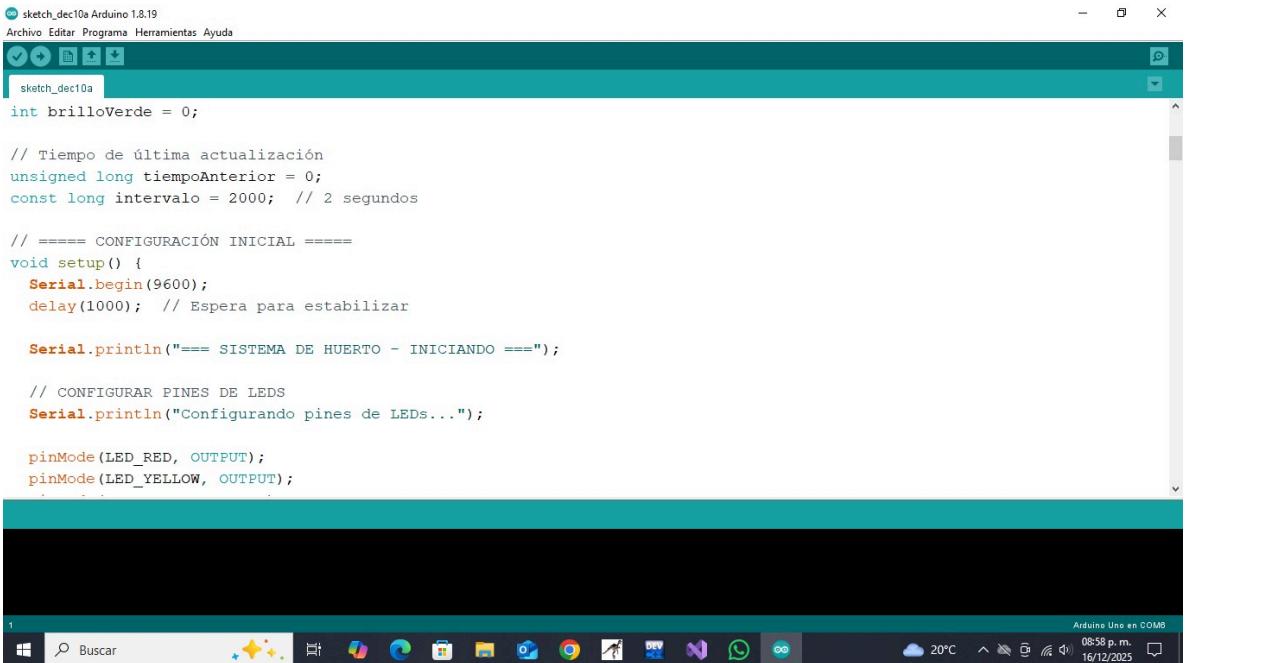
// Tiempo de última actualización
unsigned long tiempoAnterior = 0;
const long intervalo = 2000; // 2 segundos

// ===== CONFIGURACIÓN INICIAL =====
void setup() {
    Serial.begin(9600);
    delay(1000); // Espera para estabilizar

    Serial.println("==== SISTEMA DE HUERTO - INICIANDO ====");

    // CONFIGURAR PINES DE LEDS
    Serial.println("Configurando pines de LEDs...");

    pinMode(LED_RED, OUTPUT);
    pinMode(LED_YELLOW, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
}
```



```
Arduino Uno en COM8
08:58 p.m. 16/12/2025
```

```
1
Arduino Uno en COM8
08:58 p.m. 16/12/2025
```

```
1
Arduino Uno en COM8
08:58 p.m. 16/12/2025
```

sketch\_dec10a Arduino 1.8.19

Archivo Editar Programa Herramientas Ayuda

```

pinMode(LED_GREEN, OUTPUT);

// APAGAR TODOS LOS LEDS (seguro)
digitalWrite(LED_RED, LOW);
digitalWrite(LED_YELLOW, LOW);
digitalWrite(LED_GREEN, LOW);

// PRUEBA INICIAL DE LEDS
pruebaLEDsInicial();

Serial.println("Sistema listo. Envioando datos cada 2 segundos...");
Serial.println("Formato: DATA,temperatura,humedad,humedadSuelo");
Serial.println("Comandos: LED,R,valor / LED,Y,valor / LED,G,valor");
Serial.println("=====");
}

// ===== PRUEBA INICIAL DE LEDS =====
void pruebaLEDsInicial() {
    Serial.println("Realizando prueba de LEDs...");

    // Prueba LED ROJO
    Serial.println("1. Probando LED ROJO (pin 9)...");
    analogWrite(LED_RED, 100); // Brillo medio
    delay(1000);
}

```

Arduino Uno en COM8

08:58 p. m.  
16/12/2025

sketch\_dec10a Arduino 1.8.19

Archivo Editar Programa Herramientas Ayuda

```

pinMode(LED_GREEN, OUTPUT);

// APAGAR TODOS LOS LEDS (seguro)
digitalWrite(LED_RED, LOW);
digitalWrite(LED_YELLOW, LOW);
digitalWrite(LED_GREEN, LOW);

// PRUEBA INICIAL DE LEDS
pruebaLEDsInicial();

Serial.println("Sistema listo. Envioando datos cada 2 segundos...");
Serial.println("Formato: DATA,temperatura,humedad,humedadSuelo");
Serial.println("Comandos: LED,R,valor / LED,Y,valor / LED,G,valor");
Serial.println("=====");
}

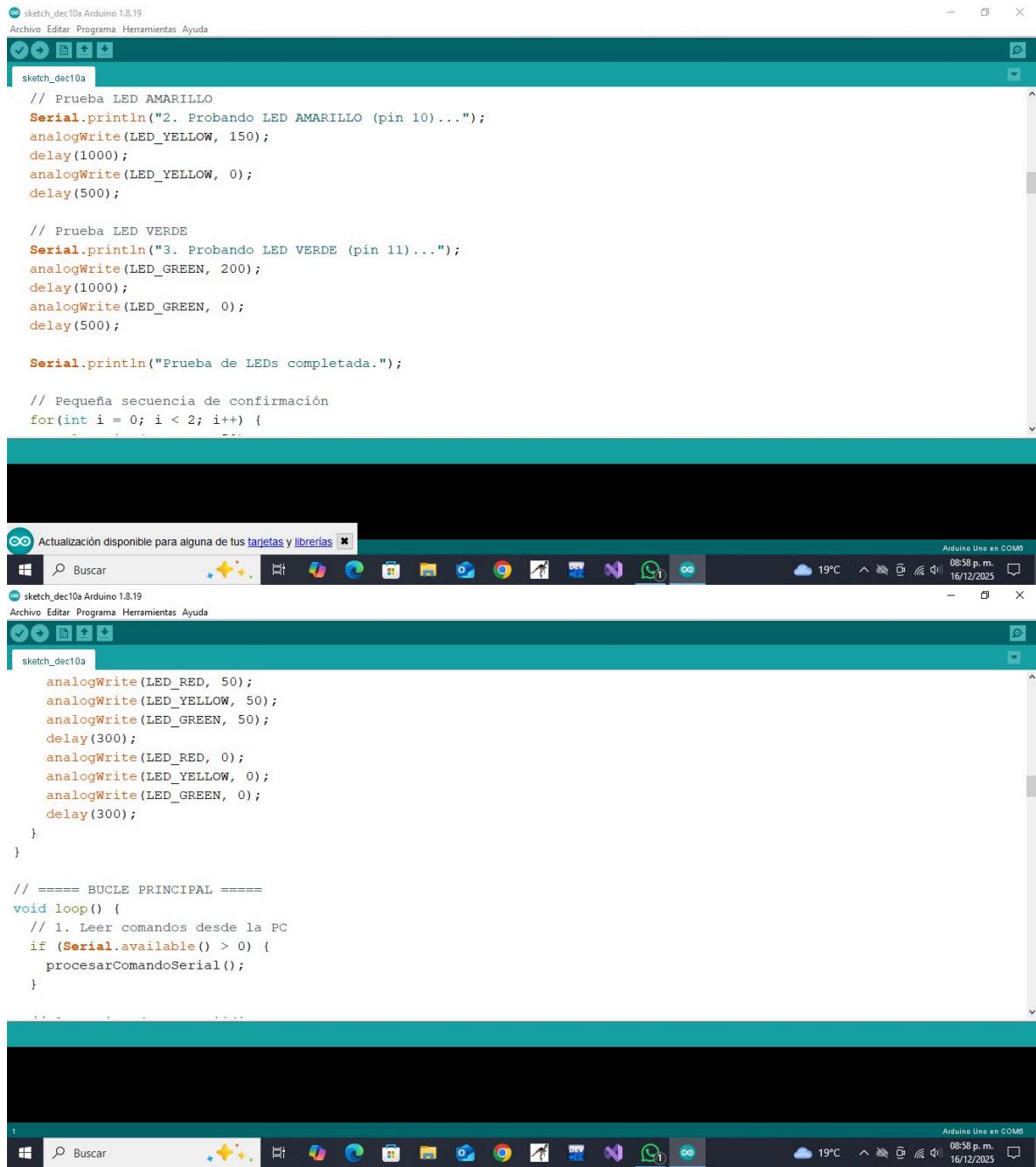
// ===== PRUEBA INICIAL DE LEDS =====
void pruebaLEDsInicial() {
    Serial.println("Realizando prueba de LEDs...");

    // Prueba LED ROJO
    Serial.println("1. Probando LED ROJO (pin 9)...");
    analogWrite(LED_RED, 100); // Brillo medio
    delay(1000);
}

```

Arduino Uno en COM8

08:58 p. m.  
16/12/2025



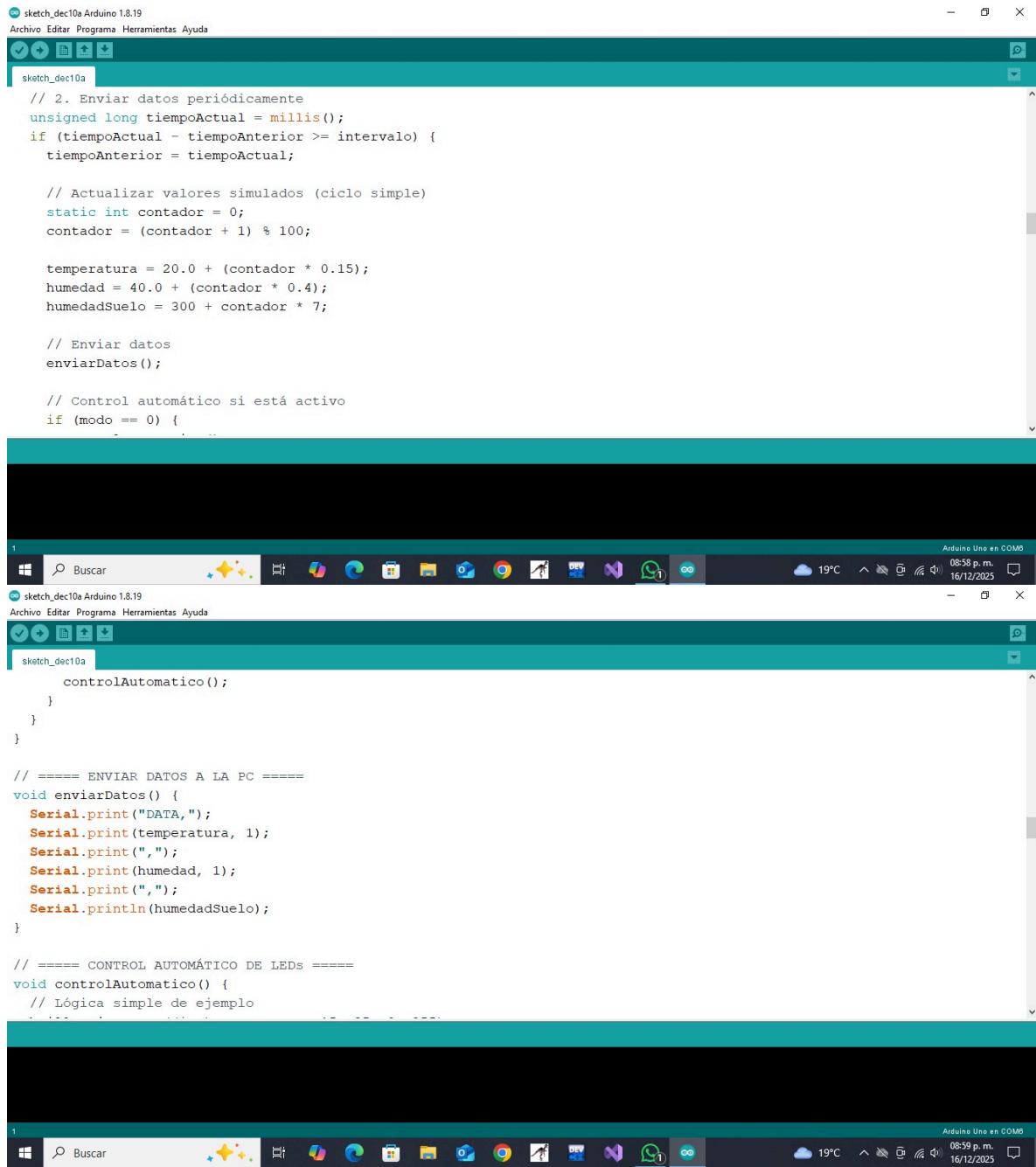
```
// Prueba LED AMARILLO
Serial.println("2. Probando LED AMARILLO (pin 10)...");
analogWrite(LED_YELLOW, 150);
delay(1000);
analogWrite(LED_YELLOW, 0);
delay(500);

// Prueba LED VERDE
Serial.println("3. Probando LED VERDE (pin 11)...");
analogWrite(LED_GREEN, 200);
delay(1000);
analogWrite(LED_GREEN, 0);
delay(500);

Serial.println("Prueba de LEDs completada.");

// Pequeña secuencia de confirmación
for(int i = 0; i < 2; i++) {
    analogWrite(LED_RED, 50);
    analogWrite(LED_YELLOW, 50);
    analogWrite(LED_GREEN, 50);
    delay(300);
    analogWrite(LED_RED, 0);
    analogWrite(LED_YELLOW, 0);
    analogWrite(LED_GREEN, 0);
    delay(300);
}

// ===== BUCLE PRINCIPAL =====
void loop() {
    // 1. Leer comandos desde la PC
    if (Serial.available() > 0) {
        procesarComandoSerial();
    }
}
```



```
// 2. Enviar datos periódicamente
unsigned long tiempoActual = millis();
if (tiempoActual - tiempoAnterior >= intervalo) {
    tiempoAnterior = tiempoActual;

    // Actualizar valores simulados (ciclo simple)
    static int contador = 0;
    contador = (contador + 1) % 100;

    temperatura = 20.0 + (contador * 0.15);
    humedad = 40.0 + (contador * 0.4);
    humedadSuelo = 300 + contador * 7;

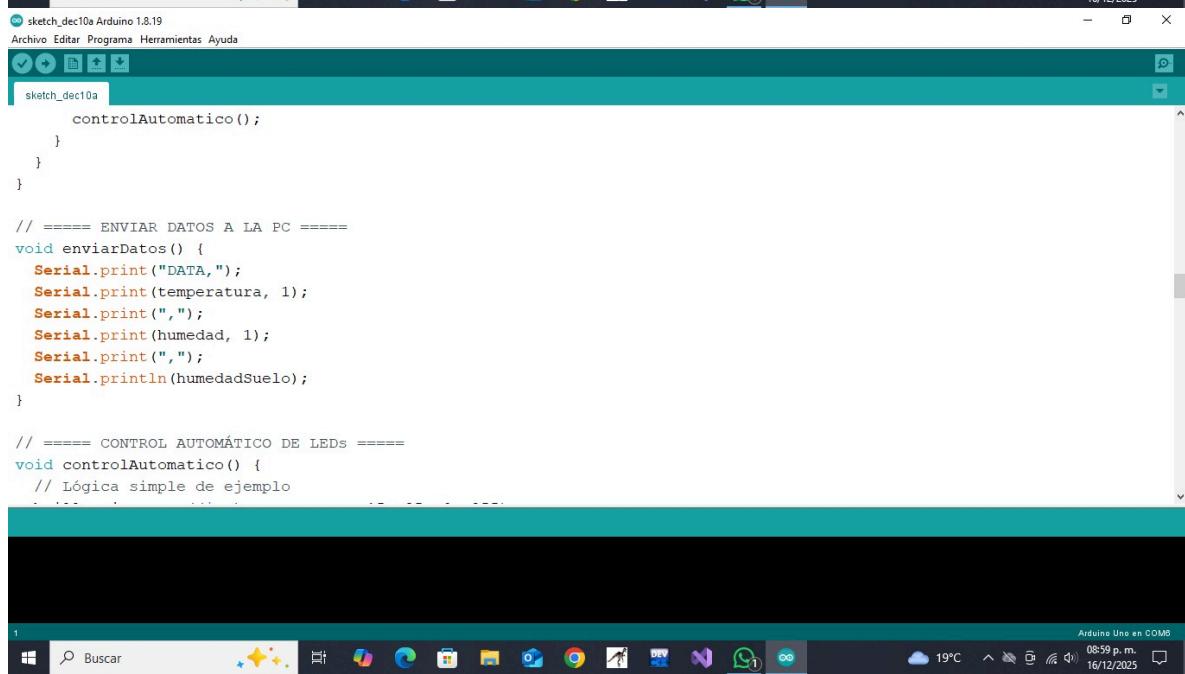
    // Enviar datos
    enviarDatos();

    // Control automático si está activo
    if (modo == 0) {
        ...
    }
}

// ===== ENVIAR DATOS A LA PC =====
void enviarDatos() {
    Serial.print("DATA,");
    Serial.print(temperatura, 1);
    Serial.print(",");
    Serial.print(humedad, 1);
    Serial.print(",");
    Serial.println(humedadSuelo);
}

// ===== CONTROL AUTOMÁTICO DE LEDS =====
void controlAutomatico() {
    // Lógica simple de ejemplo
    ...
}
```

Arduino Uno en COM8  
08:58 p. m.  
16/12/2025



```
controlAutomatico();
```

Arduino Uno en COM8  
08:59 p. m.  
16/12/2025

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
controlAutomatico();
}
}

// ===== ENVIAR DATOS A LA PC =====
void enviarDatos() {
    Serial.print("DATA,");
    Serial.print(temperatura, 1);
    Serial.print(",");
    Serial.print(humedad, 1);
    Serial.print(",");
    Serial.println(humedadSuelo);
}

// ===== CONTROL AUTOMÁTICO DE LEDS =====
void controlAutomatico() {
    // Lógica simple de ejemplo
}

// ===== CONTROL AUTOMÁTICO DE LEDS =====
void controlAutomatico() {
    // Lógica simple de ejemplo
    brilloRojo = map((int)temperatura, 15, 35, 0, 255);
    brilloRojo = constrain(brilloRojo, 0, 255);

    brilloAmarillo = map((int)humedad, 30, 80, 0, 255);
    brilloAmarillo = constrain(brilloAmarillo, 0, 255);

    brilloVerde = map(humedadSuelo, 200, 900, 0, 255);
    brilloVerde = constrain(brilloVerde, 0, 255);

    // Aplicar brillos
    analogWrite(LED_RED, brilloRojo);
    analogWrite(LED_YELLOW, brilloAmarillo);
    analogWrite(LED_GREEN, brilloVerde);

    // Para debug en monitor serial
}
```

```
Arduino Uno en COM8
09:02 p.m.
16/12/2025
```

```
Arduino Uno en COM8
09:03 p.m.
16/12/2025
```

The screenshot shows three stacked windows of the Arduino IDE. The top window displays the following code:

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
Serial.print("AUTO - LEDs R:");
Serial.print(brilloRojo);
Serial.print(" Y:");
Serial.print(brilloAmarillo);
Serial.print(" G:");
Serial.println(brilloVerde);
}

// ===== PROCESAR COMANDOS SERIALES =====
void procesarComandoSerial() {
String comando = Serial.readStringUntil('\n');
comando.trim();

if (comando.length() == 0) return;

Serial.print("Comando recibido: ");
Serial.println(comando);
```

The middle window shows the same code, but with additional logic for processing serial commands:

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
Serial.print("Comando recibido: ");
Serial.println(comando);

// Comando LED (ej: LED,R,100)
if (comando.startsWith("LED,")) {
    char color = comando.charAt(4); // Obtener el color (R, Y o G)
    int valor = comando.substring(6).toInt();
    valor = constrain(valor, 0, 255);

    switch(color) {
        case 'R':
        case 'r':
            brilloRojo = valor;
            analogWrite(LED_RED, brilloRojo);
            Serial.print("LED Rojo: ");
            Serial.println(valor);
            break;
```

The bottom window shows the same code again, with the serial output visible in the terminal window:

```
Arduino Uno en COM8
09:03 p.m.
16/12/2025
```

The image displays three separate windows of the Arduino IDE, each showing a different sketch. The top window shows a sketch for controlling yellow and green LEDs based on character input. The middle window shows a sketch for handling mode commands (MODE, 0, 1) and setting LED brightnesses. The bottom window shows a command-line interface for testing the sketch.

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
case 'Y':
case 'y':
    brilloAmarillo = valor;
    analogWrite(LED_YELLOW, brilloAmarillo);
    Serial.print("LED Amarillo: ");
    Serial.println(valor);
    break;

case 'G':
case 'g':
    brilloVerde = valor;
    analogWrite(LED_GREEN, brilloVerde);
    Serial.print("LED Verde: ");
    Serial.println(valor);
    break;
}

1
Arduino Uno en COM8
09:03 p. m.
16/12/2025
```

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
// Comando MODO (ej: MODE,0 o MODE,1)
else if (comando.startsWith("MODE,")) {
    int nuevoModo = comando.substring(5).toInt();
    if (nuevoModo == 0 || nuevoModo == 1) {
        modo = nuevoModo;
        Serial.print("Modo cambiado a: ");
        Serial.println(modo == 0 ? "AUTOMATICO" : "MANUAL");
    }
}

// Comando de prueba
.
.
.

1
Arduino Uno en COM8
09:04 p. m.
16/12/2025
```

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
}

// Comando de prueba
else if (comando == "TEST") {
    Serial.println("Ejecutando prueba de LEDs...");
    pruebaLEDs();
}

// Comando de estado
else if (comando == "STATUS") {
    Serial.println("== ESTADO ACTUAL ==");
    Serial.print("Modo: ");
    Serial.println(modo == 0 ? "AUTOMATICO" : "MANUAL");
    Serial.print("Temp: ");
    Serial.print(temperatura, 1);
    Serial.println("°C");
    Serial.print("Hum: ");
    Serial.print(humedad, 1);
    Serial.println("%");
}

Serial.println("Suelo: ");
Serial.println(humedadSuelo);
Serial.print("LEDs - R:");
Serial.print(brilloRojo);
Serial.print(" Y:");
Serial.print(brilloAmarillo);
Serial.print(" G:");
Serial.println(brilloVerde);
Serial.println("=====");

// ===== PRUEBA DE LEDs (comando TEST) =====
void pruebaLEDs() {
    // Apagar todo primero
    analogWrite(LED_RED, 0);
    analogWrite(LED_YELLOW, 0);
}
```

Arduino Uno en COM8  
09:04 p.m.  
16/12/2025

```
sketch_dec10a Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
sketch_dec10a
}

Serial.println("%");
Serial.print("Suelo: ");
Serial.println(humedadSuelo);
Serial.print("LEDs - R:");
Serial.print(brilloRojo);
Serial.print(" Y:");
Serial.print(brilloAmarillo);
Serial.print(" G:");
Serial.println(brilloVerde);
Serial.println("=====");

// ===== PRUEBA DE LEDs (comando TEST) =====
void pruebaLEDs() {
    // Apagar todo primero
    analogWrite(LED_RED, 0);
    analogWrite(LED_YELLOW, 0);
}
```

Arduino Uno en COM8  
09:04 p.m.  
16/12/2025

The image displays three vertically stacked screenshots of the Arduino IDE interface, showing different stages of a sketch named "sketch\_dec10a".

**Top Window:**

```
analogWrite(LED_GREEN, 0);
delay(500);

// Secuencia de prueba
analogWrite(LED_RED, 255);
delay(500);
analogWrite(LED_RED, 0);

analogWrite(LED_YELLOW, 255);
delay(500);
analogWrite(LED_YELLOW, 0);

analogWrite(LED_GREEN, 255);
delay(500);
analogWrite(LED_GREEN, 0);

// Todos juntos con diferentes intensidades
analogWrite(LED_RED, 100);
```

**Middle Window:**

```
analogWrite(LED_YELLOW, 0);

analogWrite(LED_GREEN, 255);
delay(500);
analogWrite(LED_GREEN, 0);

// Todos juntos con diferentes intensidades
analogWrite(LED_RED, 100);
analogWrite(LED_YELLOW, 150);
analogWrite(LED_GREEN, 200);
delay(1000);

// Apagar todo
analogWrite(LED_RED, 0);
analogWrite(LED_YELLOW, 0);
analogWrite(LED_GREEN, 0);

Serial.println("Prueba completada.");
}
```

**Bottom Window:**

```
Arduino Uno en COM8
09:04 p.m.
16/12/2025
```