Diseño de Interfaces con Qt — Clase 3

Entrada de datos con QLineEdit

Dr. Rubén Estrada Marmolejo
Universidad de Guadalajara — Curso de Interfaces

Práctica guiada — 9 de septiembre de 2025

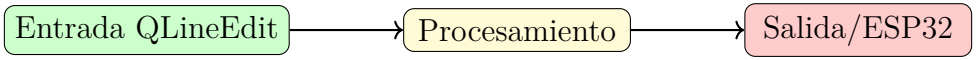
1. Índice de la sesión

Índice

2. Objetivos de la sesión

¿Qué aprenderemos hoy?

1. Dominar el widget **QLineEdit** en Qt Creator
2. Capturar y procesar datos (texto y números)
3. Convertir entre formatos (texto a número)
4. Preparar datos para enviar a hardware ESP32



- **Resultado final**: Crear interfaces que capturen datos del usuario y los preparen para controlar hardware
- **Aplicación práctica**: Los ejercicios de hoy son la base para comunicación serial con microcontroladores

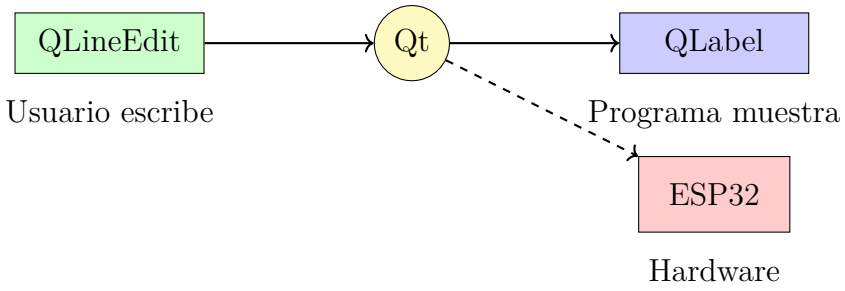
3. Widgets de entrada y salida

Widgets de Entrada

- **QLineEdit**: Texto de una línea
- **QSpinBox**: Números enteros
- **QDoubleSpinBox**: Números decimales
- **QCheckBox**: Opciones sí/no
- **QRadioButton**: Selección única

Widgets de Salida

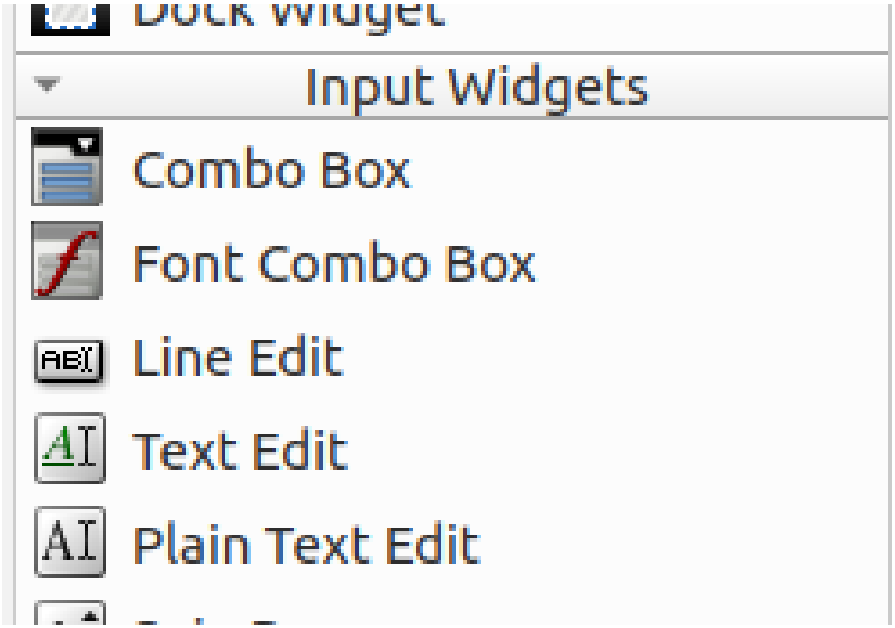
- **QLabel**: Texto estático o dinámico
- **QTextEdit**: Texto multilínea
- **QLCDNumber**: Display numérico
- **QProgressBar**: Barra de progreso



- **Flujo típico**: Entrada → Procesamiento → Salida → Hardware
- **Ejemplo práctico**: Temperatura en QLineEdit → Conversión → Envío serial a ESP32

4. QLineEdit: concepto

- Widget de texto de una sola línea.
- Sirve para que el usuario ingrese datos de tipo texto o números.
- Se puede leer su contenido o modificarlo desde el programa.



Práctica 1: Leer texto de un QLineEdit

Instrucciones

1. Crea un nuevo proyecto en Qt Creator.
2. Agrega en el formulario:
   - Un **QLineEdit** (para escribir texto).
   - Un **QPushButton** (para enviar).
   - Un **QLabel** (para mostrar el resultado).
3. Conecta el botón para que, al presionarlo, el texto del `QLineEdit` aparezca en el `QLabel`.

```
1 void MainWindow::on_pushButton_clicked() {
2     QString texto = ui->lineEdit->text();
3     ui->label->setText("Escribiste: " + texto);
4 }
```

Listing 1: Ejemplo de código en el slot del botón

Pregunta

¿Qué aparece en el QLabel si dejas el QLineEdit vacío y presionas el botón?

5. QLineEdit: concepto y características

¿Qué es QLineEdit?

- Widget de **una sola línea** de texto
- Permite entrada de **texto y números**
- Se puede **leer** (.text()) y **modificar** (.setText())
- Base para interfaces de control de hardware

Aplicación con ESP32

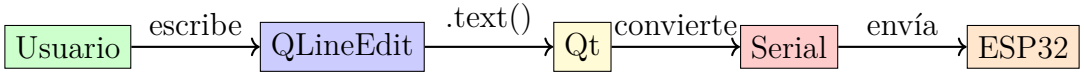
- Capturar valores para enviar a pines GPIO
- Recibir comandos de control
- Parámetros de configuración

QLineEdit

{"pin":13,"state":1}

Ej: JSON para control GPIO

```
QString comando = ui->lineEdit->text();
serial.write(comando.toUtf8());
```



Práctica 2: Modificar texto desde el programa

Instrucciones

1. Usa el mismo proyecto anterior.
2. Agrega un segundo **QPushButton** llamado “Limpiar”.
3. Programa el botón para que:
   - El QLineEdit muestre un mensaje automático al inicio (`.setText()`).
   - Al presionar “Limpiar”, el QLineEdit quede vacío (`.clear()`).

```
1 void MainWindow::on_pushButtonLimpiar_clicked() {
2     ui->lineEdit->clear(); // Deja el campo vaco
3 }
4
5 void MainWindow::on_MainWindow_showEvent(QShowEvent *) {
6     ui->lineEdit->setText("Escribe aqu tu nombre");
7 }
```

Listing 2: Ejemplo de uso de setText() y clear()

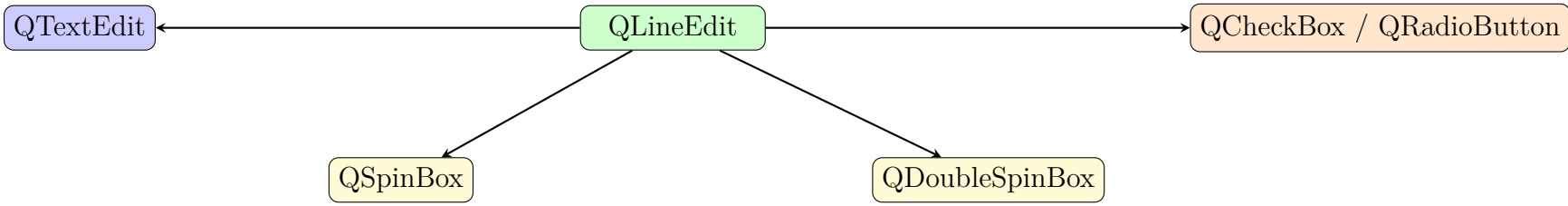
Pregunta

¿Cuál es la diferencia entre usar `.clear()` y `.setText()`?

6. Comparación con otros widgets

¿Dónde usar QLineEdit y dónde no?

Widget	Uso típico y diferencia
QLineEdit	Entrada de texto corto o números simples (ej. nombre, valor de pin, comando rápido).
QTextEdit	Entrada o despliegue de texto largo y multilínea (ej. logs, mensajes recibidos de la ESP32).
QSpinBox	Entrada de números enteros en un rango fijo (ej. 0–255 para valores PWM).
QDoubleSpinBox	Entrada de valores decimales con control de precisión (ej. 3.14 para un setpoint).
QCheckBox	Opciones binarias (sí/no, encender/apagar). Útil cuando no hay necesidad de escribir texto.
QRadioButton	Selección exclusiva entre varias opciones (ej. elegir modo de trabajo: Manual/Automático).



- **QLineEdit** es versátil, pero no siempre el más cómodo.
- Elegir el widget adecuado mejora la experiencia del usuario.
- Ejemplo: mejor usar `QSpinBox` que un `QLineEdit` para valores numéricos limitados.

7. Métodos principales de QLineEdit

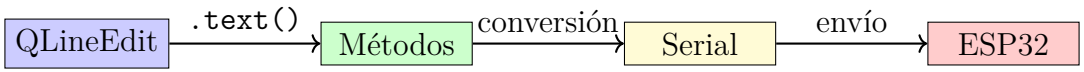
Métodos Básicos

- `.text()` → Obtiene texto como QString
- `.setText()` → Establece texto programáticamente
- `.clear()` → Limpia el contenido
- `.placeholderText()` → Texto guía

Métodos Avanzados

- `.setValidator()` → Restringe entrada
- `.setMaxLength()` → Límite de caracteres
- `.returnPressed()` → Señal al presionar Enter

```cpp
// Ejemplo: Control de LED con ESP32
void MainWindow::onButtonClick() {
    // Leer comando del QLineEdit
    QString comando = ui->lineEditComando->text();

    // Validar formato (ej: "PIN13=HIGH")
    if(comando.contains("=")) {
        // Enviar por serial al ESP32
        serial->write(comando.toUtf8());
        ui->labelStatus->setText("Enviado: " + comando);
    } else {
        ui->labelStatus->setText("Error: Formato invalido")
    ;
    }

    // Limpiar para siguiente entrada
    ui->lineEditComando->clear();
}
```



- **Importante**: Siempre validar entrada antes de enviar a hardware
- **Práctica recomendada**: Usar `.setValidator()` para entradas numéricas

Práctica 3: Capturar y mostrar datos

Instrucciones

1. Crea un proyecto nuevo con un **QLineEdit** y un **QLabel**.
2. Agrega un botón "Mostrar" (**QPushButton**).
3. Programa el botón para que al hacer clic:
   - Lea el texto del QLineEdit con `.text()`.
   - Lo copie en el QLabel con `.setText()`.
4. Prueba también la señal `returnPressed()` del QLineEdit para actualizar el QLabel al presionar Enter.

```
1 void MainWindow::on_pushButtonMostrar_clicked() {
2     QString entrada = ui->lineEdit->text();
3     ui->label->setText("Escribiste: " + entrada);
4 }
5 // Alternativa: usando Enter en el QLineEdit
6 connect(ui->lineEdit, &QLineEdit::returnPressed, this, [this]() {
7     ui->label->setText("Escribiste: " + ui->lineEdit->text());
8 });
```

Listing 3: Ejemplo básico de captura y despliegue

Reflexión

¿Qué diferencia hay entre usar un botón y la señal `returnPressed()`?

8. Conversiones numéricas para hardware

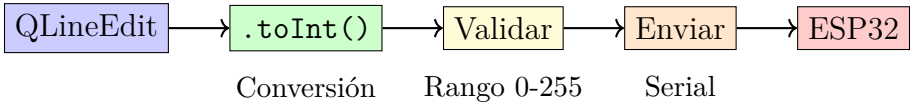
Métodos de Conversión

- `.toInt(&ok)` → Entero (32 bits)
- `.toDouble(&ok)` → Decimal (double)
- `.toFloat(&ok)` → Decimal (float)
- `.toUInt(&ok)` → Entero sin signo

Importante para ESP32

- Validar siempre antes de enviar
- ESP32 usa principalmente enteros
- Rangos típicos: 0-255 (PWM), 0-4095 (ADC)

```cpp
// Ejemplo: Control PWM en ESP32
void MainWindow::onPWMButtonClick() {
    bool ok;
    int pwmValue = ui->lineEditPWM->text().toInt(&ok);

    if(ok && pwmValue >= 0 && pwmValue <= 255) {
        // Conversin exitosa y en rango
        QString comando = "PWM:" + QString::number(pwmValue);
        serial->write(comando.toUtf8());
        ui->labelStatus->setText("PWM: " + QString::number(pwmValue));
    } else {
        // Error de conversion o fuera de rango
        ui->labelStatus->setText("Error: Valor 0-255 required");
        ui->lineEditPWM->setStyleSheet("background-color: #FFCCCC;");
    }
}
```



- **Mejoras practicas**: Siempre validar rango después de la conversión
- **Feedback visual**: Cambiar color de fondo en error

Práctica 4: Validación de entradas numéricas

Instrucciones

1. En tu proyecto, agrega dos **QLineEdit** para capturar números enteros.
2. Agrega un botón "Sumar" (**QPushButton**).
3. Al hacer clic:
   - Convierte los textos con `.toInt(&ok)`.
   - Si ambos son válidos, calcula la suma y muéstrala en un **QLabel**.
   - Si alguno no es válido, muestra un mensaje de error en el QLabel y cambia el fondo del QLineEdit problemático a rojo.

```cpp
void MainWindow::on_pushButtonSumar_clicked() {
    bool ok1, ok2;
    int num1 = ui->lineEditNum1->text().toInt(&ok1);
    int num2 = ui->lineEditNum2->text().toInt(&ok2);

    if(ok1 && ok2) {
        int suma = num1 + num2;
        ui->labelResultado->setText("Suma: " + QString::number(suma));
        ui->lineEditNum1->setStyleSheet("");
        ui->lineEditNum2->setStyleSheet("");
    } else {
        ui->labelResultado->setText("Error: ingresa solo nmeros");
        if(!ok1) ui->lineEditNum1->setStyleSheet("background-color:#FFCCCC;");
        if(!ok2) ui->lineEditNum2->setStyleSheet("background-color:#FFCCCC;");
    }
}
```

Listing 4: Ejemplo de validación con toInt()

Reflexión

¿Qué pasaría si no validamos los datos antes de usarlos en cálculos o enviarlos a la ESP32?

9. Concatenación de texto para comandos

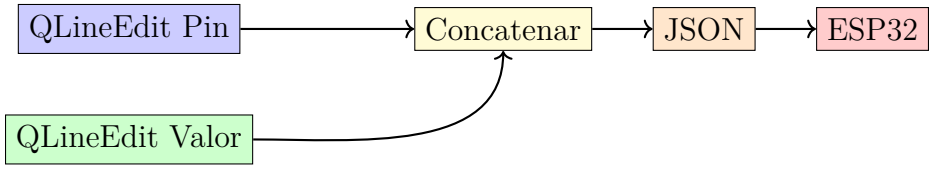
Métodos de Concatenación

- **operador +** → `str1 + str2`
- **.arg()** → `"Pino %1".arg(13)`
- **.append()** → `str.append("valor")`
- **QStringLiteral** → Más eficiente

Para Hardware

- Crear comandos estructurados
- Formatear datos para serial
- Protocolos: JSON, CSV, custom

```cpp
// Ejemplo: Comando JSON para ESP32
void MainWindow::onSendCommand() {
    QString pin = ui->lineEditPin->text();
    QString value = ui->lineEditValue->text();

    // Concatenacion para JSON
    QString comando = "{\"pin\":" + pin +
                      ",\"value\":" + value +
                      ",\"mode\":\"output\"}";

    // Alternativa con .arg() (mas legible)
    // QString comando =
    //   QString("{\"pin\":%1,\"value\":%2,\"mode\":\"
     output\"}")
    //   .arg(pin).arg(value);

    serial->write(comando.toUtf8());
    ui->textEditLog->append("Enviado: " + comando);
}
```



- **Recomendación**: Usar `.arg()` para comandos complejos
- **Ventaja**: Más legible y menos propenso a errores

Práctica 5: Construir un comando JSON

Instrucciones

1. Agrega tres **QLineEdit**: *Pin*, *Valor* y *Modo*.
2. Agrega un **QPushButton**: "Generar JSON".
3. Agrega un **QLabel** o **QTextEdit** para mostrar el resultado.
4. Al presionar el botón:
   - Lee los campos con `.text()`.
   - Valida que *Pin* sea número (`.toInt(&ok)`).
   - Forma un JSON con `.arg()` y añade `\n` al final.
   - Muestra el JSON en la interfaz (y *no lo envíes* todavía).

```
1  void MainWindow::on_btnGenerarJSON_clicked() {
2      bool ok = false;
3      int pin = ui->lineEditPin->text().toInt(&ok);
4      const QString valor = ui->lineEditValor->text().trimmed();   // p.ej. "1", "0", "255", "HIGH"
5      const QString modo  = ui->lineEditModo->text().trimmed();    // p.ej. "output"
6
7      if(!ok) {
8          ui->labelEstado->setText("Error: Pin debe ser entero.");
9          ui->lineEditPin->setStyleSheet("background:#FFCCCC;");
10         return;
```

```
    }
    ui->lineEditPin->setStyleSheet(""); // limpiar error

    // Construccin legible con .arg() y salto de lnea final
    const QString json = QString("{\"pin\":%1,\"valor\":\"%2\",\"modo\":\"%3\"}\n")
                         .arg(pin).arg(valor).arg(modo);

    // Mostrar en la UI (usar QTextEdit si quieres varias lneas)
    ui->textEditJSON->clear();
    ui->textEditJSON->append(json);

    // (Prximo paso en otra prctica: serial->write(json.toUtf8()); )
    ui->labelEstado->setText("JSON generado.");
}
```

Listing 5: Crear JSON con validación mínima

Comprobación

- Prueba: Pin=13, Valor=1, Modo=output
  Debe quedar: `{"pin":13,"valor":"1","modo":.°utput"}\n`
- ¿Qué ocurre si *Pin* está vacío o no es número?

10. Conversión número a texto para interfaces

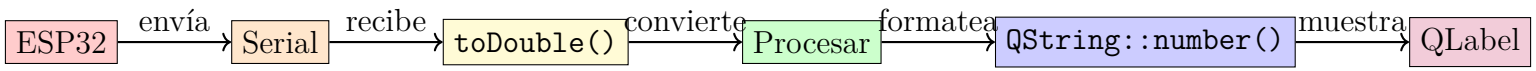
Métodos de Conversión

- **`QString::number(int)`** → Entero
- **`QString::number(double)`** → Decimal
- **`QString::number(float)`** → Float
- **`.arg() con formatos`** → `"%1".arg(25.5, 0, 'f', 2)`

Formatos Comunes

- **Entero**: `QString::number(123)`
- **Decimal 2 digits**: `QString::number(3.1416, 'f', 2)`
- **Hexadecimal**: `QString::number(255, 16)`
- **Binario**: `QString::number(5, 2)`

```
// Ejemplo: Mostrar datos del ESP32
void MainWindow::onSerialDataReceived() {
    QByteArray data = serial->readAll();
    QString response = QString::fromUtf8(data);

    // Supongamos que ESP32 envia: "TEMP:25.6,HUM:60"
    if(response.startsWith("TEMP:")) {
        // Extraer valor numerico
        QString tempStr = response.mid(5, 4);
        bool ok;
        double temperature = tempStr.toDouble(&ok);

        if(ok) {
            // Convertir a texto con formato
            QString displayText =
                "Temperatura: " +
                QString::number(temperature, 'f', 1) +
                " C";
            ui->labelTemperature->setText(displayText);
        }
    }
}
```



- **Flujo bidireccional**: Entrada usuario → ESP32 → Salida visual
- **Importante**: Formatear números para mejor legibilidad
- **Ejemplo real**: Temperaturas, valores PWM, lecturas ADC

Práctica 6: Formatear y presentar números en la interfaz

Instrucciones

1. En tu formulario agrega:
   - **QLineEdit** *lineEditTemp* para temperatura (ej. 25.6789)
   - **QLineEdit** *lineEditDec* para decimales (ej. 1, 2, 3)
   - **QLineEdit** *lineEditPWM* para un entero (0–255)
   - **QPushButton** *btnFormatear*
   - **QLabel** *labelSalida* (para mostrar el resultado)
2. Al presionar *btnFormatear*:
   - Convierte *lineEditTemp* a `double` y *lineEditDec* a `int`.
   - Muestra **temperatura** con `QString::number(temp, 'f', dec)`.
   - Convierte **PWM** a `hex` y `bin` con `QString::number(valor, base)`.
   - Presenta todo en *labelSalida* con un solo `setText()`.

```
#include <QDoubleValidator>
#include <QIntValidator>

void MainWindow::inicializarValidadores() {
    // Llama esto en el constructor despus de ui->setupUi(this);
```

```
6       auto *vTemp = new QDoubleValidator(-50.0, 150.0, 4, this); // min, max, decimales
7       vTemp->setNotation(QDoubleValidator::StandardNotation);
8       ui->lineEditTemp->setValidator(vTemp);
9
10      auto *vDec  = new QIntValidator(0, 6, this);    // 0 a 6 decimales
11      ui->lineEditDec->setValidator(vDec);
12
13      auto *vPWM  = new QIntValidator(0, 255, this);  // rango PWM tpico
14      ui->lineEditPWM->setValidator(vPWM);
15  }
16
17  void MainWindow::on_btnFormatear_clicked() {
18      bool okT=false, okD=false, okP=false;
19
20      const double temp = ui->lineEditTemp->text().toDouble(&okT);
21      const int dec = ui->lineEditDec->text().toInt(&okD);
22      const int pwm = ui->lineEditPWM->text().toInt(&okP);
23
24      if(!(okT && okD && okP)) {
25          ui->labelSalida->setText("Error: verifica temperatura/decimales/PWM.");
26          if(!okT) ui->lineEditTemp->setStyleSheet("background:#FFCCCC;");
27          if(!okD) ui->lineEditDec->setStyleSheet("background:#FFCCCC;");
28          if(!okP) ui->lineEditPWM->setStyleSheet("background:#FFCCCC;");
29          return;
30      }
31      // Limpiar estilos de error si todo est bien
32      ui->lineEditTemp->setStyleSheet("");
```

```
33      ui->lineEditDec->setStyleSheet("");
34      ui->lineEditPWM->setStyleSheet("");
35
36      // 1) Formato de temperatura con 'dec' decimales
37      const QString tempFmt = QString::number(temp, 'f', dec) + " C";
38
39      // 2) PWM en decimal, hexadecimal y binario
40      const QString pwmDec = QString::number(pwm);
41      const QString pwmHex = QString::number(pwm, 16).toUpper(); // base 16
42      const QString pwmBin = QString::number(pwm, 2);            // base 2
43
44      // 3) Presentacin en una sola etiqueta
45      const QString salida = QString("Temperatura: %1\nPWM: %2 (hex 0x%3, bin %4)")
46                             .arg(tempFmt, pwmDec, pwmHex, pwmBin);
47
48      ui->labelSalida->setText(salida);
49  }
```

Listing 6: Formateo con QString::number y validación básica

Prueba / Reto

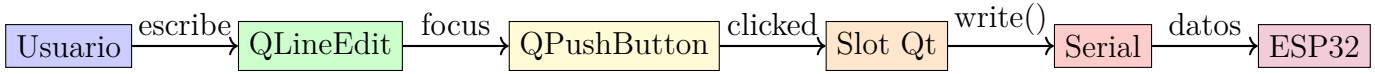
- Prueba con: Temp = 25.6789 y Dec = 1, 2, 3 (observa el redondeo).
- Prueba PWM = 160 → debería verse como `0xA0` en hex y su binario.
- Reto: agrega la **unidad** (°C) solo si *dec* ≥ 1; si no, mostrar solo el entero.

11. Flujo completo: Usuario → Qt → ESP32

Pasos del Flujo

1. Usuario escribe comando en QLineEdit
2. Botón presionado (clicked signal)
3. Slot lee texto con `.text()`
4. Validar y convertir datos
5. Formatear comando para ESP32
6. Enviar por puerto serial
7. Recibir respuesta del ESP32
8. Mostrar resultado en interfaz

```cpp
// CONEXION seal-slot en constructor
connect(ui->btnEnviar, &QPushButton::clicked,
        this, &MainWindow::enviarComandoESP32);

void MainWindow::enviarComandoESP32() {
    // 1. Leer entrada usuario
    QString entrada = ui->lineEditComando->text();

    // 2. Validar y convertir
    bool ok;
    int valor = entrada.toInt(&ok);

    if(ok && valor >= 0 && valor <= 255) {
        // 3. Formatear comando
        QString comando =
            QString("{\"action\":\"pwm\",\"value\":%1}")
            .arg(valor);

        // 4. Enviar a ESP32
        serial->write(comando.toUtf8());
        ui->textEditLog->append("Enviado: " + comando);
    } else {
        ui->labelStatus->setText("Error: Valor 0-255");
    }
}
```



- **Flujo completo**: Interfaz → Lógica → Hardware → Feedback
- **Importante**: Validación en cada paso del proceso
- **Next step**: En la siguiente clase, recibiremos datos del ESP32

Práctica 7: Enviar JSON al ESP32 y recibir respuesta

Objetivo

- Conectar la GUI de Qt con la ESP32 a través de `QSerialPort`.
- Formar un comando en formato JSON y enviarlo.
- Recibir y mostrar la respuesta del ESP32 en la interfaz.

Instrucciones

1. En tu GUI agrega:
   - **QLineEdit** para escribir el pin (ej: 13).
   - **QLineEdit** para escribir el estado (ej: 1 o 0).
   - **QPushButton** "Enviar".
   - **QTextEdit** para mostrar las respuestas.
2. Forma el JSON con el formato: `{"pin":13,"estado":1}\n`
3. Envía con `serial->write(json.toUtf8());`
4. Muestra la respuesta recibida en el `QTextEdit`.

```
1 void MainWindow::on_btnEnviar_clicked() {
```

```
    if (!serial || !serial->isOpen()) {
        ui->textEditLog->append("Error: Puerto no abierto.");
        return;
    }
    QString pin = ui->lineEditPin->text();
    QString estado = ui->lineEditEstado->text();

    QString json = QString("{\"pin\":%1,\"estado\":%2}\n")
                   .arg(pin).arg(estado);

    serial->write(json.toUtf8());
    ui->textEditLog->append("Enviado: " + json);
}

void MainWindow::leerRespuestaSerial() {
    while (serial->canReadLine()) {
        QString linea = QString::fromUtf8(serial->readLine()).trimmed();
        ui->textEditLog->append("ESP32: " + linea);
    }
}
```

Listing 7: Ejemplo de envío y recepción en Qt

Prueba / Reto

- Envía un comando válido: `{"pin":13,"estado":1}`
- Observa si el LED realmente enciende/apaga.
- Reto: haz que la ESP32 responda con JSON, por ejemplo: `{"ok":true,"pin":13,"estado":1}`

Práctica 8: Feedback visual (estado del LED)

Objetivo

- Mostrar en la GUI si el LED quedó **encendido** o **apagado**.
- Cambiar color de un `QLabel` según la respuesta del ESP32.
- (Opcional) Parsear JSON de respuesta para hacerlo robusto.

Instrucciones

1. Agrega un **QLabel** llamado *labelLedEstado*.
2. Cuando llegue una respuesta del ESP32:
   - Si contiene `"LED encendido"` → texto “LED ENCENDIDO” en **verde**.
   - Si contiene `"LED apagado"` → texto “LED APAGADO” en **rojo**.
3. (Opcional) Si la respuesta es JSON `{"ok":true,"estado":1}`, usa ese dato en lugar de texto libre.

```
1 void MainWindow::actualizarEstadoLEDDesdeRespuesta(const QString &respuesta) {
2     // Caso 1: respuesta en texto simple
3     if (respuesta.contains("LED encendido", Qt::CaseInsensitive)) {
4         ui->labelLedEstado->setText("LED ENCENDIDO");
5         ui->labelLedEstado->setStyleSheet("color: white; background:#2E7D32; padding:4px; font-
```

```
    weight:bold;");
    } else if (respuesta.contains("LED apagado", Qt::CaseInsensitive)) {
        ui->labelLedEstado->setText("LED APAGADO");
        ui->labelLedEstado->setStyleSheet("color: white; background:#C62828; padding:4px; font-
    weight:bold;");
    } else {
        ui->labelLedEstado->setText("Estado desconocido");
        ui->labelLedEstado->setStyleSheet("color: #333; background:#FFEE58; padding:4px;");
    }
}

// Llama a esta funcin desde tu lector de serial:
void MainWindow::leerRespuestaSerial() {
    while (serial->canReadLine()) {
        const QString linea = QString::fromUtf8(serial->readLine()).trimmed();
        ui->textEditLog->append("ESP32: " + linea);
        actualizarEstadoLEDDesdeRespuesta(linea);
    }
}
```

Listing 8: Cambiar color del QLabel según la respuesta

Opcional: respuesta en JSON

```cpp
// Si tu ESP32 responde JSON, por ejemplo: {"ok":true,"pin":13,"estado":1}
#include <QJsonDocument>
#include <QJsonObject>

void MainWindow::actualizarEstadoDesdeJSON(const QString &linea) {
    const QJsonDocument doc = QJsonDocument::fromJson(linea.toUtf8());
    if (!doc.isObject()) return;
    const QJsonObject obj = doc.object();
    const int estado = obj.value("estado").toInt(-1);

    if (estado == 1) {
        ui->labelLedEstado->setText("LED ENCENDIDO");
        ui->labelLedEstado->setStyleSheet("color: white; background:#2E7D32; padding:4px; font-weight:bold;");
    } else if (estado == 0) {
        ui->labelLedEstado->setText("LED APAGADO");
        ui->labelLedEstado->setStyleSheet("color: white; background:#C62828; padding:4px; font-weight:bold;");
    }
}
```

Prueba / Reto

- Envía `{"pin":13,"estado":1}\n` y verifica que el estado cambie a **verde**.
- Envía `{"pin":13,"estado":0}\n` y verifica que el estado cambie a **rojo**.
- **Reto**: añade un `QIcon` (/) junto al texto según el estado.

12. Resumen de la teoría

- QLineEdit captura texto escrito por el usuario.
- Métodos clave: `text()`, `setText()`, `toInt()`, `toDouble()`.
- Se pueden concatenar textos y mostrar resultados en un QLabel.
- Con `QString::number()` convertimos resultados numéricos a texto.

13. Preguntas rápidas de comprobación

Marca la opción correcta

0 ¿Qué hace `.text()` en un `QLineEdit`?

- Limpia el contenido
- Devuelve el texto que escribió el usuario
- Convierte el texto a número

0 ¿Qué método usarías para colocar un texto desde el programa?

- `.toInt()`
- `.placeholderText()`
- `.setText()`

0 ¿Qué parámetro opcional indica si una conversión numérica fue válida?

- `&ok` en `.toInt(&ok)`
- `&ok` en `.setText(&ok)`
- `&err` en `.toInt(&err)`

0 ¿Cuál es la forma recomendada de restringir que un `QLineEdit` acepte solo números?

- Usar `.clear()`
- Usar `.setValidator()`
- Usar `.setText()`

0 ¿Qué método usarías para mostrar un número en un `QLabel`?

- `QString::number()`
- `.toDouble()`
- `.placeholderText()`

14. Ejercicio 1: Formatear comando para ESP32

Objetivo del Ejercicio

- Crear interfaz con 2 QLineEdit y 1 QPush-Button
- Concatenar valores para formar comando JSON
- Mostrar comando formateado en QLabel
- Preparar base para envío a ESP32

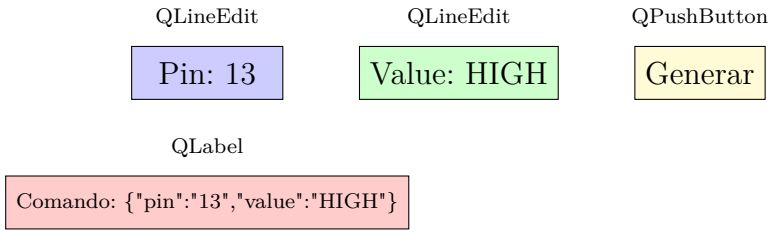
Interfaz Requerida

- **QLineEdit 1**: Número de pin (ej: 13)
- **QLineEdit 2**: Valor (ej: HIGH, 255, 1)
- **QPushButton**: "Generar Comando"
- **QLabel**: Muestra el comando JSON

```
// Ejemplo de resultado esperado
void MainWindow::onGenerarComando() {
    QString pin = ui->lineEditPin->text();
    QString valor = ui->lineEditValor->text();

    // Formatear comando JSON
    QString comando =
        QString("{\"pin\":%1,\"value\":\"%2\"}")
        .arg(pin).arg(valor);

    // Mostrar en interfaz
    ui->labelComando->setText("Comando: " + comando);

    // Para siguiente ejercicio:
    // serial->write(comando.toUtf8());
}
```

Comando Esperado

```
{"pin":"13","value":"HIGH"}
{"pin":"5","value":"255"}
{"pin":"2","value":"1"}
```



- **Propósito**: Practicar concatenación para protocolos seriales
- **Base**: Este ejercicio será extendido en el Ejercicio 3
- **Tiempo estimado**: 10-15 minutos

15. Ejercicio 2: Validación y control de PWM

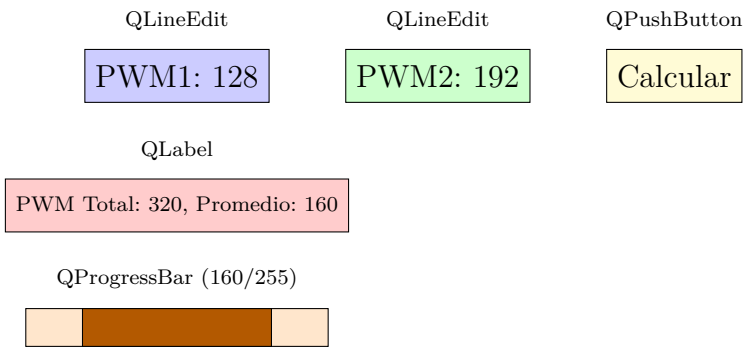
Objetivo del Ejercicio

- Crear interfaz para control PWM
- Validar rangos numéricos (0-255)
- Practicar conversión y validación
- Simular envío de datos a ESP32

Interfaz Requerida

- **QLineEdit 1**: Valor PWM 1 (0-255)
- **QLineEdit 2**: Valor PWM 2 (0-255)
- **QPushButton**: Çalcular Total"
- **QLabel**: Muestra resultado y estado
- **QProgressBar**: Barra visual del PWM

```
1  void MainWindow::onCalcularPWM() {
2      bool ok1, ok2;
3      int pwm1 = ui->lineEditPWM1->text().toInt(&ok1);
4      int pwm2 = ui->lineEditPWM2->text().toInt(&ok2);
5
6      if(ok1 && ok2 && pwm1 >= 0 && pwm1 <= 255 &&
7         pwm2 >= 0 && pwm2 <= 255) {
8
9          int total = pwm1 + pwm2;
10         int promedio = total / 2;
11
12         ui->labelResultado->setText(
13             QString("PWM Total: %1, Promedio: %2")
14             .arg(total).arg(promedio));
15
16         ui->progressBar->setValue(promedio);
17
18     } else {
19         ui->labelResultado->setText("Error: Valores 0-255")
       ;
20         ui->lineEditPWM1->setStyleSheet("background: #
       FFCCCC;");
21         ui->lineEditPWM2->setStyleSheet("background: #
       FFCCCC;");
22     }
23 }
```



- **Aplicación real**: Control de intensidad LED con ESP32
- **Importante**: Validar rangos PWM (0-255)
- **Extensión**: Agregar `.setValidator()` para solo números
- **Tiempo estimado**: 15-20 minutos

16. Ejercicio 3: Registro de datos en CSV

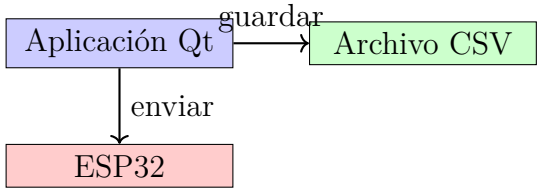
Objetivo del Ejercicio

- Extender el ejercicio anterior
- Guardar datos en archivo CSV
- Incluir timestamp automático
- Crear histórico de operaciones

Estructura CSV

- **Timestamp**: Fecha y hora
- **PWM1**: Valor del primer PWM
- **PWM2**: Valor del segundo PWM
- **Total**: Suma de PWM
- **Promedio**: Valor promedio
- **Estado**: Éxito/Error

```cpp
void MainWindow::guardarCSV(int pwm1, int pwm2,
                            int total, int promedio,
                            bool exito) {
    QFile file("registro_pwm.csv");
    if (file.open(QIODevice::Append | QIODevice::Text)) {
        QTextStream stream(&file);
        QDateTime now = QDateTime::currentDateTime();

        stream << now.toString("yyyy-MM-dd HH:mm:ss") << ",
    "
               << pwm1 << ","
               << pwm2 << ","
               << total << ","
               << promedio << ","
               << (exito ? "EXITO" : "ERROR") << "\n";

        file.close();
        ui->labelStatus->setText("Datos guardados en CSV");
    }
}

// Llamar desde onCalcularPWM():
guardarCSV(pwm1, pwm2, total, promedio, true);
```



Timestamp	PWM1	PWM2	Total	Promedio	Estado
2024-01-15 10:30:15	128	192	320	160	EXITO
2024-01-15 10:31:22	255	100	355	177	EXITO
2024-01-15 10:32:05	300	200	-	-	ERROR

- **Aplicación real**: Log de operaciones para debugging
- **Importante**: Verificar permisos de escritura
- **Extensión**: Agregar nombre de archivo con fecha
- **Tiempo estimado**: 20-25 minutos

17. Preguntas de reflexión y mejores prácticas

Preguntas Técnicas

- ¿Por qué es crucial validar datos antes de enviar a hardware?
- ¿Qué podría pasar si enviamos .ABC.en lugar de 255 a un pin PWM?
- ¿Cómo afecta la falta de validación en la estabilidad del ESP32?
- ¿Por qué usar `.setValidator()` en lugar de solo `.toInt()`?

Escenarios de Falla

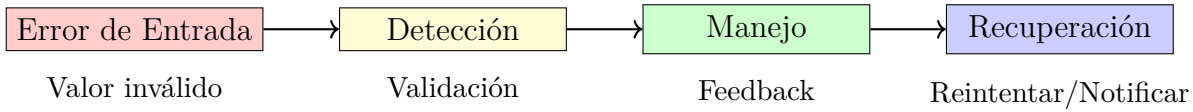
- Usuario escribe .alto.en lugar de "HIGH"
- Valor fuera de rango (PWM: 300 en lugar de 255)
- Caracteres especiales en comandos JSON
- Desconexión repentina del USB
- Buffer serial lleno o congestionado

Seguridad y Robustez

- ¿Cómo prevenir que valores erróneos dañen el hardware?
- ¿Qué protocolos de seguridad implementar para operaciones críticas?
- ¿Cómo manejar timeout en comunicación serial?
- ¿Qué hacer si el ESP32 no responde?

Mejores Prácticas

- Validar → Convertir → Validar rango → Enviar
- Usar feedback visual inmediato
- Loggear todas las operaciones
- Planear para fallas de comunicación
- Documentar protocolos claramente



- **Reflexión final**: En hardware, los errores de software tienen consecuencias físicas
- **Próxima clase**: Implementaremos estas validaciones con QCheckBox y QRadioButton
- **Tarea**: Pensar en 3 posibles fallas en sus proyectos y cómo prevenirlas

18. Integración con el flujo completo

Lo que llevamos

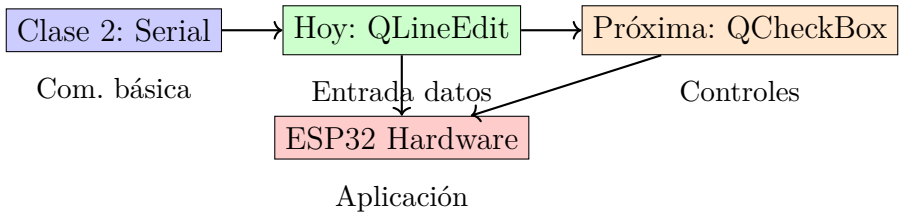
- **Clase 1**: Fundamentos de Qt y serial
- **Clase 2**: Comunicación básica ESP32
- **Hoy**: Entrada de datos con QLineEdit
- Validación, conversión y formato
- Guardado de datos en CSV

Próxima clase

- **QCheckBox**: Opciones múltiples
- **QRadioButton**: Selección única
- Comandos JSON automáticos
- Interfaces más complejas
- Agrupación de controles

Ejemplo próximo tema

```
// Prxima clase: JSON automtico
QJsonObject json;
json["pin"] = ui->spinBoxPin->value();
json["value"] = ui->lineEditValue->text();
json["mode"] = ui->checkBoxDigital->isChecked()
                ? "digital" : "analog";
json["output"] = ui->radioButtonHigh->isChecked()
                 ? "HIGH" : "LOW";

QJsonDocument doc(json);
serial->write(doc.toJson());
```



- **Flujo completo dominado**: Usuario → Qt → ESP32 → Hardware
- **Próximo objetivo**: Interfaces más complejas y robustas
- **Preparación**: Traer ideas de proyectos para la siguiente clase

De la entrada del usuario al control real de hardware
Fundamentos para crear interfaces profesionales con microcontroladores

19. Recursos y Material Adicional

Documentación Oficial

- QLineEdit Documentation
- QString Documentation
- QSerialPort Documentation

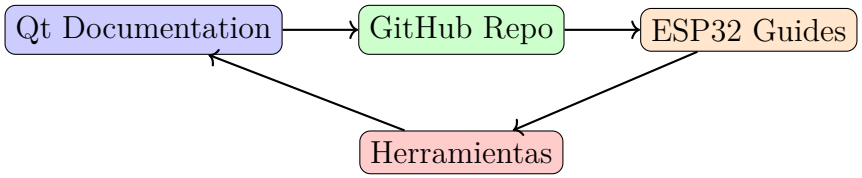
Repositorio GitHub

- Código completo de hoy
- Proyectos base para empezar
- Soluciones de ejercicios
- `github.com/esmarr58/qt-basic-examples`

Ejemplos ESP32

- Código Arduino para ESP32
- Protocolos de comunicación
- Ejemplos de parsing JSON

- Serial Monitor avanzado
- JSON validator online
- Qt Creator shortcuts
- ESP32 programming tools



- **Foro**: Dudas y discusiones en plataforma institucional
- **Office hours**: Horarios de asesoría personalizada
- **Evaluación**: Los ejercicios serán parte del proyecto final

20. Preguntas de Comprensión

Conceptuales

1. ¿Cuál es la diferencia principal entre `.text()` y `.setText()`?
2. ¿Por qué es necesario usar `&ok` en `.toInt(&ok)`?
3. ¿Qué ventaja tiene usar `.arg()` sobre el operador `+`?
4. ¿Cómo previene `.setValidator()` errores de hardware?

De Aplicación

1. Diseña un comando JSON para controlar 2 LEDs
2. ¿Qué pasos seguirías para debuggear comunicación serial?
3. Propón 3 validaciones para un sistema crítico
4. Explica el flujo usuario→hardware→feedback

Prácticas

1. ¿Qué método usarías para recibir un valor decimal?
2. ¿Cómo formatearías un número para mostrar 2 decimales?
3. ¿Qué validación necesita un valor PWM para ESP32?
4. ¿Cómo manejarías un error de conversión?

Soluciones Breves

1. `.text()` lee, `.setText()` escribe
2. Para detectar errores de conversión
3. Más legible y menos propenso a errores
4. Evita entradas inválidas desde el origen

21. Glosario de Términos

Qt Concepts

QLineEdit Widget para entrada de texto de una línea

QString Clase para manejo de cadenas en Qt

Señal-Slot Mecanismo de comunicación entre objetos

Validator Restricción de entrada de datos

SerialPort Comunicación por puerto serial

Hardware

ESP32 Microcontrolador popular con WiFi/BT

PWM Modulación por ancho de pulsos

GPIO Pines de propósito general

Serial Comunicación serie UART

JSON Formato de intercambio de datos

Métodos

`.text()` Obtiene el contenido del QLineEdit

`.setText()` Establece el texto programáticamente

`.toInt()` Convierte texto a entero

`.toDouble()` Convierte texto a decimal

`.arg()` Formateo avanzado de cadenas

Conceptos

Validación Verificar que los datos son correctos

Conversión Cambio entre formatos (texto/número)

Concatenación Unión de cadenas de texto

Protocolo Reglas de comunicación

Feedback Retroalimentación al usuario

22. Contacto y Soporte

ruben.estrada@academicos.udg.mx
Horario de asesoría: Lunes y Miércoles 16:00-18:00
github.com/esmarr58
Plataforma: Classroom/Moodle

¿Preguntas?

Modificado el 9 de septiembre de 2025