

Proyectos de inicio con Arduino

Primeros pasos con la placa EchidnaBlack2 y Arduino IDE



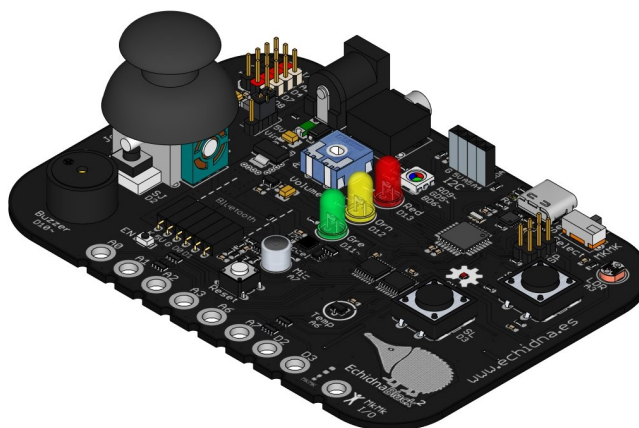
Echidna Educación



Índice

1. Introducción	3
2. Proyectos con Arduino	9
2.1 Hola Mundo	10
2.2 Semáforo.....	13
2.3 Interruptor de luz	17
2.4 Timbre	21
2.5 Brillo LED	24
2.6 Interruptor crepuscular	28
2.7 Piano de frutas	31
2.8 El echidna dice la temperatura.....	35
2.9 Mezclamos colores	39
2.10 Vúmetro	44
2.11 Regulador de luz con joystick	49
2.12 Nivel de burbuja	53
2.13 Control desde el ordenador.....	60
3. Licencia	65

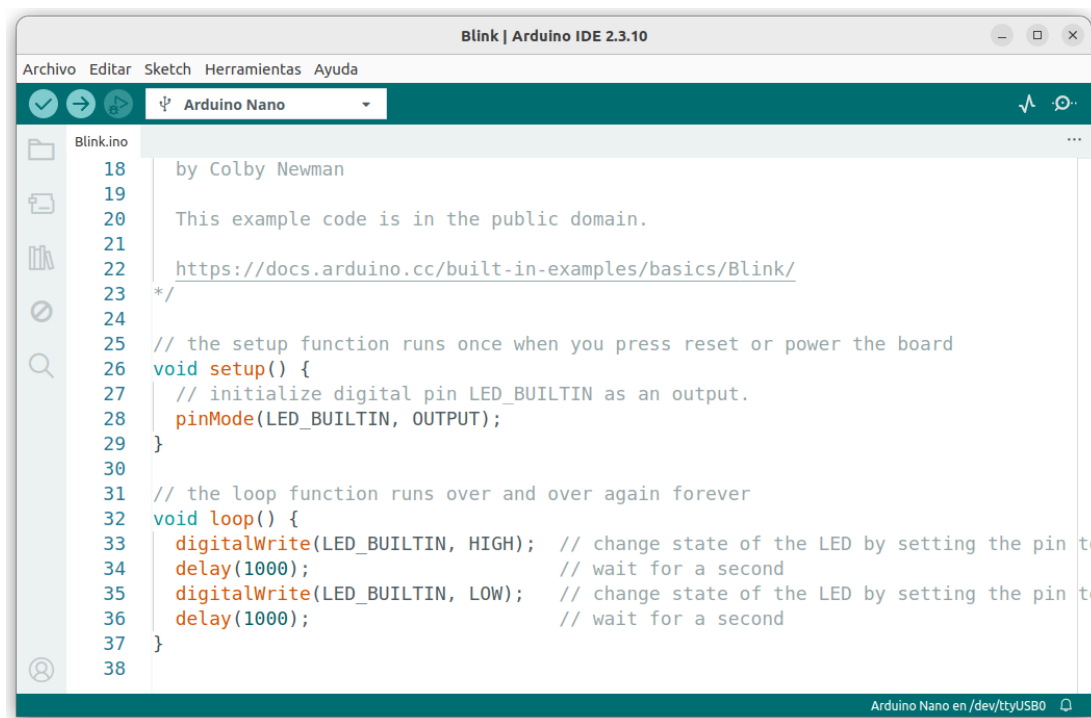
EchidnaBlack2 es una **placa** con diversos componentes integrados (LED, pulsadores, zumbador, sensores...) pensada para aprender los fundamentos de la **programación** y la **robótica**. Su objetivo es fomentar el **pensamiento computacional** en Primaria, Secundaria, Bachillerato y F.P., y estimular la creatividad con proyectos prácticos que conectan el mundo digital y el analógico.



Te proponemos una serie de **proyectos sencillos** para dar tus primeros pasos. Cada proyecto presenta un componente de la placa y la instrucción de Arduino que lo controla.

Instalar Arduino IDE

1. Descarga la última versión de **Arduino IDE 2** desde la página oficial: www.arduino.cc/en/software. Elige la descarga para tu sistema operativo.
2. Instálalo como cualquier otro programa. Si necesitas ayuda, consulta la [guía de instalación de Arduino](#).
3. Abre Arduino IDE. La primera vez puede tardar un poco porque descarga algunos componentes.



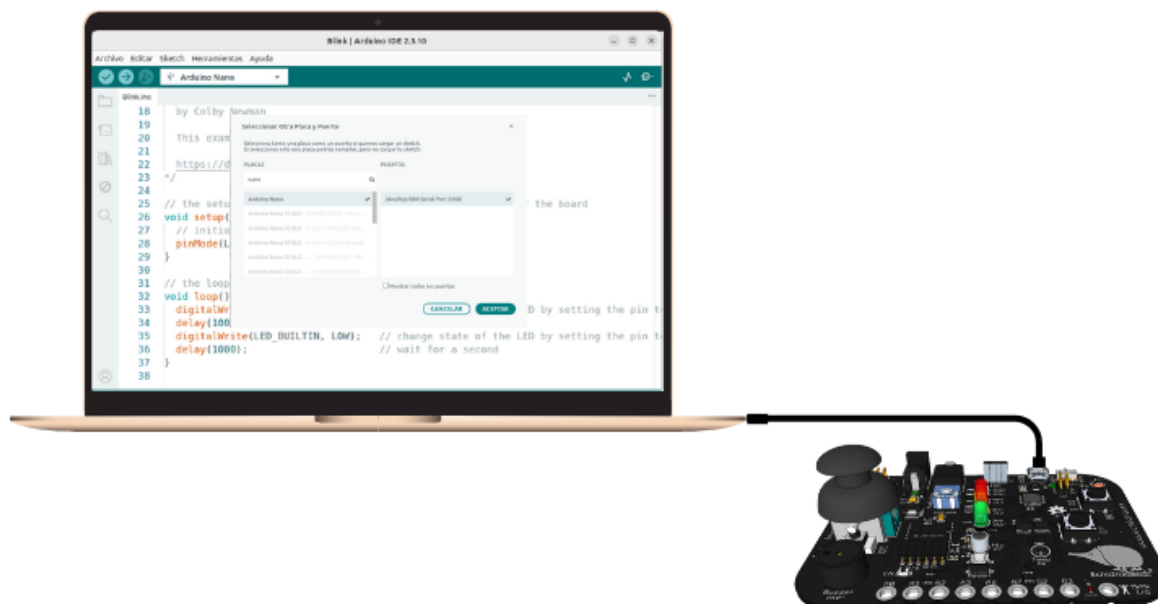
La placa se comunica con el ordenador mediante el chip **CH340**. En GNU/Linux no hace falta instalar nada; en Windows y macOS tienes que instalar su controlador (driver), el **Driver CH341**. Encontrarás los enlaces de descarga en las [preguntas frecuentes del manual](#).

En GNU/Linux, si Arduino IDE no puede acceder al puerto de la placa, da permiso a tu usuario desde una terminal con `sudo usermod -a -G dialout $USER` y vuelve a iniciar sesión.

Conectar la placa

Conecta la EchidnaBlack2 al ordenador con el cable **USB-C** y abre Arduino IDE. Antes de cargar un programa tienes que decirle al IDE **qué placa** vas a programar y **a qué puerto** está conectada. En el menú **Herramientas** elige:

- **Placa:** Arduino AVR Boards → **Arduino Nano**.
- **Procesador:** **ATmega328P**.
- **Puerto:** el puerto USB al que está conectada la placa. Su nombre depende del sistema operativo y el número puede variar según los dispositivos que tengas conectados:
 - GNU/Linux: `/dev/ttyUSB0`
 - Windows: `COM3`
 - macOS: `/dev/cu.usbserial-1410`



Estructura de un programa

Un programa de Arduino se llama **sketch** y siempre tiene, como mínimo, dos **funciones**: `setup()` y `loop()`. Es lo que aparece cuando creas un programa nuevo en Arduino IDE:

```
1 // aquí van las constantes y las variables
2
3 void setup() {
4   // se ejecuta una sola vez, al encender la placa
5 }
6
7 void loop() {
8   // se repite una y otra vez, para siempre
9 }
```

- **Antes de `setup()`**: declaramos las **constantes** (como los pines de los componentes) y las **variables** que usará el programa.
- **`setup()`**: se ejecuta **una sola vez**, al encender la placa o al cargar el programa. La usamos para preparar la placa, por ejemplo, para indicar qué pines son entradas y cuáles salidas.
- **`loop()`**: se ejecuta **una y otra vez**, de forma infinita, mientras la placa tenga alimentación. Aquí va lo que queremos que la placa haga continuamente: leer sensores, encender LED, hacer sonar el zumbador...

Al escribir código, ten en cuenta estas reglas:

- Cada instrucción termina con **punto y coma (;)**.



- Las **llaves** (`{` y `}`) marcan dónde empieza y dónde termina cada bloque de instrucciones, como el contenido de `setup()` o de `loop()`.
- Arduino distingue entre **mayúsculas y minúsculas**: `digitalWrite` funciona, pero `digitalwrite` da error.
- Lo que va detrás de `//` es un **comentario**: sirve para explicar el programa a las personas que lo leen y la placa no lo ejecuta.

Cargar un programa en la placa

Con la placa conectada y elegidos la placa, el procesador y el puerto, ya puedes cargar tu programa:

1. **Escribe el código** en el editor de Arduino IDE (o cópialo del proyecto).
2. **Guárdalo** con **Archivo → Guardar**. Arduino IDE guarda cada programa en una carpeta con su mismo nombre.
3. Pulsa el botón **Verificar** (el de la marca de verificación, arriba a la izquierda). El IDE comprueba que el código está bien escrito y lo **compila**, es decir, lo traduce a instrucciones que entiende el microcontrolador. Si hay algún error, lo verás en la parte inferior de la ventana, indicando la línea en la que está.
4. Pulsa el botón **Subir** (el de la flecha, junto al anterior). El IDE vuelve a compilar el programa y lo **carga** en la placa. Al terminar, la parte inferior de la ventana indica que la carga se ha completado.

En cuanto termina la carga, la placa empieza a ejecutar el programa. El programa queda guardado en la placa: aunque la desconectes, cuando vuelvas a alimentarla seguirá funcionando hasta que cargues otro.

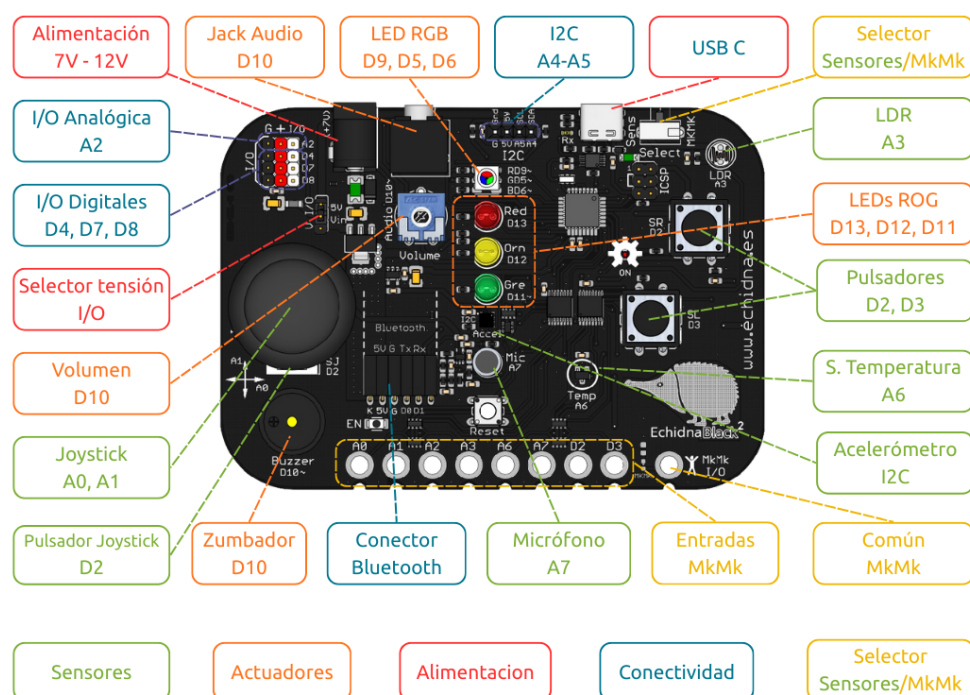
Si la carga falla, revisa que:

- La placa está conectada y has elegido el **puerto** correcto.
- Has elegido la placa **Arduino Nano** con el procesador **ATmega328P**.
- No hay otro programa usando el puerto, como EchidnaML.

¡ATENCIÓN! Al cargar un programa desde Arduino IDE se borra el programa **StandardFirmata** que necesita EchidnaML para comunicarse con la placa. Si después quieres volver a usar la placa con EchidnaML, tendrás que instalarlo de nuevo, como se explica en el [manual](#).

Pines de la EchidnaBlack2

Cada componente de la placa está conectado a un **pin** del microcontrolador. En los programas usaremos estos números para indicar con qué componente queremos trabajar. No hace falta que los memorices: en la placa, cada componente lleva **serigrafiado** al lado su número de pin (por ejemplo, Red D13 junto al LED rojo o Temp A6 junto al sensor de temperatura). Los pines marcados con el símbolo ~ admiten **PWM**.



Estos son los pines de los componentes que usaremos en los proyectos:

Componente	Pin	Tipo
LED rojo	D13	Digital
LED naranja	D12	Digital
LED verde	D11	Digital/Analógico
LED RGB: rojo / verde / azul	D9 / D5 / D6	Digital/Analógico
Zumbador	D10	Digital/Analógico
Pulsador SR (derecho)	D2	Digital
Pulsador SL (izquierdo)	D3	Digital
Joystick: eje X / eje Y	A0 / A1	Analógico
Sensor de luz (LDR)	A3	Analógico
Sensor de temperatura	A6	Analógico
Micrófono	A7	Analógico
Acelerómetro	I2C: A4 (SDA) / A5 (SCL)	Digital
Entradas MkMk analógicas	A0, A1, A2, A3, A6, A7	Analógico
Entradas MkMk digitales	D2, D3	Digital

- **Digital:** solo trabaja con dos valores, encendido o apagado (HIGH o LOW).
- **Analógico:** lee valores intermedios, entre 0 y 1023, como la cantidad de luz o la posición del joystick.



- **Digital/Analógico:** es un pin digital que, además, puede dar valores intermedios, entre 0 y 255, por **PWM** (modulación por ancho de pulso). Así podemos regular el brillo del LED verde y del LED RGB o hacer sonar el zumbador.

En el código, los pines digitales se escriben solo con su número (13) y los analógicos con la letra A (A3).

Las **entradas MkMk** (modo Makey Makey) comparten pin con otros componentes de la placa. Para usarlas hay que poner el selector del modo de funcionamiento en **MkMk**.



2. Proyectos con Arduino

En estos proyectos programamos la placa **EchidnaBlack2** con **Arduino IDE**, escribiendo código en **C/C++**. Cada proyecto presenta un componente nuevo de la placa y va creciendo en complejidad.

Proyectos

1. [Hola Mundo](#)
2. [Semáforo](#)
3. [Interruptor de luz](#)
4. [Timbre](#)
5. [Brillo LED](#)
6. [Interruptor crepuscular](#)
7. [Piano de frutas](#)
8. [El echidna dice la temperatura](#)
9. [Mezclamos colores](#)
10. [Vúmetro](#)
11. [Regulador de luz con joystick](#)
12. [Nivel de burbuja](#)
13. [Control desde el ordenador](#)

Estructura de los proyectos

Todos los proyectos siguen la misma estructura:

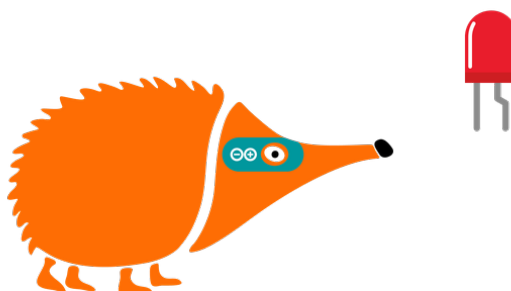
1. **Qué vamos a hacer:** descripción del proyecto, qué vamos a aprender y qué vamos a usar.
2. **Programamos:** el programa completo y cómo funciona, explicado por partes.
3. **Mejóralo:** propuestas para ampliar el proyecto por tu cuenta.

Estilo de programación

Los programas siguen la **línea didáctica de Arduino** más que la de un programador experto en C++: priorizamos que el código sea **fácil de entender y de leer**, aunque no sea la forma más corta o más eficiente de escribirlo. Por eso usamos nombres de variables y constantes claros en español, comentarios que explican qué hace cada parte y estructuras sencillas, aunque a veces repitamos instrucciones que se podrían escribir de forma más compacta.

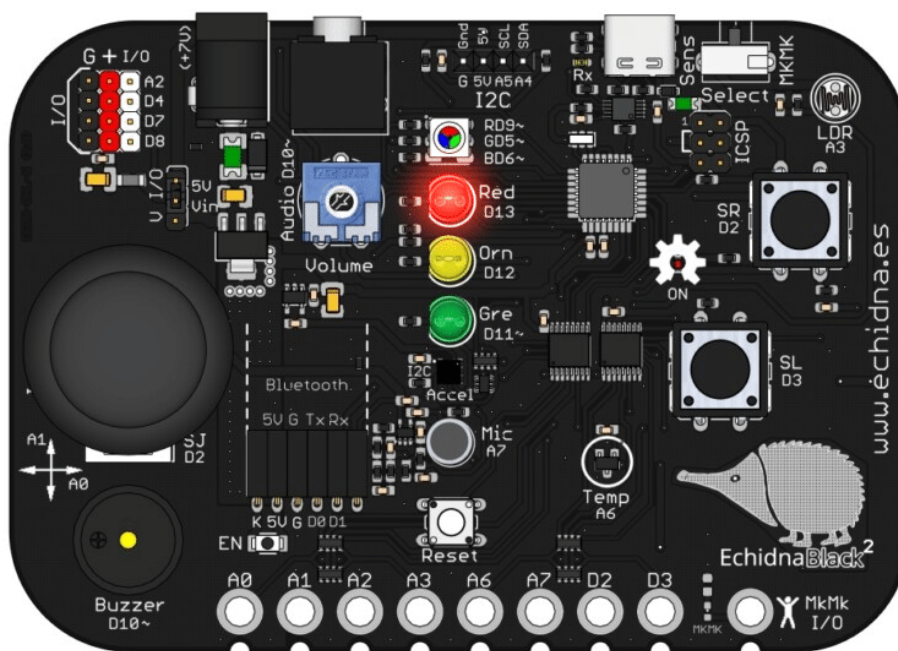


2.1 Hola Mundo



1. Qué vamos a hacer

En este primer proyecto haremos que el **LED rojo** de la placa EchidnaBlack2 se encienda y se apague de forma continua (parpadeo).



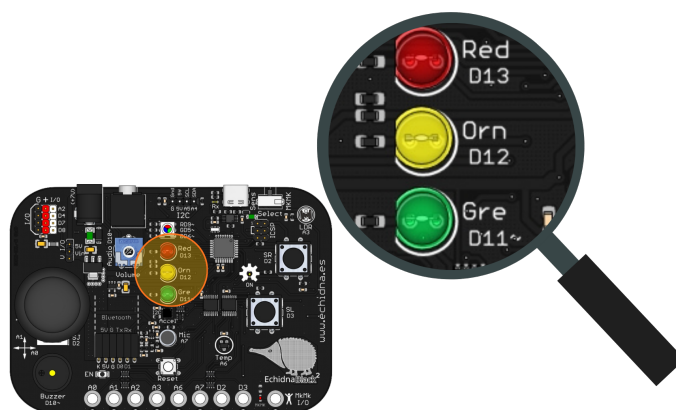
1.1 Qué vamos a aprender

- A crear nuestro primer programa en Arduino IDE.
- A utilizar la función `loop()`, que se repite para siempre, para realizar una **programación cíclica** (tareas que se repiten sin fin).
- A controlar el estado (encendido/apagado) de un componente de la placa.

1.2 Qué vamos a usar

Componentes

- **LED rojo (pin 13):** Se enciende y se apaga para crear el parpadeo.



Programación

Para encender y apagar los LED usamos la función `digitalWrite(pin, valor)`:

- **pin**: el número del pin al que está conectado el LED (el rojo, el 13).
- **valor**: HIGH para **encender** el LED (el pin da 5 V) o LOW para **apagarlo** (el pin da 0 V).

2. Programamos

Este programa utiliza un bucle continuo para ejecutar la siguiente secuencia lógica, creando un parpadeo constante en el LED rojo.

Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Hola Mundo: enciende y apaga el LED rojo cada segundo
2
3 const int ledRojo = 13; // LED rojo conectado al pin 13
4
5 // setup() se ejecuta una sola vez, al encender la placa
6 void setup() {
7   pinMode(ledRojo, OUTPUT); // el pin del LED es una salida
8 }
9
10 // loop() se repite una y otra vez, para siempre
11 void loop() {
12   digitalWrite(ledRojo, HIGH); // enciende el LED
13   delay(1000);                 // espera 1 segundo (1000 ms)
14   digitalWrite(ledRojo, LOW);  // apaga el LED
15   delay(1000);                 // espera 1 segundo
16 }
```

Cómo funciona:

Las líneas que empiezan por `//` son **comentarios**: explican el programa a las personas que lo leen y la placa no las ejecuta.



La constante `ledRojo` (línea 3): guarda el número del pin del LED rojo. Así, en el resto del programa escribimos `ledRojo` en lugar de 13, el código se entiende mejor y, si cambiara el pin, solo tendríamos que cambiarlo en un sitio.

La función `setup()` (líneas 6 a 8): se ejecuta **una sola vez**, al encender la placa o al cargar el programa. Con `pinMode(ledRojo, OUTPUT)` indicamos que el pin del LED es una **salida**: la placa va a enviar señales por él.

La función `loop()` (líneas 11 a 16): crea un ciclo infinito que ejecuta los pasos en orden, de arriba a abajo:

1. **`digitalWrite(ledRojo, HIGH)`**: Envía la señal para encender el LED.
2. **`delay(1000)`**: Mantiene el LED encendido durante un segundo. El tiempo de `delay` se indica en **milisegundos** (1000 ms = 1 s).
3. **`digitalWrite(ledRojo, LOW)`**: Envía la señal para apagar el LED.
4. **`delay(1000)`**: Mantiene el LED apagado durante un segundo antes de volver al paso 1.

3. Mejóralo

Una vez que consigas hacer parpadear el LED, prueba a realizar estas modificaciones por tu cuenta:

1. **Ritmo rápido**: Cambia el tiempo de espera a 0.2 segundos. ¿Qué le ocurre al parpadeo? ¿Qué ocurre si sigues bajando el tiempo de espera?

Pista: recuerda que `delay` cuenta en milisegundos: 0,2 segundos son 200.

2. **Sombra de señal**: Intenta que el LED esté encendido mucho tiempo (2 segundos) y apagado muy poco tiempo (0.1 segundos).

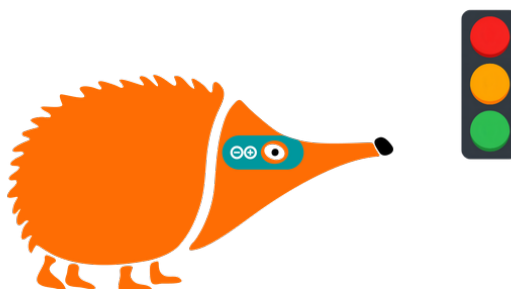
Pista: cada `delay` controla una fase: el primero, cuánto tiempo está encendido el LED, y el segundo, cuánto tiempo está apagado.

3. **Parpadeo alterno**: Haz que el LED rojo y el LED verde se enciendan por turnos: cuando uno está encendido, el otro está apagado.

Pista: declara otra constante para el LED verde (pin 11) y no olvides indicar en `setup()` que su pin también es una salida.

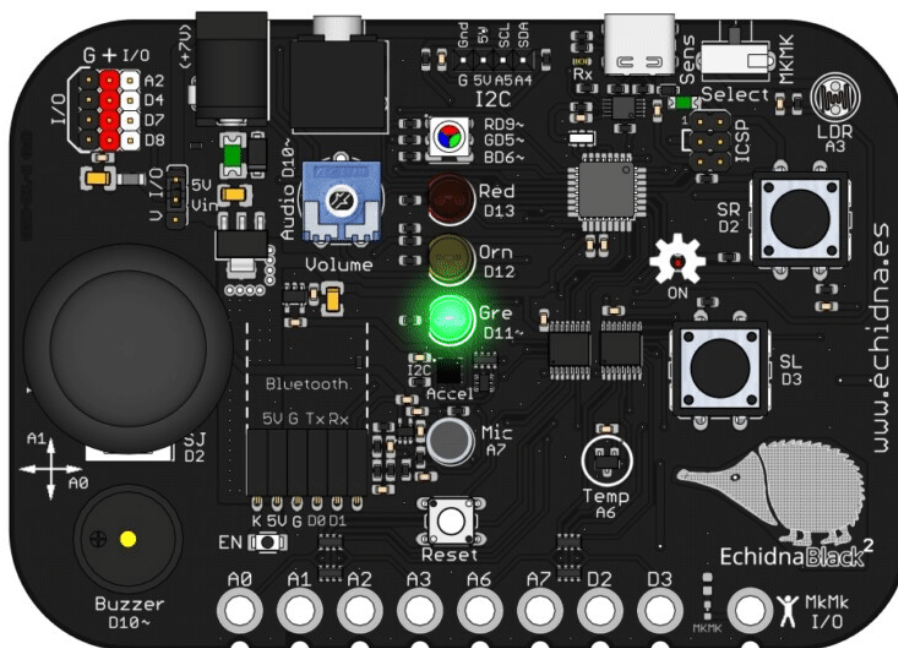


2.2 Semáforo



1. Qué vamos a hacer

Vamos a realizar un semáforo en el que el LED verde se enciende durante 5 segundos, luego se enciende el LED naranja durante 2 segundos y finalmente el LED rojo durante 5 segundos. El ciclo se repite continuamente.



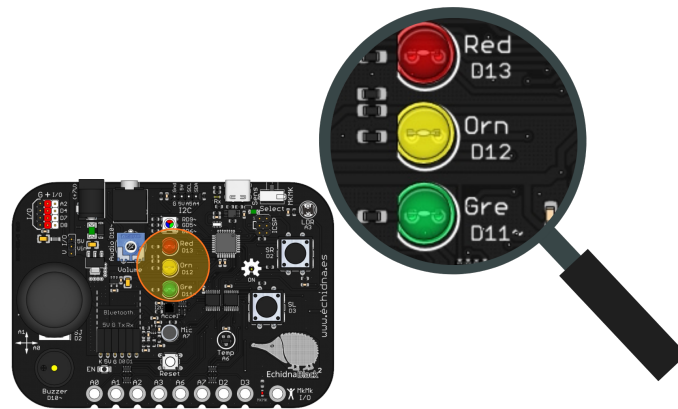
1.1 Qué vamos a aprender

- A diseñar **secuencias temporizadas** (eventos que ocurren en un orden y tiempo exactos).
- A controlar múltiples componentes digitales (LED) de forma coordinada.
- A estructurar un ciclo de estados infinito (**programación cíclica**).

1.2 Qué vamos a usar

Componentes

- **LED verde (pin 11):** Fase de paso.
- **LED naranja (pin 12):** Fase de precaución.
- **LED rojo (pin 13):** Fase de detención.



Programación

Para encender y apagar los LED ya conoces `digitalWrite` y para esperar, `delay`. Antes de usar un pin hay que decirle a la placa cómo lo vamos a usar con la función `pinMode(pin, modo)`:

- **pin**: el número del pin que queremos configurar.
- **modo**: OUTPUT si es una **salida** (la placa envía señales por él, como a un LED) o INPUT si es una **entrada** (la placa lee señales por él, como de un pulsador; lo veremos en el próximo proyecto).

2. Programamos

La programación se basa en una secuencia cíclica donde cada LED permanece encendido durante un tiempo específico y luego pasa al siguiente estado de forma automática.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Semáforo: verde 5 s, naranja 2 s y rojo 5 s
2
3 const int ledVerde = 11;    // LED verde conectado al pin 11
4 const int ledNaranja = 12;  // LED naranja conectado al pin 12
5 const int ledRojo = 13;     // LED rojo conectado al pin 13
6
7 void setup() {
8   // los pines de los tres LED son salidas
9   pinMode(ledVerde, OUTPUT);
10  pinMode(ledNaranja, OUTPUT);
11  pinMode(ledRojo, OUTPUT);
12 }
13
14 void loop() {
15   // fase verde: 5 segundos
16   digitalWrite(ledVerde, HIGH);
17   delay(5000);
18   digitalWrite(ledVerde, LOW);
19
20   // fase naranja: 2 segundos
21   digitalWrite(ledNaranja, HIGH);
22   delay(2000);
23   digitalWrite(ledNaranja, LOW);
24
25   // fase roja: 5 segundos
26   digitalWrite(ledRojo, HIGH);
27   delay(5000);
28   digitalWrite(ledRojo, LOW);
29 }
```

Cómo funciona:

Las constantes (líneas 3 a 5): guardan el pin de cada LED, para usar en el programa nombres como `ledVerde` en lugar de números.

La función `setup()` (líneas 7 a 12): con `pinMode` indicamos que los pines de los tres LED son **salidas**. Cada pin que usemos necesita su propio `pinMode`.

La función `loop()` (líneas 14 a 29): repite para siempre las tres fases del semáforo:

1. El LED verde se enciende durante 5 segundos (`delay(5000)`). Al finalizar este tiempo, se apaga.
2. El LED naranja se enciende durante 2 segundos, y luego se apaga.
3. El LED rojo se enciende durante 5 segundos. Transcurrido este tiempo, se apaga.



Luego, el ciclo vuelve a comenzar con la luz verde y se repite de forma indefinida.

3. Mejóralo

Prueba a perfeccionar tu semáforo con algunas de estas tres mejoras que te proponemos:

1. **Naranja intermitente:** Modifica la fase intermedia para que el LED naranja no se quede fijo, sino que **parpadee 3 veces rápidas** (encendido 0.3 segundos / apagado 0.3 segundos) antes de pasar al rojo.

Pista: puedes copiar tres veces el encendido y el apagado del LED naranja o, mejor, usar un bucle `for`, que repite las instrucciones de su interior el número de veces que le indiquemos: `for (int i = 0; i < 3; i++) { ... }` las repite 3 veces.

2. **Semáforo de peatones:** Usa el **LED RGB** de la placa como semáforo de peatones: debe estar en **rojo** mientras los coches tienen el verde o el naranja, y en **verde** cuando los coches tienen el rojo.

Pista: el LED RGB tiene un LED rojo (pin 9) y uno verde (pin 5) que se encienden y se apagan con `digitalWrite`, como los demás. Declara una constante para cada uno y configúralos como salidas.

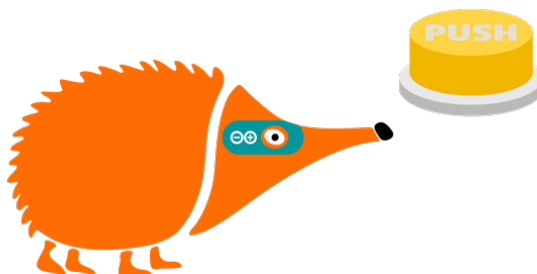
3. **Semáforo sonoro adaptado:** Añade el **zumbador** de la placa para avisar a personas con discapacidad visual:

- **Fase verde:** Sonido intermitente lento.
- **Fase roja:** Silencio.

Pista: el zumbador está en el pin 10. `analogWrite(zumbador, 125)` lo hace sonar y `analogWrite(zumbador, 0)` lo apaga. Para el sonido intermitente, usa un bucle `for` que repita 5 veces: sonar 0,5 segundos y callar 0,5 segundos ($5 \times 1 \text{ s} = \text{los 5 segundos de la fase verde}$).



2.3 Interruptor de luz



1. Qué vamos a hacer

Vamos a programar un sistema de encendido y apagado manual con dos botones:

- Al presionar el **pulsador derecho (SR)**, el **LED rojo** se encenderá.
- Al presionar el **pulsador izquierdo (SL)**, el **LED rojo** se apagará.

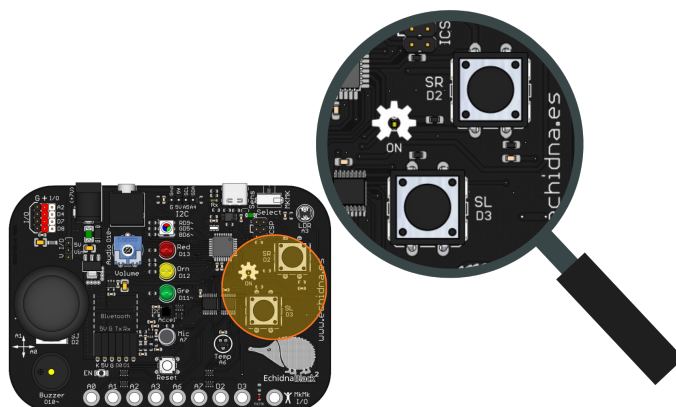
1.1 Qué vamos a aprender

- A leer **entradas digitales** (saber si un botón está presionado o no).
- A diferenciar claramente entre una **Entrada (Input)** y una **Salida (Output)** en robótica:
 - **Entrada (SR y SL):** Detecta la orden del usuario.
 - **Salida (LED rojo):** Produce la respuesta (luz).
- A tomar decisiones en el programa mediante la estructura condicional **if ... else** (si ... si no): si se cumple la condición se ejecuta una parte del programa y, si no, la otra.
- A utilizar **condicionales anidados** (un **if** dentro de otro).
- A guardar valores que cambian en **variables**.
- A controlar el estado de un actuador (LED) mediante eventos físicos (pulsaciones).

1.2 Qué vamos a usar

Componentes

- **Pulsador SR (Switch Right / Derecho, pin 2):** Para encender el LED.
- **Pulsador SL (Switch Left / Izquierdo, pin 3):** Para apagar el LED.
- **LED rojo (pin 13):** Componente que cambia de estado según el botón pulsado.





Programación

Los pines de los pulsadores son **entradas**, así que en `setup()` los configuramos con `pinMode(pin, INPUT)`. Para leer el estado de un pulsador usamos la función `digitalRead(pin)`:

- **pin**: el número del pin del pulsador (SR, el 2; SL, el 3).
- Devuelve HIGH si el pulsador está **pulsado** y LOW si **no** lo está.

2. Programamos

El programa comprueba si el pulsador derecho está presionado; en ese caso, enciende el LED rojo. Si no lo está y presionamos el pulsador izquierdo, el LED se apaga.

Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Interruptor de luz: SR enciende el LED rojo y SL lo apaga
2
3 const int pulsadorSR = 2; // pulsador derecho conectado al pin 2
4 const int pulsadorSL = 3; // pulsador izquierdo conectado al pin 3
5 const int ledRojo = 13;   // LED rojo conectado al pin 13
6
7 int estadoSR = LOW; // guarda si SR está pulsado (HIGH) o no (LOW)
8 int estadoSL = LOW; // guarda si SL está pulsado (HIGH) o no (LOW)
9
10 void setup() {
11     pinMode(pulsadorSR, INPUT); // los pulsadores son entradas
12     pinMode(pulsadorSL, INPUT);
13     pinMode(ledRojo, OUTPUT);    // el LED es una salida
14 }
15
16 void loop() {
17     // lee el estado de los dos pulsadores
18     estadoSR = digitalRead(pulsadorSR);
19     estadoSL = digitalRead(pulsadorSL);
20
21     if (estadoSR == HIGH) {
22         digitalWrite(ledRojo, HIGH); // SR pulsado: enciende el LED
23     } else {
24         if (estadoSL == HIGH) {
25             digitalWrite(ledRojo, LOW); // SL pulsado: apaga el LED
26         }
27     }
28 }
```

Cómo funciona:



Las constantes (líneas 3 a 5): guardan los pines de los dos pulsadores y del LED rojo.

Las variables (líneas 7 y 8): estadoSR y estadoSL guardan lo que leemos en cada pulsador. A diferencia de las constantes, su valor **cambia** mientras el programa funciona: HIGH si el pulsador está pulsado y LOW si no lo está.

La función setup() (líneas 10 a 14): configura los pines de los pulsadores como **entradas** (INPUT), porque la placa lee por ellos, y el del LED como **salida** (OUTPUT).

La función loop() (líneas 16 a 28): primero lee los dos pulsadores con digitalRead y guarda el resultado en las variables. Después decide qué hacer con if ... else. Entre los paréntesis del if va la **condición**: estadoSR == HIGH se cumple si SR está pulsado (con == comparamos si dos valores son iguales). El programa revisa continuamente:

SI el pulsador derecho (SR) está presionado:

--> Enciende el LED rojo inmediatamente.

SI NO (es decir, si SR no está presionado el programa comprueba la segunda condición):

SI el pulsador izquierdo (SL) está presionado:

--> Apaga el LED rojo.

Si no pulsamos ninguno, el programa no cambia el LED, que se queda como estaba.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Luz cruzada (biestable):** Añade el **LED verde** al programa para que funcionen de forma alterna:

- Al pulsar **SR**: LED rojo encendido y LED verde apagado.
- Al pulsar **SL**: LED rojo apagado y LED verde encendido.

Pista: declara una constante para el LED verde (pin 11), configúralo como salida y, dentro de cada if, cambia los dos LED.

2. **Los dos a la vez:** Haz que, si pulsas **SR y SL a la vez**, se encienda el **LED naranja**, y que se apague en cuanto sueltes alguno de los dos.

Pista: para que se cumplan dos condiciones a la vez se usa && («y»): if (estadoSR == HIGH && estadoSL == HIGH). Usa un else para apagar el LED naranja cuando no se cumpla.

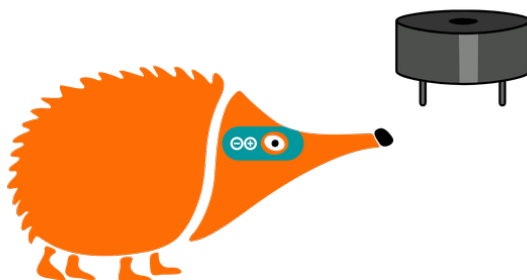
3. **Pulsador con memoria (conmutador):** Programa un solo pulsador (por ejemplo, **SL**) para que funcione como el interruptor de la luz de tu habitación: la primera vez que lo pulsas enciende el LED, y al volverlo a pulsar lo apaga.



Pista: necesitas una variable que recuerde si el LED está encendido o apagado, y esperar a que sueltes el pulsador antes de seguir con `while`, que repite lo que tiene dentro mientras se cumpla la condición.



2.4 Timbre



1. Qué vamos a hacer

Vamos a programar un timbre eléctrico: el **zumbador** emitirá un sonido únicamente mientras mantengamos presionado el **pulsador izquierdo (SL)**, y se callará en cuanto lo soltemos.

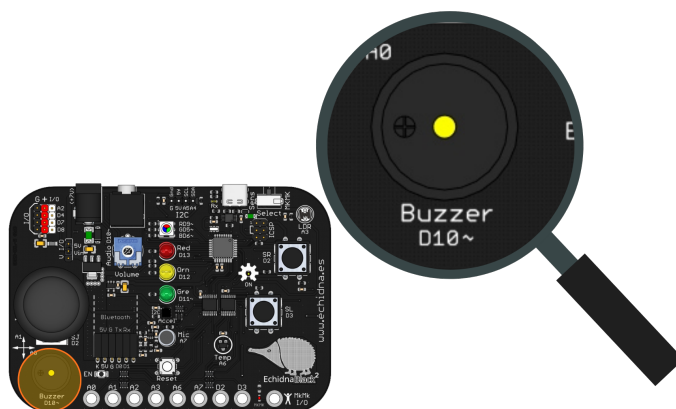
1.1 Qué vamos a aprender

- A controlar un actuador de sonido (**zumbador**) mediante un pulsador.
- A evaluar estados en tiempo real (pulsado o sin pulsar).
- A usar el condicional **if ... else** para que el zumbador suene solo mientras el pulsador está presionado.
- A **modular** el valor de una salida mediante **PWM** con la función `analogWrite`.

1.2 Qué vamos a usar

Componentes

- **Pulsador SL (Switch Left / Izquierdo, pin 3)**: Componente de entrada para activar el sonido.
- **Zumbador pasivo (Buzzer, pin 10)**: Componente de salida que genera pitidos.



Programación

Para hacer sonar el zumbador usamos la función `analogWrite(pin, valor)`:

- **pin**: el número del pin del zumbador, el 10.
- **valor**: un número entre 0 y 255. Con 125 el zumbador suena y con 0 se calla.



El zumbador de la placa es **pasivo**: no suena con una salida digital (`digitalWrite`), porque necesita una señal que se encienda y se apague muy rápido para hacerlo vibrar. Esa señal es la **PWM** (modulación por ancho de pulso), y por eso el zumbador está en un pin de tipo **Digital/Analógico** (ver la tabla de pines de la [Introducción](#)) y lo controlamos con `analogWrite`. El volumen se ajusta con el potenciómetro **Volume** de la placa.

2. Programamos

El programa revisa continuamente si el pulsador SL está presionado: si lo está, el zumbador suena y, si no, se calla.

Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Timbre: el zumbador suena mientras pulsamos SL
2
3 const int pulsadorSL = 3; // pulsador izquierdo conectado al pin 3
4 const int zumbador = 10;  // zumbador conectado al pin 10
5
6 int estadoSL = LOW; // guarda si SL está pulsado (HIGH) o no (LOW)
7
8 void setup() {
9     pinMode(pulsadorSL, INPUT); // el pulsador es una entrada
10    pinMode(zumbador, OUTPUT);  // el zumbador es una salida
11 }
12
13 void loop() {
14     // lee el estado del pulsador
15     estadoSL = digitalRead(pulsadorSL);
16
17     if (estadoSL == HIGH) {
18         analogWrite(zumbador, 125); // SL pulsado: el zumbador suena
19     } else {
20         analogWrite(zumbador, 0);   // SL sin pulsar: el zumbador se calla
21     }
22 }
```

Cómo funciona:

Las constantes (líneas 3 y 4): guardan los pines del pulsador SL y del zumbador.

La variable (línea 6): `estadoSL` guarda lo que leemos en el pulsador: **HIGH** si está pulsado y **LOW** si no lo está.

La función `setup()` (líneas 8 a 11): configura el pin del pulsador como **entrada** (**INPUT**) y el del zumbador como **salida** (**OUTPUT**).



La función `loop()` (líneas 13 a 22): lee el pulsador con `digitalRead` y guarda el resultado en `estadoSL`. Después, con `if ... else`, decide qué hacer. A diferencia del proyecto anterior, aquí **las dos ramas** hacen algo: una enciende el zumbador y la otra lo apaga. El programa revisa continuamente:

```
SI el pulsador SL está presionado:
    --> El zumbador suena.

SI NO (es decir, si SL está sin pulsar):
    --> El zumbador se calla.
```

De esta forma, el zumbador solo suena mientras mantenemos el pulsador presionado.

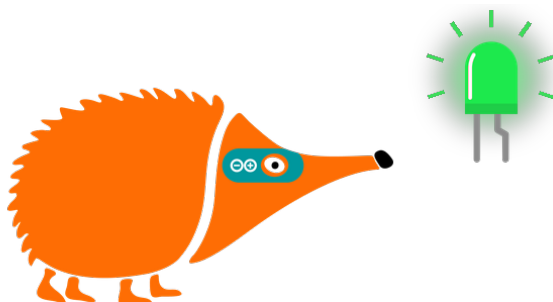
3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **LED testigo:** Añade dos LED que indiquen el estado del timbre: el **LED verde** encendido mientras el zumbador suena y el **LED rojo** encendido mientras está en silencio.
Pista: declara las constantes del LED rojo (pin 13) y del LED verde (pin 11), configúralos como salidas y, en cada rama del `if ... else`, cambia los dos LED con `digitalWrite`.
2. **Alarma intermitente:** Modifica el programa para que, al mantener pulsado **SL**, el sonido no sea continuo, sino que emita pitidos intermitentes tipo alarma: 100 milisegundos de sonido y 100 milisegundos de silencio.
Pista: dentro del `if`, haz sonar el zumbador, espera con `delay(100)`, apágalo y vuelve a esperar con `delay(100)`.
3. **Timbre de dos tonos:** Programa el pulsador **SR** para que emita un tono más **agudo** que el pulsador **SL**. ¡Intenta combinar pulsaciones cortas y largas para enviar mensajes secretos en código Morse a tus compañeros!
Pista: con `analogWrite` el zumbador siempre suena con el mismo tono. Para elegir el tono usa la función `tone(pin, frecuencia)`, donde la **frecuencia** se indica en hercios (Hz): cuanto mayor es, más agudo es el sonido (por ejemplo, 440 para SL y 880 para SR). Para que deje de sonar, usa `noTone(pin)`.

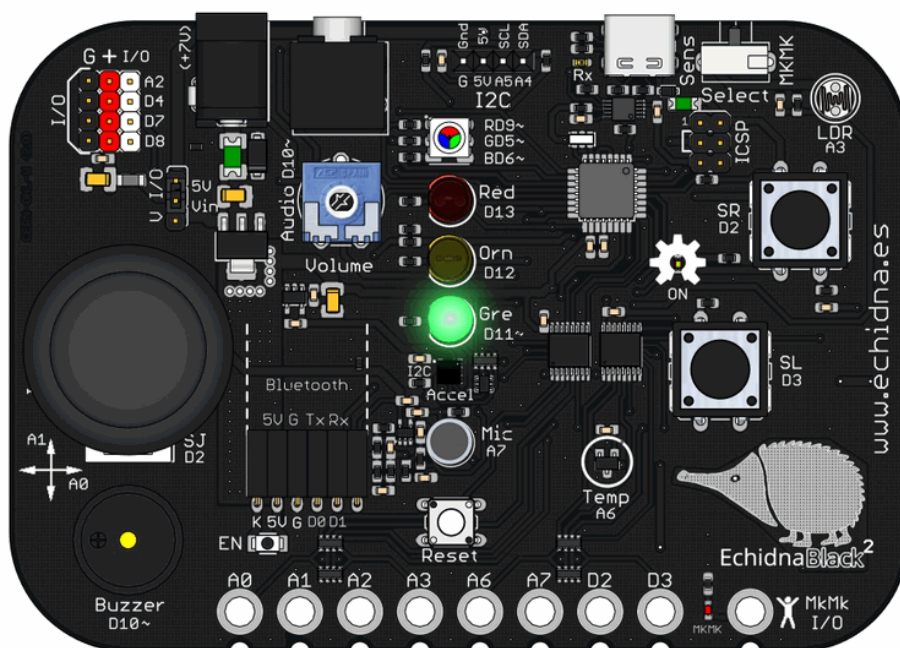


2.5 Brillo LED



1. Qué vamos a hacer

Vamos a hacer que el **LED verde** de la placa se encienda **poco a poco**, hasta brillar al máximo, y que después se apague también poco a poco, como la luz de un cine al empezar y terminar la película.



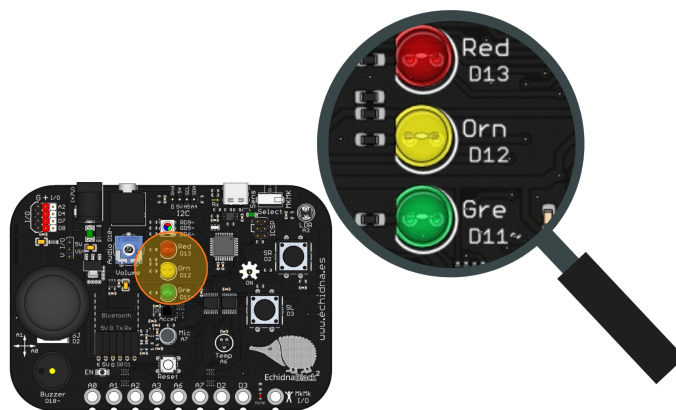
1.1 Qué vamos a aprender

- A regular el **brillo** de un LED con valores de 0 a 255 mediante **PWM** con `analogWrite`.
- A repetir instrucciones un número de veces con el bucle **for**, contando hacia arriba y hacia abajo.

1.2 Qué vamos a usar

Componentes

- **LED verde (pin 11):** Se enciende y se apaga poco a poco.



Programación

En el proyecto [Timbre](#) usamos `analogWrite` para hacer sonar el zumbador. El pin del LED verde también es de tipo **Digital/Analógico**: en la placa lleva el símbolo ~ (Gre D11~; ver la tabla de pines de la [Introducción](#)). Así que, además de encenderlo y apagarlo con `digitalWrite`, con `analogWrite` podemos darle un **brillo** entre 0 (apagado) y 255 (brillo máximo); con 128, por ejemplo, luce a media potencia. La señal **PWM** enciende y apaga el LED tan deprisa que no vemos el parpadeo: cuanto más tiempo está encendido en cada parpadeo, más brillante lo vemos. Los pines del LED rojo (D13) y del naranja (D12) no admiten PWM, por eso usamos el verde.

Para que el brillo cambie poco a poco tendríamos que escribir 256 `analogWrite` seguidos, uno por cada valor. En lugar de eso, usamos el bucle `for`, que repite las instrucciones que tiene entre llaves { }:

```
for (int brillo = 0; brillo <= 255; brillo = brillo + 1) {  
  // instrucciones que se repiten  
}
```

Entre los paréntesis tiene tres partes, separadas por punto y coma:

- **Inicio** (`int brillo = 0`): crea la variable que cuenta las vueltas y le da su primer valor.
- **Condición** (`brillo <= 255`): el bucle se repite mientras se cumpla. El operador `<=` significa «menor o igual que».
- **Incremento** (`brillo = brillo + 1`): cómo cambia la variable al final de cada vuelta; aquí, suma 1.

2. Programamos

El programa tiene dos bucles `for`, uno detrás de otro: el primero sube el brillo del LED verde de 0 a 255 y el segundo lo baja de 255 a 0. Al terminar, el LED se queda apagado un segundo y todo vuelve a empezar.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Brillo LED: el LED verde se enciende y se apaga poco a poco
2
3 const int ledVerde = 11; // LED verde conectado al pin 11
4
5 const int espera = 10; // tiempo entre un brillo y el siguiente, en milisegundos
6
7 void setup() {
8   pinMode(ledVerde, OUTPUT); // el LED es una salida
9 }
10
11 void loop() {
12   // encendido gradual: el brillo sube de 0 a 255
13   for (int brillo = 0; brillo <= 255; brillo = brillo + 1) {
14     analogWrite(ledVerde, brillo);
15     delay(espera);
16   }
17
18   // apagado gradual: el brillo baja de 255 a 0
19   for (int brillo = 255; brillo >= 0; brillo = brillo - 1) {
20     analogWrite(ledVerde, brillo);
21     delay(espera);
22   }
23
24   delay(1000); // espera 1 segundo con el LED apagado
25 }
```

Cómo funciona:

Las constantes (líneas 3 a 5): guardan el pin del LED verde y la **espera**, el tiempo que pasa entre un brillo y el siguiente.

La función setup() (líneas 7 a 9): configura el pin del LED como **salida**.

La función loop() (líneas 11 a 25): tiene dos bucles for seguidos.

- **El primer bucle** (líneas 13 a 16) crea la variable **brillo** con el valor 0 y, en cada vuelta, enciende el LED con ese brillo, espera 10 milisegundos y suma 1. Se repite mientras **brillo** sea menor o igual que 255, así que da 256 vueltas y el LED pasa de apagado a brillo máximo en unos 2,5 segundos.
- **El segundo bucle** (líneas 19 a 22) hace lo contrario: empieza en 255 y, en cada vuelta, **resta** 1 (**brillo = brillo - 1**). Se repite mientras **brillo** sea mayor o igual que 0 (el operador **>=** significa «mayor o igual que»), así que el LED se va apagando poco a poco.



Al final, `delay(1000)` deja el LED apagado un segundo antes de que `loop()` vuelva a empezar:

```
PARA cada brillo de 0 a 255:  
  --> Enciende el LED verde con ese brillo.  
  --> Espera 10 milisegundos.
```

```
PARA cada brillo de 255 a 0:  
  --> Enciende el LED verde con ese brillo.  
  --> Espera 10 milisegundos.
```

```
Espera 1 segundo con el LED apagado.
```

La variable `brillo` se crea dentro de cada `for` y solo existe mientras dura ese bucle, por eso podemos usar el mismo nombre en los dos.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Respiración lenta:** Haz que el LED se encienda y se apague mucho más despacio, como si respirara tranquilamente. ¿Y si lo quieres muy rápido?

Pista: la constante espera marca cuánto dura cada uno de los 256 pasos. Con 30 milisegundos, cada subida dura unos 7,7 segundos; con 2, apenas medio segundo.

2. **Latido:** Haz que el LED lata como un corazón: que se encienda **deprisa** y se apague **despacio**, con una pausa corta entre un latido y el siguiente.

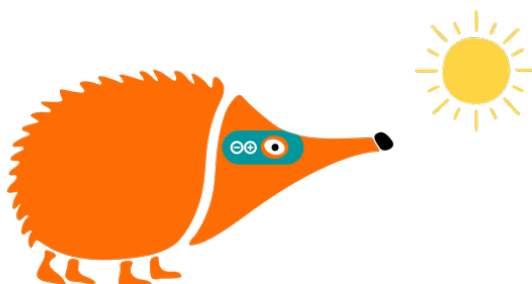
Pista: usa dos constantes en lugar de espera, una para la subida (por ejemplo, 2) y otra para la bajada (por ejemplo, 8), y pon cada una en el `delay` de su bucle. Acorta también la pausa del final, por ejemplo a medio segundo.

3. **Regulador de brillo:** Usa los pulsadores como el regulador de luz de una lámpara: mientras mantienes pulsado **SR**, el LED verde brilla cada vez más, y mientras mantienes pulsado **SL**, brilla cada vez menos.

Pista: ahora no hace falta un `for`: declara una variable global `brillo` y, en cada vuelta de `loop()`, súmale 1 si **SR** está pulsado y réstale 1 si lo está **SL**. Para que no se salga de los valores de `analogWrite`, súmale solo si es menor que 255 y réstale solo si es mayor que 0, con un `if` dentro de otro, como en el [Interruptor de luz](#).



2.6 Interruptor crepuscular



1. Qué vamos a hacer

Vamos a programar un sistema automático similar al de las farolas de la calle: el **LED verde** se encenderá automáticamente cuando la luz ambiental baje (de noche) y se apagará cuando haya suficiente luz (de día).

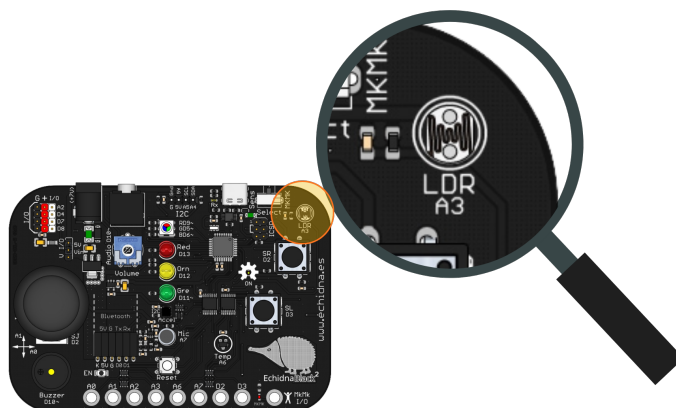
1.1 Qué vamos a aprender

- A programar un **sistema automático** que reaccione al entorno.
- A utilizar el **sensor de luz (LDR)** para medir la iluminación ambiental.
- A leer **entradas analógicas**, que dan valores intermedios y no solo HIGH o LOW.
- A usar un **operador de comparación** (<) para que el condicional `if ... else` decida a partir de la lectura de un sensor.
- A trabajar con **umbrales numéricos** para definir estados (día/noche).

1.2 Qué vamos a usar

Componentes

- **Sensor de luz (LDR, pin A3):** Mide la cantidad de luz que recibe. Cuanto más oscuro esté el entorno, menor será el valor que leemos.
- **LED verde (pin 11):** Funcionará como nuestra luz automática.





Programación

El sensor de luz está en un pin de tipo **Analógico** (ver la tabla de pines de la [Introducción](#)). Para leerlo usamos la función `analogRead(pin)`:

- **pin**: el pin analógico del sensor, el A3.
- Devuelve un número entre **0** (no hay luz) y **1023** (muchísima luz).

Los pines analógicos son **entradas** por defecto, así que no hace falta configurarlos con `pinMode`.

2. Programamos

El programa revisa continuamente el valor del sensor de luz: si es menor que un cierto **umbral**, enciende el LED; en caso contrario, lo apaga.

Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Interruptor crepuscular: el LED verde se enciende cuando hay poca luz
2
3 const int sensorLuz = A3; // sensor de luz (LDR) conectado al pin A3
4 const int ledVerde = 11; // LED verde conectado al pin 11
5
6 const int umbral = 200; // por debajo de este valor consideramos que es de noche
7
8 int valorLuz = 0; // guarda la luz que mide el sensor (de 0 a 1023)
9
10 void setup() {
11     pinMode(ledVerde, OUTPUT); // el LED es una salida
12 }
13
14 void loop() {
15     // lee la cantidad de luz
16     valorLuz = analogRead(sensorLuz);
17
18     if (valorLuz < umbral) {
19         digitalWrite(ledVerde, HIGH); // poca luz (de noche): enciende el LED
20     } else {
21         digitalWrite(ledVerde, LOW); // mucha luz (de día): apaga el LED
22     }
23 }
```

Cómo funciona:

Las constantes (líneas 3 a 6): guardan los pines del sensor de luz y del LED verde, y el **umbral**, el valor que separa el día de la noche. Al tenerlo en una constante, si queremos ajustarlo solo hay que cambiar un número.

La variable (línea 8): `valorLuz` guarda la luz que mide el sensor, un número entre 0 y 1023.



La función `setup()` (líneas 10 a 12): configura el pin del LED como **salida** (OUTPUT). El sensor no necesita `pinMode`, porque está en un pin analógico.

La función `loop()` (líneas 14 a 23): lee el sensor con `analogRead` y guarda el resultado en `valorLuz`. Después, con `if ... else`, lo compara con el umbral. El operador `<` («menor que») comprueba si `valorLuz` es más pequeño que `umbral`. El programa revisa continuamente:

```
SI el sensor de luz registra valores menores de 200:  
  --> Se enciende el LED verde.
```

```
SI NO (si registra valores mayores o iguales a 200):  
  --> Se apaga el LED verde.
```

El valor 200 actúa como el umbral que define cuándo debe encenderse o apagarse la luz.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Lee el sensor y calibra tu aula:** Averigua qué valores lee el sensor de luz en tu mesa, con la mano encima y sin ella, y ajusta el umbral para que la luz se encienda solo cuando tapes el sensor con la mano.

Pista: para ver el valor del sensor en el ordenador, usa el **Monitor serie**. En `setup()`, abre la comunicación con `Serial.begin(9600)`; en `loop()`, escribe un texto con `Serial.print("texto")` y el valor con `Serial.println(valorLuz)`, que además salta a la línea siguiente. Después de cargar el programa, abre el Monitor serie (menú **Herramientas > Monitor Serie**) con la velocidad en **9600 baudios**. Elige un umbral entre el valor con la mano encima y el valor sin ella.

2. **Luz blanca:** Sustituye el LED verde por el **LED RGB** para que ilumine más: cuando haya oscuridad, se encenderá en color **blanco**, y cuando haya mucha luz, se apagará.

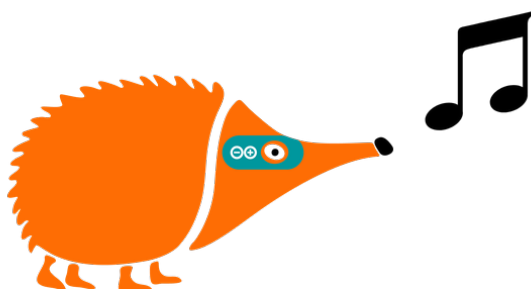
Pista: el LED RGB tiene tres colores, cada uno en su pin: rojo en el 9, verde en el 5 y azul en el 6. Si encendemos los tres a la vez con `digitalWrite`, se mezclan y vemos luz **blanca**.

3. **Medidor de luz:** Convierte la placa en un medidor de luz: cuanta más luz haya, más LED se encenderán. Con muy poca luz, ninguno; con algo de luz, el **verde**; con bastante, el **verde** y el **naranja**; y con mucha luz, los tres (**verde, naranja y rojo**).

Pista: necesitas varios umbrales (por ejemplo, 200, 500 y 800). Para comprobar varias condiciones una detrás de otra, usa `else if` («si no, si»): el programa comprueba cada condición en orden y ejecuta solo la primera que se cumple.



2.7 Piano de frutas



1. Qué vamos a hacer

Vamos a convertir la entrada **MkMk A0** de nuestra placa en una tecla de piano táctil. Cada vez que toques la entrada (o un objeto conductor conectado a ella, como una fruta), el **zumbador** de la placa tocará una nota musical.

1.1 Qué vamos a aprender

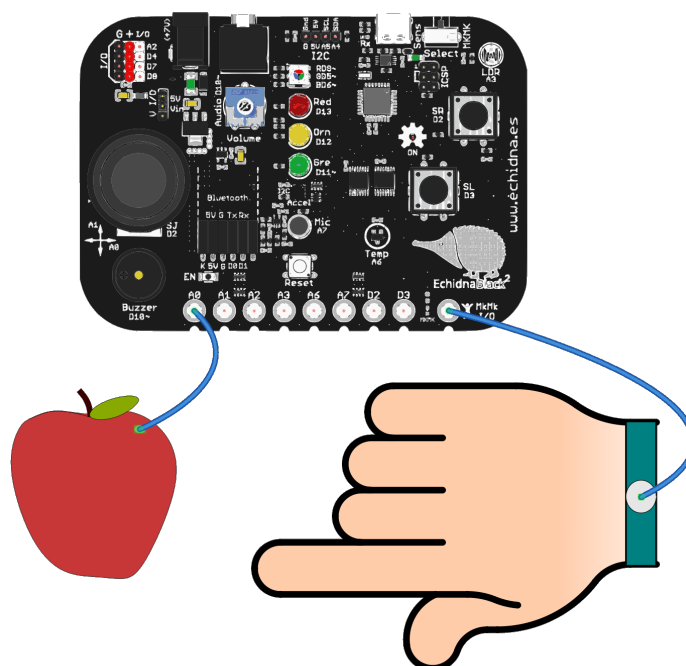
- A detectar la conductividad eléctrica de objetos cotidianos usando el **modo Makey Makey (MkMk)**.
- A conectar cables de cocodrilo en los conectores externos de la placa.
- A entender cómo funciona un **circuito cerrado** a través de nuestro propio cuerpo.
- A tocar **notas musicales** con el zumbador, indicando su **frecuencia** y su **duración**.

1.2 Qué vamos a usar

Componentes

- **Entrada MkMk A0 (pin A0):** Conector táctil para detectar contacto o conductividad.
- **Conector común MkMk I/O (5V):** Indispensable para cerrar el circuito con tu cuerpo. Está en el extremo derecho de la fila de conectores y proporciona 5V.
- **Cables de cocodrilo:** Para conectar objetos externos (frutas, plastilina, papel de aluminio, etc.).
- **Zumbador pasivo (pin 10):** Toca la nota musical.

¡ATENCIÓN! Para que funcione el modo MkMk debemos poner el selector del modo de funcionamiento hacia la derecha, y se nos encenderá el LED testigo en la parte inferior.



En la imagen, la fruta está conectada a la entrada **A0** y la pulsera al conector común **Mkmk I/O** (5V). Al tocar la fruta con la mano, una corriente muy pequeña pasa a través de tu cuerpo, el circuito se cierra y la placa lo detecta en la entrada A0.

Programación

La entrada A0 es **analógica** (ver la tabla de pines de la [Introducción](#)), así que la leemos con `analogRead`, como el sensor de luz. Si el valor supera **350** (en una escala de 0 a 1023), el circuito está cerrado: hay contacto.

Para tocar la nota usamos la función `tone(pin, frecuencia, duración)`:

- **pin**: el número del pin del zumbador, el 10.
- **frecuencia**: la nota que suena, en hercios (Hz). Cuanto mayor es, más aguda es la nota. El **Do central** del piano son 262 Hz.
- **duración**: el tiempo que suena la nota, en milisegundos. Pasado ese tiempo, el zumbador se calla solo.

2. Programamos

El programa revisa continuamente la entrada Mkmk A0: **cuando detecta contacto**, el zumbador toca la nota **Do central** durante **250 milisegundos** (un cuarto de segundo).



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Piano de frutas: al tocar la fruta conectada a A0 suena la nota Do
2
3 const int entradaMkMk = A0; // entrada MkMk conectada al pin A0
4 const int zumbador = 10;    // zumbador conectado al pin 10
5
6 const int umbral = 350;     // por encima de este valor hay contacto
7 const int notaDo = 262;     // frecuencia de la nota Do central, en Hz
8 const int duracion = 250;   // duración de la nota, en milisegundos
9
10 int valorMkMk = 0; // guarda el valor que lee la entrada (de 0 a 1023)
11
12 void setup() {
13     pinMode(zumbador, OUTPUT); // el zumbador es una salida
14 }
15
16 void loop() {
17     // lee la entrada MkMk
18     valorMkMk = analogRead(entradaMkMk);
19
20     if (valorMkMk > umbral) {
21         tone(zumbador, notaDo, duracion); // hay contacto: suena la nota Do
22         delay(duracion);                  // espera a que termine la nota
23     }
24 }
```

Cómo funciona:

Las constantes (líneas 3 a 8): guardan los pines de la entrada MkMk y del zumbador, el **umbral** que indica si hay contacto, y la **frecuencia** y la **duración** de la nota.

La variable (línea 10): `valorMkMk` guarda el valor que leemos en la entrada A0, un número entre 0 y 1023.

La función `setup()` (líneas 12 a 14): configura el pin del zumbador como **salida** (OUTPUT). La entrada A0 no necesita `pinMode`, porque es un pin analógico.

La función `loop()` (líneas 16 a 25): lee la entrada con `analogRead` y la compara con el umbral. El operador `>` («mayor que») comprueba si `valorMkMk` es más grande que `umbral`. Si hay contacto, `tone` toca la nota Do y `delay` espera a que termine antes de volver a leer la entrada. El programa revisa continuamente:

SI la entrada MkMk A0 detecta contacto (valor mayor que 350):
--> Suena la nota Do central durante 250 milisegundos.



Aquí no hay `else`: si no hay contacto, no ocurre nada y el programa vuelve a comprobar la entrada. Si mantienes la mano en la fruta, la nota se repite una y otra vez.

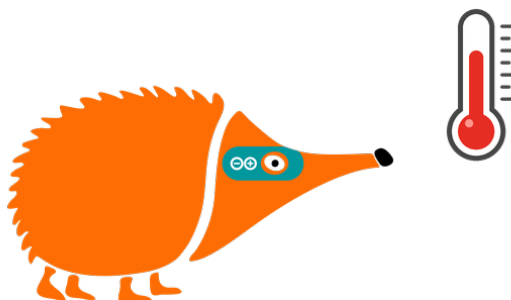
3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Luz y sonido:** Haz que el **LED verde** se encienda mientras suena la nota y se apague al terminar.
Pista: declara la constante del LED verde (pin 11) y configúralo como salida. Dentro del `if`, enciéndelo con `digitalWrite` antes de `tone` y apágalo después del `delay`.
2. **Escala musical:** Conecta más frutas a las entradas **A1, A2 y A3** con otros cables de cocodrilo y crea un mini teclado de cuatro teclas: **Do** (262 Hz), **Re** (294 Hz), **Mi** (330 Hz) y **Fa** (349 Hz).
Pista: declara una constante para cada entrada y para cada nota, y escribe un `if` por tecla. Puedes poner `analogRead` directamente dentro de la condición: `if (analogRead(teclaRe) > umbral)`.
3. **Teclado de colores:** Añade a tu escala el **LED RGB** para que cada tecla lo encienda de un color distinto: **Do** en rojo, **Re** en verde, **Mi** en azul y **Fa** en blanco.
Pista: recuerda que el LED RGB tiene el rojo en el pin 9, el verde en el 5 y el azul en el 6, y que con los tres encendidos se ve blanco. En cada `if`, enciende el color antes de tocar la nota y, al final de `loop()`, apaga los tres colores.



2.8 El echidna dice la temperatura



1. Qué vamos a hacer

Vamos a convertir a nuestra placa en un **hombre del tiempo**: cada segundo medirá la **temperatura** con su sensor y nos la dirá en el **Monitor serie** de Arduino IDE con una frase como «Hola, ahora hace una temperatura de 23.75 °C».

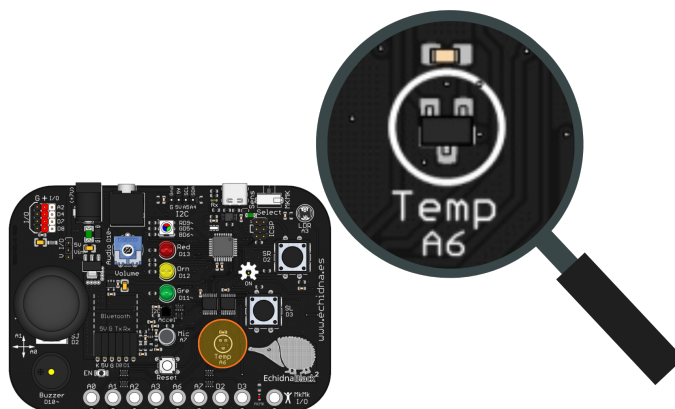
1.1 Qué vamos a aprender

- A utilizar el **sensor de temperatura** para medir la temperatura ambiente en **grados Celsius** (°C).
- A convertir la lectura de un sensor en una magnitud real mediante una **fórmula**.
- A guardar números con **decimales** en variables de tipo float.
- A enviar mensajes de la placa al ordenador y verlos en el **Monitor serie**.

1.2 Qué vamos a usar

Componentes

- **Sensor de temperatura (pin A6)**: Mide la temperatura del ambiente. Entrega un voltaje que depende de la temperatura.





Programación

El sensor está en un pin **analógico** (ver la tabla de pines de la [Introducción](#)), así que lo leemos con `analogRead`, que nos da un número entre 0 y 1023. Para pasar ese número a grados Celsius usamos esta fórmula, ajustada a la EchidnaBlack2:

```
temperatura = (lectura × 0.4658) - 50
```

Para que la placa nos diga la temperatura, usamos el **Monitor serie**: una ventana de Arduino IDE donde vemos los mensajes que la placa envía al ordenador por el cable USB. Si hiciste la primera mejora del [Interrupción crepuscular](#), ya lo has usado; aquí lo vemos con detalle. Para enviar los mensajes usamos la función `Serial.print(dato)`:

- **dato**: lo que queremos mostrar. Puede ser un **texto**, entre comillas ("Hola"), o el valor de una **variable** (temperatura).
- `Serial.println(dato)` hace lo mismo, pero después **salta a la línea siguiente**, así cada mensaje empieza en una línea nueva.

Antes de enviar mensajes hay que abrir la comunicación en `setup()` con `Serial.begin(9600)`, donde 9600 es la **velocidad** en baudios.

2. Programamos

El programa lee el sensor de temperatura, convierte la lectura a grados Celsius y escribe en el Monitor serie una frase formada por tres partes: el texto «Hola, ahora hace una temperatura de », el valor de la temperatura y el símbolo «°C». Después espera un segundo y vuelve a empezar.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Echidna dice la temperatura: muestra la temperatura en el Monitor serie
2
3 const int sensorTemperatura = A6; // sensor de temperatura conectado al pin A6
4
5 int lectura = 0; // guarda el valor que lee el sensor (de 0 a 1023)
6 float temperatura = 0.0; // guarda la temperatura en grados Celsius, con
decimales
7
8 void setup() {
9   Serial.begin(9600); // abre la comunicación con el ordenador a 9600 baudios
10 }
11
12 void loop() {
13   // lee el sensor y convierte el valor a grados Celsius
14   lectura = analogRead(sensorTemperatura);
15   temperatura = (lectura * 0.4658) - 50.0; // conversión ajustada a
EchidnaBlack2
16
17   // escribe la frase en el Monitor serie
18   Serial.print("Hola, ahora hace una temperatura de ");
19   Serial.print(temperatura);
20   Serial.println(" °C");
21
22   delay(1000); // espera 1 segundo hasta la siguiente medida
23 }
```

Después de cargar el programa, abre el **Monitor serie** desde el menú **Herramientas > Monitor Serie** y comprueba que la velocidad está en **9600 baudios**. Verás aparecer una frase nueva cada segundo.



Cómo funciona:

La constante (línea 3): guarda el pin del sensor de temperatura.



Las variables (líneas 5 y 6): lectura guarda el número que da el sensor, entre 0 y 1023; es de tipo `int`, que solo admite números **enteros**. temperatura guarda los grados Celsius y es de tipo **float**, que admite números con **decimales** (por ejemplo, 23.75). En el código, los decimales se escriben con **punto**, no con coma.

La función `setup()` (líneas 8 a 10): abre la comunicación con el ordenador con `Serial.begin(9600)`. El sensor no necesita `pinMode`, porque está en un pin analógico.

La función `loop()` (líneas 12 a 23): lee el sensor con `analogRead` y aplica la fórmula para obtener la temperatura. En el código, el signo de multiplicar es el asterisco (*). Después escribe la frase en tres partes: los dos primeros `Serial.print` escriben el texto y el valor seguidos, en la misma línea, y el `Serial.println` final añade « °C» y salta de línea. El programa repite continuamente:

```
Lee la temperatura del sensor.  
--> Escribe en el Monitor serie la frase con la temperatura.  
--> Espera 1 segundo.
```

Para probarlo, calienta el sensor tocándolo suavemente con el dedo y observa cómo sube la temperatura.

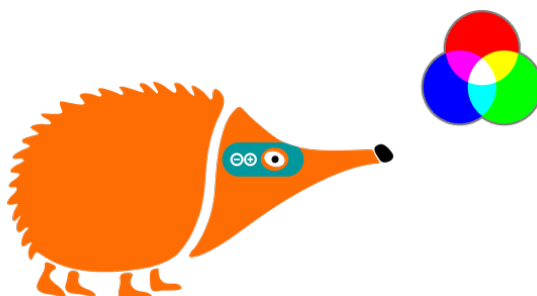
3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Como la tecla t:** En EchidnaML, el echidna dice la temperatura al pulsar la tecla **t**. Haz que la placa diga la temperatura solo cuando pulses el **pulsador SR**.
Pista: declara la constante del pulsador SR (pin 2) y configúralo como entrada. En `loop()`, mete todo lo que había dentro de un `if` que compruebe si SR está pulsado: `if (digitalRead(pulsadorSR) == HIGH)`.
2. **Grados Fahrenheit:** Añade el **pulsador SL** para que, al pulsarlo, la placa diga la temperatura en **grados Fahrenheit** (°F). Con SR seguirá diciéndola en grados Celsius.
Pista: para pasar de grados Celsius a Fahrenheit, multiplica por 1.8 y suma 32. Guarda el resultado en una nueva variable `float` llamada `fahrenheit` y añade un segundo `if` para el pulsador SL (pin 3).
3. **Frío o calor:** Partiendo del programa original, haz que el **LED rojo** se encienda cuando la temperatura supere un valor (por ejemplo, 28 °C) y que, si no lo supera, se encienda el **LED verde**. Para probarlo, calienta el sensor tocándolo suavemente con el dedo.
Pista: declara las constantes del LED rojo (pin 13) y del LED verde (pin 11) y configúralos como salidas. Guarda el valor límite en una constante `float` llamada `umbral` y compáralo con la temperatura con `if ... else`, como en el interruptor crepuscular.



2.9 Mezclamos colores



1. Qué vamos a hacer

Vamos a hacer que el **LED RGB** de la placa recorra, uno detrás de otro, un montón de colores distintos mezclando sus tres colores básicos: **rojo**, **verde** y **azul**. A la vez, veremos en el **Monitor serie** la mezcla que forma cada color.

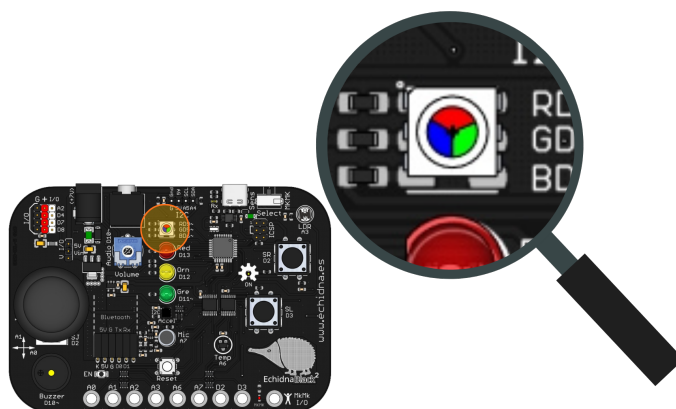
1.1 Qué vamos a aprender

- A utilizar el **LED RGB** y a **mezclar** sus tres colores (rojo, verde y azul) para conseguir otros.
- A ajustar la **intensidad** de cada color con valores de 0 a 255 mediante **PWM** con `analogWrite`, como el brillo del LED verde en [Brillo LED](#).
- A poner un bucle **dentro de otro** (bucles **anidados**) para recorrer todas las combinaciones.

1.2 Qué vamos a usar

Componentes

- **LED RGB (pines 9, 5 y 6)**: Componente que tiene dentro tres LED: uno **rojo** (R, Red, pin 9), uno **verde** (G, Green, pin 5) y uno **azul** (B, Blue, pin 6). Al mezclar la luz de los tres podemos conseguir más de 16 millones de colores.



Programación

Los pines del LED RGB son de tipo **Digital/Analógico** (ver la tabla de pines de la [Introducción](#)), así que con `analogWrite` damos a cada color un valor entre 0 (apagado) y 255 (máxima intensidad), como hicimos con el LED verde en [Brillo LED](#). Por ejemplo, el **naranja Echidna** es mucho rojo (254), algo de verde (109) y casi nada de azul (4).



Ya conoces el bucle `for` del proyecto [Brillo LED](#). Allí la variable sumaba 1 en cada vuelta; aquí sumará `paso`, así que cada color irá de `paso` en `paso`. Para probar **todas las combinaciones** de colores, ponemos un bucle **dentro de otro** (bucles **anidados**): por cada vuelta del bucle de fuera, el de dentro da todas sus vueltas.

```
for (int rojo = 0; rojo <= 255; rojo = rojo + paso) {  
  for (int verde = 0; verde <= 255; verde = verde + paso) {  
    // se repite para cada pareja de rojo y verde  
  }  
}
```

Si cada bucle da 6 vueltas, lo que hay dentro del segundo se repite $6 \times 6 = 36$ veces, una por cada pareja de valores. Nuestro programa usa tres bucles anidados, uno por color.

2. Programamos

El programa usa tres bucles `for`, uno dentro de otro: uno para el rojo, otro para el verde y otro para el azul. Cada color toma los valores 0, 51, 102, 153, 204 y 255, y el programa prueba **todas las combinaciones**: $6 \times 6 \times 6 = 216$ colores. Cada color se ve durante una décima de segundo, así que la vuelta completa dura unos 22 segundos.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Mezclamos colores: recorre los colores del LED RGB con tres bucles for
  anidados
2
3 const int rgbRojo = 9;    // LED RGB: color rojo en el pin 9
4 const int rgbVerde = 5;   // LED RGB: color verde en el pin 5
5 const int rgbAzul = 6;    // LED RGB: color azul en el pin 6
6
7 const int paso = 51;      // cuánto sube cada color en cada vuelta (0, 51, 102,
153, 204, 255)
8 const int espera = 100;   // tiempo que se ve cada color, en milisegundos
9
10 void setup() {
11   pinMode(rgbRojo, OUTPUT); // los tres colores del LED RGB son salidas
12   pinMode(rgbVerde, OUTPUT);
13   pinMode(rgbAzul, OUTPUT);
14   Serial.begin(9600);       // para ver en el Monitor serie la mezcla de cada
  color
15 }
16
17 void loop() {
18   for (int rojo = 0; rojo <= 255; rojo = rojo + paso) {
19     for (int verde = 0; verde <= 255; verde = verde + paso) {
20       for (int azul = 0; azul <= 255; azul = azul + paso) {
21         // mezcla los tres colores
22         analogWrite(rgbRojo, rojo);
23         analogWrite(rgbVerde, verde);
24         analogWrite(rgbAzul, azul);
25
26         // muestra la mezcla en el Monitor serie
27         Serial.print(rojo);
28         Serial.print(" ");
29         Serial.print(verde);
30         Serial.print(" ");
31         Serial.println(azul);
32
33         delay(espera);
34       }
35     }
36   }
37 }
```



Después de cargar el programa, abre el **Monitor serie (Herramientas > Monitor Serie, a 9600 baudios)** para ver los valores de rojo, verde y azul de cada color.

Cómo funciona:

Las constantes (líneas 3 a 8): guardan los pines de los tres colores del LED RGB, el **paso** (cuánto sube cada color de una vuelta a la siguiente) y la **espera** (cuánto tiempo se ve cada color).

La función setup() (líneas 10 a 15): configura los tres pines del LED RGB como **salidas** y abre la comunicación con el ordenador con `Serial.begin(9600)`.

La función loop() (líneas 17 a 37): tiene los tres bucles `for` anidados. Las variables rojo, verde y azul se crean dentro de cada `for` y cuentan los valores de cada color. Dentro del bucle más interior, `analogWrite` enciende cada color con su valor, `Serial.print` escribe la mezcla en el Monitor serie y `delay` deja ver el color.

Los bucles anidados funcionan como el cuentakilómetros de un coche: el bucle de **dentro** (el azul) es el que gira más rápido. Cada vez que el azul termina sus seis valores, el verde sube un paso; y cada vez que el verde termina los suyos, sube el rojo:

```
PARA cada valor del rojo (0, 51, ... 255):  
  PARA cada valor del verde (0, 51, ... 255):  
    PARA cada valor del azul (0, 51, ... 255):  
      --> Enciende el LED RGB con esa mezcla.  
      --> Escribe la mezcla en el Monitor serie.  
      --> Espera 100 milisegundos.
```

Cuando los tres bucles terminan, `loop()` vuelve a empezar y el recorrido de colores se repite.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Más colores:** Cambia las constantes del programa para que el LED RGB recorra **muchos más colores** y comprueba cuánto dura ahora la vuelta completa.

Pista: cuanto más pequeño sea **paso**, más valores toma cada color y más combinaciones hay. Para que cada color llegue justo a 255, elige un paso que divida exactamente a 255, como 85, 51, 17, 15 o 5. Si hay muchos colores, baja también la espera para que la vuelta no dure demasiado.

2. **De rojo a azul:** Haz que el LED RGB pase poco a poco del **rojo** al **azul**, pasando por los morados, y después vuelva del azul al rojo.

Pista: basta con un `for` para cada sentido (no anidados). Mientras el azul sube, el rojo baja: si el azul vale `valor`, el rojo vale `255 - valor`, así que entre los dos siempre suman 255. Usa un paso pequeño (5) y una espera corta (30) para que el cambio sea suave.

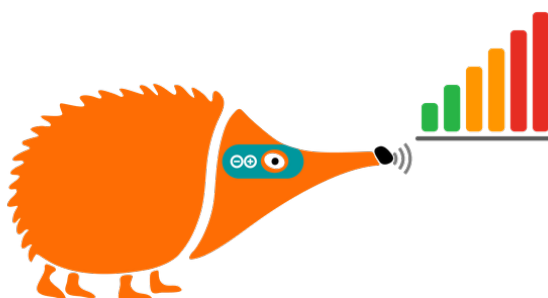


3. **Termómetro de colores:** Usa el **sensor de temperatura** para que el LED RGB cambie de color según la temperatura: **azul** si hace frío (menos de 20 °C), **verde** si la temperatura es agradable y **rojo** si hace calor (más de 28 °C).

Pista: lee la temperatura como en el proyecto [El echidna dice la temperatura](#) y usa `if`, `else if` y `else`, como en el medidor de luz del [Interruptor crepuscular](#). Para cada color, da 255 al que quieres encender y 0 a los otros dos con `analogWrite`.



2.10 Vúmetro



1. Qué vamos a hacer

Vamos a construir un **vúmetro** o **semáforo de ruido**, que muestra con los LED de la placa cuánto ruido hay a nuestro alrededor. Con silencio se encenderá solo el **LED verde**; si el ruido aumenta, se encenderá también el **naranja**; y si hay mucho ruido, se encenderán los **tres LED**.

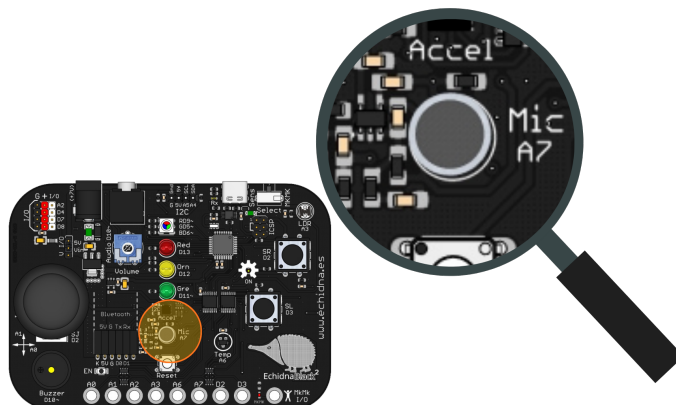
1.1 Qué vamos a aprender

- A utilizar el **micrófono** para medir la **intensidad del sonido**.
- A cambiar la **referencia** de las entradas analógicas para medir señales pequeñas con más precisión.
- A usar `if`, `else if` y `else` para distinguir **tres niveles**.
- A trabajar con **umbrales numéricos** para definir estados (silencio, ruido moderado y mucho ruido).
- A entender por qué algunas señales, como el sonido, **cambian constantemente**.

1.2 Qué vamos a usar

Componentes

- **Micrófono (pin A7):** Convierte las vibraciones del sonido en una señal eléctrica. Da valores bajos con silencio y valores más altos cuanto más intenso es el sonido.
- **LED verde (pin 11), naranja (pin 12) y rojo (pin 13):** Nos indicarán el nivel de ruido.





Programación

El micrófono está en un pin **analógico** (ver la tabla de pines de la [Introducción](#)), así que lo leemos con `analogRead`. El problema es que su señal es muy **pequeña**: con la configuración normal, `analogRead` reparte los valores de 0 a 1023 entre 0 y 5 V, y el micrófono apenas llega a dar unos pocos valores. Para solucionarlo usamos la función `analogReference(tipo)`:

- **tipo**: la tensión de **referencia**, es decir, la tensión que corresponde al valor máximo, 1023. Con `INTERNAL` usamos una referencia interna de **1,1 V** en lugar de 5 V.

Así, los valores de 0 a 1023 se reparten entre 0 y 1,1 V, y la misma señal del micrófono nos da valores casi cinco veces más grandes. La referencia se cambia una sola vez, en `setup()`, y afecta a todas las lecturas con `analogRead`.

2. Programamos

Revisamos continuamente el valor del micrófono y, según el nivel de ruido, encendemos o apagamos cada uno de los tres LED.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Vúmetro: los LED verde, naranja y rojo indican el nivel de ruido
2
3 const int microfono = A7; // micrófono conectado al pin A7
4 const int ledVerde = 11; // LED verde conectado al pin 11
5 const int ledNaranja = 12; // LED naranja conectado al pin 12
6 const int ledRojo = 13; // LED rojo conectado al pin 13
7
8 const int umbralSilencio = 2; // por debajo de este valor hay silencio
9 const int umbralRuido = 20; // por debajo de este valor el ruido es moderado
10
11 int valorSonido = 0; // guarda el valor que lee el micrófono
12
13 void setup() {
14     pinMode(ledVerde, OUTPUT);
15     pinMode(ledNaranja, OUTPUT);
16     pinMode(ledRojo, OUTPUT);
17
18     analogReference(INTERNAL); // referencia de 1,1 V: el micrófono da valores más altos
19 }
20 void loop() {
21     // lee el micrófono
22     valorSonido = analogRead(microfono);
23
24     if (valorSonido < umbralSilencio) {
25         // silencio: solo el LED verde
26         digitalWrite(ledVerde, HIGH);
27         digitalWrite(ledNaranja, LOW);
28         digitalWrite(ledRojo, LOW);
29     } else if (valorSonido < umbralRuido) {
30         // ruido moderado: verde y naranja
31         digitalWrite(ledVerde, HIGH);
32         digitalWrite(ledNaranja, HIGH);
33         digitalWrite(ledRojo, LOW);
34     } else {
35         // mucho ruido: los tres LED
36         digitalWrite(ledVerde, HIGH);
37         digitalWrite(ledNaranja, HIGH);
38         digitalWrite(ledRojo, HIGH);
39     }
40 }
```



Cómo funciona:

Las constantes (líneas 3 a 9): guardan los pines del micrófono y de los tres LED, y los dos **umbrales** que separan los niveles de ruido.

La variable (línea 11): `valorSonido` guarda el valor que lee el micrófono en cada momento.

La función `setup()` (líneas 13 a 18): configura los tres LED como **salidas** y cambia la referencia de las entradas analógicas a 1,1 V con `analogReference(INTERNAL)`.

La función `loop()` (líneas 20 a 40): lee el micrófono con `analogRead` y compara el valor con los umbrales usando `if`, `else if` y `else`: el programa comprueba las condiciones en orden y ejecuta solo la primera que se cumple. El programa revisa continuamente:

```
SI el micrófono registra valores menores de 2 (silencio):  
    --> Se enciende el LED verde y se apagan el naranja y el rojo.  
  
SI NO, SI registra valores menores de 20 (entre 2 y 20, ruido moderado):  
    --> Se encienden los LED verde y naranja y se apaga el rojo.  
  
SI NO (20 o más, mucho ruido):  
    --> Se encienden los tres LED.
```

Es probable que veas que los LED **parpadean** aunque el ruido sea constante. Esto ocurre porque la señal del sonido cambia muy deprisa: el micrófono capta las vibraciones de la onda sonora y cada lectura da un valor distinto. En la primera propuesta de Mejóralo aprenderás a solucionarlo.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Calibra tu aula:** Observa qué valores mide el micrófono con silencio, hablando en voz baja y dando una palmada, y ajusta los umbrales (2 y 20) para que el vúmetro funcione bien en tu clase.

Pista: envía el valor del micrófono al ordenador con `Serial.begin(9600)` en `setup()` y `Serial.println(valorSonido)` en `loop()`, como en el proyecto [El echidna dice la temperatura](#). Además del **Monitor serie**, prueba el **Serial Plotter** (menú **Herramientas > Serial Plotter**), que dibuja los valores como una gráfica y deja ver muy bien cómo cambia el sonido.

2. **Una lectura más estable:** Para que los LED no parpadeen, en lugar de usar cada lectura del micrófono, calcula la **media** de varias (por ejemplo, de 32) y usa esa media en los condicionales.

Pista: necesitas una variable que vaya sumando las lecturas (suma). Pon la suma a 0, haz las lecturas con un bucle `for`, como en el proyecto [Brillo LED](#), y al terminar divide la suma entre el número de lecturas.

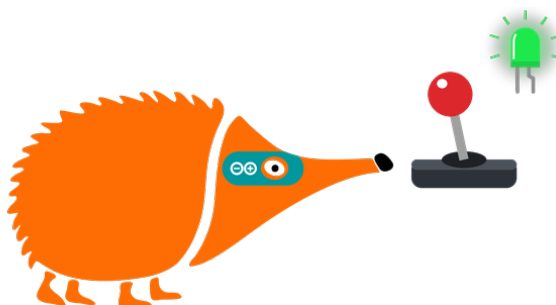


3. **Encendido por palmada:** Programa un interruptor que funcione con palmadas: con una palmada, el **LED rojo** se enciende, y con otra palmada, se apaga.

Pista: una palmada da un valor muy alto en el micrófono, así que basta con un solo umbral. Necesitas una variable que recuerde si el LED está encendido o apagado, como en el pulsador con memoria del proyecto [Interruptor de luz](#). Después de cada palmada, espera un poco con `delay` para que el final del sonido no cuente como otra palmada.



2.11 Regulador de luz con joystick



1. Qué vamos a hacer

Vamos a usar el **joystick** como el regulador de luz de una lámpara: cuanto más lo alejemos del centro, hacia la izquierda o hacia la derecha, más brillará el **LED verde**. Con el joystick en el centro, el LED estará apagado.

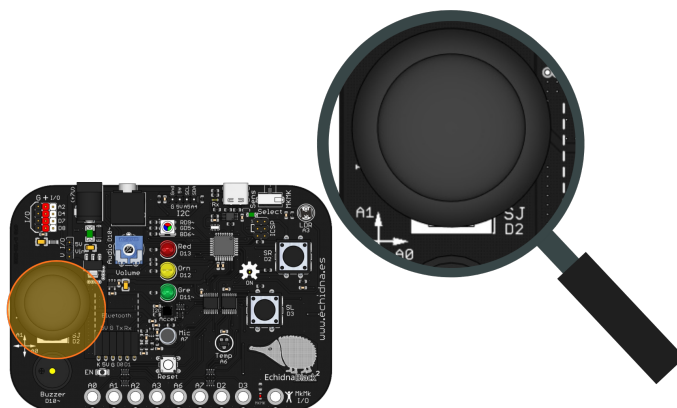
1.1 Qué vamos a aprender

- Cómo funciona un **joystick**: dos potenciómetros, uno para cada eje, que leemos con `analogRead`.
- A convertir un rango de valores en otro con la función **map**, por ejemplo, la posición del joystick en un brillo.
- A repasar `analogWrite` y el condicional `if ... else if ... else`.

1.2 Qué vamos a usar

Componentes

- **Joystick (eje X en el pin A0 y eje Y en el pin A1)**: Palanca que se mueve en dos direcciones. Por dentro tiene dos **potenciómetros**, uno por eje, que dan un valor según la posición de la palanca. En este proyecto usamos solo el eje X.
- **LED verde (pin 11)**: Su brillo se regula con PWM, como en el proyecto [Brillo LED](#).





Programación

Los ejes del joystick están en pines **analógicos**, así que los leemos con `analogRead`. Con la palanca en reposo, los dos ejes dan valores en torno a 512. El eje X baja hasta 0 a la **izquierda** y sube hasta 1023 a la **derecha**; el eje Y da 0 **abajo** y 1023 **arriba**.

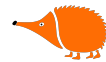
El problema es que `analogWrite` necesita un brillo entre 0 y 255. Para pasar de un rango a otro usamos la función `map(valor, desdeMin, desdeMax, hastaMin, hastaMax)`:

- **valor**: el número que queremos convertir, por ejemplo, la lectura del joystick.
- **desdeMin** y **desdeMax**: el rango en el que está ese número.
- **hastaMin** y **hastaMax**: el rango al que lo queremos llevar.

Por ejemplo, `map(valorX, 532, 1023, 0, 255)` convierte un valor entre 532 y 1023 en otro entre 0 y 255: 532 da 0, 1023 da 255 y los valores intermedios, lo que corresponda en proporción. Si en el primer rango ponemos los números **al revés**, `map(valorX, 492, 0, 0, 255)`, el rango se invierte: 492 da 0 y 0 da 255.

2. Programamos

El programa lee el eje X del joystick. Si la palanca está en el centro (entre 492 y 532), apaga el LED; si se ha movido a la izquierda o a la derecha, convierte su posición en un brillo con `map`: cuanto más lejos del centro, más brillo.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Regulador de luz: el LED verde brilla más cuanto más movemos el joystick
2
3 const int joystickX = A0; // eje X del joystick en el pin A0
4 const int ledVerde = 11; // LED verde conectado al pin 11
5
6 const int centroMin = 492; // por debajo, el joystick está hacia la izquierda
7 const int centroMax = 532; // por encima, hacia la derecha
8
9 int valorX = 0; // guarda la posición del joystick en el eje X (de 0 a 1023)
10 int brillo = 0; // guarda el brillo del LED (de 0 a 255)
11
12 void setup() {
13     pinMode(ledVerde, OUTPUT); // el LED es una salida
14 }
15
16 void loop() {
17     // lee la posición del joystick
18     valorX = analogRead(joystickX);
19
20     if (valorX > centroMax) {
21         // hacia la derecha: de 532 a 1023 pasa a 0 a 255
22         brillo = map(valorX, centroMax, 1023, 0, 255);
23     } else if (valorX < centroMin) {
24         // hacia la izquierda: de 492 a 0 pasa a 0 a 255
25         brillo = map(valorX, centroMin, 0, 0, 255);
26     } else {
27         brillo = 0; // en el centro: apagado
28     }
29
30     analogWrite(ledVerde, brillo); // enciende el LED con ese brillo
31 }
```

Cómo funciona:

Las constantes (líneas 3 a 7): guardan los pines del joystick y del LED, y los dos límites de la **zona del centro**, centroMin y centroMax, donde el LED está apagado.

Las variables (líneas 9 y 10): valorX guarda la posición del joystick y brillo, el brillo que le daremos al LED.

La función setup() (líneas 12 a 14): configura el pin del LED como **salida**. El joystick no necesita pinMode, porque está en un pin analógico.



La función `loop()` (líneas 16 a 31): lee el joystick y, con `if ... else if ... else`, decide el brillo:

```
SI el joystick da más de 532 (movido a la derecha):  
    --> El brillo va de 0 (en 532) a 255 (en 1023).  
SI NO, SI da menos de 492 (movido a la izquierda):  
    --> El brillo va de 0 (en 492) a 255 (en 0).  
SI NO (está en el centro):  
    --> El brillo es 0.
```

La zona del centro, de 492 a 532, evita que el LED se encienda un poco con el joystick suelto, porque la palanca nunca vuelve exactamente a 512. Por último, `analogWrite` enciende el LED con el brillo calculado.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Lee el joystick:** Mira qué valores dan los ejes X e Y del joystick de tu placa en reposo y al moverlo, y ajusta las constantes `centroMin` y `centroMax` si el LED se enciende un poco con el joystick suelto.

Pista: envía los dos valores al ordenador y ábrelos con el **Serial Plotter** (menú **Herramientas > Serial Plotter**), que los dibuja como una gráfica, como en la primera mejora del [Vúmetro](#). Si escribes antes de cada valor un nombre seguido de dos puntos (X: e Y:), el Serial Plotter dibuja cada eje con su color y su nombre.

2. **Mando de colores:** Usa los dos ejes del joystick para mezclar colores en el **LED RGB**: el eje X controla el **rojo** y el eje Y, el **azul**. Con la palanca en el centro, el LED se verá morado, y al moverla cambiará la mezcla.

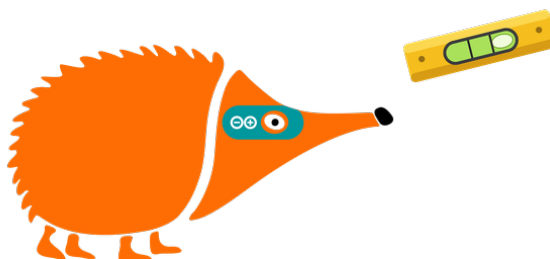
Pista: lee los dos ejes y convierte cada uno, de 0 a 1023, en una intensidad de 0 a 255 con `map`. El rojo del LED RGB está en el pin 9 y el azul, en el 6.

3. **Theremín:** Convierte la placa en un **theremín**, un instrumento que se toca sin tocarlo: al mover el joystick, el **zumbador** cambia de nota, más grave hacia la izquierda y más aguda hacia la derecha. Con el joystick en el centro, silencio.

Pista: convierte la posición del joystick en una frecuencia con `map`, por ejemplo, de 200 Hz a la izquierda a 1000 Hz a la derecha, y hazla sonar con `tone(zumbador, frecuencia)`. Para que no suene todo el rato, comprueba si el joystick está **fuera** del centro: para saber si se cumple **una u otra** condición se usa `||` («o»), `if (valorX < centroMin || valorX > centroMax)`. Si está en el centro, calla el zumbador con `noTone(zumbador)`.



2.12 Nivel de burbuja



1. Qué vamos a hacer

Vamos a convertir la placa en un **nivel de burbuja**, como el que usan los albañiles para comprobar si algo está recto. El **acelerómetro** medirá cuánto inclinamos la placa y la luz se moverá por la columna de LED (verde, naranja, rojo y el azul del LED RGB): con la placa horizontal, todos apagados; al inclinarla un poco, se enciende uno de los LED del centro, y al inclinarla mucho, uno de los extremos.

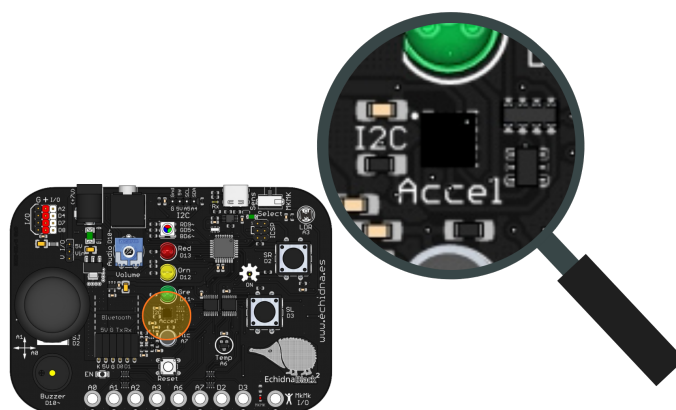
1.1 Qué vamos a aprender

- Qué mide un **acelerómetro**: la **gravedad** en cada uno de sus ejes, que cambia al inclinar la placa.
- A instalar y usar una **librería**, un conjunto de instrucciones ya programadas para manejar un componente.
- A repasar el condicional **if ... else if ... else** para elegir entre varios casos.

1.2 Qué vamos a usar

Componentes

- **Acelerómetro (I2C, pines A4 y A5)**: Mide la aceleración en tres ejes (X, Y y Z). Con la placa quieta, lo único que mide es la **gravedad**, así que nos dice hacia dónde está inclinada. Se comunica con el microcontrolador por **I2C**, una conexión de dos cables (SDA en A4 y SCL en A5).
- **LED verde (pin 11), naranja (pin 12) y rojo (pin 13) y el azul del LED RGB (pin 6)**: Forman la columna por la que se mueve la luz.



Programación

El acelerómetro no se lee con `analogRead`: envía sus medidas por I2C y entenderlas a mano sería complicado. Para eso usamos una **librería**, que ya trae programadas las instrucciones para manejarlo. Antes de usarla hay que instalarla en Arduino IDE, una sola vez:

1. Abre el menú **Herramientas > Gestionar bibliotecas**.
2. Busca **Adafruit LIS3DH** (LIS3DH es el modelo del acelerómetro de la placa).
3. Pulsa **Instalar** y, cuando pregunte si quieres instalar también las librerías que necesita, elige **Instalar todo**.

Después, en el programa usamos la librería así:

- **`#include <Adafruit_LIS3DH.h>`**: al principio del programa, añade la librería.
- **`Adafruit_LIS3DH acelerometro;`** crea el **acelerómetro** en el programa, con el nombre que elijamos (aquí, `acelerometro`).
- **`acelerometro.begin(0x18)`**: en `setup()`, pone en marcha el acelerómetro. `0x18` es su **dirección** en la conexión I2C.
- **`acelerometro.read()`**: en `loop()`, lee el acelerómetro.
- **`acelerometro.y_g`**: después de leer, nos da la aceleración del **eje Y** en **g** (la gravedad de la Tierra): unos 0 con la placa horizontal, que crece hacia 1 al inclinarla hacia un lado y baja hacia -1 al inclinarla hacia el otro. Igual, `acelerometro.x_g` y `acelerometro.z_g` dan los ejes X y Z.

Como la medida tiene decimales, la guardamos en una variable de tipo `float`.

2. Programamos

El programa lee continuamente la inclinación del eje Y y, con varios `else if`, enciende el LED que le corresponde: verde o naranja hacia un lado, ninguno en horizontal, y rojo o azul hacia el otro.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:



```
1 // Nivel de burbuja: la luz se mueve por los LED al inclinar la placa
2
3 #include <Adafruit_LIS3DH.h> // librería del acelerómetro
4
5 const int rgbAzul = 6;      // LED RGB: color azul en el pin 6
6 const int ledVerde = 11;    // LED verde conectado al pin 11
7 const int ledNaranja = 12;  // LED naranja conectado al pin 12
8 const int ledRojo = 13;     // LED rojo conectado al pin 13
9
10 const float inclinacionPoca = 0.2; // algo inclinada (en g)
11 const float inclinacionMucha = 0.6; // muy inclinada (en g)
12
13 Adafruit_LIS3DH acelerometro; // el acelerómetro de la placa
14
15 float inclinacion = 0.0; // guarda la aceleración del eje Y, en g
16
17 void setup() {
18     pinMode(rgbAzul, OUTPUT); // los LED son salidas
19     pinMode(ledVerde, OUTPUT);
20     pinMode(ledNaranja, OUTPUT);
21     pinMode(ledRojo, OUTPUT);
22     acelerometro.begin(0x18); // pone en marcha el acelerómetro
23 }
24
25 void loop() {
26     // lee el acelerómetro y guarda la inclinación del eje Y
27     acelerometro.read();
28     inclinacion = acelerometro.y_g;
29
30     if (inclinacion < -inclinacionMucha) {
31         // muy inclinada hacia un lado: LED verde
32         digitalWrite(ledVerde, HIGH);
33         digitalWrite(ledNaranja, LOW);
34         digitalWrite(ledRojo, LOW);
35         digitalWrite(rgbAzul, LOW);
36     } else if (inclinacion < -inclinacionPoca) {
37         // algo inclinada hacia ese lado: LED naranja
38         digitalWrite(ledVerde, LOW);
39         digitalWrite(ledNaranja, HIGH);
40         digitalWrite(ledRojo, LOW);
41         digitalWrite(rgbAzul, LOW);
42     } else if (inclinacion < inclinacionPoca) {
43         // horizontal: todos apagados
```



```
44     digitalWrite(ledVerde, LOW);
45     digitalWrite(ledNaranja, LOW);
46     digitalWrite(ledRojo, LOW);
47     digitalWrite(rgbAzul, LOW);
48 } else if (inclinacion < inclinacionMucha) {
49     // algo inclinada hacia el otro lado: LED rojo
50     digitalWrite(ledVerde, LOW);
51     digitalWrite(ledNaranja, LOW);
52     digitalWrite(ledRojo, HIGH);
53     digitalWrite(rgbAzul, LOW);
54 } else {
55     // muy inclinada hacia el otro lado: azul del LED RGB
56     digitalWrite(ledVerde, LOW);
57     digitalWrite(ledNaranja, LOW);
58     digitalWrite(ledRojo, LOW);
59     digitalWrite(rgbAzul, HIGH);
60 }
61 }
```

Cómo funciona:

La librería (línea 3): `#include` añade al programa la librería del acelerómetro que hemos instalado.

Las constantes (líneas 5 a 11): guardan los pines de los cuatro LED y los dos umbrales de inclinación, `0.2 g` (algo inclinada) y `0.6 g` (muy inclinada). Son de tipo `float` porque tienen decimales.

El acelerómetro y la variable (líneas 13 y 15): `Adafruit_LIS3DH` acelerometro; crea el acelerómetro en el programa, y `inclinacion` guardará la medida del eje Y.

La función `setup()` (líneas 17 a 23): configura los pines de los LED como salidas y pone en marcha el acelerómetro con `acelerometro.begin(0x18)`.



La función `loop()` (líneas 25 a 61): lee el acelerómetro con `acelerometro.read()` y guarda en `inclinacion` el valor del eje Y. Después, como en el [Vúmetro](#), una cadena de `if ... else if ... else` comprueba los casos en orden, de la inclinación más negativa a la más positiva, y ejecuta solo el primero que se cumple. Fíjate en que `-inclinacionMucha` es el umbral con el signo cambiado, `-0.6`:

```
SI la inclinación es menor que -0.6:
    --> Enciende el LED verde.
SI NO, SI es menor que -0.2:
    --> Enciende el LED naranja.
SI NO, SI es menor que 0.2 (está casi horizontal):
    --> Apaga todos los LED.
SI NO, SI es menor que 0.6:
    --> Enciende el LED rojo.
SI NO (es 0.6 o más):
    --> Enciende el azul del LED RGB.
```

Como el verde, el naranja, el rojo y el LED RGB están uno encima de otro en la placa, al inclinarla la luz se desplaza por la columna, como la burbuja de un nivel.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Lee el acelerómetro:** Mira cómo cambian los valores de los ejes X e Y cuando inclinas la placa hacia delante, hacia atrás y hacia los lados.

Pista: envía los dos valores al ordenador y ábrelos con el **Serial Plotter**, como en la primera mejora del [Regulador de luz con joystick](#): escribe X: antes del valor del eje X e Y: antes del valor del eje Y. Los ejes del acelerómetro se leen con `acelerometro.x_g` y `acelerometro.y_g`.

2. **Alarma de caída:** Haz que el **zumbador** pite cuando la placa esté muy inclinada, hacia cualquiera de los dos lados.

Pista: el zumbador está en el pin 10 y suena con `analogWrite(zumbador, 125)`. Para comprobar si se cumple **una u otra** condición usa `||` («o»), como en el `theremín` del [Regulador de luz con joystick](#): `if (inclinacion > inclinacionMucha || inclinacion < -inclinacionMucha)`.

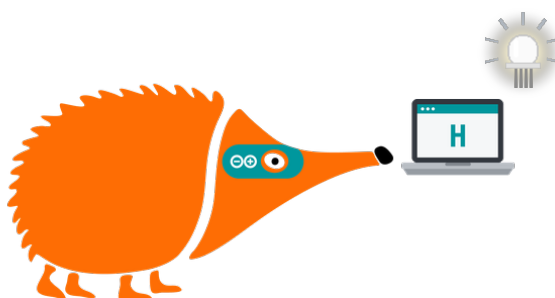
3. **La luz que rueda:** Haz que la luz **ruede** por los cuatro LED hacia el lado al que inclinas la placa: mientras está inclinada, pasa de un LED al siguiente y, al llegar al último, vuelve a empezar por el otro extremo. Con la placa horizontal, la luz se queda quieta.



Pista: usa una variable `posicion` que diga qué LED está encendido (1 el verde, 2 el naranja, 3 el rojo y 4 el azul). En cada vuelta de `loop()`, súmale 1 si la placa está inclinada hacia un lado y réstale 1 si lo está hacia el otro. Si pasa de 4, vuelve a 1, y si baja de 1, pasa a 4. Después, enciende solo el LED de esa posición y espera un poco con `delay` para que se vea el movimiento.



2.13 Control desde el ordenador



1. Qué vamos a hacer

Vamos a usar el teclado del ordenador como un **mando a distancia**: escribiendo una **H** en el **Monitor serie**, el **LED RGB** se encenderá en color blanco, y escribiendo una **L**, se apagará. La placa nos contestará en el Monitor serie para confirmar que ha recibido la orden.

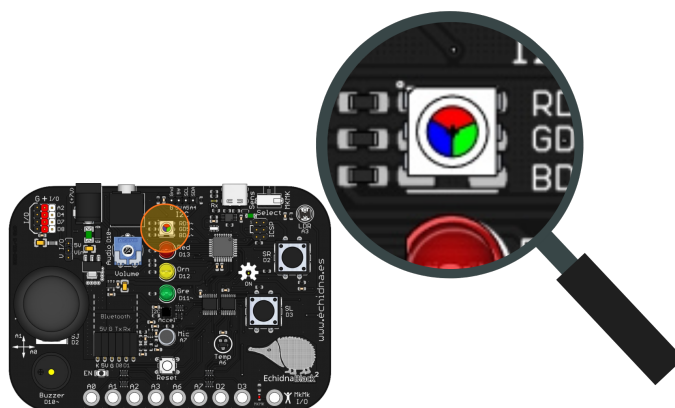
1.1 Qué vamos a aprender

- Que el **Monitor serie** no solo sirve para que la placa nos envíe datos, sino también para **enviarle órdenes** desde el ordenador.
- A comprobar si han llegado datos con `Serial.available()` y a leerlos con `Serial.read()`.
- A guardar un **carácter** en una variable de tipo `char` y a compararlo con una letra.

1.2 Qué vamos a usar

Componentes

- **LED RGB (pines 9, 5 y 6)**: Con sus tres colores encendidos a la vez da luz **blanca**.
- **Ordenador conectado por el cable USB**: Desde el Monitor serie de Arduino IDE escribiremos las órdenes.





Programación

En el proyecto [El echidna dice la temperatura](#) la placa enviaba mensajes al ordenador con `Serial.print`. La comunicación también funciona al revés: lo que escribimos en el Monitor serie llega a la placa **carácter a carácter** y espera guardado hasta que el programa lo lee. Para eso usamos dos funciones:

- `Serial.available()`: dice cuántos caracteres han llegado y están esperando a ser leídos. Si es mayor que 0, hay algo que leer.
- `Serial.read()`: lee el primero de los caracteres que están esperando.

El carácter leído lo guardamos en una variable de tipo **char**, que guarda un carácter, y lo comparamos con una letra escrita entre comillas **simples**: `if (letra == 'H')`.

Para enviar una orden, abre el Monitor serie (**Herramientas > Monitor Serie**, a **9600 baudios**), escribe la letra en la caja de texto de la parte superior y pulsa **Intro**. Fíjate en que 'H' y 'h' son caracteres distintos: escribe las letras en mayúsculas.



2. Programamos

El programa comprueba continuamente si ha llegado algún carácter desde el ordenador. Si ha llegado, lo lee: con una **H** enciende los tres colores del LED RGB y con una **L** los apaga. Cualquier otro carácter no hace nada.



Escribe el siguiente código en Arduino IDE y cárgalo en la placa:

```
1 // Control desde el ordenador: H enciende el LED RGB en blanco y L lo apaga
2
3 const int rgbRojo = 9;   // LED RGB: color rojo en el pin 9
4 const int rgbVerde = 5;  // LED RGB: color verde en el pin 5
5 const int rgbAzul = 6;   // LED RGB: color azul en el pin 6
6
7 char letra = ' '; // guarda la letra que llega del ordenador
8
9 void setup() {
10   pinMode(rgbRojo, OUTPUT); // los tres colores del LED RGB son salidas
11   pinMode(rgbVerde, OUTPUT);
12   pinMode(rgbAzul, OUTPUT);
13   Serial.begin(9600);       // abre la comunicación a 9600 baudios
14 }
15
16 void loop() {
17   // ¿ha llegado alguna letra desde el ordenador?
18   if (Serial.available() > 0) {
19     letra = Serial.read(); // lee la letra
20
21     if (letra == 'H') {
22       // H (de HIGH): enciende los tres colores, luz blanca
23       digitalWrite(rgbRojo, HIGH);
24       digitalWrite(rgbVerde, HIGH);
25       digitalWrite(rgbAzul, HIGH);
26       Serial.println("LED encendido");
27     }
28
29     if (letra == 'L') {
30       // L (de LOW): apaga los tres colores
31       digitalWrite(rgbRojo, LOW);
32       digitalWrite(rgbVerde, LOW);
33       digitalWrite(rgbAzul, LOW);
34       Serial.println("LED apagado");
35     }
36   }
37 }
```

Cómo funciona:

Las constantes (líneas 3 a 5): guardan los pines de los tres colores del LED RGB.

La variable (línea 7): `letra` es de tipo `char` y guarda el carácter que llega del ordenador.



La función `setup()` (líneas 9 a 14): configura los pines del LED RGB como **salidas** y abre la comunicación con el ordenador con `Serial.begin(9600)`.

La función `loop()` (líneas 16 a 37): con `Serial.available() > 0` comprueba si ha llegado algo. Si no ha llegado nada, no hace nada y vuelve a empezar; si ha llegado, lee el carácter con `Serial.read()` y lo compara con las dos órdenes:

```
SI ha llegado algún carácter:
  --> Lo lee y lo guarda en letra.
  SI es una H:
    --> Enciende el LED RGB en blanco y contesta «LED encendido».
  SI es una L:
    --> Apaga el LED RGB y contesta «LED apagado».
```

Las letras **H** y **L** recuerdan a **HIGH** y **LOW**. Los demás caracteres se leen, pero no hacen nada; por ejemplo, el **salto de línea** que el Monitor serie puede enviar al pulsar Intro.

3. Mejóralo

Prueba a realizar algunas de las siguientes modificaciones al proyecto:

1. **Elige el color:** Añade tres órdenes más: con la **R** se enciende solo el rojo, con la **G**, solo el verde, y con la **B**, solo el azul (Red, Green y Blue en inglés).

Pista: añade un `if` para cada letra, como los de la H y la L. En cada uno, enciende el color que corresponde y apaga los otros dos, y contesta con un mensaje.

2. **Brillo desde el teclado:** Haz que, al escribir un número de 0 a 255 terminado en `>` (por ejemplo, `128>`), el LED RGB se encienda en blanco con ese **brillo**.

Pista: los números también llegan **carácter a carácter**: 128 llega como `'1'`, `'2'` y `'8'`. Para saber si un carácter es una cifra, comprueba `letra >= '0' && letra <= '9'`, y para convertirlo en su valor, réstale `'0'` (`'8' - '0'` vale 8). Ve juntando las cifras en una variable `numero` con `numero = numero * 10 + (letra - '0')`: 1, después 12 y después 128. Cuando llegue el `>`, usa el número como brillo con `analogWrite` y vuelve a poner `numero` a 0 para el siguiente.

3. **Color a la carta:** Elige el color exacto del LED RGB escribiendo la intensidad de cada color seguida de su letra: `r` para el rojo, `g` para el verde y `b` para el azul. Por ejemplo, `254r109g4b` pone el LED del **naranja Echidna** del proyecto [Mezclamos colores](#).

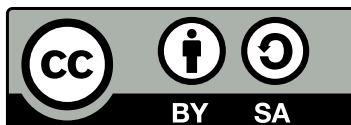
Pista: parte de la mejora anterior. En lugar de un solo `>`, ahora hay tres letras que terminan el número: cada una guarda `numero` en la variable de su color (rojo, verde o azul) y lo vuelve a poner a 0. Al final de `loop()`, enciende cada color con su valor.



Para saber más: estos programas también sirven para controlar la placa por **Bluetooth**. Si conectas un módulo Bluetooth al conector **Bluetooth** de la placa, las órdenes pueden llegar desde un móvil o una tableta, por ejemplo, con una aplicación hecha con **App Inventor**, y el programa las lee igual, con `Serial.available()` y `Serial.read()`. El módulo usa los mismos pines del puerto serie que el cable USB (D0 y D1), así que desconéctalo mientras cargas un programa.



3. Licencia



Proyectos de inicio con Arduino: proyectos sencillos para iniciarse en la programación de la placa **EchidnaBlack2** con **Arduino IDE**.

Autoría: Echidna Educación.

Este recurso se publica bajo la licencia [Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional \(CC BY-SA 4.0\)](https://creativecommons.org/licenses/by-sa/4.0/). Esto significa que puedes:

- **Compartir:** copiar y redistribuir el material en cualquier medio o formato.
- **Adaptar:** remezclar, transformar y crear a partir del material con cualquier finalidad, incluso comercial.

Siempre que cumplas estas condiciones:

- **Reconocimiento:** debes citar la autoría (Echidna Educación), enlazar la licencia e indicar si has hecho cambios.
- **Compartir igual:** si remezclas, transformas o creas a partir del material, debes difundir tu obra con la misma licencia.