

Fase práctica

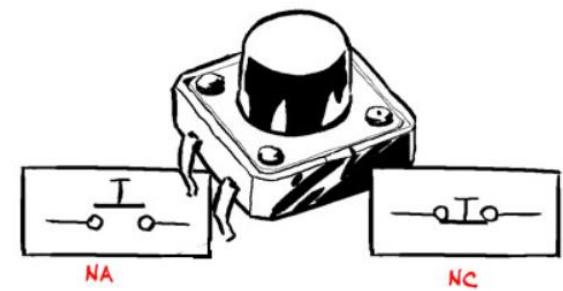
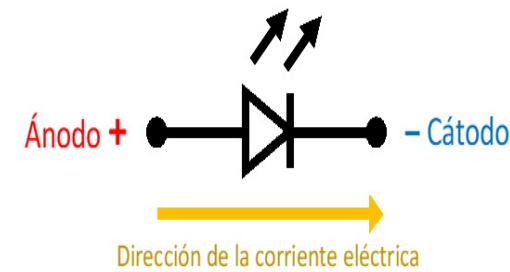
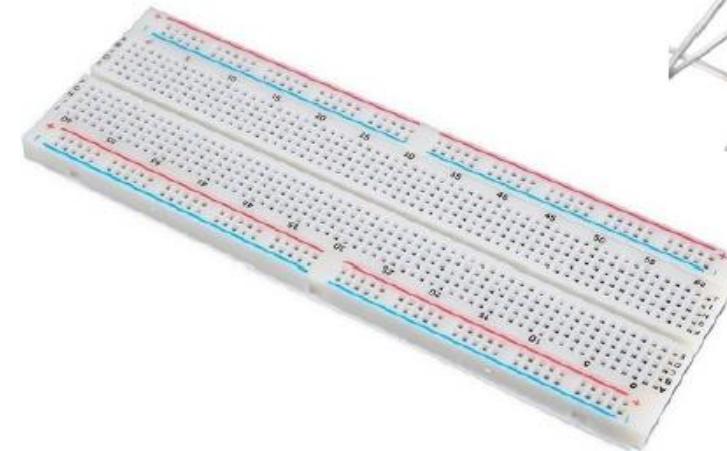
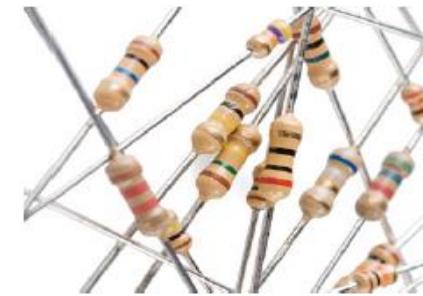
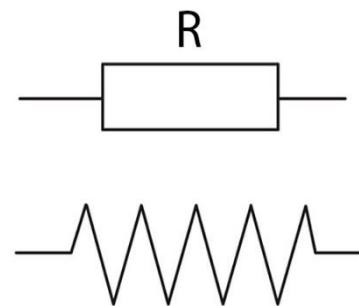
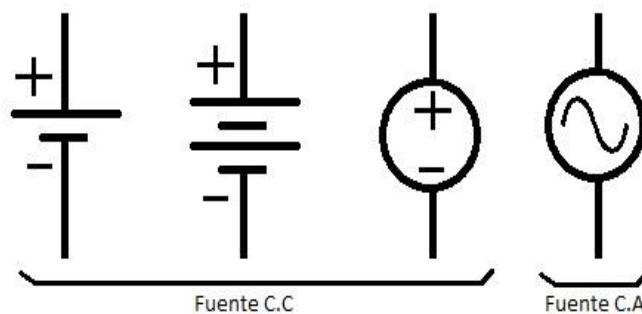
Encender un led

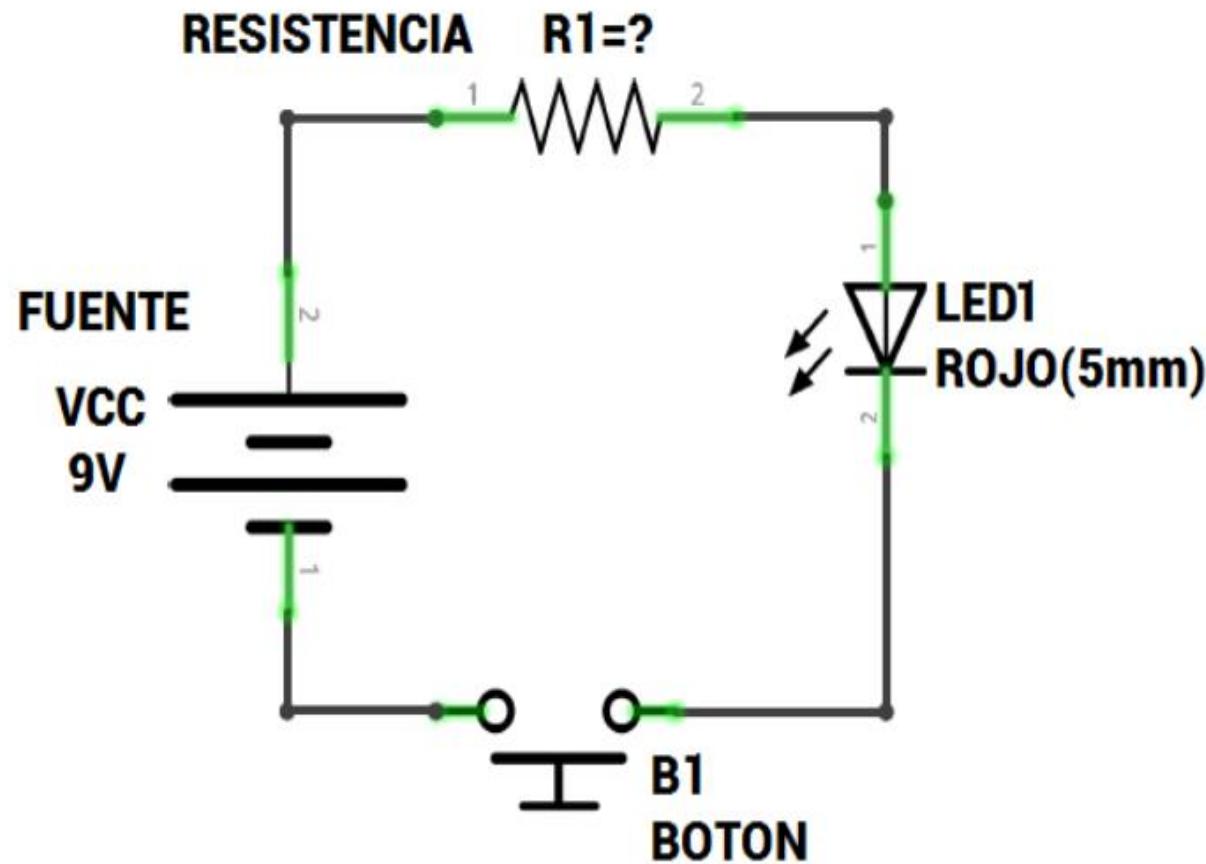
TOULOU
LAUSE
TRE
C

Necesitamos:

- 03 leds.
- 01 resistencia (220 ohm).
- 01 protoboard.
- 01 fuente de alimentación.
- Pulsadores.

Símbolos:





CALCULAR EL VALOR DE LA RESISTENCIA

$$R1 = \frac{VCC - VLED}{ILED}$$

R1= ?
VCC= 9V
VLED= 2.2 ó 2.6 V
ILED= 20mA

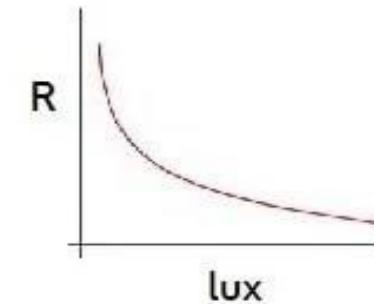
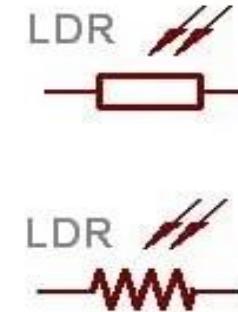
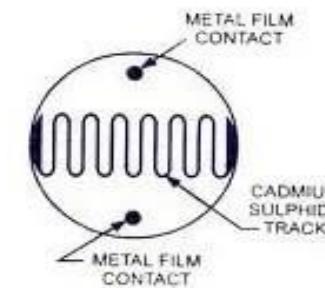
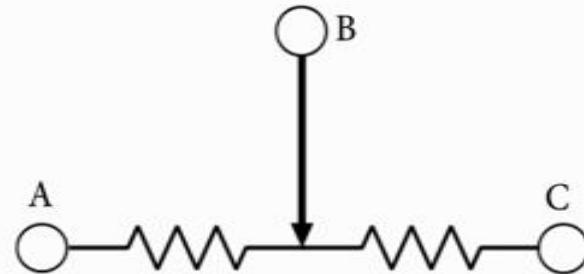
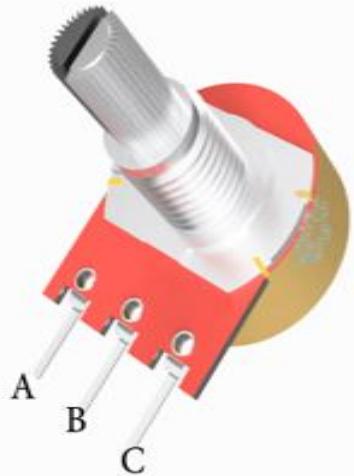
$$R1 = \frac{9V - 2.2V}{0.02A} = 340\Omega \approx 330\Omega$$

**VALOR
COMERCIAL**

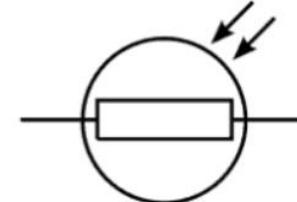
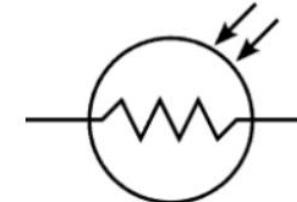
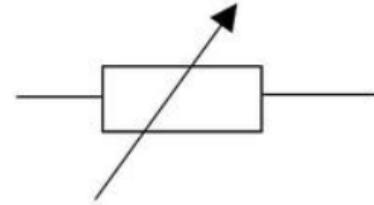
Variar la intensidad de los leds

TOULOU
LAUSE
TRE
C

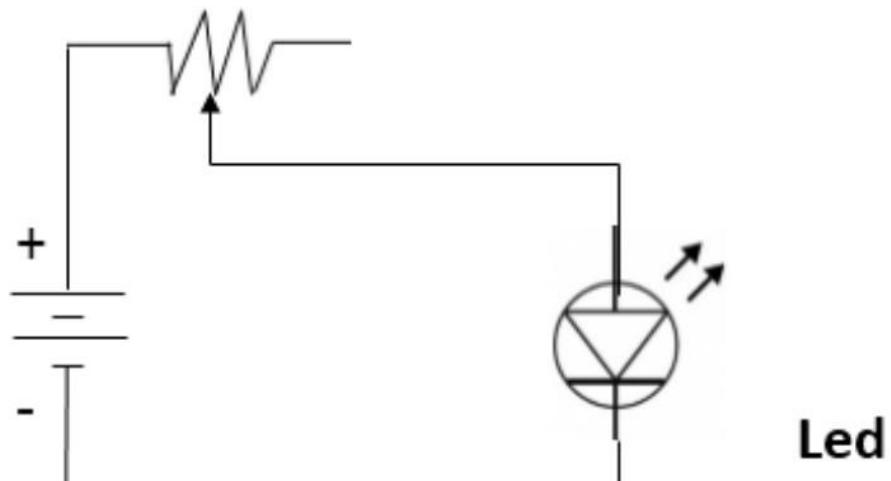
Que mas necesitamos:



Símbolos:

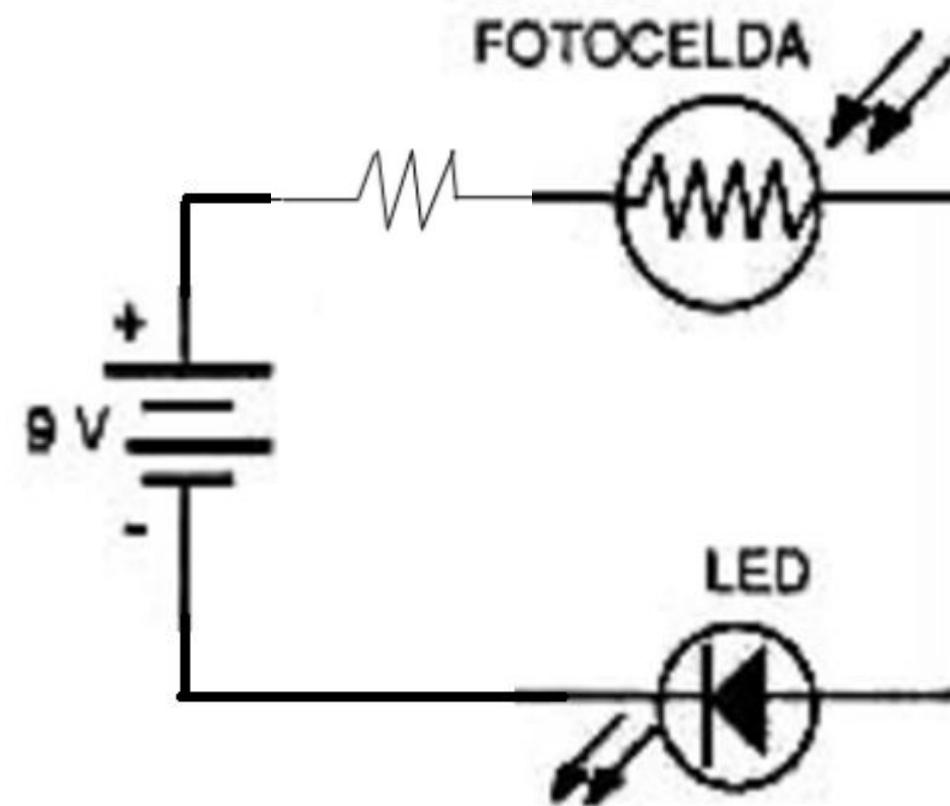


Potenciómetro 100K



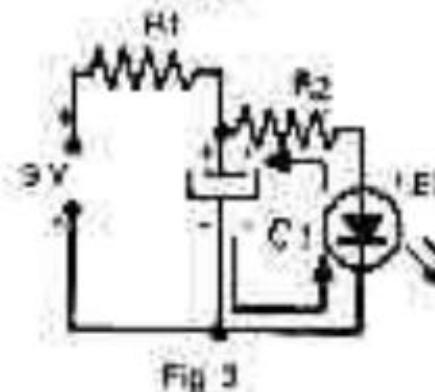
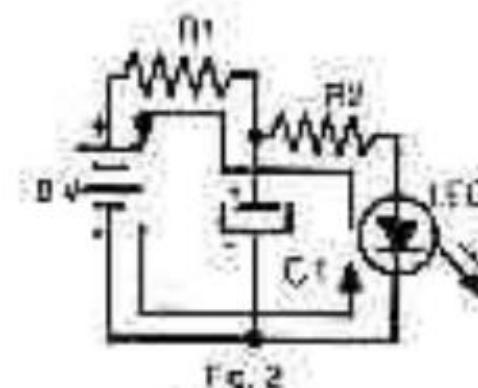
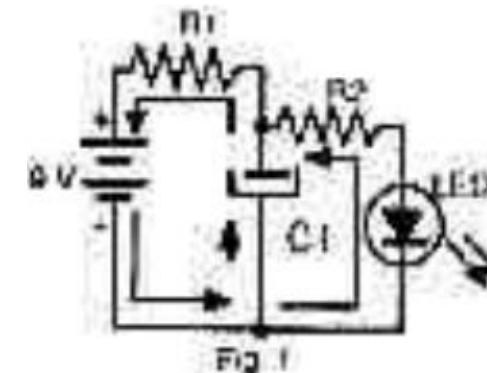
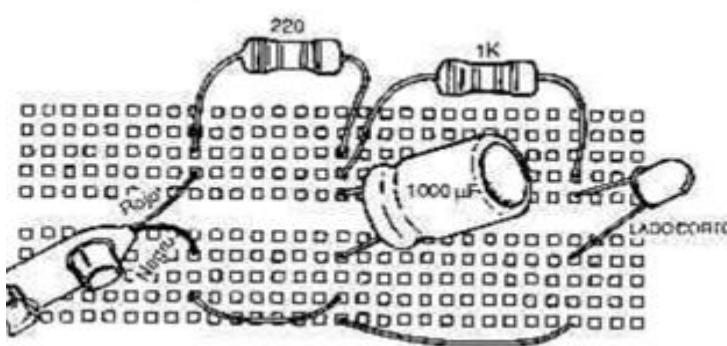
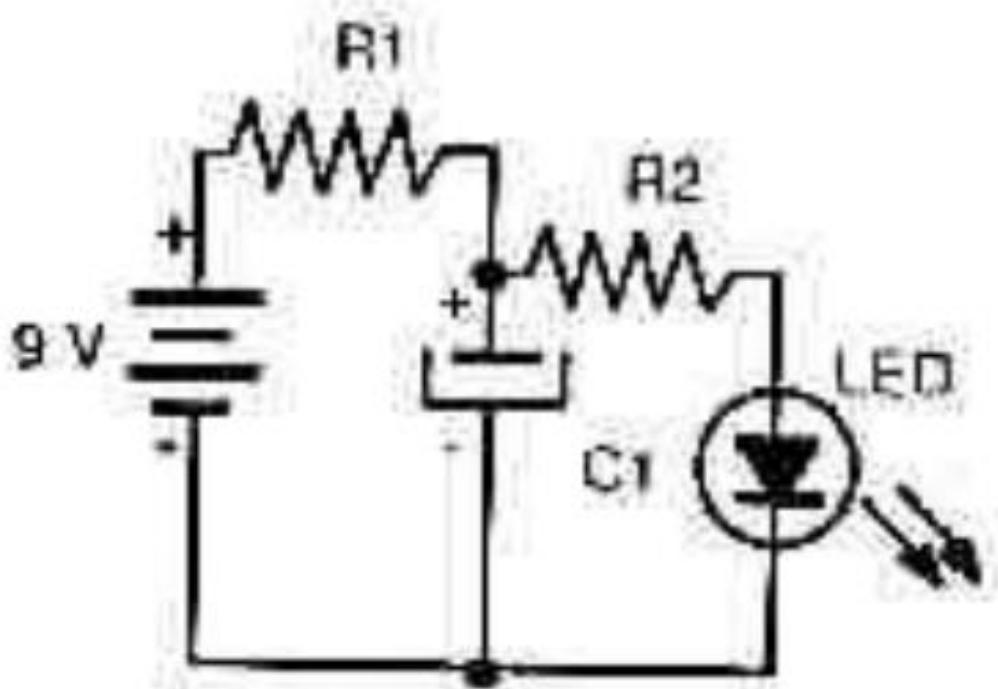
Led

**Resistencia
100 ohmios**





Condensadores



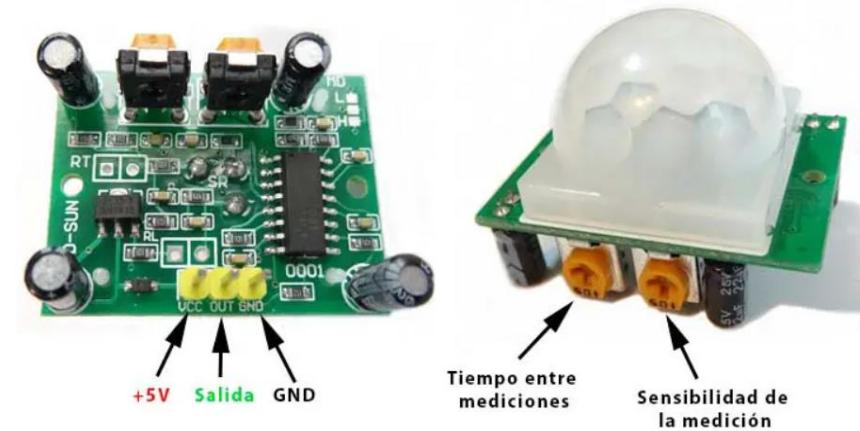
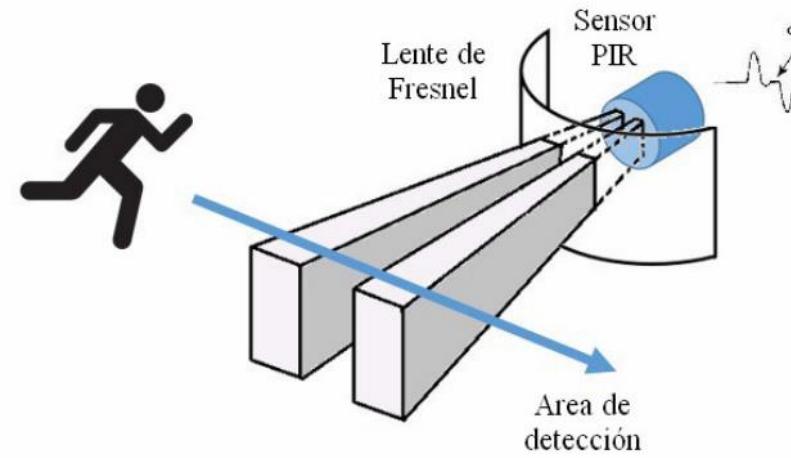
Detector de movimiento

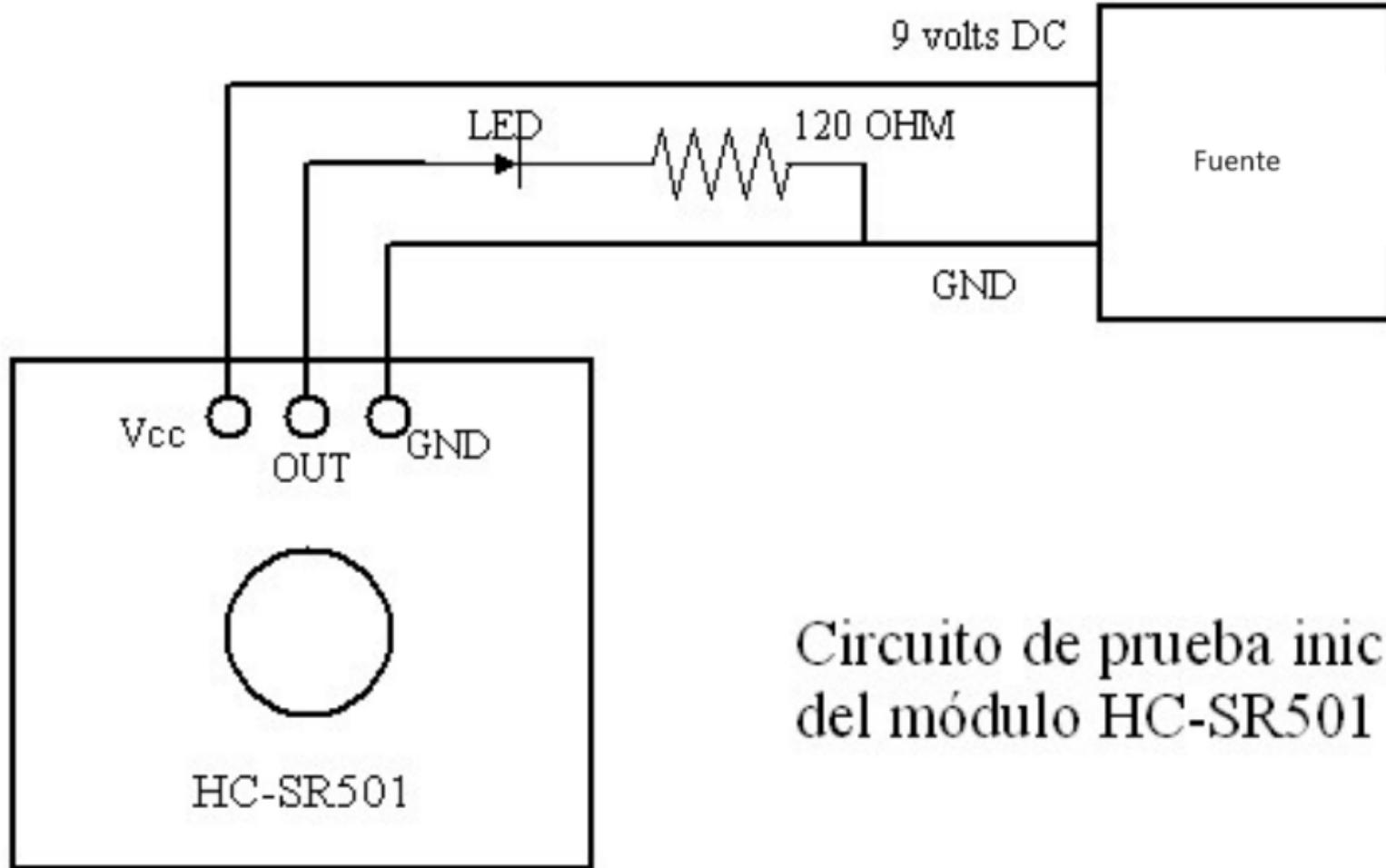
Que mas necesitamos:

01 Sensor de movimiento PIR HC-SR501

Características:

- Dimensiones: (3,6 x 2,5 x 2,0 cm)
- Rango de temperatura de funcionamiento: (0 a 50C)
- El módulo incluye el sensor, lente, controlador PIR BISS0001, regulador y todos los componentes de apoyo para una fácil utilización.
- Rango de detección: 3 m a 7 m, ajustable.
- Salida activa alta a 3.3 V
- Tiempo en estado activo de la salida configurable.
- Consumo de corriente en reposo: < 50 uA
- Voltaje de alimentación: 4.5 VDC a 20 VDC





Utilizando arduino

Entradas / Salidas digitales

Los microcontroladores, y entre ellos el que incorpora Arduino, trabajan con **señales digitales binarias**, que son aquellas que sólo pueden adoptar dos únicos valores.

En los microcontroladores, estas señales son **tensiones**, que pueden tomar dos valores: alto y bajo, que suelen ser **5 V** y **0 V**.

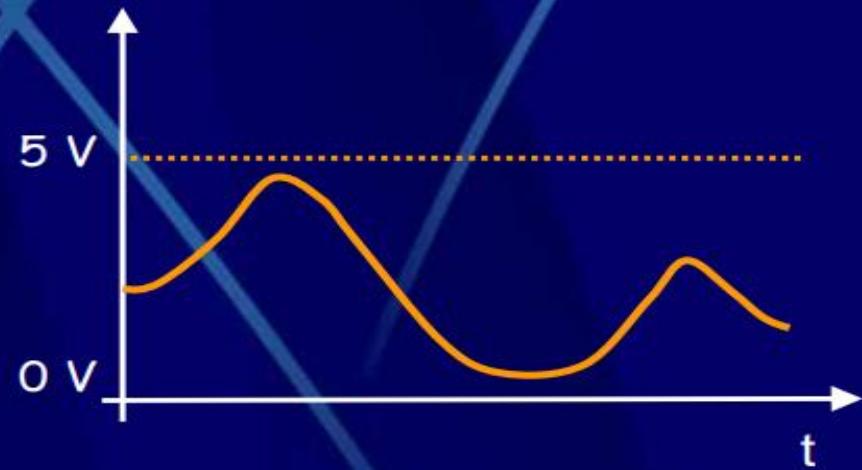


- Las **entradas digitales** de un microcontrolador sólo pueden diferenciar estos dos niveles de tensión o cercanos a ellos.
- Las **salidas digitales** de los microcontroladores sólo pueden aplicar estos dos niveles de tensión a lo que se conecte a ellas.

Entradas / Salidas analógicas

Las **señales analógicas** son aquellas que pueden adoptar infinitos valores entre dos valores extremos (normalmente entre 0 V y 5 V).

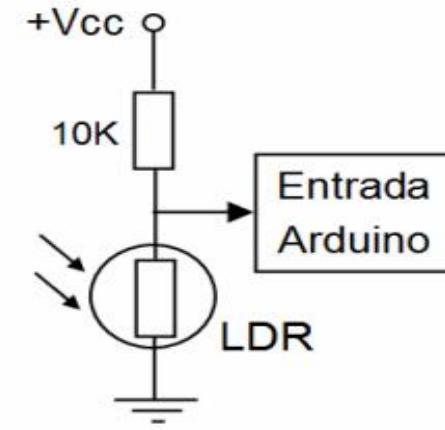
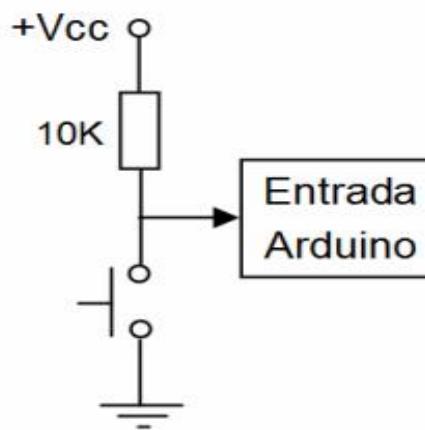
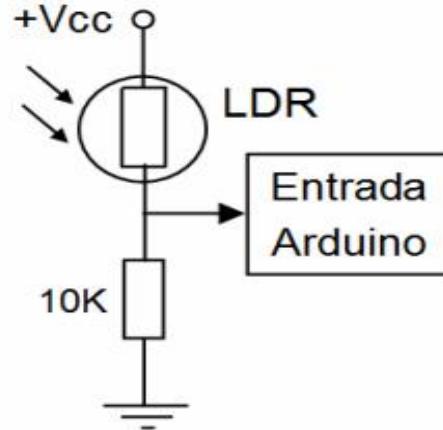
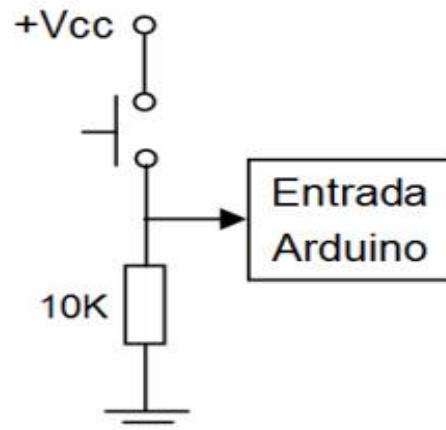
- Como los microcontroladores operan internamente con señales digitales, incorporan un convertidor analógico-digital que transforma las señales analógicas recibidas en las **entradas analógicas** en un valor numérico.
- Las señales suministradas por las **salidas analógicas** de los microcontroladores son realmente señales digitales constituidas por pulsos sucesivos (onda cuadrada) cuya anchura se puede variar, obteniéndose una tensión “pseudanalógica” en la salida entre 0 y 5 V.



Configuración de entradas a Arduino

Una entrada a un microcontrolador, como el de Arduino, debe estar siempre a un valor de tensión claramente definido (o bajo o alto). Una entrada desconectada o a un valor impreciso dará lugar a errores al tomar lecturas de dicha entrada. Se evita este problema colocando **resistencias a masa o a fuente**, como se indica en los esquemas siguientes. Las salidas de Arduino incorporan una resistencias internas conectadas a fuente (resistencias pull up) que se pueden activar o desactivar.

ENTRADAS

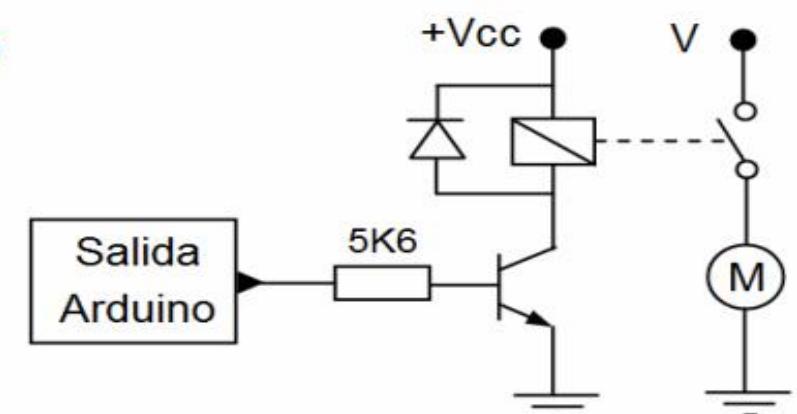
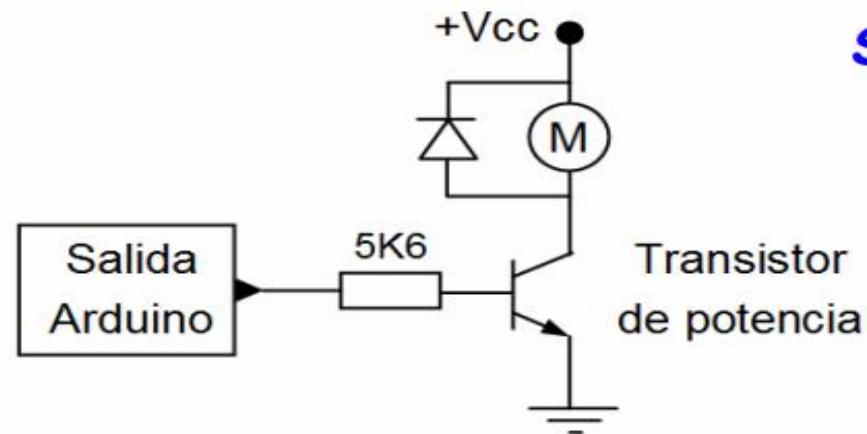


Protección de las salidas de Arduino

Los microcontroladores pueden suministrar en sus salidas **intensidades muy pequeñas**. Arduino, concretamente, unos 40 mA. Se deteriorarían si conectáramos en ellas actuadores que absorben intensidades mayores.

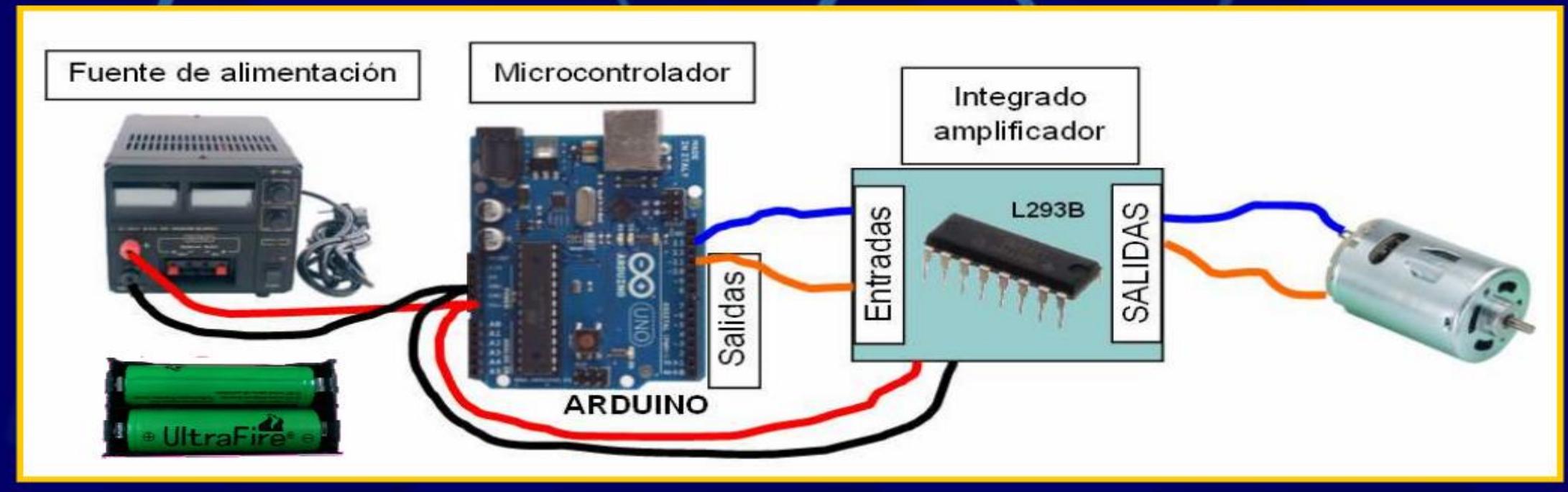
Una solución es conectar las salidas del microcontrolador a las bases de **transistores** en cuyos colectores se conectan los actuadores o los relés que los accionan.

SALIDAS



Protección de las salidas: integrados amplificadores

Otra solución para proteger las salidas de los microcontroladores es conectarlas a las entradas de unos chips que contienen **amplificadores**, los cuales les aportan a los actuadores las intensidades necesarias desde pilas o desde fuentes de alimentación y no desde las salidas del microcontrolador.



Diseño de control programado

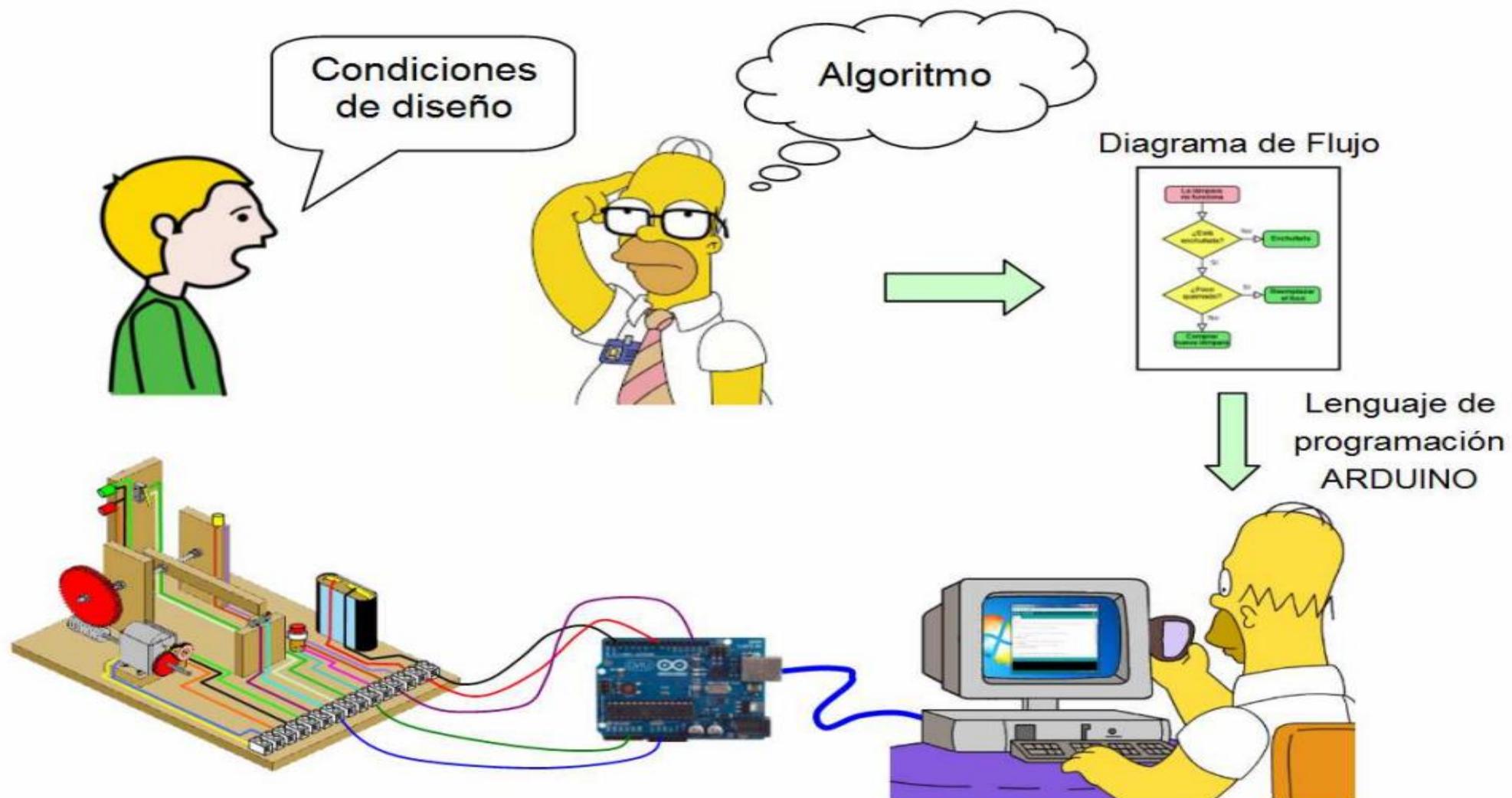
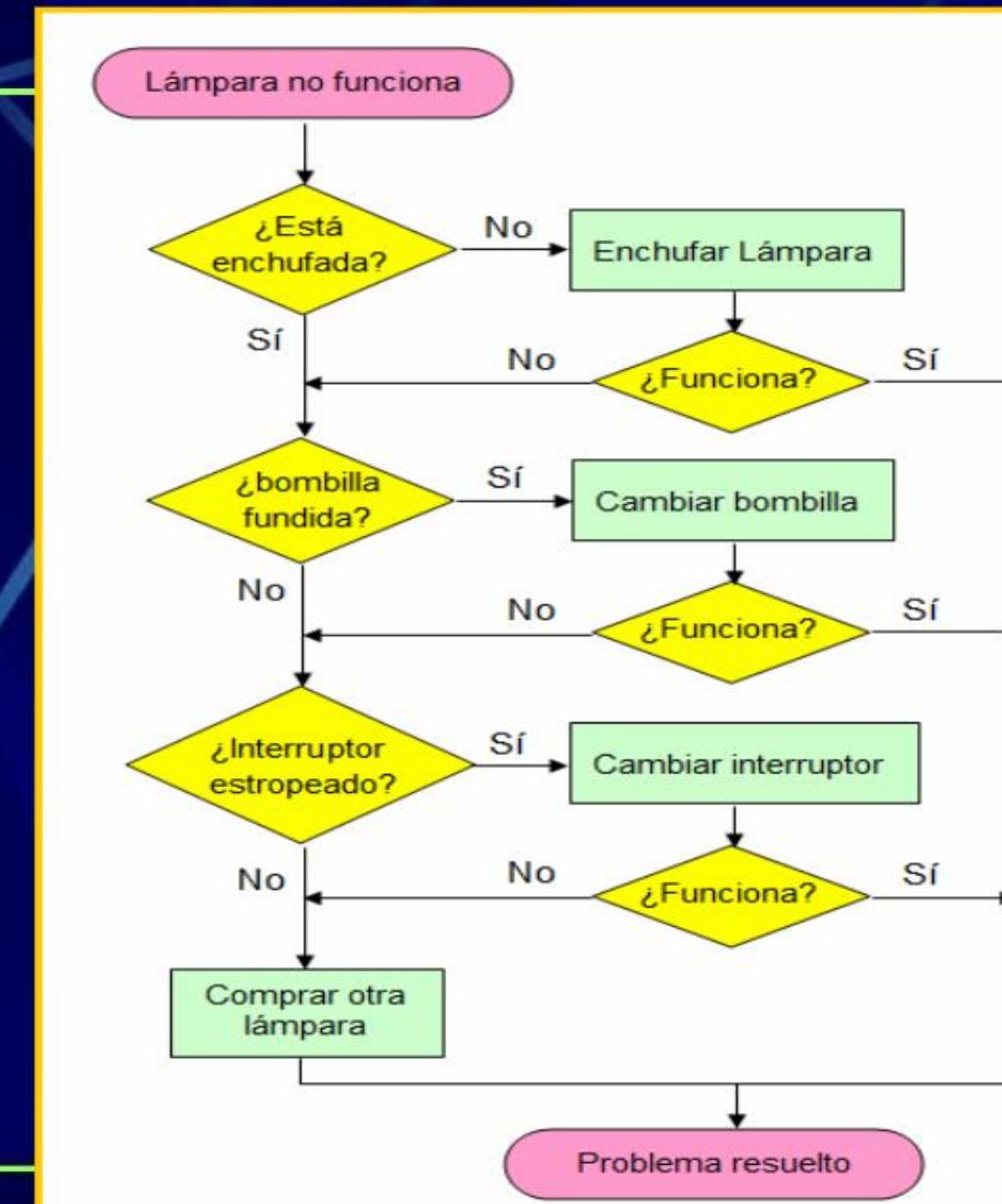


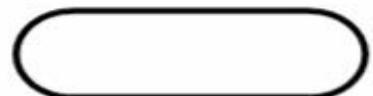
Diagrama de flujo

Un **diagrama de flujo** es una representación gráfica de un algoritmo, lo que facilita su diseño, su compresión y su traducción a un lenguaje de programación.

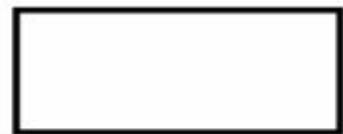


Algoritmo y diagrama de flujo

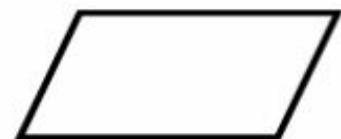
En los diagramas de flujo se utiliza una serie de **símbolos normalizados**:



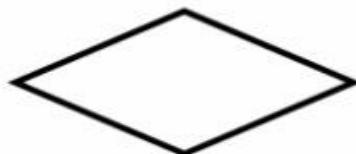
Inicio o fin de función



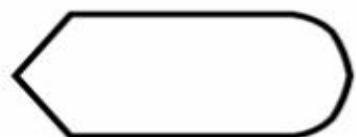
Procesamiento



Entrada o salida de datos



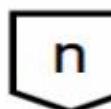
Decisión



Salida por pantalla

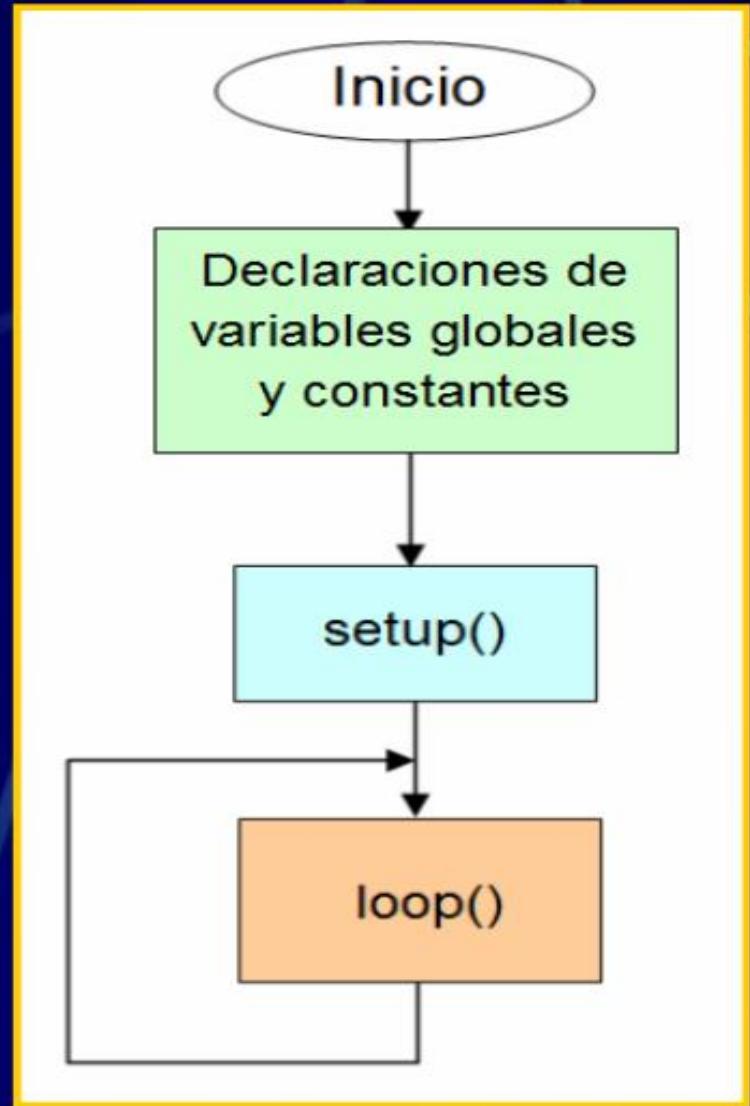


Conector en misma página



Conector en otra página

Esquema de funcionamiento de Arduino

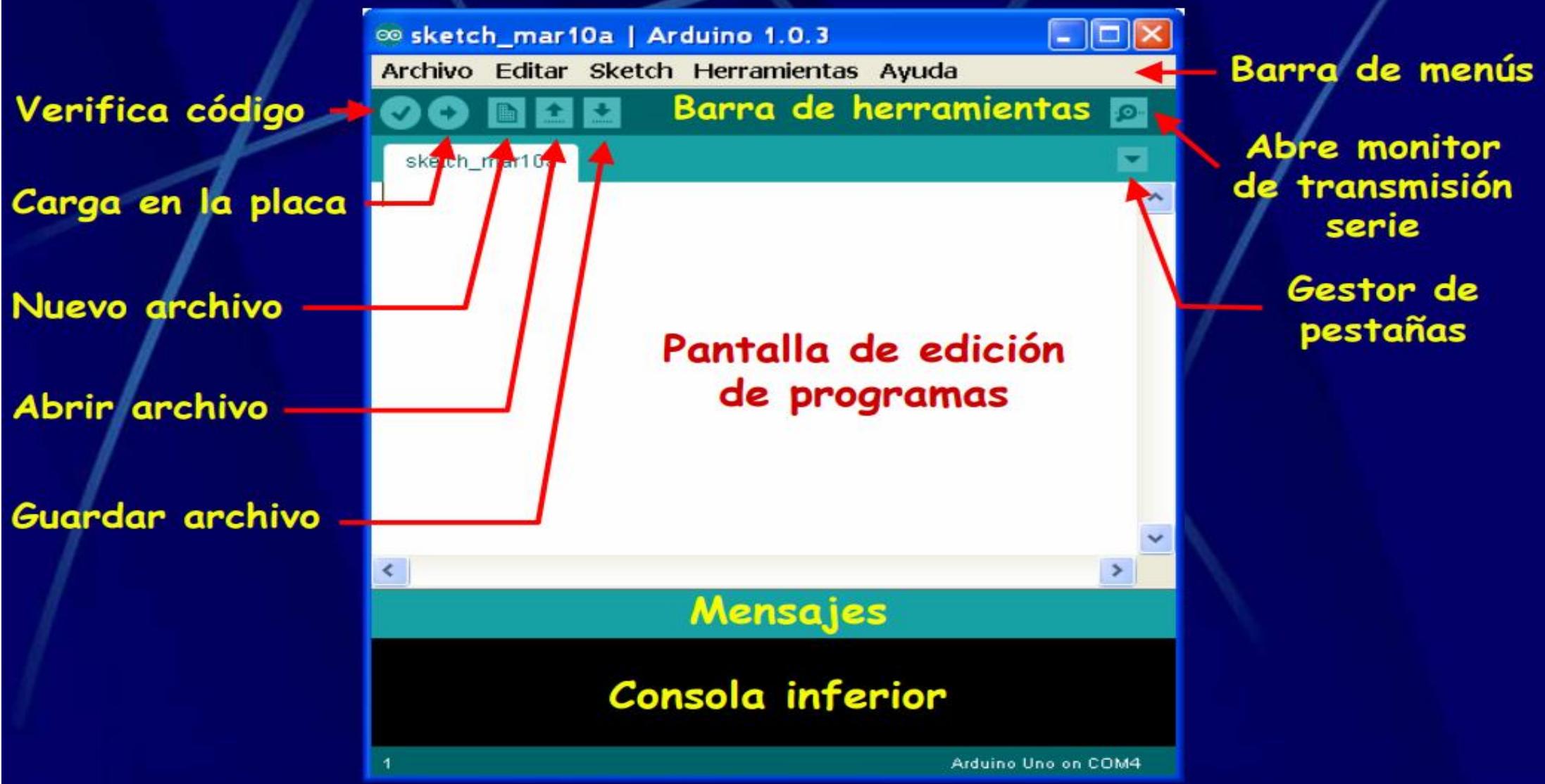


Las declaraciones de variables globales y de las constantes suelen colocarse delante de la función `setup()`

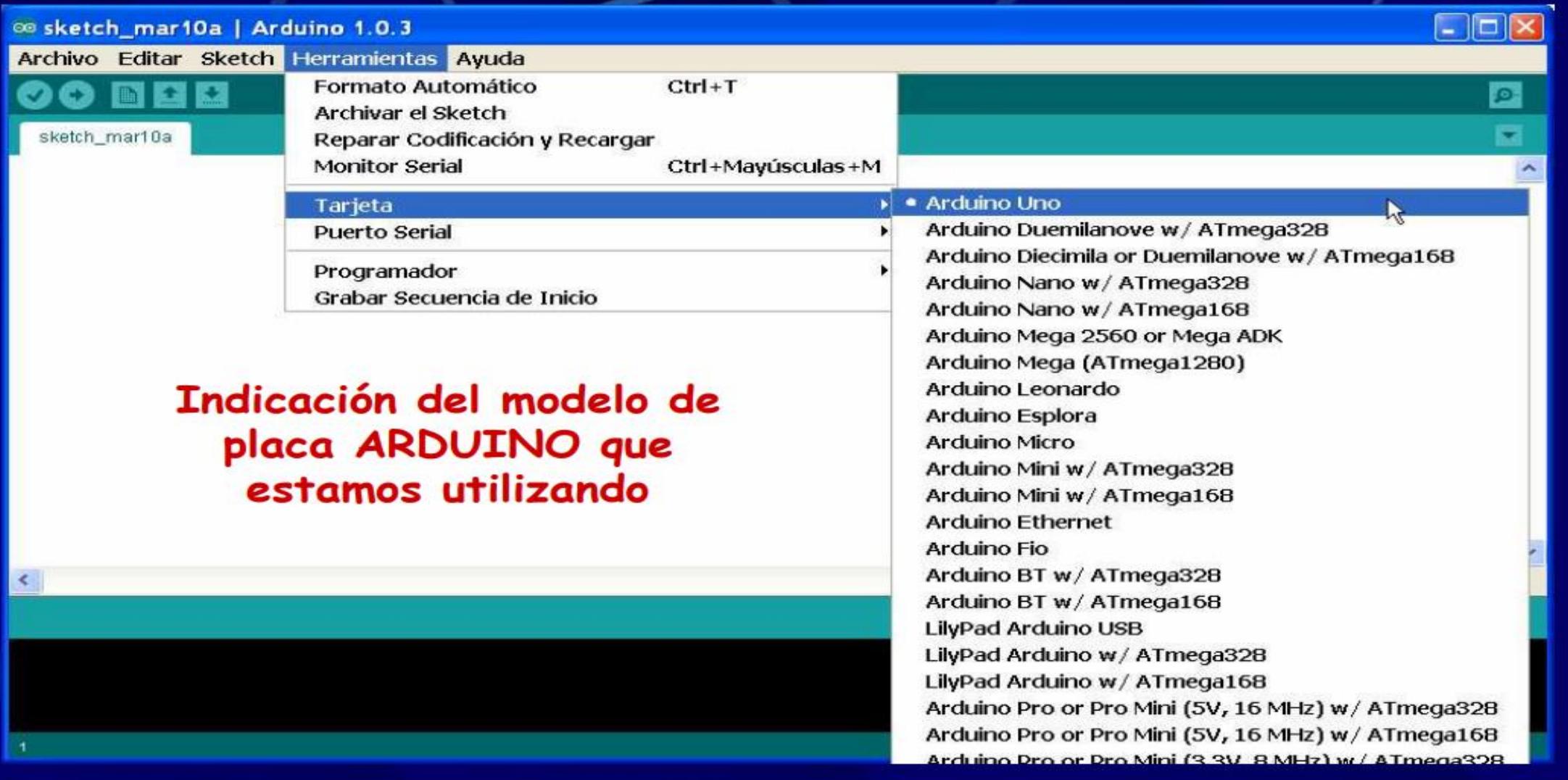
La función `setup()` se ejecuta una sola vez cuando alimentamos la placa o cada vez que se presiona el botón reset de la placa. En esta función se suelen incluir las definiciones del modo en que se usarán los pines.

A continuación se ejecuta la función `loop()` de forma cíclica hasta que se corta la alimentación o se presiona el botón reset de la placa.

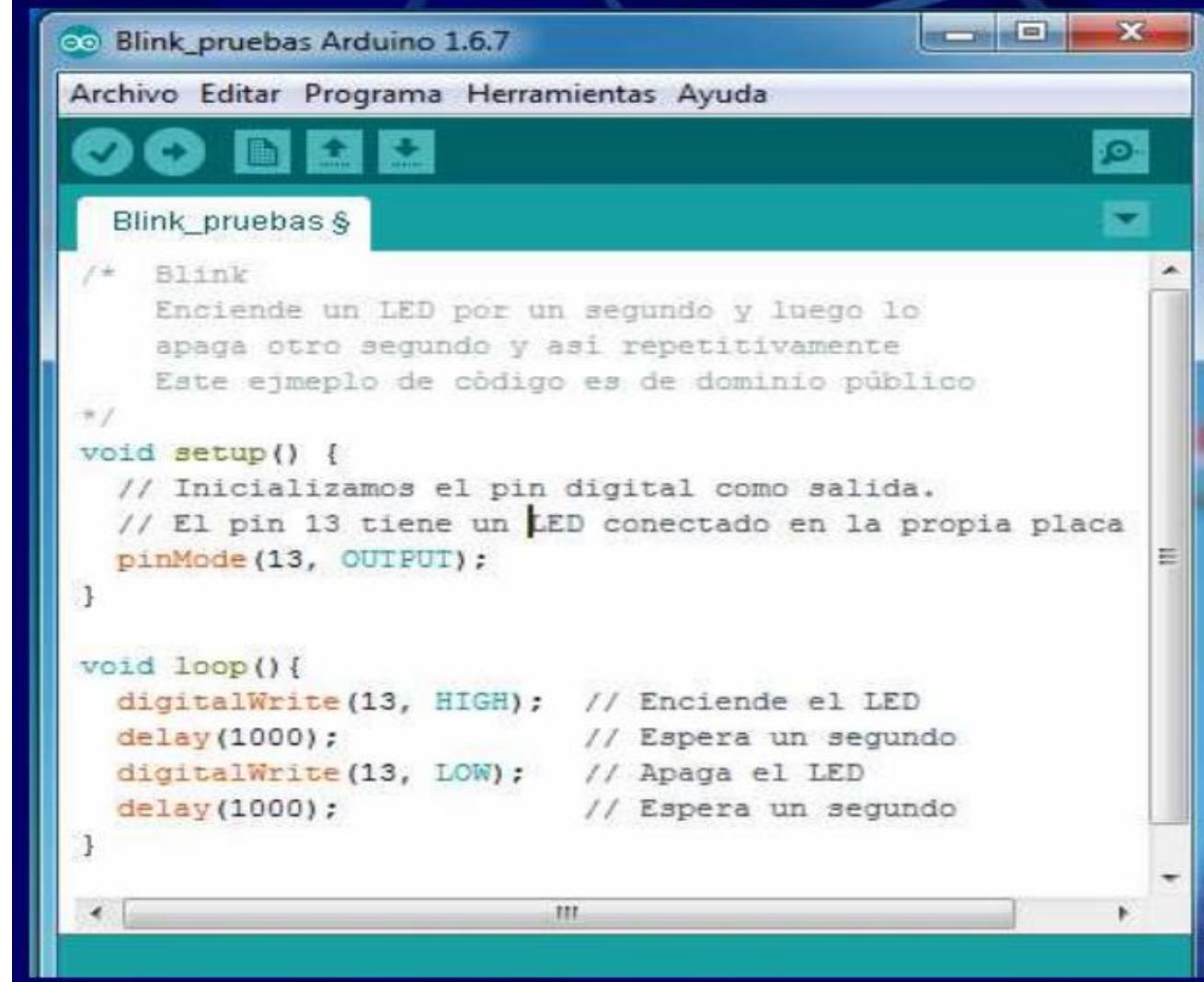
El entorno integrado de ARDUINO



Indicación del modelo de placa ARDUINO



Programación en ARDUINO: Comentarios, setup() y loop()



The screenshot shows the Arduino IDE interface with the title bar "Blink_pruebas Arduino 1.6.7". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main window displays the code for the "Blink" example:

```
/* Blink
   Enciende un LED por un segundo y luego lo
   apaga otro segundo y así repetitivamente
   Este ejemplo de código es de dominio público
*/
void setup() {
  // Inicializamos el pin digital como salida.
  // El pin 13 tiene un LED conectado en la propia placa
  pinMode(13, OUTPUT);
}

void loop(){
  digitalWrite(13, HIGH);    // Enciende el LED
  delay(1000);              // Espera un segundo
  digitalWrite(13, LOW);     // Apaga el LED
  delay(1000);              // Espera un segundo
}
```

Todos los programas deben contener como mínimo las funciones **setup** y **loop**.

Función setup, se ejecuta una sola vez. Se utiliza normalmente para definir las entradas y salidas.

Función loop, se ejecuta cíclicamente una y otra vez. Contiene el cuerpo del programa.

Comentarios, son notas o aclaraciones para hacer comprensible el programa. Aparecen en color gris.

Programación en ARDUINO: Los tipos de datos

The screenshot shows the Arduino IDE interface with the title bar "Blink_modificado_1 | Arduino 1.0.3". The menu bar includes "Archivo", "Editar", "Sketch", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for save, open, and run. The code editor contains the following code:

```
/* Blink
   Enciende un LED por un segundo y luego lo
   apaga otro segundo y así repetitivamente
   Este ejemplo de código es de dominio público
*/
int ledPin = 13; // LED conectado en el pin digital
//13 integrado en la placa Arduino
void setup() {
  // inicializamos el pin digital como salida.
  pinMode(ledPin, OUTPUT);
}
void parpadea(){
  digitalWrite(ledPin, HIGH); // Enciende el LED
  delay(1000); // Espera un segundo
  digitalWrite(ledPin, LOW); // Apaga el LED
  delay(1000); // Espera un segundo
}
void loop() {
  parpadea();
}
```

➤ Toda variable utilizada debe ser declarada previamente, indicando el tipo de datos que contendrá. Para las funciones se indica el tipo de datos que devuelve, si es el caso.

- **void**: sólo para funciones que no devuelven nada.
- **boolean**: true o false.
- **char**: caracteres.
- **int**: valores enteros cortos.
- **unsigned int**: enteros cortos sin signo.
- **long**: valores enteros largos.
- **unsigned long**: enteros largos sin signo.
- **float**: valores decimales.

Declarar variables

Declaración de Variables Enteras (int):

Para declarar una variable entera, utiliza la palabra clave "int" seguida del nombre de la variable y, opcionalmente, un valor inicial. Por ejemplo:

```
int contador = 0; // Declaración de una variable entera llamada "contador"  
con valor inicial 0
```

Declaración de Variables de Punto Flotante (float):

Para declarar una variable de punto flotante, utiliza la palabra clave "float" seguida del nombre de la variable y, opcionalmente, un valor inicial. Por ejemplo:

```
float temperatura = 25.5; // Declaración de una variable de punto flotante  
llamada "temperatura" con valor inicial 25.5
```

Declaración de Variables de Texto (String):

Para declarar una variable de texto, utiliza la palabra clave "String" seguida del nombre de la variable y, opcionalmente, un valor inicial. Por ejemplo:

```
String mensaje = "Hola, Arduino"; // Declaración de una variable de texto llamada  
"mensaje" con valor inicial "Hola, Arduino"
```

Declaración de Variables de Tipo Carácter (char):

Para declarar una variable de tipo carácter, utiliza la palabra clave "char" seguida del nombre de la variable y, opcionalmente, un valor inicial. Por ejemplo:

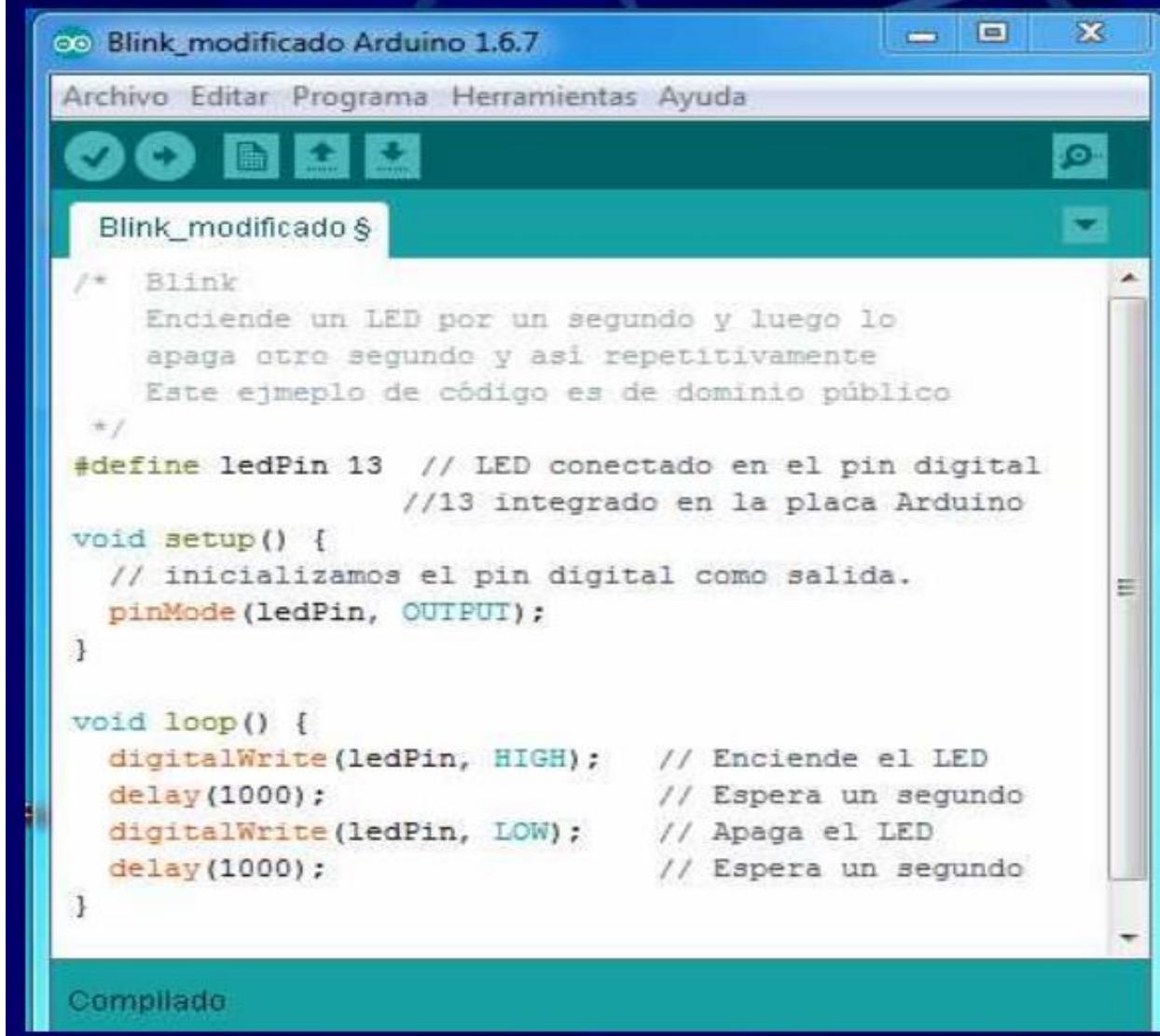
```
char letra = 'A'; // Declaración de una variable de tipo carácter llamada "letra" con  
valor inicial 'A'
```

Declaración de Variables Booleanas (bool):

Para declarar una variable booleana, utiliza la palabra clave "bool" seguida del nombre de la variable y, opcionalmente, un valor inicial. Por ejemplo:

```
bool estaEncendido = true; // Declaración de una variable booleana llamada  
"estaEncendido" con valor inicial verdadero (true)
```

Programación en ARDUINO: Constantes



The screenshot shows the Arduino IDE interface with the title bar "Blink_modificado Arduino 1.6.7". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main window displays the code for the "Blink_modificado" sketch. The code defines a constant `ledPin` as 13, which is used in the `setup()` and `loop()` functions to control an LED connected to pin 13.

```
/* Blink
   Enciende un LED por un segundo y luego lo
   apaga otro segundo y así repetitivamente
   Este ejemplo de código es de dominio público
*/
#define ledPin 13 // LED conectado en el pin digital
               // 13 integrado en la placa Arduino
void setup() {
  // inicializamos el pin digital como salida.
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);      // Enciende el LED
  delay(1000);                   // Espera un segundo
  digitalWrite(ledPin, LOW);       // Apaga el LED
  delay(1000);                   // Espera un segundo
}
```

Compilado

Para poder recordar mejor el uso que hagamos de los pines podemos asignarles nombres relacionados con dicho uso, así no tenemos que recordar los números.

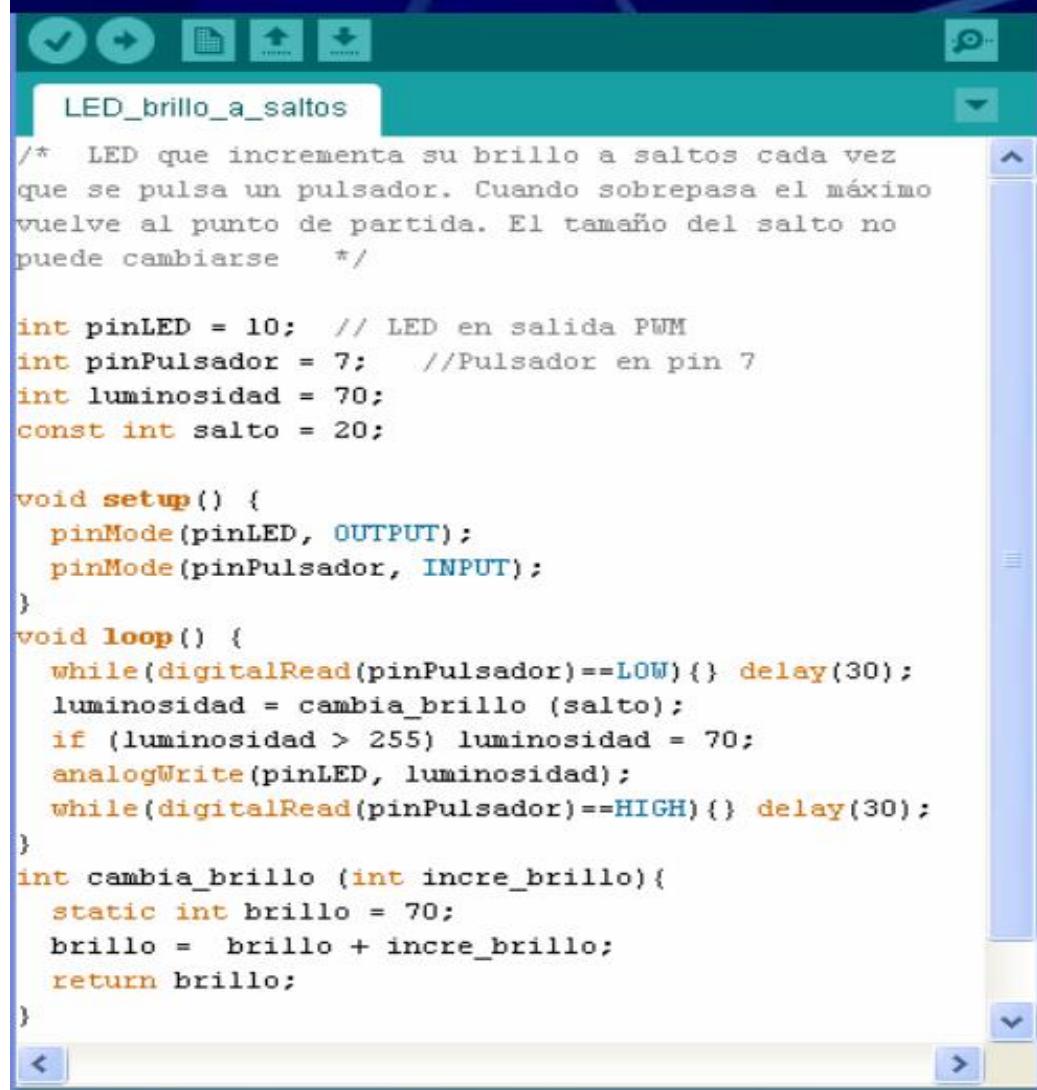
Estos nombres se llaman **constantes**, y se definen utilizando la instrucción

#define const número

Observa que esta instrucción no acaba en punto y coma (:)

Todas las constantes deben declararse antes de usarse.

Programación en ARDUINO: const



```
LED_brillo_a_saltos
/*
 * LED que incrementa su brillo a saltos cada vez
 * que se pulsa un pulsador. Cuando sobrepasa el máximo
 * vuelve al punto de partida. El tamaño del salto no
 * puede cambiarse */
int pinLED = 10; // LED en salida PWM
int pinPulsador = 7; //Pulsador en pin 7
int luminosidad = 70;
const int salto = 20;

void setup() {
  pinMode(pinLED, OUTPUT);
  pinMode(pinPulsador, INPUT);
}
void loop() {
  while(digitalRead(pinPulsador)==LOW){} delay(30);
  luminosidad = cambia_brillo (salto);
  if (luminosidad > 255) luminosidad = 70;
  analogWrite(pinLED, luminosidad);
  while(digitalRead(pinPulsador)==HIGH){} delay(30);
}
int cambia_brillo (int incre_brillo){
  static int brillo = 70;
  brillo = brillo + incre_brillo;
  return brillo;
}
```

➤ Si calificamos a una variable con la palabra clave **const**, la hacemos de sólo lectura, o sea, no permitimos que el programa pueda cambiar su valor.

const tipovar nombrevar=valor;

Programación en ARDUINO: Variables globales y locales



The screenshot shows the Arduino IDE interface with a sketch titled "pulsador_LED". The code is as follows:

```
/* Dispone de un LED y un pulsador. Cuando el pulsador
está pulsado enciende el LED y cuando no está pulsado
apaga el LED
*/
#define ledPin 13 // LED conectado al pin digital 13
#define pulPin 7 // pulsador conectado al pin digital 7
int val = 0; // variable para almacenar el valor leido

void setup()
{
    pinMode (ledPin, OUTPUT); // pin digital 13 como salida
    pinMode (pulPin, INPUT); // pin digital 7 como entrada
}

void loop()
{
    val = digitalRead (pulPin); // lee el pin de entrada
    digitalWrite (ledPin, val); // establece el LED al valor
                               // dado por el pulsador
}
```

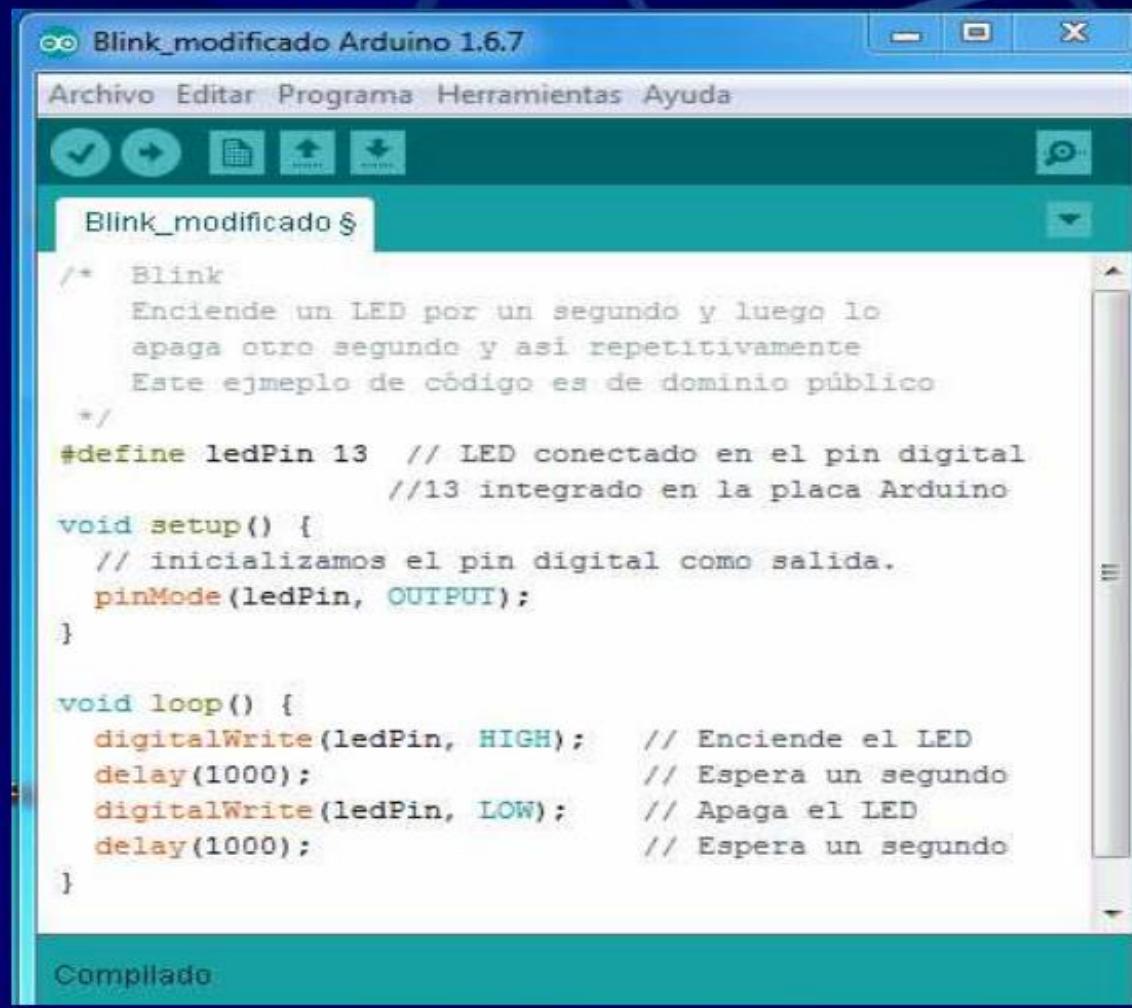
Una **variable** es un modo de nombrar y guardar un valor que puede variar para su uso posterior por el programa.

Cualquier variable debe ser declarada antes de utilizarse

➤ **Variables globales**, se declaran al inicio del programa, delante de la función **setup**. Pueden usarse en cualquier parte del programa.

➤ **Variables locales**, sólo pueden usarse dentro de la función en la que se declaran.

Programación en ARDUINO: Entradas / Salidas digitales



The screenshot shows the Arduino IDE interface with the title bar "Blink_modificado Arduino 1.6.7". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for save, upload, and other functions. The code editor window contains the "Blink_modificado" sketch. The code is as follows:

```
/* Blink
   Enciende un LED por un segundo y luego lo
   apaga otro segundo y así repetitivamente
   Este ejemplo de código es de dominio público
*/
#define ledPin 13 // LED conectado en el pin digital
               // 13 integrado en la placa Arduino

void setup() {
  // inicializamos el pin digital como salida.
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);    // Enciende el LED
  delay(1000);                 // Espera un segundo
  digitalWrite(ledPin, LOW);    // Apaga el LED
  delay(1000);                 // Espera un segundo
}
```

The status bar at the bottom says "Compilado".

➤ Los pines digitales de Arduino pueden funcionar tanto como entradas como salidas. El modo hay que declararlo previamente con la instrucción:

- **pinMode (pin, modo)**

El parámetro 'modo' puede adoptar los valores INPUT u OUTPUT.

➤ Se lee en una entrada digital con la función:

- **digitalRead (pin)**

➤ Se escribe en una entrada digital con la función:

- **digitalWrite (pin, valor)**

El parámetro 'valor' puede valer HIGH o LOW o valores equivalentes.

Programación en ARDUINO: Funciones de tiempo

The screenshot shows the Arduino IDE interface with a sketch titled "Imprime_tiempo". The code uses the `delay` function to print the time elapsed since the start of the program every second.

```
/* Imprime en el monitor serie del entorno integrado de Arduino el tiempo transcurrido desde el inicio del programa, en milisegundos. Se realiza una impresión cada segundo */

unsigned long tiempo; // variable para almacenar el tiempo transcurrido

void setup () {
    Serial.begin(9600); //abre el puerto serie
}

void loop () {
    Serial.print("Tiempo transcurrido: ");
    //Guarda en la variable el tiempo transcurrido
    tiempo = millis();
    // Imprime el tiempo y hace un salto de línea
    Serial.println (tiempo);
    // Espera un segundo antes de siguiente lectura
    delay(1000);
}
```

➤ Las funciones de tiempo permiten realizar temporizaciones en los programas.

- **delay (valor)**

Pausa el programa durante el número de milisegundos indicado por "valor".

- **millis ()**

Devuelve el número de milisegundos transcurridos desde que Arduino empezó a correr el programa actual.

➤ **Advertencia:** Tener en cuenta al usar `delay` que mientras el programa está pausado no hay lectura de entradas, por lo que si hay un cambio en éstas no será captado por la placa.

Encender un led

```
int ledPin = 10; // Pin del LED

void setup() {
    pinMode(ledPin, OUTPUT); // Configura el pin del LED como salida
}

void loop() {
    digitalWrite(ledPin, HIGH); // Enciende el LED
}
```

LED parpadeante:

```
int ledPin = 13; // Pin del LED

void setup() {
    pinMode(ledPin, OUTPUT); // Configura el pin del LED como salida
}

void loop() {
    digitalWrite(ledPin, HIGH); // Enciende el LED
    delay(1000); // Espera 1 segundo
    digitalWrite(ledPin, LOW); // Apaga el LED
    delay(1000); // Espera 1 segundo
}
```

Semáforo:

```
int redPin = 2;  
int yellowPin = 3;  
int greenPin = 4;  
  
void setup() {  
    pinMode(redPin, OUTPUT);  
    pinMode(yellowPin, OUTPUT);  
    pinMode(greenPin, OUTPUT);  
}
```

```
void loop() {  
    digitalWrite(redPin, HIGH);  
    delay(2000); // Luz roja durante 2 segundos  
    digitalWrite(redPin, LOW);  
  
    digitalWrite(yellowPin, HIGH);  
    delay(1000); // Luz amarilla durante 1 segundo  
    digitalWrite(yellowPin, LOW);  
  
    digitalWrite(greenPin, HIGH);  
    delay(2000); // Luz verde durante 2 segundos  
    digitalWrite(greenPin, LOW);  
}
```

Entradas y salidas digitales

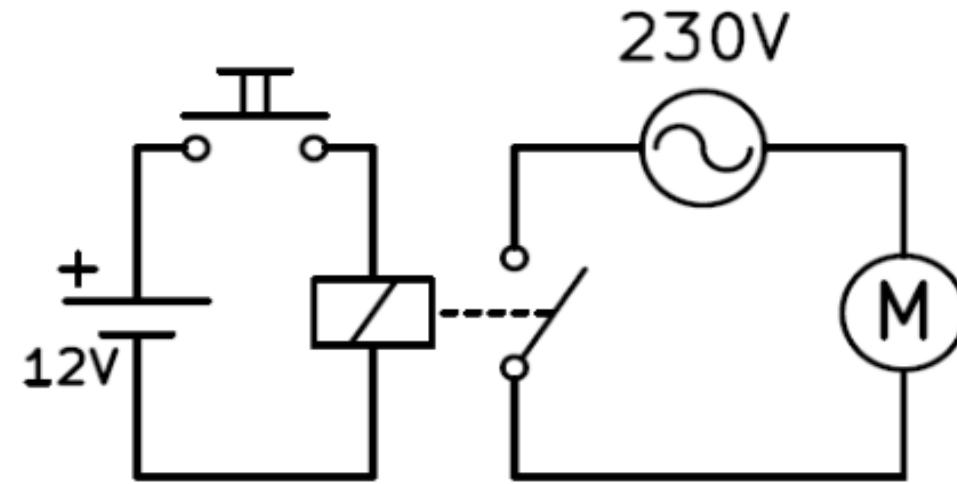
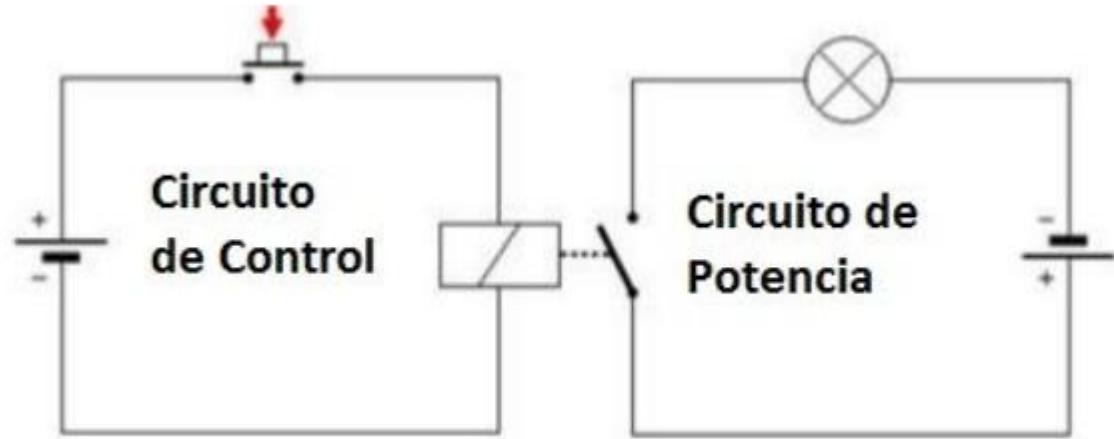
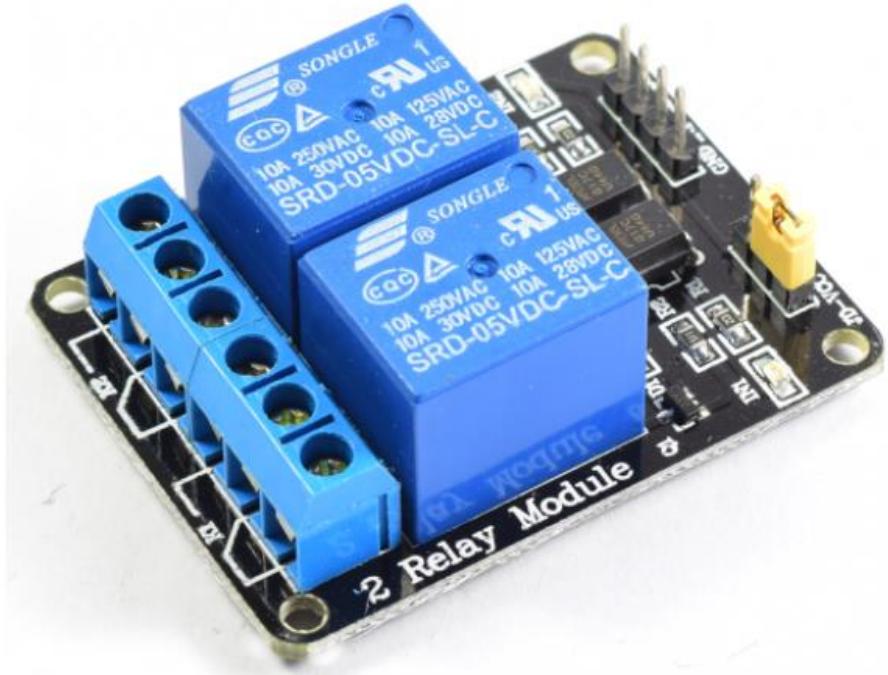
```
const int buttonPin1 = 2; // Primer botón, conectado al pin 2
const int buttonPin2 = 3; // Segundo botón, conectado al pin 3
const int ledPin1 = 4;   // Primer LED, conectado al pin 4
const int ledPin2 = 5;   // Segundo LED, conectado al pin 5

int buttonState1;
int buttonState2;

void setup() {
    pinMode(buttonPin1, INPUT); // Configura el primer botón como entrada
    pinMode(buttonPin2, INPUT); // Configura el segundo botón como entrada
    pinMode(ledPin1, OUTPUT);  // Configura el primer LED como salida
    pinMode(ledPin2, OUTPUT);  // Configura el segundo LED como salida
}
```

```
void loop() {  
    buttonState1 = digitalRead(buttonPin1); // Lee el estado del primer botón  
    buttonState2 = digitalRead(buttonPin2); // Lee el estado del segundo botón  
  
    digitalWrite(ledPin1, buttonState1); // Enciende/apaga el primer LED según  
    el estado del primer botón  
    digitalWrite(ledPin2, buttonState2); // Enciende/apaga el segundo LED  
    según el estado del segundo botón  
}
```

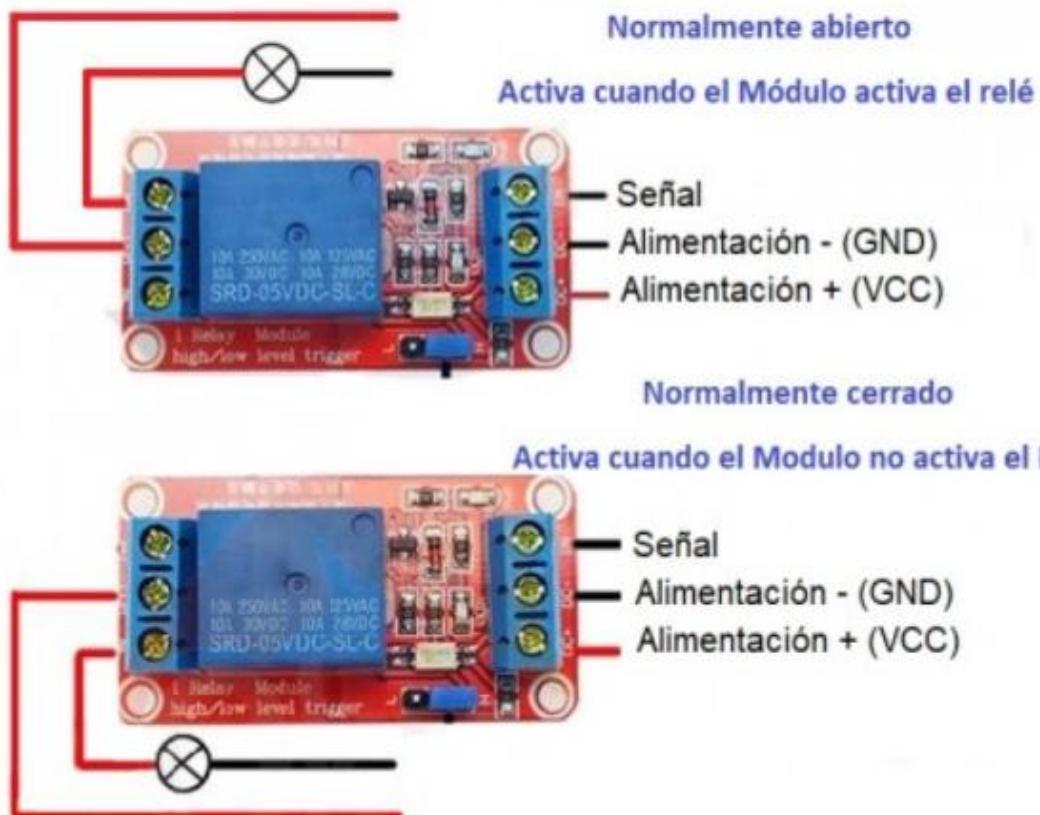
Módulo Relé



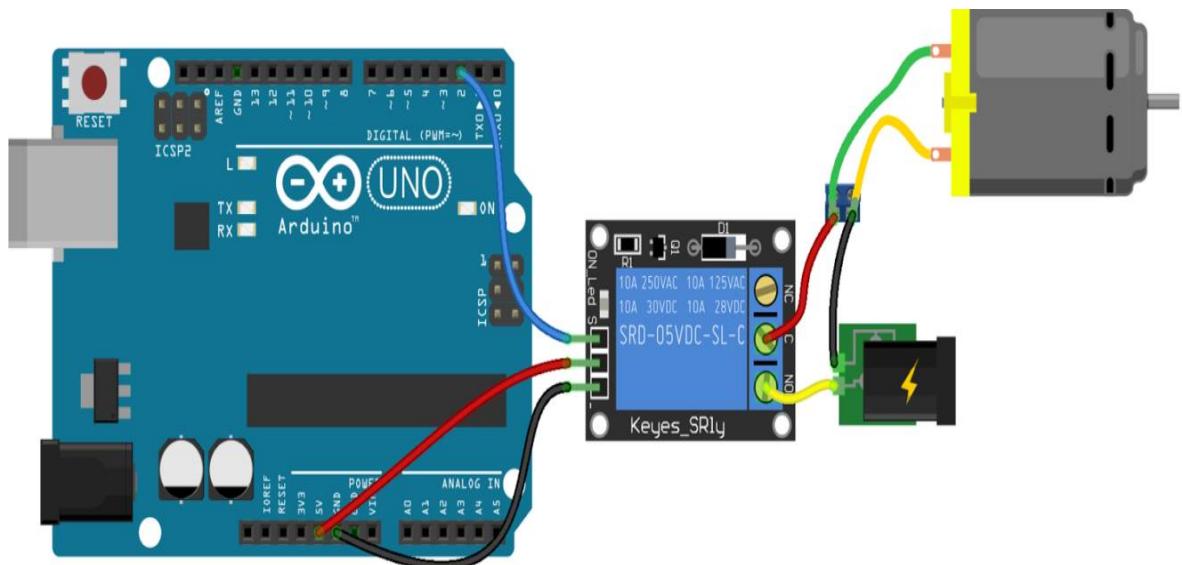
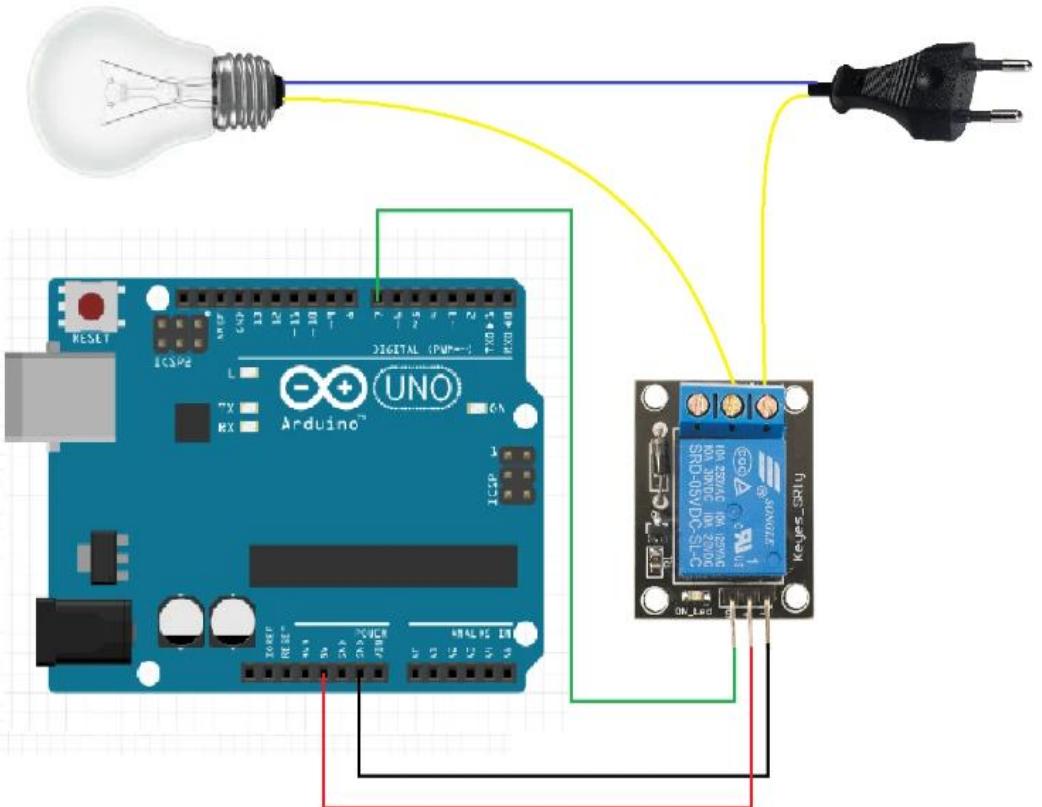
Módulo / Interface 1 Relé

5V / 9V / 12V / 24V Opto-acoplado

2 alternativas para conexión del equipo a controlar



⚠ Carga: Tensión máxima 250VAC.
Intensidad máxima 10A.



```
// Definir el pin de control del relé
int relePin = 7;

void setup() {
    // Configurar el pin del relé como salida
    pinMode(relePin, OUTPUT);
}

void loop() {
    // Encender el relé (cerrar el contacto)
    digitalWrite(relePin, HIGH);
    delay(1000); // Esperar 1 segundo

    // Apagar el relé (abrir el contacto)
    digitalWrite(relePin, LOW);
    delay(1000); // Esperar 1 segundo
}
```

Estructura de control

Sentencia IF

En Arduino, como en la mayoría de los lenguajes de programación, "if" es una estructura de control que se utiliza para tomar decisiones en función de condiciones específicas. El "if" en Arduino permite ejecutar un bloque de código si una condición dada se evalúa como verdadera (true). La estructura básica de un "if" en Arduino es la siguiente:

```
if (condición) {  
    // Código a ejecutar si la condición es verdadera  
}
```

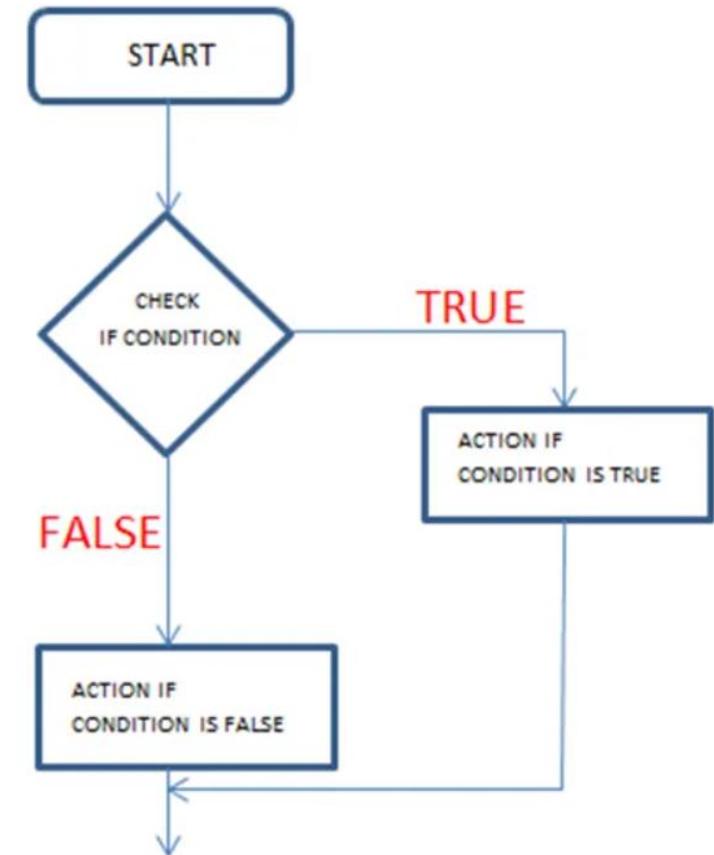


Diagrama de flujo de la sentencia if

Programación en ARDUINO: Estructura condicional if

The screenshot shows the Arduino IDE interface with a sketch named "Encendido_LED_pulsador". The code defines pins 4, 7, and 8 as outputs and inputs with pull-up resistors. It sets up pin 4 as an output and pins 7 and 8 as inputs. In the loop function, it reads the state of both inputs. If input 7 is LOW, it turns on the LED connected to pin 4. If input 8 is LOW, it turns off the LED.

```
#define pinLV 4
#define pinE1 7
#define pinE2 8
int estadointerruptorE1;
int estadointerruptorE2;

void setup() {
  pinMode(pinLV, OUTPUT);
  pinMode(pinE1, INPUT_PULLUP);
  pinMode(pinE2, INPUT_PULLUP);
}

void loop() {
  estadointerruptorE1=digitalRead(pinE1);
  if(estadointerruptorE1==LOW){
    digitalWrite(pinLV, HIGH);
  }
  estadointerruptorE2=digitalRead(pinE2);
  if(estadointerruptorE2==LOW){
    digitalWrite(pinLV, LOW);
  }
}
```

➤ La estructura if decide si ejecutar o no una o varias instrucciones en función de que se cumpla una condición.

if (condición) instrucción;

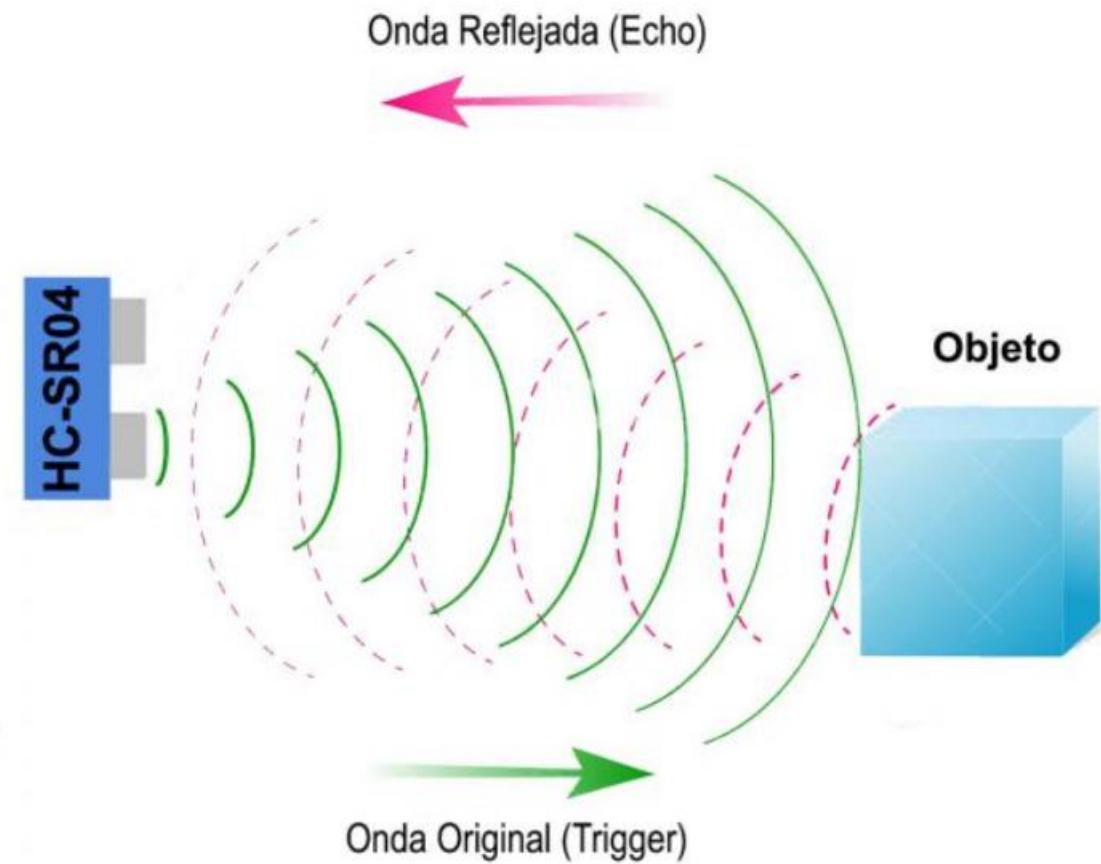
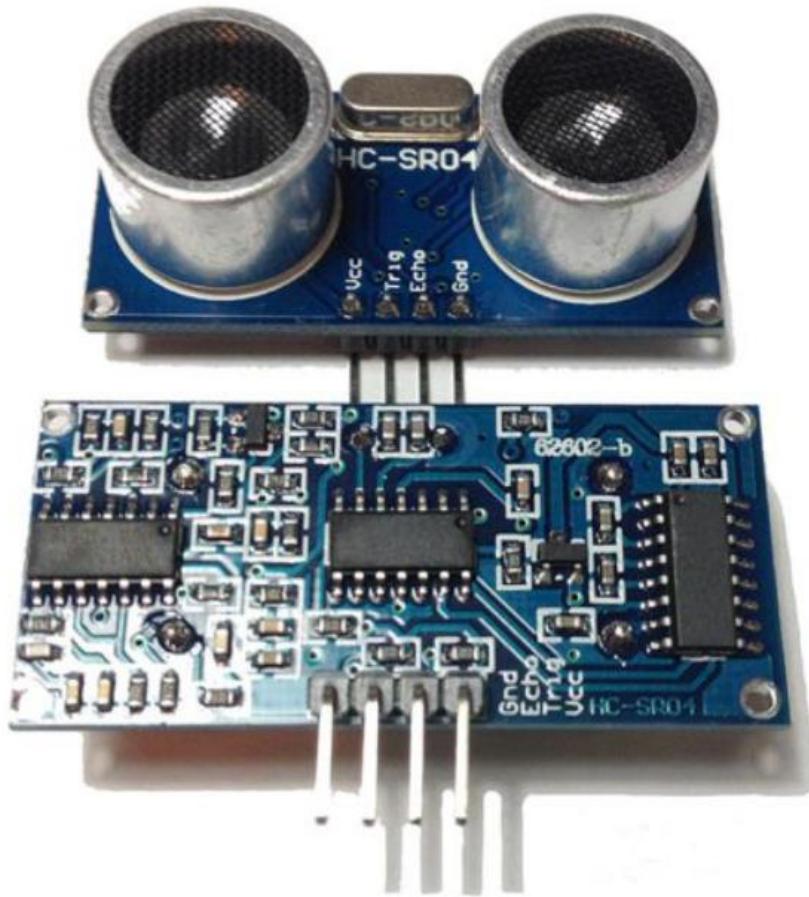
```
if (condición) {
  instrucción_1;
  instrucción_2;...
}
```

➤ Para expresar la condición se utilizan operadores:

- de comparación: ==, !=, <, >, <=, >=

- booleanos: &&, ||, !

Diagrama esquemático sensor ultrassónico arduino



Características	
Alimentación	+5v DC
Frecuencia de trabajo	40 KHz
Consumo (suspendido)	< 2mA
Consumo (trabajando)	15mA
Ángulo efectivo	< 15°
Distancia	2cm a 400cm *
Resolución	0.3 cm

*A partir de 250cm la resolución no es buena

$$\text{Velocidad} = \frac{\text{Espacio}}{\text{Tiempo}} \longrightarrow \text{Espacio} = \text{Velocidad} \times \text{Tiempo}$$

$$\text{Velocidad del sonido} = 343 \text{ m/s} = 0.0343 \text{ cm/}\mu\text{s}$$

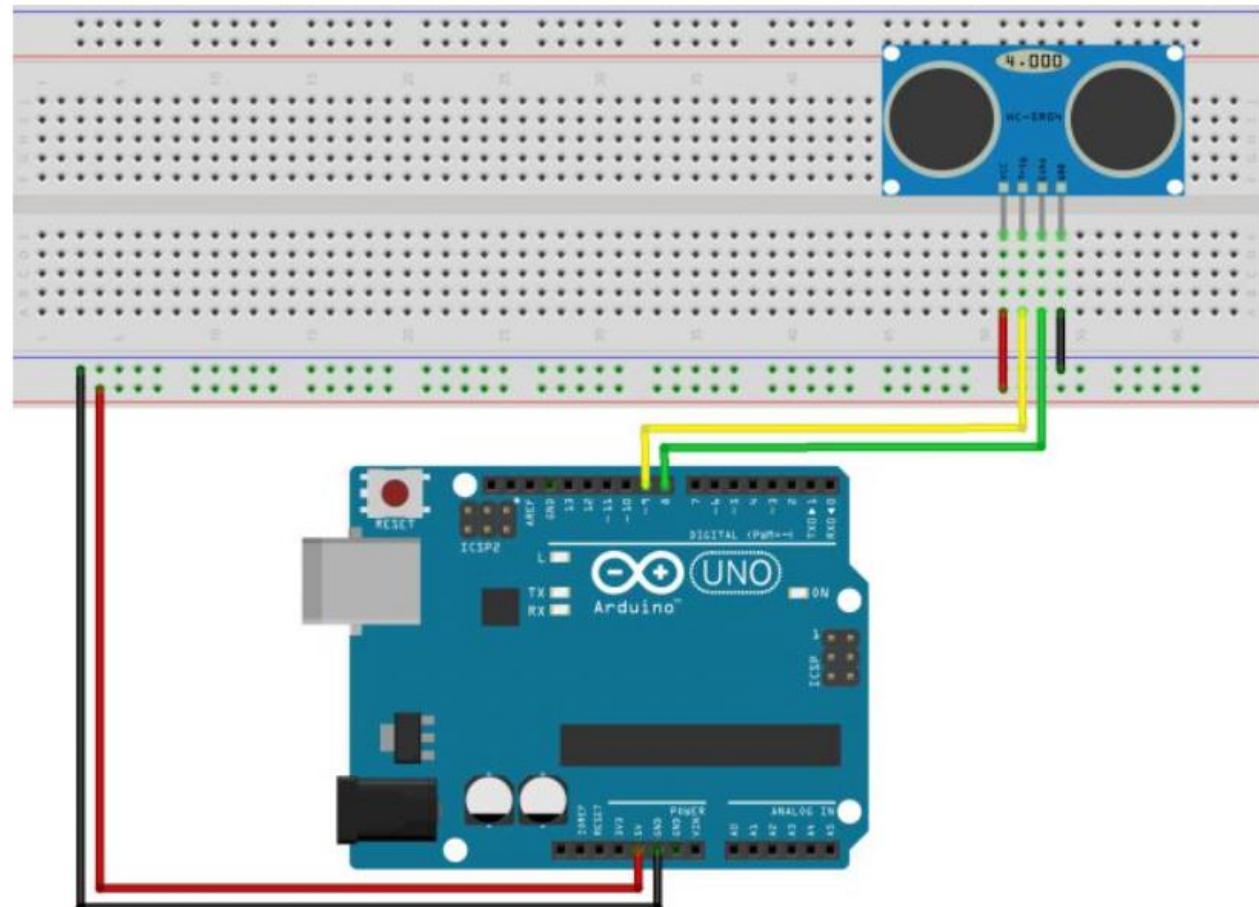
$$\text{Espacio} = 0.0343 \times \text{Tiempo}$$

*Pero como la onda ha recorrido el camino dos veces (ida y vuelta) hay que dividir entre dos para conocer la distancia a la que se encuentra el objeto.

$$\boxed{\text{Espacio} = 0.01715 \times \text{Tiempo}}$$

Ejemplo para medir distancias

El montaje es muy sencillo, tan solo hay que suministrar alimentación al modulo de ultrasonidos (conectar Vcc a +5 y GND a masa) y los pines de "echo" y "triger" conectarlos a los pines 8 y 9 respectivamente.



Programación en ARDUINO: Comunicación serie

The screenshot shows the Arduino IDE interface with the sketch titled 'lectura_analogica'. The code reads an analog input from pin 0 and prints its value to the serial monitor at 9600 bps. It includes comments explaining the purpose of each section.

```
lectura_analogica | Arduino 1.0.3
Archivo Editar Sketch Herramientas Ayuda
lectura_analogica
/*
 * Lee una entrada analógica del pin analógico 0 e
 * imprime el valor en el monitor serie a través
 * del puerto serie
 */

int analogValue = 0; // variable para almacenar
                     // el valor analógico

void setup() {
    // Abre el puerto serie a 9600 bps de velocidad
    Serial.begin (9600);
}

void loop() {
    // Lee la entrada analógica en el pin 0
    analogValue = analogRead(0);
    // Escribe el texto Valor leído delante del valor
    Serial.print ("Valor leído: ");
    // imprime el valor y hace un salto de línea
    Serial.println(analogValue);
    delay(10); // Retardo de 10 milisegundos
               //antes de la siguiente lectura
}
Guardado Terminado.
```

➤ La placa Arduino puede comunicarse en modo serie con el ordenador a través del puerto USB o con otro Arduino a través de los pines digitales 0 y 1.

➤ Podemos utilizar el monitor serial del IDE de Arduino, con las siguientes funciones:

- **Serial.begin (valor)**

Abre el puerto y establece la velocidad de transmisión.

- **Serial.print (valor)**

Imprime el valor en el monitor serial sin salto de línea.

- **Serial.println (valor)**

Imprime y añade un salto de línea.

Librería

- Abre el Arduino IDE.
- Ve al menú "Sketch" y selecciona "Incluir Biblioteca" -> "Gestor de Bibliotecas..."
- En el cuadro de búsqueda, escribe "NewPing" y presiona "Instalar" cuando aparezca la librería "NewPing" en los resultados.
- Una vez que la librería esté instalada, podrás usarla en tus proyectos.

```
#include <NewPing.h> // Incluye la librería NewPing que permite usar un sensor ultrasónico

#define TRIGGER_PIN 9 // Define el pin del trigger (emisor) del sensor ultrasónico
#define ECHO_PIN    8 // Define el pin del echo (receptor) del sensor ultrasónico
#define MAX_DISTANCE 200 // Define la distancia máxima de medición en centímetros

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // Crea un objeto "sonar" de la clase
                                                    NewPing

void setup() {
  Serial.begin(9600); // Inicializa la comunicación serial a 9600 baudios
}
```

```
void loop() {
    delay(500); // Espera medio segundo entre mediciones
    unsigned int distancia = sonar.ping_cm(); // Realiza una medición en centímetros
    utilizando el sensor ultrasónico

    if (distancia == 0) {
        Serial.println("Fuera de rango"); // Si la distancia es igual a 0, imprime "Fuera de rango"
    } else {
        Serial.print("Distancia: ");
        Serial.print(distancia);
        Serial.println(" cm"); // Si la distancia no es 0, imprime la distancia en centímetros
    }
}
```

Programación en ARDUINO: Estructura condicional if...else

The screenshot shows the Arduino IDE interface with a sketch titled "Encendido_LED_pulsador". The code defines two pins: pinLV (4) as an output and pinE1 (7) as an input with a pull-up resistor. It sets up the pins and then enters a loop where it reads the state of pinE1. If the state is LOW, it turns pinLV HIGH; otherwise, it turns pinLV LOW. The status bar at the bottom indicates the code has been saved.

```
#define pinLV 4
#define pinE1 7
int estadointerruptor;

void setup() {
  pinMode(pinLV, OUTPUT);
  pinMode(pinE1, INPUT_PULLUP);
}

void loop() {
  estadointerruptor=digitalRead(pinE1);
  if(estadointerruptor==LOW){
    digitalWrite(pinLV, HIGH);
  }
  else{
    digitalWrite(pinLV, LOW);
  }
}
```

➤ La estructura **if...else** decide ejecutar unas instrucciones u otras en función de que se cumpla una condición.

```
if (condición) instrucción_A;
else instrucción_B;
```

```
if (condición) {
  instrucciones_A; }
else {
  instrucciones_B; }
```

➤ A **else** le puede seguir otros **if**, ejecutándose múltiples pruebas.

```
if (condición1) instrucción_A;
else if (condición2) instrucción_B;
else instrucción_C;
```

Diagrama de flujo de la estructura condicional if....else

```
if (condición) {  
    instrucciones_A; }  
else {  
    instrucciones_B; }
```

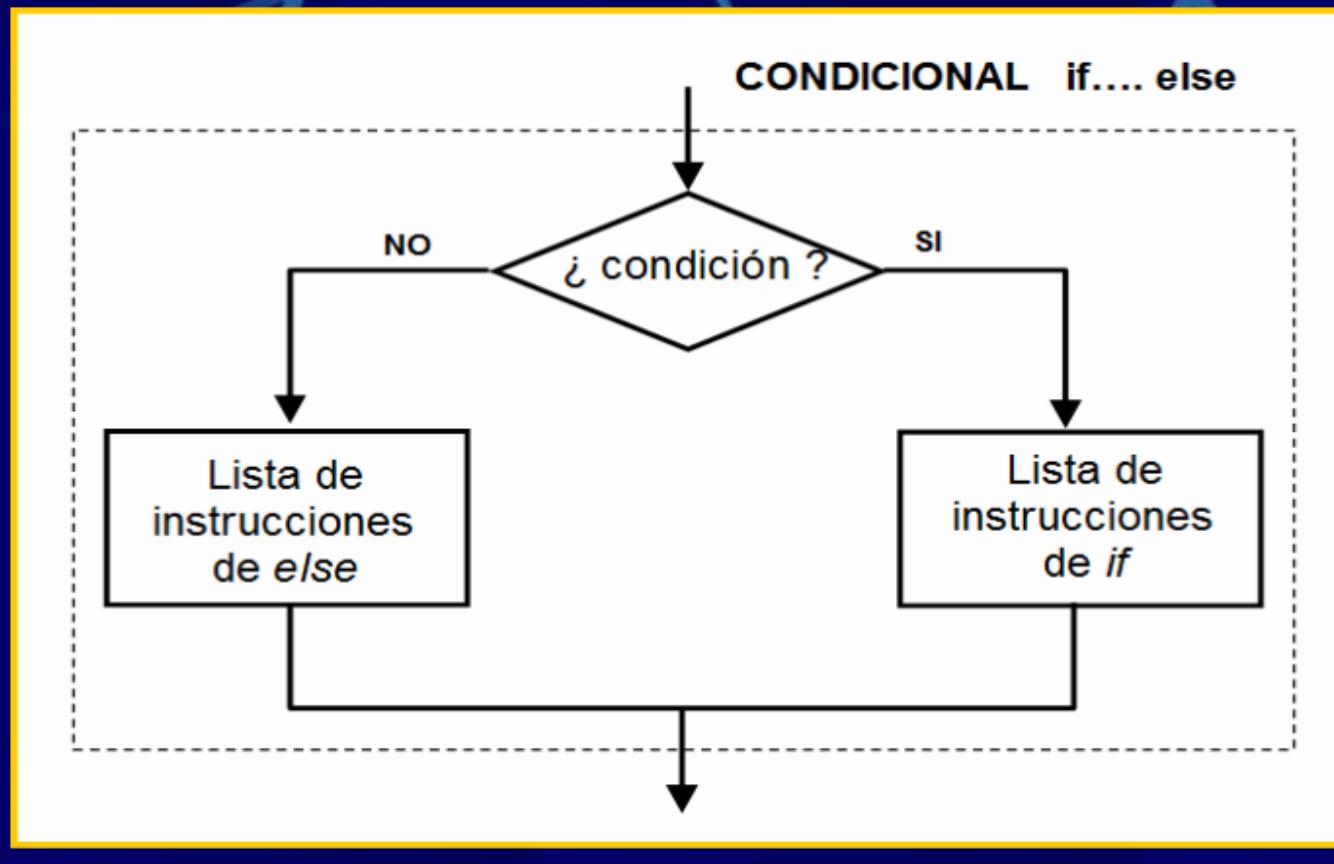
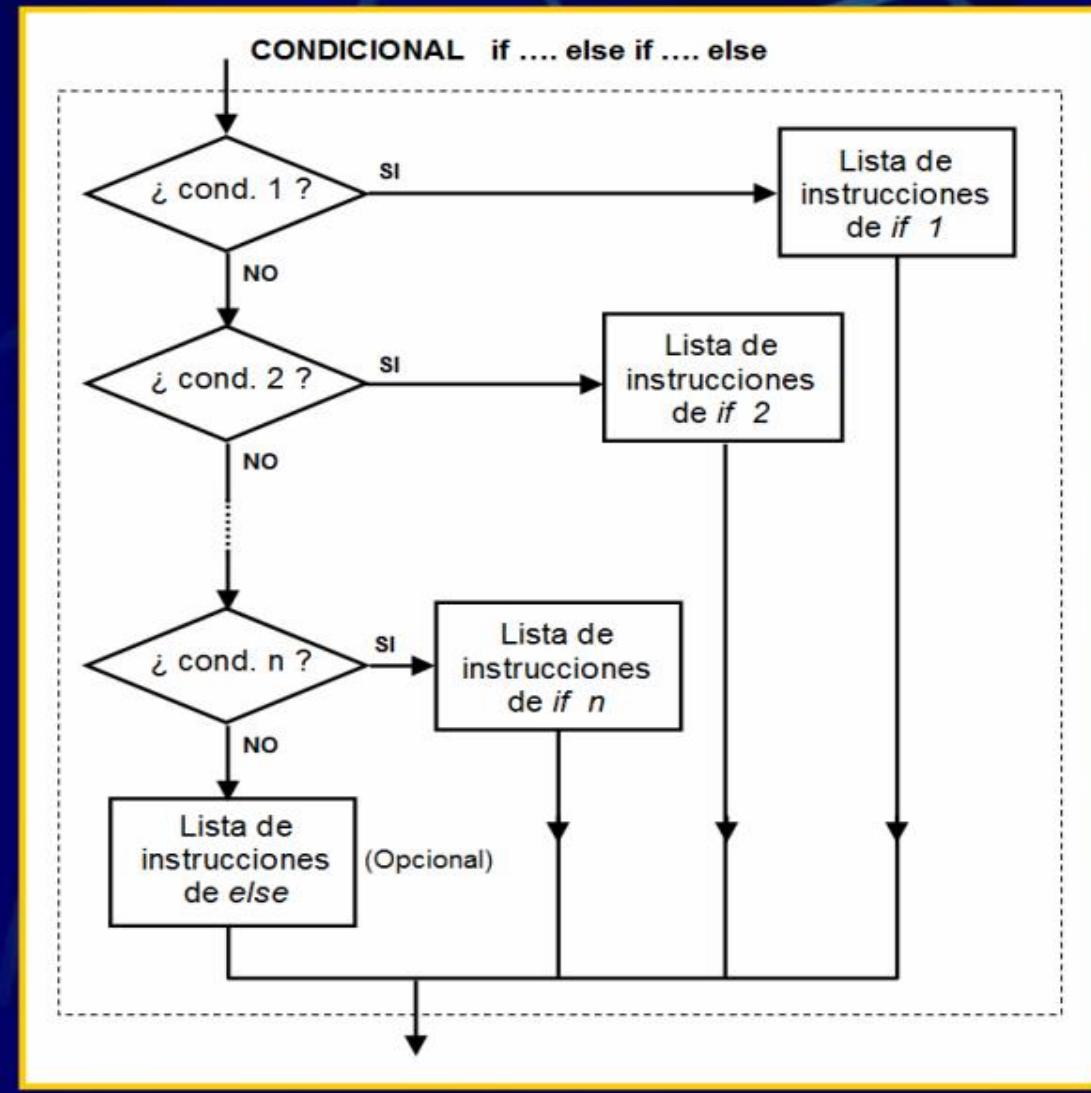


Diagrama de flujo de la estructura if....else if... else



```
if (condición_1) {  
    instrucciones_1; }  
else if (condición_2) {  
    instrucciones_2; }  
:  
:  
:  
else if (condición_N) {  
    instrucciones_N; }  
else { //es opcional  
    instrucciones_else; }
```

Medimos varias distancias

```
#include <NewPing.h> // Incluye la librería NewPing que permite usar un sensor ultrasónico
#define TRIGGER_PIN 9 // Define el pin del trigger (emisor) del sensor ultrasónico
#define ECHO_PIN 8 // Define el pin del echo (receptor) del sensor ultrasónico
#define MAX_DISTANCE 200 // Define la distancia máxima de medición en centímetros
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // Crea un objeto "sonar" de la clase NewPing
const int ledPinGreen = 2; // Pin del LED verde
const int ledPinYellow = 3; // Pin del LED amarillo
const int ledPinRed = 4; // Pin del LED rojo
void setup() {
    Serial.begin(9600); // Inicializa la comunicación serial a 9600 baudios
    pinMode(ledPinGreen, OUTPUT); // Configura el LED verde como salida
    pinMode(ledPinYellow, OUTPUT); // Configura el LED amarillo como salida
    pinMode(ledPinRed, OUTPUT); // Configura el LED rojo como salida
}
```

```
void loop() {  
    delay(500); // Espera medio segundo entre mediciones  
    unsigned int distancia = sonar.ping_cm(); // Realiza una medición en centímetros utilizando el sensor  
    ultrasónico  
  
    if (distancia == 0) {  
        Serial.println("Fuera de rango"); // Si la distancia es igual a 0, imprime "Fuera de rango"  
    } else {  
        Serial.print("Distancia: ");  
        Serial.print(distancia);  
        Serial.println(" cm"); // Si la distancia no es 0, imprime la distancia medida en centímetros
```

```
if (distancia <= 30) {  
    digitalWrite(ledPinGreen, LOW); // Apaga el LED verde  
    digitalWrite(ledPinYellow, LOW); // Apaga el LED amarillo  
    digitalWrite(ledPinRed, HIGH); // Enciende el LED rojo  
} else if (distancia > 30 && distancia <= 60) {  
    digitalWrite(ledPinGreen, LOW); // Apaga el LED verde  
    digitalWrite(ledPinYellow, HIGH); // Enciende el LED amarillo  
    digitalWrite(ledPinRed, LOW); // Apaga el LED rojo  
} else {  
    digitalWrite(ledPinGreen, HIGH); // Enciende el LED verde  
    digitalWrite(ledPinYellow, LOW); // Apaga el LED amarillo  
    digitalWrite(ledPinRed, LOW); // Apaga el LED rojo  
}  
}  
}
```

Programación en ARDUINO: Entradas analógicas

The screenshot shows the Arduino IDE interface with the title bar 'LED_brillo_graduado | Arduino 1.0.3'. The menu bar includes 'Archivo', 'Editar', 'Sketch', 'Herramientas', and 'Ayuda'. Below the menu is a toolbar with icons for saving, opening, and running the sketch. The code editor contains the following sketch:

```
/* Establece el brillo de un LED proporcionalmente
al valor de tensión leído en el terminal intermedio
de un potenciómetro conectado al pin analógico 3 */

int ledPin = 9;      // LED conectado al pin digital 9
int analogPin = 3;   // potenciómetro conectado al pin
                     // analógico 3
int val = 0;         // variable para almacenar el
                     // valor de tensión leido

void setup () {
    pinMode (ledPin, OUTPUT); // pin como salida
}

void loop () {
    // lee el pin de entrada analógica:
    val = analogRead (analogPin);
    analogWrite (ledPin, val/4);
    // divide el valor leido entre 4 para escalarlo.
    // los valores de analogRead van de 0 a 1023,
    // los valores de analogWrite de 0 a 255
}
```

- Los pines analógicos de Arduino pueden funcionar como entradas analógicas o como pines digitales iguales a los otros (llevan una A delante del número para distinguirlos: A0,A1,...,A5).
 - No pueden funcionar como salidas analógicas.
 - Se lee en una entrada analógica con la función:
 - **analogRead (pin)**
- Devuelve un valor entre 0 y 1023 que corresponden a tensiones entre 0 V y 5 V respectivamente.

Programación en ARDUINO: Salidas analógicas

The screenshot shows the Arduino IDE interface with a sketch named "LED_brillo_graduado". The code is as follows:

```
/* Establece el brillo de un LED proporcionalmente
al valor de tensión leido en el terminal intermedio
de un potenciómetro conectado al pin analógico 3 */

int ledPin = 9;      // LED conectado al pin digital 9
int analogPin = 3;   // potenciómetro conectado al pin
                     // analógico 3
int val = 0;         // variable para almacenar el
                     // valor de tensión leido

void setup () {
    pinMode (ledPin, OUTPUT); // pin como salida
}

void loop () {
    // lee el pin de entrada analógica:
    val = analogRead (analogPin);
    analogWrite (ledPin, val/4);
    // divide el valor leído entre 4 para escalarlo.
    // los valores de analogRead van de 0 a 1023,
    // los valores de analogWrite de 0 a 255
}
```

- En realidad Arduino no tiene salidas analógicas sino que simula un nivel de tensión analógico entre 0 V y 5 V con una señal digital cuadrada con anchura de pulso modulada (PWM).
 - En la placa Arduino UNO, los pines digitales que se pueden usar para este tipo de salidas son: 3, 5, 6, 9, 10 y 11.
 - Se escribe un valor analógico en una salida digital con la función:
 - **analogWrite (pin, valor)**
- El parámetro “valor” debe estar comprendido entre 0 y 255, que corresponden a tensiones de 0 V a 5 V respectivamente.

- El Arduino Uno tiene salidas digitales y entradas analógicas, pero no tiene salidas verdaderamente analógicas. En lugar de eso, las salidas se generan utilizando modulación por ancho de pulsos (PWM).
- El Arduino Uno tiene 6 pines que pueden generar señales PWM. Estos pines son: 3, 5, 6, 9, 10 y 11. A través de la función `analogWrite()`, puedes controlar la intensidad de una señal en estos pines. Aunque esta técnica se utiliza comúnmente para controlar la velocidad de motores, la intensidad de un LED o la posición de un servo, en realidad no produce una señal analógica continua, sino que genera una serie de pulsos que varían en anchura para simular un nivel de señal analógica. La mayoría de las veces, este enfoque es suficiente para controlar la mayoría de los dispositivos que requieren niveles analógicos.

Entrada analógica y salida pwm

```
int entrada = A5; //Declaramos la entradas analógicas A0
int LEDR = 7; //Declaramos el pin digital 9 como salidas digitales PWM
int valor=0; //creamos una variable valor de tipo entero para almacenar los valores de las lecturas analógicas
void setup ()
{
pinMode(LEDR,OUTPUT); // Definimos el pin digital será de salida
}
void loop()
{
int valor = analogRead(entrada); // lee el valor del potenciómetro
analogWrite(LEDR, valor/4);
// Como las entradas analógicas tienen una resolución máxima de 1024 estados y el PWM tiene una resolución de 256, tendremos que dividir el valor de la entrada
//analógica entre 4 para hacer proporcional la lectura de la entrada analógica con la intensidad de salida de la salida digital PWM
}
```

Servo

- los servo motores tienen tres cables: rojo (alimentación), marrón o negro (tierra), y naranja o amarillo (señal). Conéctalo de la siguiente manera:
- Cable rojo: Conéctalo a 5V en Arduino.
- Cable marrón o negro: Conéctalo a GND (tierra) en Arduino.
- Cable naranja o amarillo: Conéctalo a un pin PWM, por ejemplo, pin 9 en Arduino.



```
#include <Servo.h>
// Crea una instancia del servo
Servo miServo;
void setup() {
    // Adjunta el servo al pin 9
    miServo.attach(9);
}
void loop() {
    // Mueve el servo a la posición 0 grados
    miServo.write(0);
    delay(1000); // Espera 1 segundo
    // Mueve el servo a la posición 90 grados
    miServo.write(90);
    delay(1000); // Espera 1 segundo
    // Mueve el servo a la posición 180 grados
    miServo.write(180);
    delay(1000); // Espera 1 segundo
}
```

- Este código utiliza la biblioteca Servo para controlar el servo. El servo se mueve desde 0 grados hasta 180 grados con pausas de 1 segundo en cada posición.
- Puedes ajustar las posiciones y los tiempos de espera según tus necesidades. Este es un ejemplo simple de cómo controlar un servo con Arduino, pero los servos son ideales para controlar movimientos precisos en proyectos como robótica, control de cámaras, y más.

Entrada analógica y salida en servo

Servo motor en actuar en respuesta a la entrada de un potenciómetro.

Esto te permitirá girar el servo manualmente ajustando el potenciómetro: Conecta el servo motor a tu Arduino y asegúrate de que el potenciómetro esté conectado a un pin analógico (en este ejemplo, usaremos el pin A0 para el potenciómetro).

```
#include <Servo.h>
// Crea una instancia del servo
Servo miServo;
// Pin analógico para el potenciómetro
int potPin = A0;
void setup() {
    // Adjunta el servo al pin 9
    miServo.attach(9);
}
void loop() {
    // Leer el valor del potenciómetro (rango de 0 a 1023)
    int valorPot = analogRead(potPin);
    // Mapear el valor del potenciómetro al rango de ángulo (0 a 180 grados)
    int angulo = map(valorPot, 0, 1023, 0, 180);
    // Mover el servo al ángulo calculado
    miServo.write(angulo);
    // Pequeña pausa para evitar lecturas rápidas del potenciómetro
    delay(10);
}
```

Este código utiliza un potenciómetro conectado al pin A0 para ajustar la posición del servo. Cuando giras el potenciómetro, el valor leído se mapea al rango de 0 a 180 grados y se utiliza para controlar la posición del servo

Programación en ARDUINO: Bucle repetitivo while

The screenshot shows the Arduino IDE interface with a sketch titled "LED_brillo_progresivo". The code implements a progressive LED brightness control using an interrupt. It defines pins 10 and 7, initializes variables, and sets up pin 10 as an output and pin 7 as an input. The loop function contains two nested while loops: one for the LOW state of the interrupt pin (which increments the LED brightness) and one for the HIGH state (which decrements it). Both loops include a 50ms delay.

```
LED_brillo_progresivo
/*
LED que se enciende gradualmente cuando un
interruptor pasa a on y que se apaga gradualmente
cuando el interruptor pasa a off */
int pinLED = 10; // LED en salida PWM
int pinInterruptor = 7; //Interruptor en pin 7
int valor=0;

void setup() {
  pinMode(pinLED, OUTPUT);
  pinMode(pinInterruptor, INPUT);
}

void loop() {
  while(digitalRead(pinInterruptor)==LOW){
    while(valor < 255){
      analogWrite(pinLED, valor); delay(50);
      valor++;
    }
    while(digitalRead(pinInterruptor)==HIGH){
      while(valor > 0){
        analogWrite(pinLED, valor); delay(50);
        valor--;
      }
    }
  }
}
```

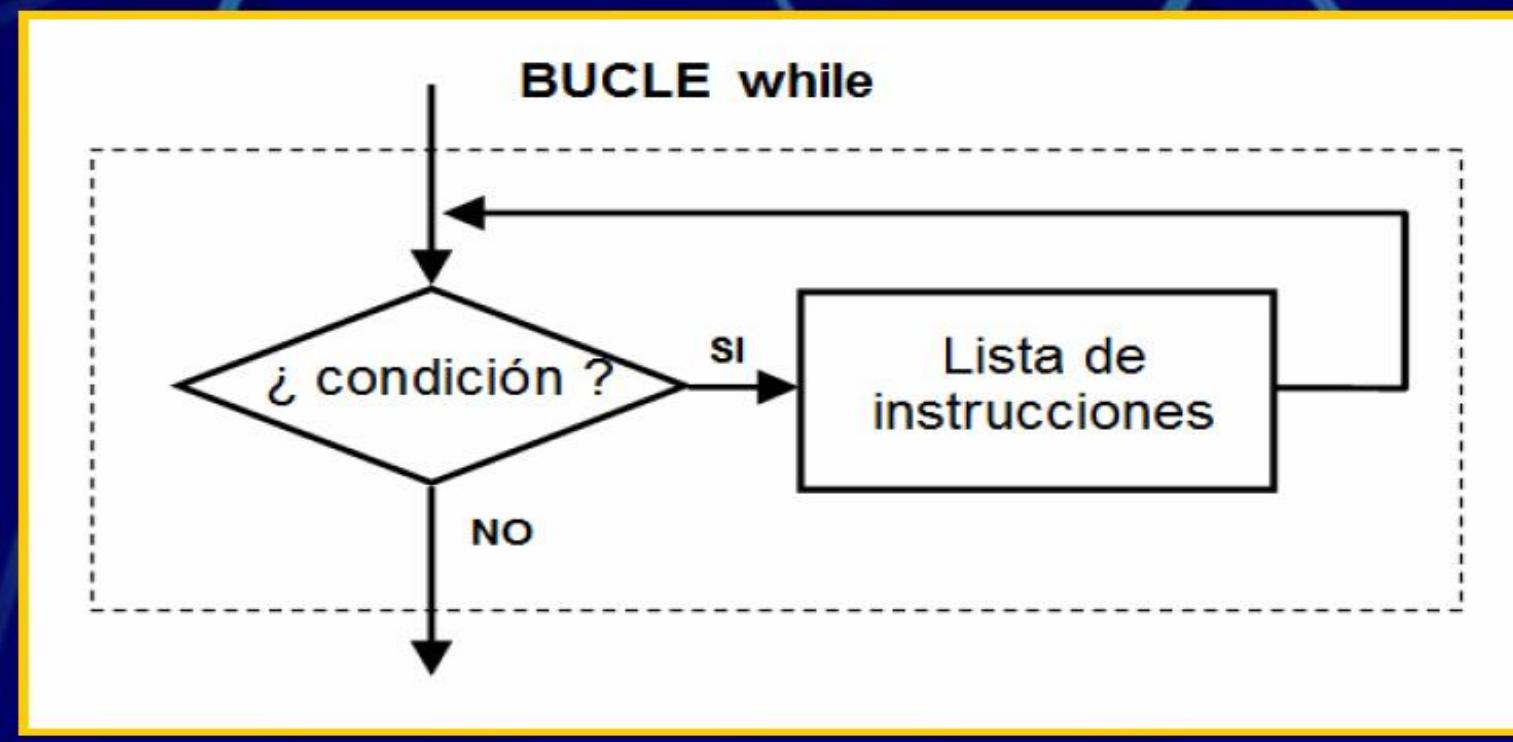
➤ El bucle **while** se repetirá indefinidamente hasta que la expresión booleana (condición) que sigue a la palabra **while** entre paréntesis antes del bloque de instrucciones se evalúe como *false*.

```
while (condición) {
  bloque de instrucciones;
}
```

➤ La condición se evalúa al principio del bucle, por lo que, si la primera vez que se evalúa ya es falsa, las instrucciones contenidas en el bucle no se ejecutarán ninguna vez.

Diagrama de flujo del bucle repetitivo while

```
while (condición) {  
    bloque de instrucciones;  
}
```



los LEDs conectados a los pines 2 y 3 parpadearán alternativamente cinco veces antes de que el programa termine

```
// Definir pines para los LEDs
int led1Pin = 2; // Pin del primer LED
int led2Pin = 3; // Pin del segundo LED
void setup() {
    // Configurar los pines de los LEDs como salidas
    pinMode(led1Pin, OUTPUT);
    pinMode(led2Pin, OUTPUT);
}
void loop() {
    // Definir una condición para el bucle while
    int contador = 0; // Inicializar un contador
```

```
while (contador < 5) { // Mientras el contador sea menor que 5
    // Encender el primer LED
    digitalWrite(led1Pin, HIGH);
    delay(500); // Esperar medio segundo (500 milisegundos)
    // Apagar el primer LED
    digitalWrite(led1Pin, LOW);
    delay(500); // Esperar medio segundo
    // Encender el segundo LED
    digitalWrite(led2Pin, HIGH);
    delay(500); // Esperar medio segundo
    // Apagar el segundo LED
    digitalWrite(led2Pin, LOW);
    delay(500); // Esperar medio segundo
    // Incrementar el contador en 1
    contador++;
}
// Cuando el contador llega a 5, el bucle while termina
```

Programación en ARDUINO: Bucle repetitivo **for**

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** LED_luz_progresiva | Arduino 1.0.3
- Menu Bar:** Archivo Editar Sketch Herramientas Ayuda
- Toolbar:** Includes icons for Save, Run, Upload, Download, and Find.
- Sketch Area:** Displays the code for 'LED_luz_progresiva'. The code is as follows:

```
/* Iluminación progresiva de un LED
conectado a un pin digital PWM al final
queda encendido a máxima luminosidad */

int pinPWM = 10; // LED en el pin 10
void setup()
{
    // No es necesaria configuración
}
void loop()
{
    for (int i = 0; i <= 255; i++) {
        analogWrite(pinPWM, i);
        delay(50);
    }
}
```
- Status Bar:** Shows 'Guardado Terminado.' and 'Tamaño binario del Sketch: 1.266 bytes (de un'.

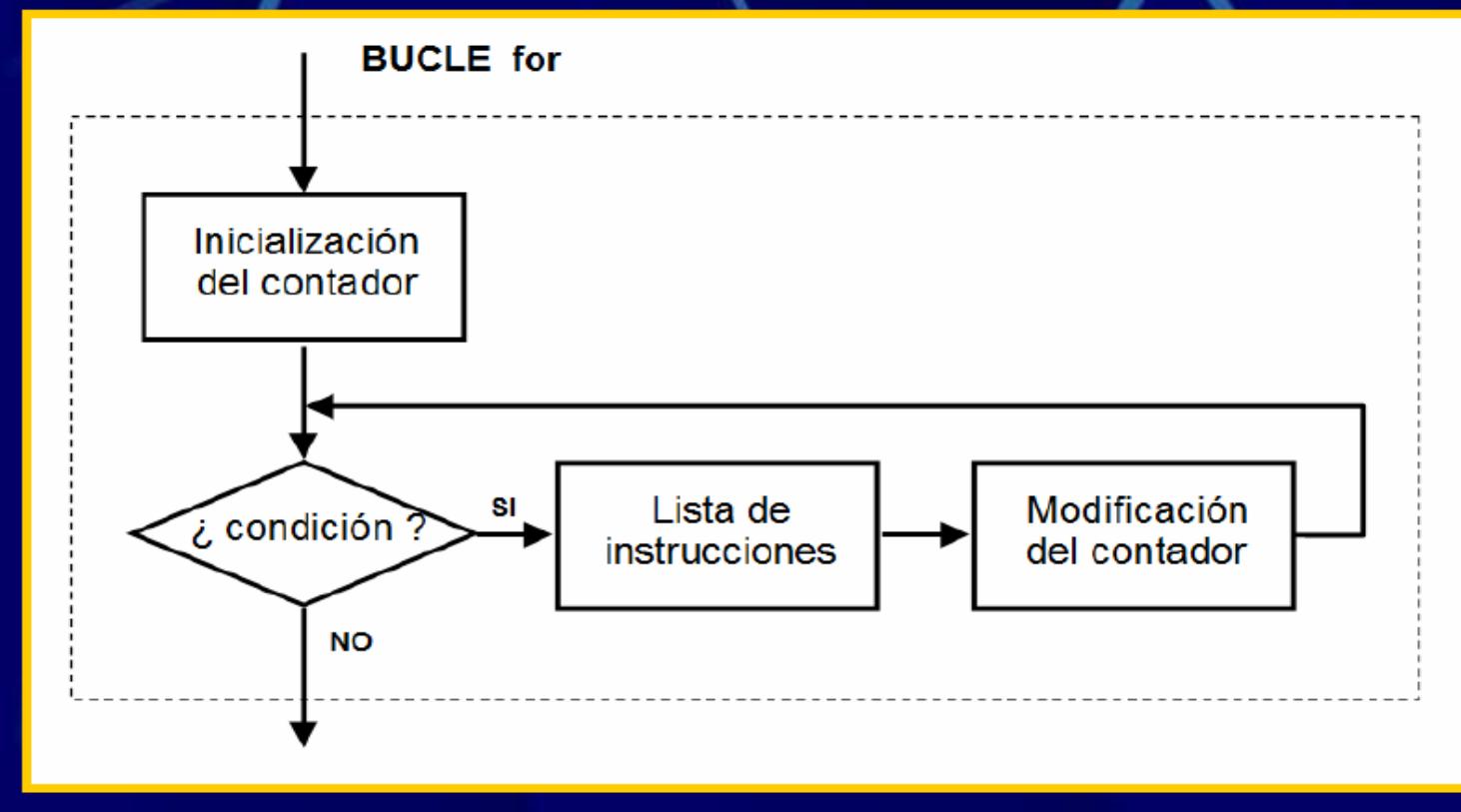
➤ La estructura **for** repite un bloque de instrucciones las veces que se quiera en tanto se cumpla una condición. Se suele usar una variable contador para guardar la cuenta de las repeticiones y decidir cuándo terminar el bucle.

```
for (inicio; condición; modificación)
{
    bloque de instrucciones;
}
```

➤ El **inicio** se ejecuta sólo una vez al principio. En cada pasada se comprueba la **condición**. Si es cierta, se ejecutan las **instrucciones** y la **modificación**. El bucle termina cuando la **condición** se vuelve falsa.

Diagrama de flujo del bucle repetitivo **for**

```
for (inicio; condición; modificación)
{
    bloque de instrucciones;
}
```



```
// Pin del LED
int ledPin = 13; // El LED incorporado en la mayoría de las placas Arduino

void setup() {
    // Configurar el pin del LED como salida
    pinMode(ledPin, OUTPUT);
}

void loop() {
    int parpadeos = 5; // Número de veces que parpadeará el LED

    // Hacer que el LED parpadee
    for (int i = 0; i < parpadeos; i++) {
        digitalWrite(ledPin, HIGH); // Encender el LED
        delay(500); // Esperar medio segundo
        digitalWrite(ledPin, LOW); // Apagar el LED
        delay(500); // Esperar medio segundo
    }
}
```

Asistente de Estacionamiento

```
// Librería para el sensor ultrasónico
#include <Ultrasonic.h>
// Definir pines para el sensor ultrasónico y los LEDs
#define TRIGGER_PIN 9
#define ECHO_PIN 10
#define LED_VERDE 2
#define LED_AMARILLO 3
#define LED_ROJO 4
// Distancia segura para estacionar (en centímetros)
#define DISTANCIA_SEGURA 30
// Inicializar el sensor ultrasónico
Ultrasonic ultrasonic(TRIGGER_PIN, ECHO_PIN);
```

```
void setup() {  
    // Configurar los pines de los LEDs como salidas  
    pinMode(LED_VERDE, OUTPUT);  
    pinMode(LED_AMARILLO, OUTPUT);  
    pinMode(LED_ROJO, OUTPUT);  
    // Iniciar comunicación con el monitor serial  
    Serial.begin(9600);  
    Serial.println("Asistente de Estacionamiento activado.");  
}
```

```
void loop() {
    // Realizar una medición de distancia
    long distancia = ultrasonic.read();
    // Comprobar si el vehículo está a una distancia segura para estacionar
    if (distancia <= DISTANCIA_SEGURA) {
        // Encender el LED rojo (detenerse)
        digitalWrite(LED_ROJO, HIGH);
        digitalWrite(LED_VERDE, LOW);
        digitalWrite(LED_AMARILLO, LOW);
        Serial.println("¡Detente! Estás a una distancia insegura.");
    }
}
```

```
} else if (distancia <= (DISTANCIA_SEGURA * 2)) {  
    // Encender el LED amarillo (reduzca la velocidad)  
    digitalWrite(LED_AMARILLO, HIGH);  
    digitalWrite(LED_VERDE, LOW);  
    digitalWrite(LED_ROJO, LOW);  
    Serial.println("Reduce la velocidad, estás cerca del obstáculo.");  
}  
else {  
    // Encender el LED verde (puedes avanzar)  
    digitalWrite(LED_VERDE, HIGH);  
    digitalWrite(LED_ROJO, LOW);  
    digitalWrite(LED_AMARILLO, LOW);  
    Serial.println("Puedes avanzar y estacionar de manera segura.");  
}  
  
// Esperar un breve momento antes de la próxima medición  
delay(500);  
}
```

Program. en ARDUINO: Estructura condicional switch...case

The screenshot shows the Arduino IDE interface with a sketch titled "Nivel_iluminacion". The code reads an analog input from an LDR and prints the level to the Serial monitor. It uses a switch statement to map the analog value to a light level (0, 1, or 2) and prints the corresponding message.

```
Archivo Editar Sketch Herramientas Ayuda  
Nivel_iluminacion  
/* Lee una entrada analógica donde se mide el nivel de iluminación sobre una LDR, e imprime un mensaje */  
  
int valorleido = 0; // variable para almacenar lectura  
int nivel = 0; // variable para nivel de iluminación  
  
void setup() {  
    Serial.begin (9600); // abre el puerto serie  
}  
void loop() {  
    valorleido = analogRead(0);  
    if (valorleido <= 400) nivel=0;  
    if (valorleido > 400) nivel=1;  
    if (valorleido > 800) nivel=2;  
    switch (nivel) {  
        case 0:  
            Serial.println ("Nivel de iluminación bajo");  
            break;  
        case 1:  
            Serial.println ("Nivel de iluminación medio");  
            break;  
        case 2:  
            Serial.println ("Nivel de iluminación alto");  
            break;  
    }  
    delay(5000); // Retardo de 5 segundos  
}
```

➤ La estructura **switch...case** compara el valor de una variable con unas etiquetas. Cuando una coincide se ejecuta su bloque de instrucciones. Opcionalmente puede llevar **default**.

```
switch (variable) {  
    case etiqueta1:  
        bloque de instrucciones 1;  
        break;  
    case etiqueta2:  
        bloque de instrucciones 2;  
        break;  
    default: //es opcional  
        bloque instrucciones def.;  
}
```

Program. en ARDUINO: Estructura condicional switch...case

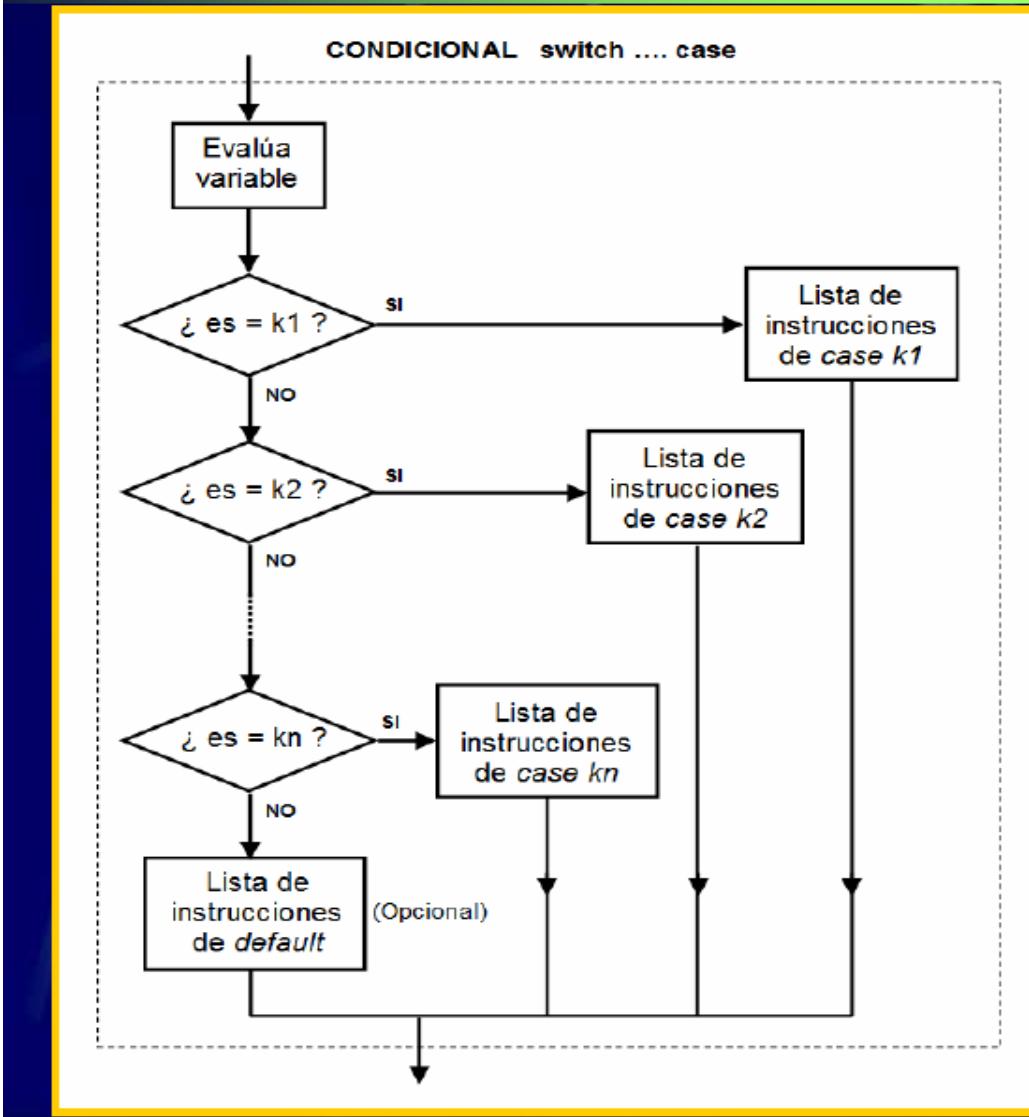
The screenshot shows the Arduino IDE interface with a sketch titled "Nivel_iluminacion". The code reads an analog input from an LDR to determine light level and prints it to the Serial monitor. It uses an if-else chain to set a variable 'nivel' based on the reading, and then a switch statement to map 'nivel' values (0, 1, or 2) to specific messages.

```
Archivo Editar Sketch Herramientas Ayuda  
Nivel_iluminacion  
/* Lee una entrada analógica donde se mide el nivel de iluminación sobre una LDR, e imprime un mensaje */  
  
int valorleido = 0; // variable para almacenar lectura  
int nivel = 0; // variable para nivel de iluminación  
  
void setup() {  
    Serial.begin (9600); // abre el puerto serie  
}  
void loop() {  
    valorleido = analogRead(0);  
    if (valorleido <= 400) nivel=0;  
    if (valorleido > 400) nivel=1;  
    if (valorleido > 800) nivel=2;  
    switch (nivel) {  
        case 0:  
            Serial.println ("Nivel de iluminación bajo");  
            break;  
        case 1:  
            Serial.println ("Nivel de iluminación medio");  
            break;  
        case 2:  
            Serial.println ("Nivel de iluminación alto");  
            break;  
    }  
    delay(5000); // Retardo de 5 segundos  
}
```

➤ La estructura **switch...case** compara el valor de una variable con unas **etiquetas**. Cuando una coincide se ejecuta su bloque de instrucciones. Opcionalmente puede llevar **default**.

```
switch (variable) {  
    case etiqueta1:  
        bloque de instrucciones 1;  
        break;  
    case etiqueta2:  
        bloque de instrucciones 2;  
        break;  
    default: //es opcional  
        bloque instrucciones def.;  
}
```

Diagrama de flujo de la estructura switch.....case



```
switch (variable) {  
    case etiqueta_k1:  
        bloque de instrucciones 1;  
        break;  
    case etiqueta_k2:  
        bloque de instrucciones 2;  
        break;  
    .  
    .  
    .  
    case etiqueta_kn:  
        bloque de instrucciones n;  
        break;  
    default: //es opcional  
        bloque instrucciones def.;  
}
```

luces en una casa inteligente

```
// Definir pines para los pulsadores e LEDs
int pulsadorSalaPin = 2;
int pulsadorCocinaPin = 3;
int pulsadorDormitorioPin = 4;
int ledSalaPin = 5;
int ledCocinaPin = 6;
int ledDormitorioPin = 7;

void setup() {
    // Configurar los pines de los pulsadores como entradas con resistencia de pull-up
    pinMode(pulsadorSalaPin, INPUT_PULLUP);
    pinMode(pulsadorCocinaPin, INPUT_PULLUP);
    pinMode(pulsadorDormitorioPin, INPUT_PULLUP);

    // Configurar los pines de los LEDs como salidas
    pinMode(ledSalaPin, OUTPUT);
    pinMode(ledCocinaPin, OUTPUT);
    pinMode(ledDormitorioPin, OUTPUT);
}
```

```
void loop() {
    // Leer el estado de los pulsadores (HIGH cuando están sin presionar, LOW cuando están presionados)
    int pulsadorSalaEstado = digitalRead(pulsadorSalaPin);
    int pulsadorCocinaEstado = digitalRead(pulsadorCocinaPin);
    int pulsadorDormitorioEstado = digitalRead(pulsadorDormitorioPin);

    // Controlar las luces de acuerdo a los pulsadores
    switch (pulsadorSalaEstado) {
        case LOW:
            digitalWrite(ledSalaPin, HIGH); // Encender la luz de la sala
            break;
        case HIGH:
            digitalWrite(ledSalaPin, LOW); // Apagar la luz de la sala
            break;
    }
}
```

```
switch (pulsadorCocinaEstado) {  
    case LOW:  
        digitalWrite(ledCocinaPin, HIGH); // Encender la luz de la cocina  
        break;  
    case HIGH:  
        digitalWrite(ledCocinaPin, LOW); // Apagar la luz de la cocina  
        break;  
}  
  
switch (pulsadorDormitorioEstado) {  
    case LOW:  
        digitalWrite(ledDormitorioPin, HIGH); // Encender la luz del dormitorio  
        break;  
    case HIGH:  
        digitalWrite(ledDormitorioPin, LOW); // Apagar la luz del dormitorio  
        break;  
}  
}
```

Programación en ARDUINO: funciones matemáticas

The screenshot shows the Arduino IDE interface with a sketch named "Constrain_y_map". The code uses the constrain() function to map analog input values from a potentiometer to digital output values for two LEDs.

```
Constrain_y_map
/*
  Colocamos un potenciómetro entre +5V y 0V.
  Su patilla intermedia se conecta a entrada pinPOT
*/

int pinPOT = 0; // Potenciómetro en entrada ana 0
int pinLED_V = 9; // LED verde en salida PWM 9
int pinLED_R = 10; // LED rojo en salida PWM 10
int val_R; int val_V;

void setup() {
  pinMode(pinLED_V, OUTPUT);
  pinMode(pinLED_R, OUTPUT);
}

void loop() {
  val_R = analogRead(pinPOT);
  val_R = constrain (val_R, 0, 255);
  val_V = analogRead(pinPOT);
  val_V = map (val_V, 0, 1023, 0, 255);
  analogWrite(pinLED_R, val_R);
  analogWrite(pinLED_V, val_V);
}

/* Para un valor leído en el potenciómetro de 511,
val_R adoptará un valor de 255, mientras que val_V
adoptará un valor de 127 */

```

➤ Arduino cuenta con muchas funciones matemáticas: mínimo y máximo, raíz cuadrada, potencia, trigonométricas, etc.

➤ Son muy útiles las funciones:

constrain (x, a, b)

Esta función restringe el valor de x a estar dentro del rango $[a,b]$

map (x, di, ds, hi, hs)

Esta función reasigna el valor de x del rango $[di, ds]$ al rango $[hi, hs]$

Control de temperatura y humedad

Este ejemplo utiliza un sensor DHT11 o DHT22 para medir la temperatura y la humedad en la habitación. Cuando la temperatura supera el umbral especificado (`umbralTemperatura`), el ventilador se enciende para enfriar la habitación. Si la temperatura cae por debajo del umbral, el ventilador se apaga.

Puedes ajustar el valor de `umbralTemperatura` según tus preferencias para controlar el dispositivo de enfriamiento cuando la temperatura alcance el nivel deseado. Este ejemplo representa una aplicación realista de control de temperatura en una habitación utilizando un sensor de temperatura y un dispositivo de enfriamiento.

```
#include <DHT.h> // Biblioteca que vamos a usar

// Definir el tipo de sensor y el pin de datos
#define DHT_TYPE DHT11 // Puedes cambiarlo a DHT22 si estás usando ese sensor
#define DHT_PIN 2    // Pin de datos del sensor

// Definir el pin para el ventilador
int ventiladorPin = 3;

// Definir el umbral de temperatura en grados Celsius para encender el ventilador
float umbralTemperatura = 25.0; // Cambia este valor según tus necesidades

// Crear una instancia del sensor DHT
DHT dht(DHT_PIN, DHT_TYPE);
```

Una "instancia del sensor" en este contexto es un objeto que representa y facilita la comunicación con un sensor específico (en este caso, un sensor de temperatura y humedad DHT11 o DHT22). Esta instancia se crea utilizando la clase proporcionada por la biblioteca DHT y se utiliza para leer datos del sensor y realizar operaciones relacionadas con él.

```
void setup() {  
    // Configurar el pin del ventilador como salida  
    pinMode(ventiladorPin, OUTPUT);  
  
    // Iniciar comunicación con el monitor serial  
    Serial.begin(9600);  
  
    // Inicializar el sensor DHT  
    dht.begin();  
}
```

```
void loop() {  
    // Leer la temperatura y la humedad del sensor  
    float temperatura = dht.readTemperature();  
    float humedad = dht.readHumidity();  
    // Mostrar los datos en el monitor serial  
    Serial.print("Temperatura: ");  
    Serial.print(temperatura);  
    Serial.print("°C, Humedad: ");  
    Serial.print(humedad);  
    Serial.println("%");
```

```
// Controlar el ventilador según la temperatura
if (temperatura > umbralTemperatura) {
    digitalWrite(ventiladorPin, HIGH); // Encender el ventilador
    Serial.println("Ventilador encendido.");
} else {
    digitalWrite(ventiladorPin, LOW); // Apagar el ventilador
    Serial.println("Ventilador apagado.");
}
// Esperar un breve momento antes de la próxima lectura
delay(5000); // Leer cada 5 segundos
}
```