Control programado con ARDUINO

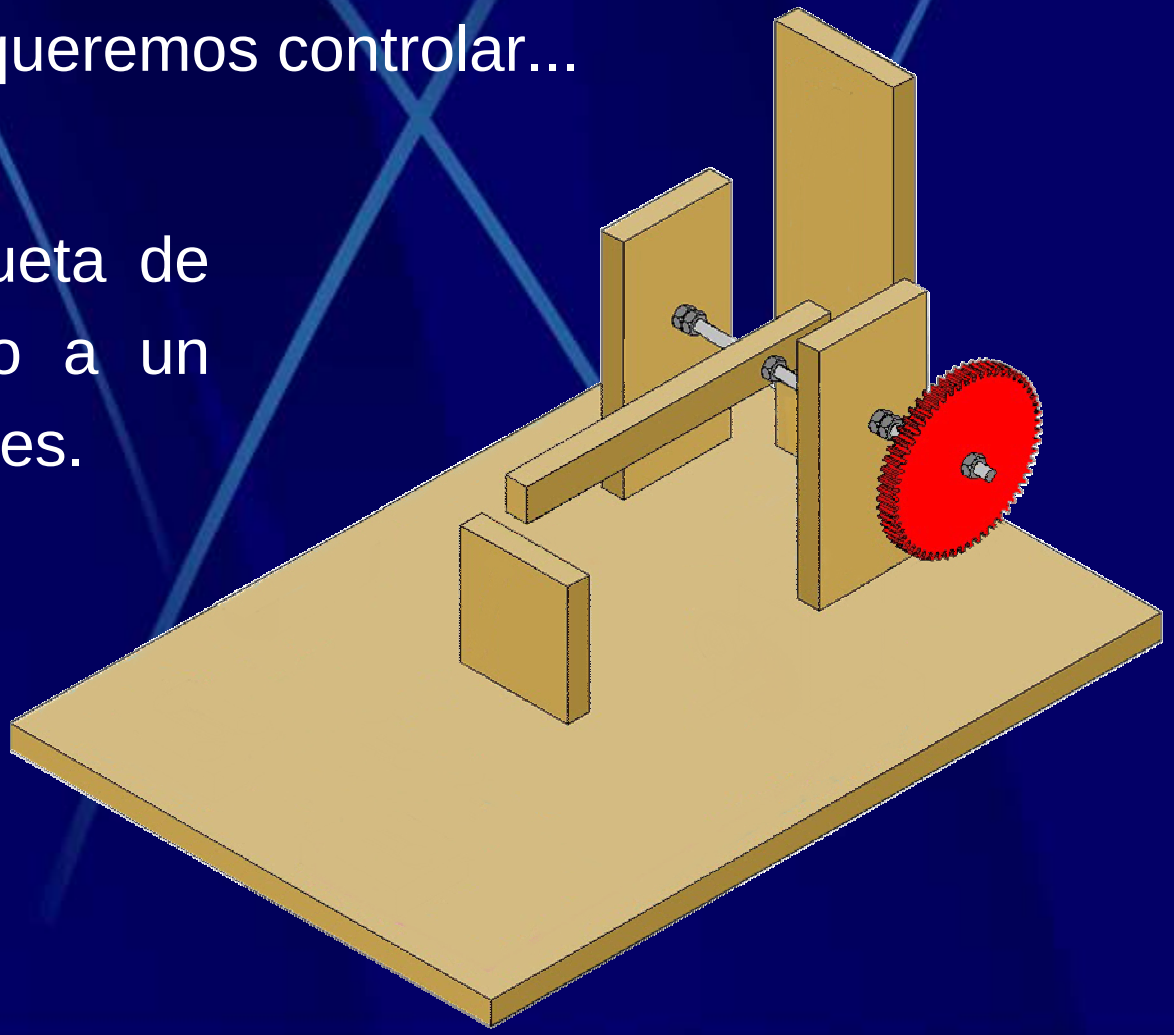
TECNOLOGÍA

IES BELLAVISTA

Control de un sistema técnico

Supongamos que hemos construido un **sistema técnico** cuyo funcionamiento queremos controlar...

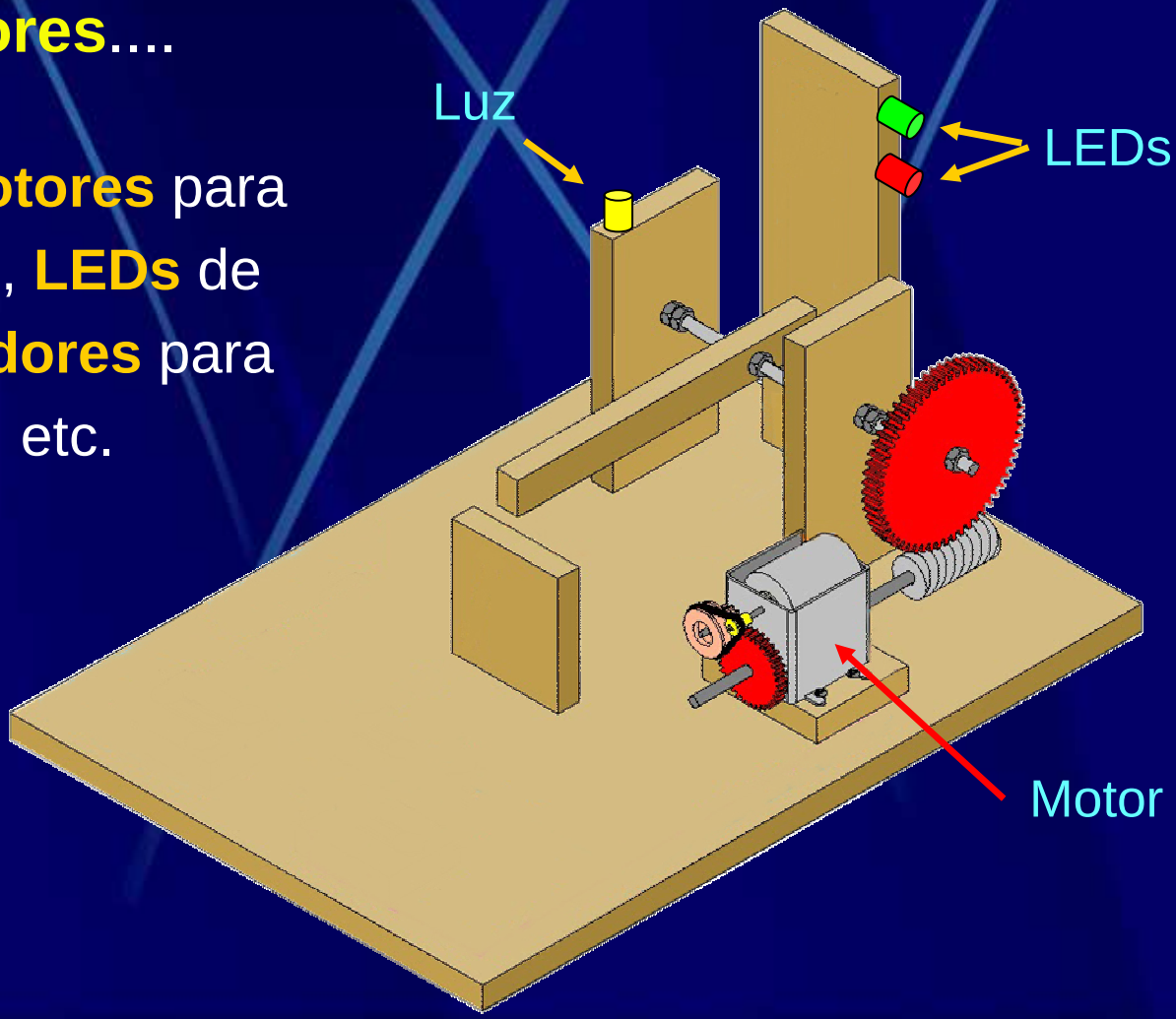
Por ejemplo, la maqueta de la barrera de acceso a un aparcamiento de coches.



Control de un sistema técnico: Actuadores

Para poder actuar sobre el sistema hemos de dotarlo de elementos **actuadores**....

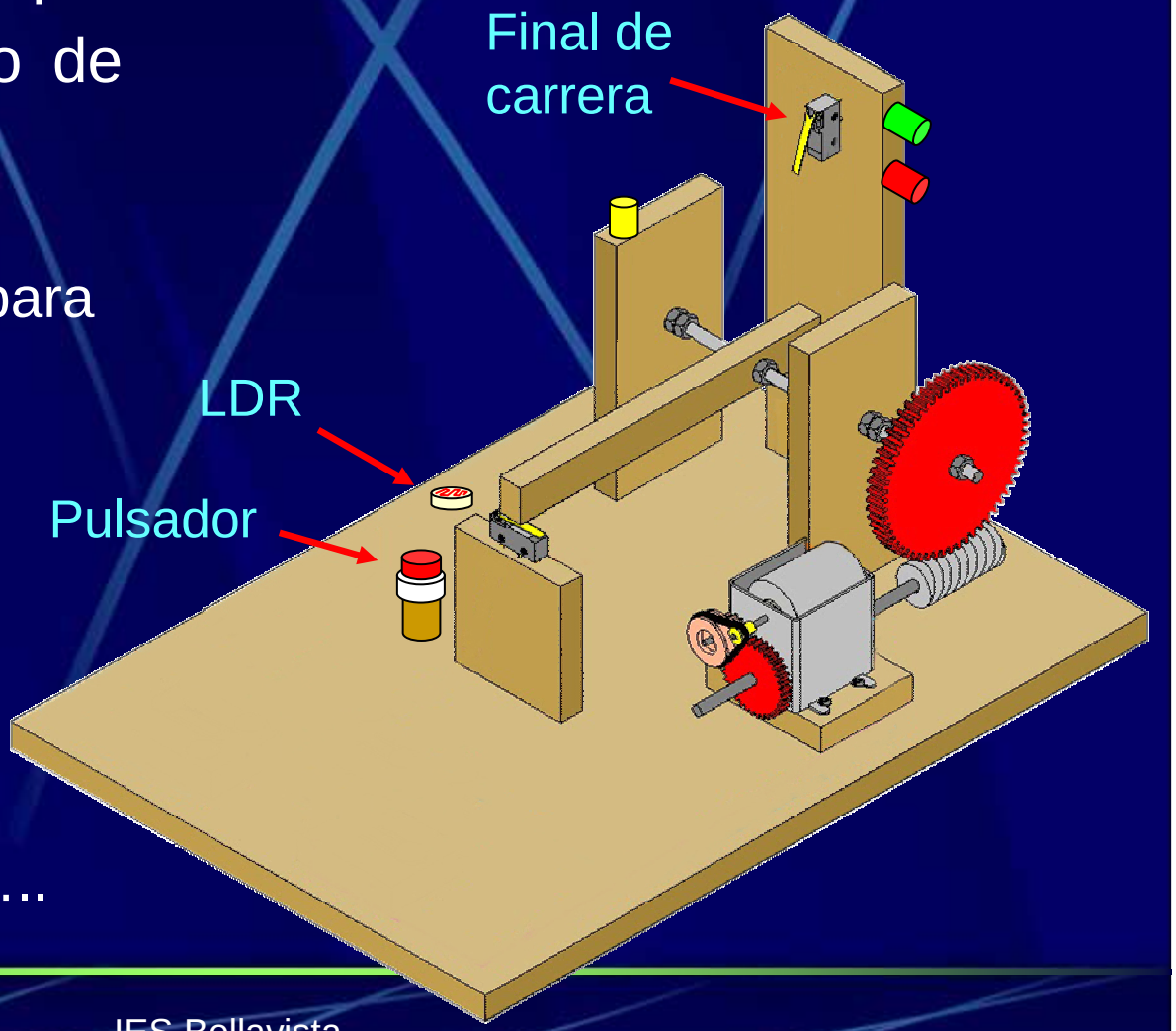
Como pueden ser **motores** para producir movimientos, **LEDs** de señalización, **zumbadores** para emitir sonidos, **luces**, etc.



Control de un sistema técnico: Sensores

También necesitamos información sobre el estado en que se encuentra el sistema, por lo que hay que dotarlo de elementos **sensores**....

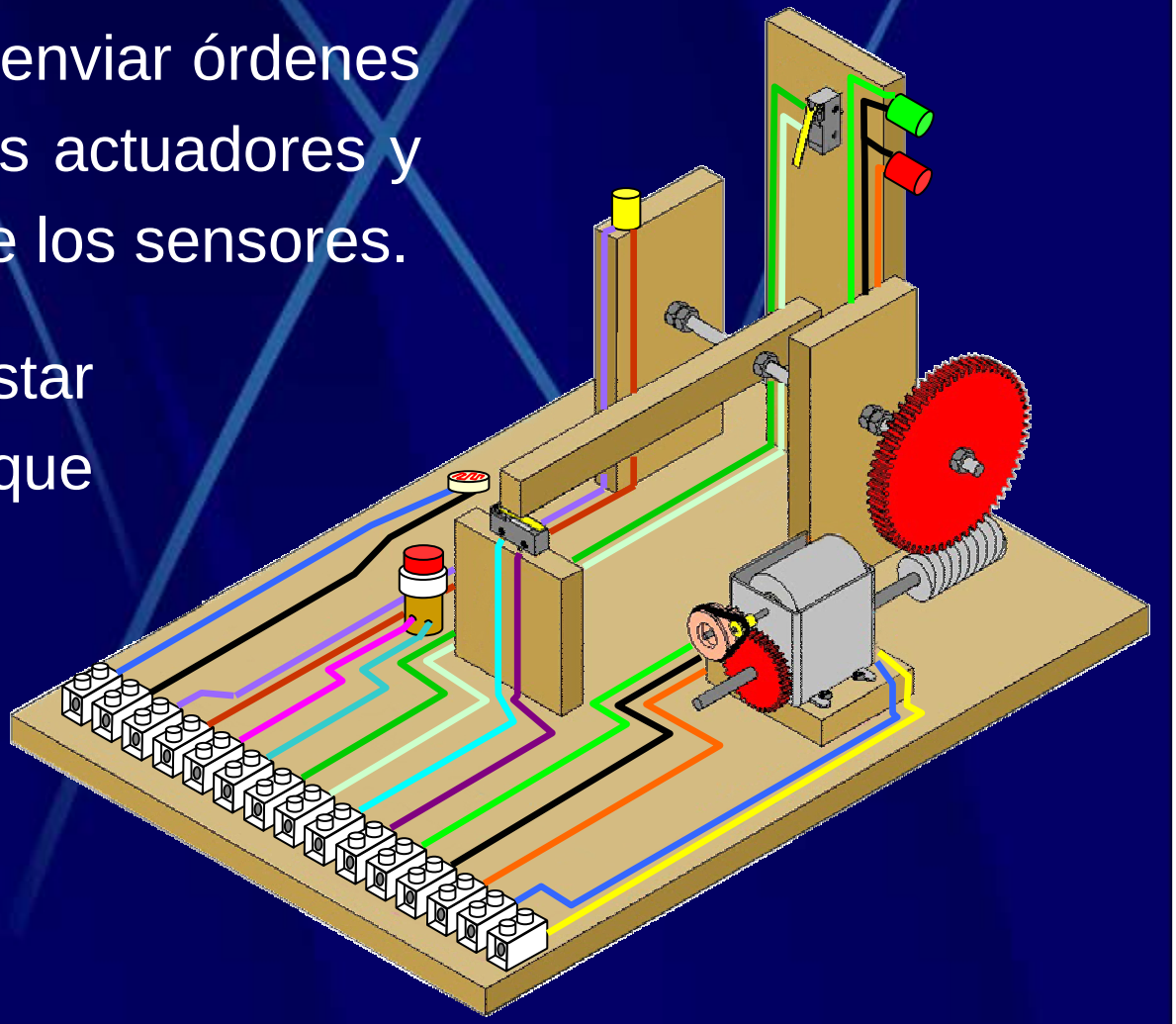
Pueden ser **pulsadores** para recibir órdenes de los usuarios, **finales de carrera** para saber dónde se encuentran sus elementos móviles, **LDRs** para detectar cambios de iluminación,…..



Control de un sistema técnico: cableado

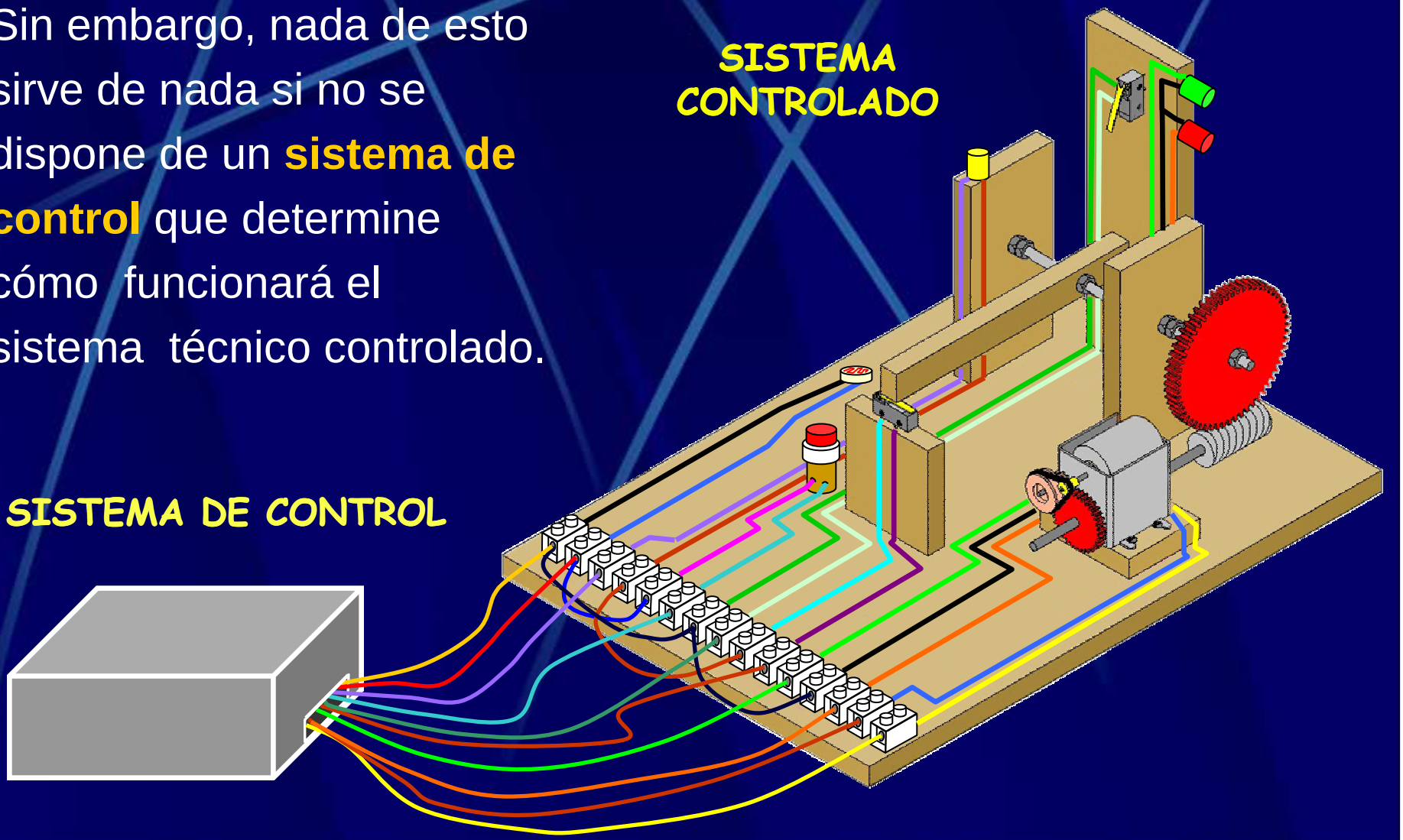
También debe dotársele de un **cableado** adecuado que permita enviar órdenes de funcionamiento a los actuadores y recibir la información de los sensores.

El cableado debe estar bien organizado para que permita distinguir con facilidad los terminales de los componentes a los que están conectados.



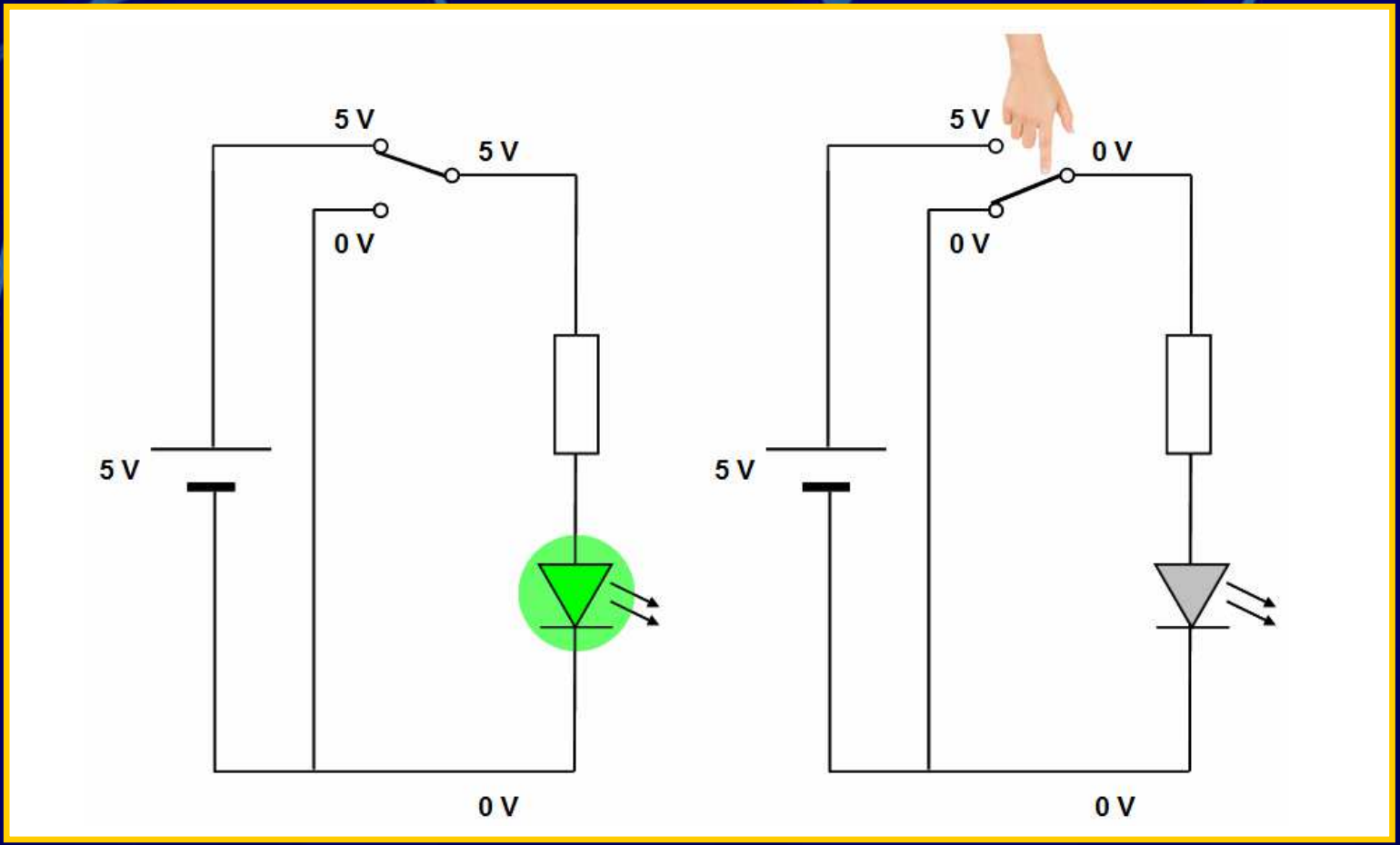
Control de un sistema técnico: sistema de control

Sin embargo, nada de esto sirve de nada si no se dispone de un **sistema de control** que determine cómo funcionará el sistema técnico controlado.



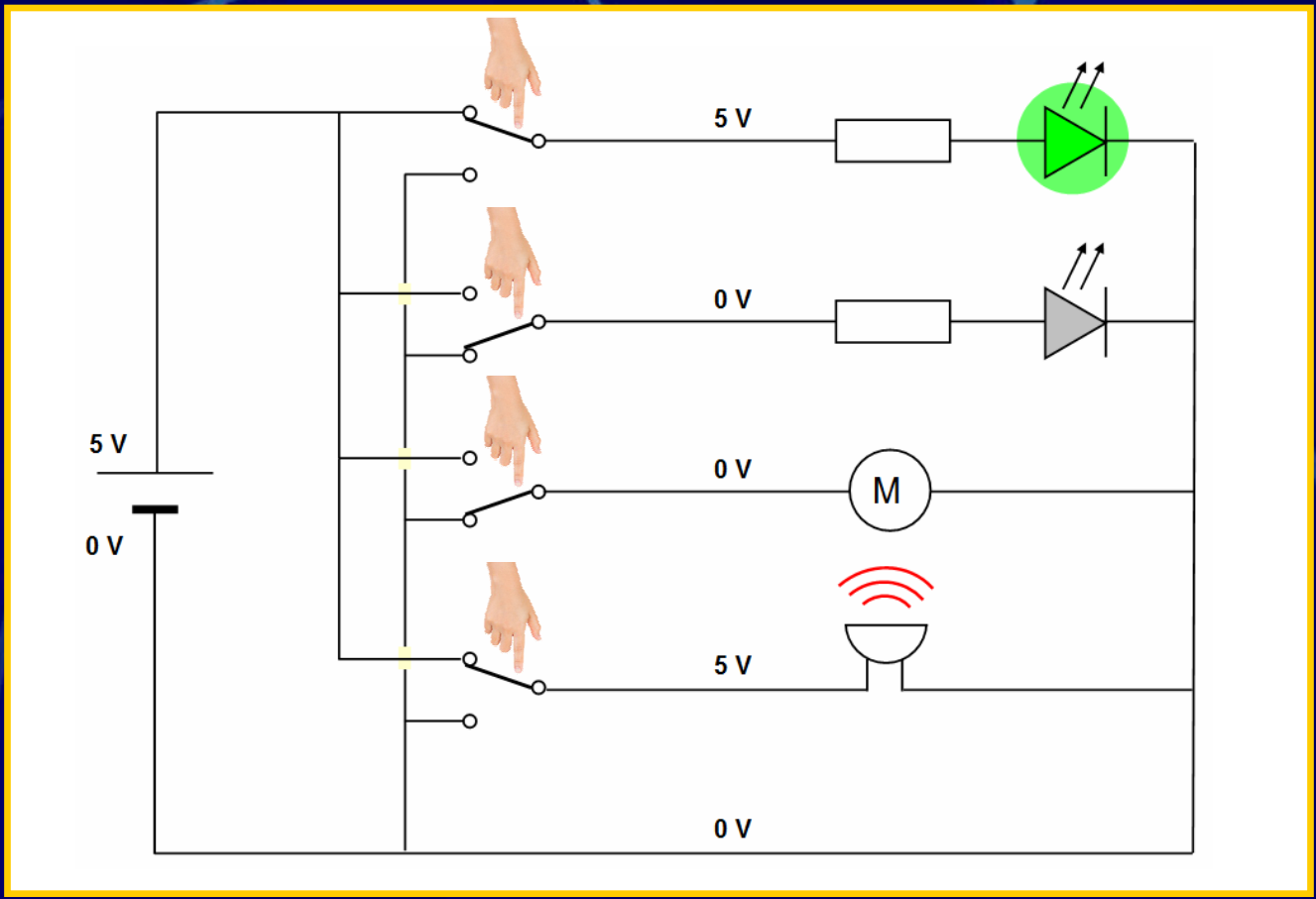
Control convencional de un sistema técnico

Podríamos controlar el funcionamiento de un actuador como, por ejemplo, el encendido y apagado de un LED, conectándolo como se indica y haciendo que una persona actúe sobre el conmutador cuando haya que cambiar el estado del LED.



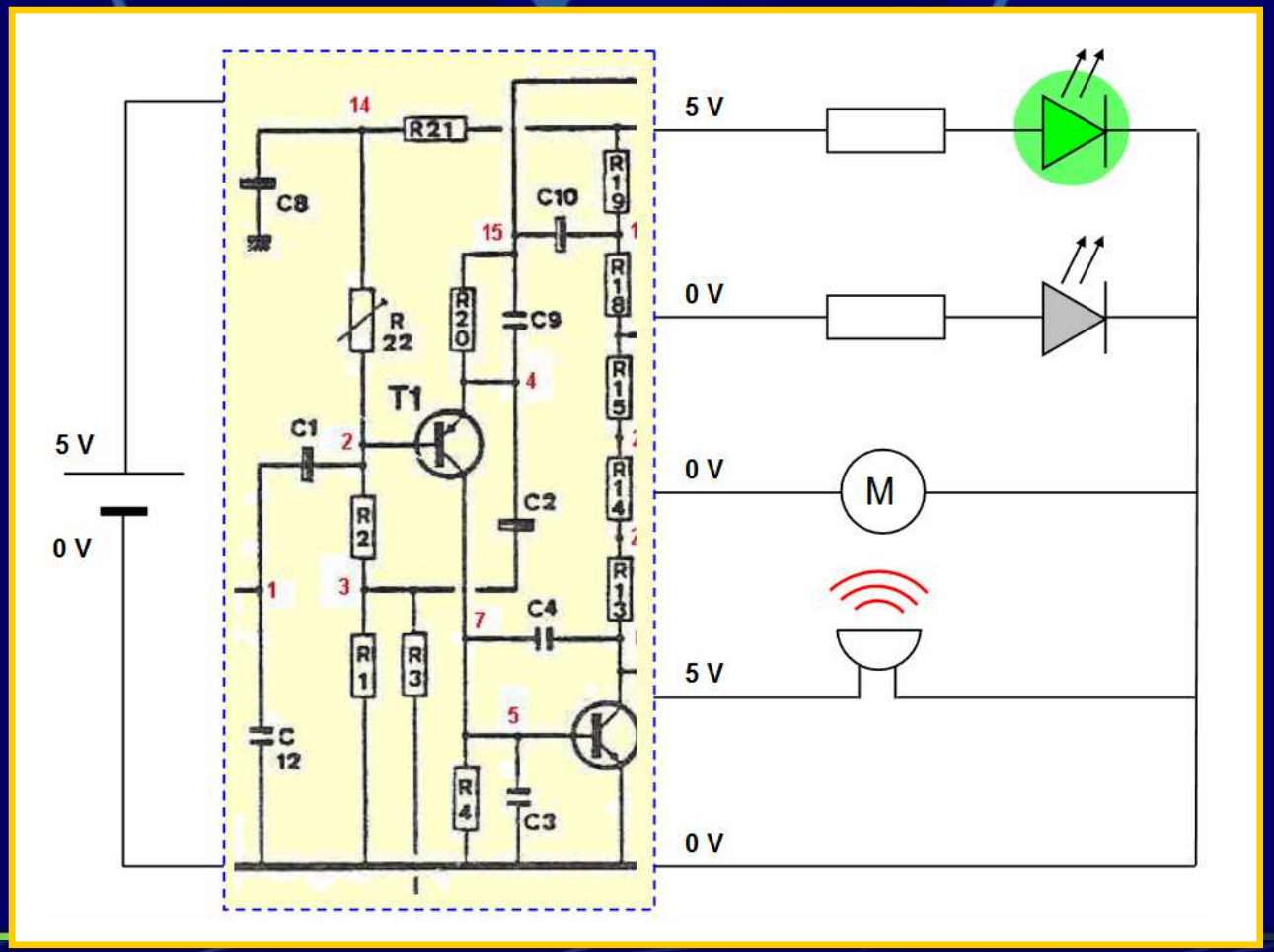
Control convencional de un sistema técnico

Igualmente podríamos controlar varios actuadores aplicándoles manualmente bien 5 V o bien 0 V según queramos que el actuador funcione o que no lo haga. Naturalmente, tener a una persona activando y desactivando no es viable.



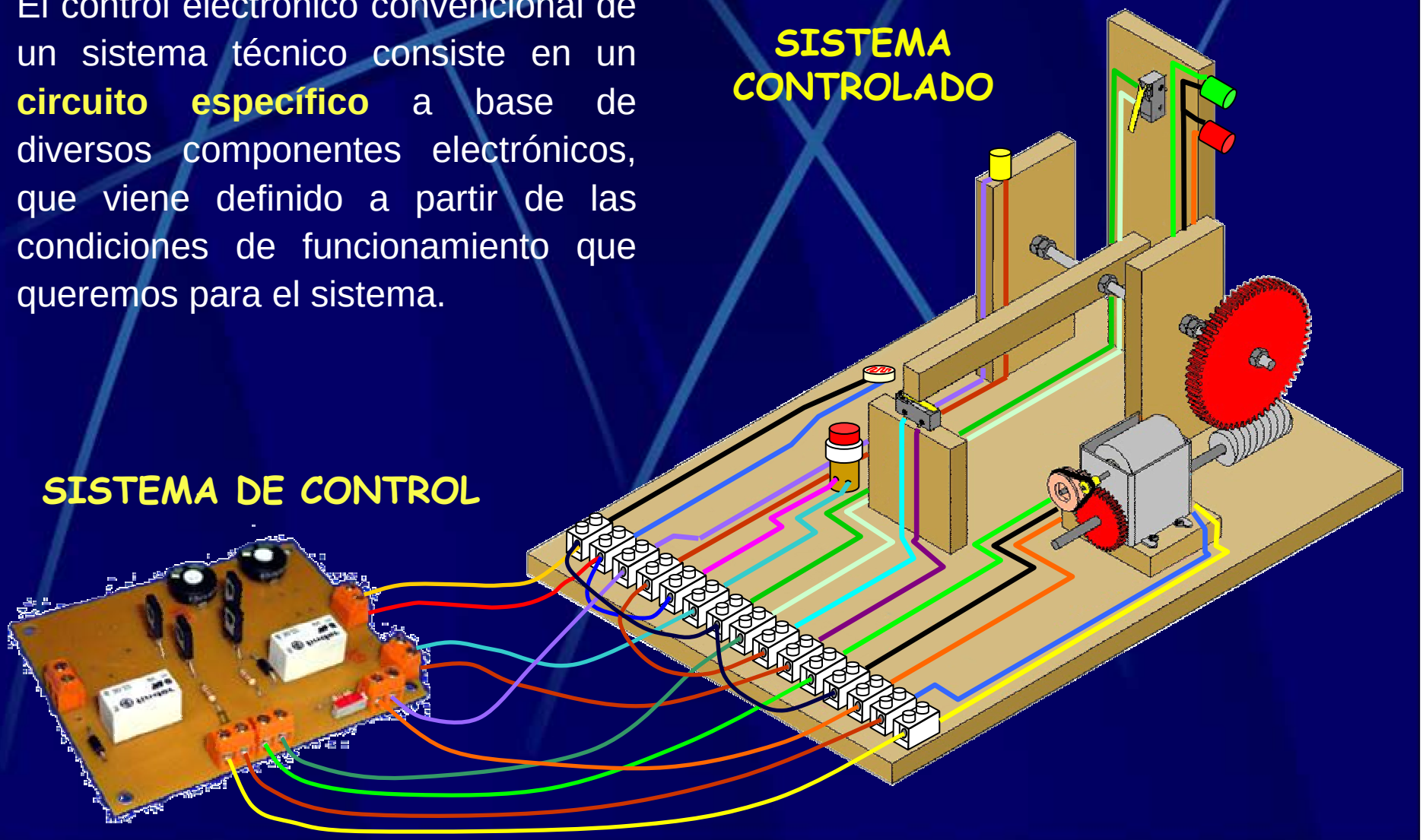
Control convencional de un sistema técnico

Como el control manual no es muy práctico, podríamos recurrir a diseñar un circuito electrónico convencional que haga lo mismo. No obstante, a poco complejo que sea el funcionamiento requerido se necesitan amplios conocimientos de electrónica.

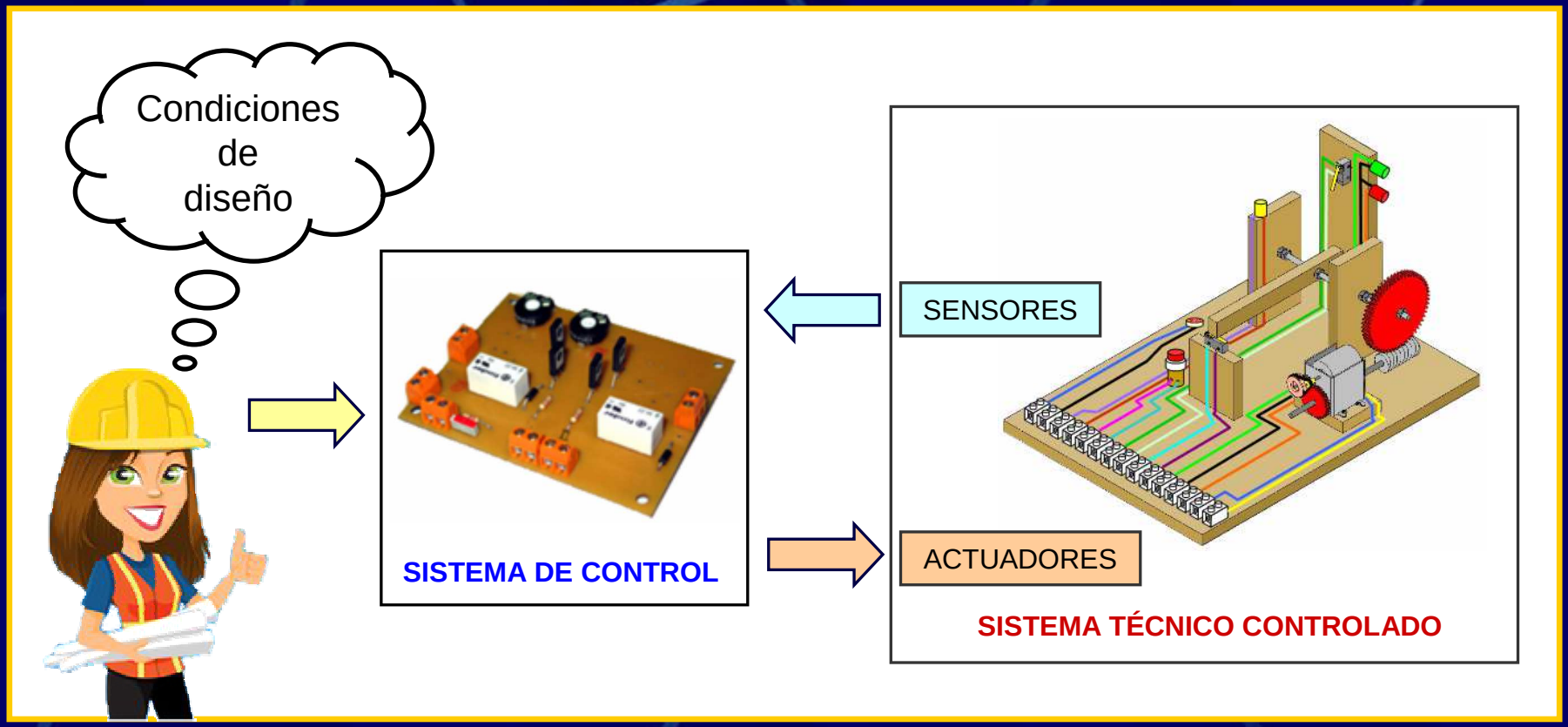


Control convencional de un sistema técnico

El control electrónico convencional de un sistema técnico consiste en un **circuito específico** a base de diversos componentes electrónicos, que viene definido a partir de las condiciones de funcionamiento que queremos para el sistema.



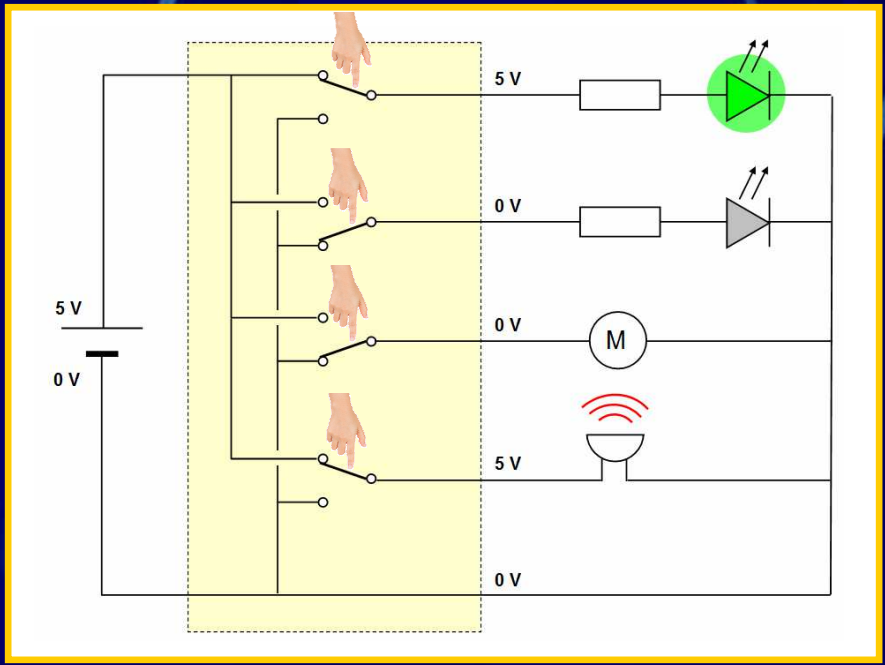
Control convencional de un sistema técnico

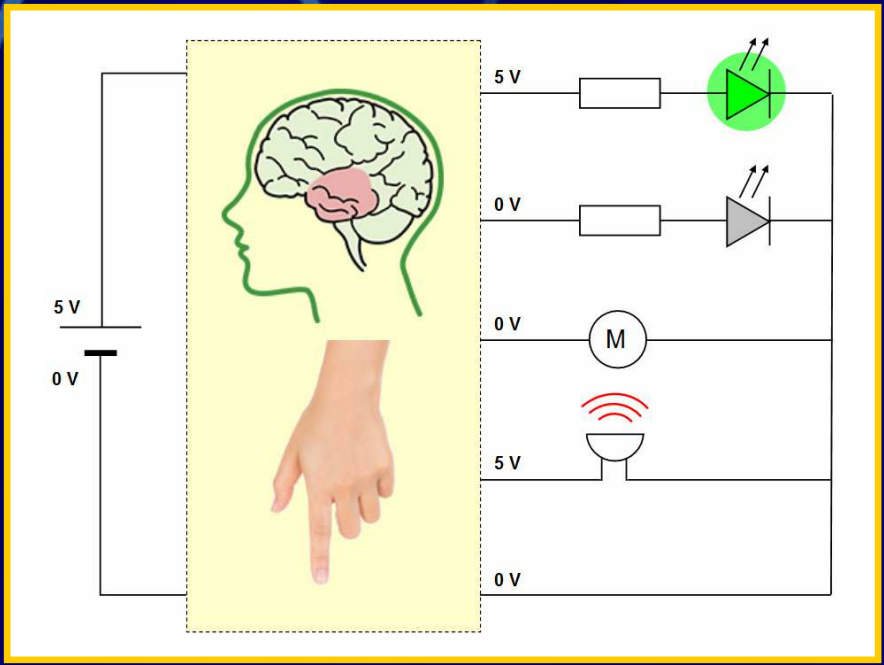


Como el circuito es específico, un problema del control convencional es que es **muy rígido**, cualquier cambio que queramos en el funcionamiento requiere cambios en los componentes y en las conexiones del circuito, lo cual no siempre será posible y debe ser realizado por personal especializado.

Pasando del control convencional al control programado

Si volvemos al esquema del control manual, podríamos sustituir el bloque de control formado por los conmutadores accionados manualmente por un circuito electrónico “genérico” cuyo modo de funcionamiento (programa) estuviera grabado en una memoria, de modo que pudierámos cambiarlo cuando quisiéramos sin tener siquiera conocimientos de electrónica.

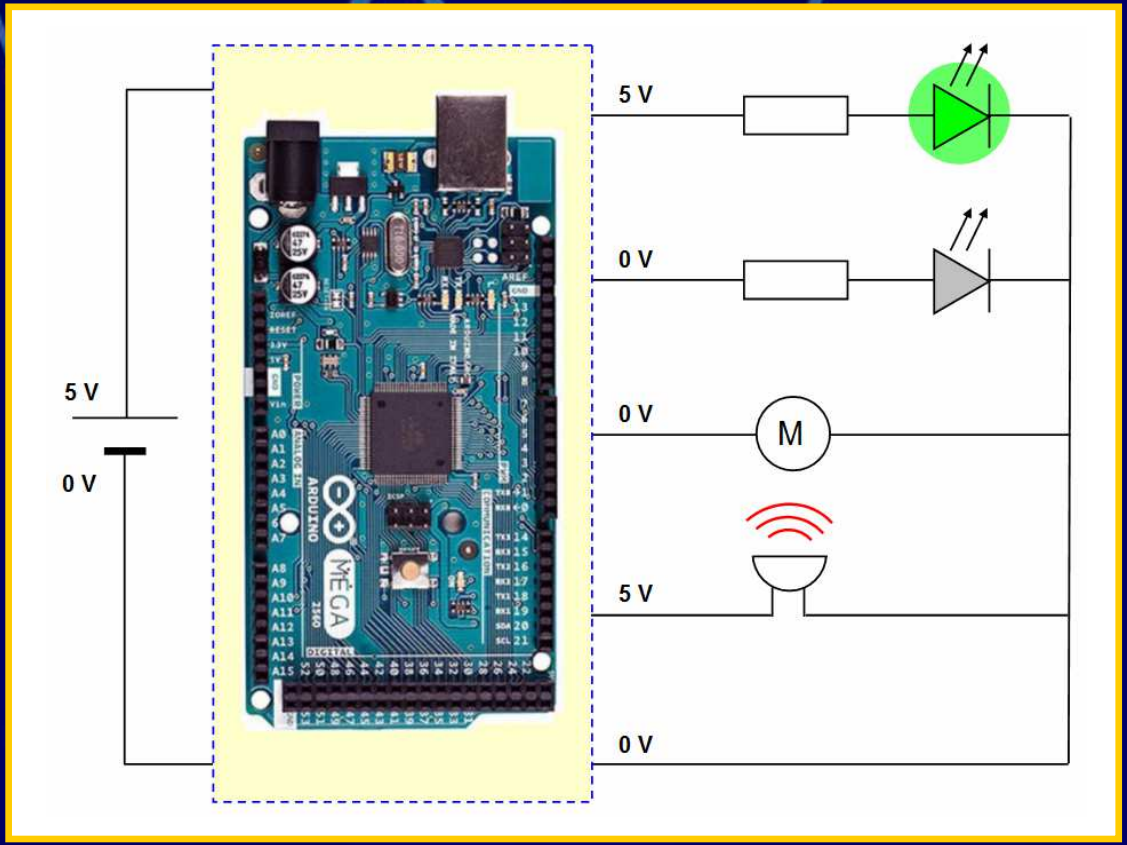




Control programado de un sistema técnico

Un **“programa”** es un conjunto de instrucciones que pueden ser leídas y ejecutadas por un microcontrolador. Como resultado de la ejecución de dichas instrucciones se activan o desactivan actuadores en el momento preciso, se lee la información que ofrecen los sensores, se hacen cálculos, se toman decisiones, etc.

El dispositivo electrónico que ejecuta el programa es muy **flexible** ya que si queremos cambiar el modo de funcionamiento del sistema controlado tan sólo tenemos que cambiar el programa almacenado en la memoria, pero no el dispositivo. El dispositivo también cuenta con **“pines”** para conectarse al sistema técnico.

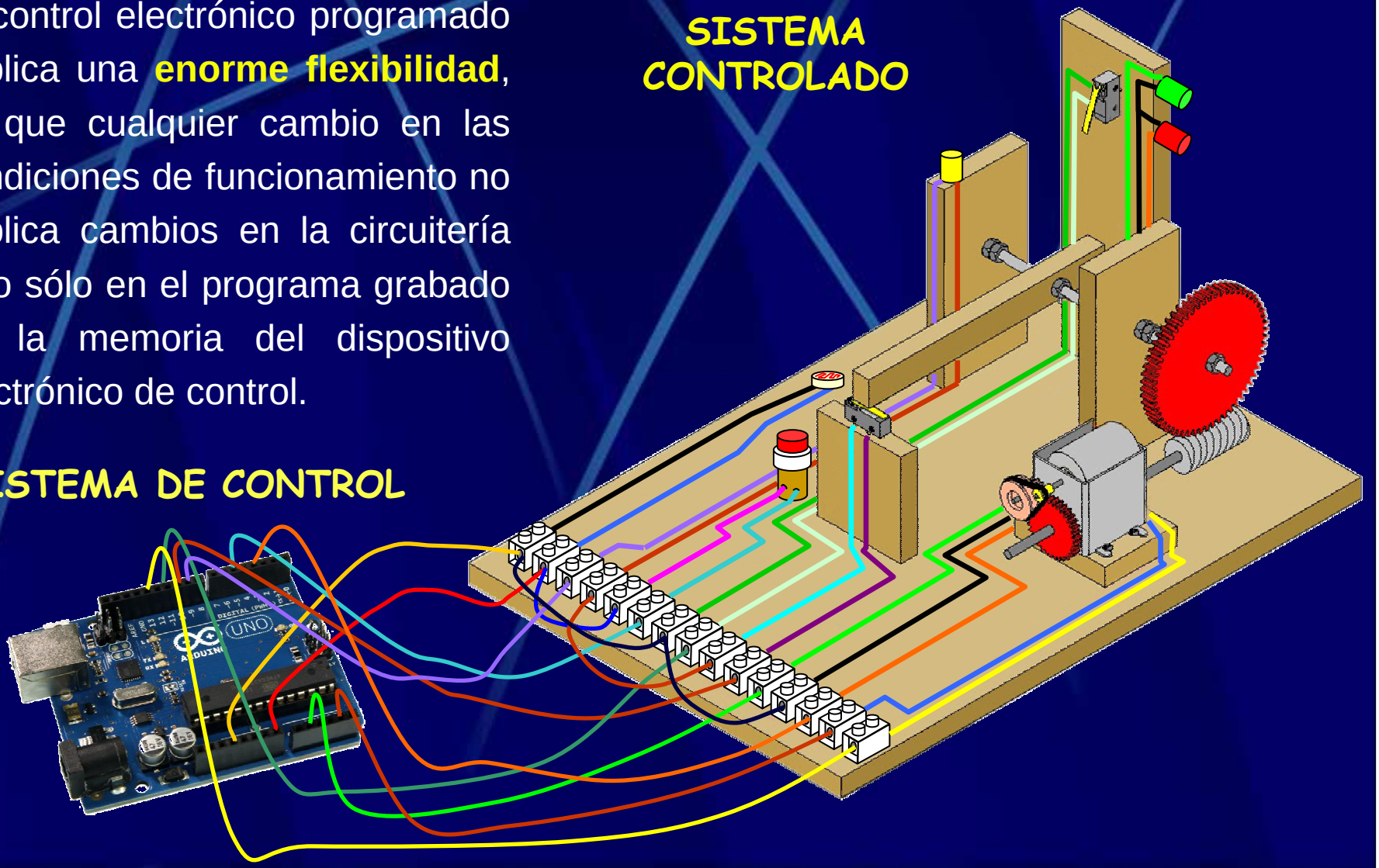


Control programado de un sistema técnico

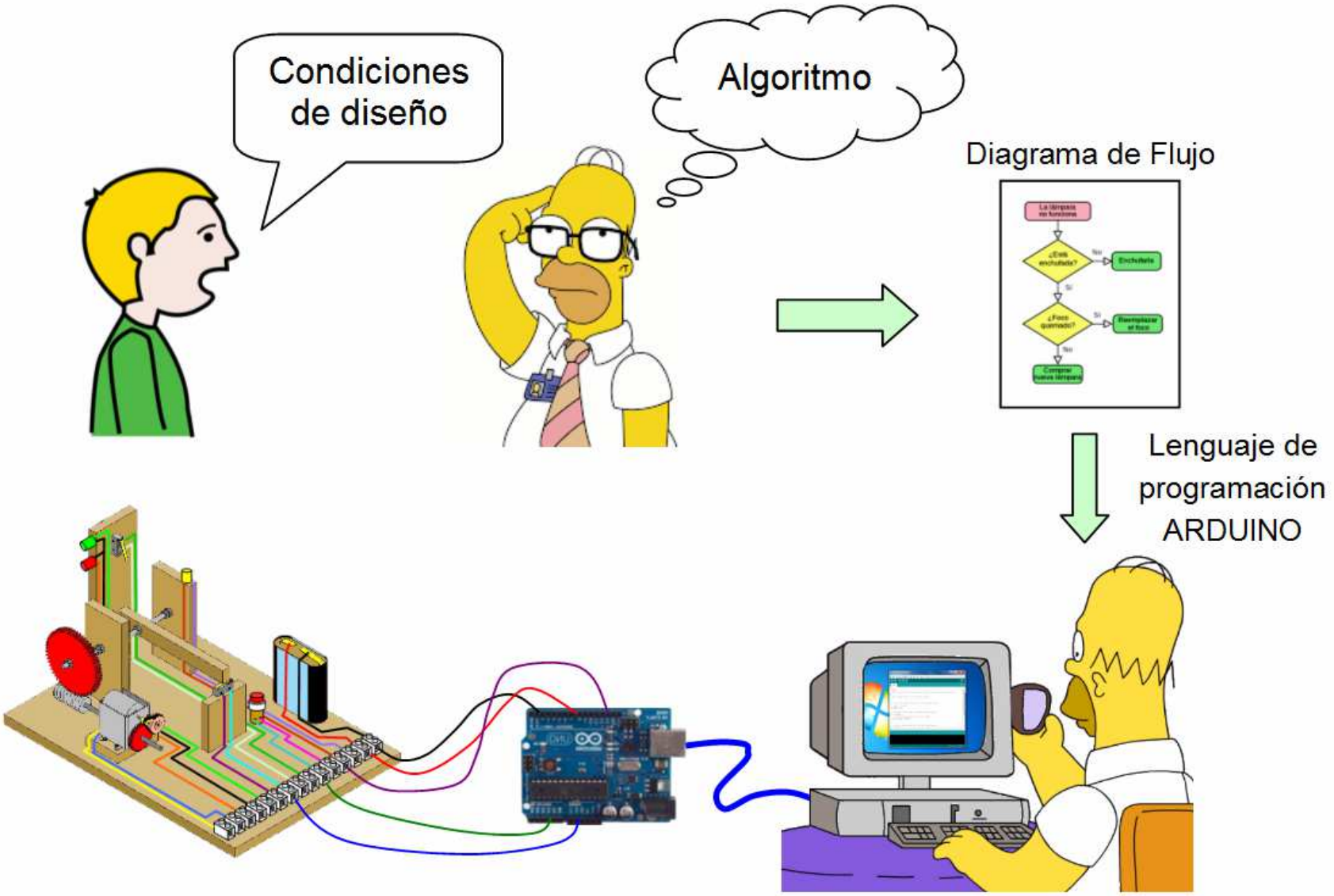
El control electrónico programado implica una **enorme flexibilidad**, ya que cualquier cambio en las condiciones de funcionamiento no implica cambios en la circuitería sino sólo en el programa grabado en la memoria del dispositivo electrónico de control.

SISTEMA CONTROLADO

SISTEMA DE CONTROL



Diseño del control programado: diagramas de flujo



Algoritmo

Un **algoritmo** es un conjunto de instrucciones ordenadas que permiten realizar una actividad (por ejemplo, resolver un problema) mediante pasos sucesivos que incluyen: evaluación de condiciones, toma de decisiones y ejecución de acciones.

Ejemplo: algoritmo para resolver el problema

"Lámpara que no funciona"

1.- Mirar si está enchufada:

No: enchufarla y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 2

Sí: continuar. Ir a 2

2.- Mirar si la bombilla está fundida

Sí: cambiar la bombilla y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 3

No: continuar. Ir a 3.

3.- Mirar si el interruptor está roto:

Sí: cambiar el interruptor y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 4

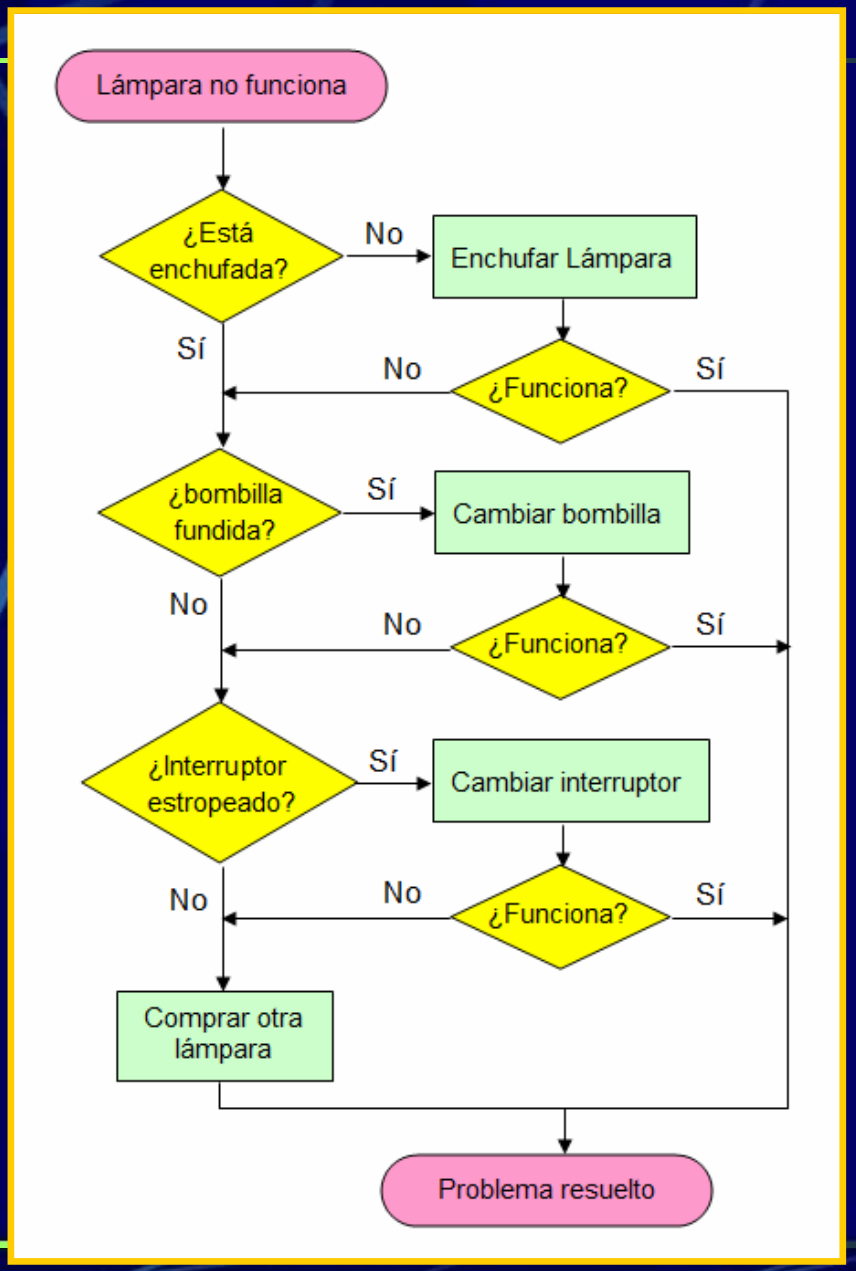
No: continuar. Ir a 4

4.- Comprar una lámpara nueva. Ir a 5

5.- Problema resuelto. Fin

Diagrama de flujo

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



Traducción del algoritmo a diagrama de flujo

Ejemplo: algoritmo para resolver el problema

"Lámpara que no funciona"

1.- Mirar si está enchufada:

No: enchufarla y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 2

Sí: continuar. Ir a 2

2.- Mirar si la bombilla está fundida

Sí: cambiar la bombilla y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 3

No: continuar. Ir a 3.

3.- Mirar si el interruptor está roto:

Sí: cambiar el interruptor y probar si funciona:

Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 4

No: continuar. Ir a 4

4.- Comprar una lámpara nueva. Ir a 5

5.- Problema resuelto. Fin

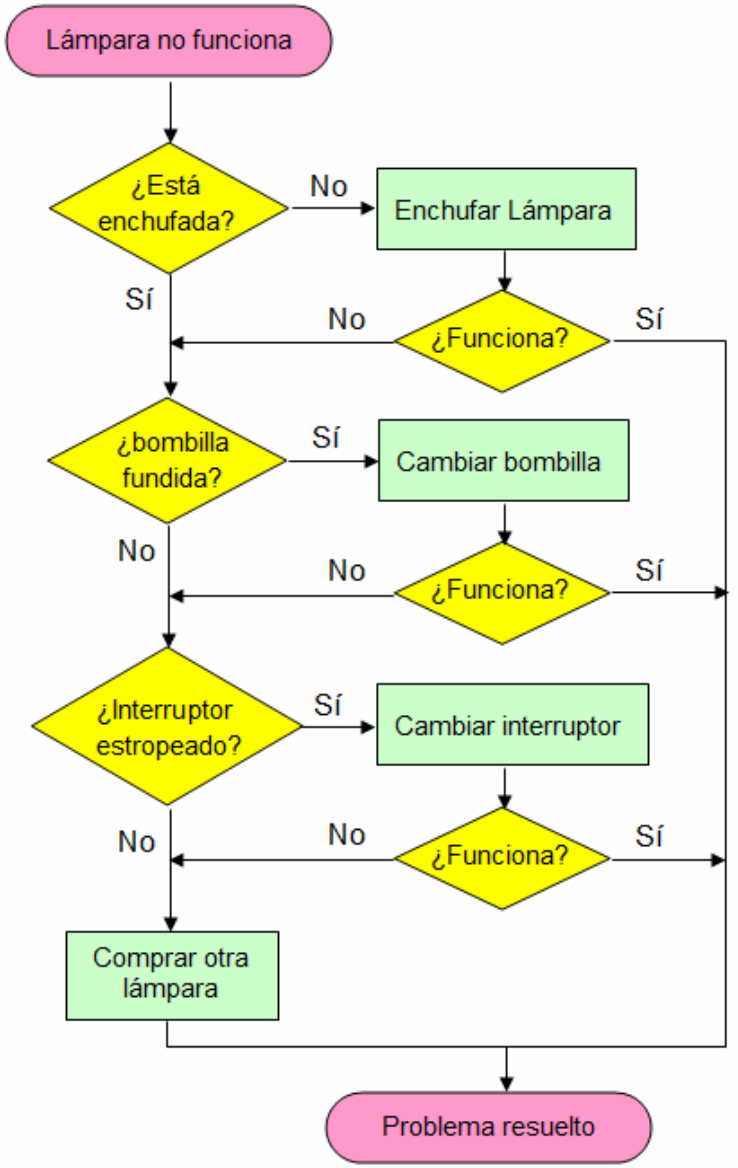


Diagrama de flujo

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

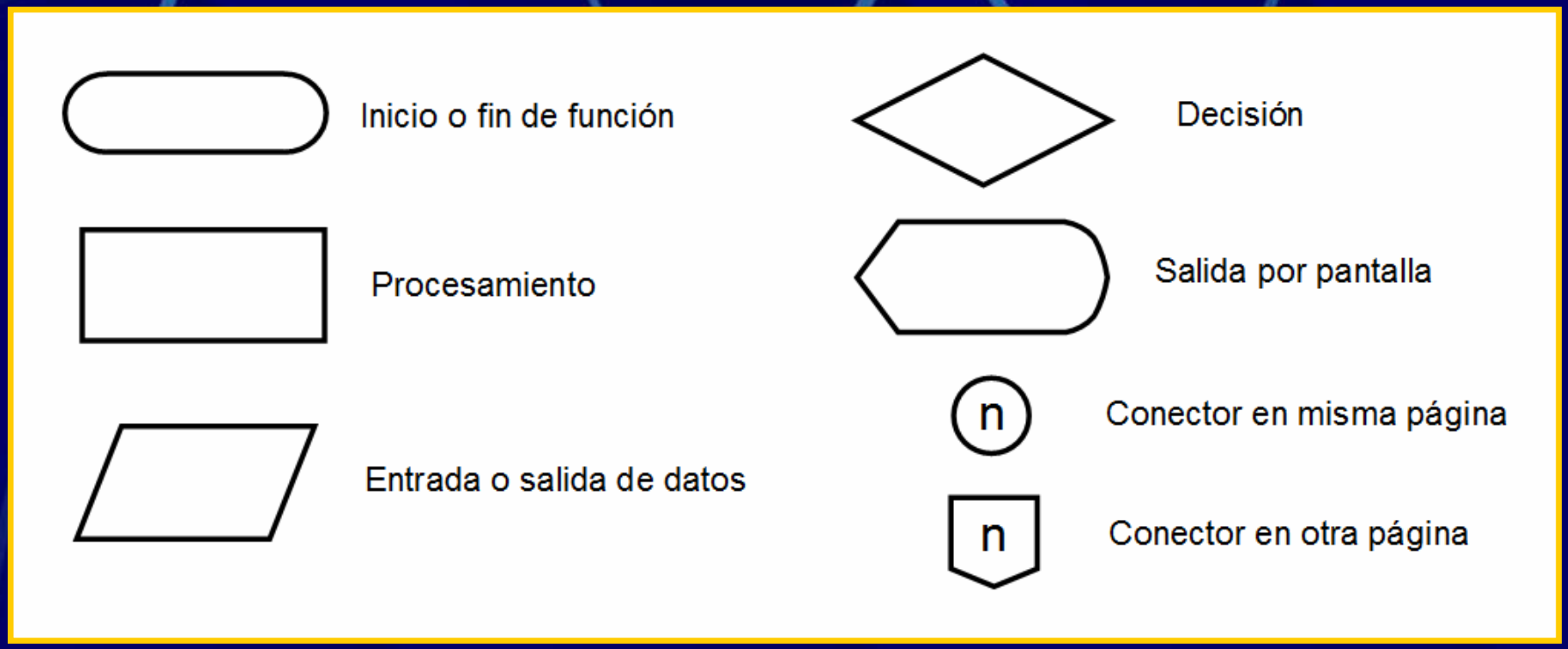
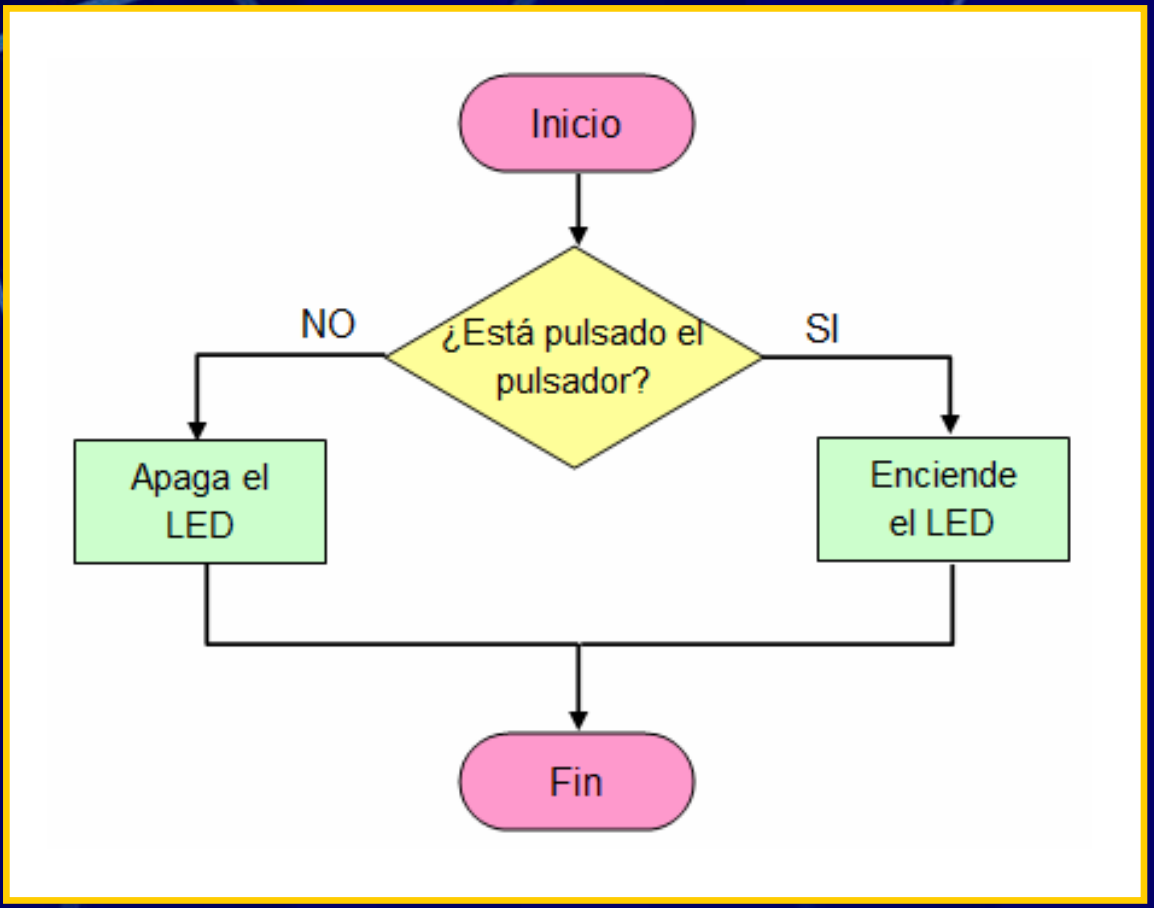


Diagrama de flujo: Ejemplo 1

Ejemplo: programa que mantiene encendido un LED mientras esté pulsado un pulsador y apagado cuando dicho pulsador no está pulsado.



Lógicamente, para que este algoritmo realice la función de forma satisfactoria tiene que ejecutarse una y otra vez de **forma cíclica** (cada vez que llega al final vuelve a empezar), de lo contrario sólo evaluaría el estado del pulsador una vez, y encendería o apagaría el LED una vez.

Diagrama de flujo: Ejemplo 2

Ejemplo: programa que enciende un LED al pulsar un pulsador llamado ON y lo apaga cuando se pulsa un pulsador llamado OFF.

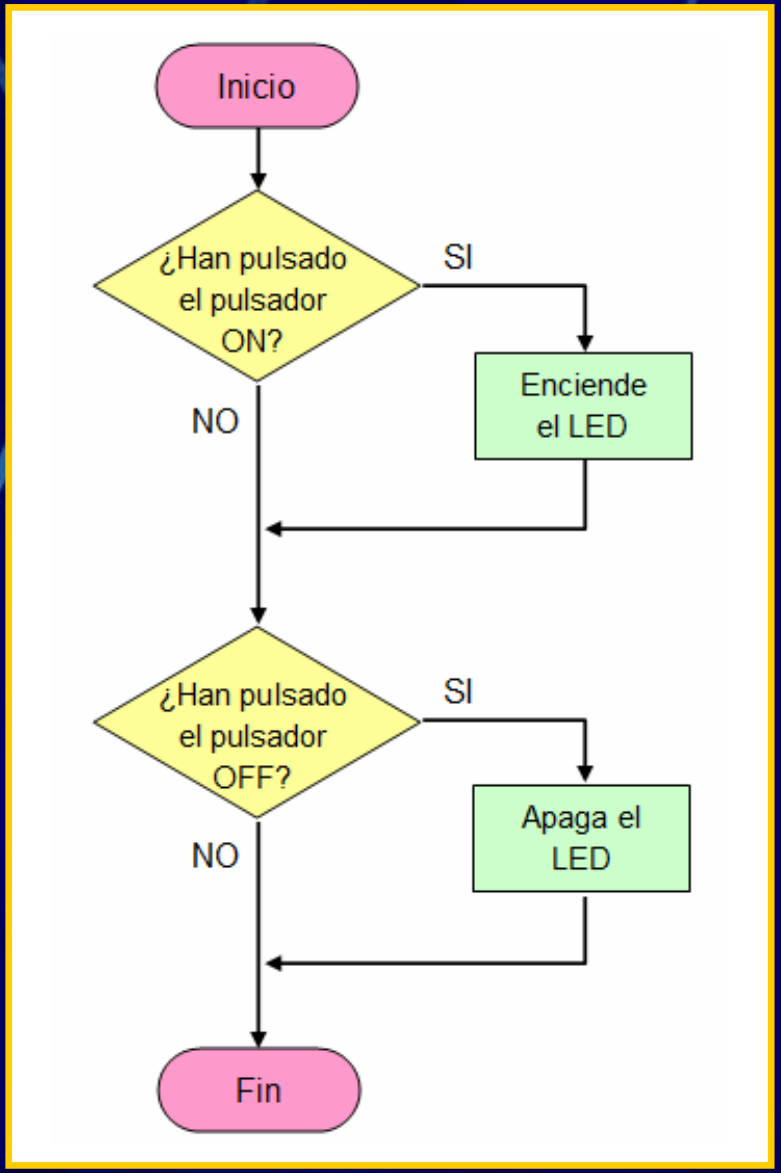


Diagrama de flujo: Ejemplo 3

Ejemplo: programa que cambia el estado de un LED de encendido a apagado o viceversa cada vez que se pulsa un único pulsador.

Sin embargo, en su momento veremos que este algoritmo tiene un problema derivado de lo rápido que ejecuta el programa la placa Arduino. En el breve tiempo que dura la pulsación, Arduino ejecuta multitud de veces el programa, de modo que nunca sabemos si la última vez que lo haga encenderá o apagará la lámpara. Ya veremos su solución.

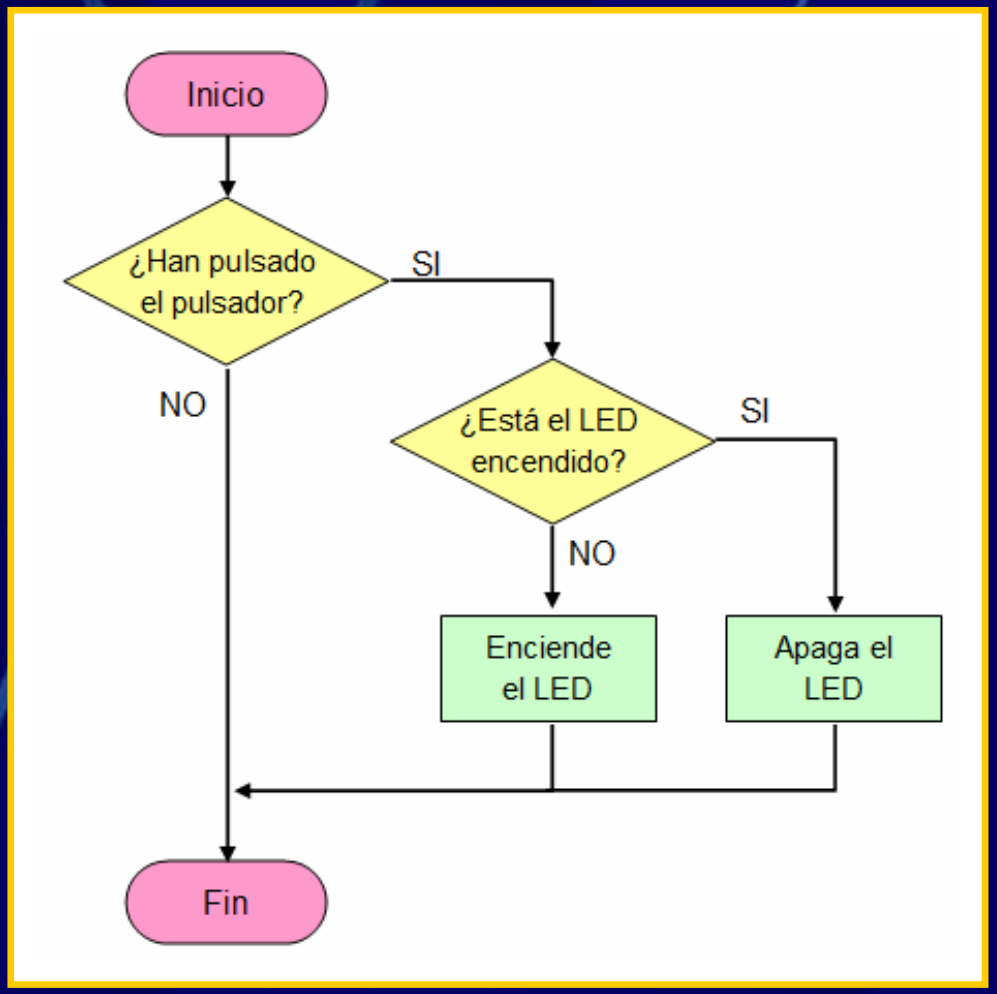


Diagrama de flujo: Ejemplo 4

Ejemplo: programa que se mantiene vigilando si se pulsa un pulsador. Cuando esto ocurre enciende un LED durante 10 segundos y luego lo apaga, quedando de nuevo a la espera de que se pulse el pulsador.

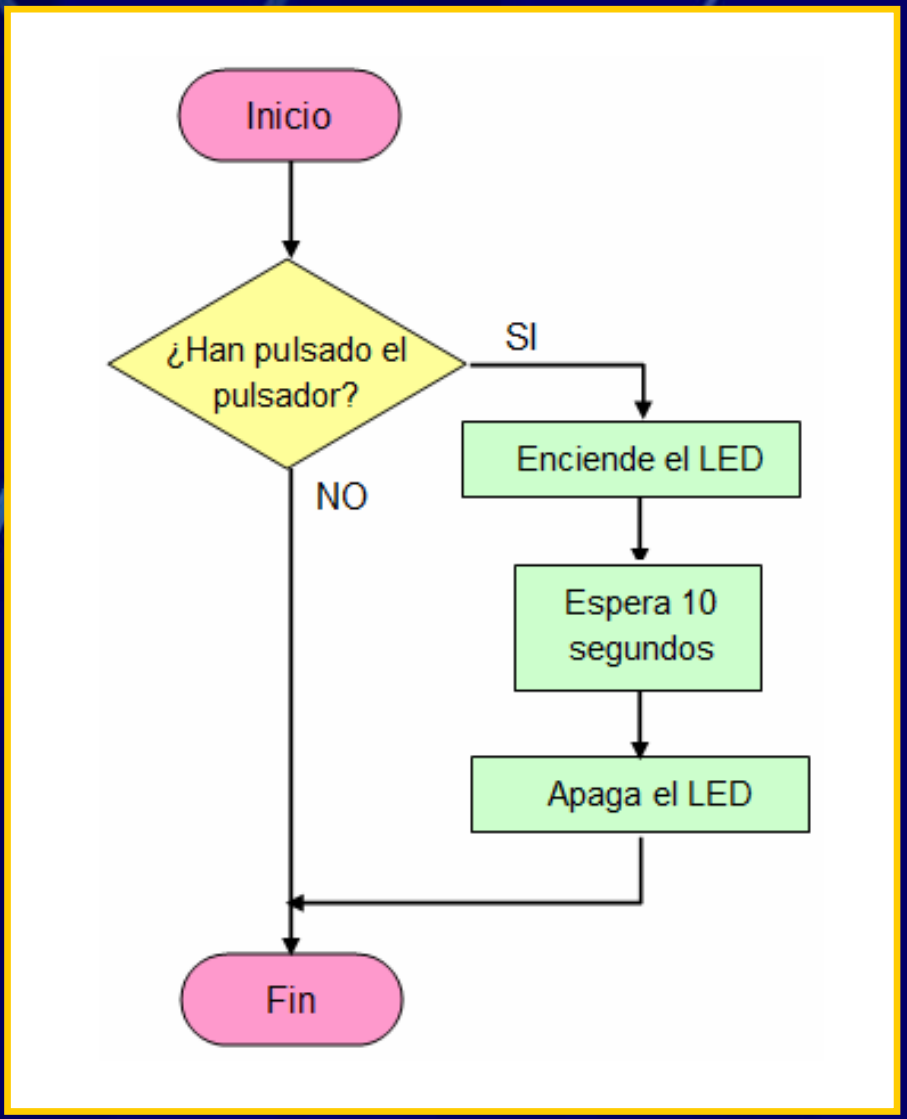
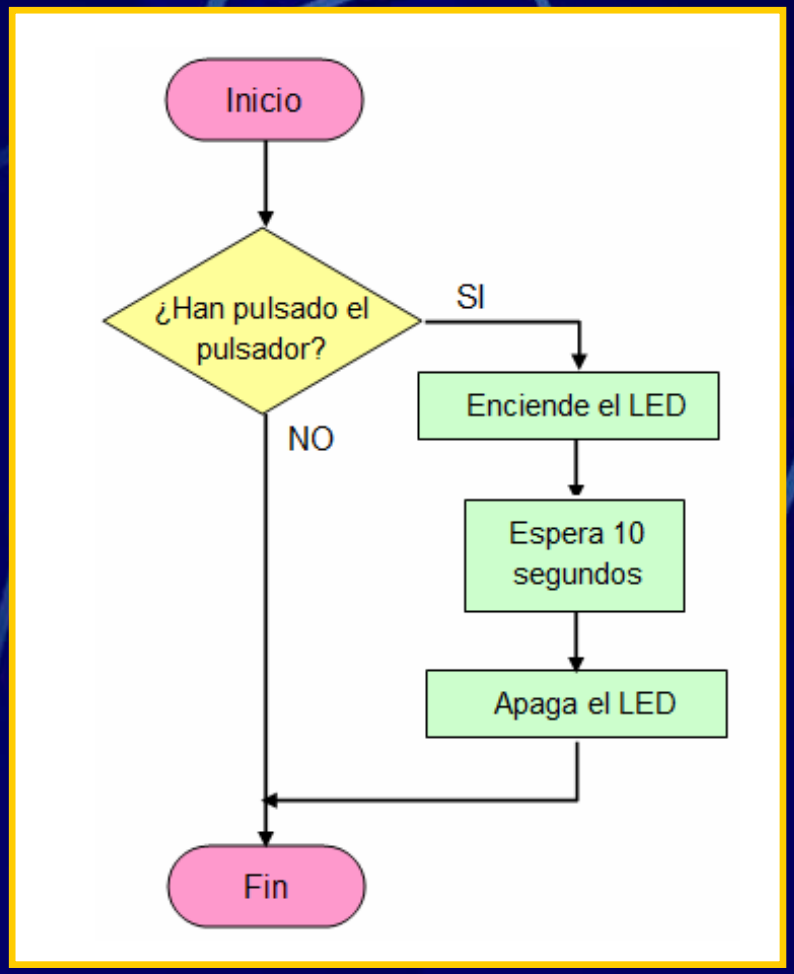


Diagrama de flujo: Ejemplo 4-bis

Otra solución:



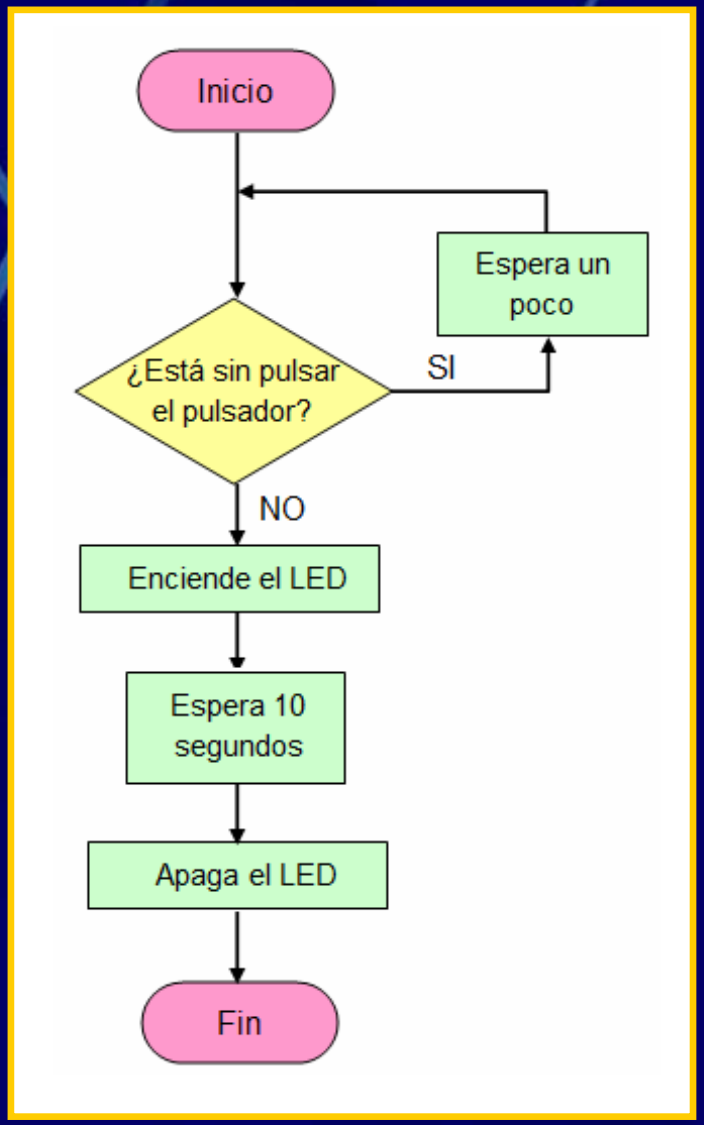


Diagrama de flujo: Ejemplo 5

Ejemplo: programa que vigila dos pulsadores, P1 y P2. Mientras se mantiene pulsado P1 hará que se encienda un LED y si se deja de pulsar lo apagará. Además, si se mantiene pulsado P2 hará que suene un zumbador y si se deja de pulsar lo callará.

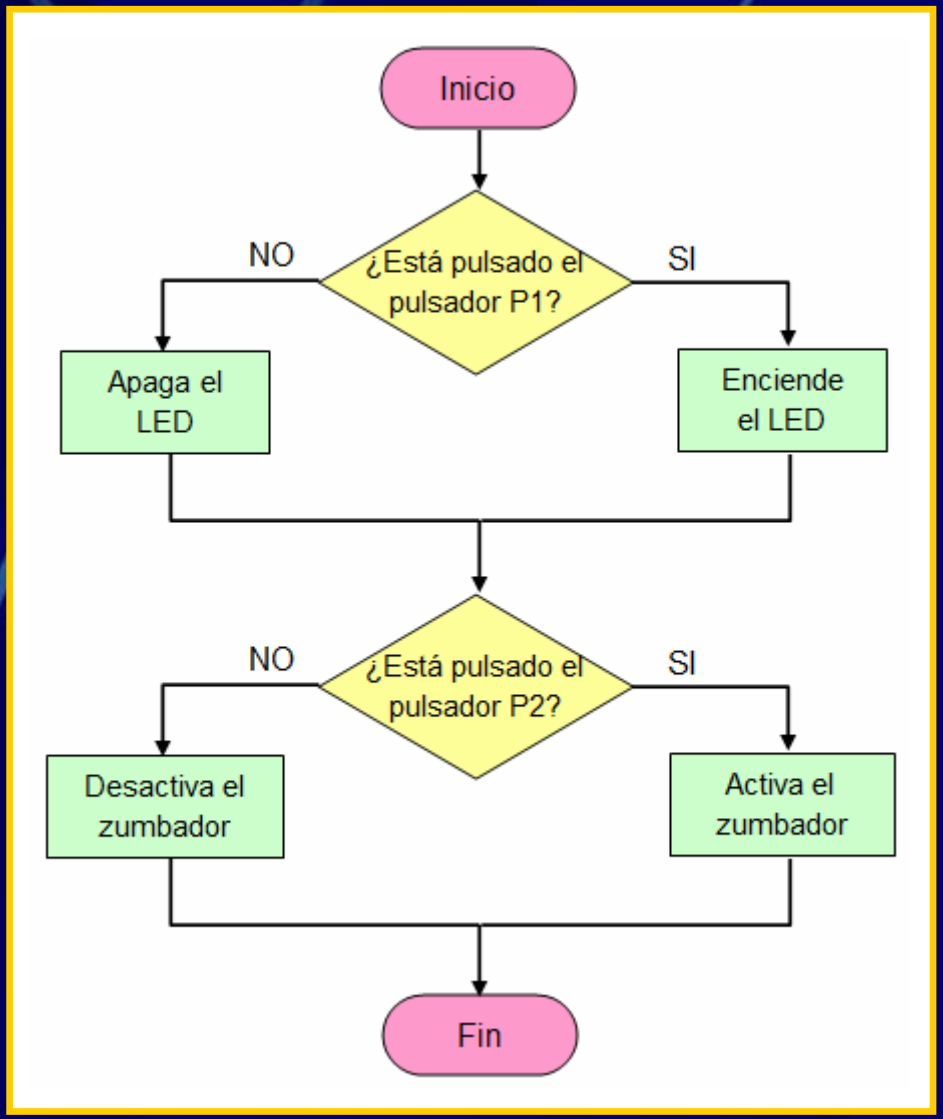


Diagrama de flujo: Ejemplo 6

Ejemplo: programa que se mantiene vigilando si se pulsa un pulsador P1. Cuando esto ocurre enciende un LED durante 10 segundos y luego lo apaga, quedando de nuevo a la espera de que se pulse el pulsador. Sin embargo, si durante esos 10 segundos se pulsa el pulsador P2 el LED se apagará inmediatamente y el sistema vuelve a esperar que se pulse P1.

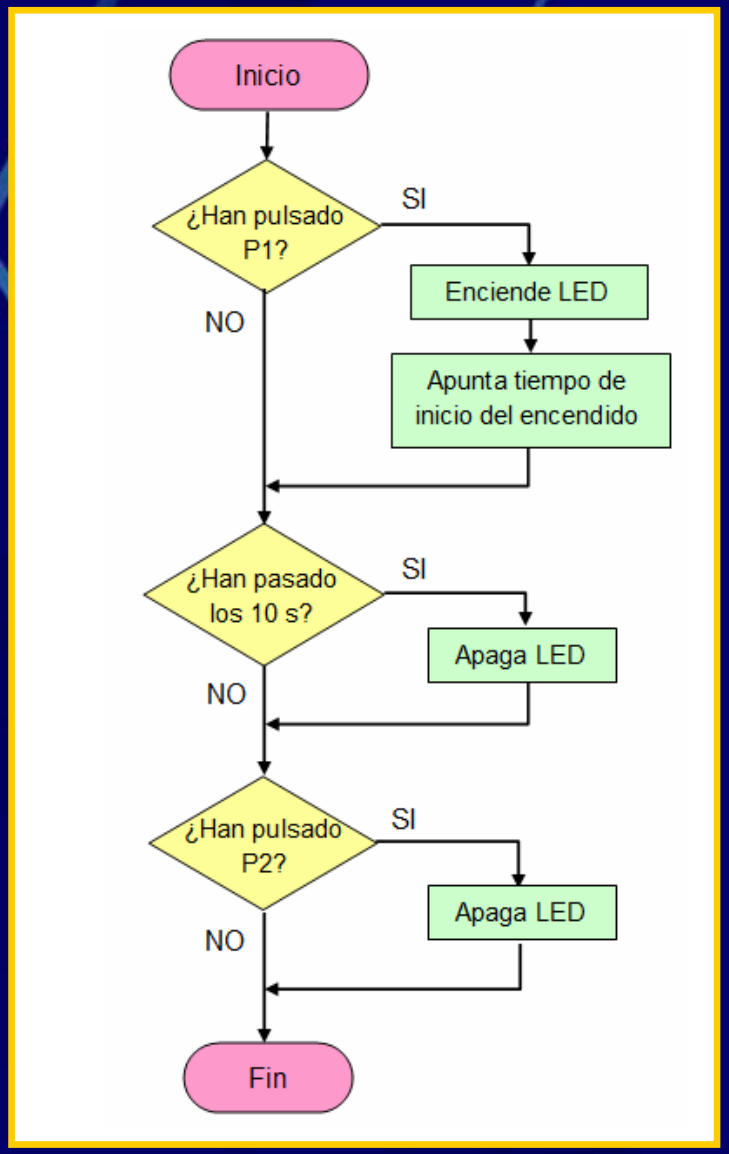


Diagrama de flujo: Ejemplo 7

Ejemplo: programa que se mantiene vigilando si se pulsa un pulsador P1. Cuando esto ocurre enciende un LED durante 10 segundos y luego lo apaga, quedando de nuevo a la espera de que se pulse el pulsador. Sin embargo, si durante esos 10 segundos se pulsa el pulsador P2 pitará el zumbador.

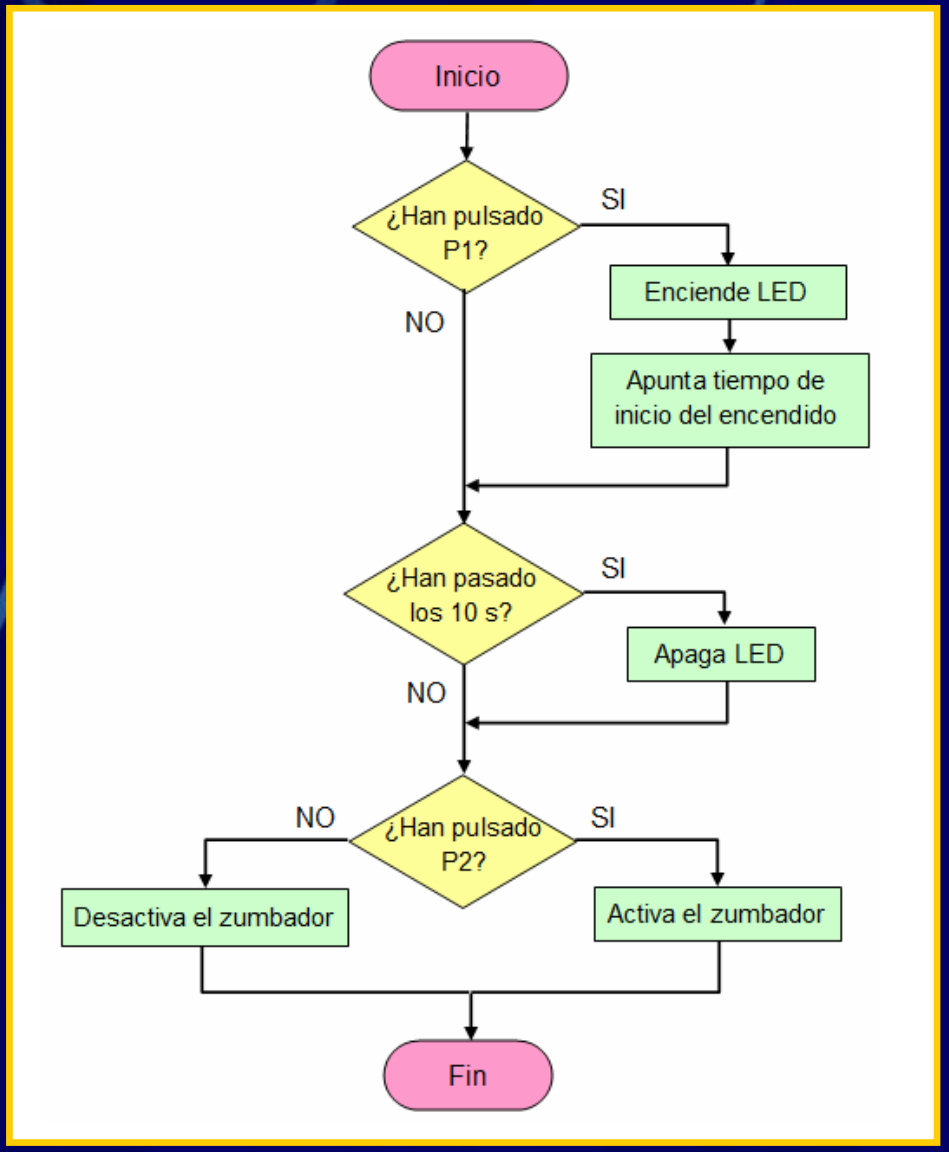


Diagrama de flujo: Ejemplo 7-bis

Otra solución:

Ejemplo: programa que se mantiene vigilando si se pulsa un pulsador P1. Cuando esto ocurre enciende un LED durante 10 segundos y luego lo apaga, quedando de nuevo a la espera de que se pulse el pulsador. Sin embargo, si durante esos 10 segundos se pulsa el pulsador P2 pitará el zumbador.

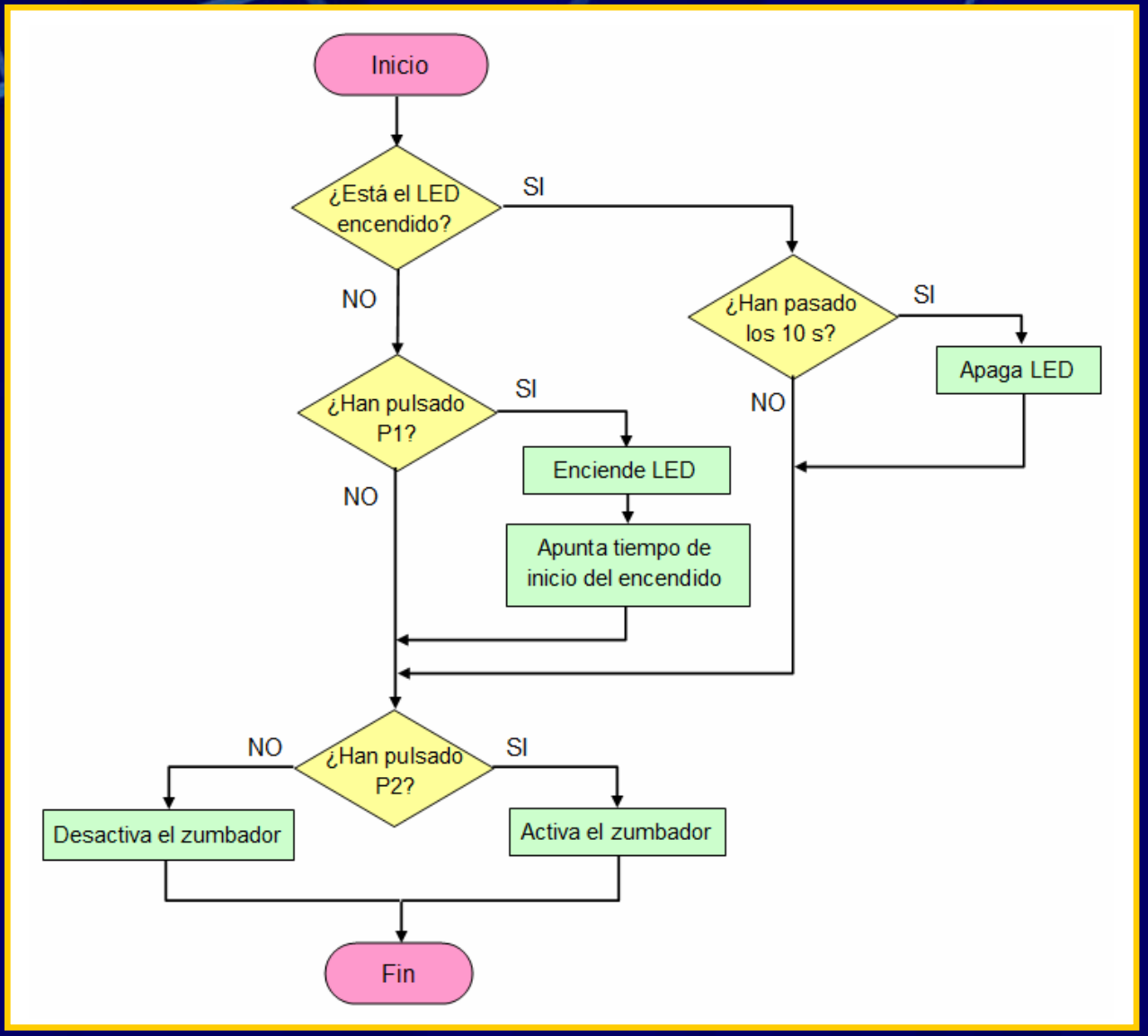
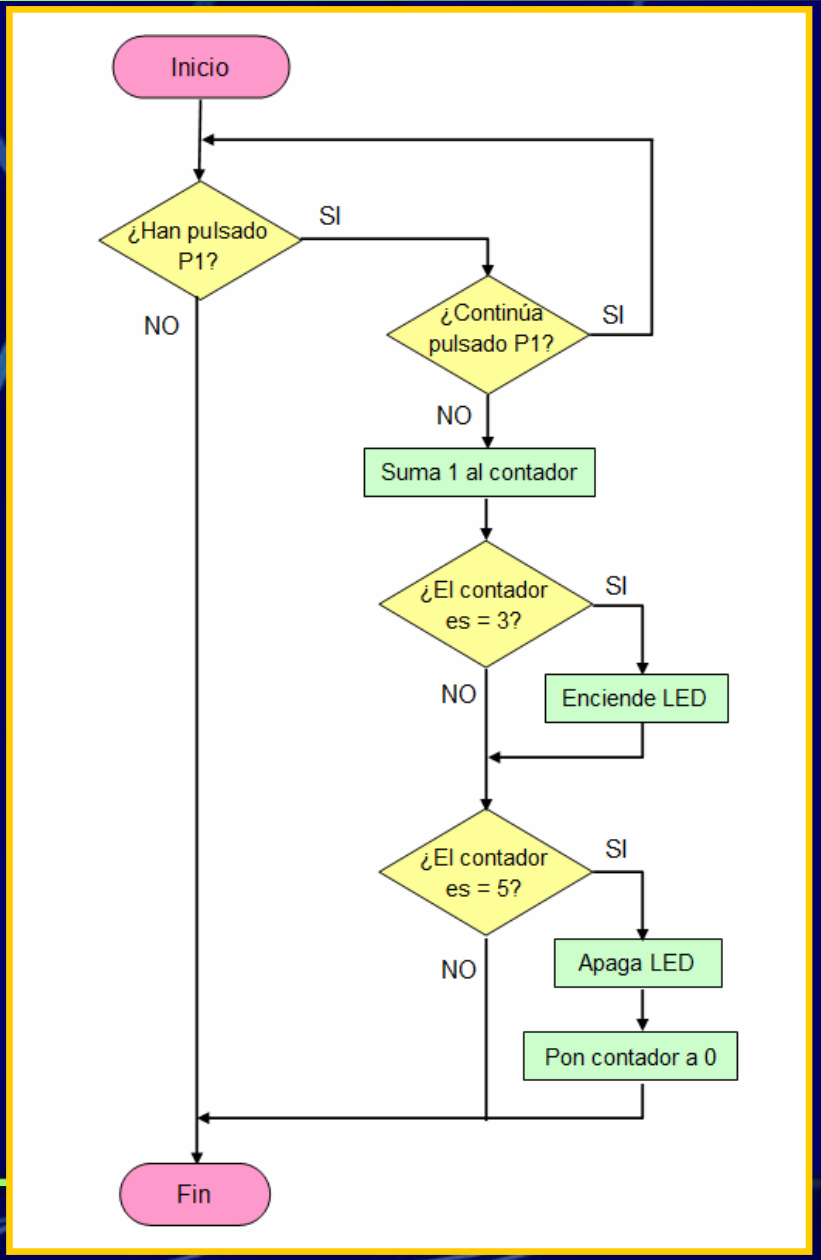
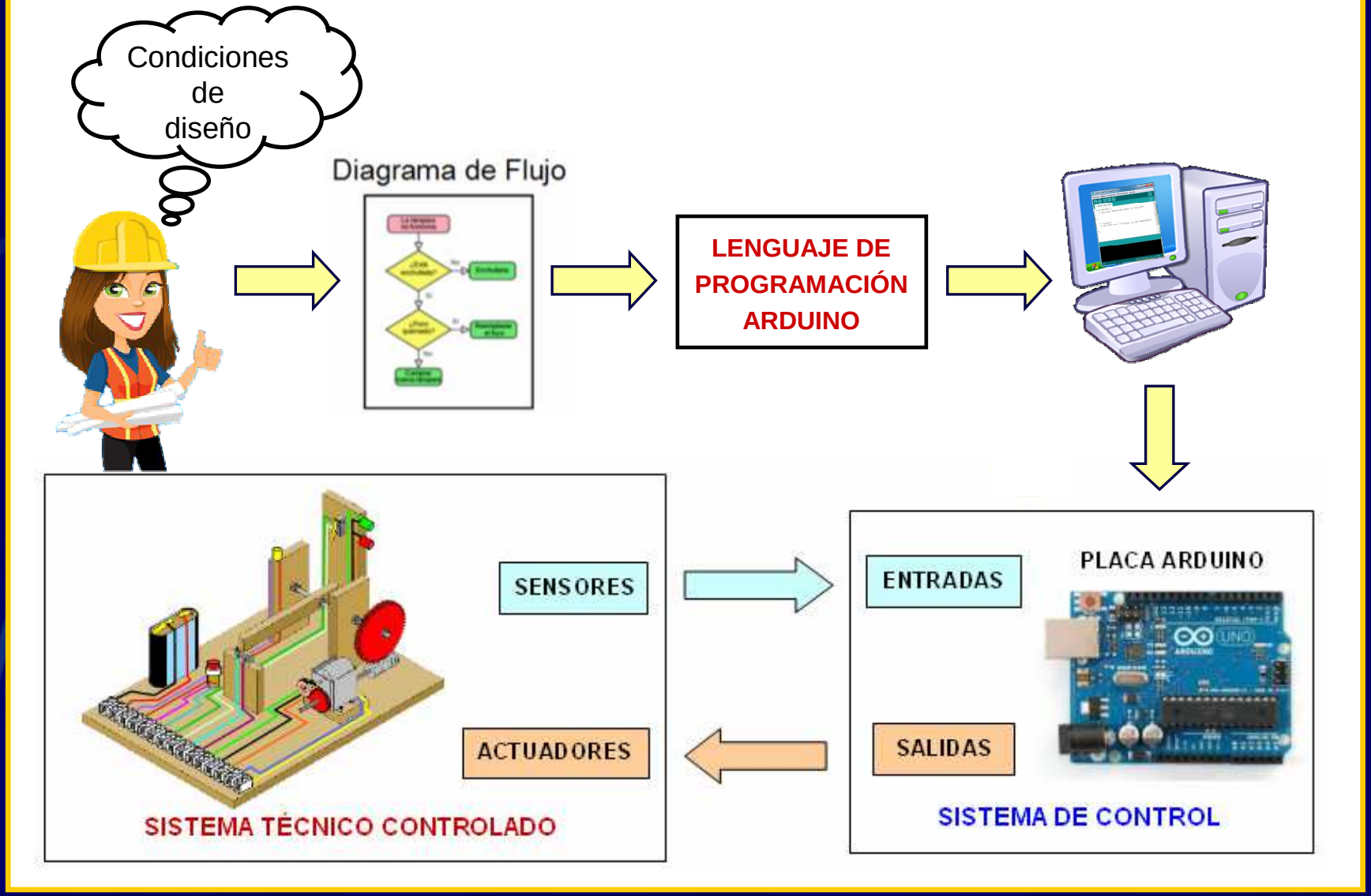


Diagrama de flujo: Ejemplo 8

Ejemplo: programa que se mantiene vigilando si se pulsa un pulsador P1. Cuando se pulsa P1 tres veces (tras cada pulsación hay que dejar de pulsar) se enciende el LED y se queda encendido. Para apagarlo bastará con dos pulsaciones seguidas también en P1.

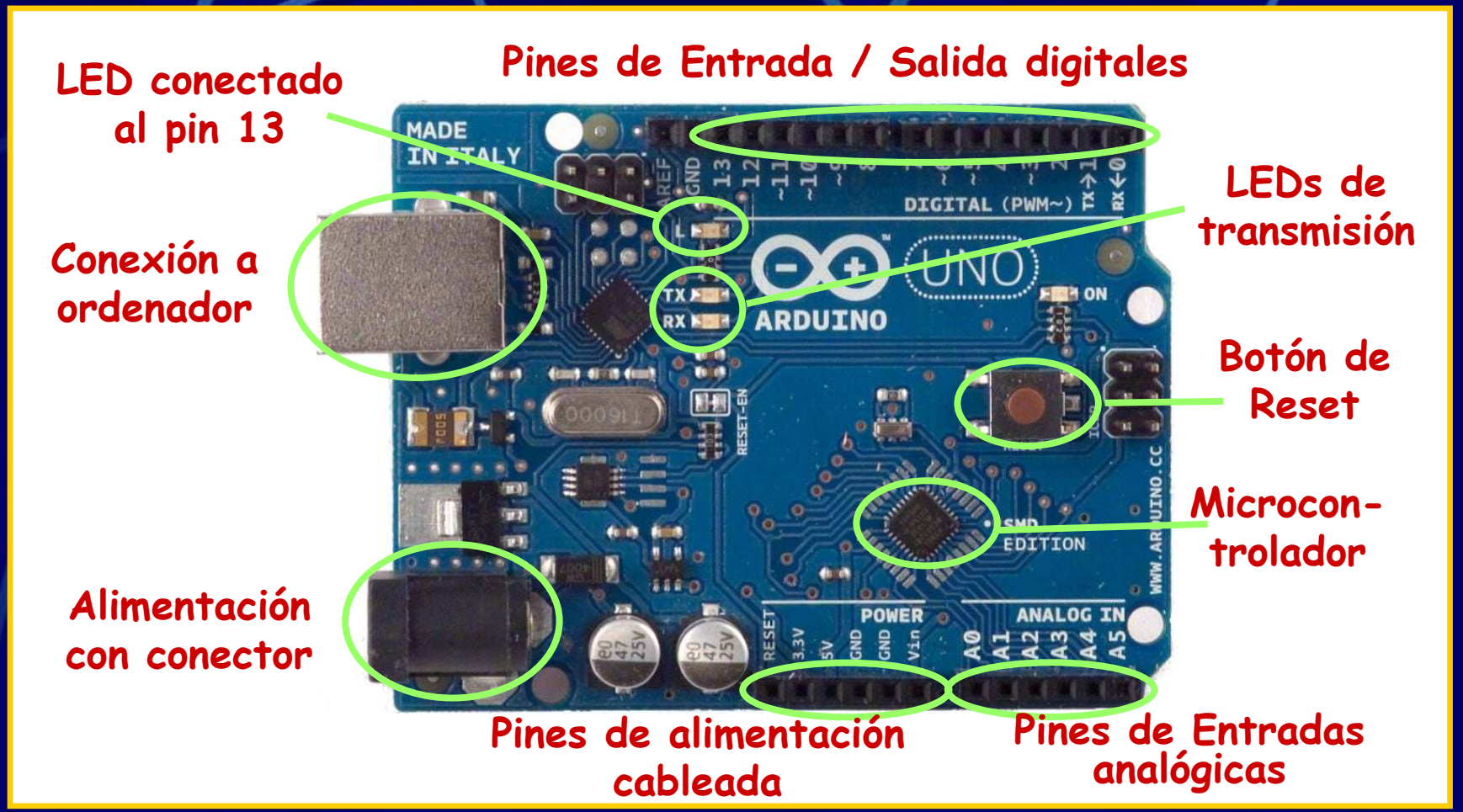


Control programado con ARDUINO



La placa electrónica ARDUINO

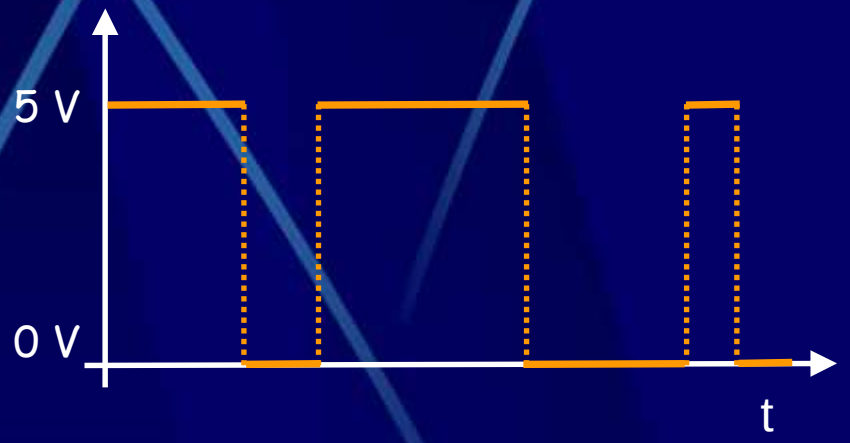
Existen varios modelos de placas ARDUINO. Nosotros disponemos de la Arduino UNO y la Arduino MEGA (que dispone de más pines de conexión con el exterior).



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.

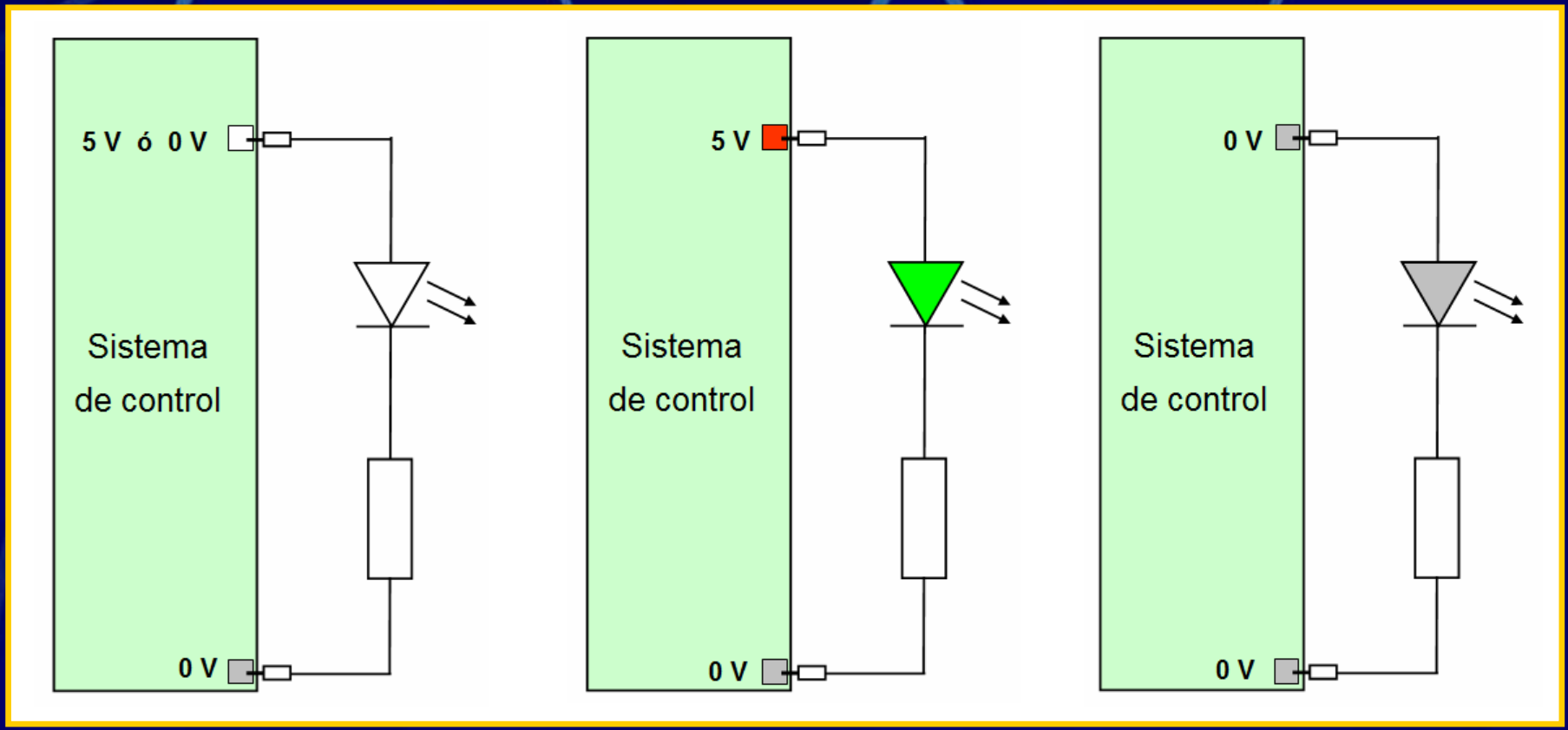
En los microcontroladores, estas señales son **tensiones**, que pueden tomar dos valores: alto y bajo, que suelen ser **5 V** y **0 V**.



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

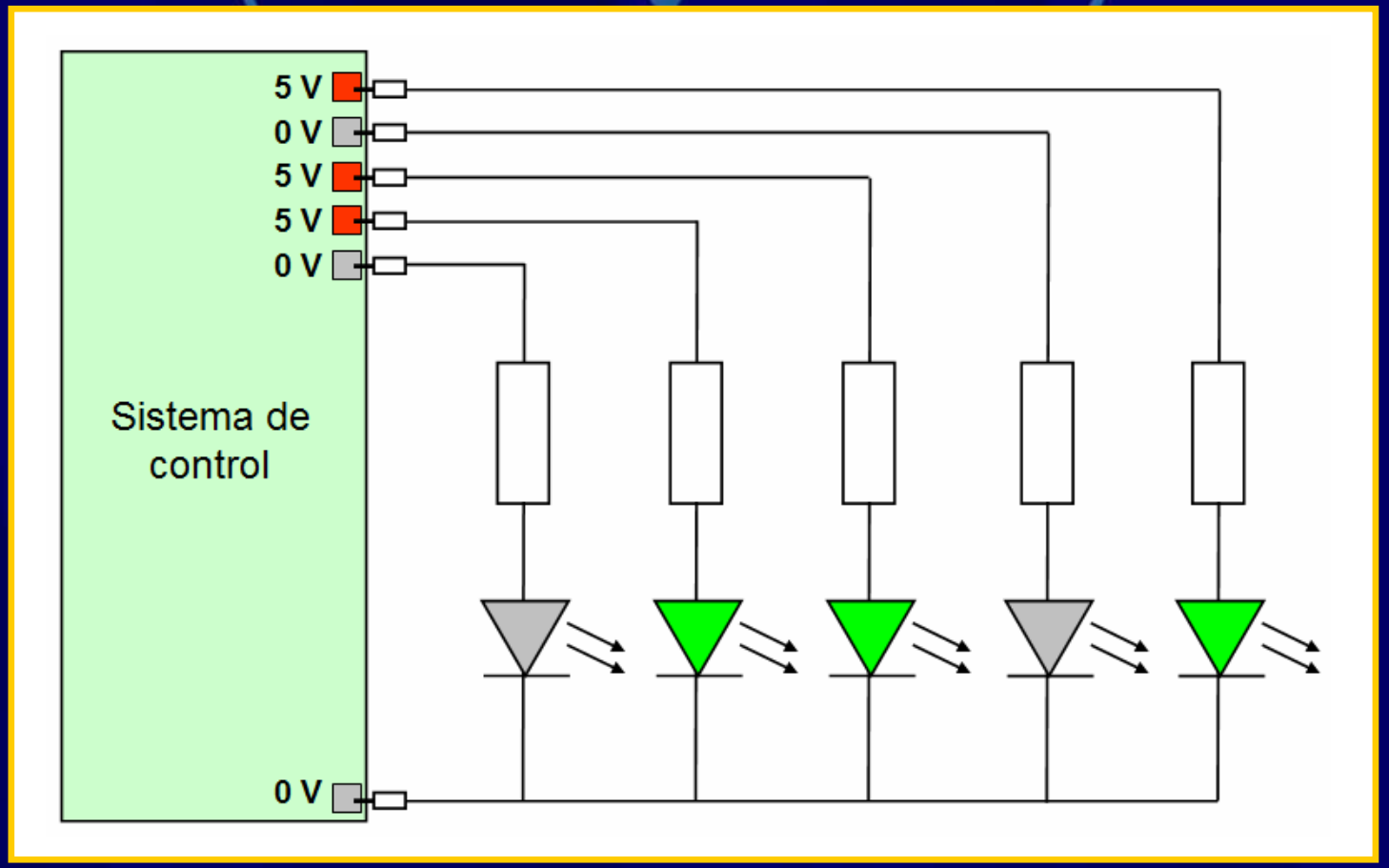
Salidas digitales en un microcontrolador

En los sistemas de control programado, un sistema de control informático (microcontrolador) ejecuta un programa almacenado en su memoria. El sistema se encarga de colocar en las salidas un valor de tensión de 0 V ó 5 V según las instrucciones del programa.



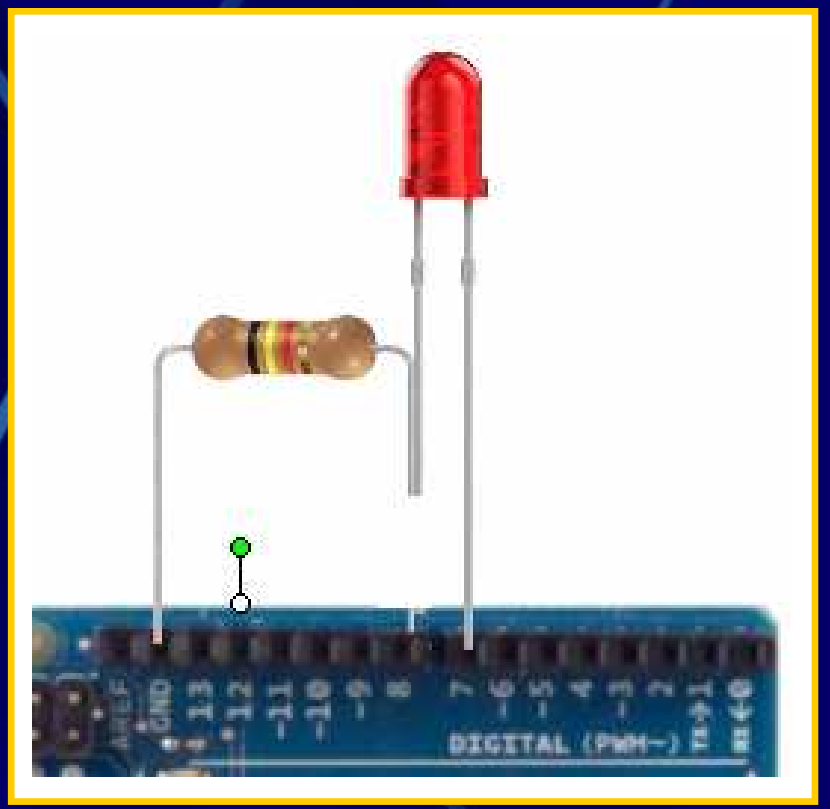
Salidas digitales en un microcontrolador

Lógicamente, el sistema de control no dispone únicamente de una salida digital, sino de muchas, cada una de las cuales controla un actuador de forma independiente.

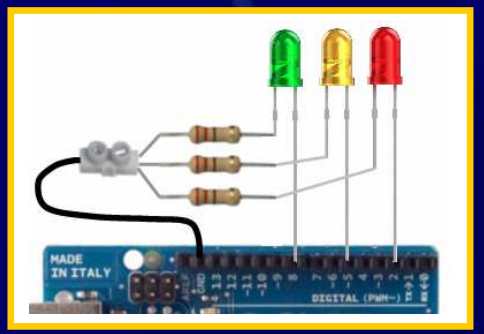


Salidas digitales en Arduino

Los pines de salida digitales de Arduino van identificados por un número, que en la placa **Arduino UNO**, son de 0 a 13.
Otros pines importantes, de los que hay varios en la placa, son los pines GND (abreviatura de “ground”, que es “tierra” en inglés americano y que es el punto que se toma como referencia de tensiones en un circuito, es decir, 0 V).



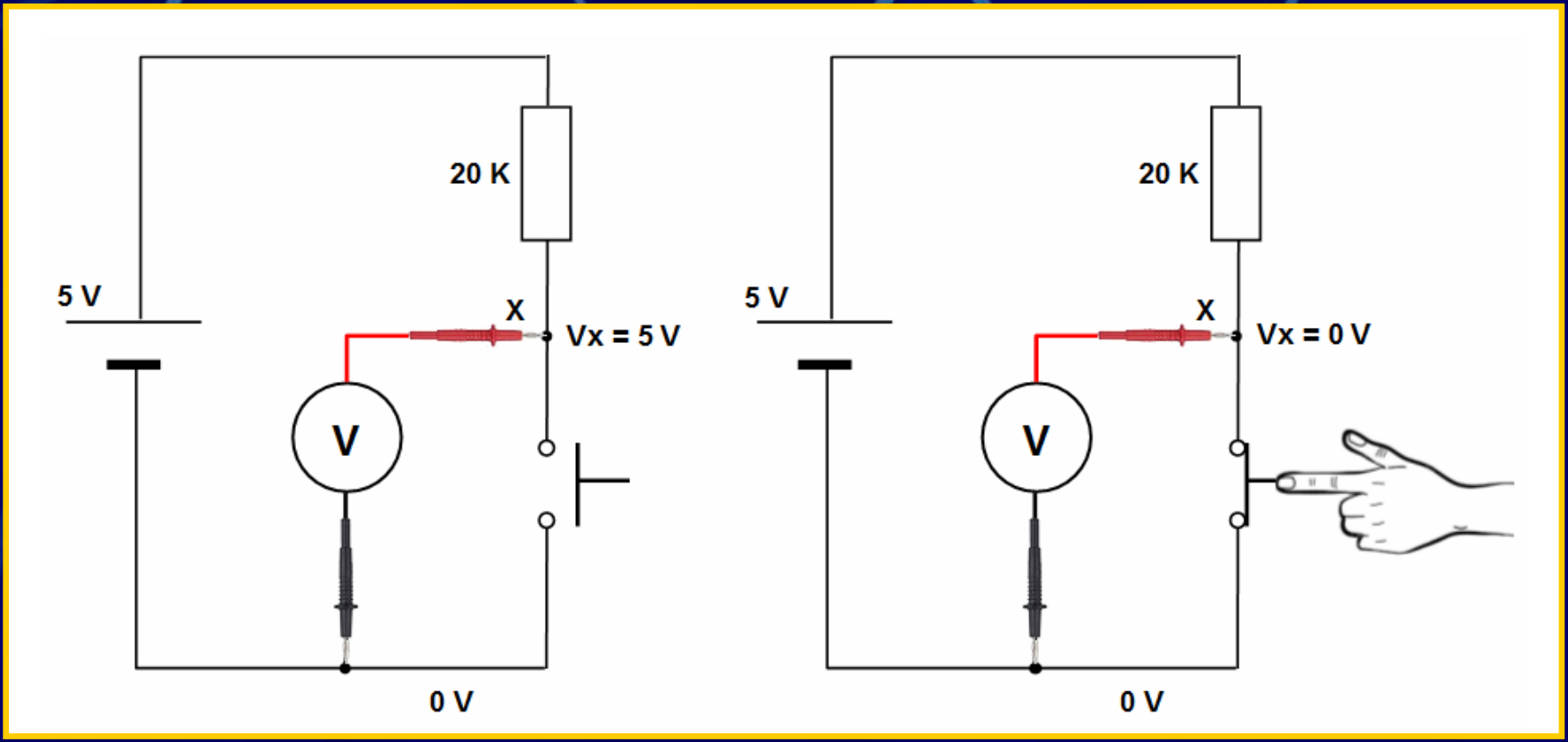
En la imagen de la figura adjunta, hemos conectado un LED, con su correspondiente resistencia en serie, entre el pin número 7 y GND.



Puede utilizarse un mismo GND para varios elementos (LEDs, zumbadores, pulsadores, etc.

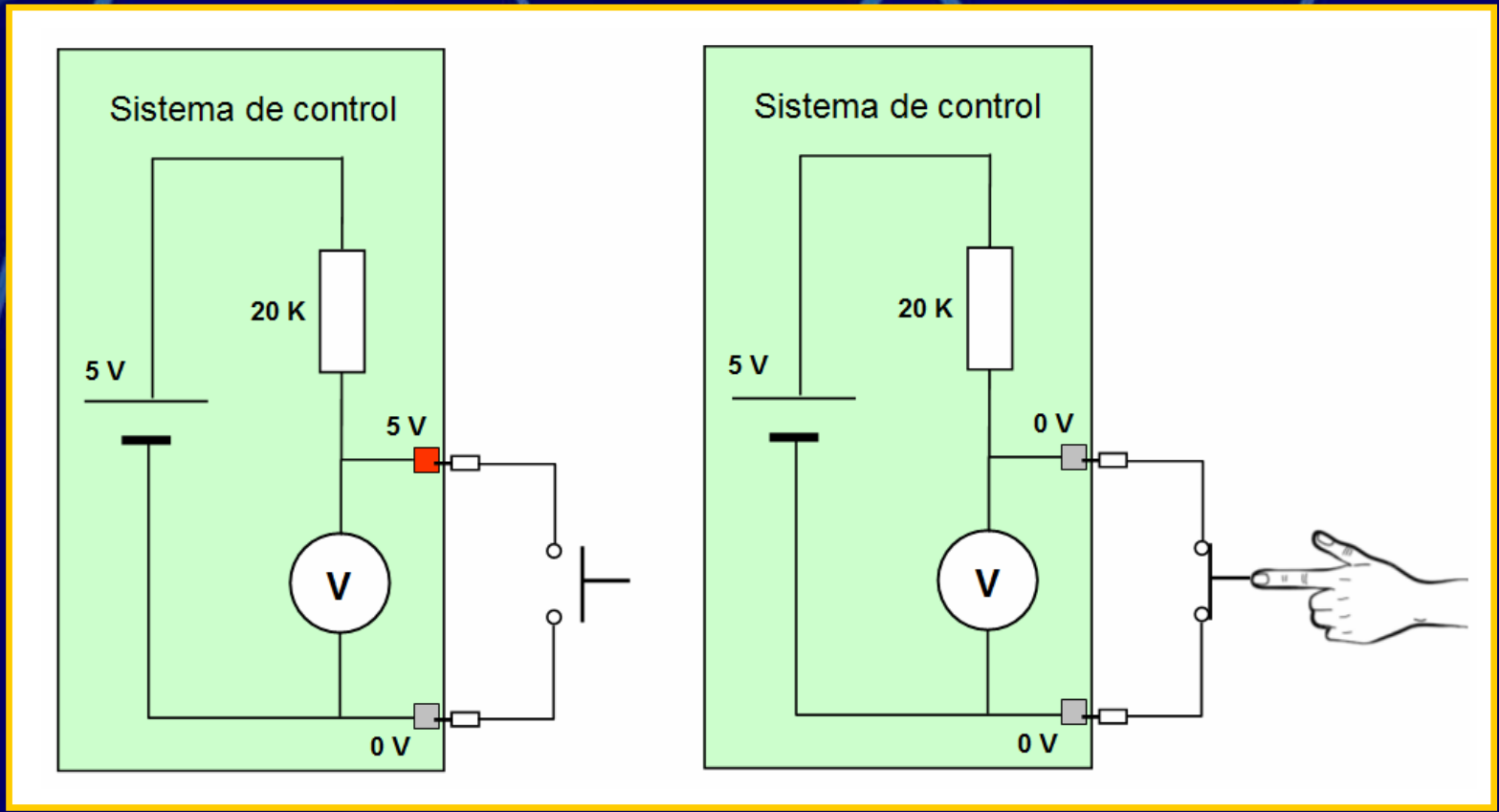
Entradas digitales

Una forma de saber si un pulsador o un interruptor está abierto o cerrado es con el montaje de la figura y midiendo con un voltímetro la tensión en el punto X. Si el pulsador está abierto no circula corriente y la tensión será 5 V. Si el pulsador está cerrado circula corriente y la tensión medida por el voltímetro será 0 V.



Entradas digitales en un microcontrolador

En los sistemas de control programado la función del voltímetro la hace el propio microcontrolador que es capaz de distinguir si en los pines donde se conectan los sensores (pulsadores, finales de carrera, etc.) hay un nivel alto de tensión (cercano a 5 V) o un nivel bajo (cercano a 0 V). La resistencia también la incluye.

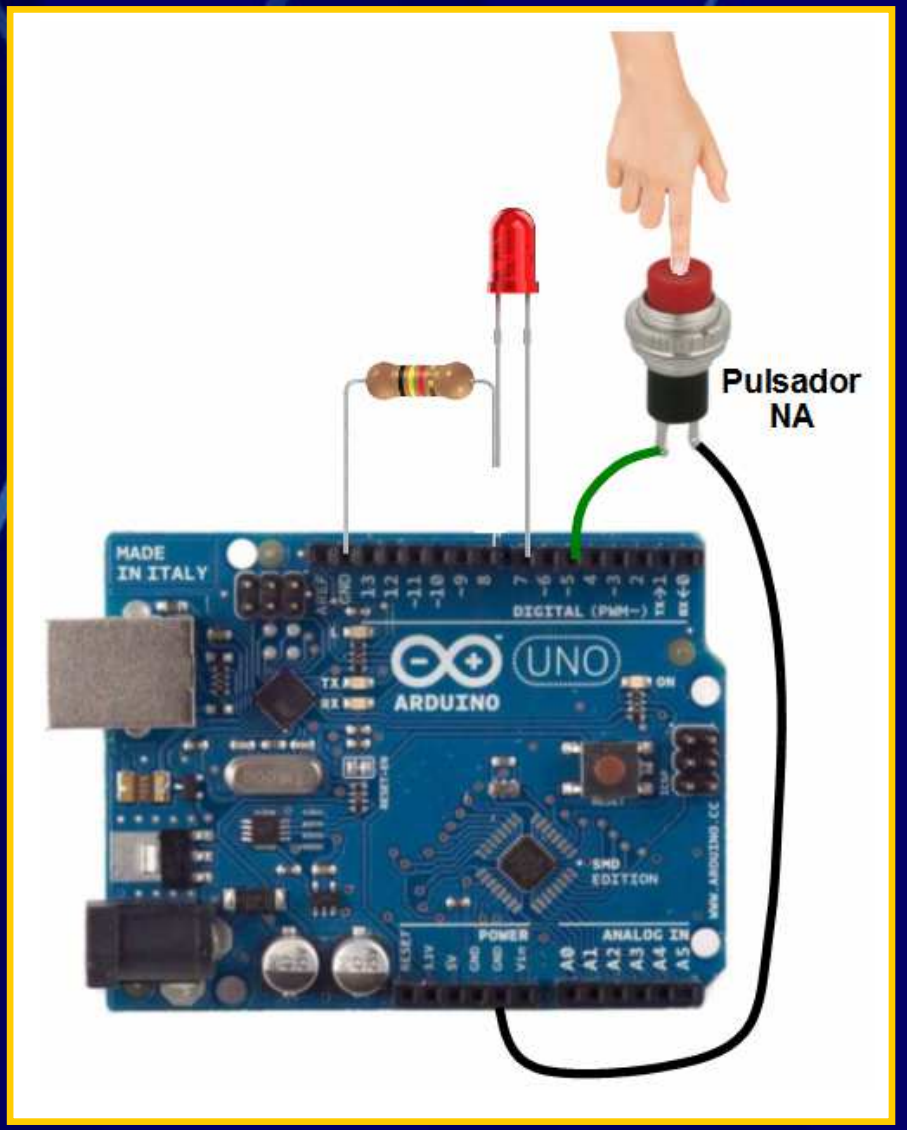


Entradas digitales en Arduino

Los pines de entrada digitales de Arduino son los mismos que los de salida. Para usarlos como entrada o salida se define su modo con una instrucción llamada pinMode() que veremos luego.

En la figura hemos conectado un LED (con su resistencia) entre el pin 7 y GND y un pulsador NA entre el pin 5 y GND.

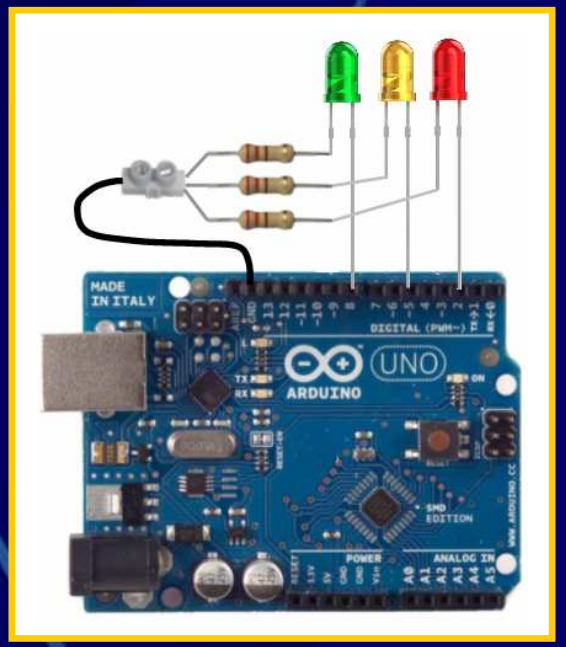
Para que el funcionamiento sea correcto tendremos que definir el pin 7 como salida y el pin 5 como entrada en nuestro programa.

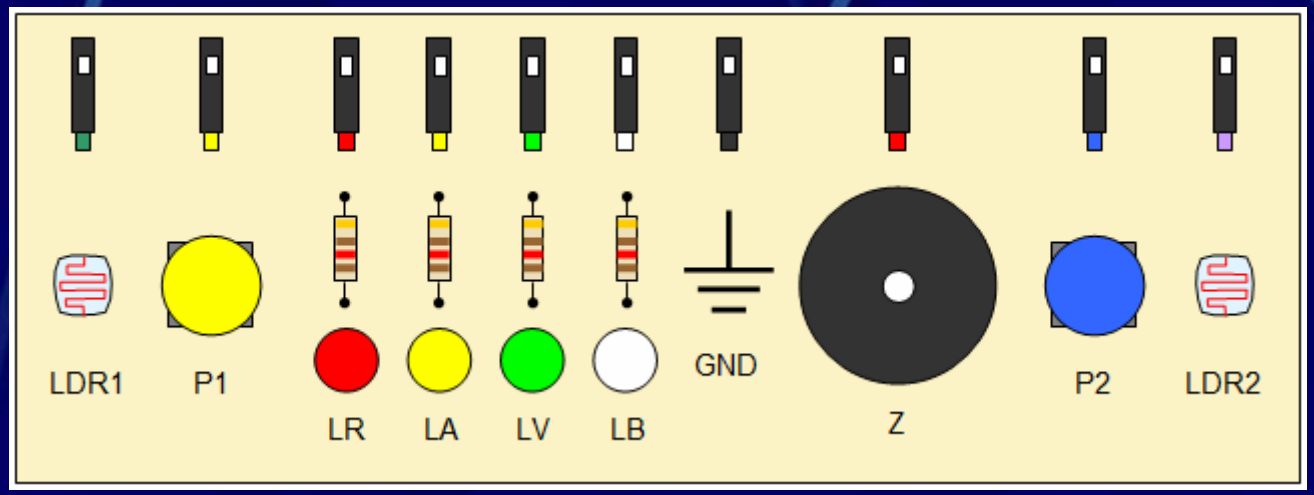


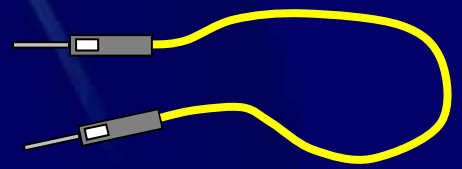
Uso del entrenador con Arduino

Para hacer nuestras prácticas, podríamos conectar los componentes (actuadores y sensores) directamente a los pines de la placa Arduino. No obstante, esto suele dar lugar a errores en las conexiones y a que se pierda más tiempo en los montajes de las prácticas.

Vamos a trabajar con un **entrenador**, que se conectará con la placa Arduino mediante conectores macho-macho.



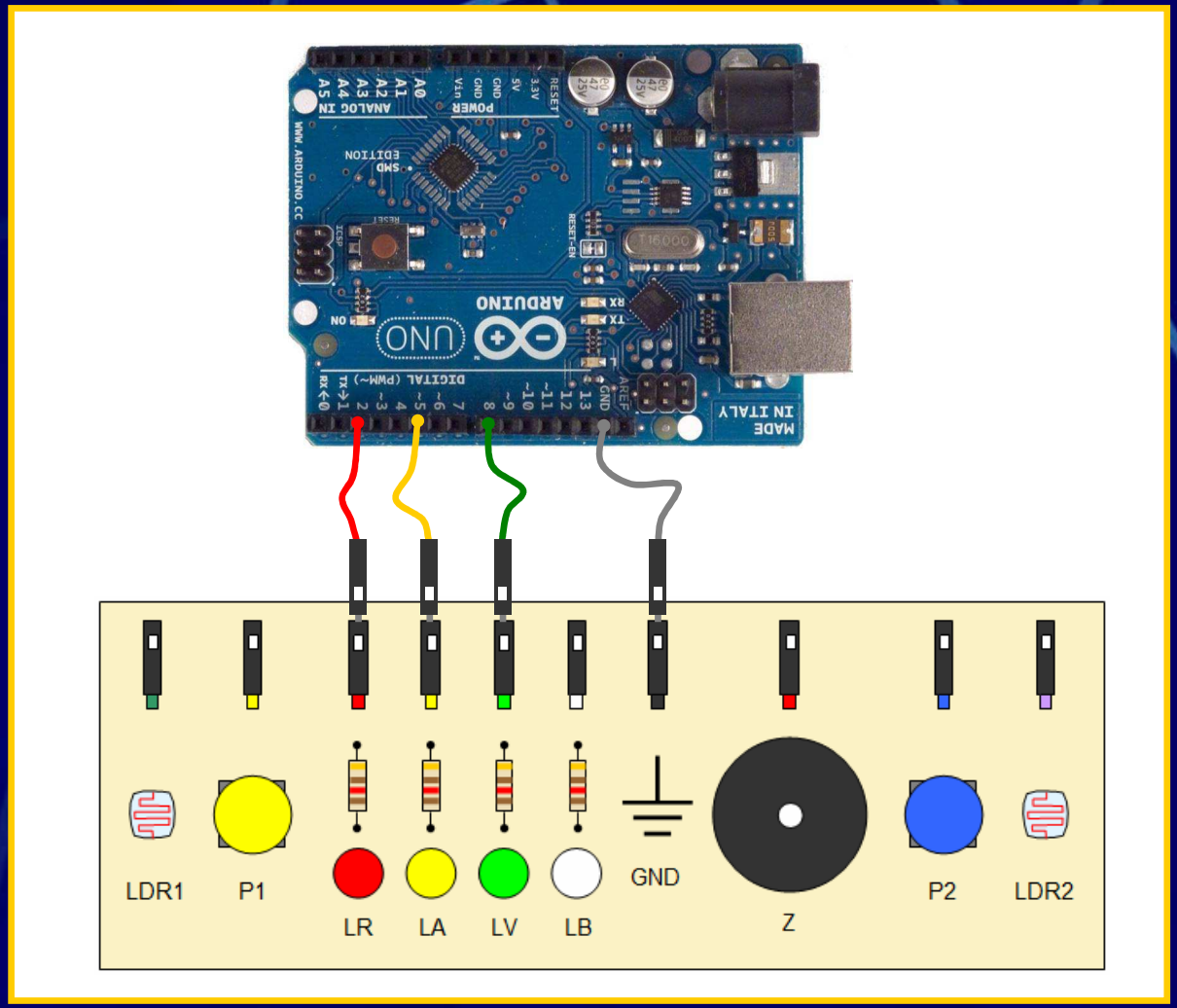


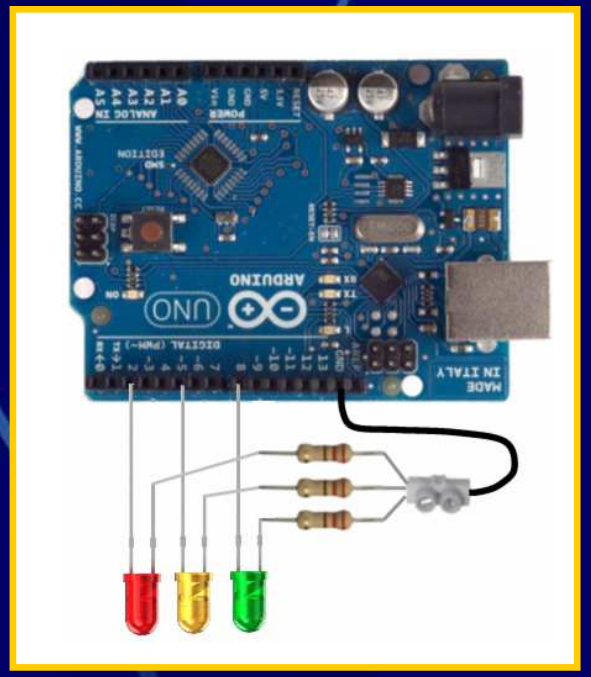


Conector macho-macho

Uso del entrenador con Arduino

Realicemos el montaje de la figura con el entrenador.

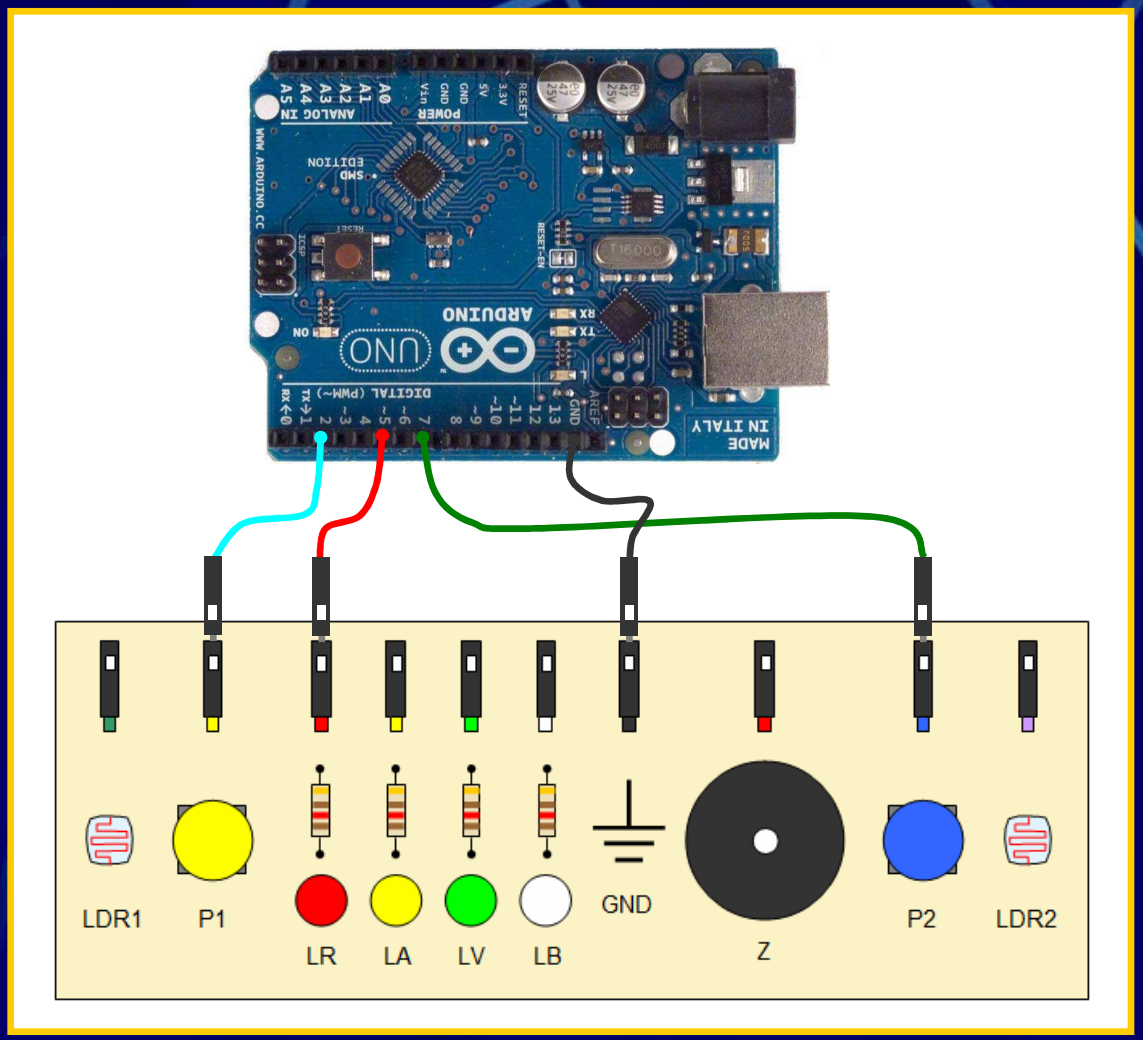


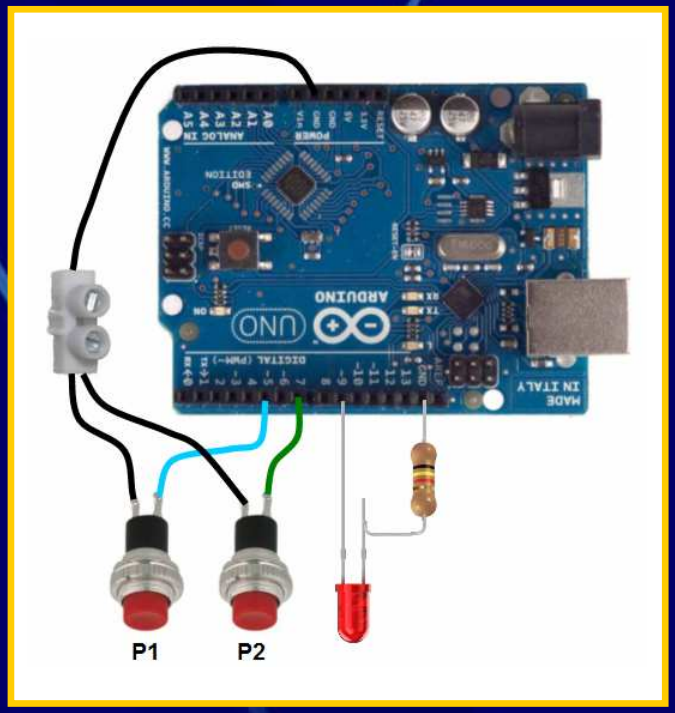


Recordar siempre conectar el conector GND del entrenador con un pin GND de la placa.

Uso del entrenador con Arduino

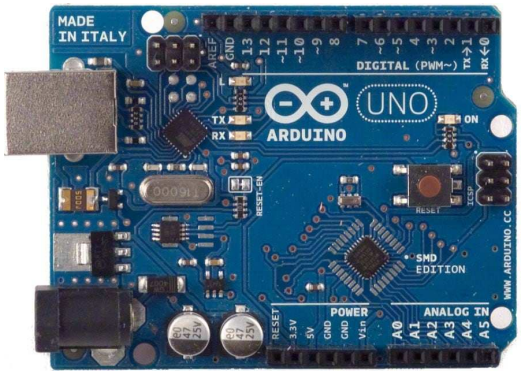
Realicemos el montaje de la figura con el entrenador.



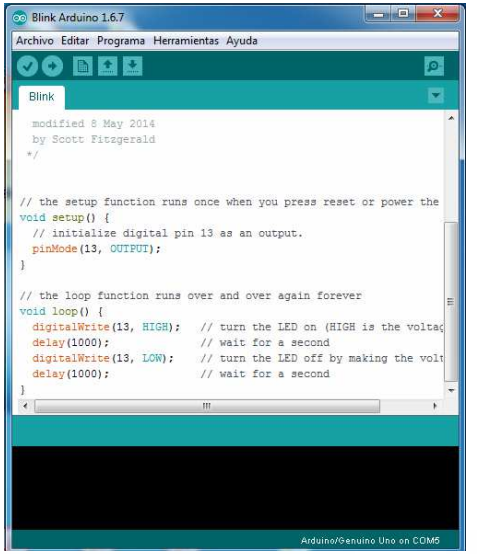


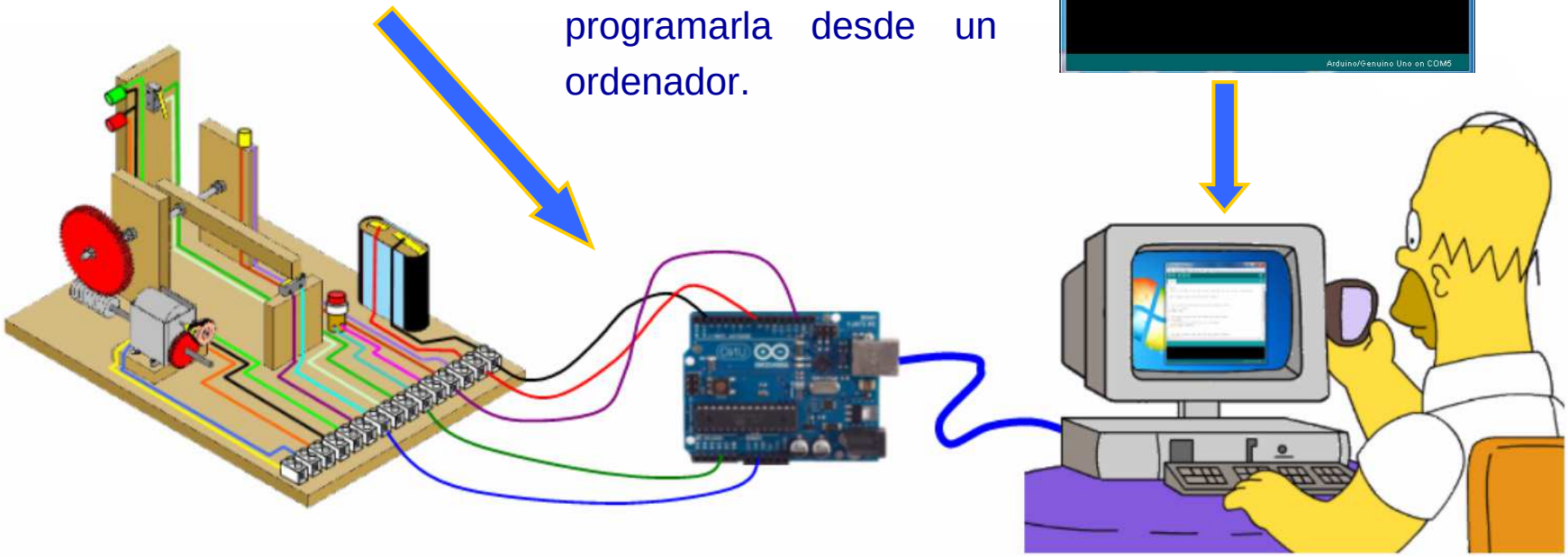
Recordar siempre conectar el conector GND del entrenador con un pin GND de la placa.

Control programado con ARDUINO

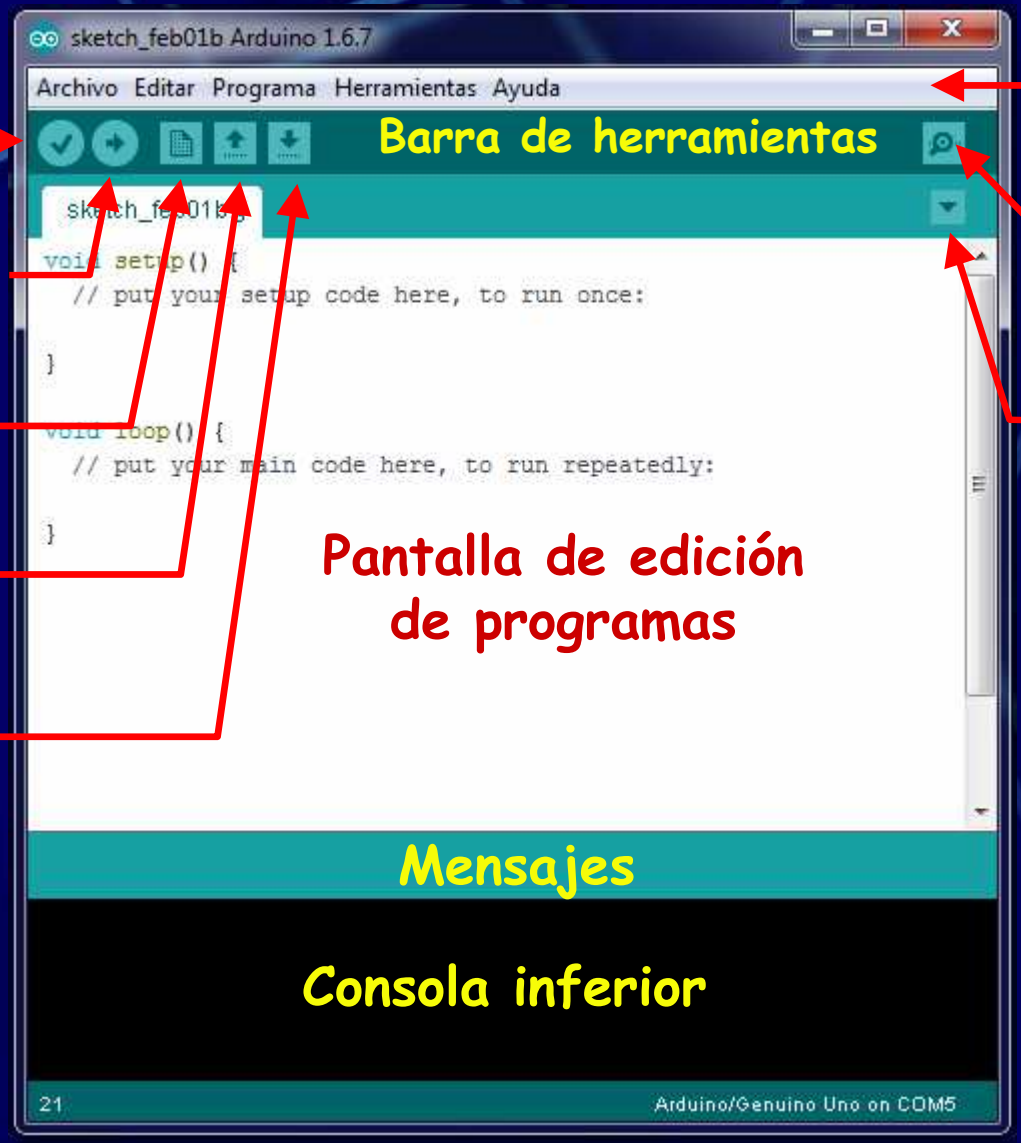


ARDUINO es una plataforma de **hardware libre** basada en una placa electrónica con un microcontrolador y un entorno de desarrollo integrado (IDE) para programarla desde un ordenador.

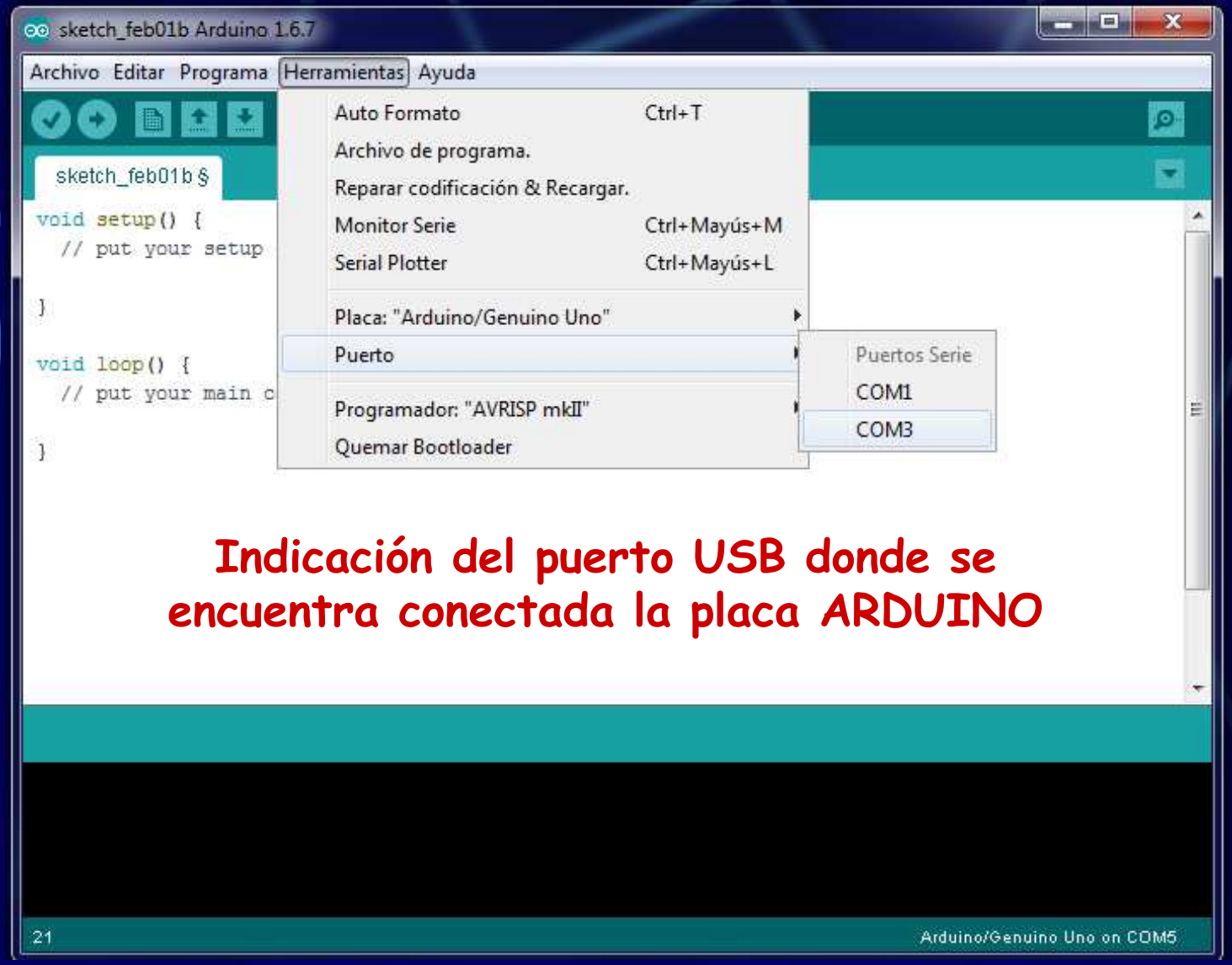




El entorno integrado de ARDUINO

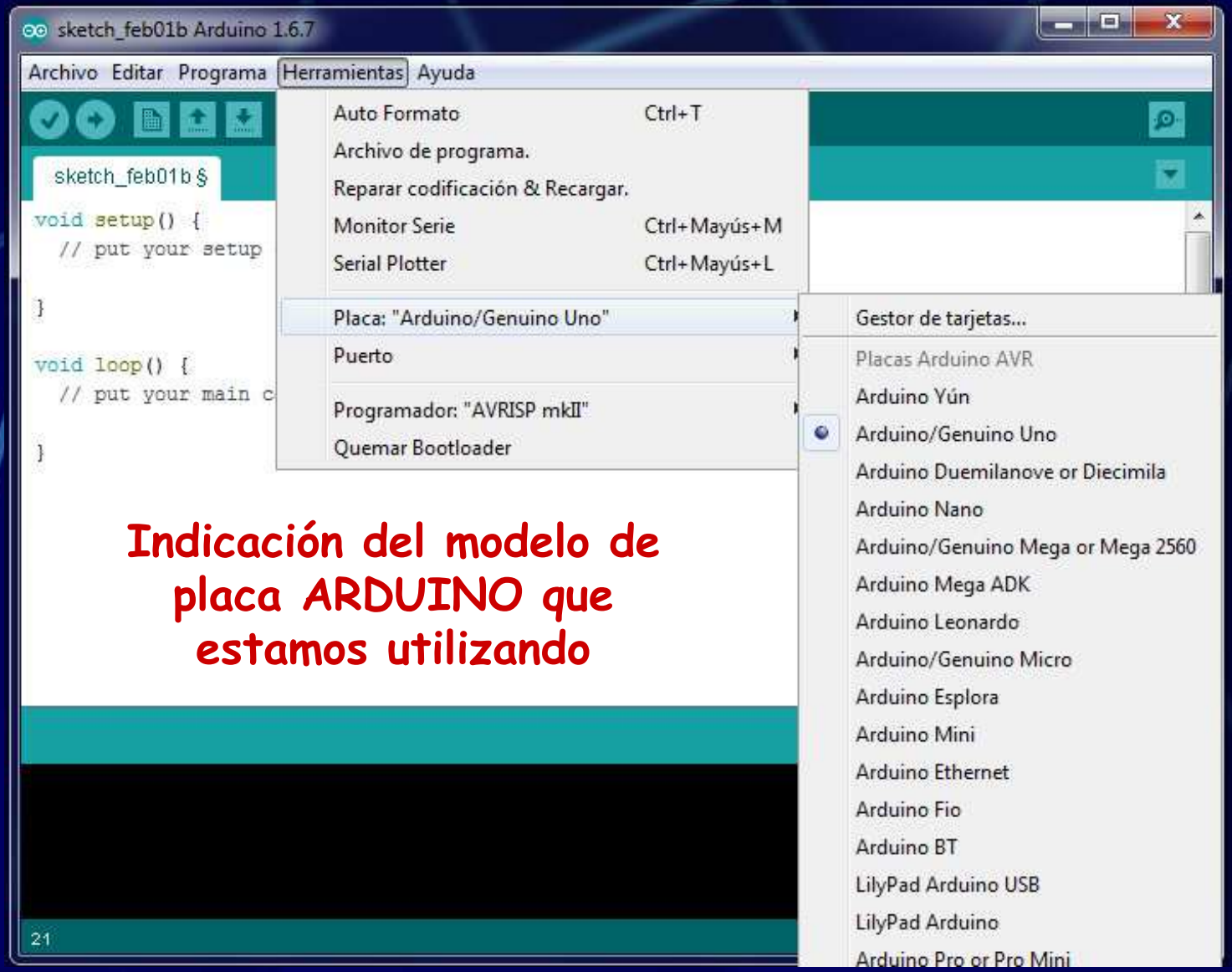


Indicación del puerto USB de la placa ARDUINO



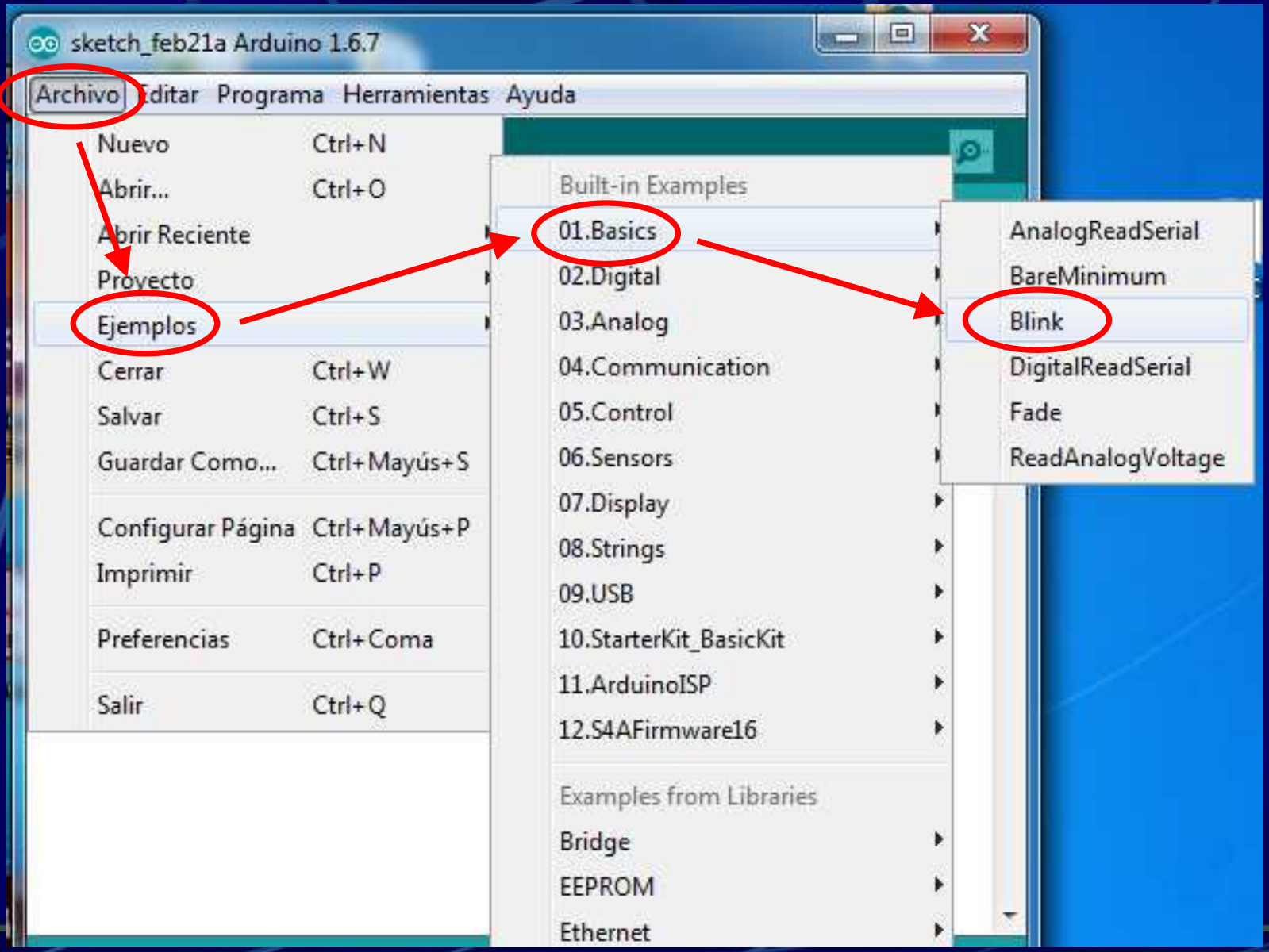
Indicación del puerto USB donde se encuentra conectada la placa ARDUINO

Indicación del modelo de la placa ARDUINO utilizada



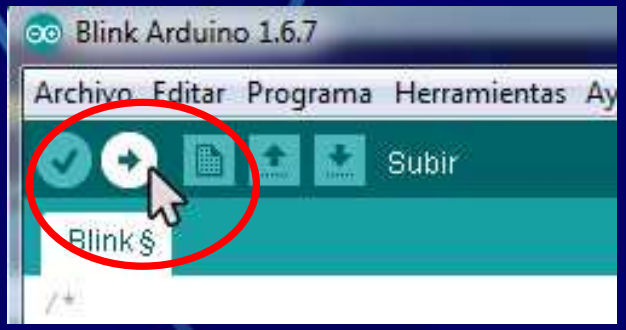
Indicación del modelo de placa ARDUINO que estamos utilizando

Cargar un programa de ejemplo para probar la placa: Blink



Cargar un programa de ejemplo para probar la placa: Blink

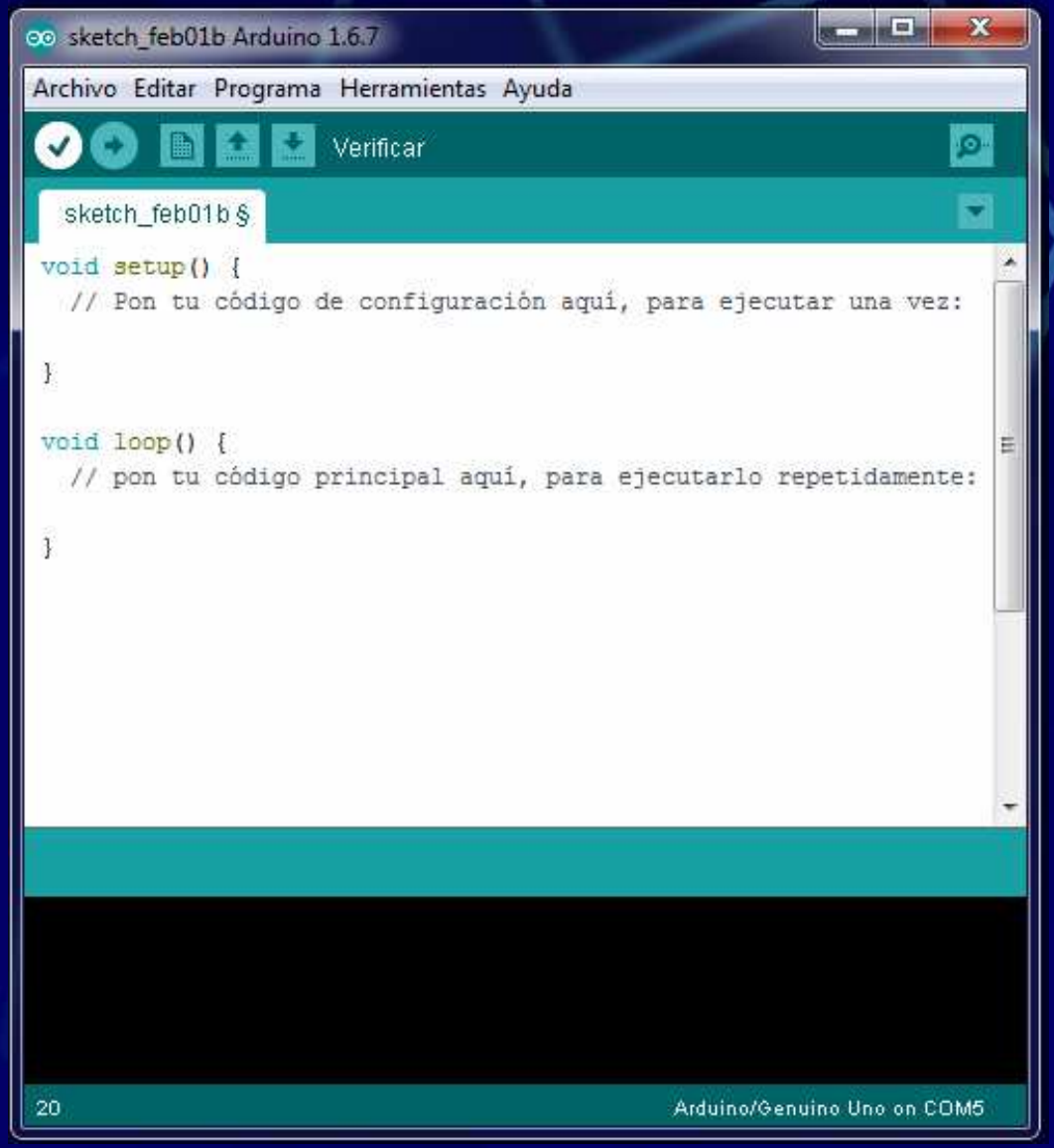
Una vez abierto el programa **Blink** lo subimos a la memoria de la placa Arduino haciendo clic sobre el icono **Subir**.



El programa Blink debe hacer parpadear un LED que lleva incorporado la placa conectado al pin 13 a intervalos de un segundo.

Comprobamos que lo hace.

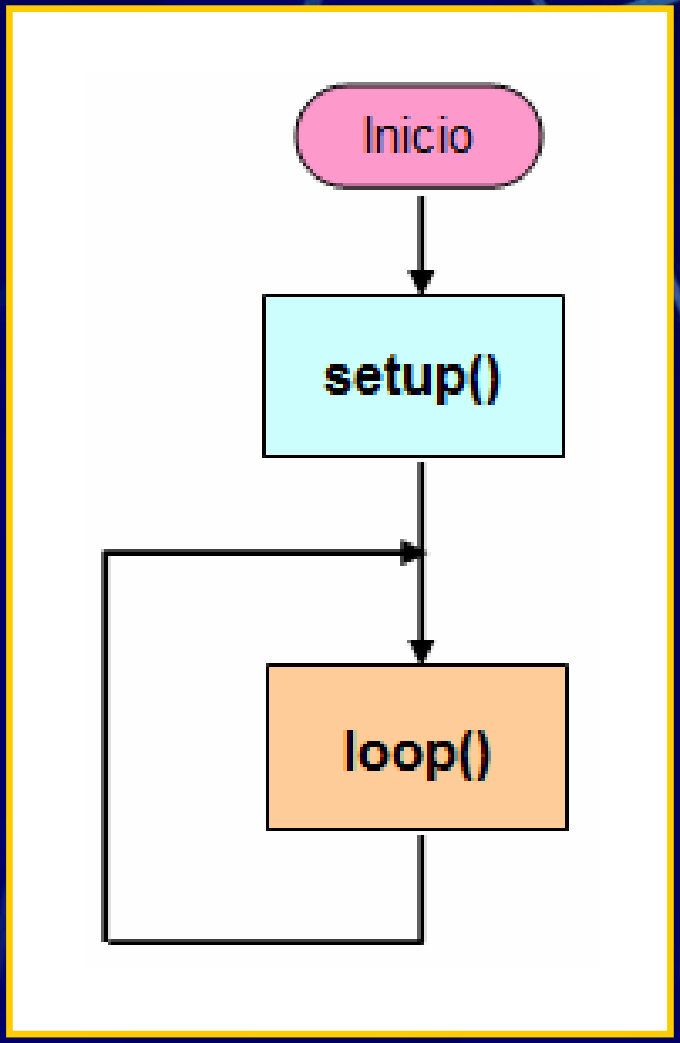
Programación en ARDUINO: Funciones setup() y loop()



Todos los programa deben contener como mínimo las funciones **setup** y **loop**, aunque éstas estén vacías.

- **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.

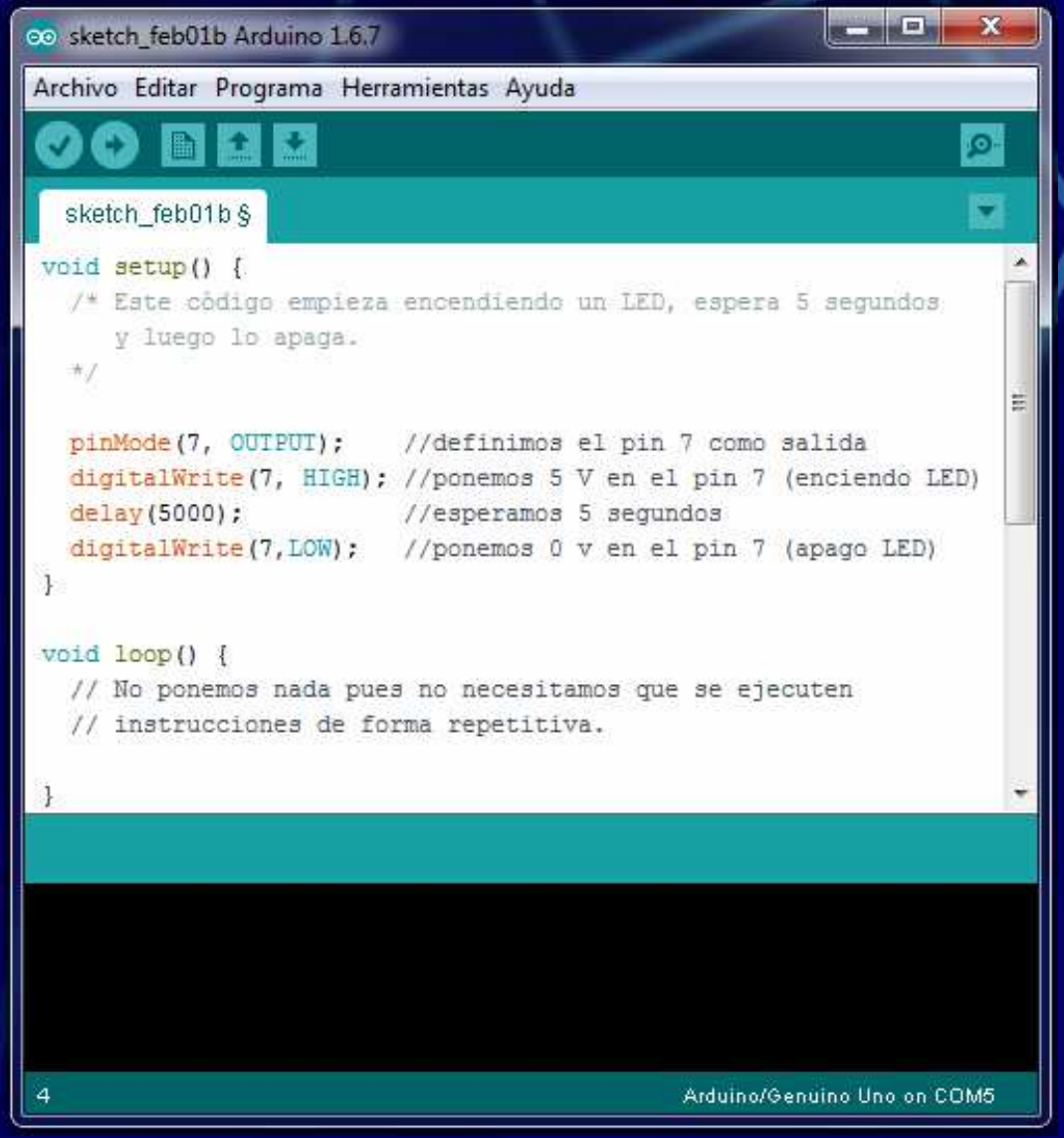
Esquema de funcionamiento de Arduino



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 la función **loop() se ejecuta de forma cíclica y permanente**. Una vez termina vuelve a empezar una y otra vez mientras que siga alimentada eléctricamente la placa o se presione su botón reset.

Programación en ARDUINO: pinMode()



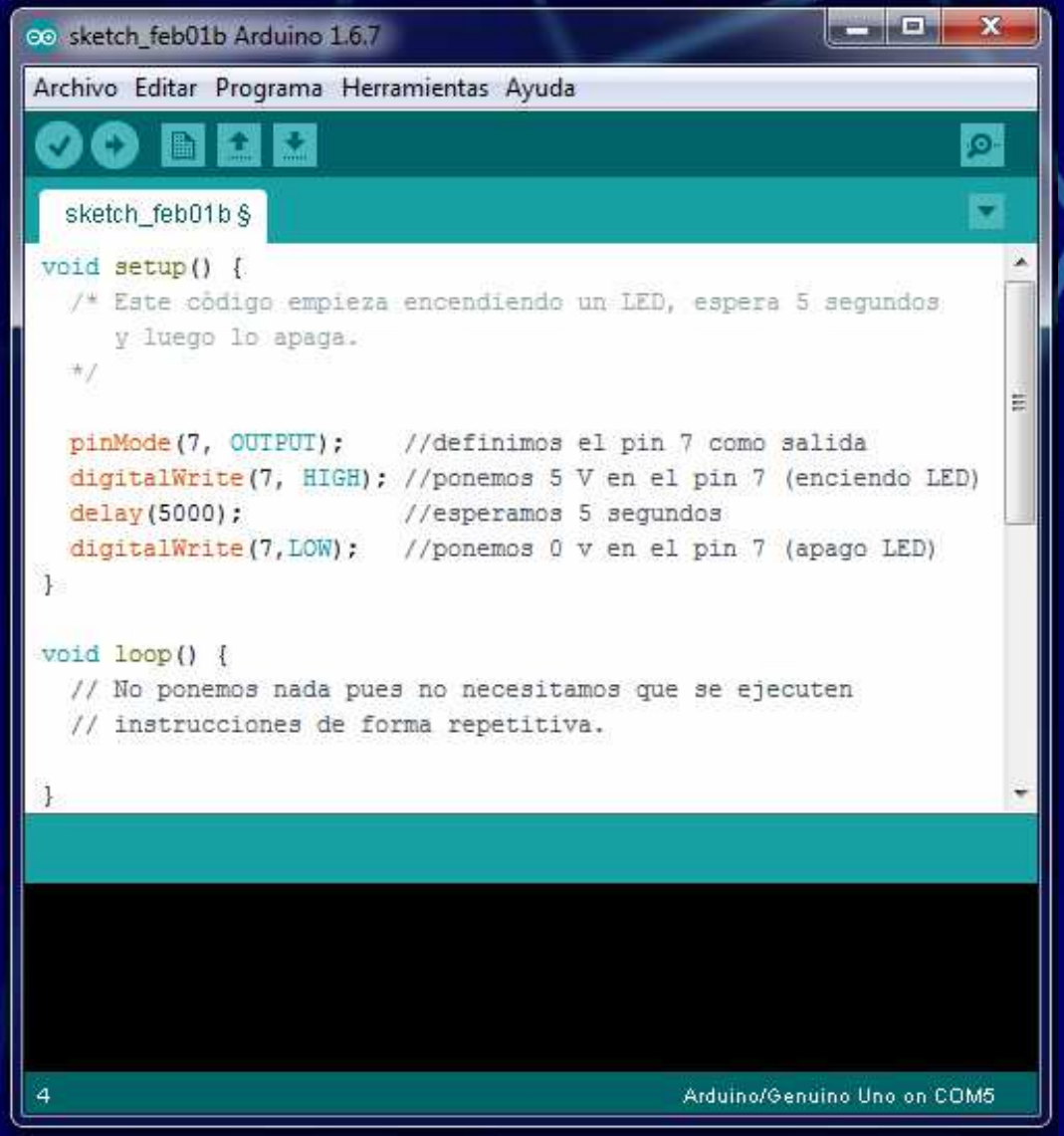
Ejemplo: programa que enciende un LED conectado en el pin 7 de Arduino, lo mantiene encendido 5 segundos y luego lo apaga.

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

pinMode (pin, modo)

Si al parámetro 'modo' le damos el valor **OUTPUT** definimos el pin como salida y si le damos el valor **INPUT_PULLUP** definimos el pin como entrada.

Programación en ARDUINO: digitalWrite()



Ejemplo: programa que enciende un LED conectado en el pin 7 de Arduino, lo mantiene encendido 5 segundos y luego lo apaga.

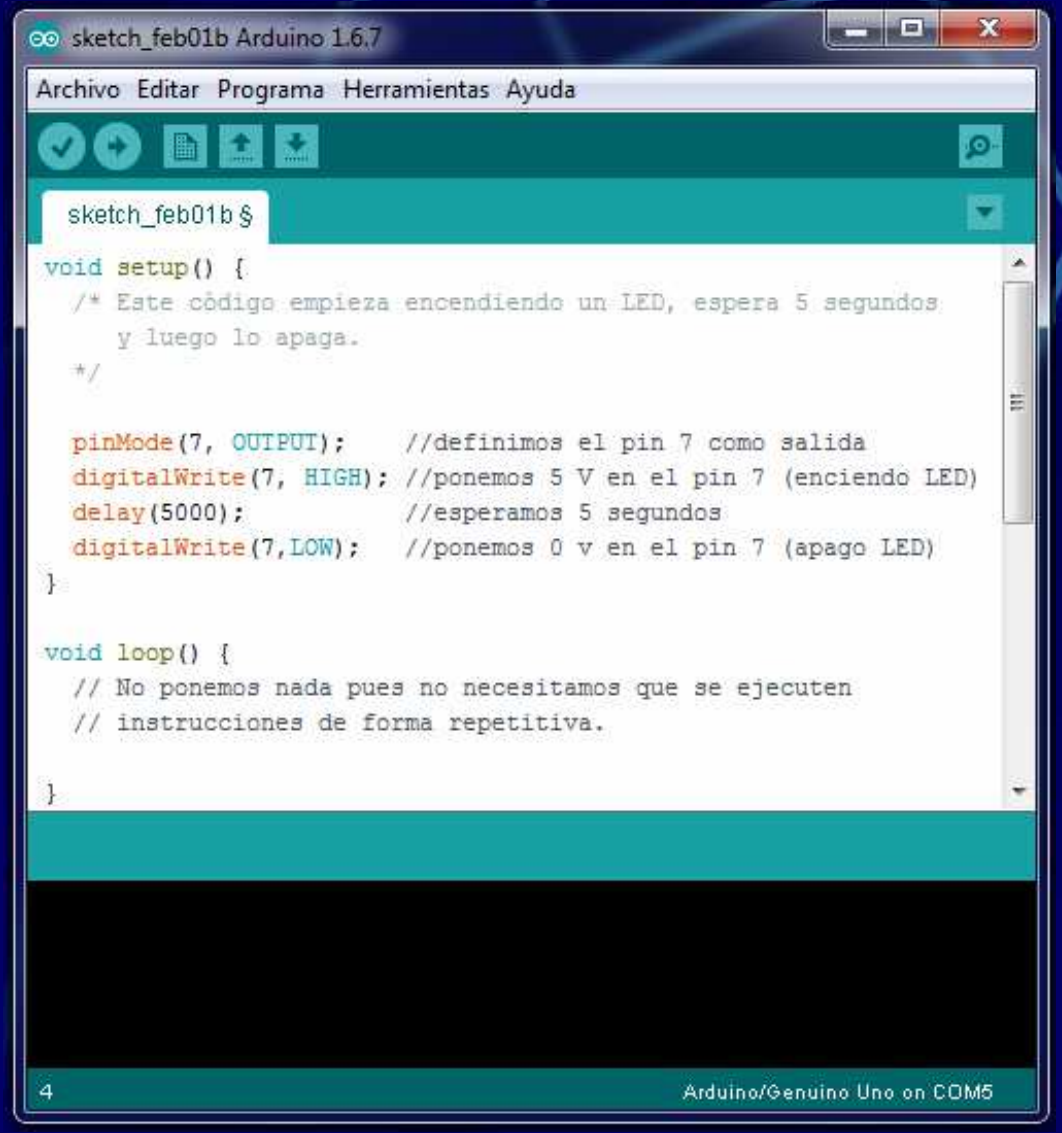
Se pone un valor de tensión en un pin definido como salida digital con la función:

digitalWrite (pin, valor)

El parámetro 'valor' puede valer **HIGH** o **LOW**.

Si al parámetro 'valor' es HIGH, ponemos en el pint una tensión de 5 V y si es LOW ponemos en el pint una tensión de 0 V.

Programación en ARDUINO: delay()



Ejemplo: programa que enciende un LED conectado en el pin 7 de Arduino, lo mantiene encendido 5 segundos y luego lo apaga.

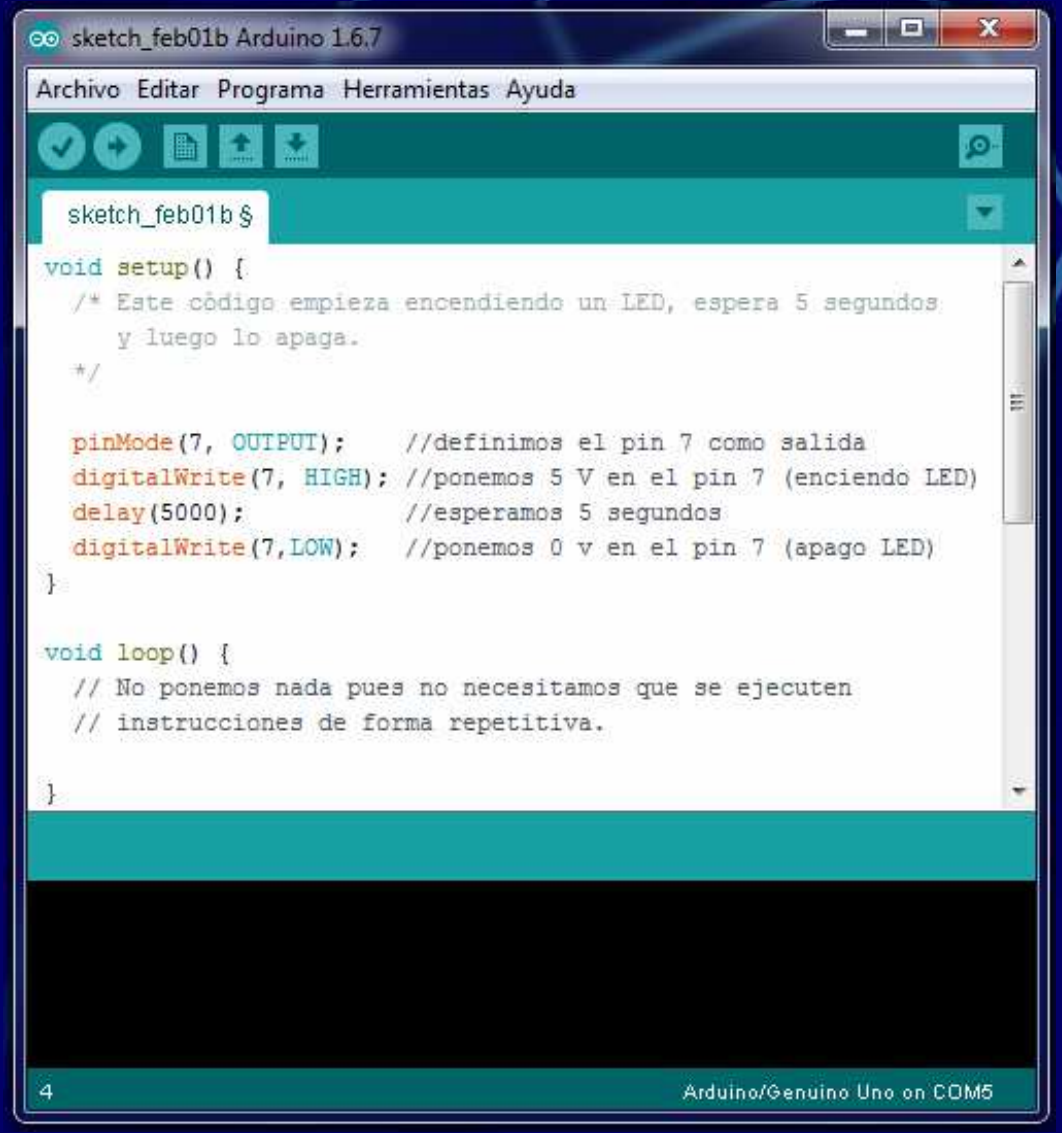
Se puede realizar fácilmente una temporización con la función delay():

delay (valor)

El parámetro 'valor' será un número que indica el tiempo de espera medido en milisegundos.

Esta función detiene la ejecución del programa durante el tiempo indicado.

Programación en ARDUINO: Comentarios



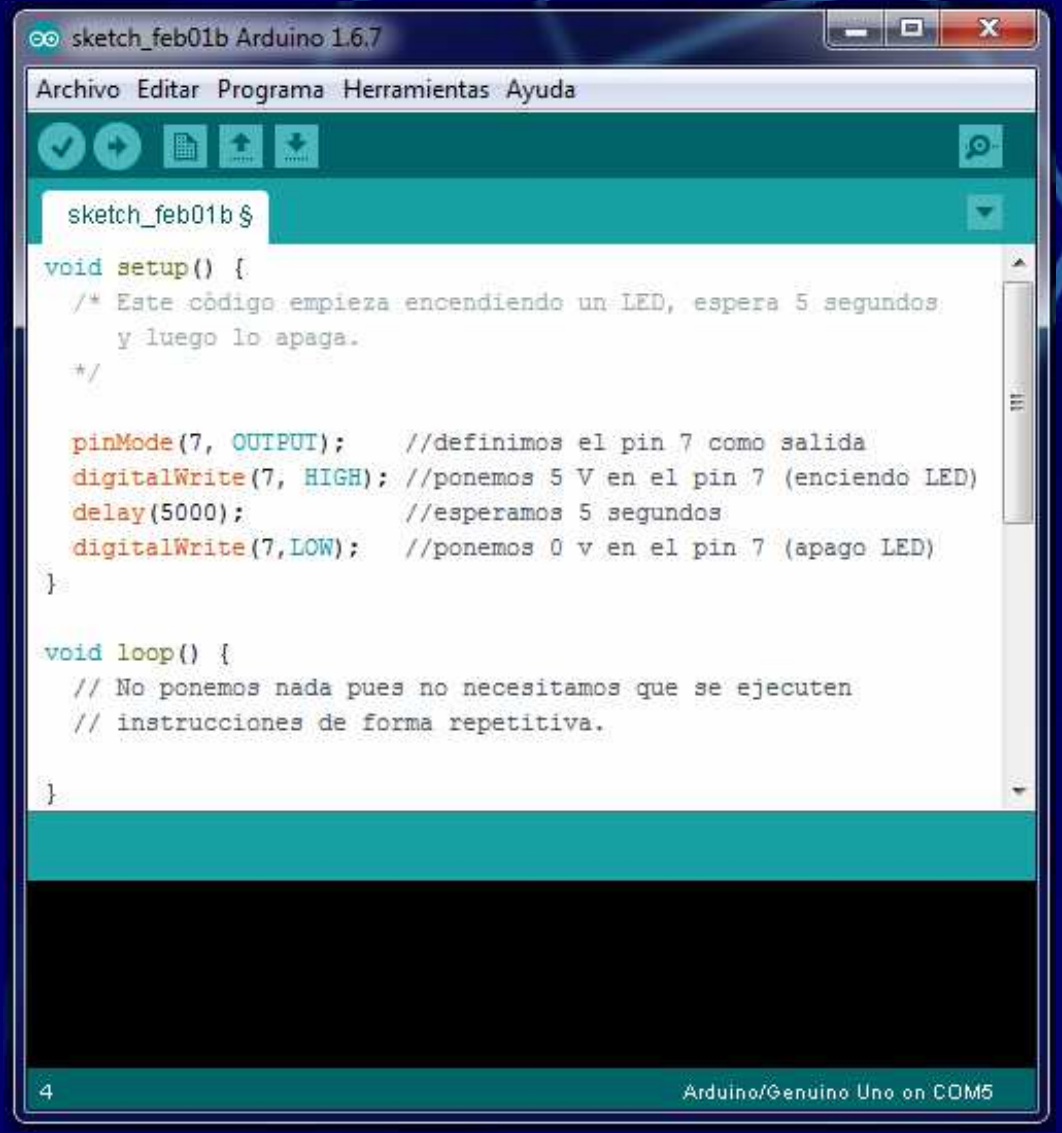
Ejemplo: programa que enciende un LED conectado en el pin 7 de Arduino, lo mantiene encendido 5 segundos y luego lo apaga.

Podemos colocar comentarios en nuestros programas para explicar el código, hacer anotaciones, …. No forman parte del código.

- Serán comentarios todas las líneas incluidas entre ***/**** y ****/***.
- Será comentario todo lo que siga a ***//*** hasta el final de línea.

Los comentarios son escritos en gris automáticamente.

Programación en ARDUINO: Elementos de sintaxis



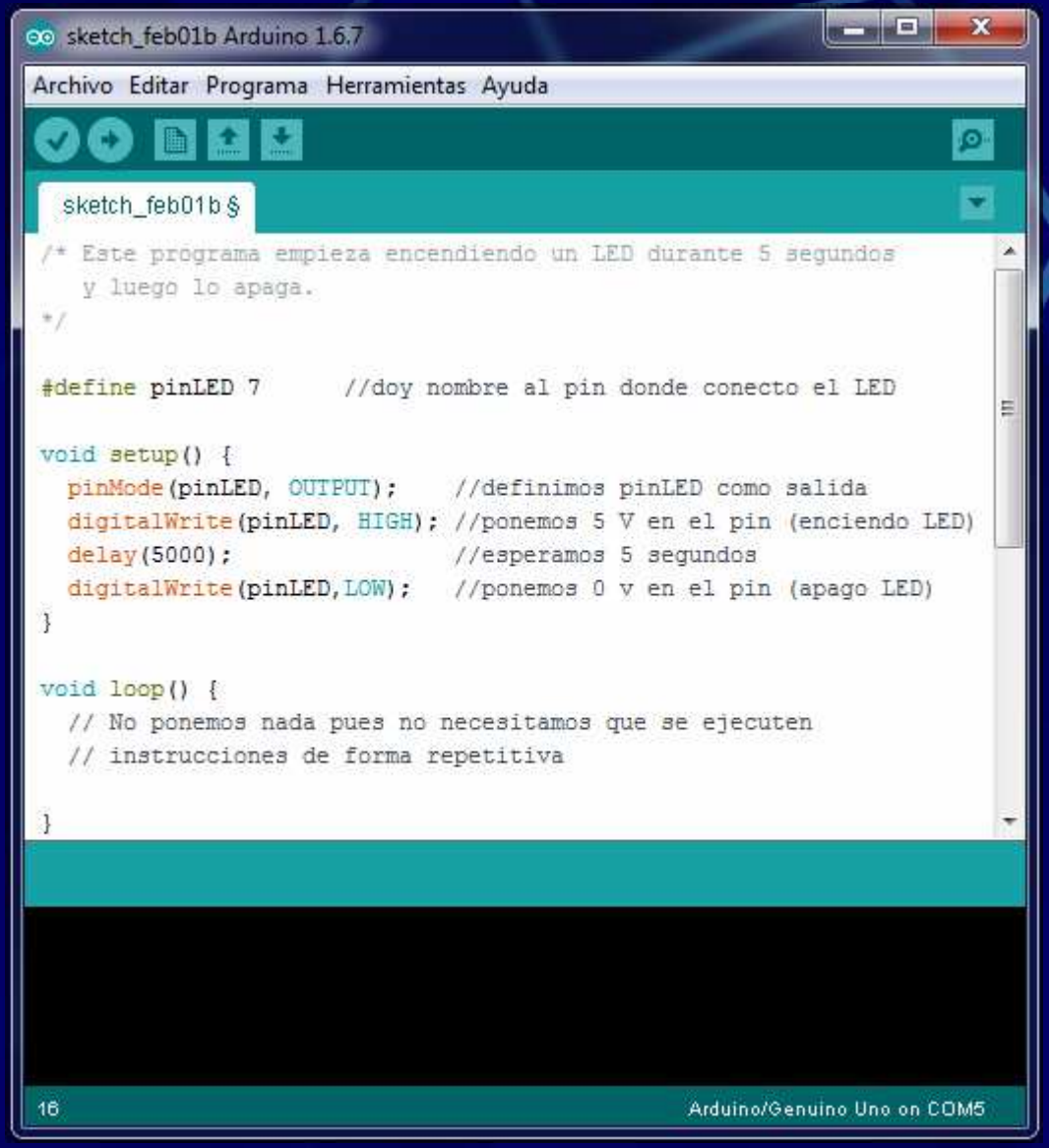
Los principales elementos de sintaxis son el punto y coma (;) y las llaves ({ }).

Toda instrucción debe ir seguida de un "punto y coma" (;). Podrían ir varias instrucciones en un mismo renglón, siempre que vayan separadas por ";".

Las llaves ({ }) se usan para delimitar el inicio y el fin de diversas construcciones:

- Funciones.
- Bucles de repetición.
- Instrucciones condicionales.

Programación en ARDUINO: Constantes



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 y hay menos errores.

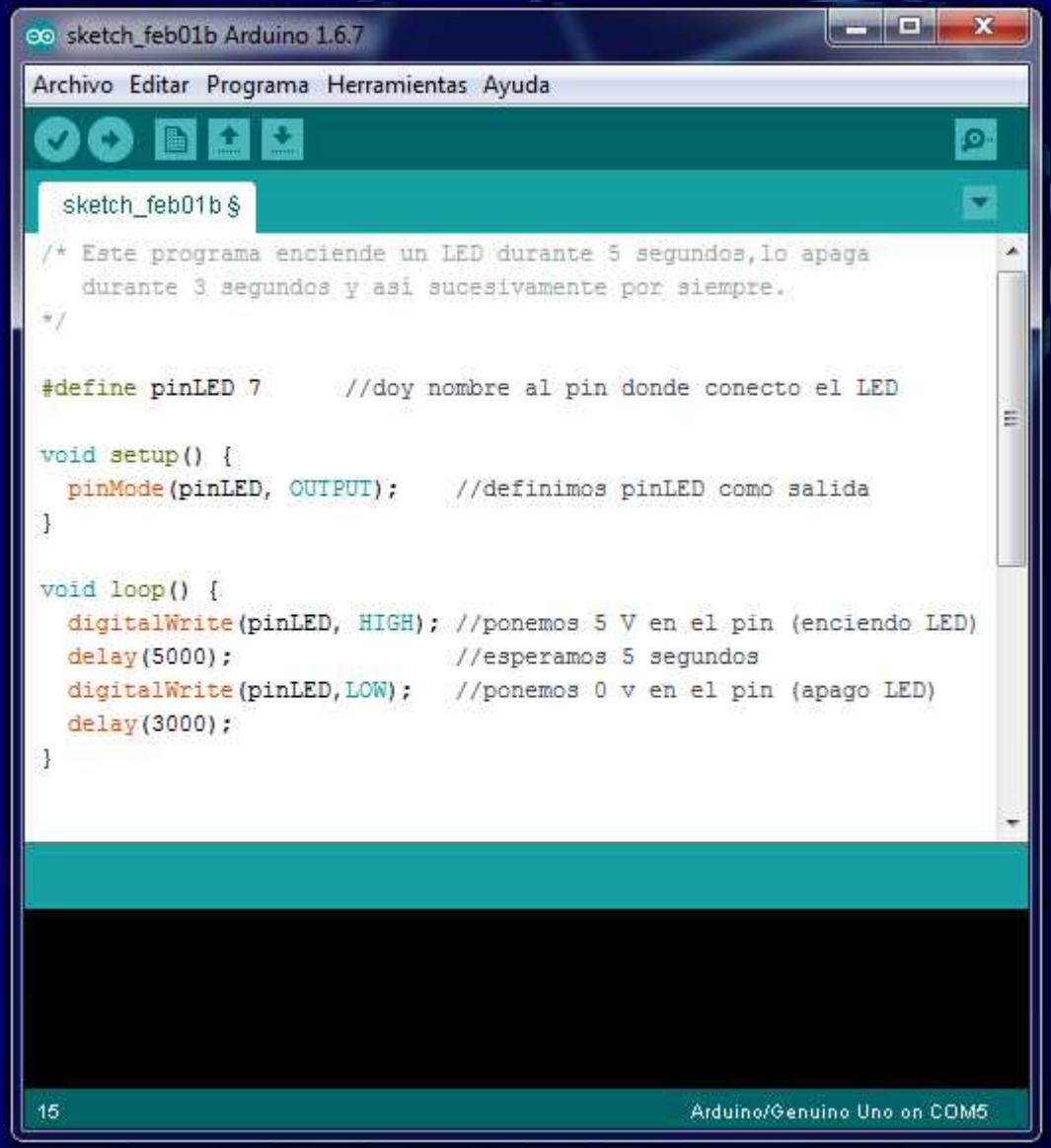
Estos nombres se llaman **constantes**, y se definen utilizando la instrucción:

#define const número

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

Todas las constantes deben declararse antes de usarse.

Programación en ARDUINO: la función loop()

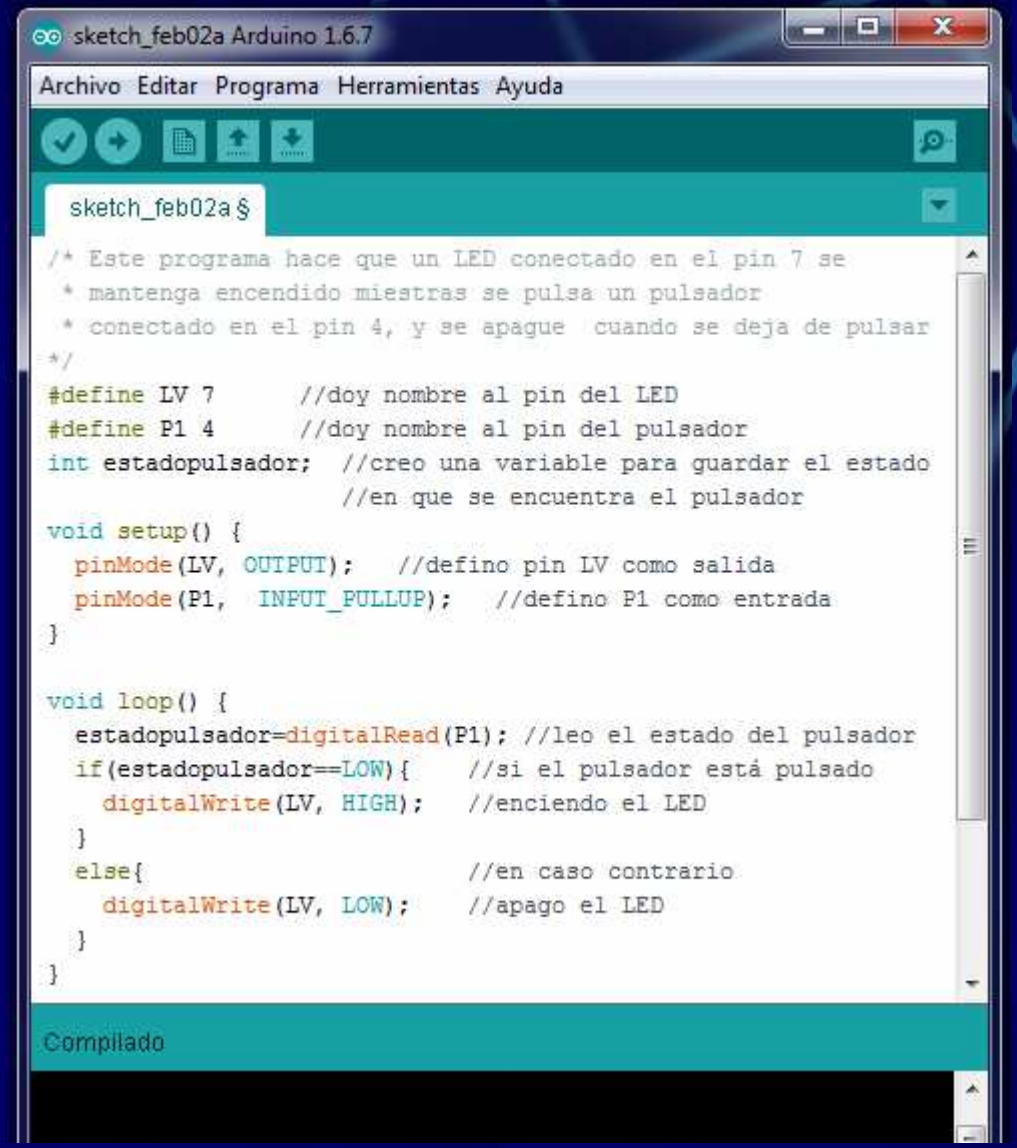


Ejemplo: programa que enciende un LED conectado en el pin 7 de Arduino durante 5 segundos, luego lo apaga 3 segundos y así sucesivamente por siempre.

Lo normal es que un sistema esté controlado de forma permanente. Mientras que la **función setup()** se ejecuta sólo una vez, la **función loop()** se ejecuta cíclicamente: cuando acaba, vuelve a ejecutarse y así de forma indefinida.

Ahora el LED parpadeará de forma indefinida.

Programación en ARDUINO: las variables



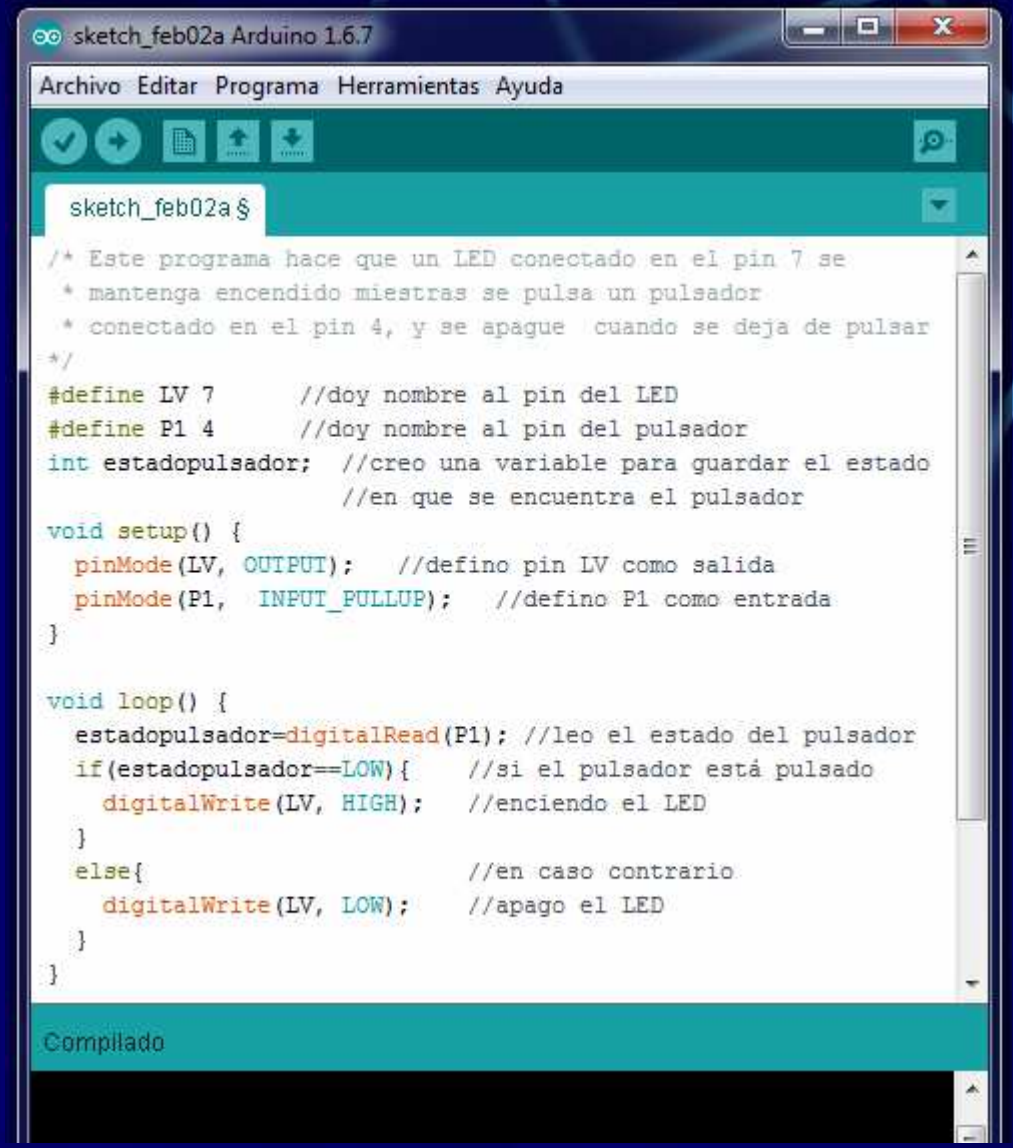
Ejemplo: programa que enciende un LED mientras esté pulsado un pulsador y que lo apaga cuando dicho pulsador no está pulsado.

Una **variable** es un espacio reservado en la memoria de Arduino con un nombre. En ese espacio se guarda su valor, que es un dato que puede variar a lo largo de la ejecución del programa.

Toda variable debe ser declarada (tipo y nombre) antes de ser usada. Si se hace delante de la función setup() es una **variable global**.

tipo nombre_variable;

Programación en ARDUINO: Los tipos de datos



Para declarar una variable hay que indicar el **tipo** de datos que va a guardar (de lo cual depende el tamaño del espacio de memoria reservado) y el nombre.

tipo nombre_variable;

Los tipos de datos básicos son:

- **void**: sólo para funciones que no devuelven nada.
- **int**: valores enteros cortos.
- **long**: valores enteros largos.
- **unsigned long**: valores enteros largos sin signo.
- **float**: valores decimales.

Usaremos sobre todo el tipo int

Programación en ARDUINO: Los nombres válidos

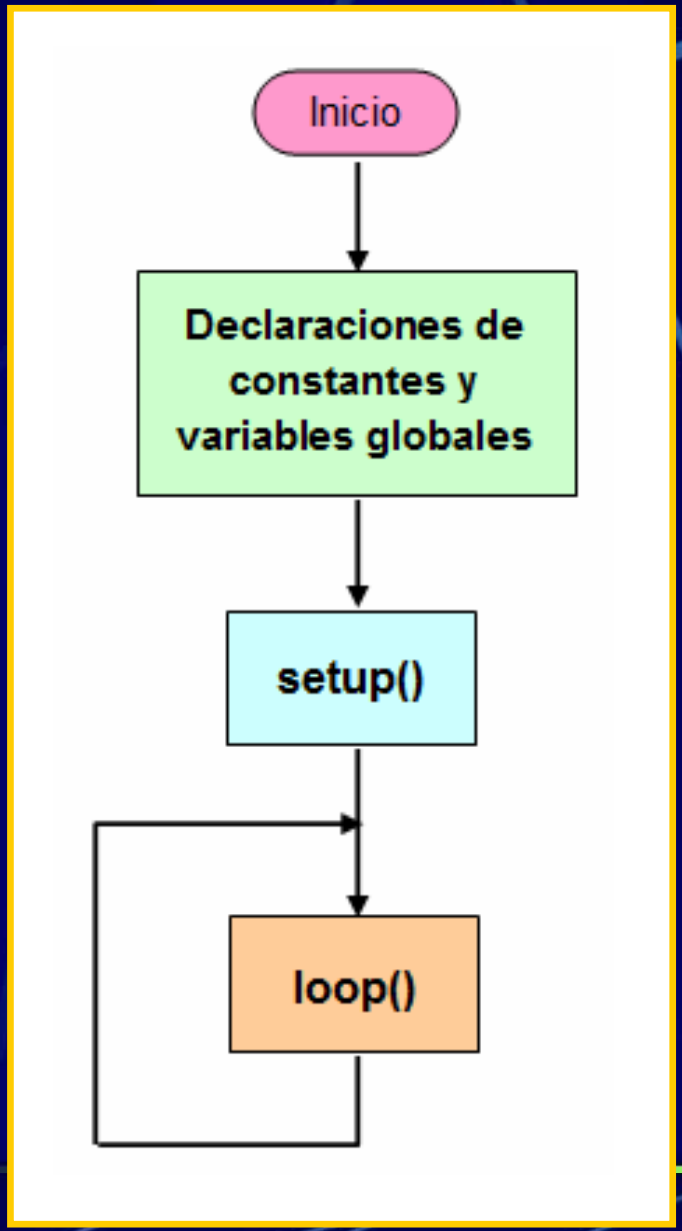
- Los nombres de los programas, de las variables, de las constantes y de las funciones no pueden contener espacios, ni signos matemáticos, ni tildes , ni signos de puntuación, ni la letra ñ. Sí puede usarse el guión bajo.

- Los nombres dados a variables, constantes y funciones propias no deben coincidir con palabras clave de Arduino (nombres de constantes, tipos, instrucciones y funciones del propio lenguaje Arduino (HIGH, PI, long, float, if, while, Serial, delay, max, min,…)

- Se distinguen mayúsculas de minúsculas. De este modo, la variable “contador” y “Contador” serían distintas.

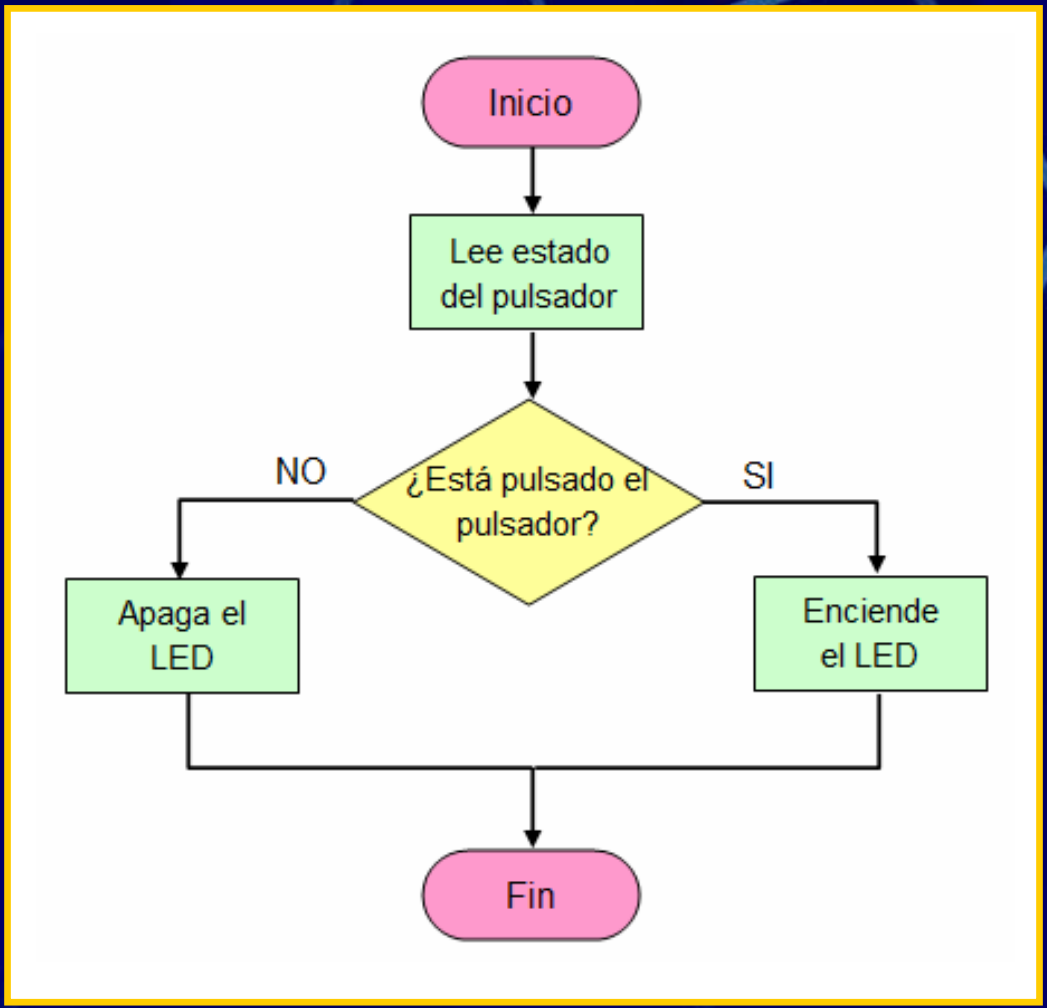
- Hay una serie de constantes con nombres reservados:
  - OUTPUT, INPUT e INPUT_PULLUP.
  - HIGH y LOW.
  - false y true.
  - PI.

Esquema de funcionamiento de Arduino



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

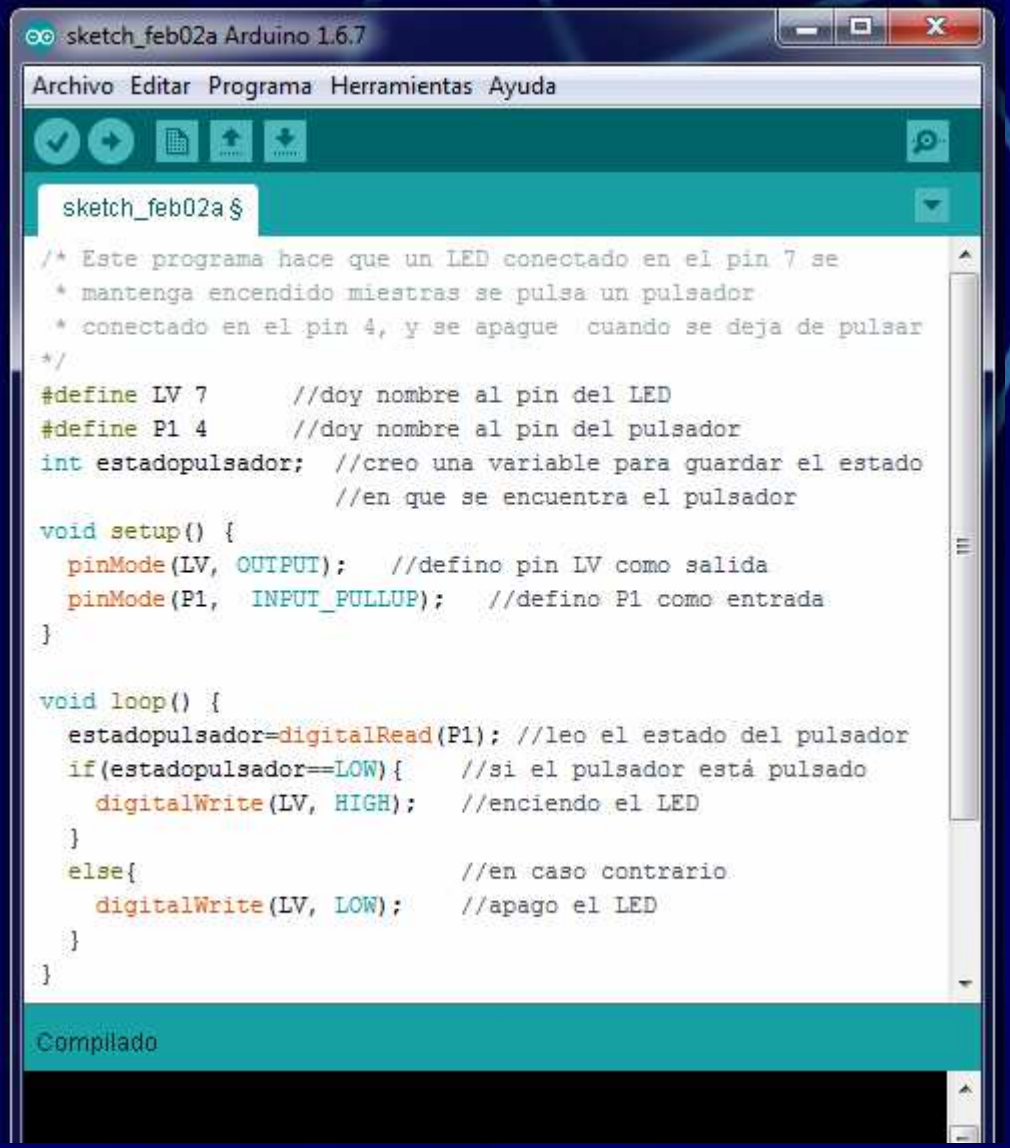
captación de datos externos y toma de decisiones



Normalmente, en el control de sistemas técnicos tendremos que "leer" el estado en que se encuentran elementos sensores (pulsadores, finales de carrera, interruptores, sensores de luz, de temperatura, de distancia, etc.) y tomar decisiones sobre lo que hay que hacer.

Ejemplo: programa que enciende un LED mientras esté pulsado un pulsador y que lo apaga cuando dicho pulsador no está pulsado.

Programación en ARDUINO: digitalRead()



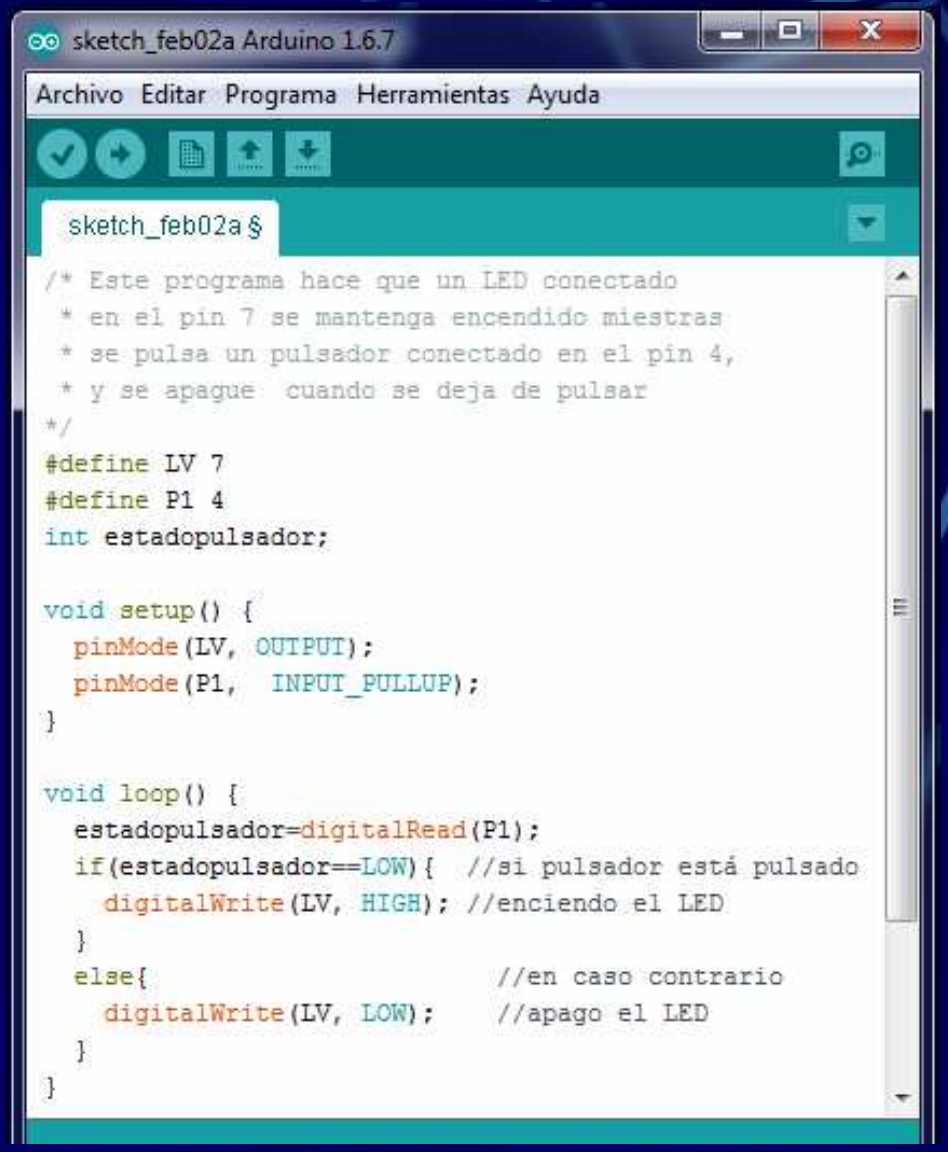
Ejemplo: programa que enciende un LED mientras esté pulsado un pulsador y que lo apaga cuando dicho pulsador no está pulsado.

Se lee el valor de tensión en un pin definido como entrada digital con la función:

digitalRead (pin)

Esta función devuelve un valor **HIGH** (si en dicho pin se mide una tensión de 5 V o cercana) o un valor **LOW** (si la tensión medida en el pin es de 0 V o cercana.

Programación en ARDUINO: Estructura condicional if...else



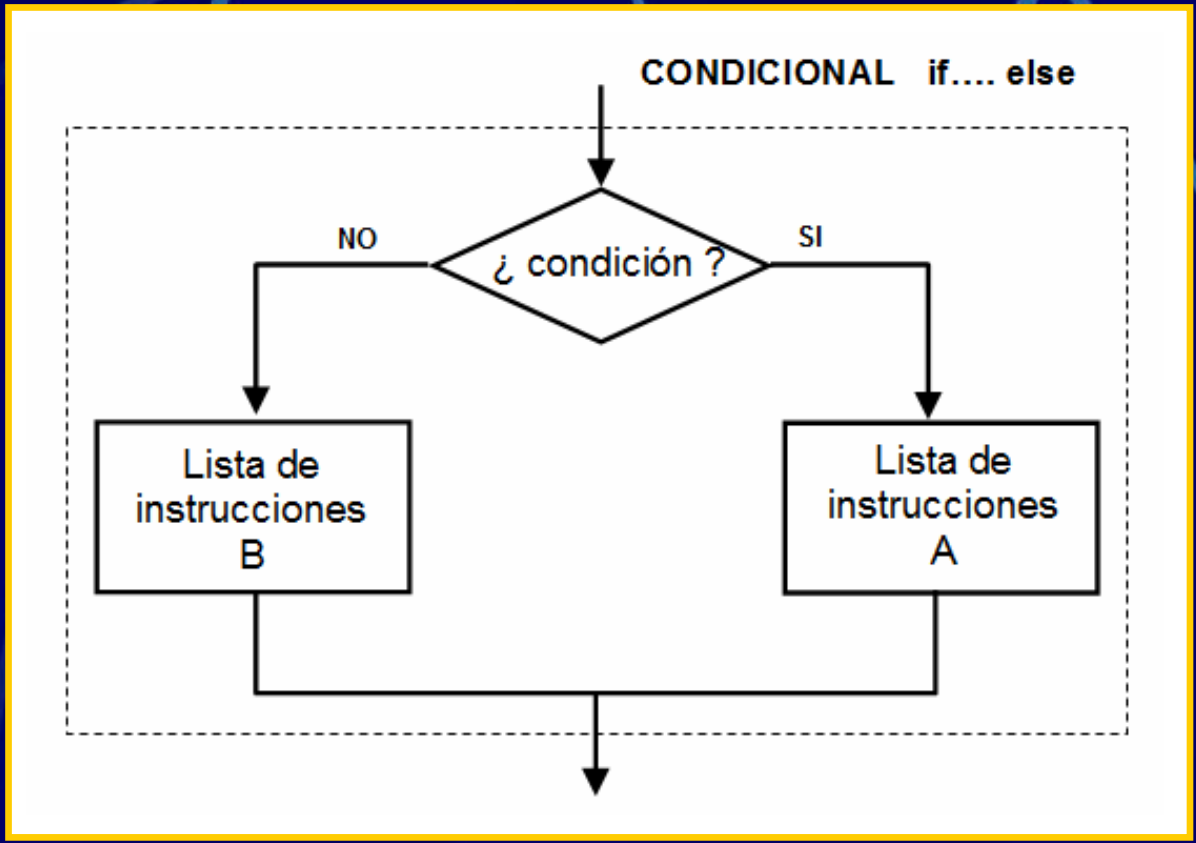
La estructura **if...else** decide ejecutar unas instrucciones u otras en función de que una condición se evalúe como verdadera (true) o falsa (false).

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

Cuando sólo tiene que ejecutarse una instrucción no son necesarias las llaves { }.

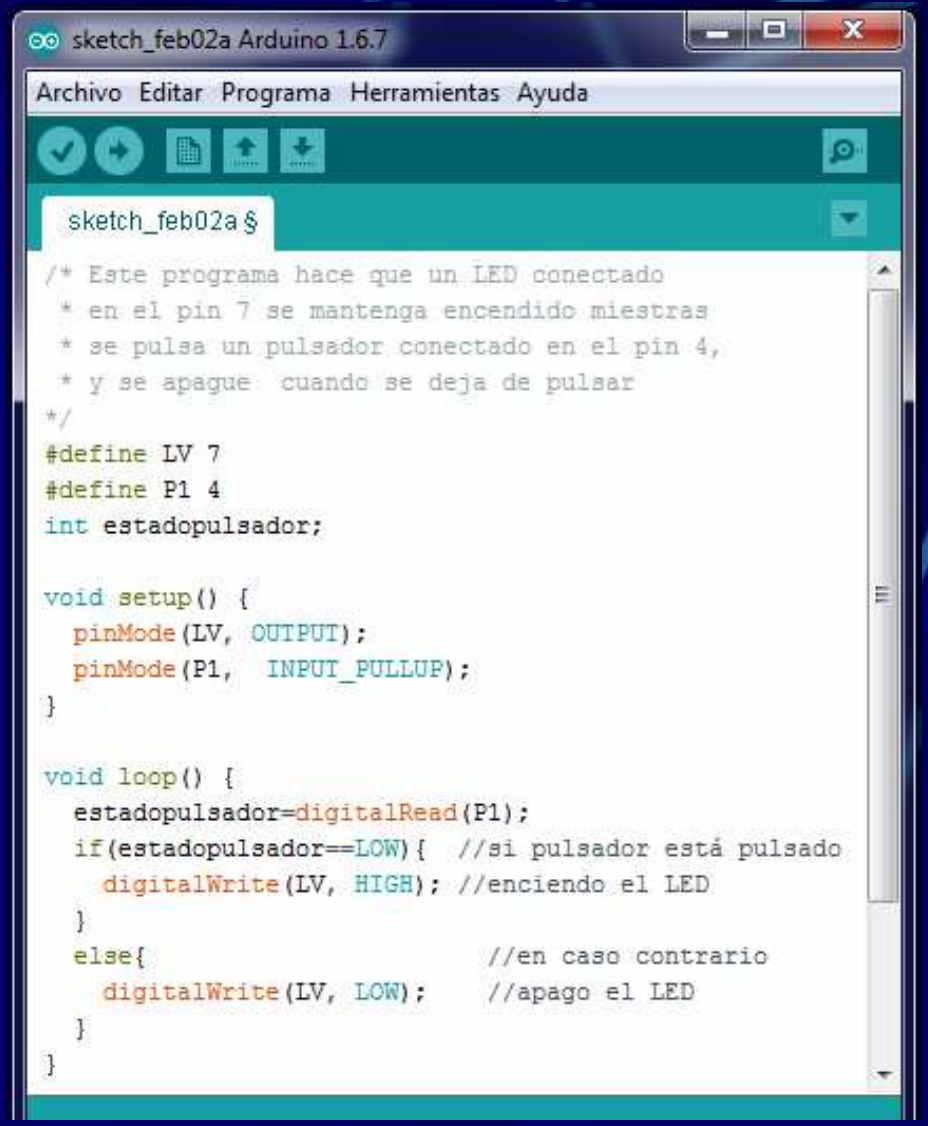
```
if (condición) instrucción_A;
else instrucción_B;
```

Diagrama de flujo de la estructura condicional if….else




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

Programación en ARDUINO: Estructura condicional if...else



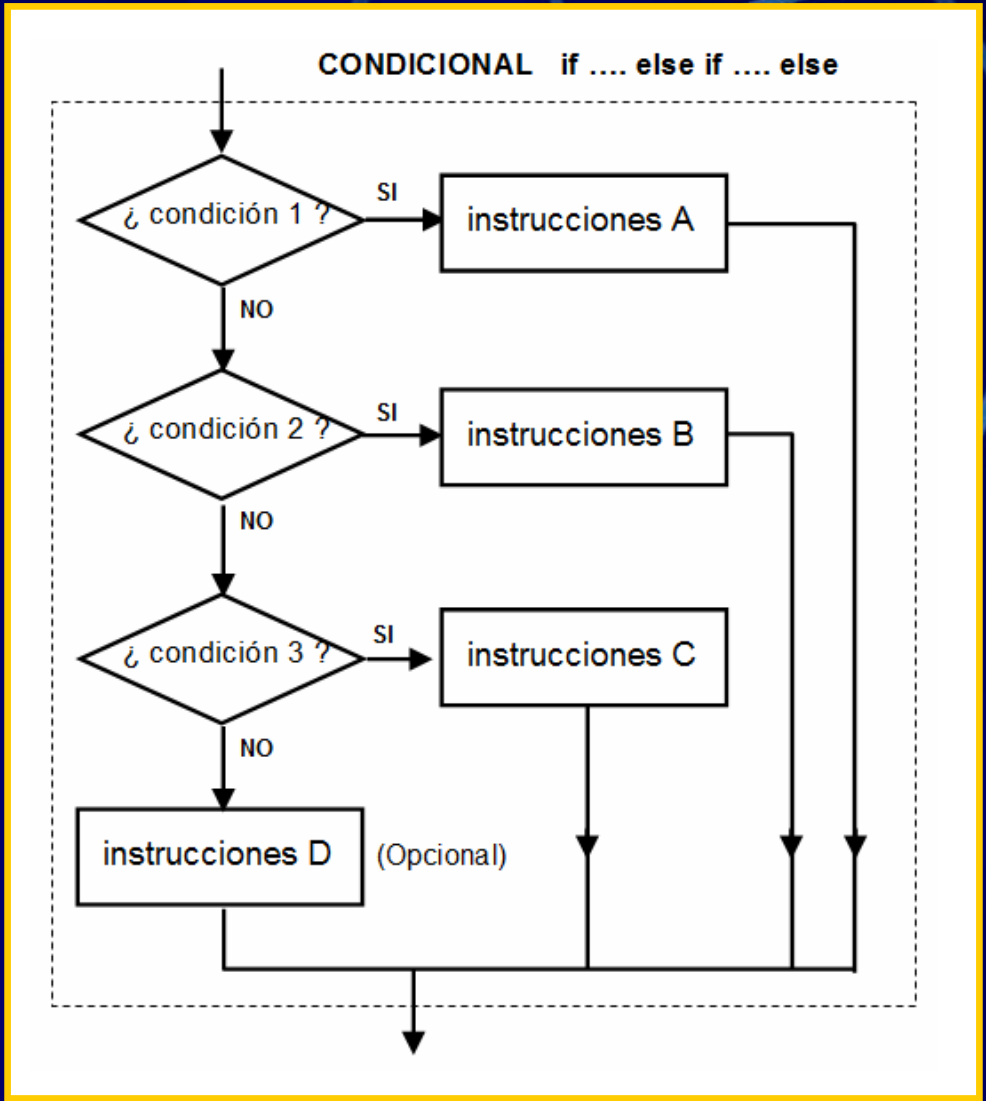
A **else** le pueden seguir otros if, ejecutándose múltiples pruebas.

if (condición1) {instrucciones_A;}
else if (condición2) {instrucciones_B;}
else if (condición3) {instrucciones_C;}
else {instrucciones_D;}

En cuanto se cumple una de las condiciones de los " if " se ejecutan las instrucciones correspondientes y ya no se evalúan el resto de " if ".

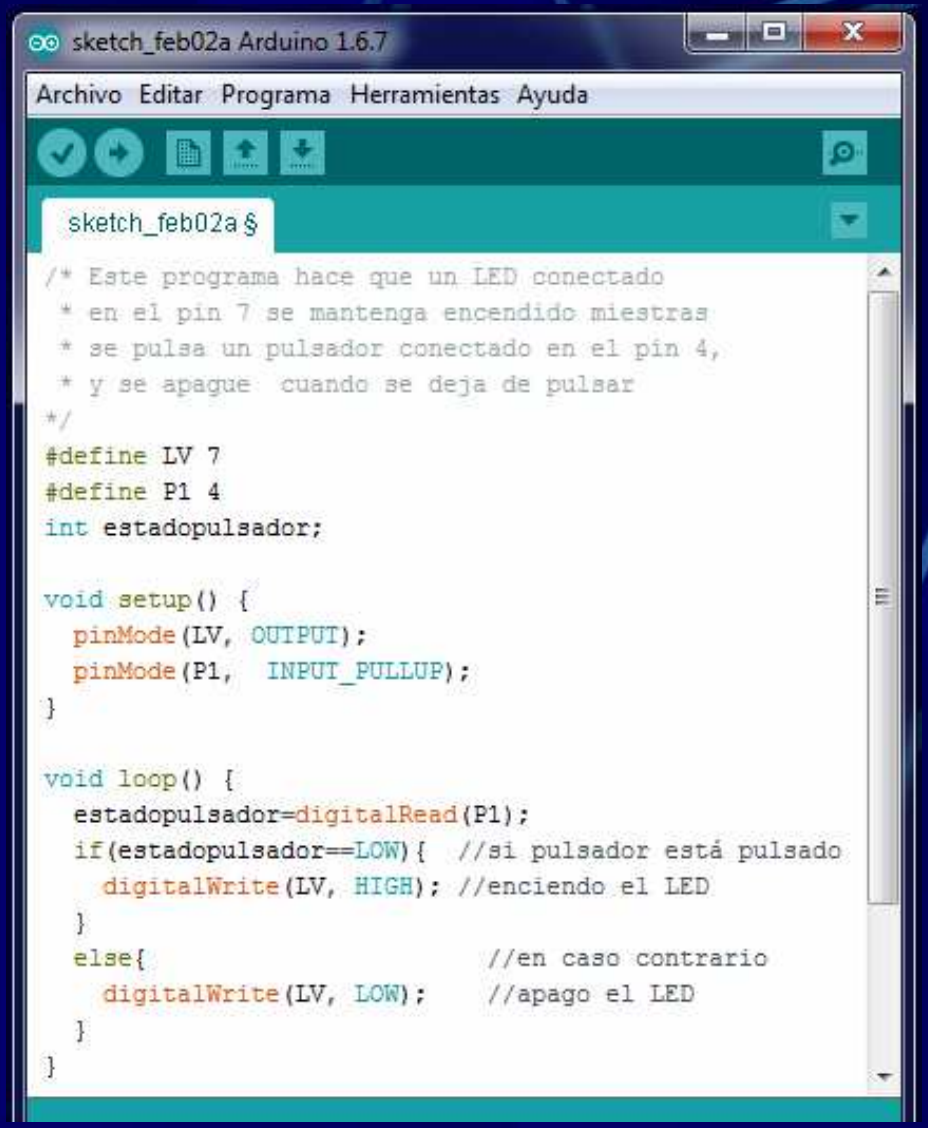
El else final es opcional, es decir, podemos querer que no se ejecute nada si no se cumple ninguna de las condiciones.

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




```
if (condición1) {instrucciones_A;}
else if (condición2) {instrucciones_B;}
else if (condición3) {instrucciones_C;}
else {instrucciones_D;}
```

Programación en ARDUINO: Estructura condicional if

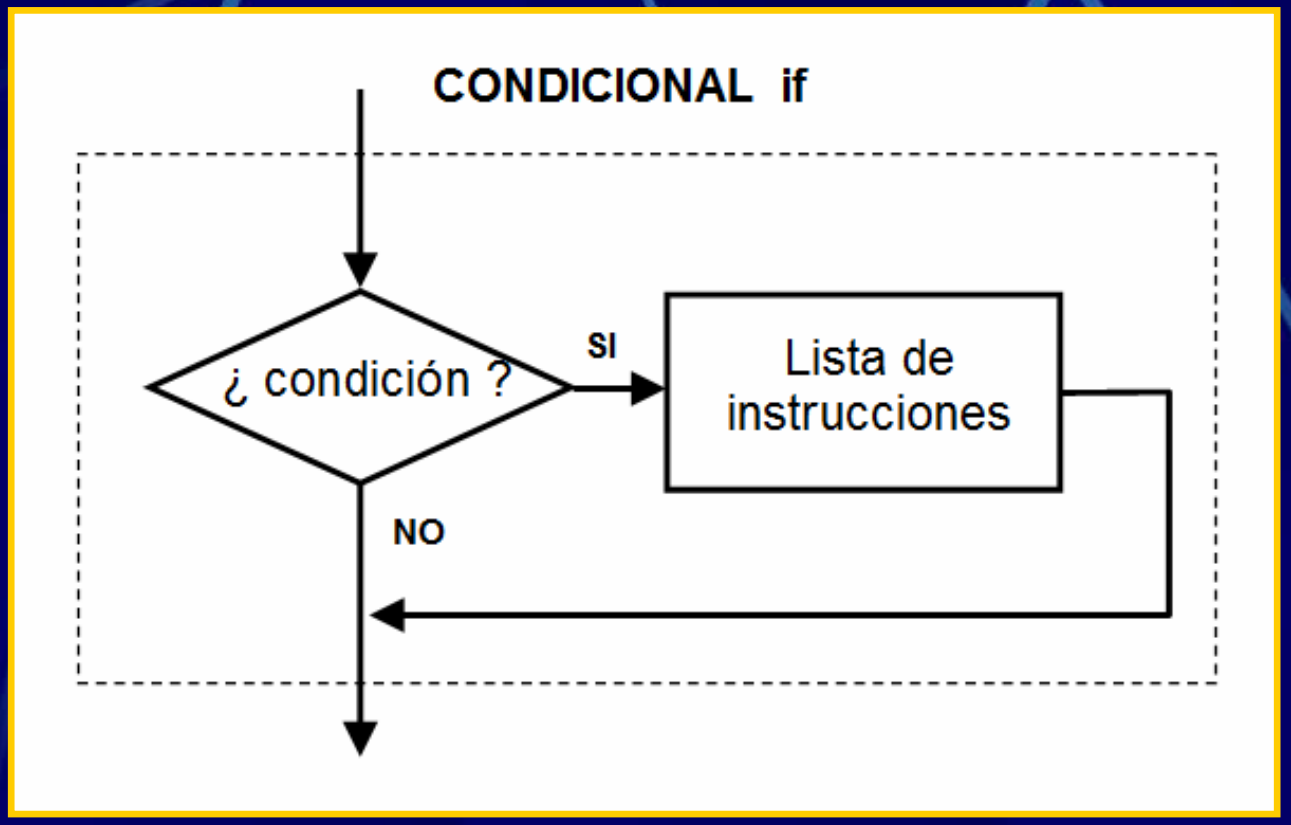


También puede ocurrir que no haya bloque else, es decir, en caso de no cumplirse la condición del if, no se tiene que ejecutar ninguna instrucción:

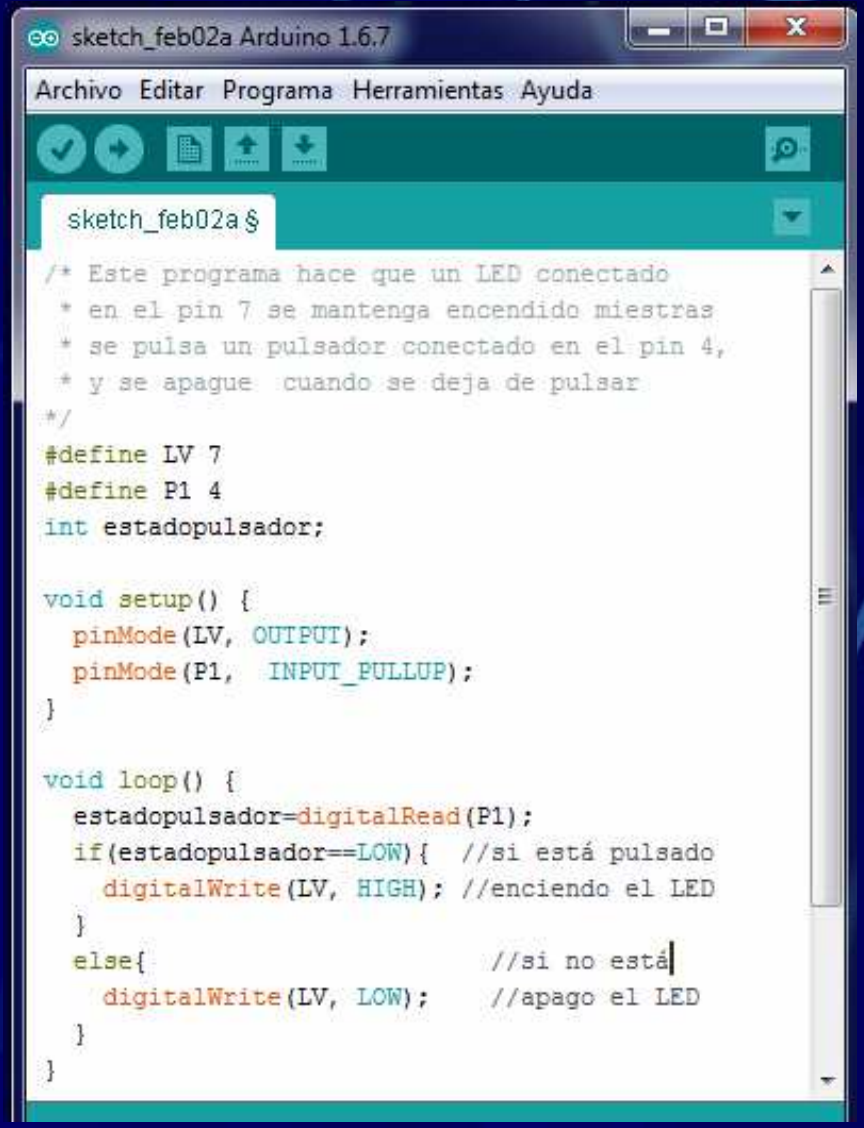
```
if (condición) {
    instrucciones;
}
```

Diagrama de flujo de la estructura condicional if

```
if (condición) {
    instrucciones;
}
```



Programación en ARDUINO: operadores condicionales



La evaluación de una condición puede devolver un valor true (si se cumple) o un valor false (si no se cumple).

Para expresar la condición se usan diversos **operadores**:

- **de comparación:**

`==` "igual que" `>` "mayor que"

`!=` "distinto de" `<=` "menor o igual que"

`<` "menor que" `>=` "mayor o igual que"

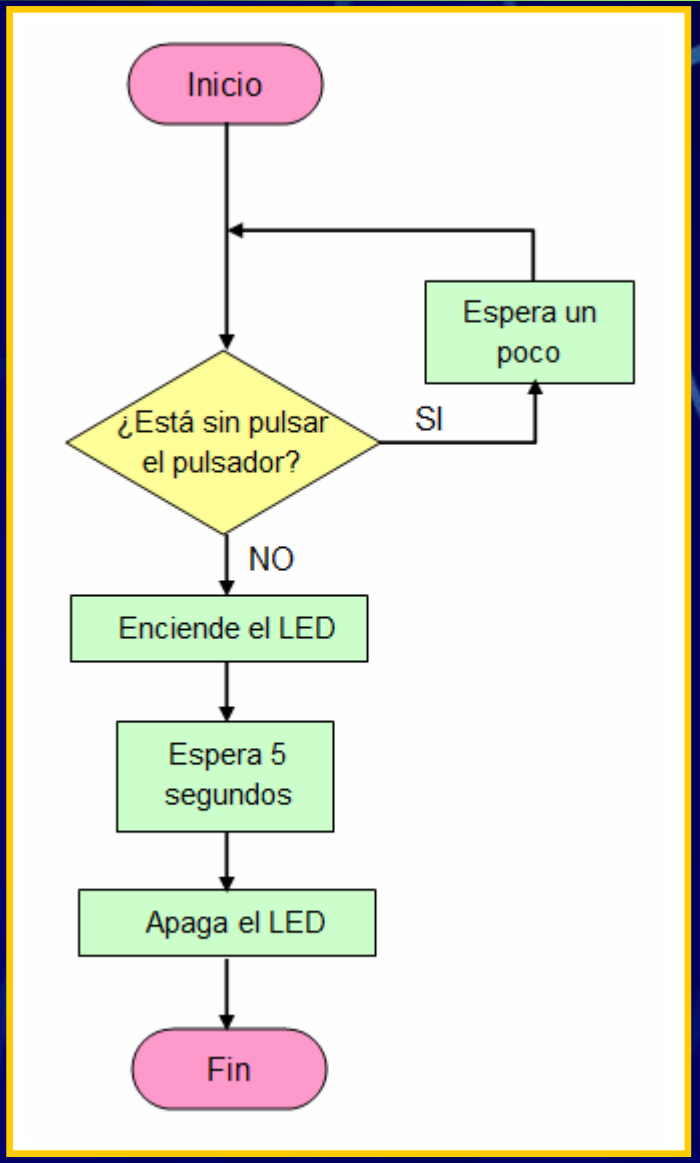
- **booleanos:**

`&&` "ambas condiciones se cumplen"

`||` "al menos una condición se cumple"

`!` "lo contrario a la condición se cumple"

Programación en ARDUINO: Bucle condicional while



Ejemplo: programa que espera hasta que se pulsa un pulsador para encender un LED durante 5 segundos, apagarlo y vuelve a esperar a que se pulse de nuevo el pulsador para volver a encenderlo.

- Diagrama de flujo de la función **loop()**.

La pequeña espera incluida entre dos consultas consecutivas del estado del pulsador se realiza porque se ha observado en la práctica que no hacerlo dar lugar a errores en la lectura de los pines. Suele ser una espera muy pequeña, de unos 10 milisegundos.

Programación en ARDUINO: Bucle condicional while

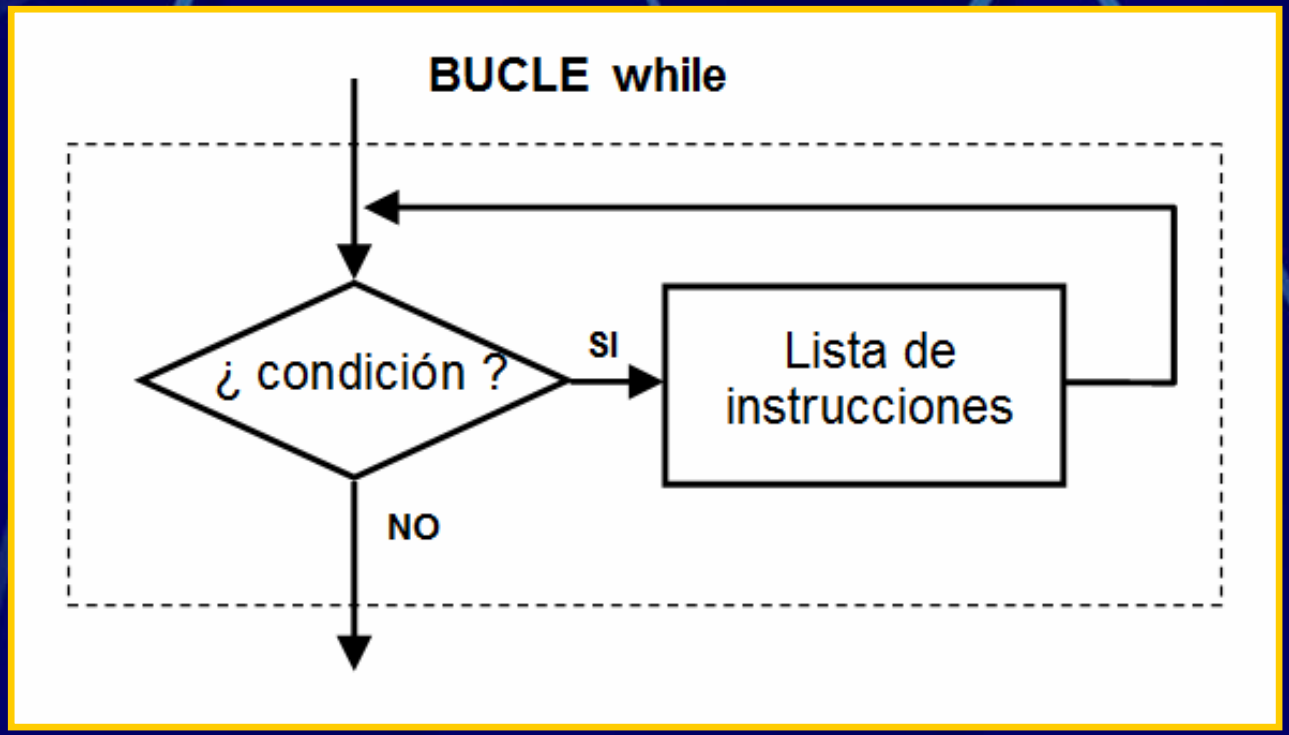
- El bucle condicional **while** repetirá indefinidamente las instrucciones que incluye hasta que la condición del while se evalúe como *false*.

```
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;
}
```



Programación en ARDUINO: Funciones de tiempo

Las funciones de tiempo permiten realizar temporizaciones en los programas.

- **delay (valor)**

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

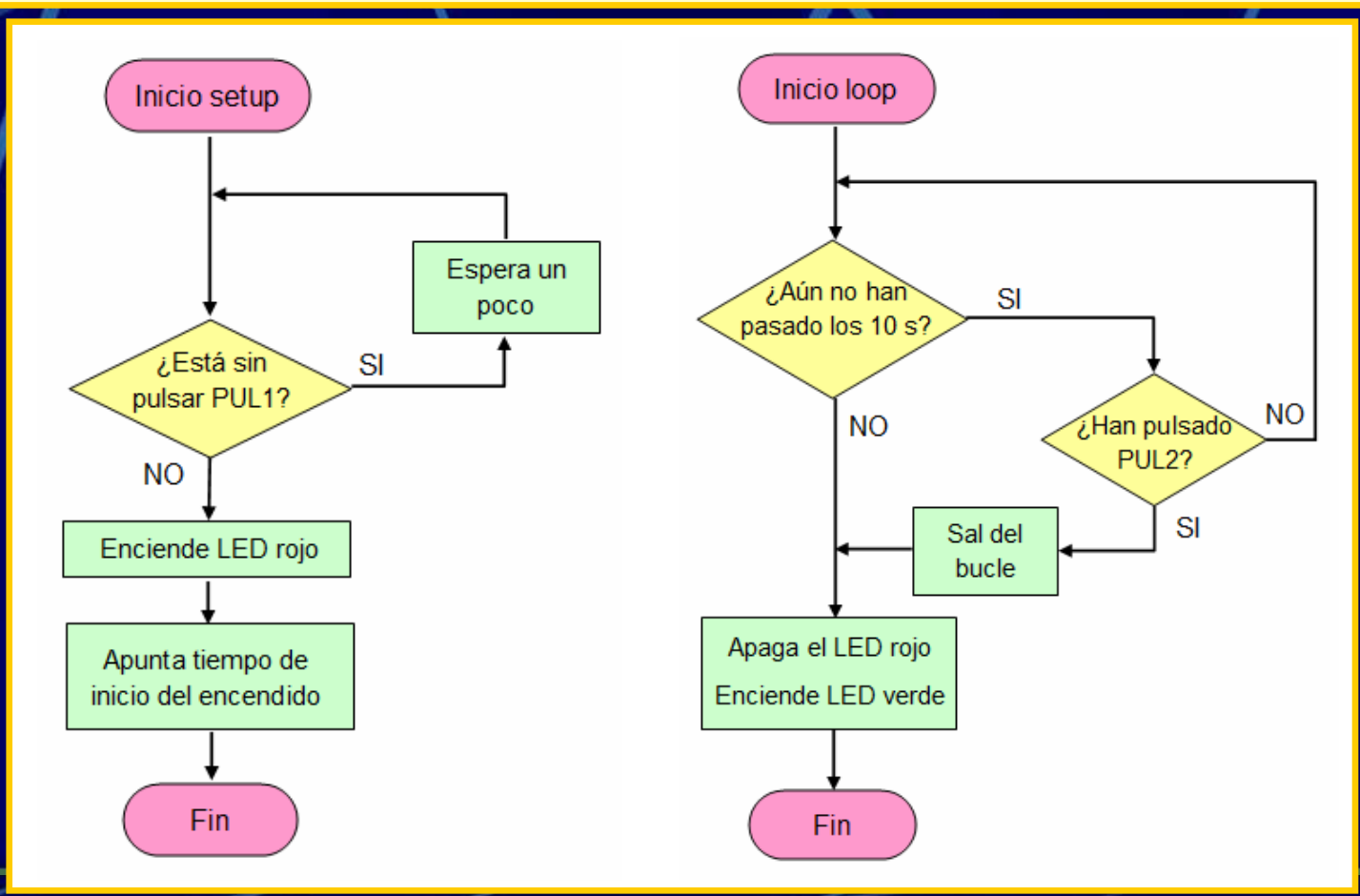
- **millis ( )**

  Devuelve el número de milisegundos transcurridos desde que Arduino empezó a correr el programa actual. El número crece rápido, por lo que si hay que guardar el valor en una variable conviene que sea del tipo unsigned long.

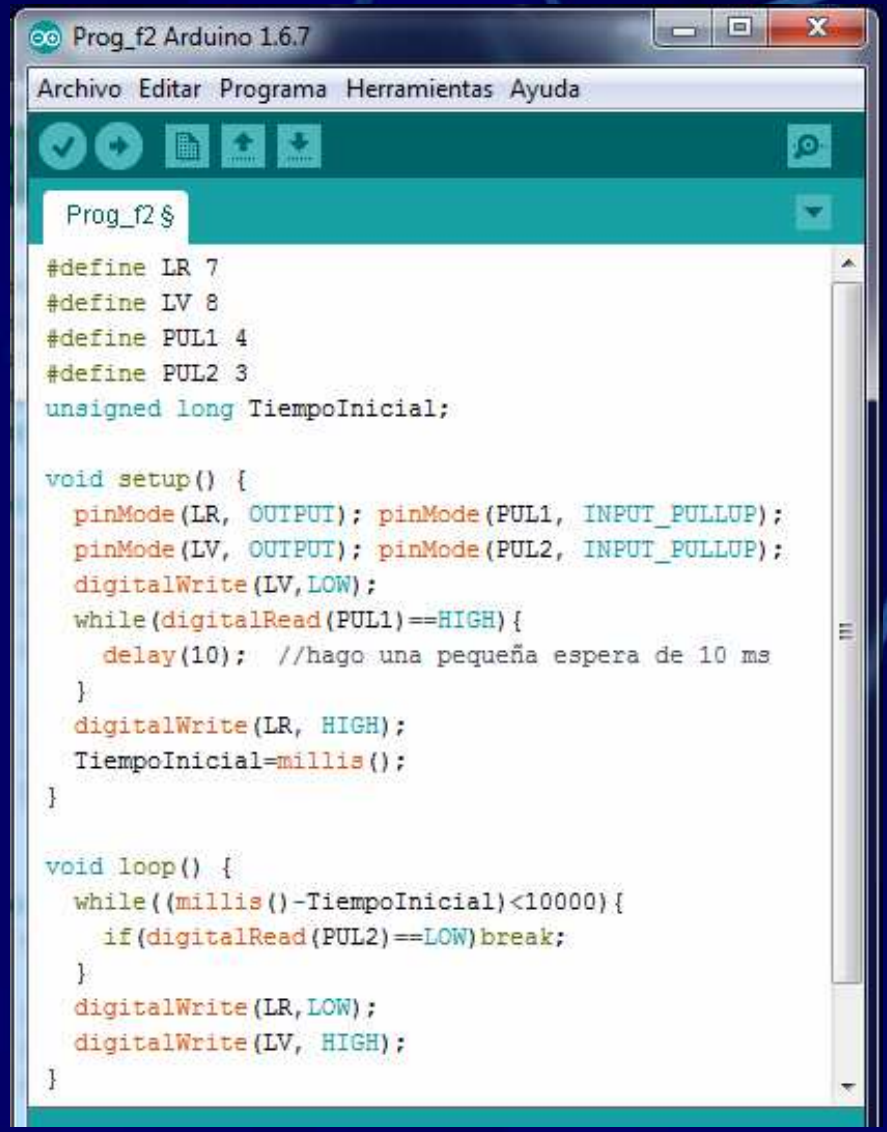
➢ Advertencia: mientras el programa está pausado con **delay()** no se leen las entradas, por lo que si hay un cambio en éstas no será captado por Arduino.

Programación en ARDUINO: salida inmediata de los bucles

Ejemplo: programa que espera a que se pulse PUL1 para encender el LED LR. Este LED se mantendrá encendido durante 10 segundos y después se encenderá el LED LV que se quedará encendido. Sin embargo, si antes de los 10 segundos se pulsa PUL2, entonces se apagará LR y se encenderá LV de inmediato sin esperar a los 10 segundos.



Programación en ARDUINO: break



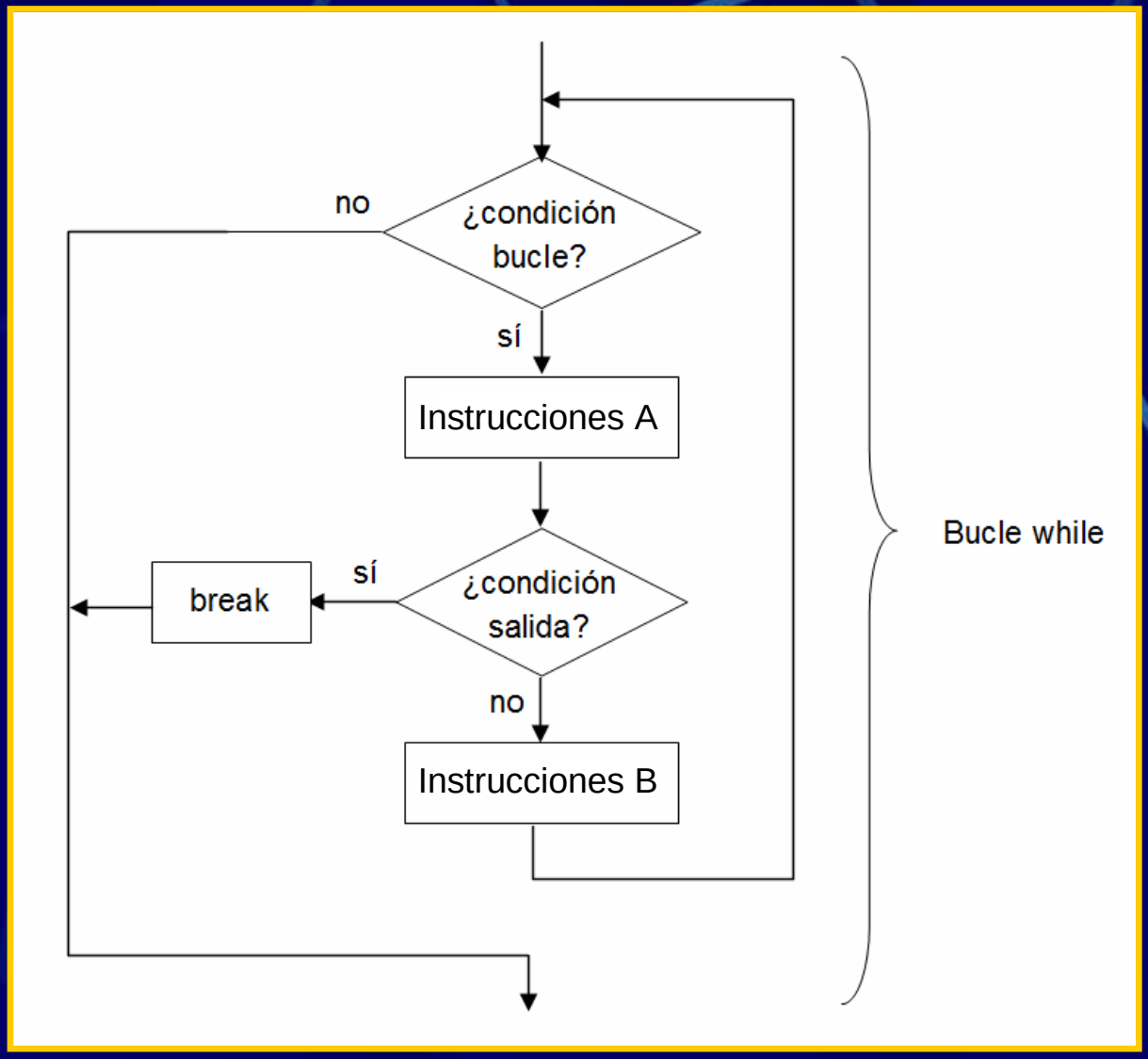

```
#define LR 7
#define LV 8
#define PUL1 4
#define PUL2 3
unsigned long TiempoInicial;

void setup() {
  pinMode(LR, OUTPUT); pinMode(PUL1, INPUT_PULLUP);
  pinMode(LV, OUTPUT); pinMode(PUL2, INPUT_PULLUP);
  digitalWrite(LV,LOW);
  while(digitalRead(PUL1)==HIGH){
    delay(10);  //hago una pequeña espera de 10 ms
  }
  digitalWrite(LR, HIGH);
  TiempoInicial=millis();
}

void loop() {
  while((millis()-TiempoInicial)<10000){
    if(digitalRead(PUL2)==LOW)break;
  }
  digitalWrite(LR,LOW);
  digitalWrite(LV, HIGH);
}
```

La instrucción **break** se utiliza para salir de forma inmediata de la estructura condicional en la que se encuentre.

Ejemplo: programa que espera a que se pulse PUL1 para encender el LED LR. Este LED se mantendrá encendido durante 10 segundos y después se encenderá el LED LV que se quedará encendido. Sin embargo, si antes de los 10 segundos se pulsa PUL2, entonces se apagará LR y se encenderá LV de inmediato sin esperar a los 10 segundos.

Gracias al **break**, al pulsar PUL2 salimos del bucle while sin tener que esperar a que se cumpla su condición de que hayan pasado los 10 segundos.

Programación en ARDUINO: break



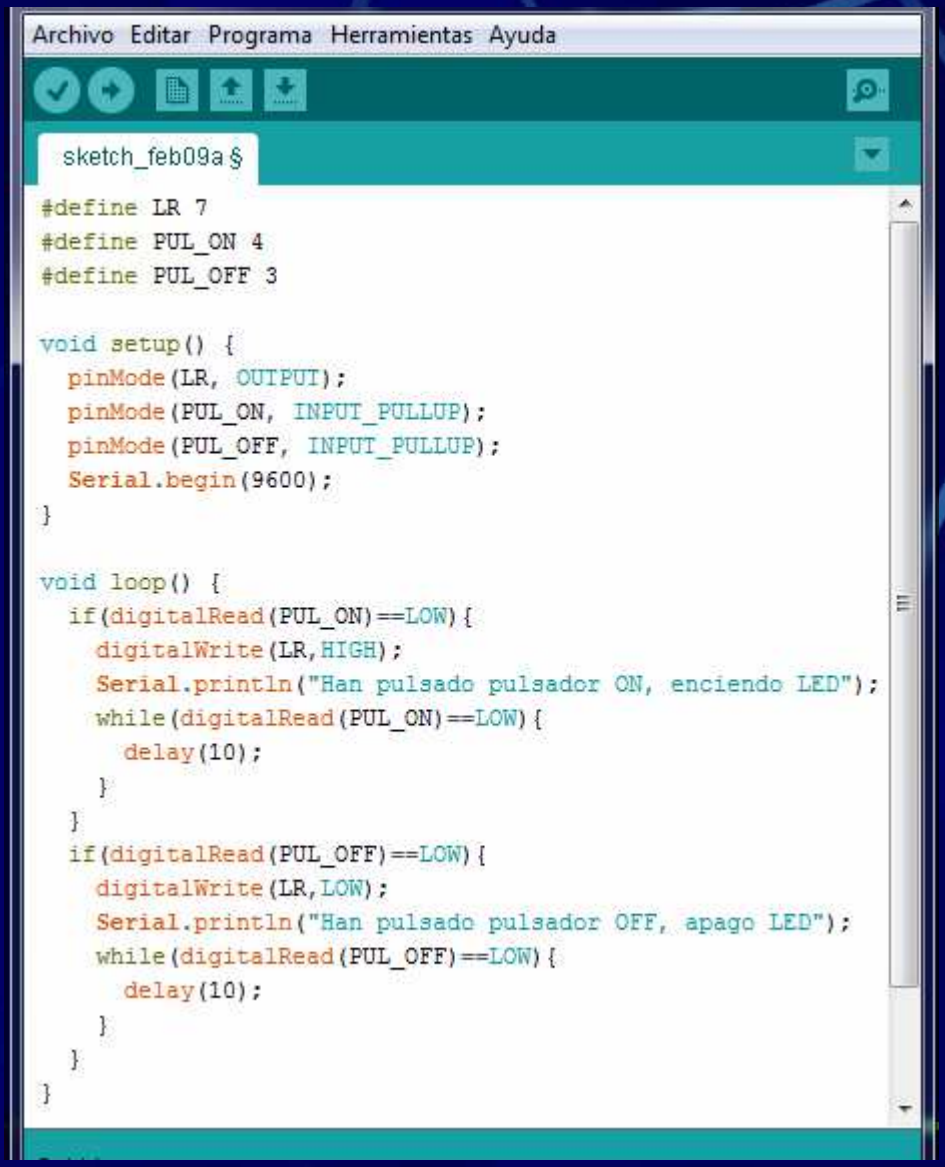
La instrucción **break** se utiliza para salir de forma inmediata de la estructura en la que se encuentre.

Habitualmente se utiliza en las estructuras repetitivas como el bucle **while** y en otras como **switch… case**.

◀ Salida de un bucle while con **break**.

Si se ejecuta **break**, se ejecutan las instrucciones A pero no las instrucciones B.

Programación en ARDUINO: El puerto serie




```
#define LR 7
#define PUL_ON 4
#define PUL_OFF 3

void setup() {
  pinMode(LR, OUTPUT);
  pinMode(PUL_ON, INPUT_PULLUP);
  pinMode(PUL_OFF, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  if(digitalRead(PUL_ON)==LOW){
    digitalWrite(LR,HIGH);
    Serial.println("Han pulsado pulsador ON, enciendo LED");
    while(digitalRead(PUL_ON)==LOW){
      delay(10);
    }
  }
  if(digitalRead(PUL_OFF)==LOW){
    digitalWrite(LR,LOW);
    Serial.println("Han pulsado pulsador OFF, apago LED");
    while(digitalRead(PUL_OFF)==LOW){
      delay(10);
    }
  }
}
```

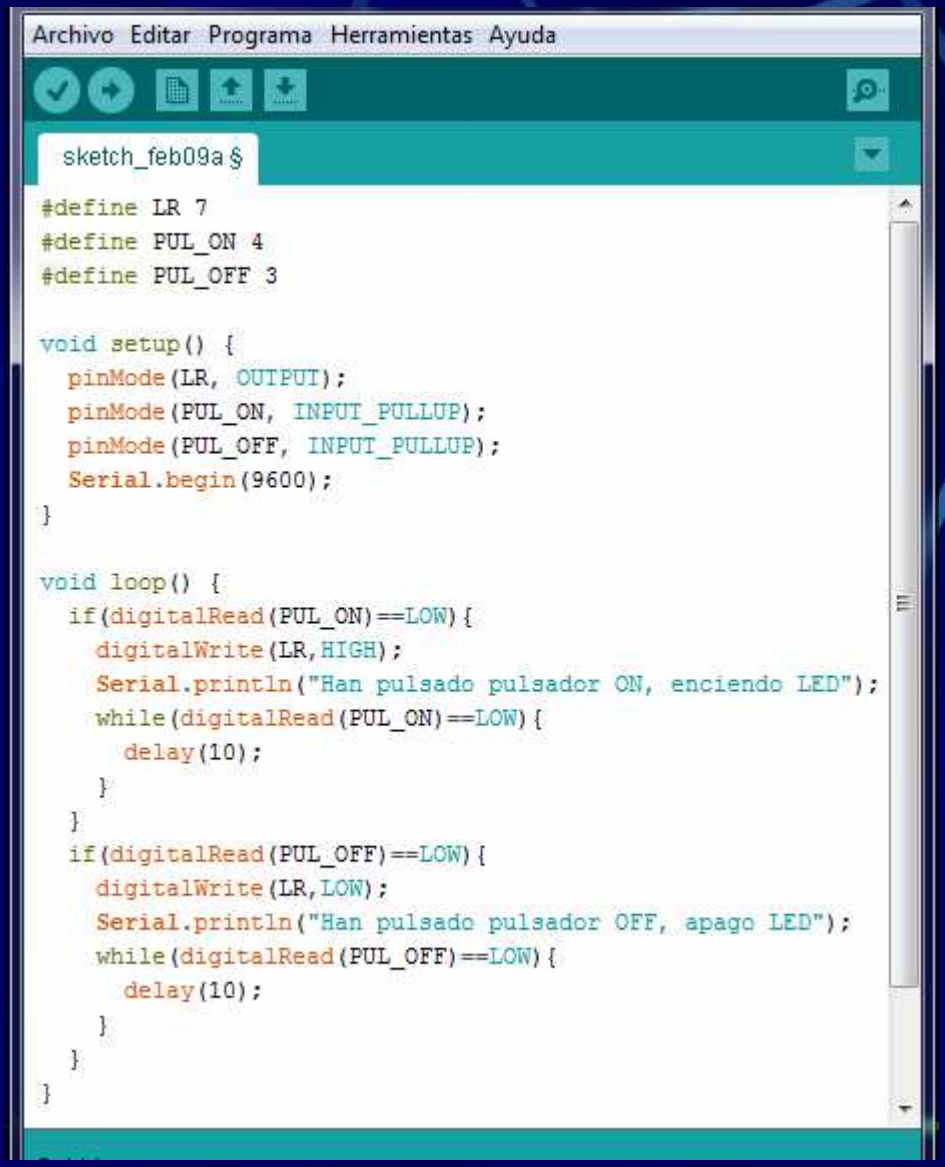
Nos puede interesar que el programa que se está ejecutando nos muestre mensajes. Esto es fácil cuando la placa Arduino está conectada a nuestro ordenador por el puerto USB (puerto serie).

Ejemplo: programa que enciende un LED al pulsar PUL_ON y lo apaga al pulsar PUL_OFF. Pero además, nos informa con un mensaje por el monitor del puerto serie cada vez que lo hace.

1º.- Debemos inicializar el puerto en nuestra función **setup()** añadiendo:

Serial.begin (9600)

Programación en ARDUINO: El puerto serie



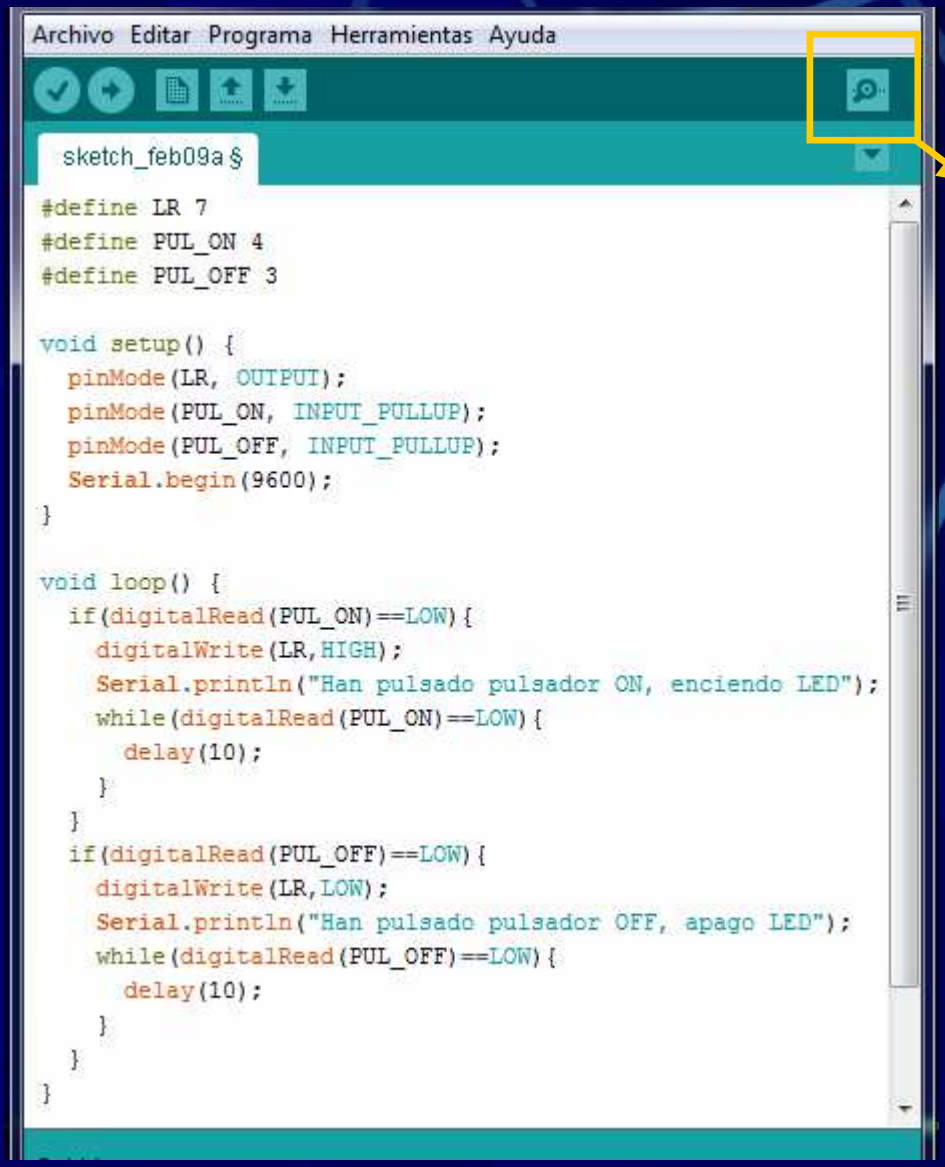
2º.- Para enviar mensajes al puerto serie, utilizaremos las funciones:

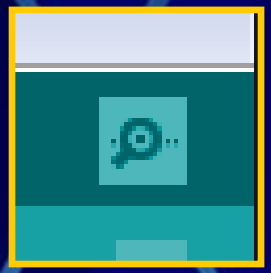
Serial.print ("mensaje")

Serial.println ("mensaje")

La diferencia entre ambas es que la segunda además de escribir un mensaje inserta una nueva línea, de forma que el siguiente mensaje se escribirá en el renglón siguiente, y no a continuación.

Programación en ARDUINO: El puerto serie





3º.- Para visualizar el monitor del puerto serie hacemos clic sobre el icono de la lupa.

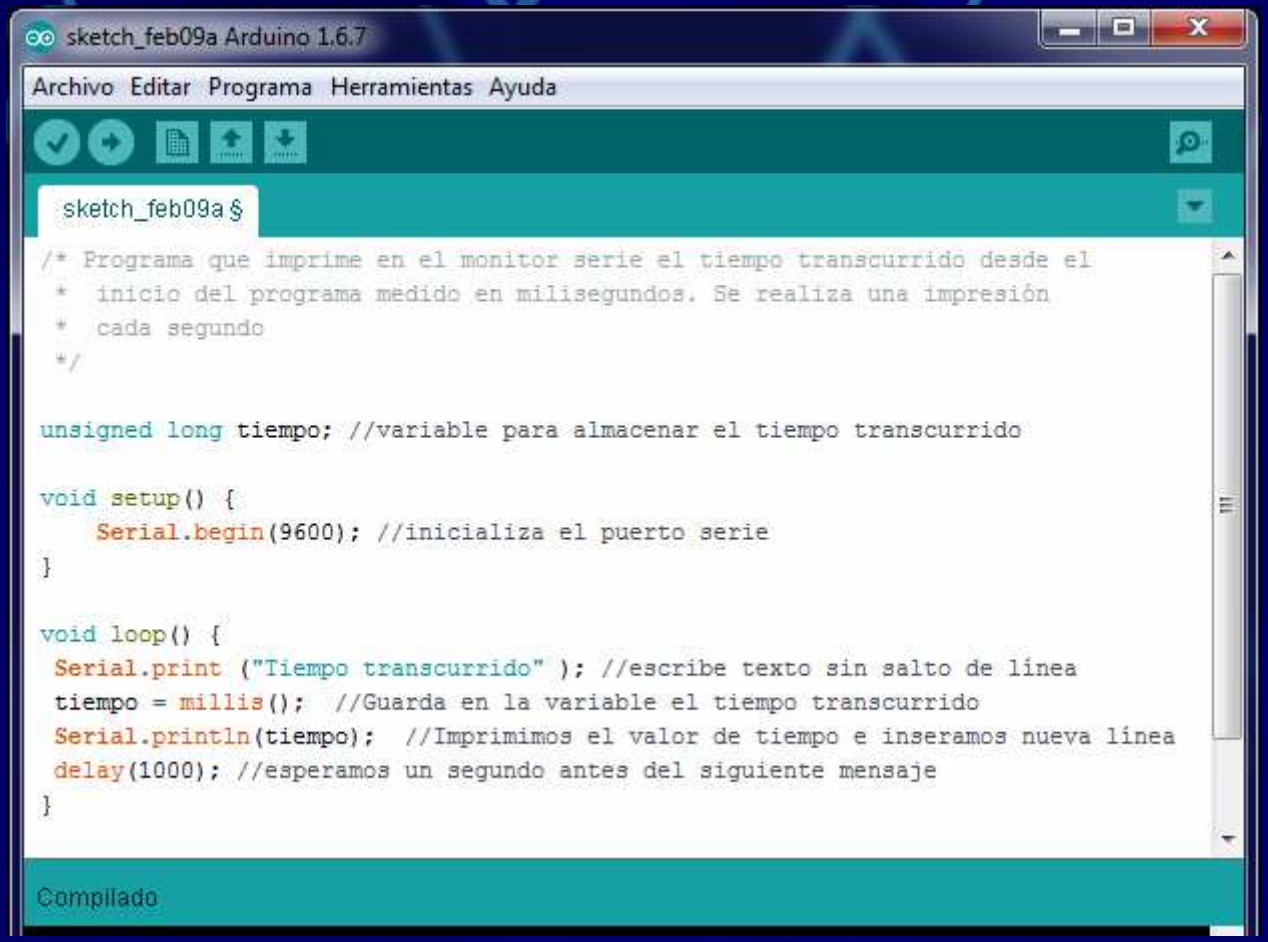
Programación en ARDUINO: El puerto serie

Además de mensajes también podemos imprimir en el monitor del puerto serie el valor de variables.

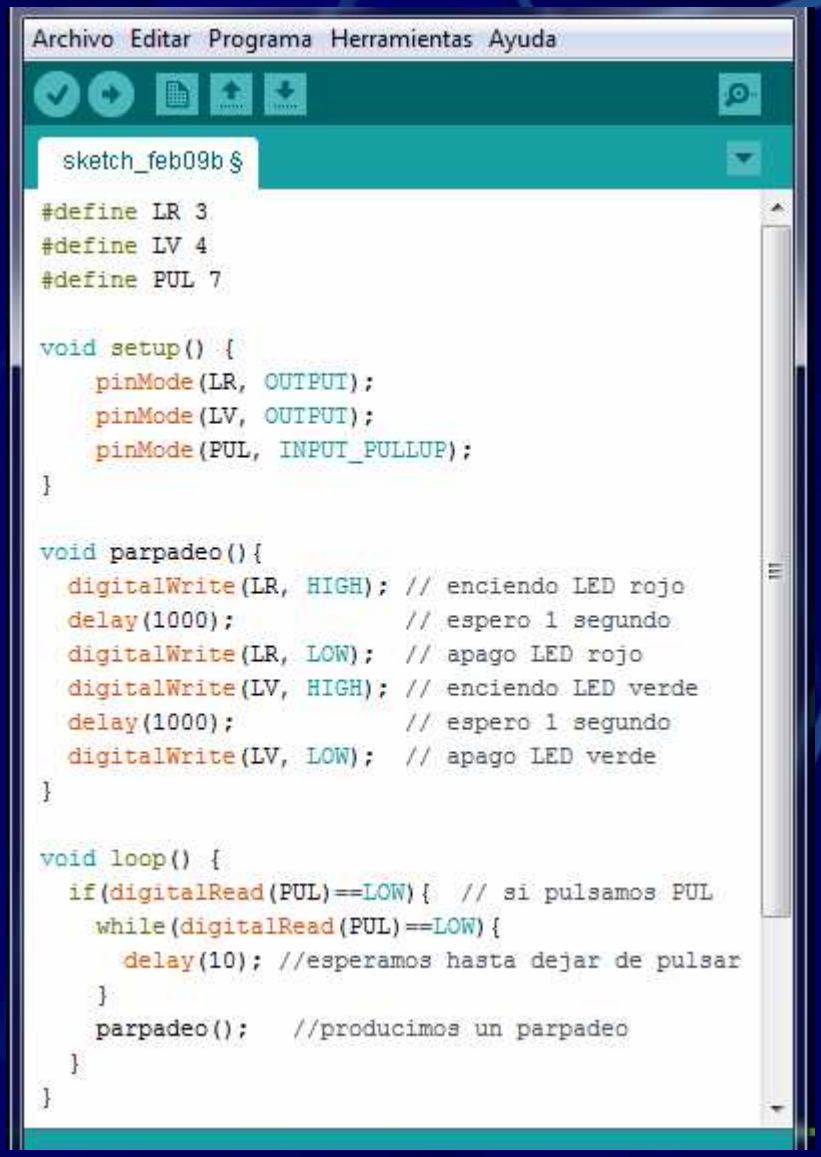
Serial.println (nombre_variable)

Ejemplo: programa que imprime en el monitor del puerto serie el tiempo transcurrido desde el inicio del programa expresado en milisegundos.

Realiza la operación una vez por segundo.



Programación en ARDUINO: Funciones del usuario



Los usuarios podemos diseñar **funciones propias** que realicen determinadas tareas y que sean llamadas por el programa cuando se requiera.

Estas funciones pueden definirse en cualquier parte del código, fuera de las funciones setup() y loop() y de otras funciones propias.

Ejemplo: programa que hace un "parpadeo" cada vez que pulsamos un pulsador. El parpadeo consiste en el encendido sucesivo de un LED rojo y un LED verde durante un segundo cada uno.

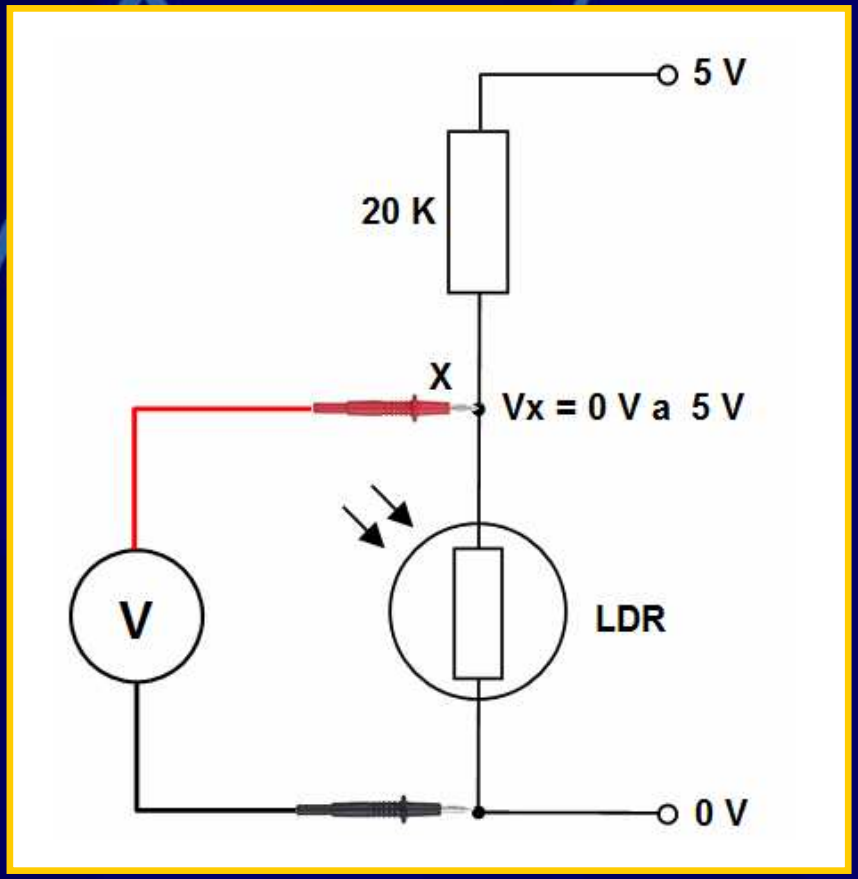
El parpadeo es realizado por una función propia definida por nosotros, que es llamada por la función loop().

Entradas analógicas

Para medir una magnitud analógica, como puede ser el nivel de luz que incide sobre una LDR (resistencia variable con la luz, cuanto más luz le incide menor es su resistencia) realizamos un montaje llamado "***divisor de tensión***" con una resistencia de valor fijo adecuado.

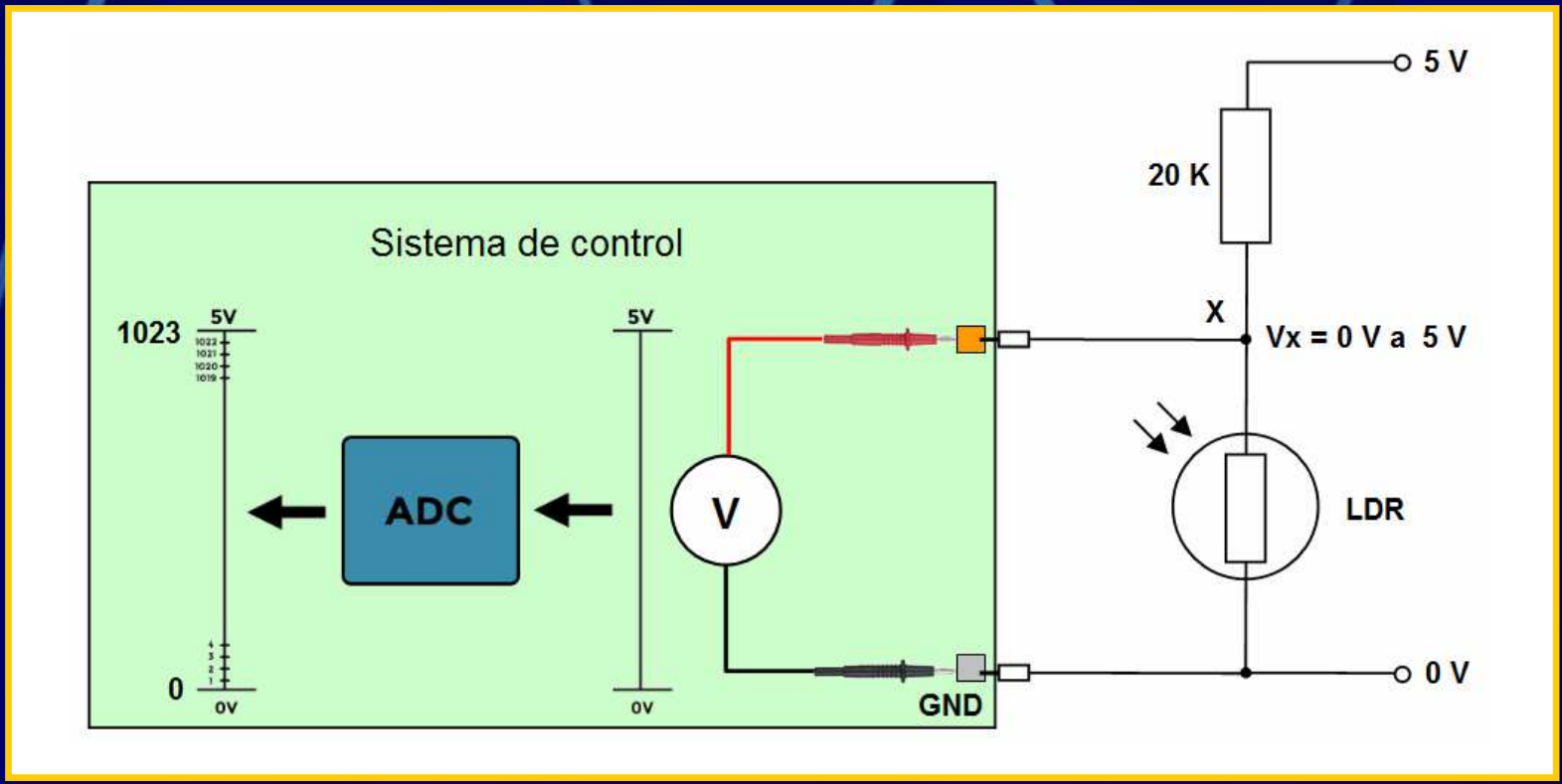
La tensión medida por el voltímetro en el punto X del circuito dependerá del nivel de luz incidente en la LDR.

- Si incide mucha luz, la resistencia de la LDR será muy pequeña respecto a la resistencia fija y, por tanto, la tensión en X será baja.
- Si incide poca luz, la resistencia de la LDR será muy grande respecto a la resistencia fija y, por tanto, la tensión en X será alta.



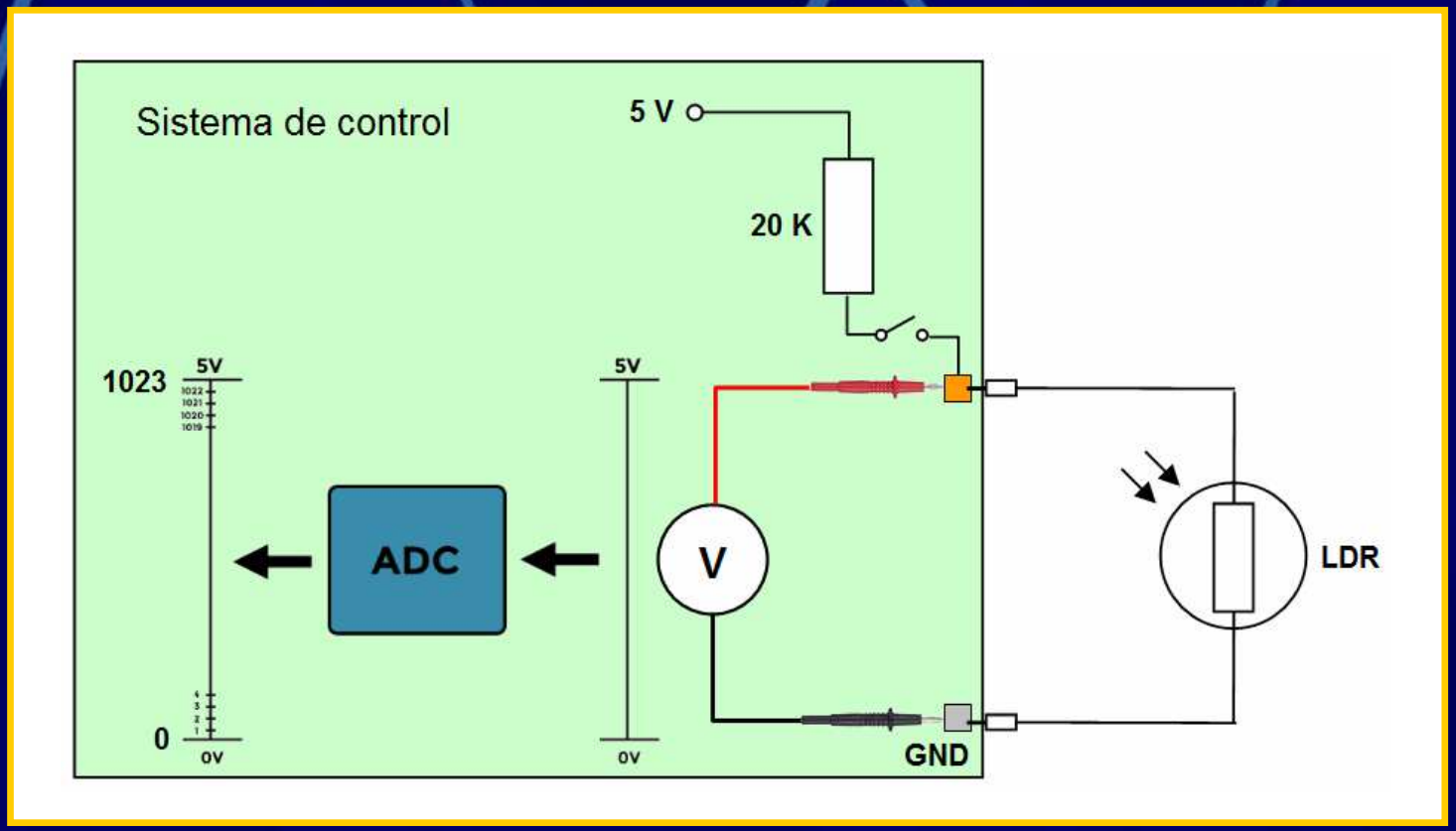
Señales analógicas en sistemas digitales

Como dijimos, la función del voltímetro la hace el microcontrolador, sin embargo, éstos no pueden trabajar con señales analógicas, por lo que incorporan un dispositivo llamado "**convertidor analógico-digital**", que transforma los valores de tensión en un número dentro de un rango. Concretamente, Arduino convierte valores de tensión de 0 a 5 V en un número comprendido entre 0 y 1023.



Señales analógicas en sistemas digitales

Para no tener que conectar resistencias externas para formar el divisor de tensión con el elemento sensor (LDR en este caso), Arduino incorpora unas resistencias internas conectadas a 5 V que puede conectar internamente al pin de entrada analógica si definimos a éste del tipo **INPUT_PULLUP** con la función **pinMode()**.



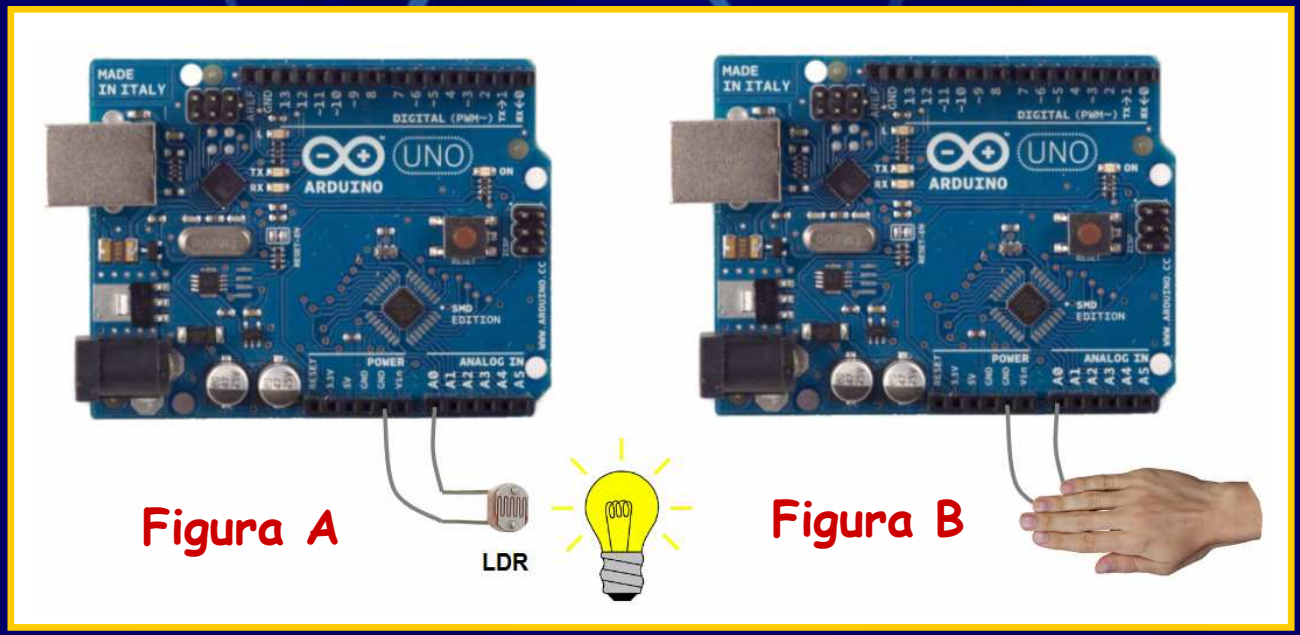
Entradas analógicas en Arduino

Las entradas analógicas de Arduino se reconocen porque se nombran con una A delante del número (A0, A1,…., A5). En el ejemplo de la figura usamos el pin A0. Vemos que la LDR está conectada entre el pin A0 y GND, luego tenemos que definir el pin A0 como INPUT_PULLUP.

```
pinMode (A0, INPUT_PULLUP)
```

Para tomar la lectura el programa usará la orden `analogRead (A0)`

Esta orden devolverá un valor comprendido entre 0 y 1023. En la figura A (donde la LDR está iluminada), el valor devuelto será bajo, en la figura B (con la LDR tapada), el valor será alto.



Programación en ARDUINO: Entradas analógicas

Se lee en una entrada analógica con la función:

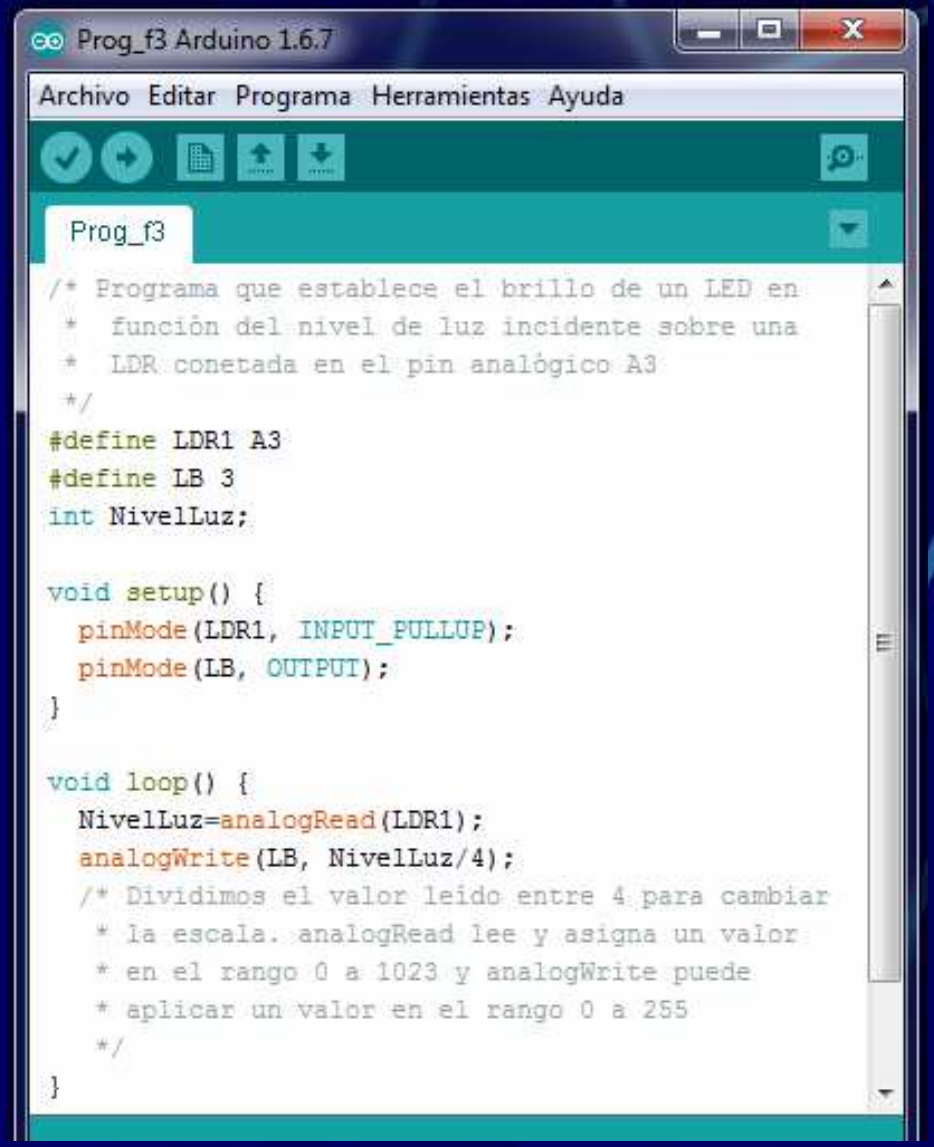
- **analogRead (pin)**

Devuelve un valor entre 0 y 1023 que corresponden a tensiones entre 0 V y 5 V respectivamente.

Ejemplo: programa que lee el nivel de luz que incide sobre la LDR conectada al pin A0 continuamente. Cuando hay luz apaga el LED LB y cuando no hay lo enciende. El límite lo marca la variable ValorLimite.

Los pines analógicos también pueden funcionar como digitales. Los digitales no pueden funcionar como analógicos.

Programación en ARDUINO: Salidas analógicas



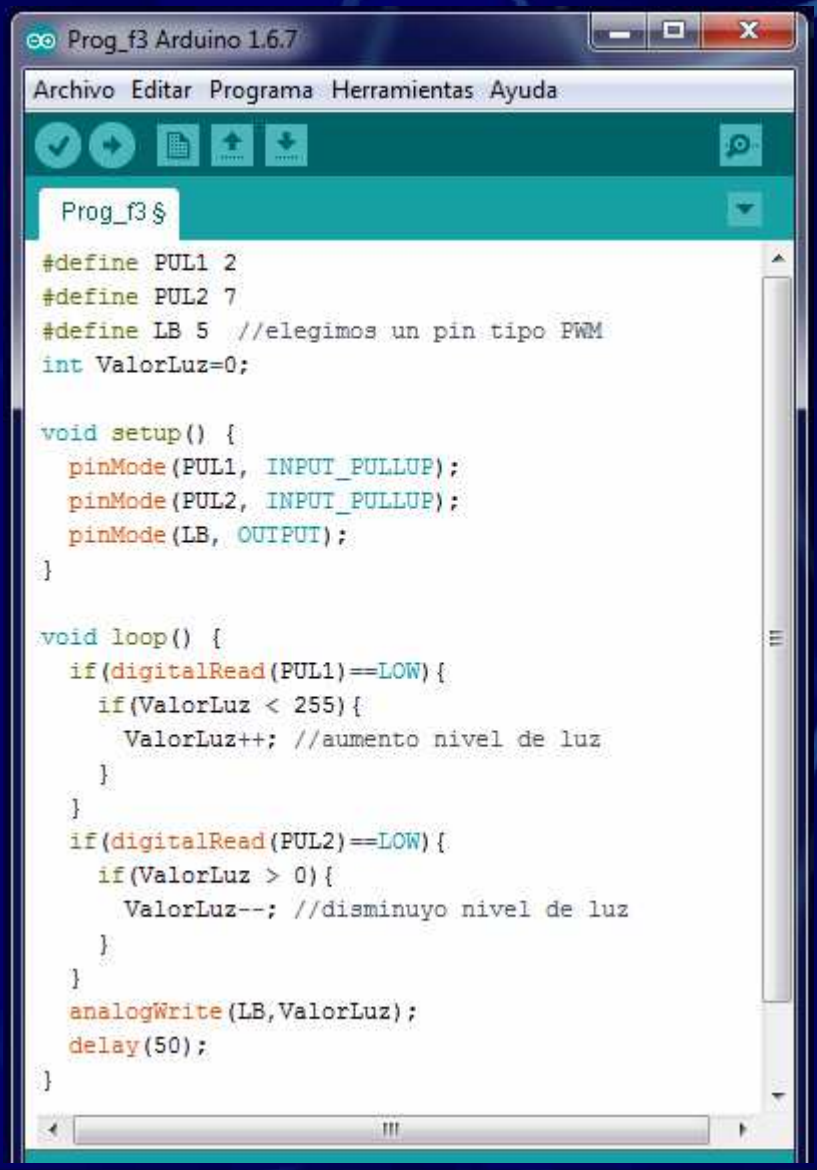

```
/* Programa que establece el brillo de un LED en
 *  función del nivel de luz incidente sobre una
 *  LDR conetada en el pin analógico A3
 */
#define LDR1 A3
#define LB 3
int NivelLuz;

void setup() {
  pinMode(LDR1, INPUT_PULLUP);
  pinMode(LB, OUTPUT);
}

void loop() {
  NivelLuz=analogRead(LDR1);
  analogWrite(LB, NivelLuz/4);
  /* Dividimos el valor leído entre 4 para cambiar
   * la escala. analogRead lee y asigna un valor
   * en el rango 0 a 1023 y analogWrite puede
   * aplicar un valor en el rango 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 (van marcados con signo ~).
- 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.

Programación en ARDUINO: Salidas analógicas

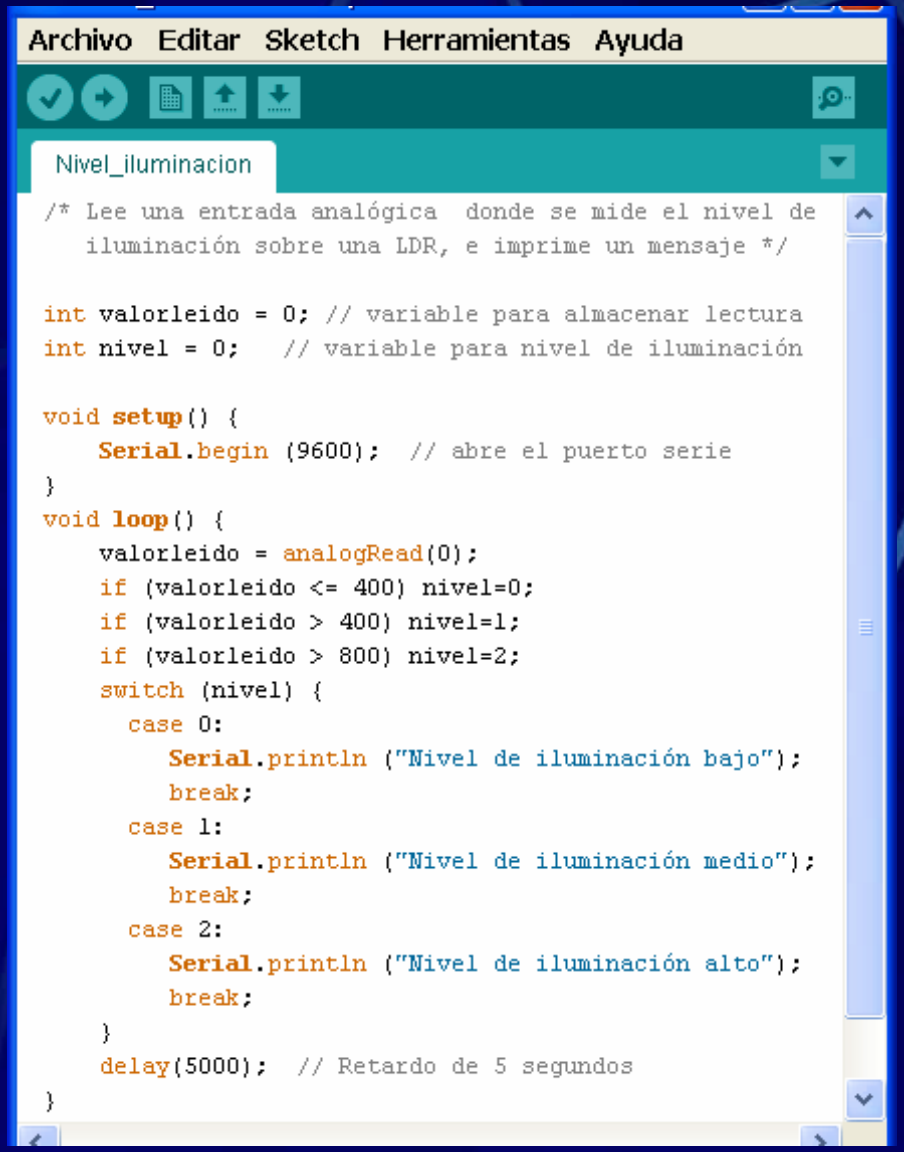



```
#define PUL1 2
#define PUL2 7
#define LB 5  //elegimos un pin tipo PWM
int ValorLuz=0;

void setup() {
  pinMode(PUL1, INPUT_PULLUP);
  pinMode(PUL2, INPUT_PULLUP);
  pinMode(LB, OUTPUT);
}

void loop() {
  if(digitalRead(PUL1)==LOW){
    if(ValorLuz < 255){
      ValorLuz++; //aumento nivel de luz
    }
  }
  if(digitalRead(PUL2)==LOW){
    if(ValorLuz > 0){
      ValorLuz--; //disminuyo nivel de luz
    }
  }
  analogWrite(LB,ValorLuz);
  delay(50);
}
```

Ejemplo: programa que hace que un LED aumente su luz gradualmente si se mantiene pulsado PUL1 y que desciende su luz gradualmente si se mantiene pulsado PUL2.

- El nombre de una variable seguido de dos signos positivos (++) lo que hace es incrementar en 1 dicha variable.
- El nombre de una variable seguido de dos signos negativos (– –) lo que hace es disminuir en 1 dicha variable.

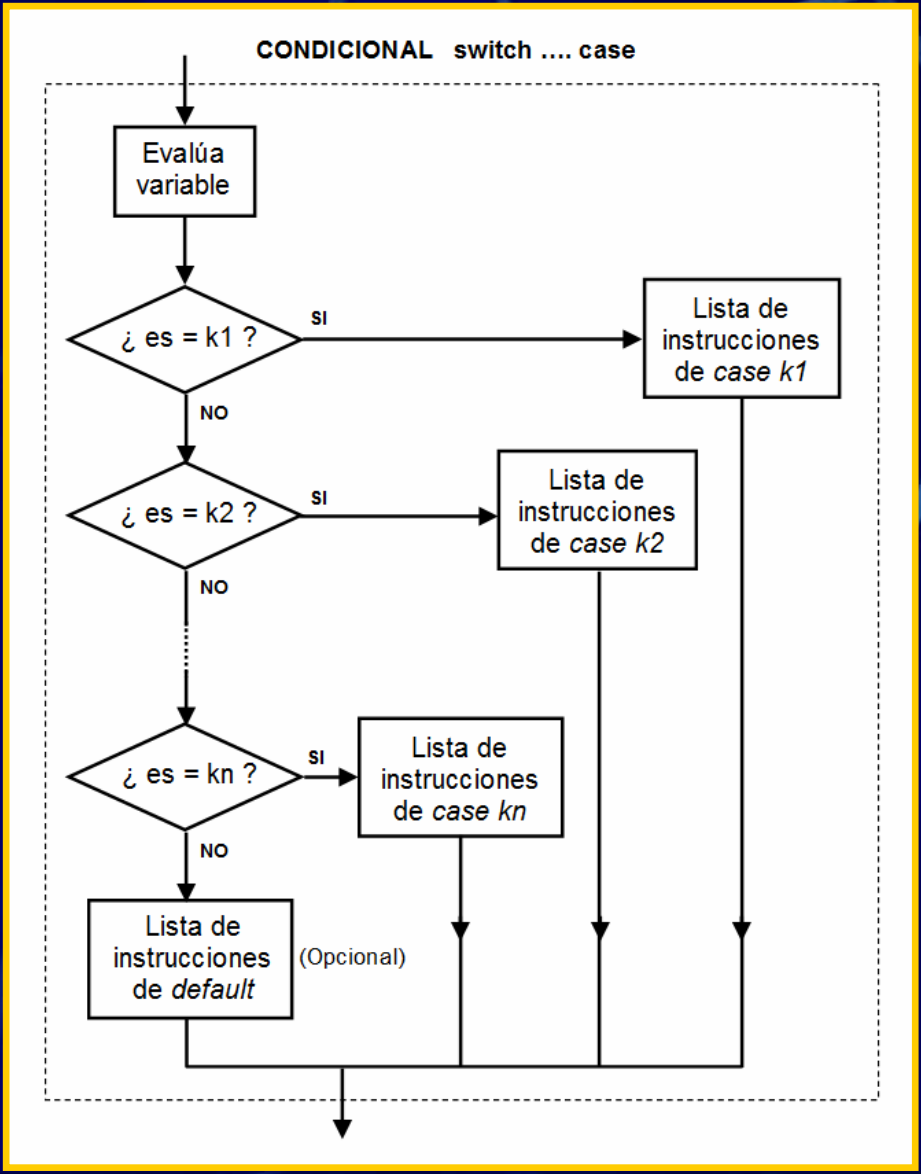
Program. en ARDUINO: Estructura condicional switch...case



- 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.;
}
```