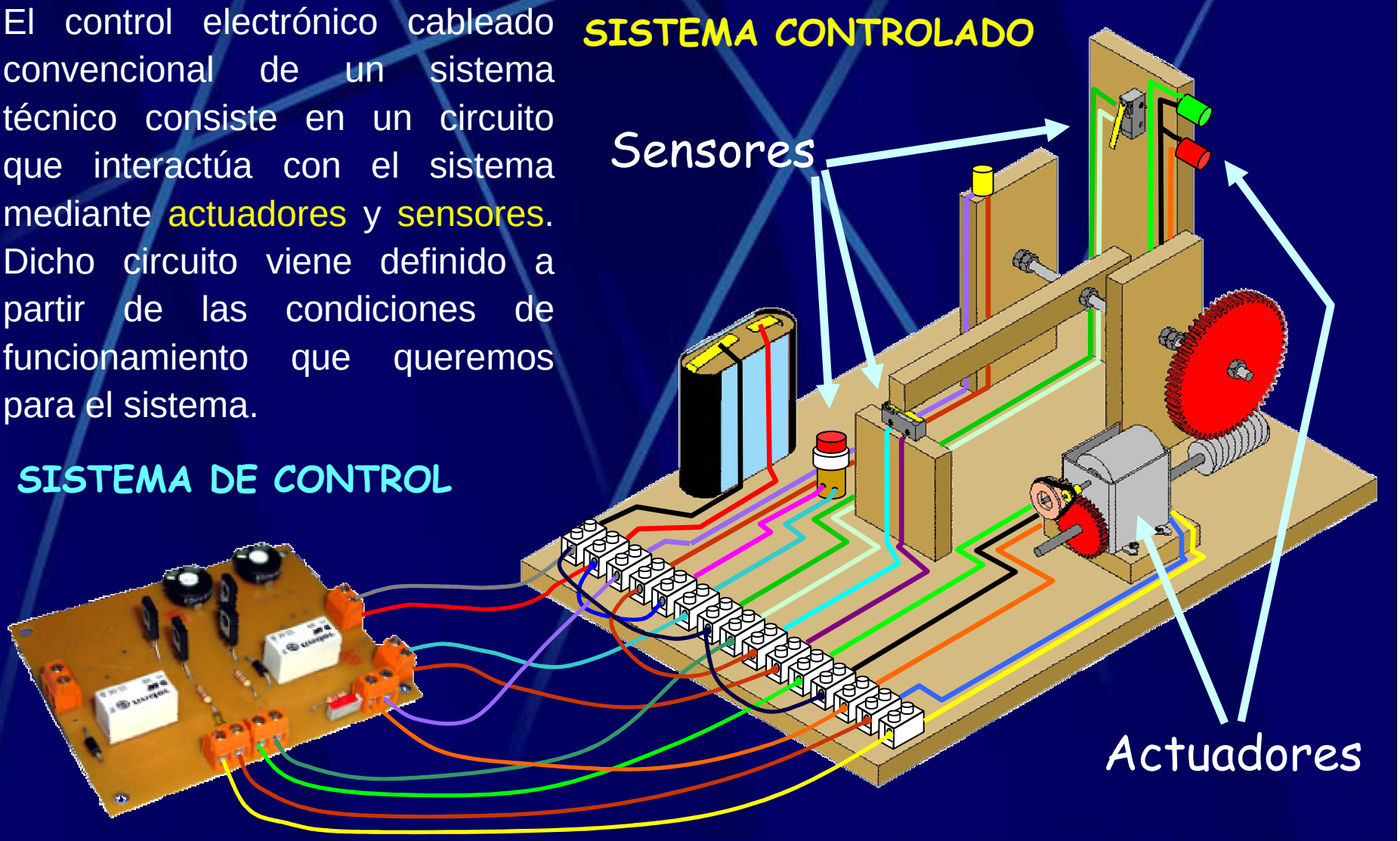
Control programado con ARDUINO

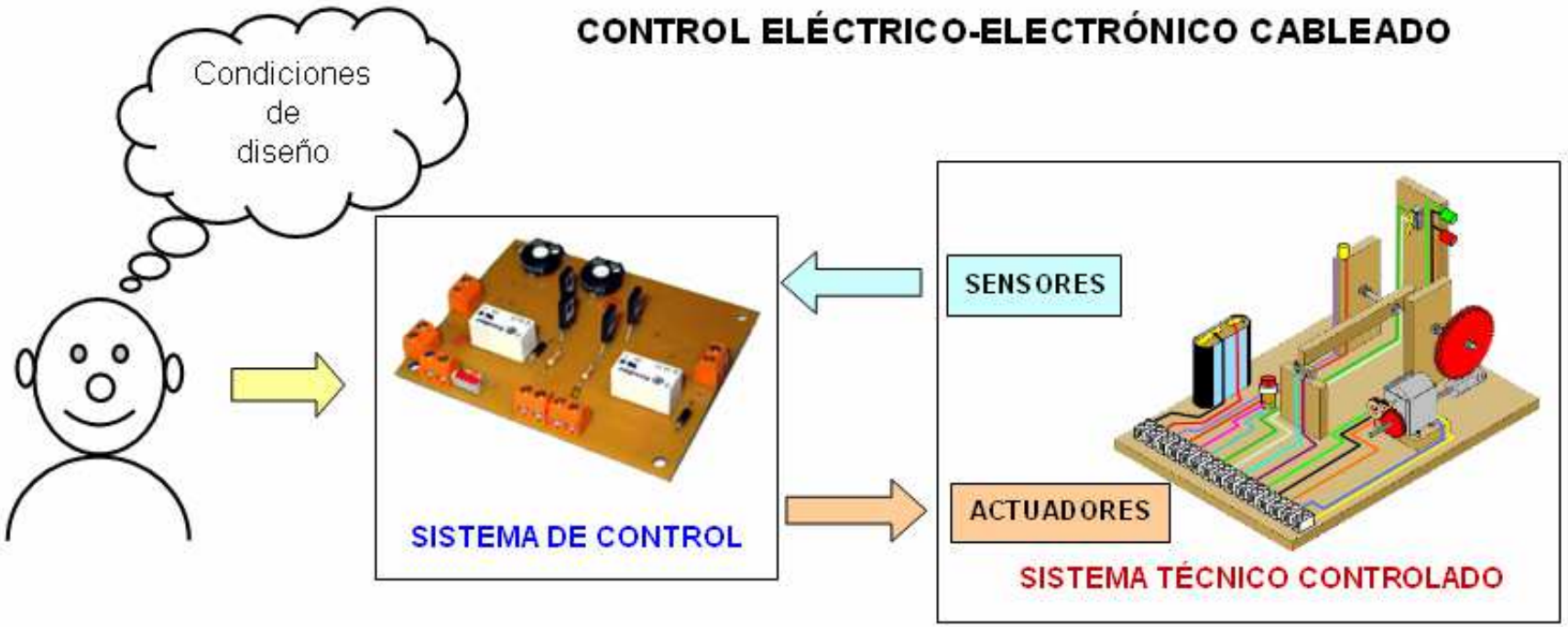
TECNOLOGÍA

Control de un sistema técnico “cableado”

El control electrónico cableado convencional de un sistema técnico consiste en un circuito que interactúa con el sistema mediante actuadores y sensores. Dicho circuito viene definido a partir de las condiciones de funcionamiento que queremos para el sistema.

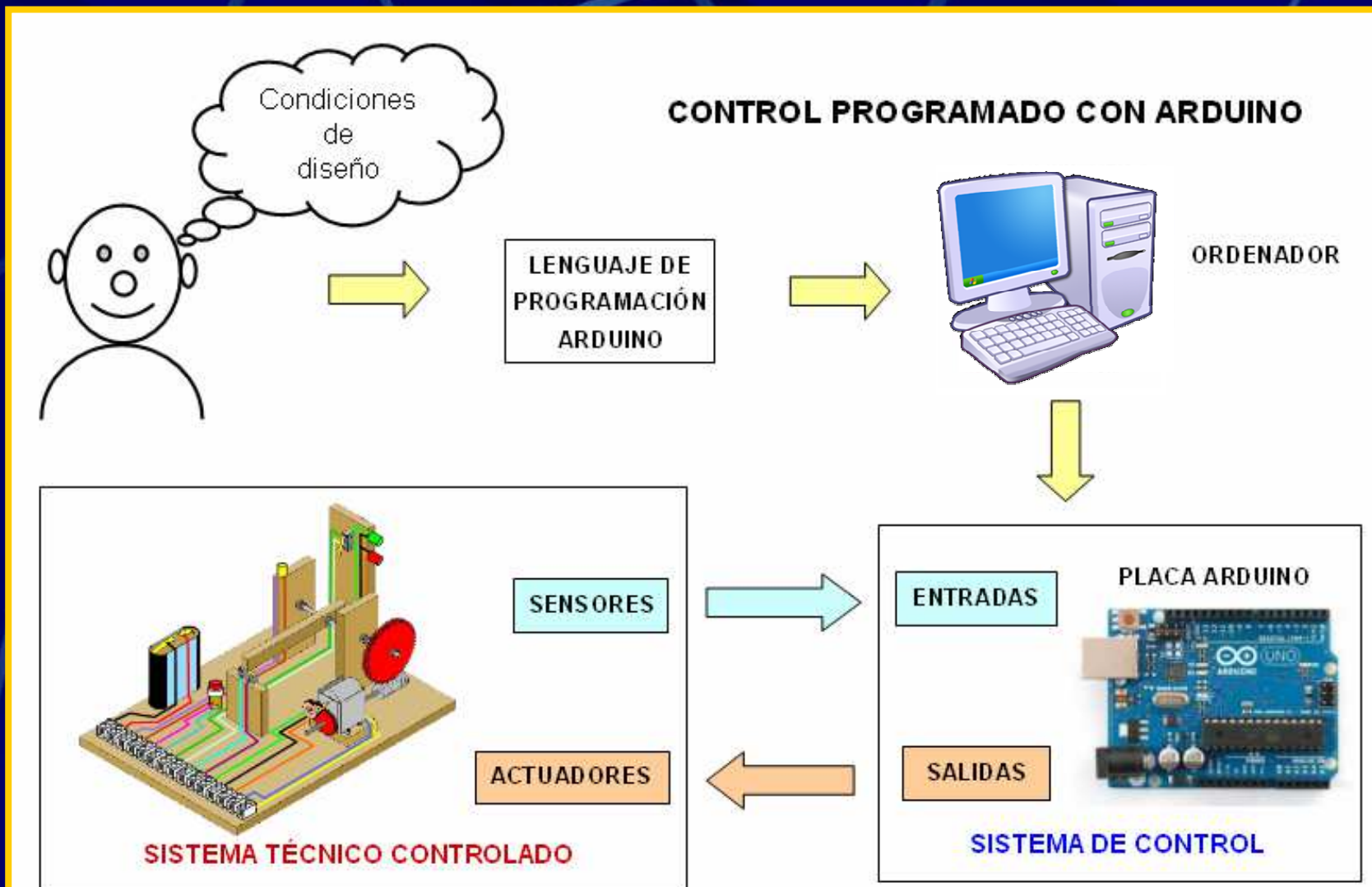


Control eléctrico-electrónico “cableado”



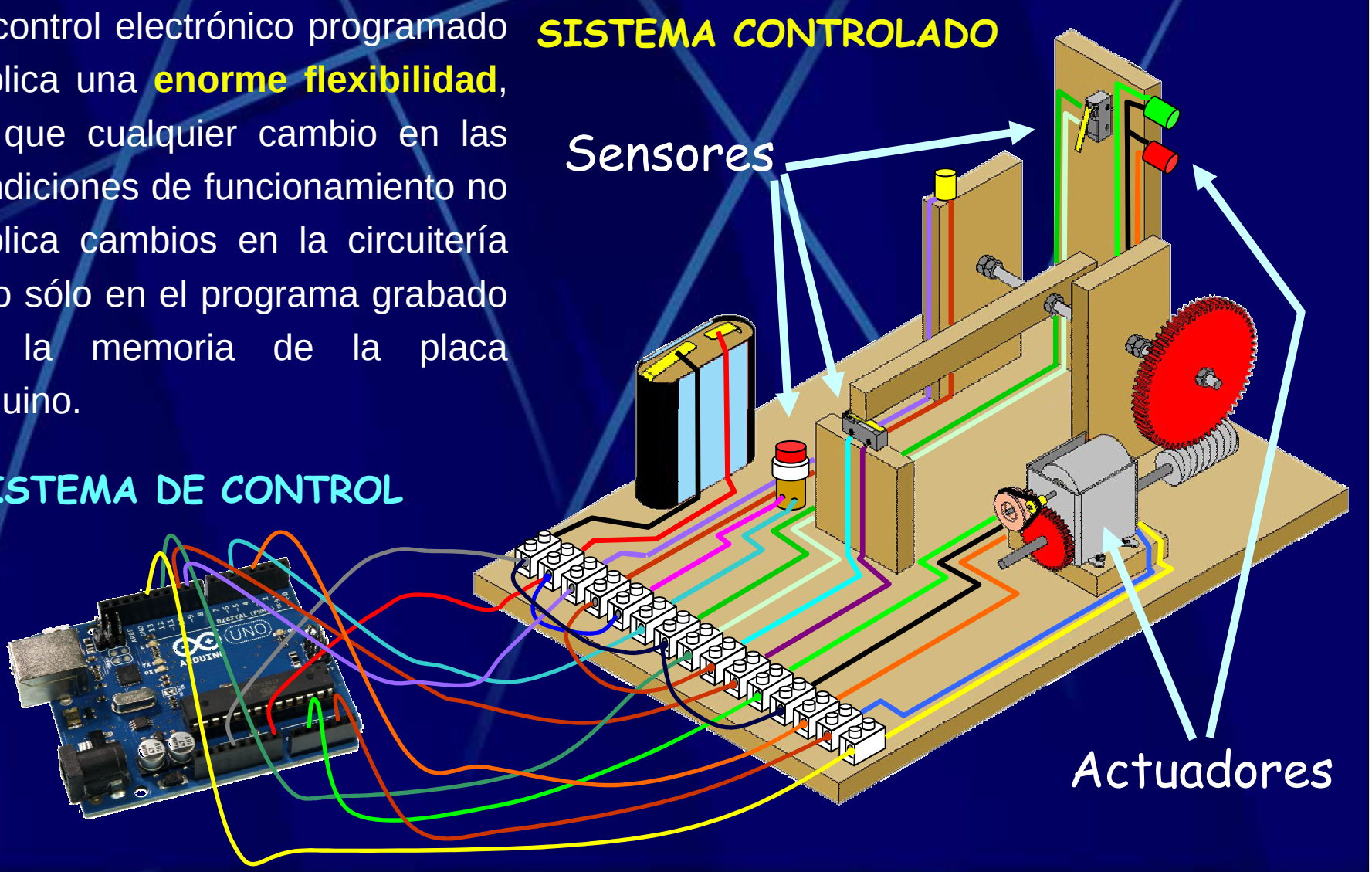
Es **muy rígido**, cualquier cambio en el funcionamiento requiere cambios en los circuitos por personal especializado.

Control programado con ARDUINO



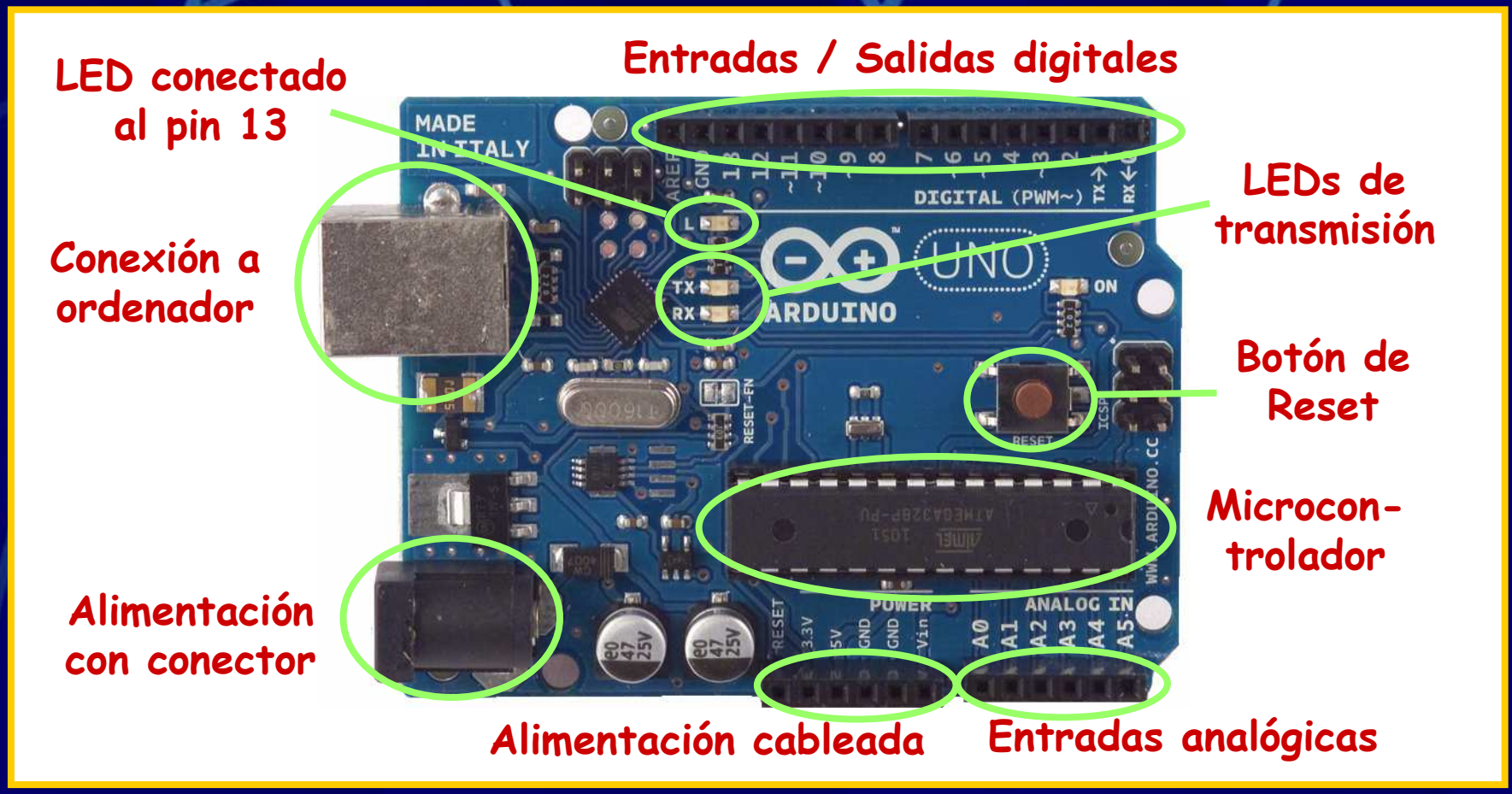
Control programado con Arduino

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 de la placa Arduino.



La plataforma ARDUINO

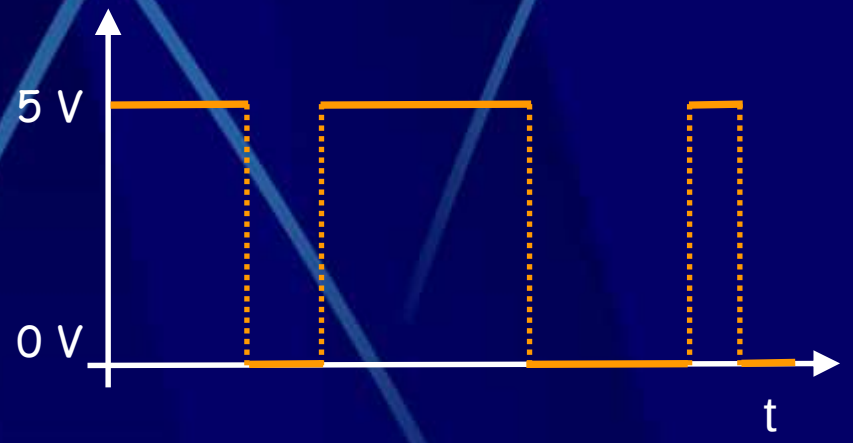
ARDUINO es una plataforma de **hardware libre** basada en una placa con un microcontrolador y un entorno de desarrollo integrado (IDE).



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**.

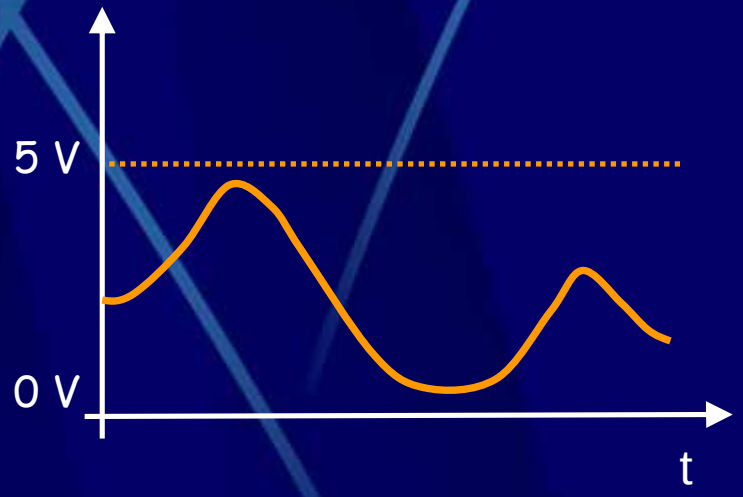


- Las **entradas digitales** de un microcontrolador sólo pueden diferenciar estos dos niveles de tensión o cercanos a ellos.
- Las **salidas digitales** de los microcontroladores sólo pueden aplicar estos dos niveles de tensión a lo que se conecte a ellas.

Entradas / Salidas analógicas

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

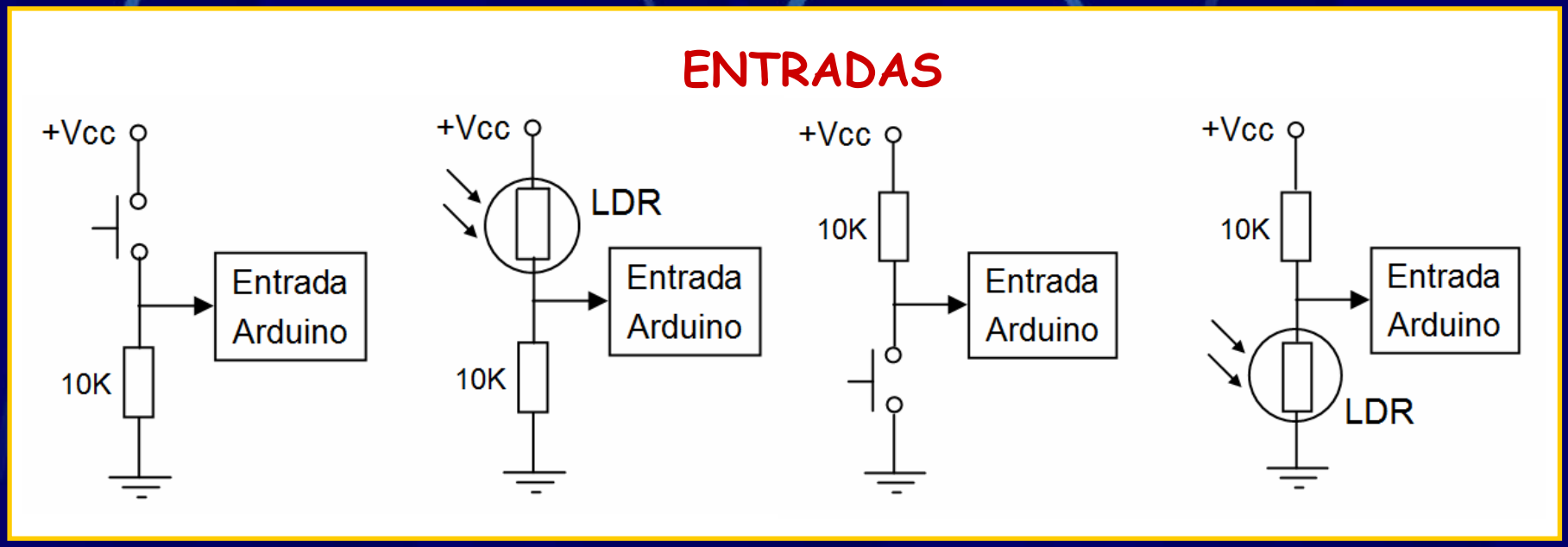
- Como los microcontroladores operan internamente con señales digitales, incorporan un convertidor analógico-digital que transforma las señales analógicas recibidas en las **entradas analógicas** en un valor numérico.



- 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 “pseudo analógica” 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.



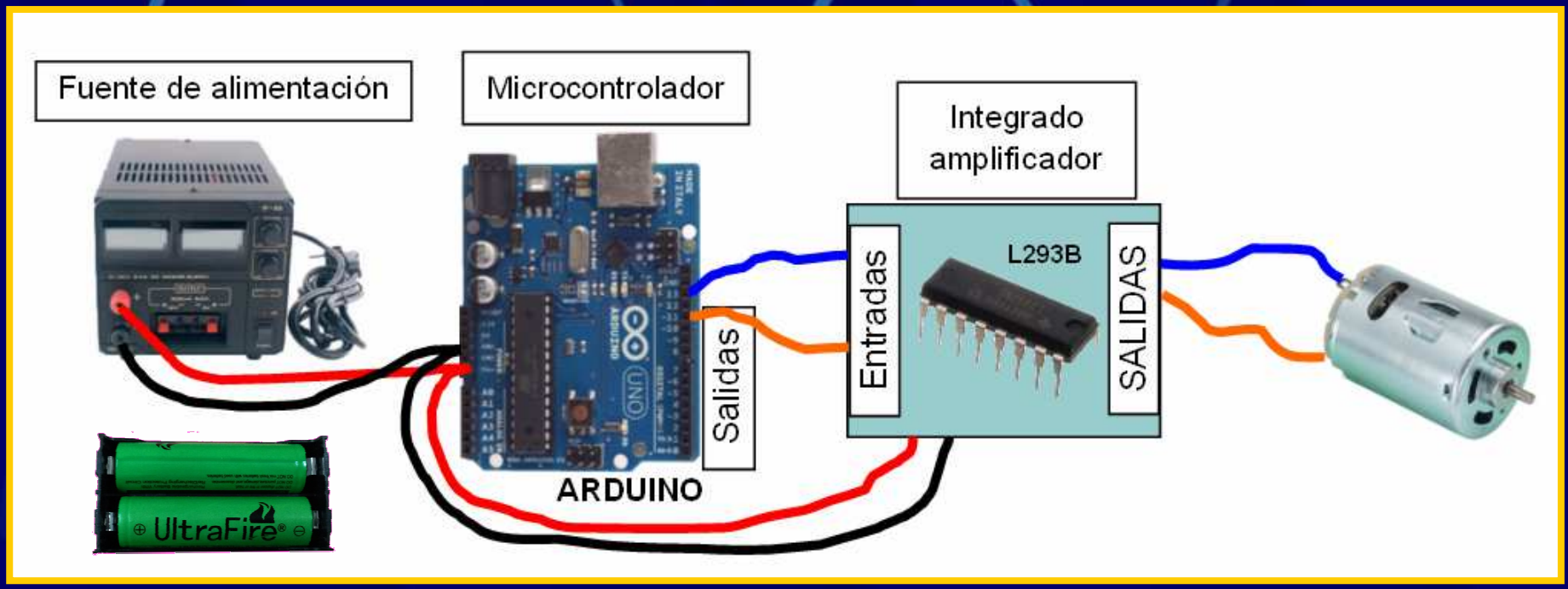
Protección de las salidas de Arduino

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.

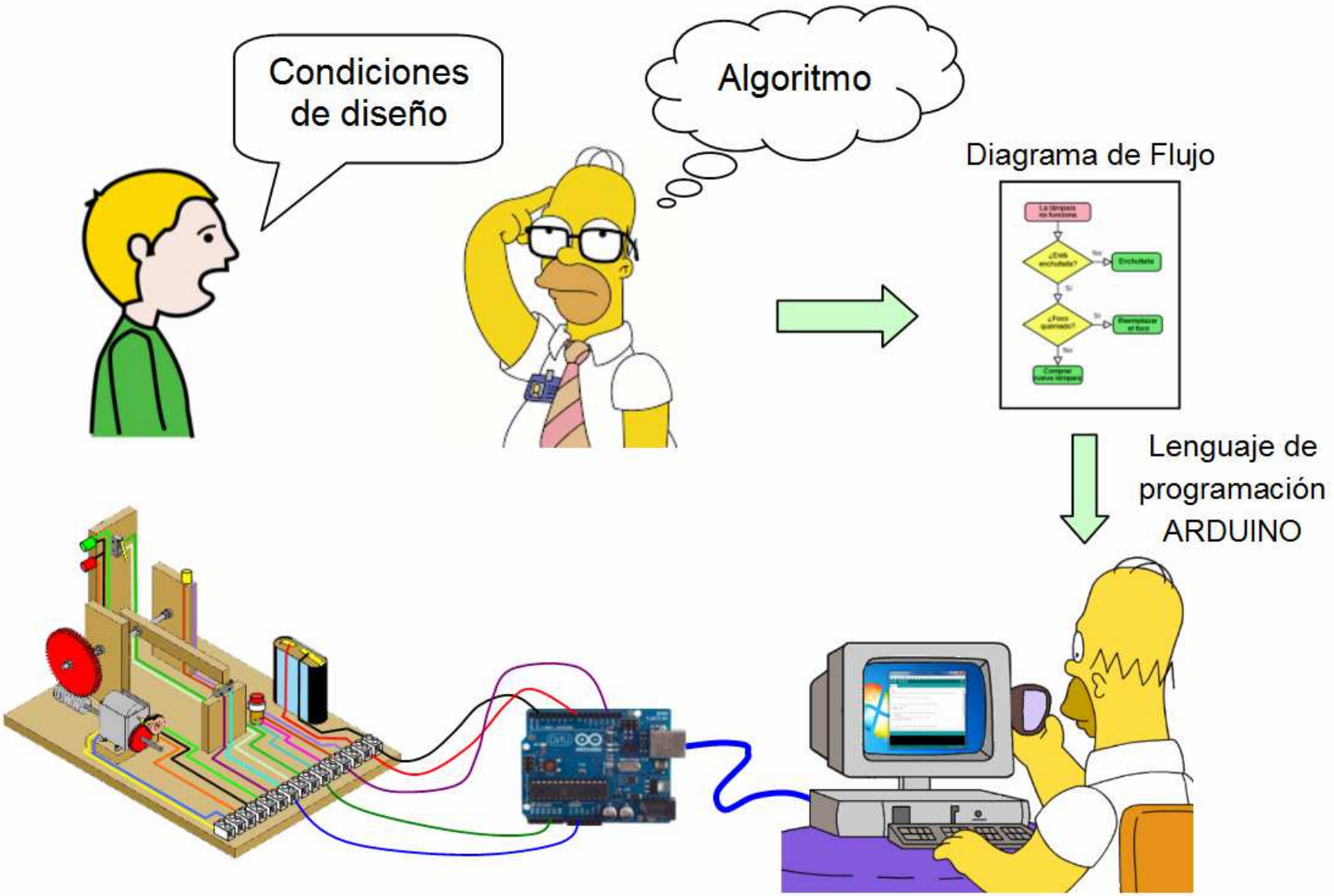
Una solución es conectar las salidas del microcontrolador a las bases de **transistores** en cuyos colectores se conectan los actuadores o los relés que los accionan.

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



Algoritmo

Un **algoritmo** es un conjunto de instrucciones ordenadas que permiten realizar una actividad, como, por ejemplo, resolver un problema, mediante pasos sucesivos.

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

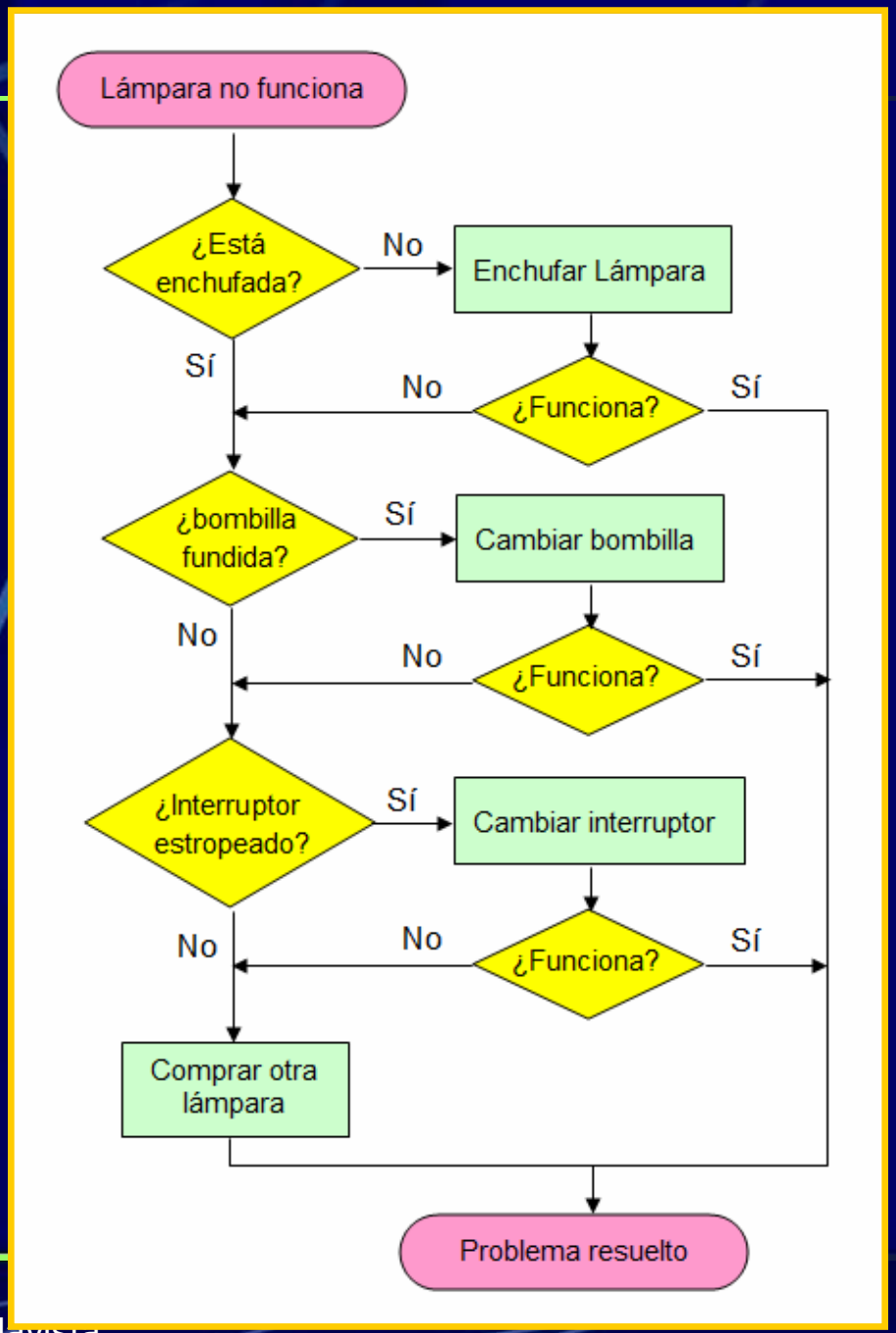
No: continuar. Ir a 4

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 compresión y su traducción a un lenguaje de programación.



Traducción Algoritmo-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:

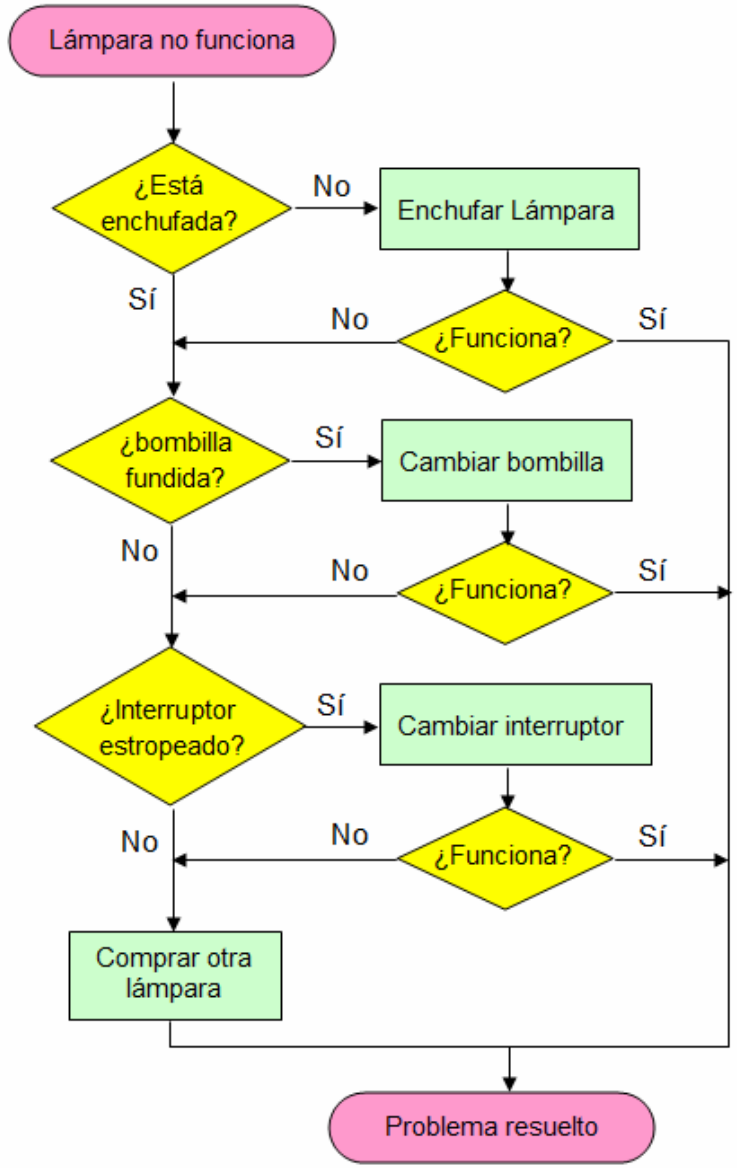
Sí funciona: problema resuelto. Ir a 5

No: continuar. Ir a 4

No: continuar. Ir a 4

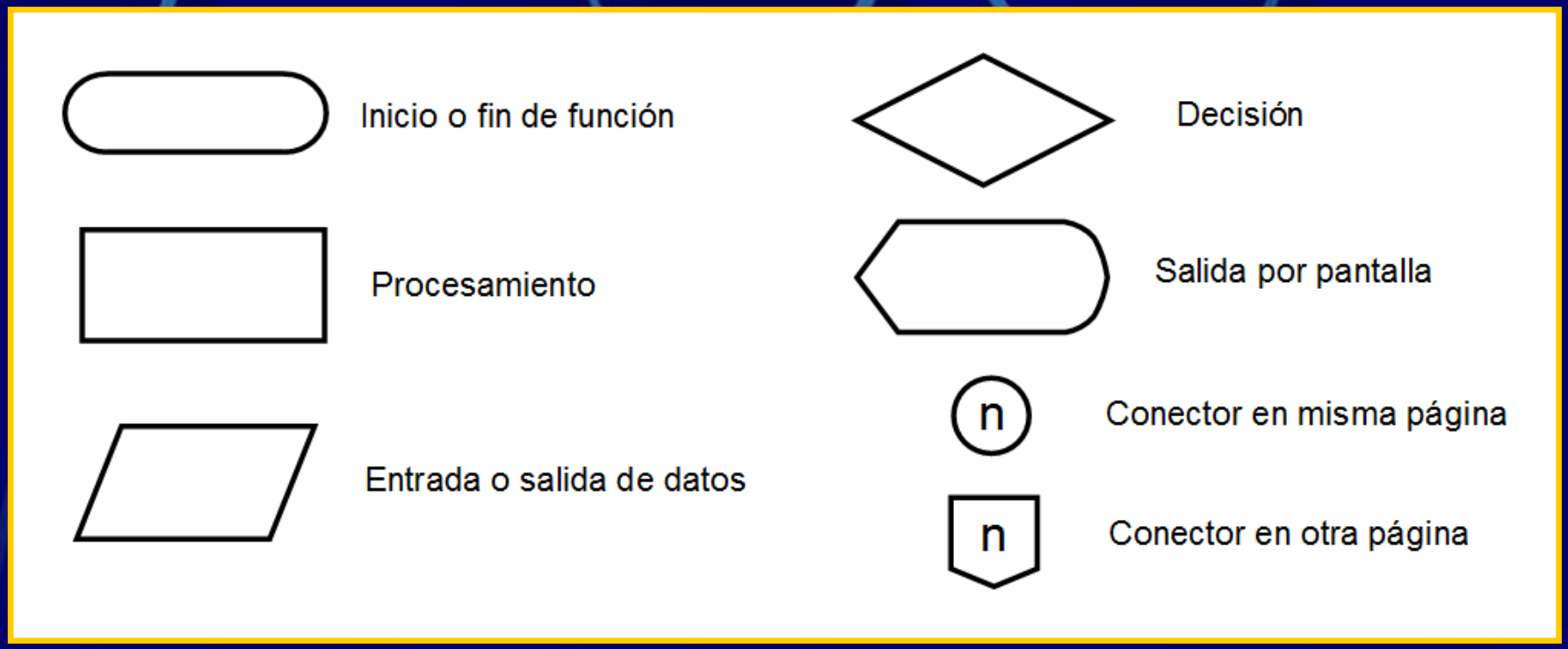
4.- Comprar una lámpara nueva. Ir a 5

5.- Problema resuelto. Fin



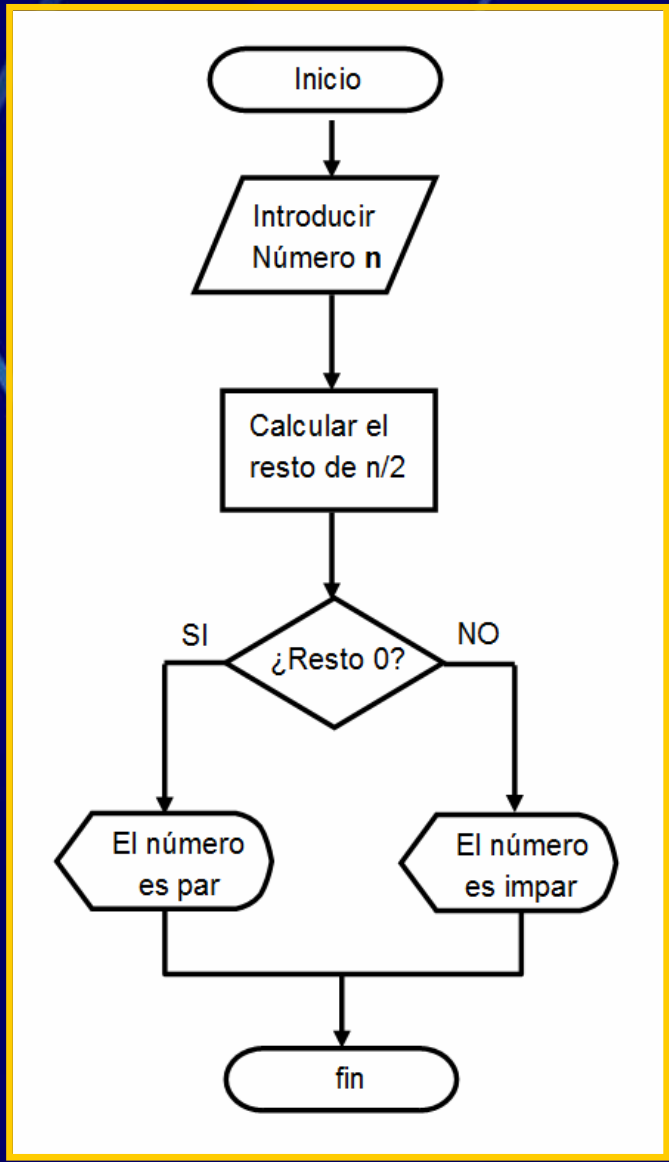
Algoritmo y diagrama de flujo

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



Algoritmo y diagrama de flujo: Ejemplo 1

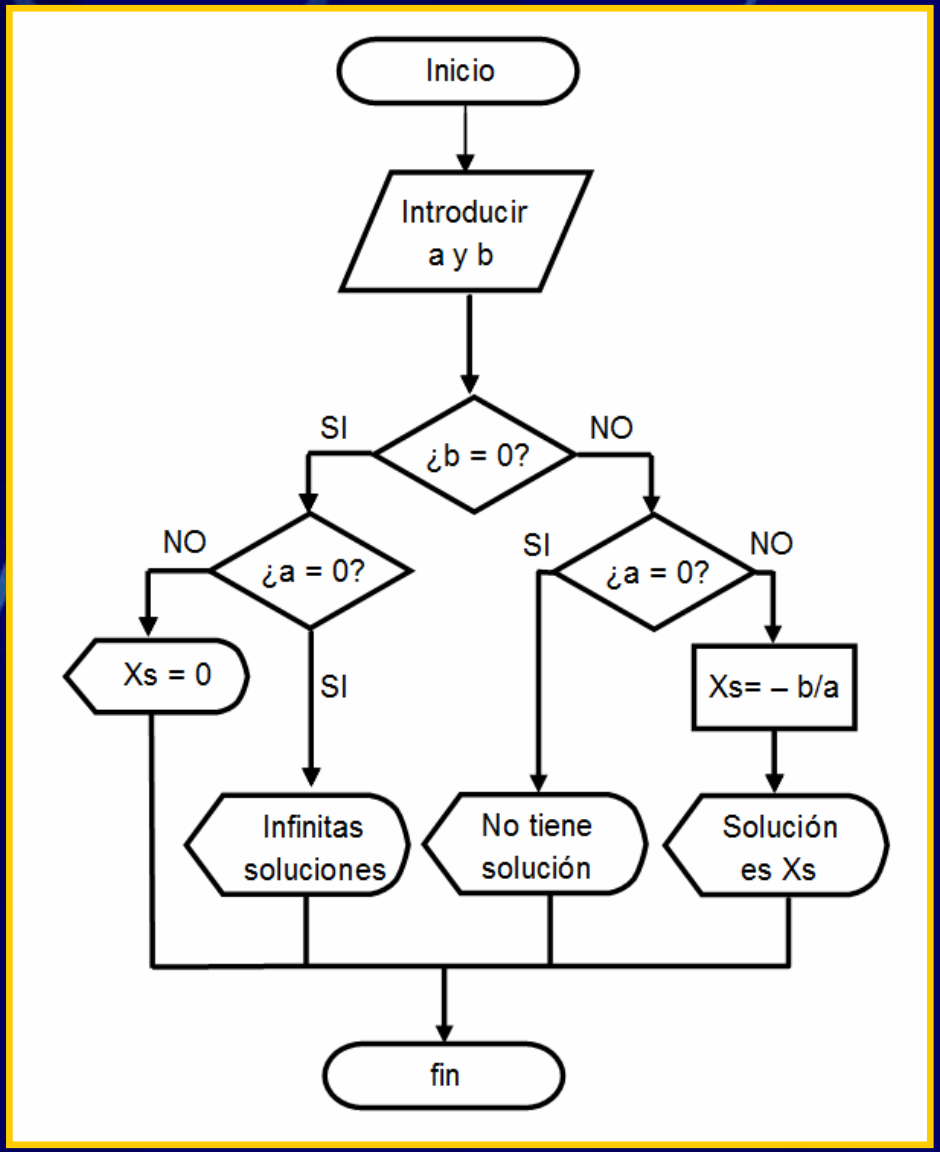
En la figura se representa el diagrama de flujo de un algoritmo para determinar si un número que se le introduce es par o impar.



Algoritmo y diagrama de flujo: Ejemplo 2

En la figura se representa el diagrama de flujo de un algoritmo que nos resuelve ecuaciones de primer grado del tipo **a x + b = 0**

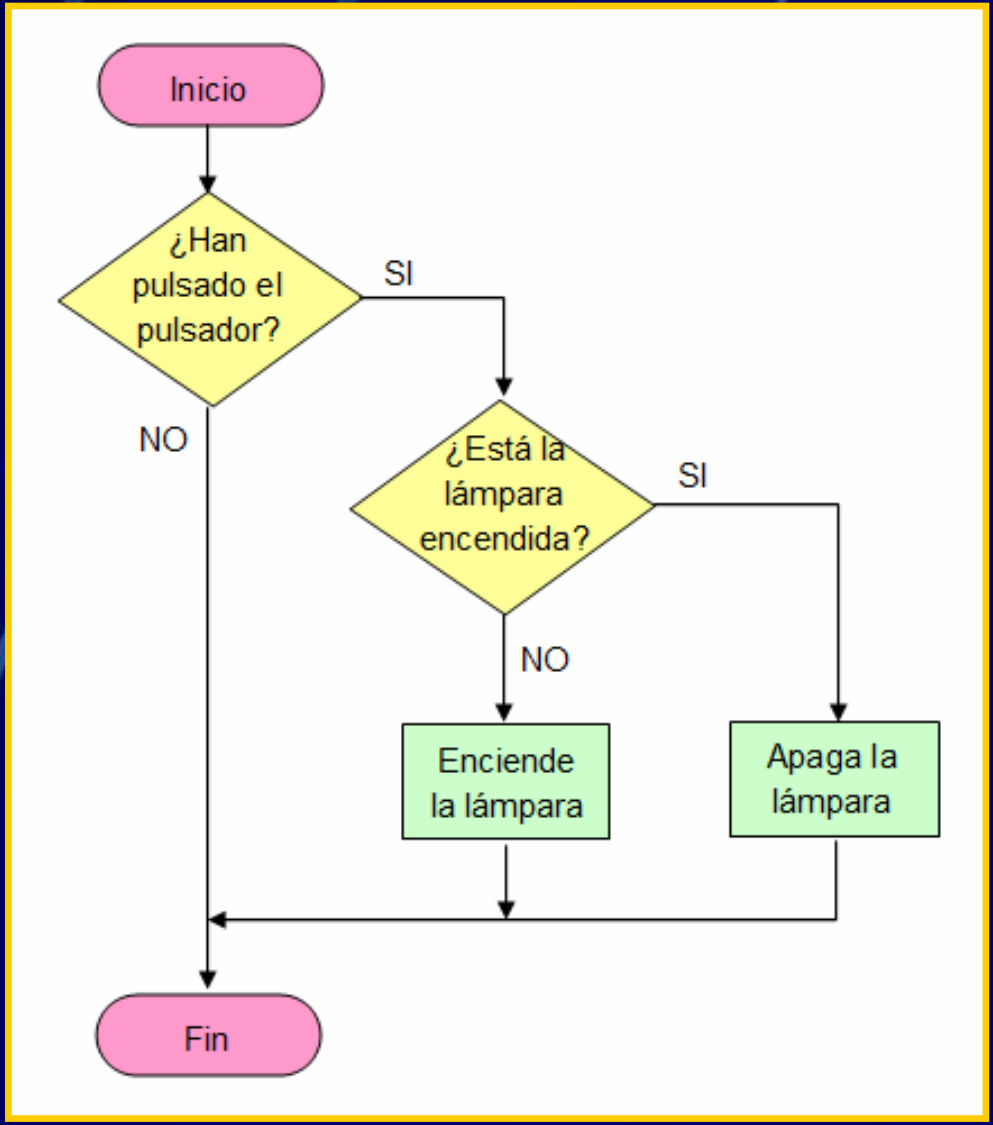
Denotamos la solución por **Xs**



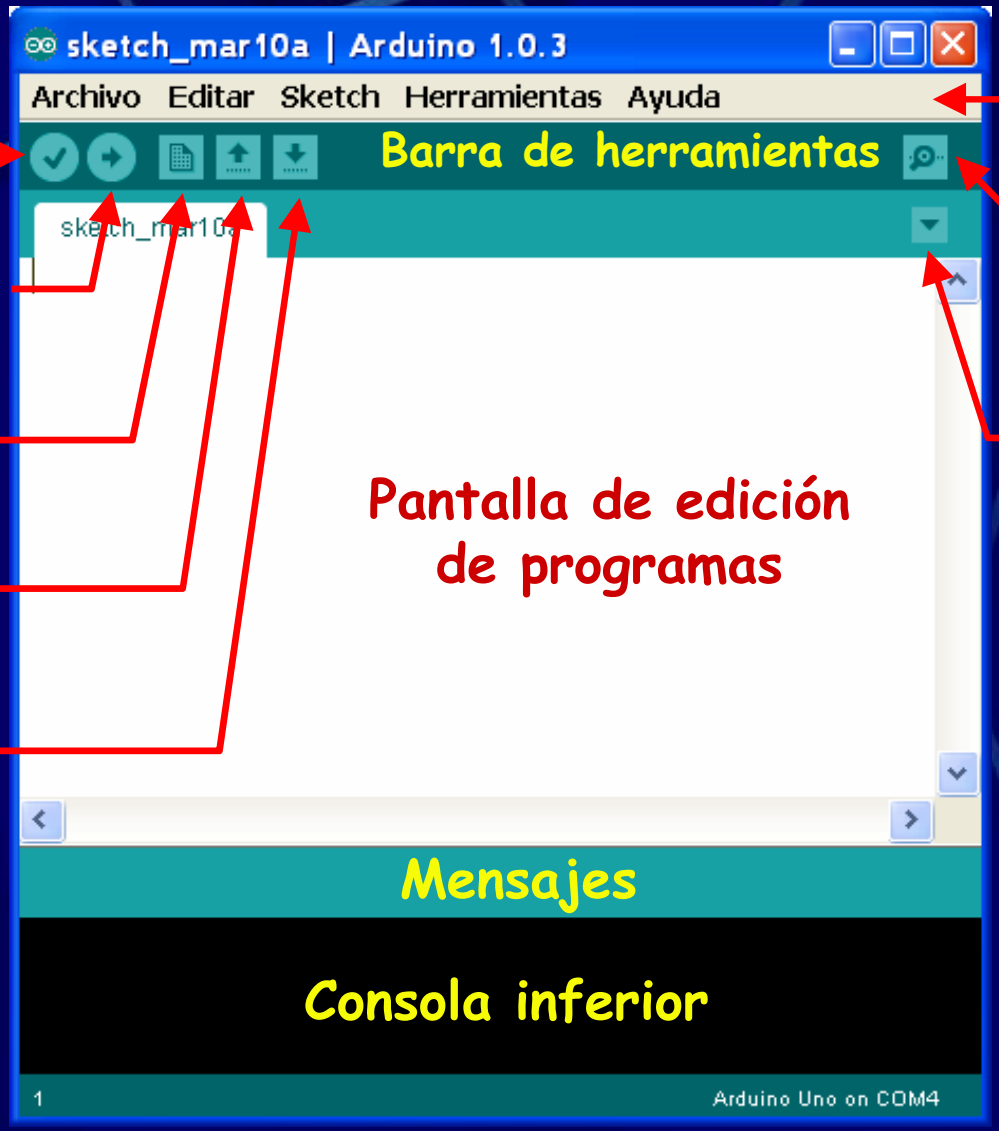
Algoritmo y diagrama de flujo: Ejemplo 3

En la figura se representa el diagrama de flujo de un algoritmo para cambiar del estado de encendida al de apagada o viceversa de una lámpara al pulsar un pulsador.

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).



El entorno integrado de ARDUINO



Verifica código

Carga en la placa

Nuevo archivo

Abrir archivo

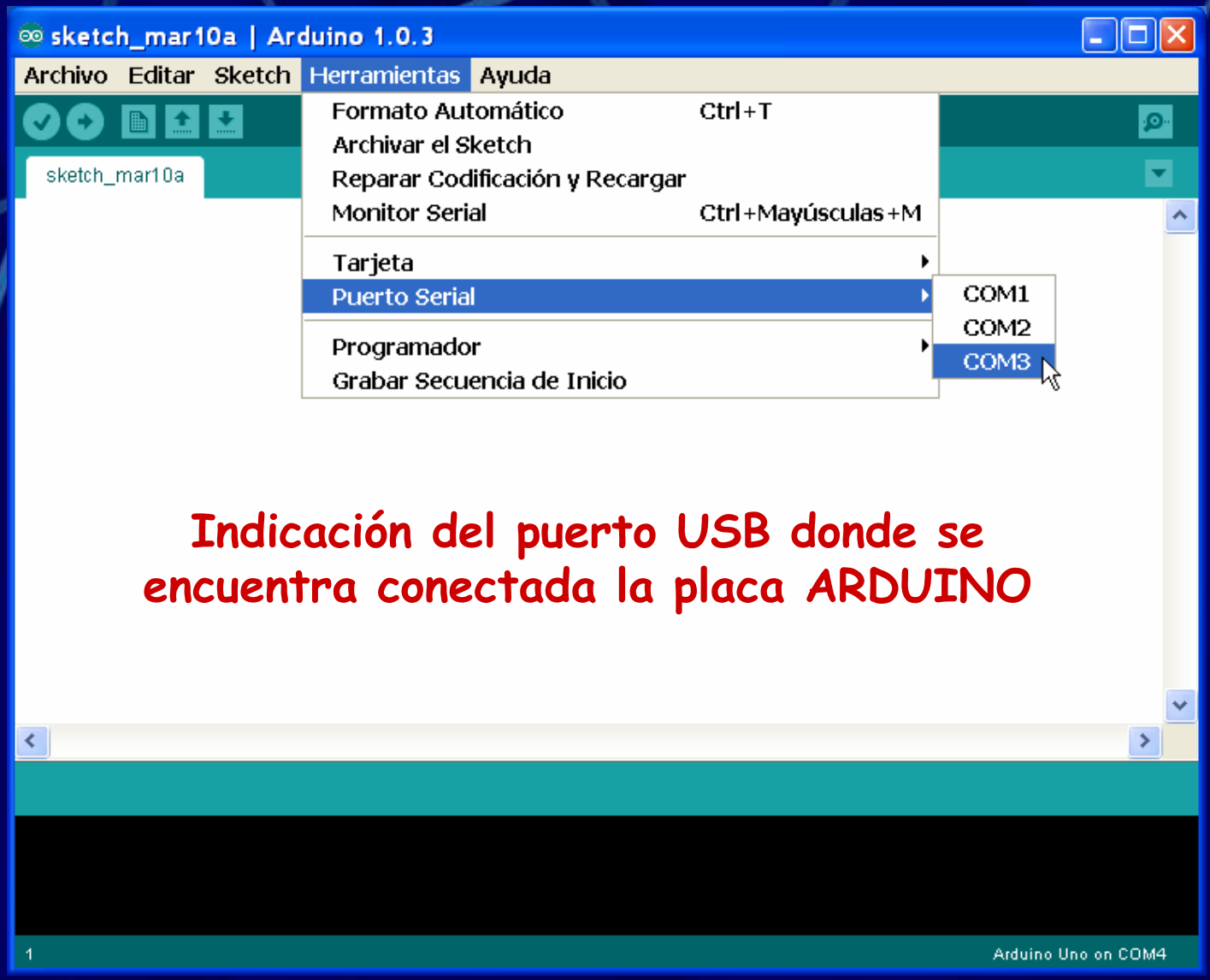
Guardar archivo

Barra de menús

Abre monitor de transmisión serie

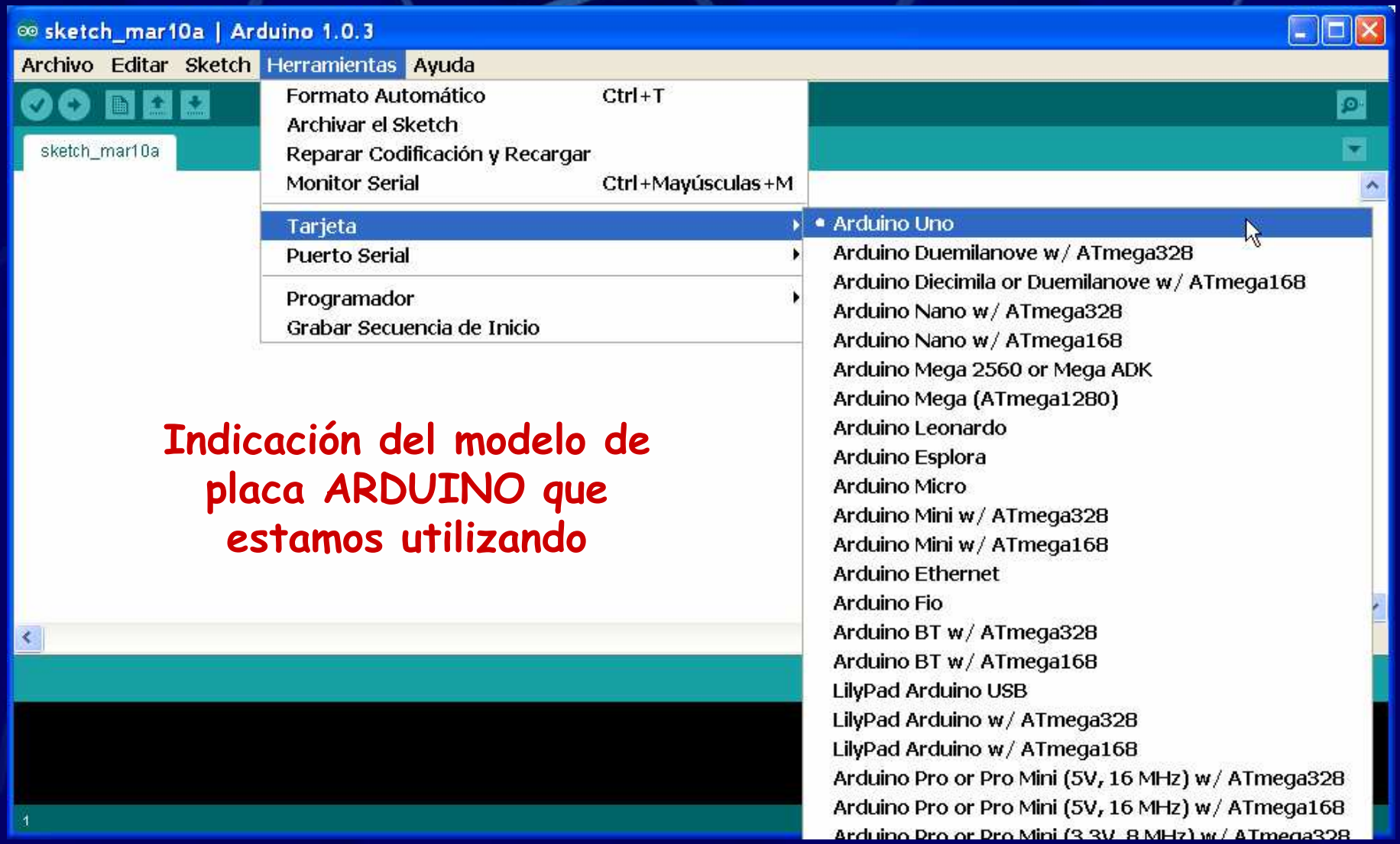
Gestor de pestañas

Indicación del puerto USB de la placa ARDUINO



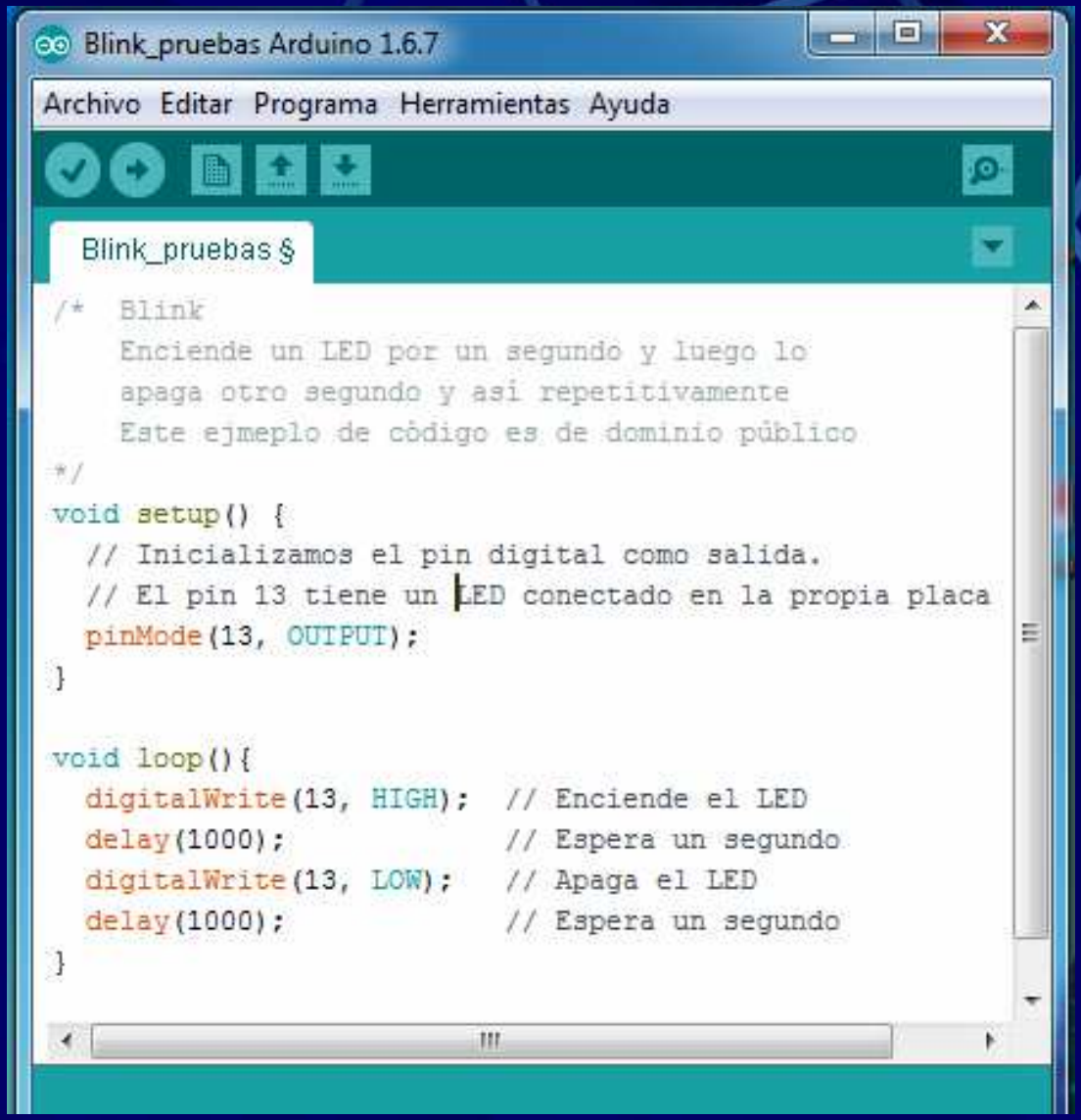
Indicación del puerto USB donde se encuentra conectada la placa ARDUINO

Indicación del modelo de placa ARDUINO



Indicación del modelo de placa ARDUINO que estamos utilizando

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



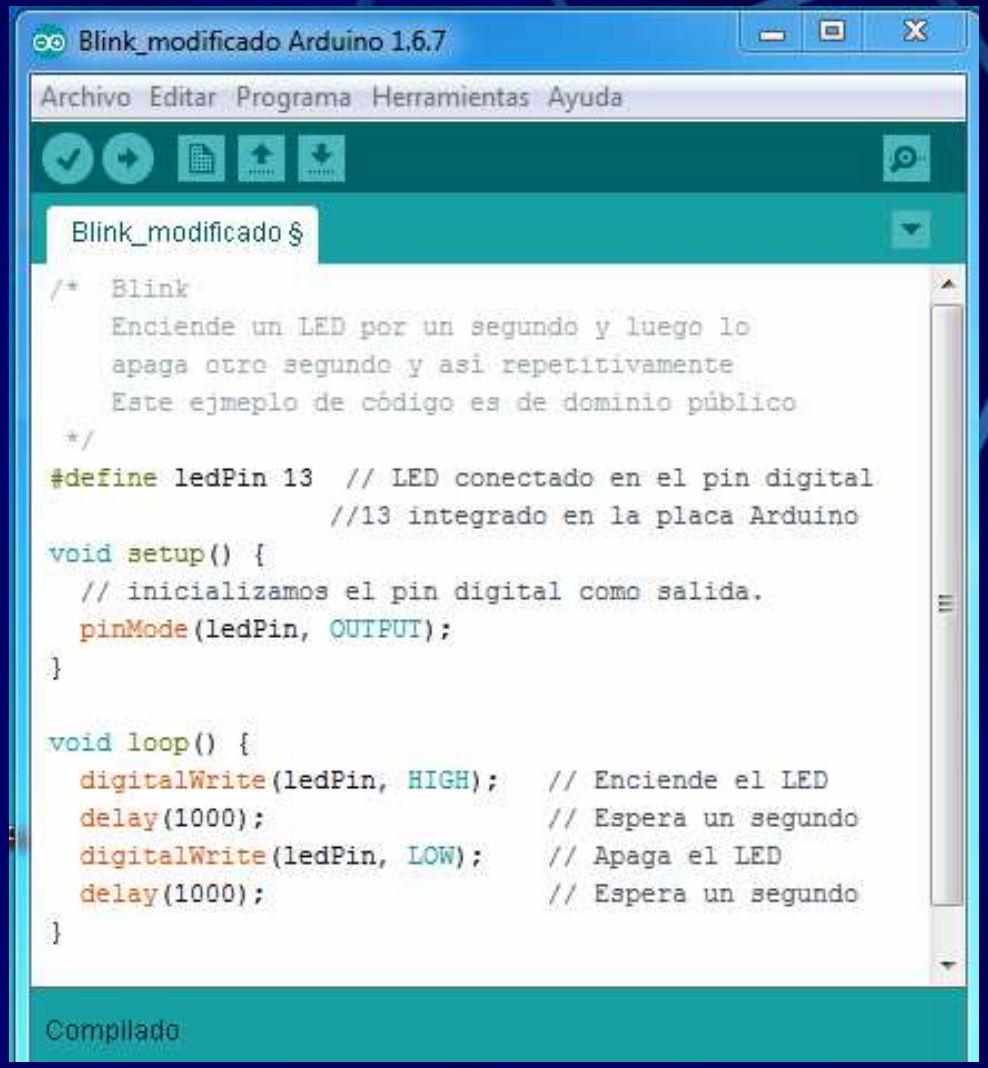
Todos los programa deben contener como mínimo las funciones **setup** y **loop**.

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: Elementos de sintaxis



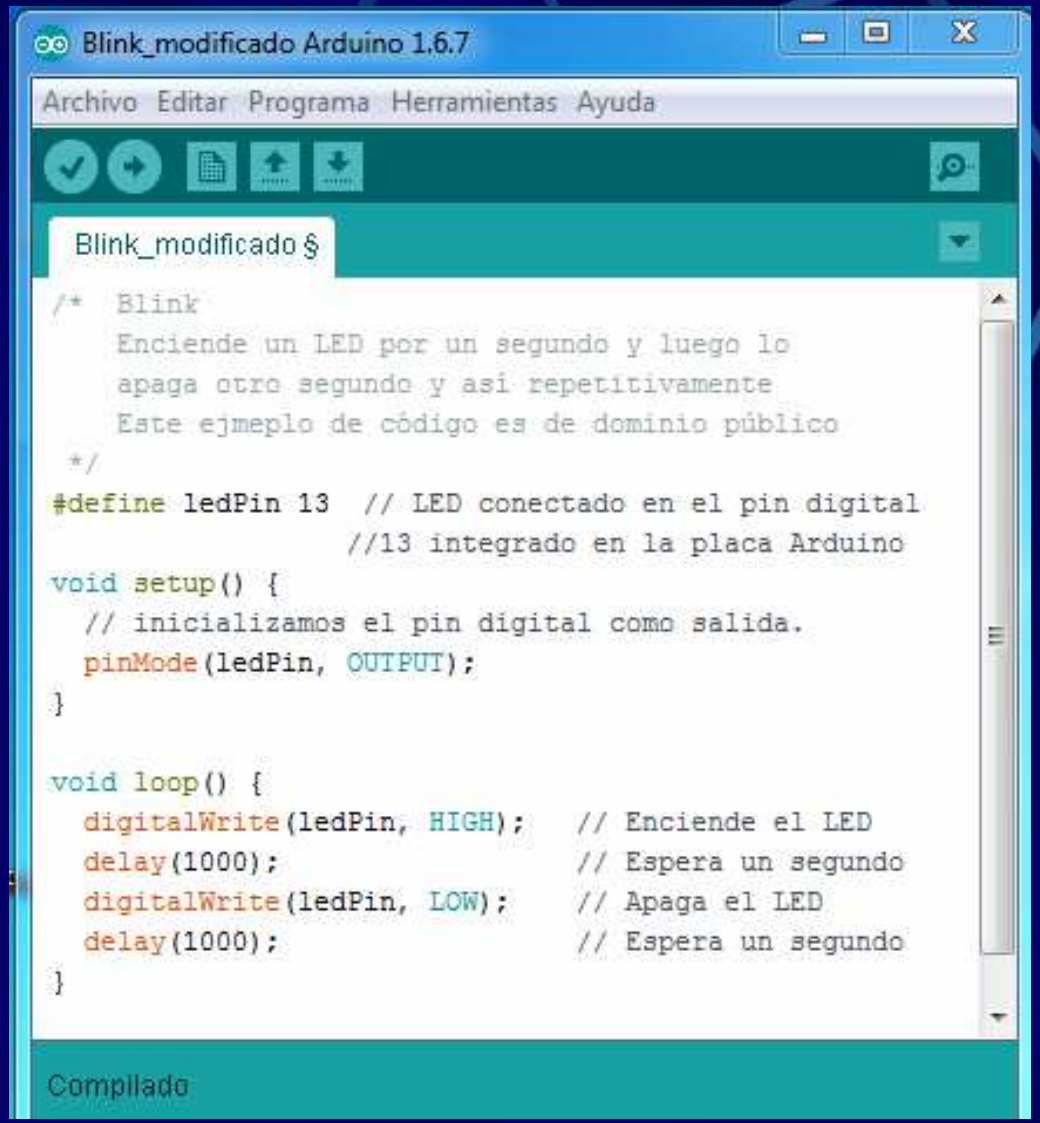
Los principales **elementos de sintaxis** son el punto y coma (;) y las llaves ({ }).

Toda instrucción debe ir seguida de un "**punto y coma**" (;). Pueden ir varias instrucciones en un mismo renglón, 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.

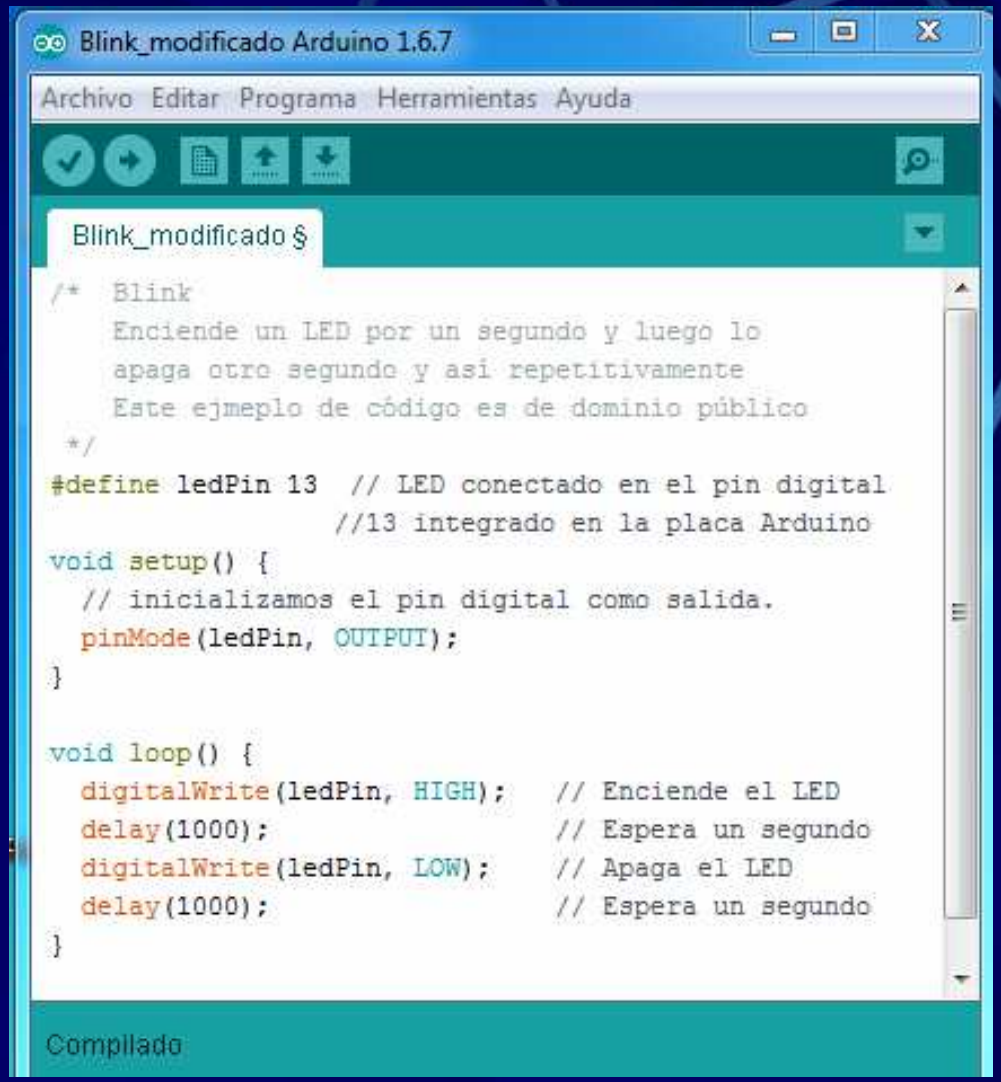
Estos nombres se llaman constantes, y se definen utilizando la intrucción

`#define` const número

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

Todas las constantes deben declararse antes de usarse.

Programación en ARDUINO: Entradas / Salidas digitales

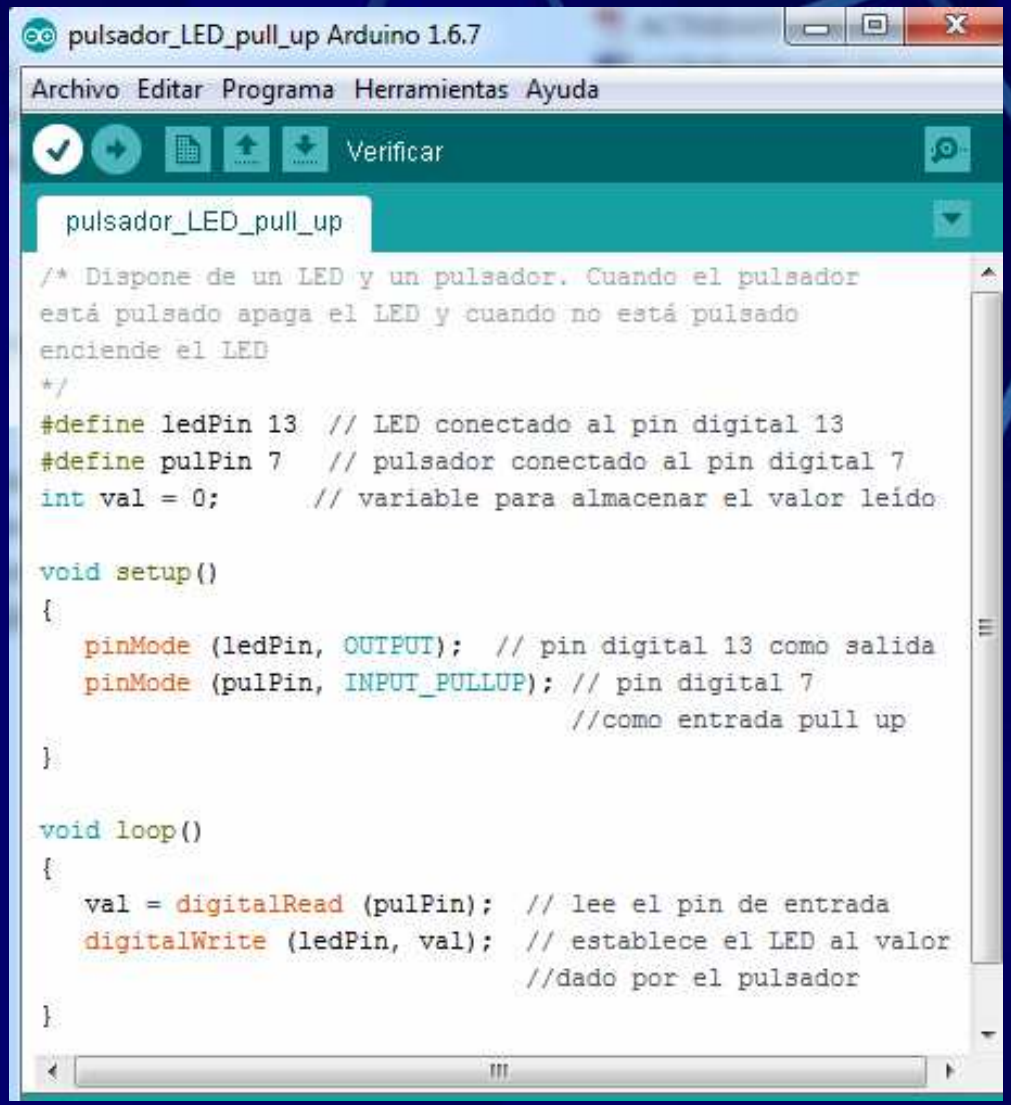


- Los pines digitales de Arduino pueden funcionar tanto como entradas como salidas. El modo hay que declararlo previamente con la instrucción:
  - **pinMode** (pin, modo)

  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: Resistencias pull up



- Para no tener que utilizar resistencias externas en las entradas de Arduino para garantizar que en todo momento presenten un HIGH o un LOW bien definidos, podemos activar las **resistencias internas pull up** (conectadas a fuente) de las entradas, que incorpora Arduino, con la función pinMode(). El modo es el siguiente:

- **pinMode** (pin, INPUT_PULLUP)

El parámetro 'pin' indica el pin para el que se activa la resistencia pull up.

Programación en ARDUINO: Resistencias pull up

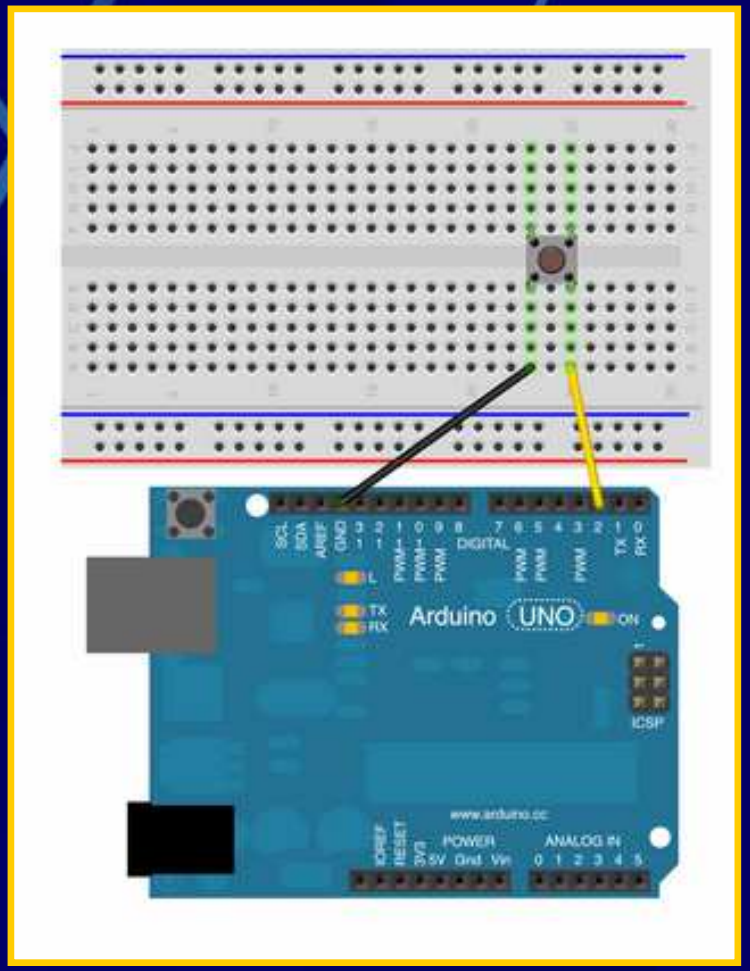
pulsador_LED_pull_up Arduino 1.6.7

Archivo Editar Programa Herramientas Ayuda

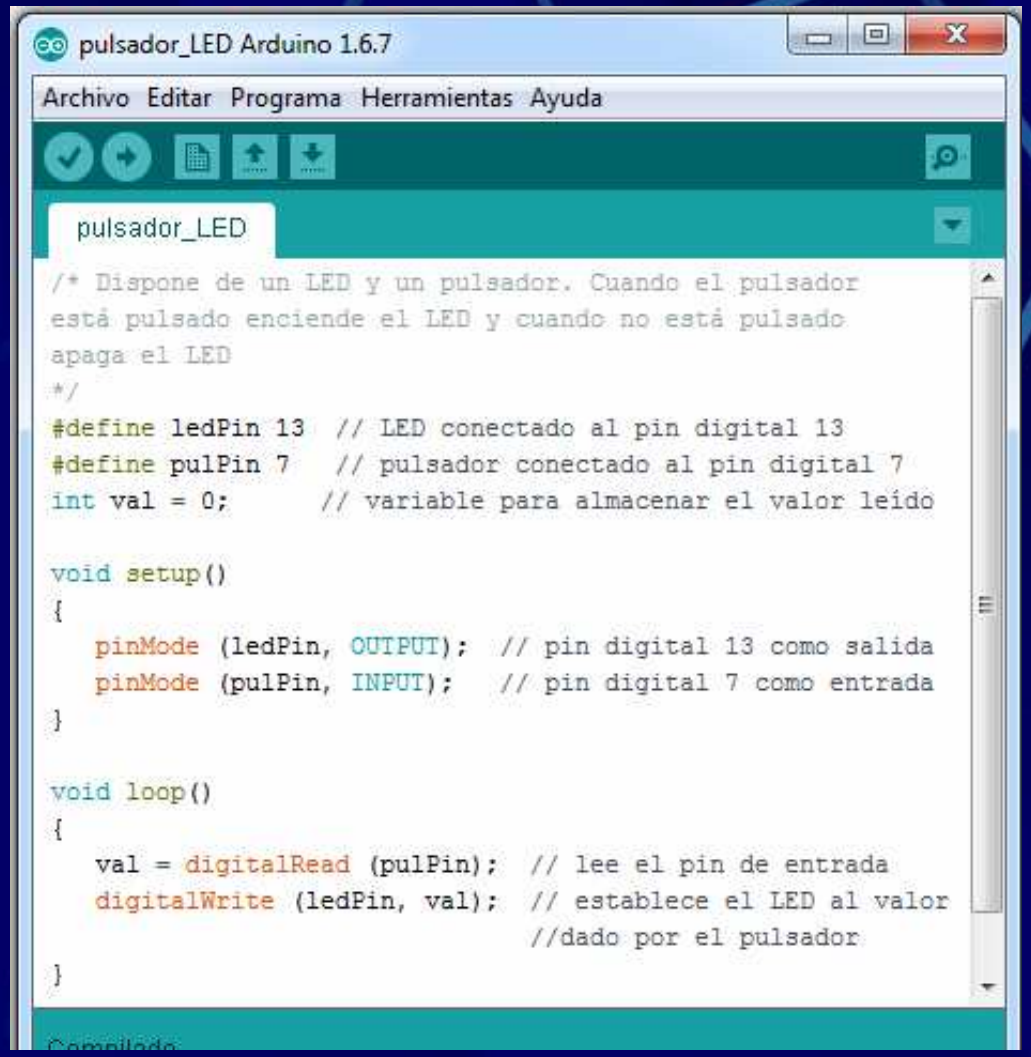
Verificar

pulsador_LED_pull_up

```
/* Dispone de un LED y un pulsador. Cuando el pulsador
está pulsado apaga el LED y cuando no está pulsado
enciende 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 leído

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

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



Programación en ARDUINO: Variables globales y locales

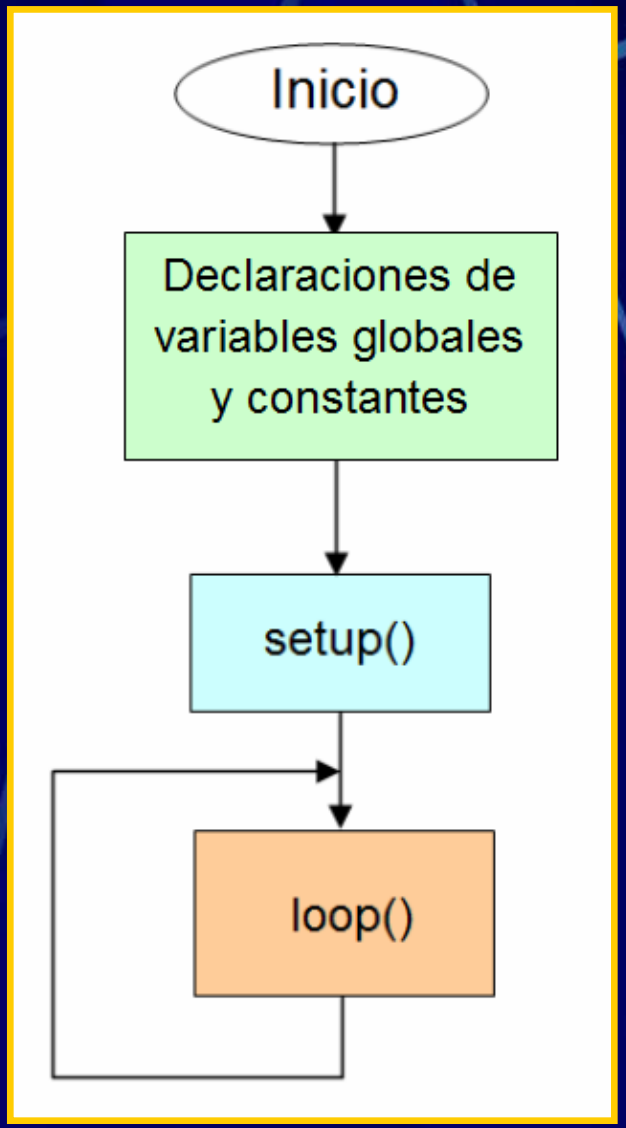


Una **variable** es un modo de nombrar y guardar un valor que puede variar para su uso posterior por el programa.

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.

Esquema de funcionamiento de Arduino

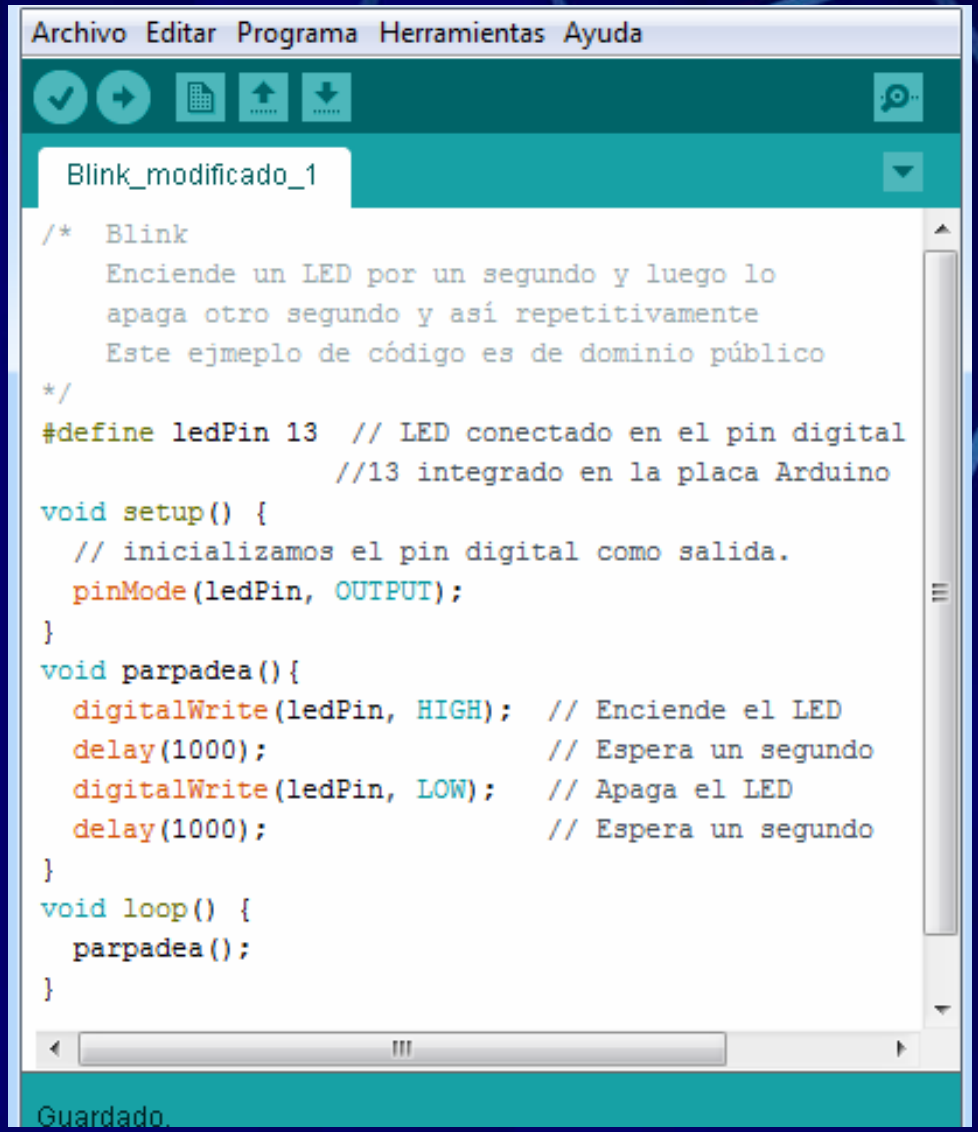


Las declaraciones de variables globales y de las constantes suelen colocarse delante de la función setup()

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.

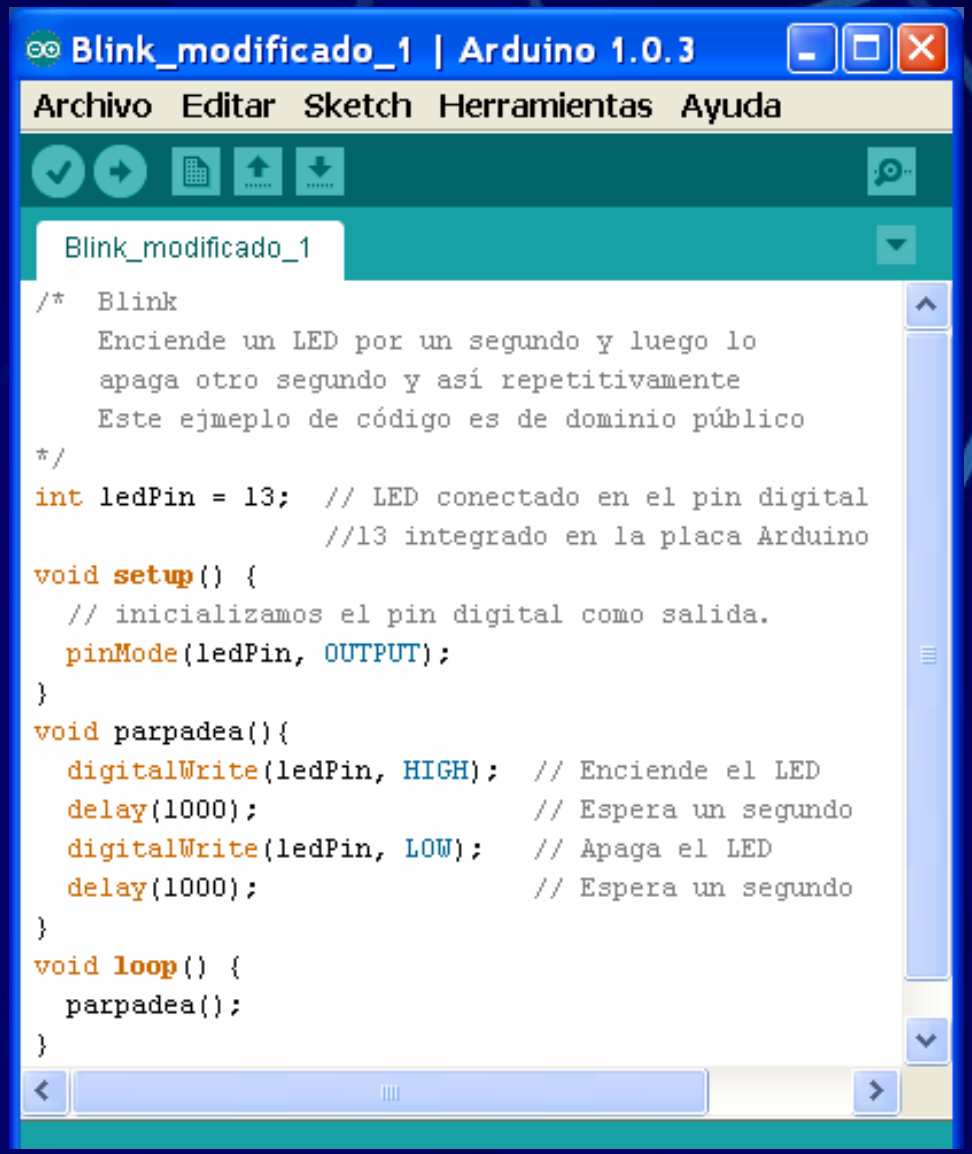
A continuación se ejecuta la función loop() de forma cíclica hasta que se corta la alimentación o se presiona el botón reset de la placa.

Programación en ARDUINO: Funciones del usuario



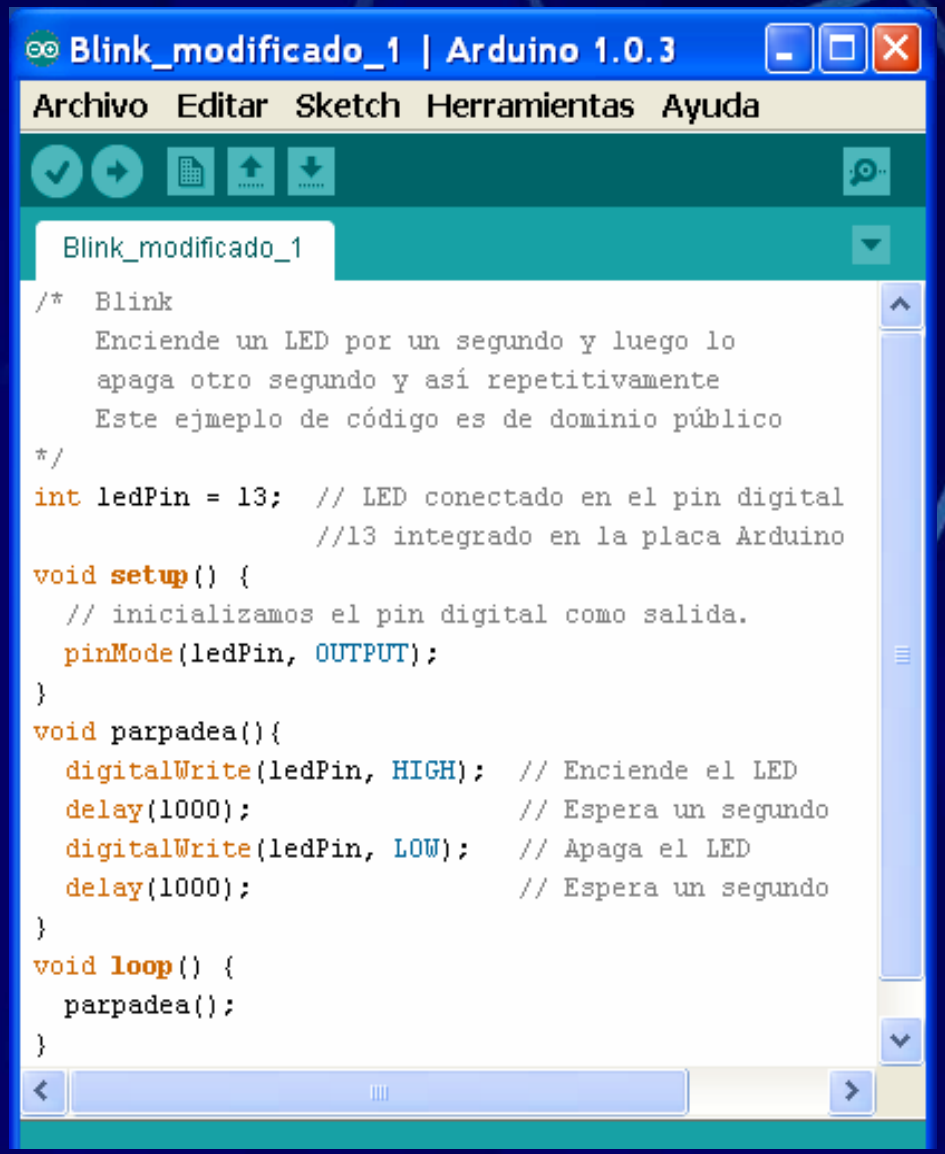
Funciones definidas por el usuario, éstas pueden definirse en cualquier parte del programa o incluso en una pestaña separada.

Programación en ARDUINO: Los nombres válidos



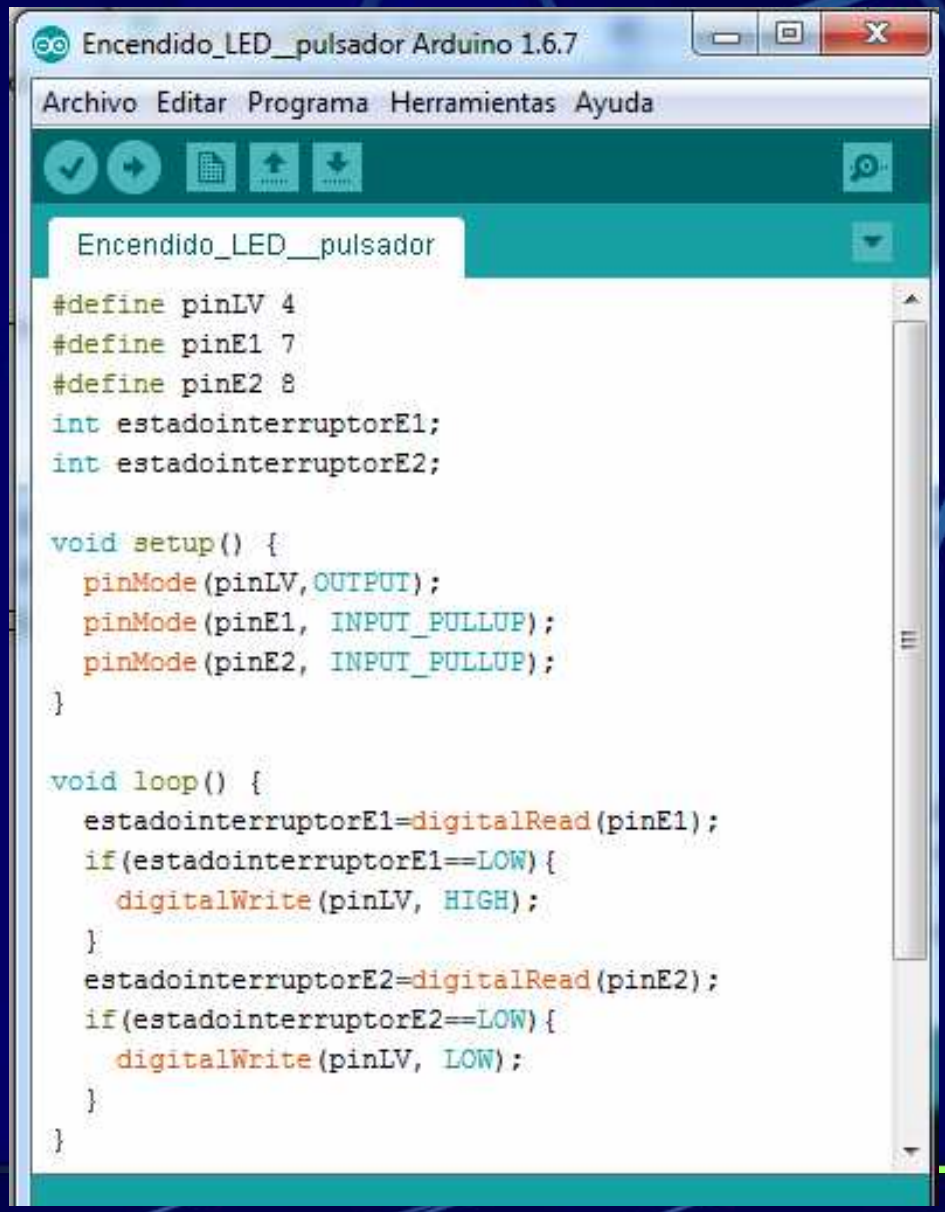
- Los nombres de los sketch, de las variables, de las constantes y de las funciones no pueden contener espacios, (puede usarse el guión bajo).
- Los nombres dados a variables, constantes y funciones no deben coincidir con palabras clave de Arduino.
- Se distinguen mayúsculas de minúsculas.
- Hay una serie de constantes con nombres reservados:
  - false y true.
  - INPUT y OUTPUT.
  - HIGH y LOW.

Programación en ARDUINO: Los tipos de datos



- 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.

Programación en ARDUINO: Estructura condicional if



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

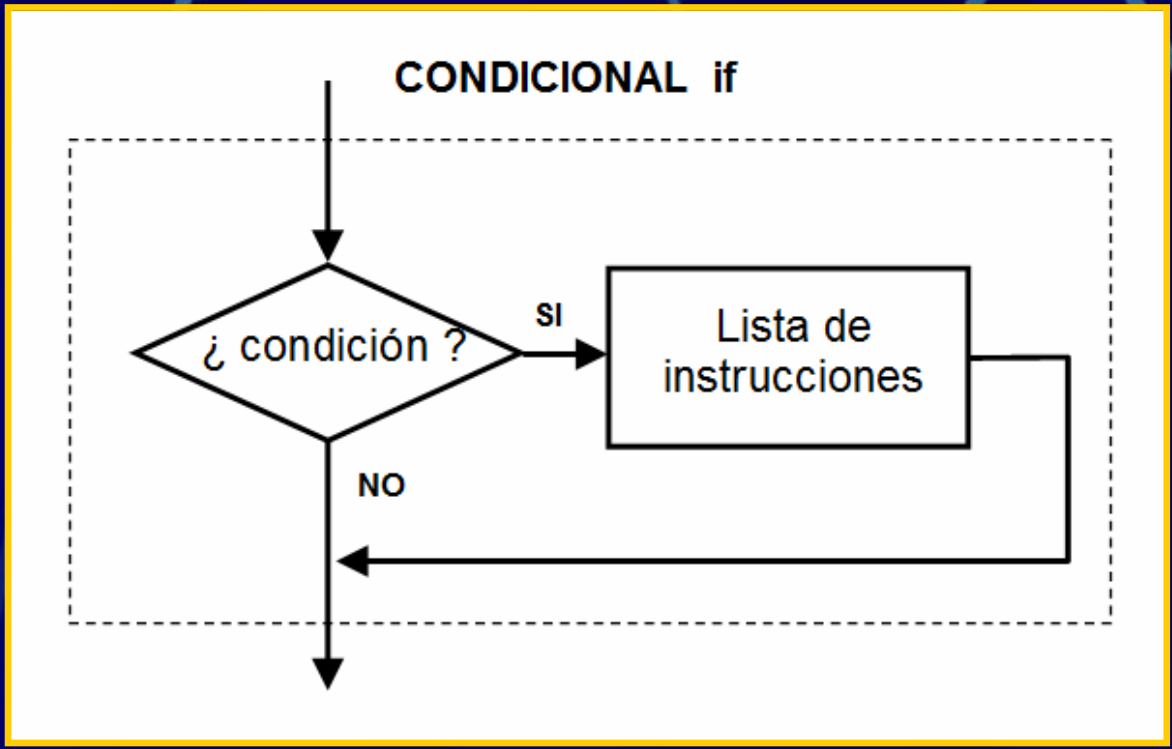
```
if (condición) instrucción;
```

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

- Para expresar la condición se utilizan operadores:
  - de comparación: ==, !=, <, >, <=, >=
  - booleanos: &&, ||, !

Diagrama de flujo de la estructura condicional if

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



Programación en ARDUINO: Estructura condicional if...else

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

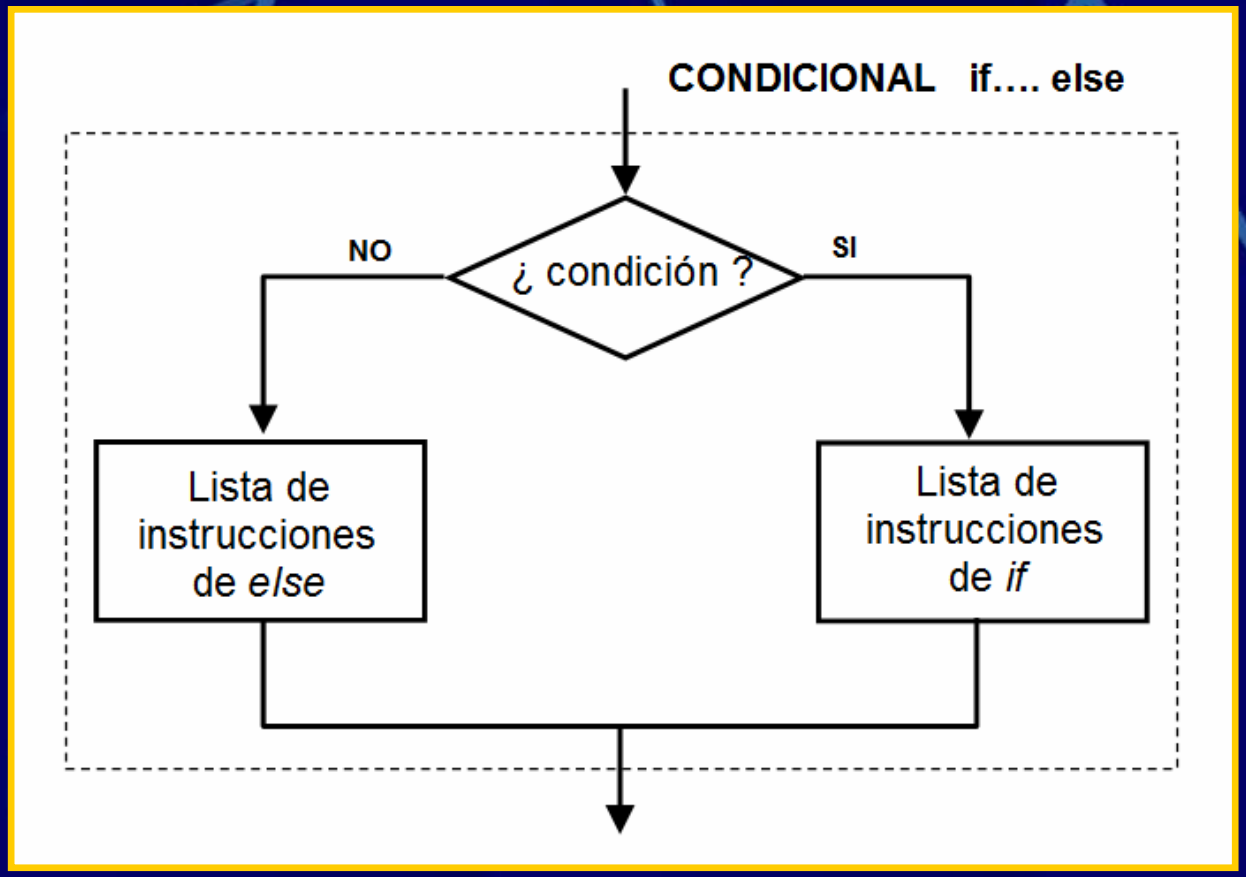
```
if (condición) instrucción_A;
else instrucción_B;
```

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

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

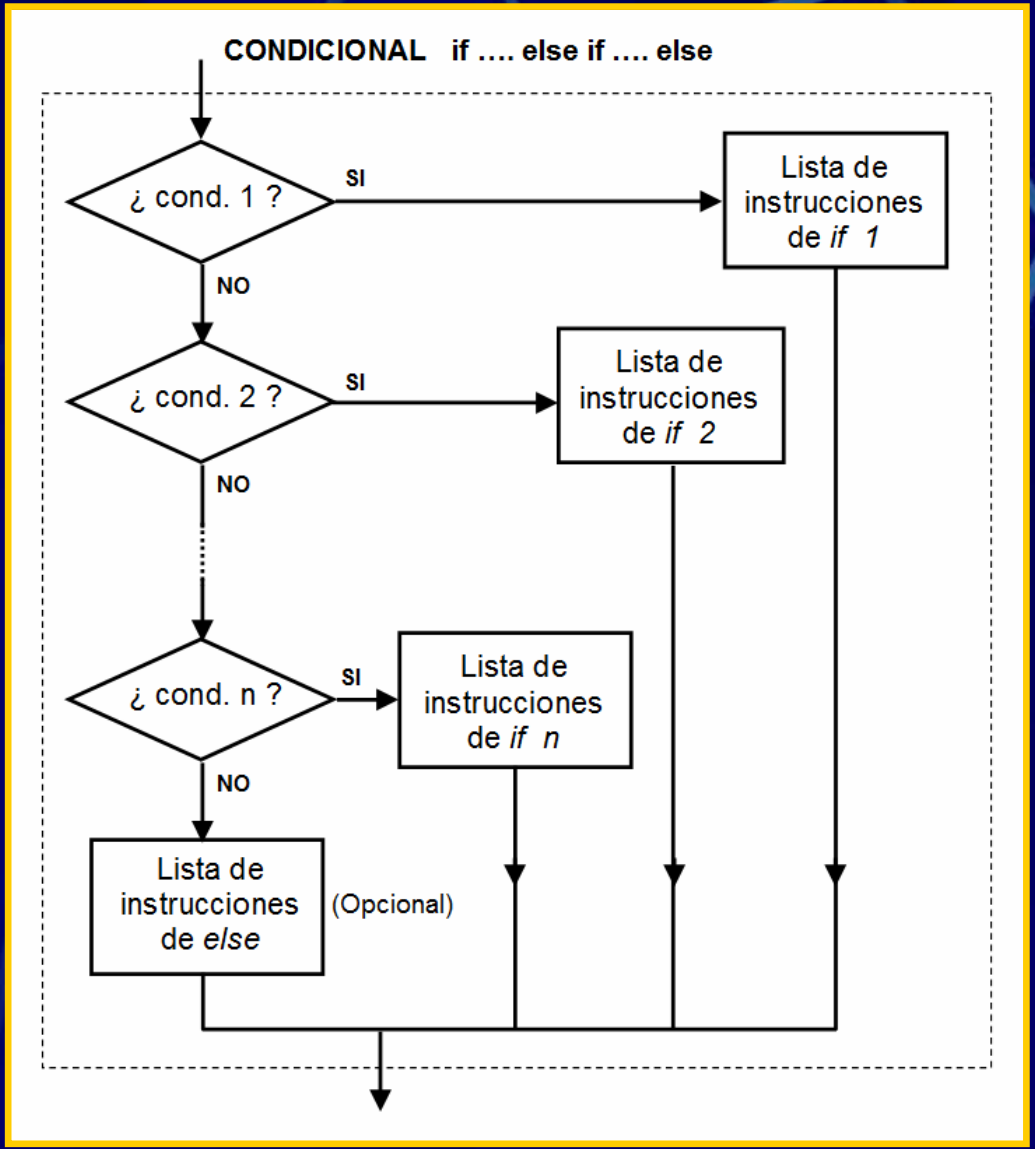
```
if (condición1) instrucción_A;
else if (condición2) instrucción_B;
else instrucción_C;
```

Diagrama de flujo de la estructura condicional if….else



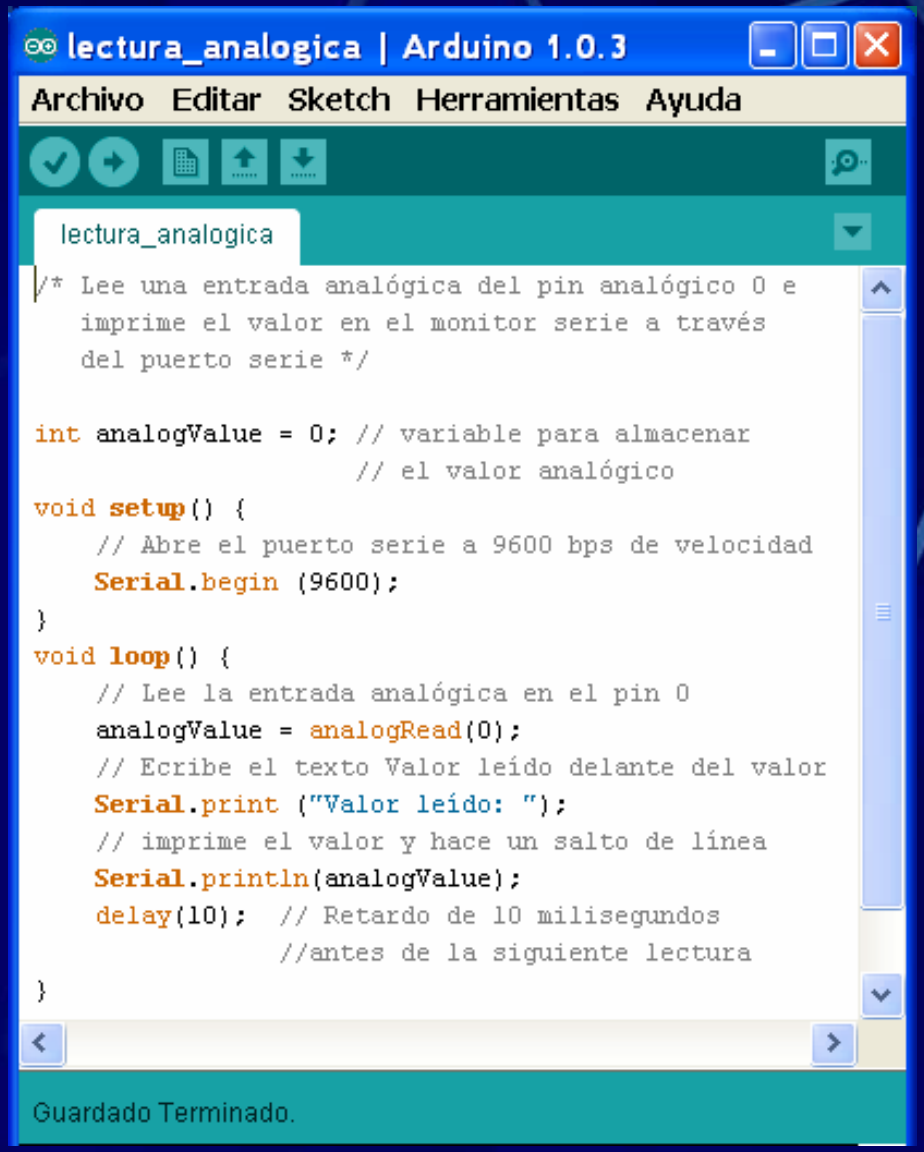

```
if (condición) {
    instrucciones_A; }
else {
    instrucciones_B; }
```

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; }
```

Programación en ARDUINO: Comunicación serie



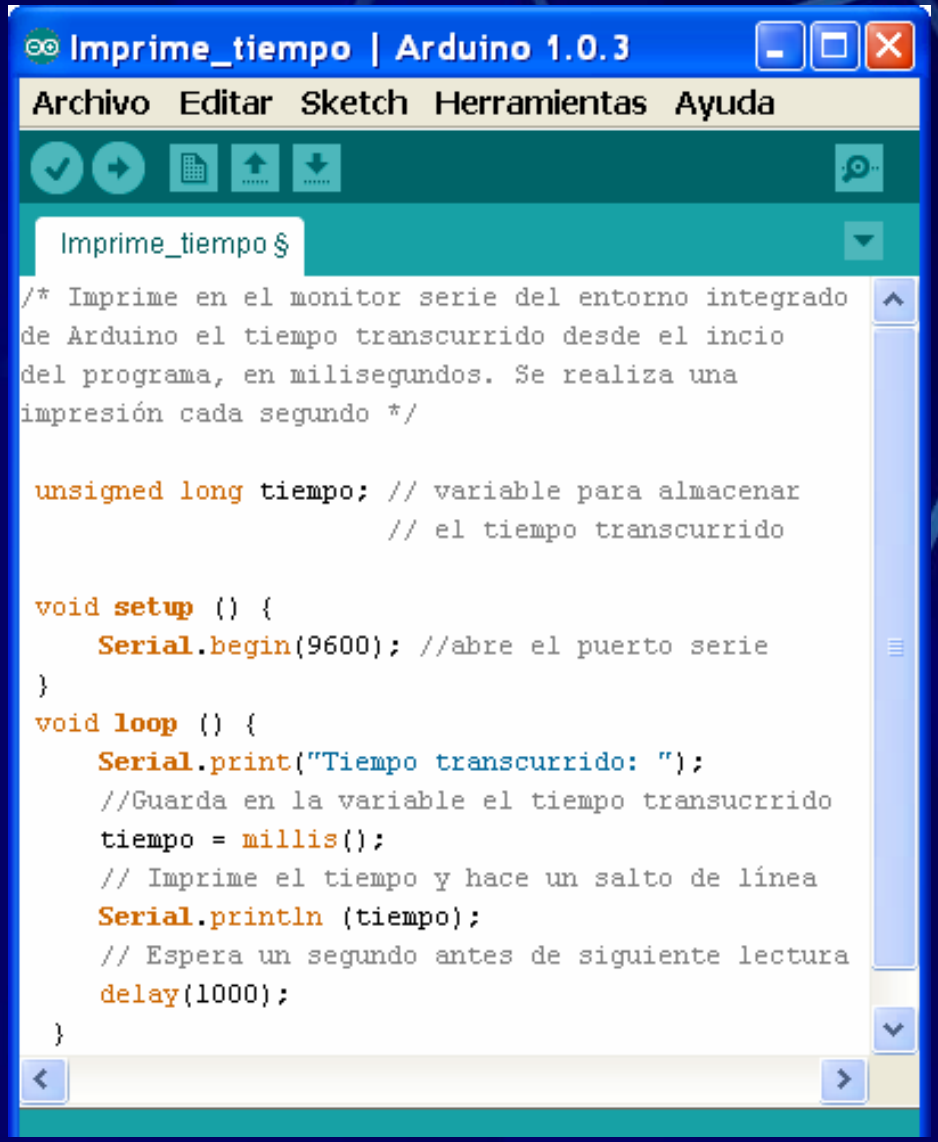
- La placa Arduino puede comunicarse en modo serie con el ordenador a través del puerto USB o con otro Arduino a través de los pines digitales 0 y 1.
- Podemos utilizar el monitor serial del IDE de Arduino, con las siguientes funciones:
  - **Serial.begin (valor)**

    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.

Programación en ARDUINO: Funciones de tiempo

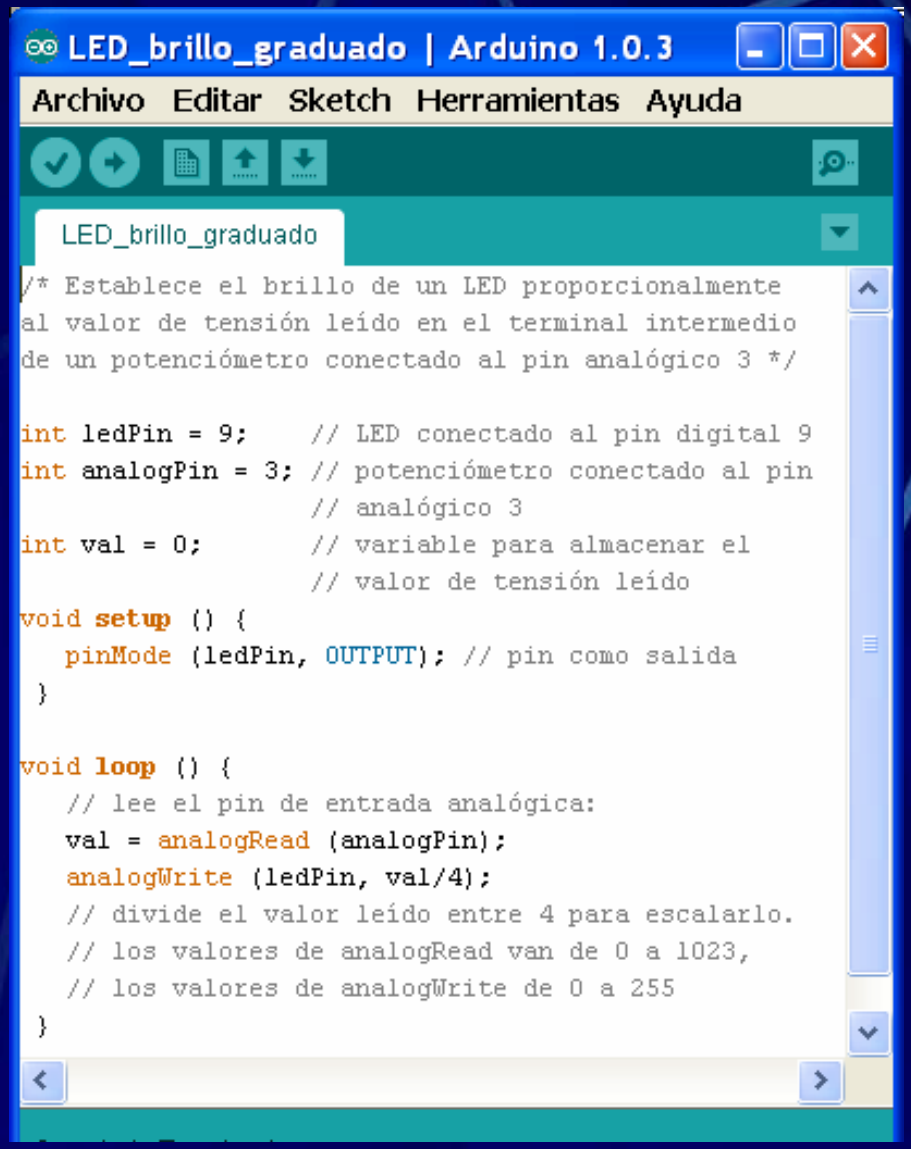


- Las funciones de tiempo permiten realizar temporizaciones en los programas.
  - **delay (valor)**

    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.
- **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.

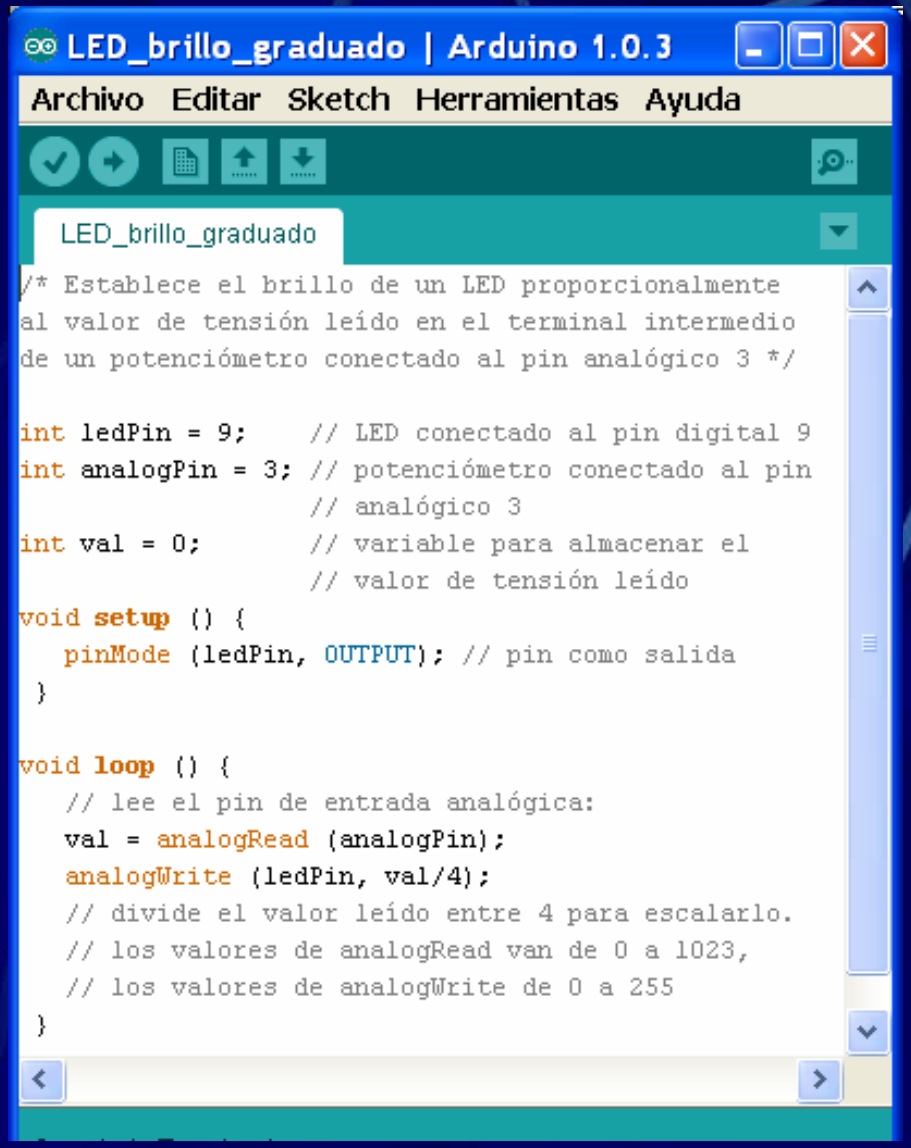
Programación en ARDUINO: Entradas analógicas



- 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.

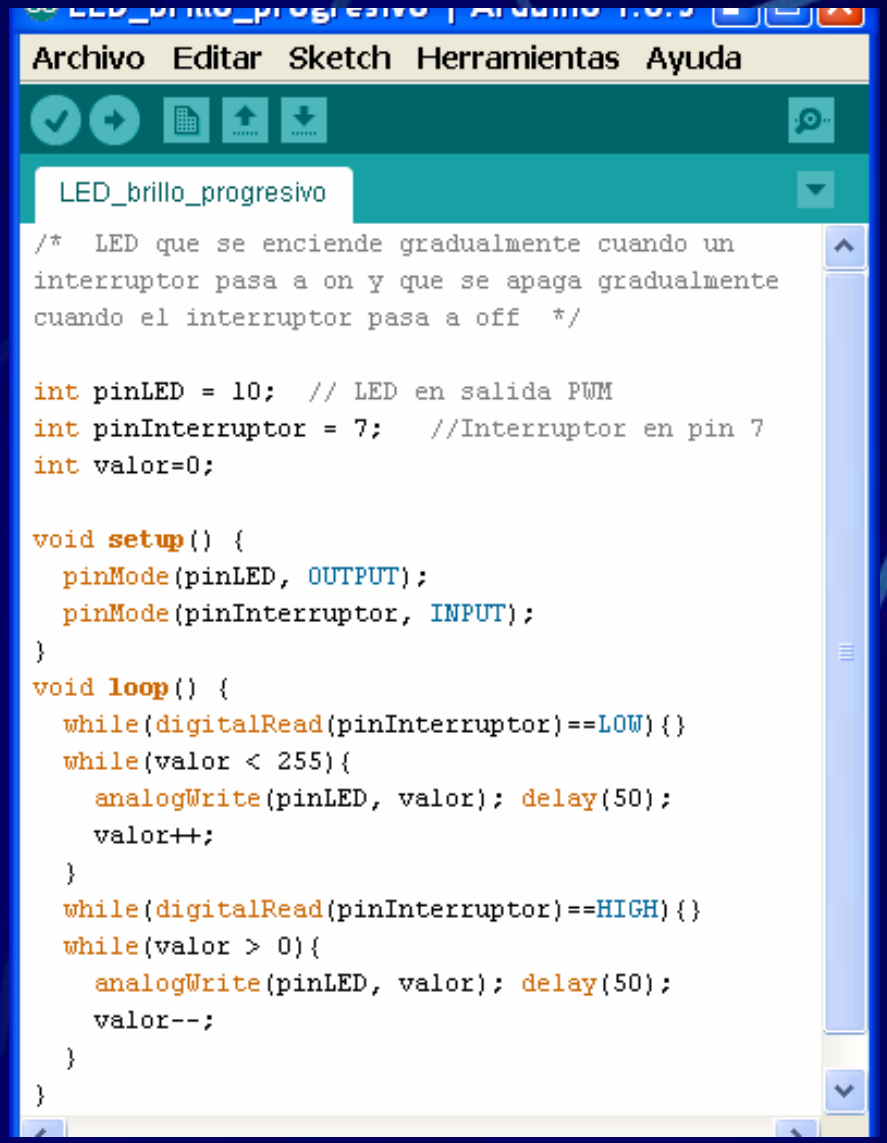
Programación en ARDUINO: Salidas analógicas



- 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.

Programación en ARDUINO: Bucle repetitivo while




```
/*  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--;
  }
}
```

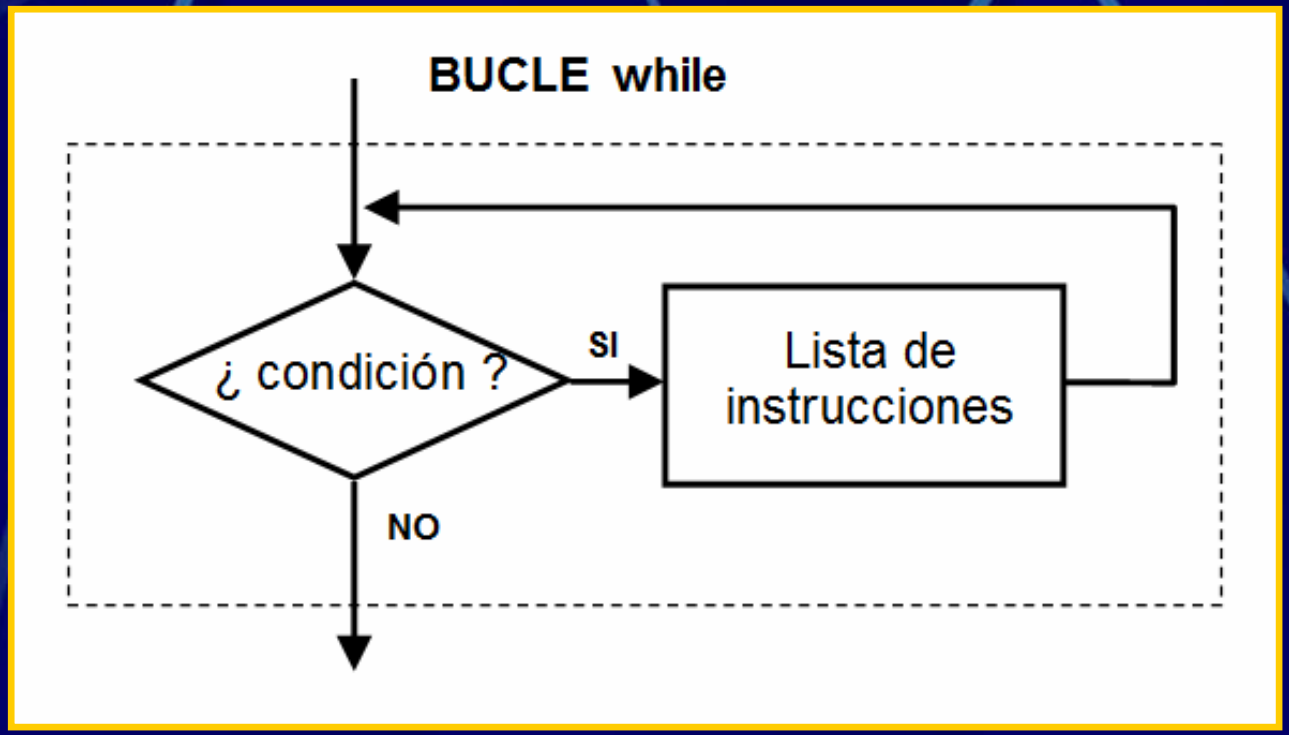
- El bucle **while** se repetirá indefinidamente hasta que la expresión booleana (condición) que sigue a la palabra while entre paréntesis antes del bloque de instrucciones se evalúe como *false*.

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

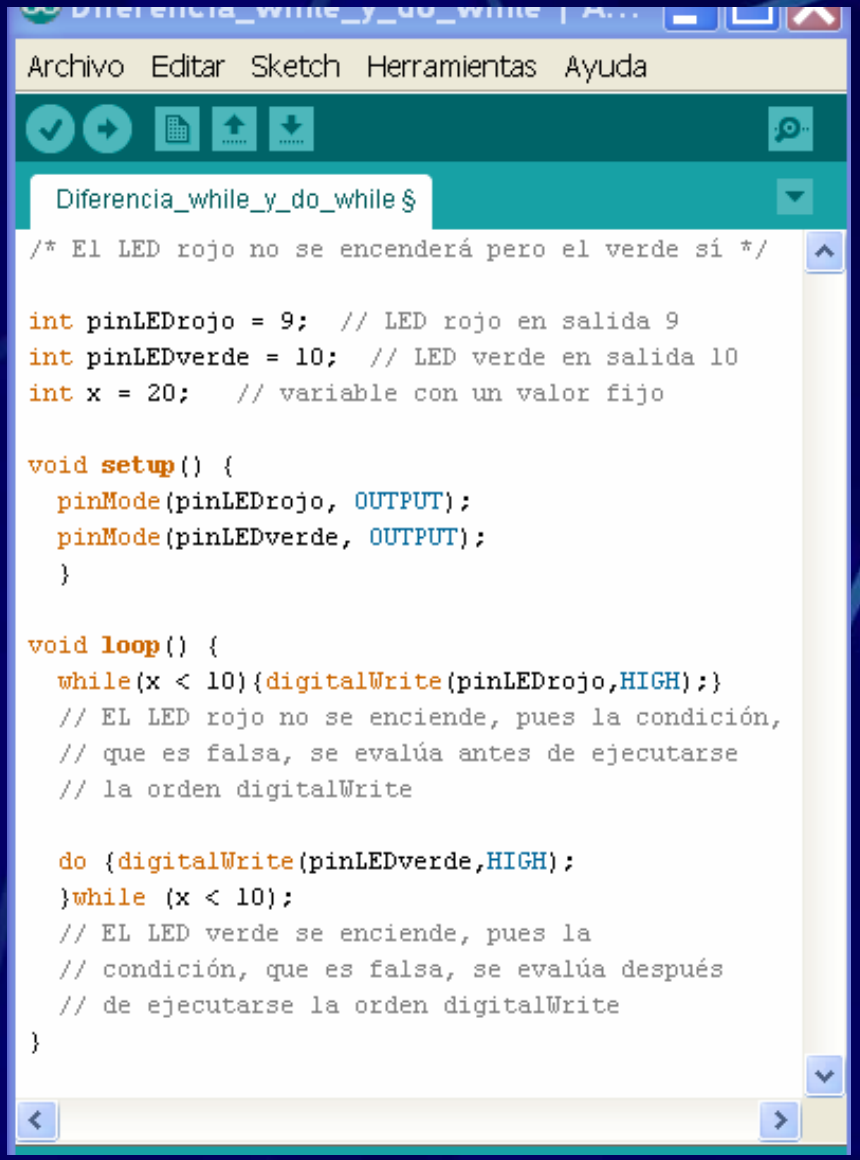
- La condición se evalúa al principio del bucle, por lo que, si la primera vez que se evalúa ya es falsa, las instrucciones contenidas en el bucle no se ejecutarán ninguna vez.

Diagrama de flujo del bucle repetitivo while

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



Programación en ARDUINO: Bucle repetitivo do - while

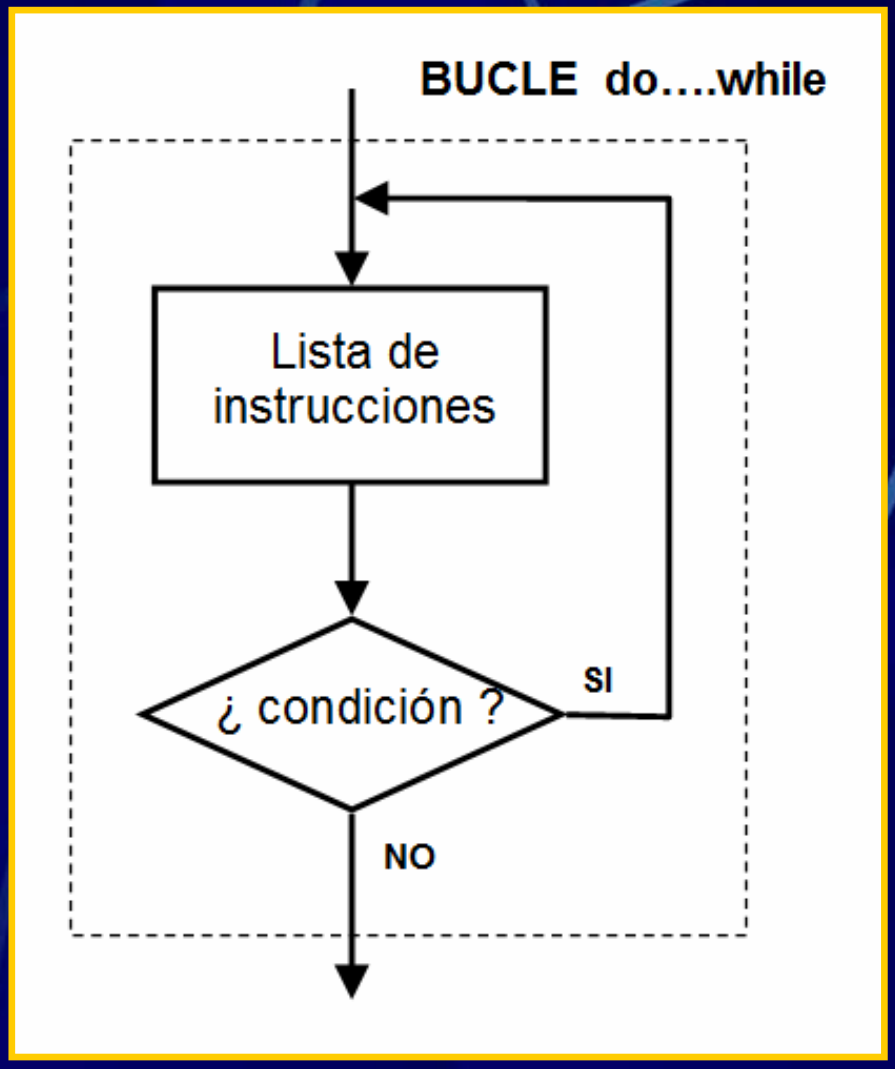


- El bucle **do - while** se repetirá indefinidamente hasta que la expresión booleana que sigue a la palabra while entre paréntesis después del bloque de instrucciones se evalúe como **false**.

```
do {
        bloque de instrucciones;
} while (expresión);
```

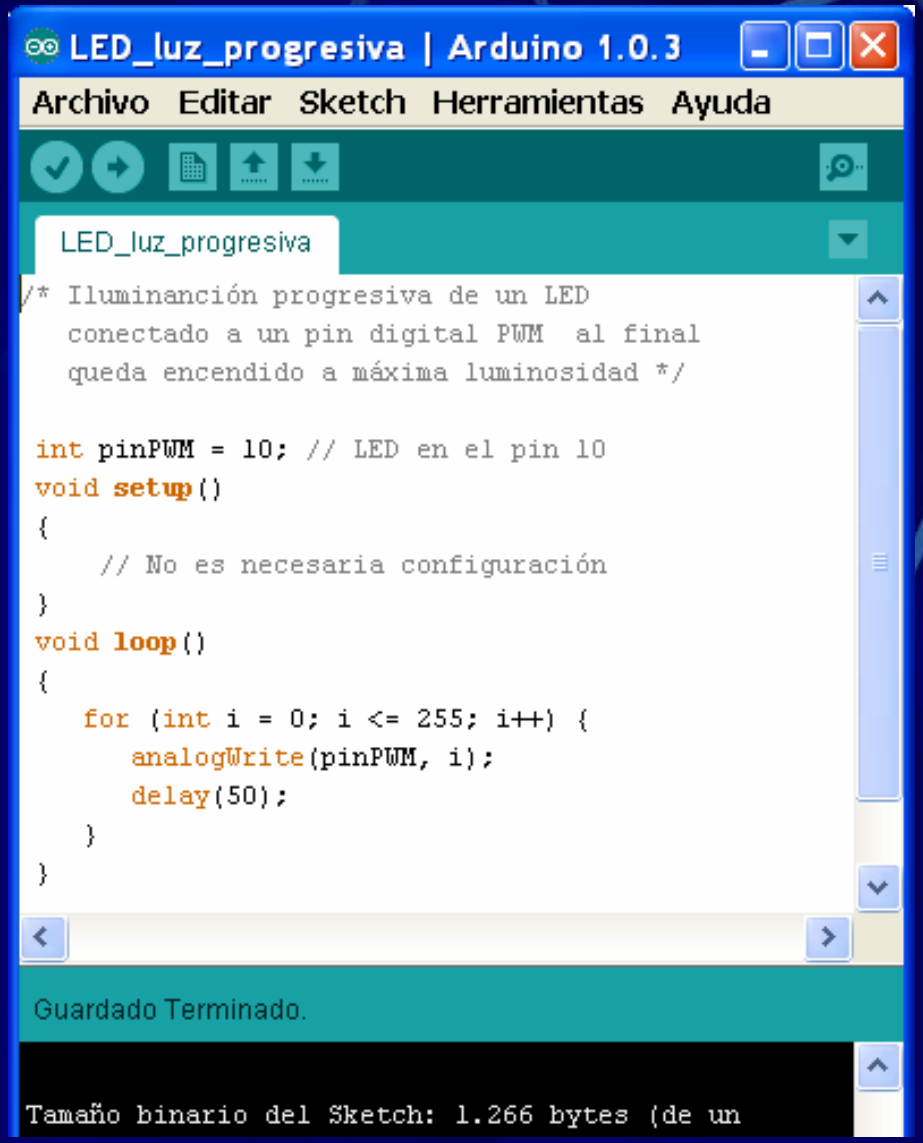
- La expresión se evalúa al final del bucle, por lo que las instrucciones contenidas en el bucle se ejecutarán como mínimo una vez.

Diagrama de flujo del bucle repetitivo do……while




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

Programación en ARDUINO: Bucle repetitivo for



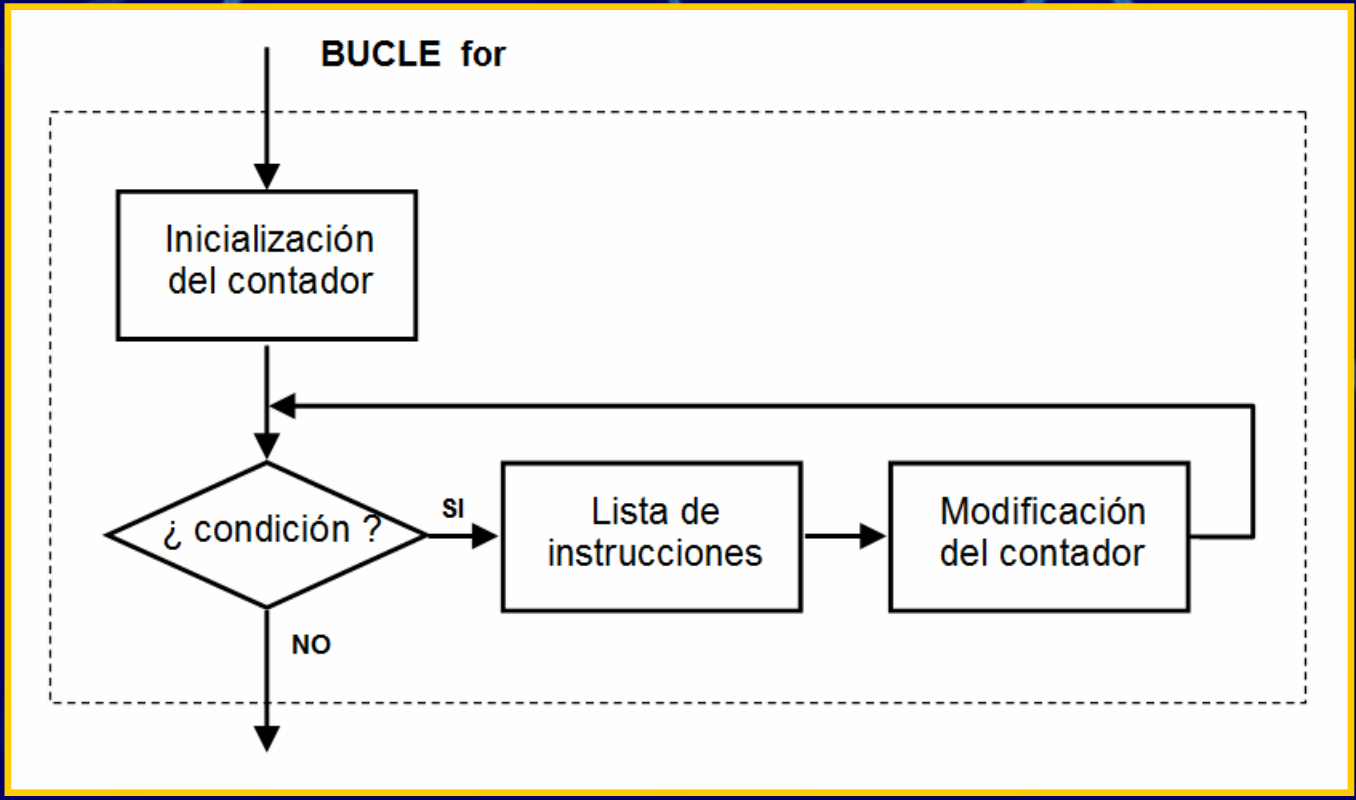
- La estructura **for** repite un bloque de instrucciones las veces que se quiera en tanto se cumpla una condición. Se suele usar una variable contador para guardar la cuenta de las repeticiones y decidir cuándo terminar el bucle.

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

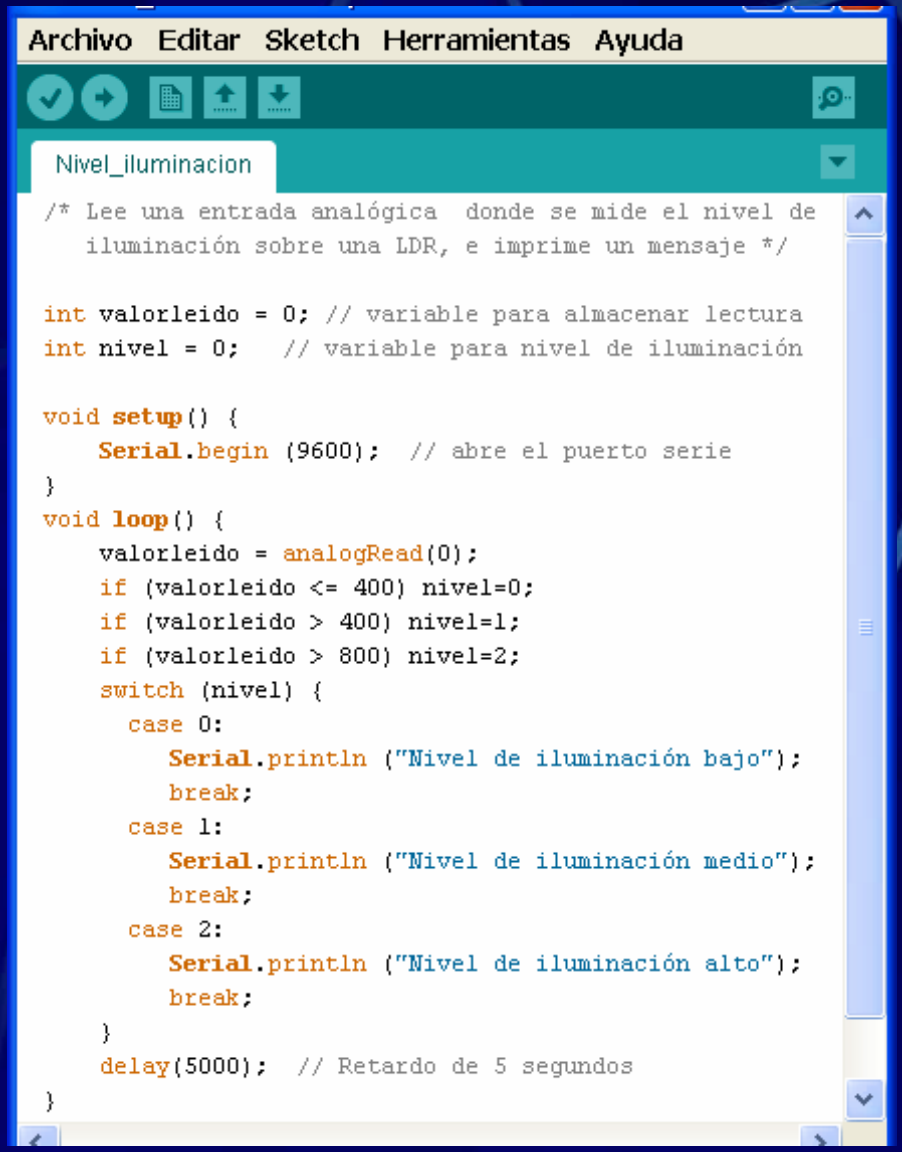
- El **inicio** se ejecuta sólo una vez al principio. En cada pasada se comprueba la **condición**. Si es cierta, se ejecutan las **instrucciones** y la **modificación**. El bucle termina cuando la condición se vuelve falsa.

Diagrama de flujo del bucle repetitivo for

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



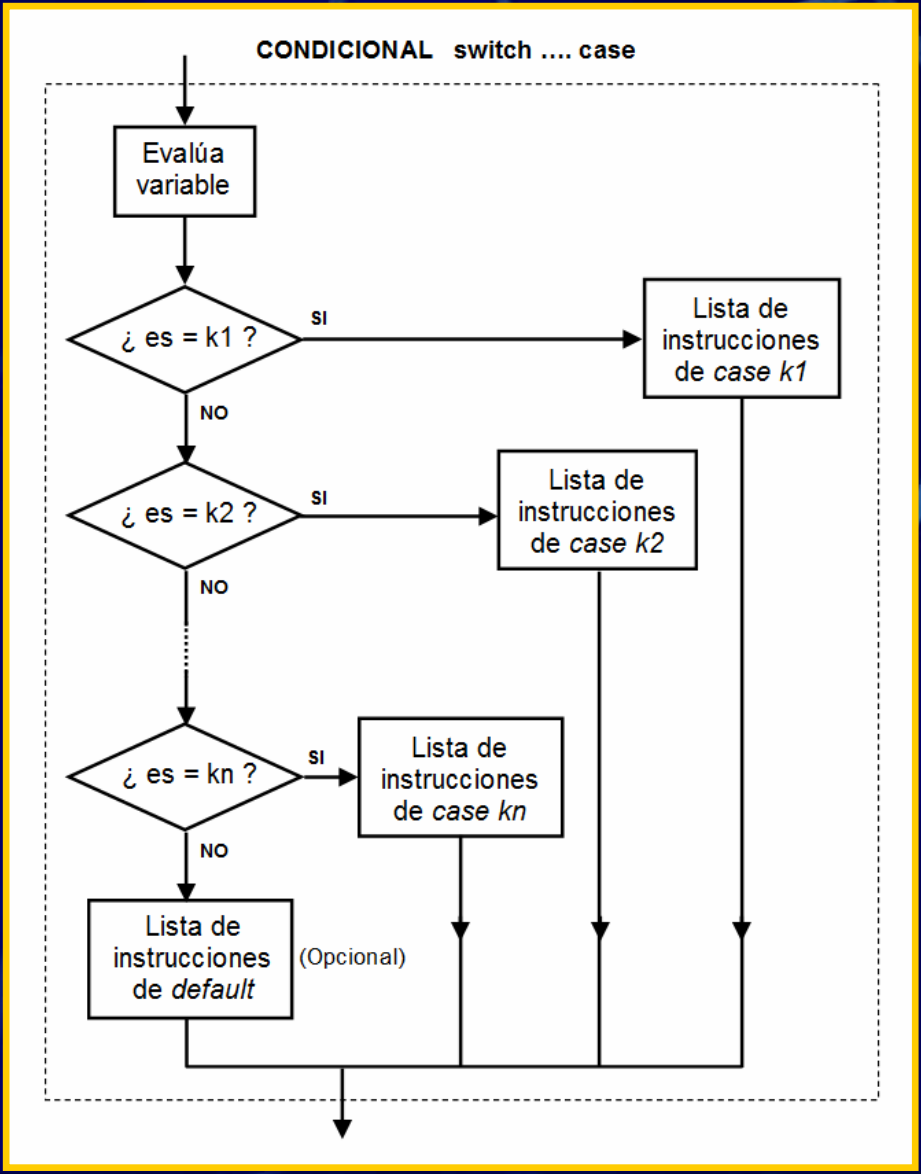
Program. en ARDUINO: Estructura condicional switch...case



- 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.;
}
```

Programación en ARDUINO: break y continue

Uso_break_y_continue §

```
/*  El LED rojo se encenderá progresivamente hasta
el valor de umbral. El LED verde se encenderá
progresivamente hasta 40, a continuación dará un
salto hasta 120 y seguirá encendiéndose
progresivamente hasta 255  */

int pinLEDrojo = 9;  // LED rojo en salida 9
int pinLEDverde = 10;  // LED verde en salida 10
int umbral = 125;   // variable con un valor fijo

void setup() {
  pinMode(pinLEDrojo, OUTPUT);
  pinMode(pinLEDverde, OUTPUT);
  }
void loop() {
  for(int x=0;x<255;x++)
  {  analogWrite(pinLEDrojo,x);
     if(x > umbral) break;
     delay(50);
  }
  for(int x=0;x<255;x++)
  {  if(x>40 && x<120) continue;
     analogWrite(pinLEDverde, x);
     delay(50);
  }
}
```

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

  Se utiliza habitualmente en las estructuras repetitivas for, while y do-while. También se utiliza para salir de la estructura switch...case.

- La instrucción **continue** salta el resto de la iteración actual de un bucle for, while o do-while, pero no sale de ellos, sino que vuelve a chequear la expresión condicional y procede a una nueva iteración

Programación en ARDUINO: return

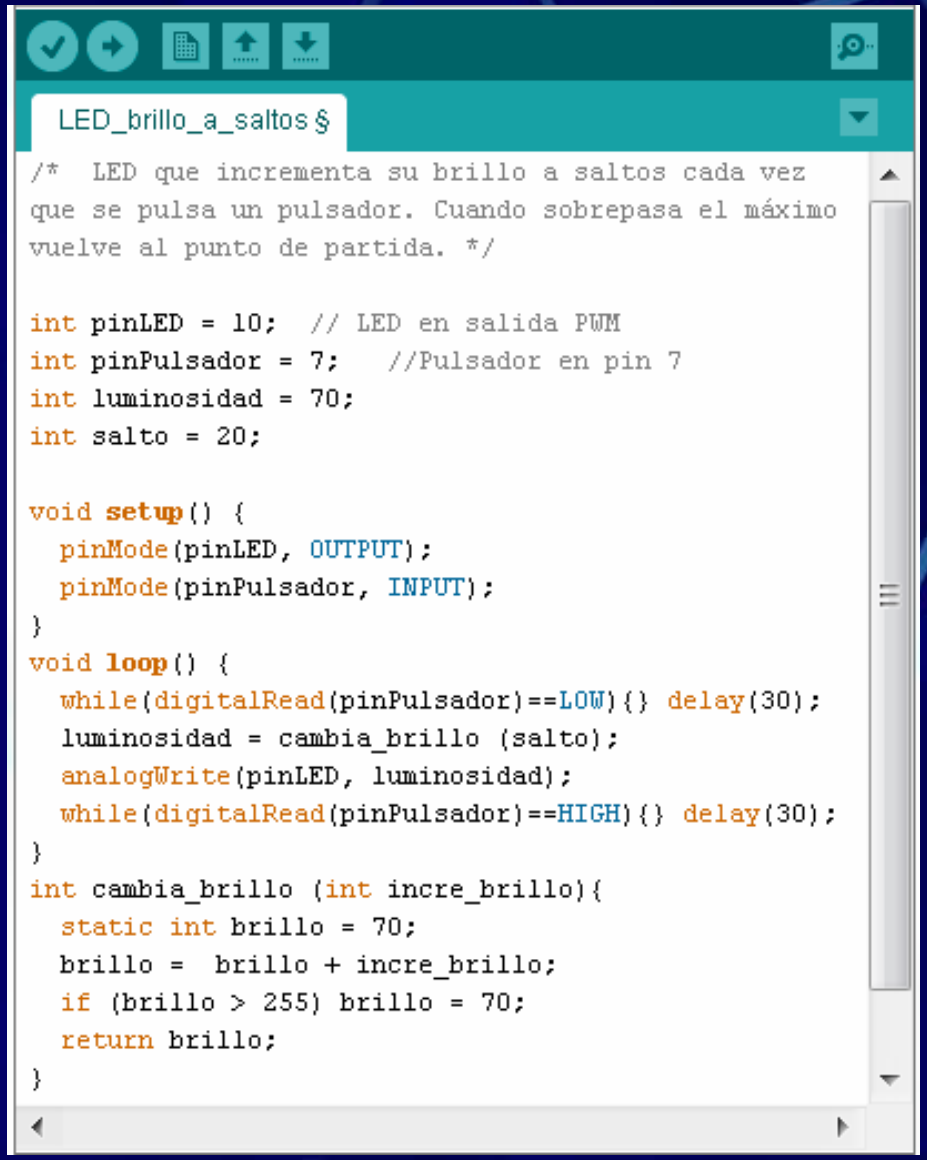
Uso_de_return §

```
/* Si valor en pinAnalog0 menor que en pinAnalog1
se escribe 0 en el monitor serie y se enciende el
LED rojo. Si no, se escribe la diferencia entre
los valores y se enciende el LED verde   */

int pinLR = 9;  // LED rojo en salida 9
int pinLV = 10;  // LED verde en salida 10
int pinAnalog0 = 0; int pinAnalog1 = 1;

void setup() {
  Serial.begin(9600);
  pinMode(pinLR, OUTPUT);
  pinMode(pinLV, OUTPUT);
 }
int operacion (int x, int y){
  int resultado;
  if(x<y){resultado = 0; digitalWrite (pinLR, HIGH);
     return resultado;}
     resultado = x - y;
     digitalWrite(pinLV, HIGH);
     return resultado;
}
void loop() {
    int i; int j; int k;
    i = analogRead (pinAnalog0);
    j = analogRead (pinAnalog1);
    k=operacion(i,j);
    Serial.println(k);
}
```

- La instrucción **return** termina la función en el punto en el que se encuentra (el resto no se ejecuta) y devuelve, si se desea, una valor desde dicha función a la función que la ha llamado.
- Hay dos formas válidas, sin devolver nada o devolviendo un valor.

```
return;
return valor;
```

Programación en ARDUINO: static



LED_brillo_a_saltos §

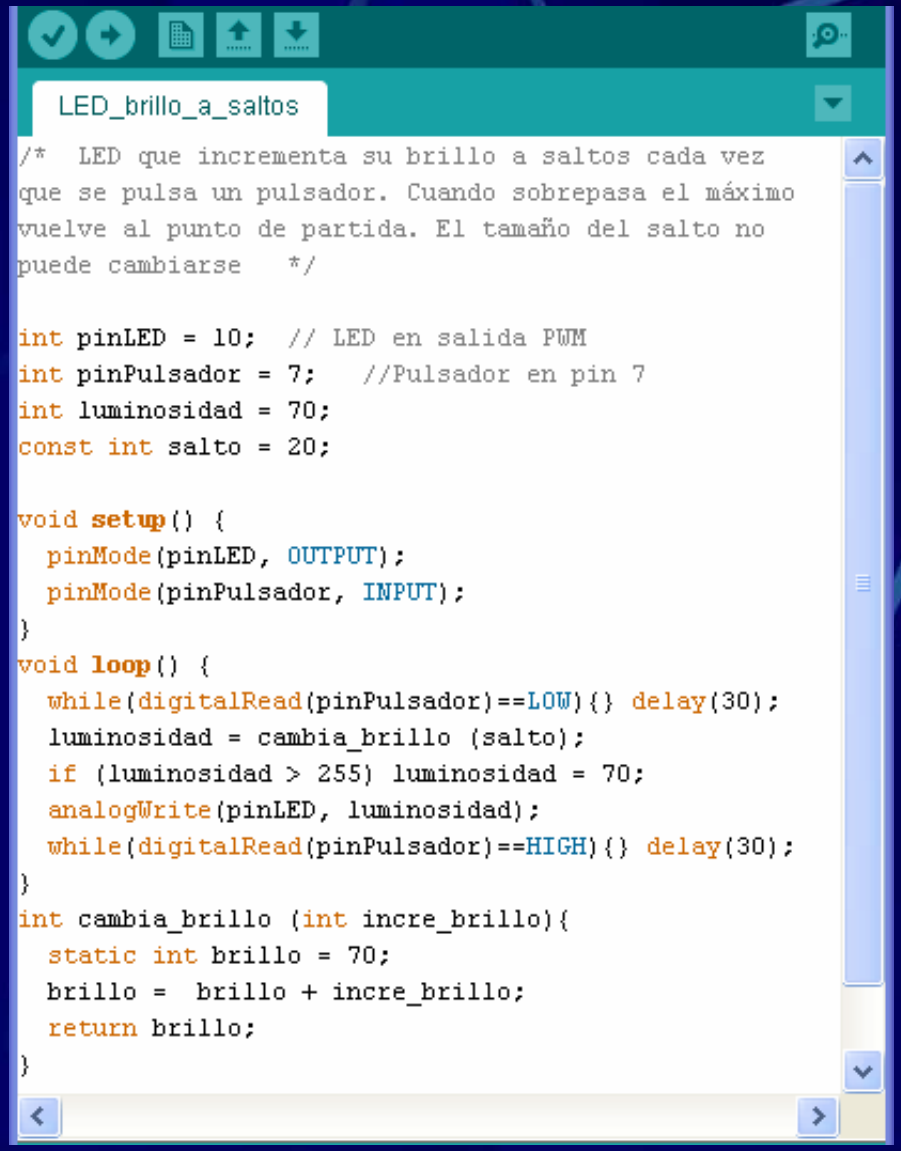
```
/*  LED que incrementa su brillo a saltos cada vez
que se pulsa un pulsador. Cuando sobrepasa el máximo
vuelve al punto de partida. */

int pinLED = 10;  // LED en salida PWM
int pinPulsador = 7;   //Pulsador en pin 7
int luminosidad = 70;
int salto = 20;

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

- Las **variables locales** definidas dentro de una función **no conservan sus valores** entre llamadas sucesivas a dicha función.
- Si calificamos a dicha variable local como **static**, adquiere la propiedad de que conserva su valor entre las sucesivas llamadas a la función.

  static tipovar nombrevar=val_ini;

- Las variables static sólo se crean y se inicializan la primera vez que se llama a la función en que se han creado.

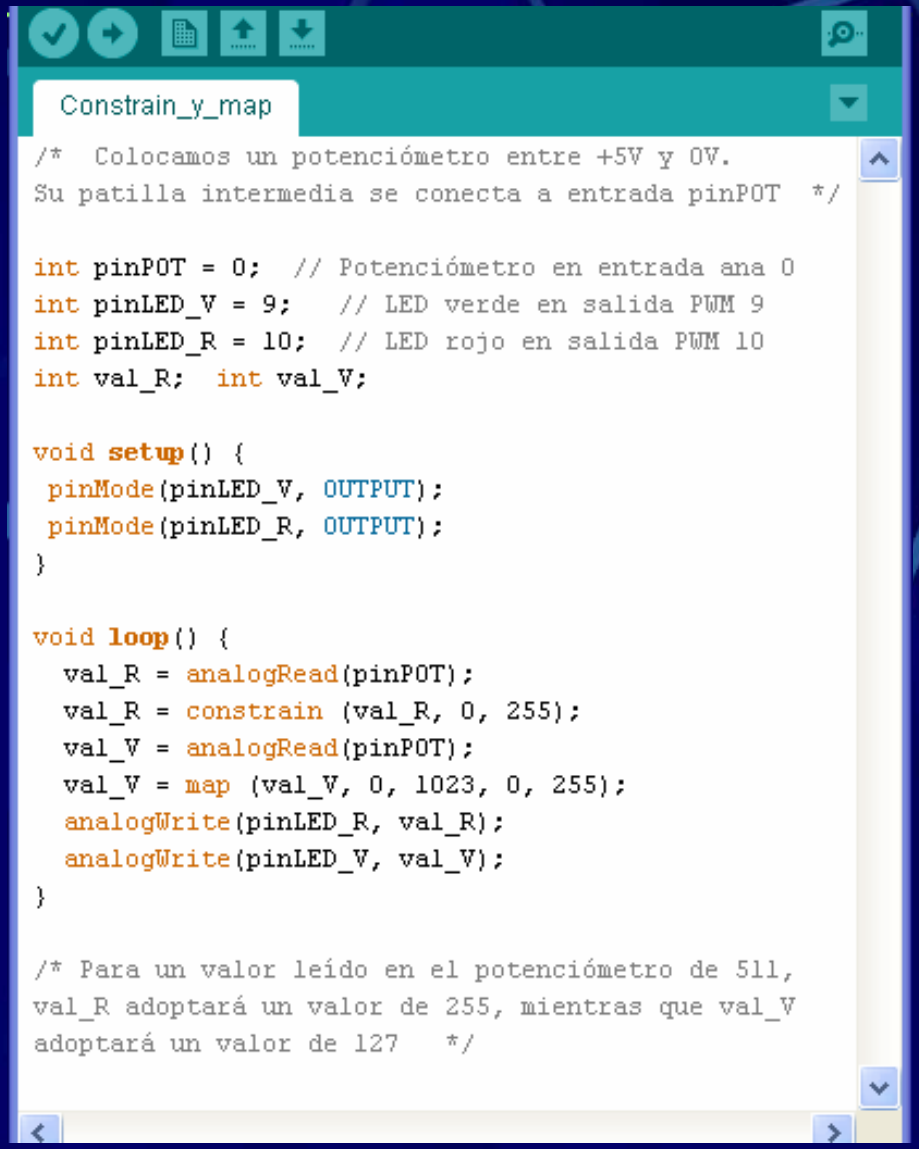
Programación en ARDUINO: const



- 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: funciones matemáticas



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]