Programa Código Escuela 4.0

Placa de expansión de Arduino. Barrera y sensores de presencia.

Índice

Introducción

En este documento vamos a trabajar **programas sencillos** para que el alumnado empiece a manejar **Arduino**.

Se utilizará principalmente la **placa de expansión** de motores para facilitar los montajes de las prácticas.

Trabajaremos prácticas sencillas con los **componentes incluidos en la dotación de CE 4.0**.

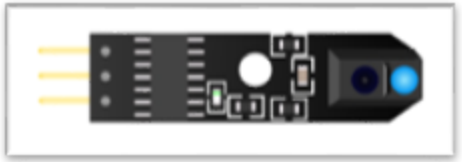
Los programas **se pueden copiar de este documento y pegar directamente en el IDE de Arduino**. Al hacerlo hay que tener cuidado, ya que puede suceder que el texto explicativo (lo que va a continuación de // en letra gris) se cambie de línea y el IDE lo interprete como código, dando errores de compilación.

Al final de las prácticas se incluye una sección de Para aprender más donde se explican con más detalles conceptos como abrir una instancia, PWM,...

Materiales

A continuación, se detallan los elementos que necesitaremos para desarrollar las prácticas.

- Arduino R4 con cable de conexión al ordenador.
- Placa de expansión para Arduino.
- Portapilas con 4 pilas AA.
- Comparador de umbral (LDR).
- 2 sensores IR.
- Servomotor de 180º.
- 4 cables dupont hembra-hembra.
- 2 cables dupont macho-hembra.



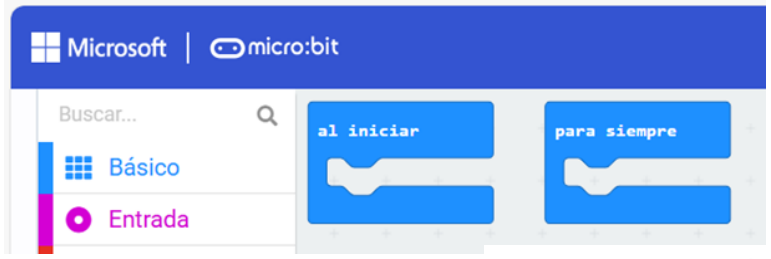
La estructura de un programa en Arduino

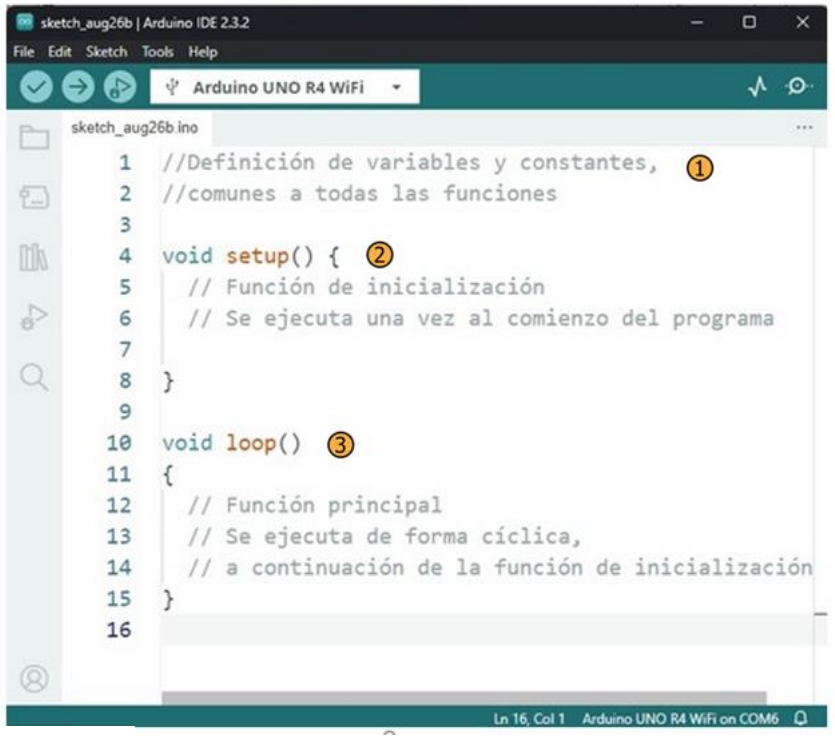
El IDE de Arduino utiliza un lenguaje de programación textual basado en el lenguaje C/C++, enriquecido con funciones propias que simplifican la programación de dispositivos. Todo programa presentará las siguientes partes:

① **Definiciones:** al inicio de un programa se describe en qué consiste y se definen las constantes, las variables globales y las bibliotecas que operarán en él.

② `void setup()`: es la función de inicialización o configuración. Incluye las funciones de configuración del programa y las acciones que deban ejecutarse una sola vez y al comienzo del programa.

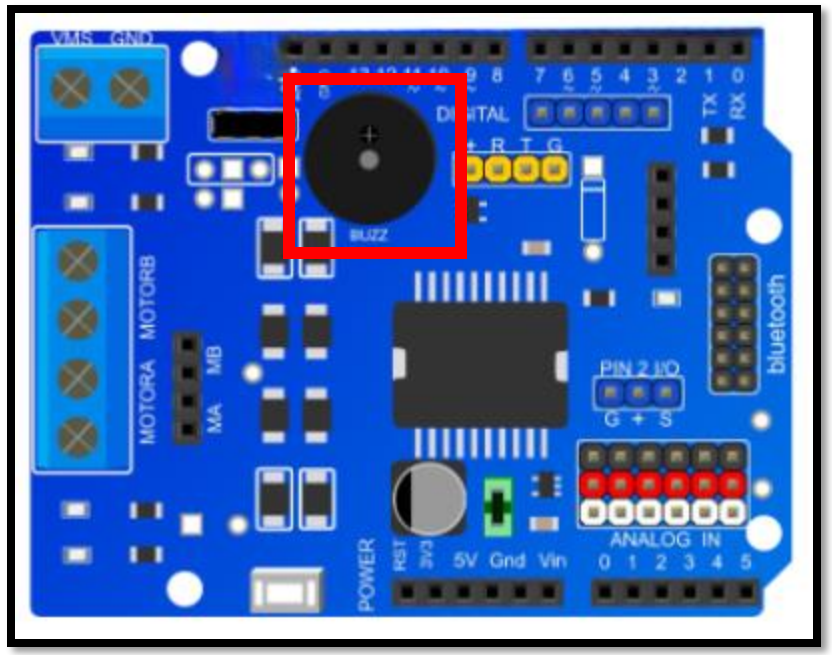
③ `void loop()`: es la función principal y actúa como un bucle infinito. Incluye todas las funciones que queramos que el programa repita de forma secuencial una y otra vez después de `void setup()`.





Zumbador. Funcionamiento

Un **zumbador (buzzer)** es un pequeño dispositivo que produce sonido mediante vibración interna.
La placa de expansión viene con uno de tipo activo, es decir, suena con solo aplicar voltaje. **Está conectado al pin 4 y a GND**.
Su funcionamiento básico consiste en enviar impulsos eléctricos desde el pin para generar tonos o pitidos.



Para programar el zumbador utilizaremos tres comandos:

- **digitalWrite(buzzer, HIGH);** envía corriente al pin que hemos definido como zumbador haciendo que emita sonido. Si utilizamos **LOW** en lugar de HIGH hará que deje de sonar.
- **delay(500);** Nos dice el tiempo, en milisegundos, durante el cuál estará sonando o en silencio.
- **tone(buzzer, 440);** con este comando podemos controlar la frecuencia del pitido o nota con la que suena, en este ejemplo sería un La. Para que deje de sonar también podemos utilizar noTone(buzzer);

Zumbador programas.

Vamos a ver dos programas: en el primero haremos que el zumbador emita sonidos intermitentes. En el segundo haremos lo mismo, pero controlando la nota que emite.

Programa 1

```
#define buzzer 4// Pin donde está conectado el
// zumbador

void setup() {
pinMode(buzzer, OUTPUT);
}

void loop() {
digitalWrite(buzzer, HIGH); // Encendido (emite
pitido)
delay(500); // pausa de medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // pausa de medio segundo
}
```

Programa 2

```
#define buzzer 4// Pin donde está conectado el zumbador

void setup() {
pinMode(buzzer, OUTPUT);
}

void loop() {
tone(buzzer, 1000); // Emite un tono de 1000 Hz
delay(500); // Espera medio segundo
noTone(buzzer); // Apaga el zumbador (silencio).
delay(500); // Espera medio segundo
}
```

Zumbador desafíos.

Utilizando como base los programas anteriores realiza los siguientes desafíos.

Desafío 1:

En código morse las letras se representan mediante puntos y rayas. **Los puntos tienen una duración de 200 ms y las rayas de 600 ms**. El espaciado entre pitidos siempre es de 200 ms.

Empieza creando un programa que emita la **letra S y** después modifícalo para que haga el **SOS**, sabiendo que:
S = ... (tres puntos).
O = - - - (tres rayas).

Si tienes tiempo puedes probar a programar otros mensajes.

Desafío 2:

Vamos a utilizar el comando tone para que Arduino reproduzca una melodía. Vamos a empezar por la primera frase de Frère Jacques.

Las frecuencias para cada nota son:
DO = 262; RE = 294; MI = 330; FA = 349; SOL = 320.

Las notas de la primera parte son: Do – Re- Mi- Silencio.

Si tienes tiempo, la segunda parte de la canción tiene las notas: MI- FA- SOL-Silencio.

Tanto las notas como el silencio deben tener una duración de unos 500 ms.

Zumbador desafíos. Soluciones.

Desafío 1:

```
#define buzzer 4// Pin donde está conectado el  zumbador
void setup() {
pinMode(buzzer, OUTPUT);
}
void loop() {
// --- S: · · · ---
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(600); // pausa entre letras
// --- O: — — — ---
digitalWrite(buzzer, HIGH); delay(600); // raya
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(600); // raya
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(600); // raya
digitalWrite(buzzer, LOW); delay(600); // pausa entre letras
// --- S: · · · ---
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(200); // pausa entre símbolos
digitalWrite(buzzer, HIGH); delay(200); // punto
digitalWrite(buzzer, LOW); delay(2000); // pausa antes de repetir
}
```

Desafío 2:

```
#define buzzer 4// Pin donde está conectado el  zumbador
void setup() {
pinMode(buzzer, OUTPUT);
}
void loop() {
// DO RE MI DO
tone(buzzer, 262); delay(500);
tone(buzzer, 294); delay(500);
tone(buzzer, 330); delay(500);
tone(buzzer, 262); delay(500);
noTone(buzzer); delay(300);
// DO RE MI DO
tone(buzzer, 262); delay(500);
tone(buzzer, 294); delay(500);
tone(buzzer, 330); delay(500);
tone(buzzer, 262); delay(500);
noTone(buzzer); delay(300);
// MI FA SOL
tone(buzzer, 330); delay(500);
tone(buzzer, 349); delay(500);
tone(buzzer, 392); delay(500);
noTone(buzzer); delay(500);
// MI FA SOL
tone(buzzer, 330); delay(500);
tone(buzzer, 349); delay(500);
tone(buzzer, 392); delay(500);
noTone(buzzer); delay(1500);
}
```

LDR-Comparador de umbral

Un **LDR** (Light Dependent Resistor) es un sensor cuya resistencia varía según la cantidad de luz que recibe. A más luz, menor resistencia; a menos luz, mayor resistencia. Normalmente se utilizan con los pines analógicos de Arduino, obteniendo un valor comprendido entre 0 y 1023.

El sensor que tenemos en nuestro kit funciona es un comparador de umbral. Esto quiere decir que enviará una señal digital (0 o 1) según detecte suficiente luz o no, respectivamente.

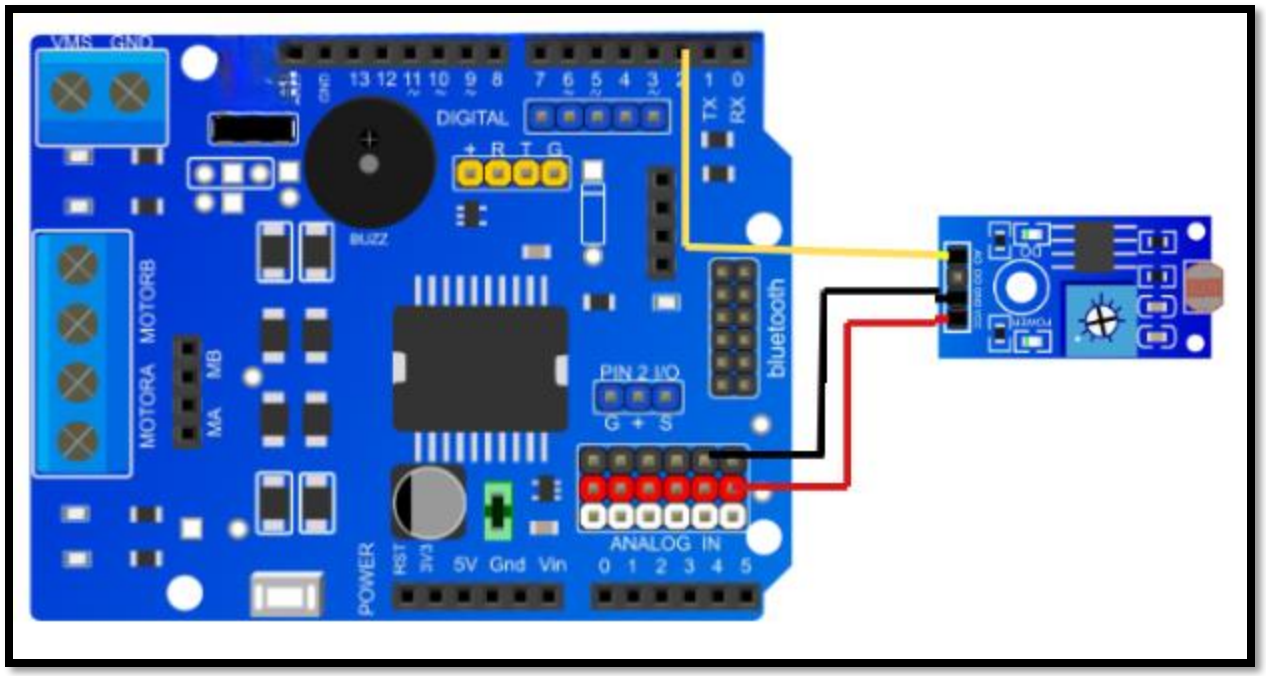
El sensor que incorpora dos LEDs, uno que se enciende cuando está conectado a la corriente y otro que se enciende cuando detecta suficiente luz.

Puedes utilizar un destornillador para calibrar el umbral del nivel de luz. Si está completamente apretado lee siempre luz, y al revés lee siempre oscuridad. Ajústalo para que lea luz justo con la iluminación ambiente de tu aula. Al pasar la mano cerca verás que se apaga el LED.

Detecta luz – Se enciende el LED – envía 0 lógico.
No detecta luz – Se apaga el LED – envía 1 lógico.

Zumbador con LDR

En esta práctica **vamos a usar el comparador de umbral como un detector de presencia**: al pasar la mano el sensor detectará oscuridad, mandando un 1 al pin 2. Cuando esto suceda el zumbador debe empezar a emitir pitidos intermitentes.



Realiza el montaje como se indica en la figura. El sensor está conectado al pin 2, y recuerda que el zumbador está en el pin 4.

Este programa es del tipo **condicional**, el zumbador se activa cuando se cumple una condición. Utilizamos la estructura de programación **if/else**.

Zumbador con LDR. Programa y desafío.

```
#define buzzer 4
#define LDR  2

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(LDR, INPUT); // LDR con salida digital
}

void loop() {
if (digitalRead(LDR) == HIGH) { // Si no detecta luz,
// la salida del sensor será 1 = HIGH
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); //Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo
} else {// Si no se cumple la condición anterior
digitalWrite(buzzer, LOW); // Apagado (silencio)
}
}
```

En este programa el zumbador sólo pita mientras el sensor está leyendo oscuridad, pero en muchos casos querremos dejar constancia de que se ha detectado presencia.

Desafío 1.

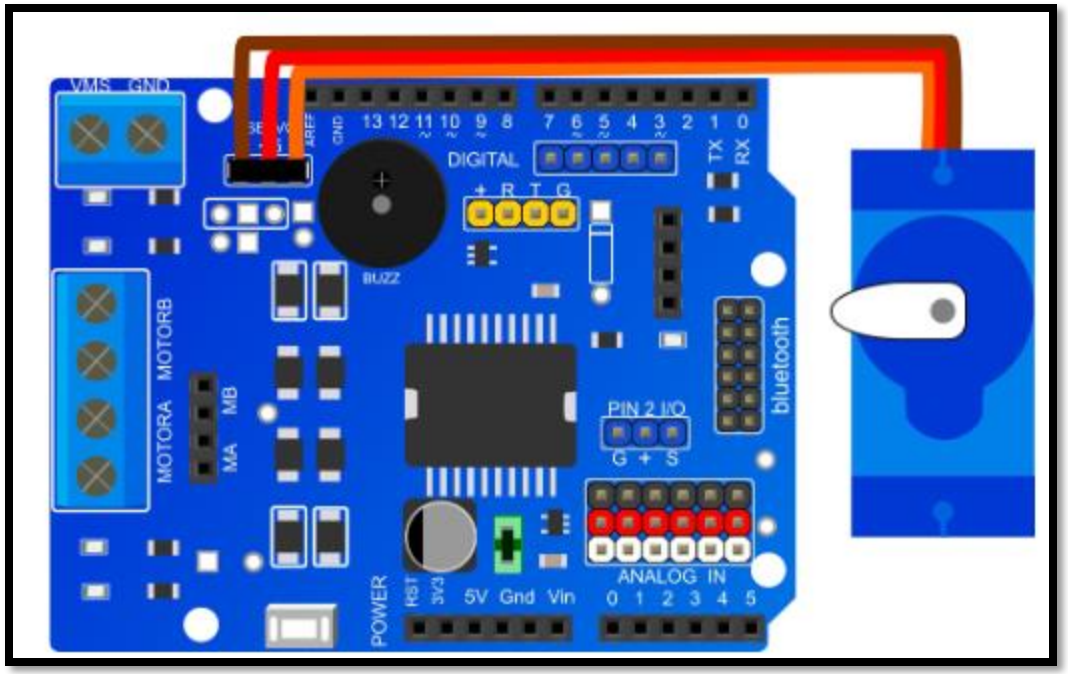
Cambia la programación para que el zumbador esté emitiendo pitidos cada 500 ms durante 2 segundos cada vez que el sensor lea un 1 (detecte presencia).
Prueba distincas secuencias o melodía que puedes programar cuando detecte presencia.

Zumbador con LDR. Solución.

```
#define buzzer 4
#define LDR  2

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(LDR, INPUT); // LDR con salida digital
}

void loop() {
if (digitalRead(LDR) == HIGH) { // Si no detecta luz, la salida del sensor será 1 = HIGH
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); //Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); //Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo

} else {// Si no se cumple la condición anterior
digitalWrite(buzzer, LOW); // Apagado (silencio)
}
}
```

Servomotor

Los servomotores de 180º se sitúan en un ángulo concreto según la información que le proporcionamos a través de nuestra programación . En este caso sencillo haremos que el servomotor vaya girando de 90º en 90º.



La placa de expansión tiene una conexión dedicada para servomotores. Es importante respetar el código de **colores de los cables: marrón oscuro va al GND (tierra), rojo a la alimentación (+5 V) y el naranja/amarillo transmite los datos al pin 9**.

Para utilizar los servos es necesario utilizar una **librería** propia. El control de los servomotores se realiza a través de **PWM**. Puedes consultar la sección para aprender más para entender mejor lo que significa.

Servomotor. Programa y desafío.

```
#include <Servo.h>
#define pin_Servo 9
Servo miServo;//nombramos al servo como miServo,
//técnicamente se llama declarar una instancia, puedes ver lo
//que significa en la sección Para aprender más.

void setup() {
servo180.attach(pin_servo180);//indicamos el pin al que
// está conectado el servo.
}
void loop() {
servo180.write (0);//hacemos que el servo se sitúe en 0º
delay (500);
servo180.write (90);//hacemos que el servo se sitúe en 90º
delay (500);
servo180.write (180);//hacemos que el servo se sitúe en 180º
delay (500);
servo180.write (90);//hacemos que el servo se sitúe en 90º
delay (500);
}
```

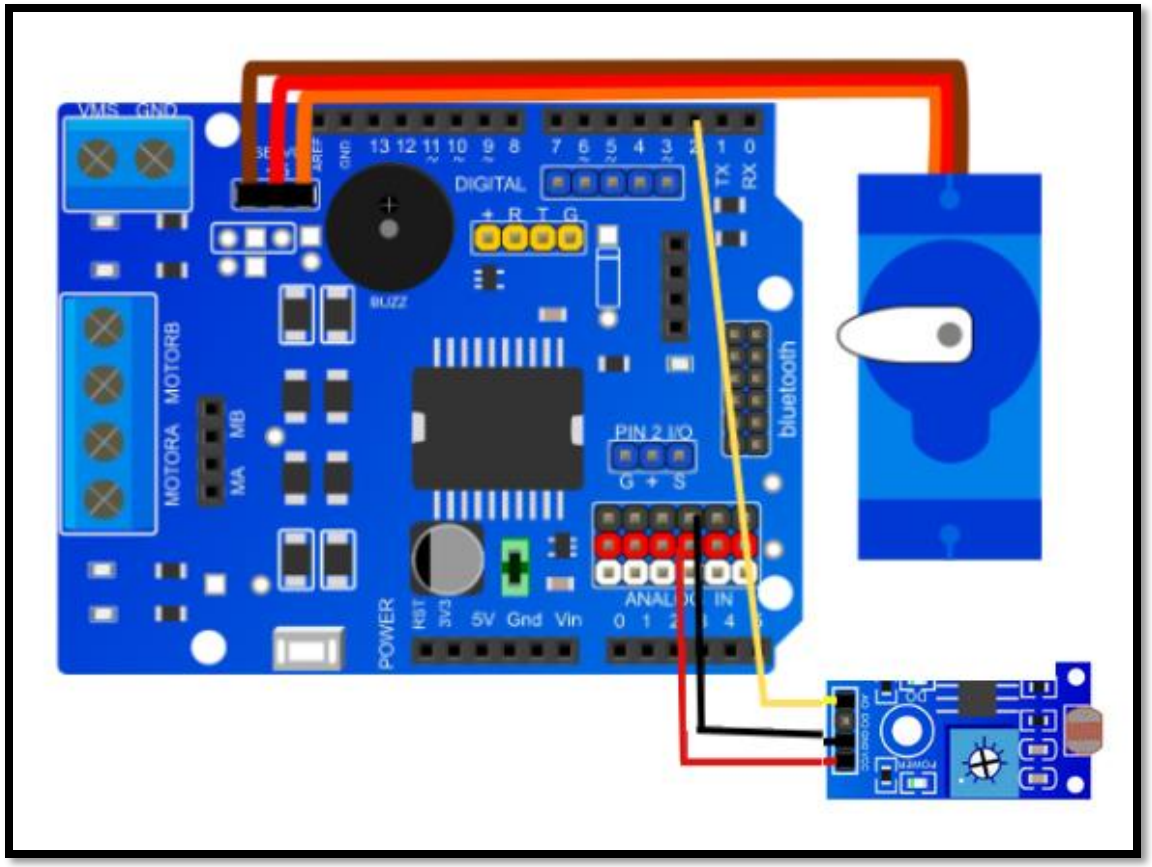
Los servos de 180º son los que tienen la etiqueta de color azul o un gomet azul en un lado.

Desafío 1:

Prueba a modificar los ángulos por los que pasa el servo y los tiempos que permanece en cada posición.

Servomotor con zumbador y LDR (barrera)

Vamos a juntar los proyectos anteriores para hacer que cuando el sensor de umbral (LDR) detecte presencia se levante el servomotor (pase de 0º a 90º) mientras el zumbador emite pitidos.



Las **conexiones** son:

- **Servomotor: pin 9.**
- **Zumbador: pin 4.**
- **Sensor umbral (LDR): pin 2.**

Servomotor con zumbador y LDR (barrera). Programa y desafío.

```
#include <Servo.h>
#define pin_Servo 9
#define buzzer 4
#define LDR  2

Servo miServo;//abrimos instancia miServo

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(LDR, INPUT); // LDR con salida digital
miServo.attach(pin_Servo);//miServo conectado pin_Servo
}

void loop() {
if (digitalRead(LDR) == HIGH) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo
} else {
digitalWrite(buzzer, LOW); // Apagado (silencio)
miServo.write(0);
}
}
```

En este programa cuando el sensor detecta presencia se levanta el servo y emite un pitido durante medio segundo y silencio durante medio segundo.
En total el servo estará levantado un segundo antes de volver a evaluar si ha presencia o no.

Desafío 1:

Un segundo puede ser poco tiempo para que la barrera está levantada. Cambia el programa para que cuando detecte presencia la barrera esté levantada al menos dos segundos. Puedes hacer que esté pitanto durante ese tiempo o que emita un pitido cada vez que el servo cambie de posición.

Zumbador con LDR y servomotor (barrera). Solución.

Zumbido intermitente durante tres segundos.

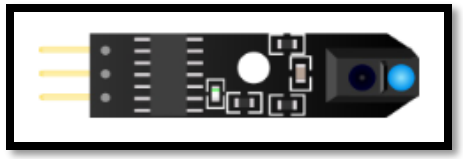
```
#include <Servo.h>
#define pin_Servo 9
#define buzzer 4
#define LDR  2

Servo miServo;//abrimos instancia miServo

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(LDR, INPUT); // LDR con salida digital
miServo.attach(pin_Servo);//miServo conectado pin_Servo
}

void loop() {
if (digitalRead(LDR) == HIGH) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(500); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(500); // Espera medio segundo

} else {
digitalWrite(buzzer, LOW); // Apagado (silencio)
miServo.write(0);
}
}
```

Zumbido al subir y al bajar la barrera.

```
#include <Servo.h>
#define pin_Servo 9
#define buzzer 4
#define LDR  2

Servo miServo;//abrimos instancia miServo

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(LDR, INPUT); // LDR con salida digital
miServo.attach(pin_Servo);//miServo conectado pin_Servo
}

void loop() {
if (digitalRead(LDR) == HIGH) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(2000); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(200); // Espera medio segundo

} else {
digitalWrite(buzzer, LOW); // Apagado (silencio)
miServo.write(0);
}
}
```

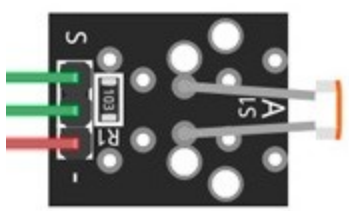
Sensor de infrarrojos (IR)

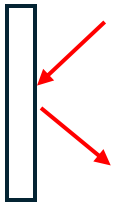
El sensor de infrarrojos consta de un emisor (azul) y un receptor (negro)



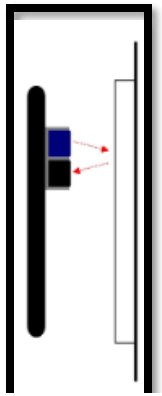
Cuando hay una superficie clara cercana (5 cm o menos) la luz se refleja con suficiente intensidad como para ser detectada.
El sensor dará una **salida digital de 1 cuando detecte luz** y de 0 cuando no lo haga.
En la parte trasera dispone de un pequeño LED que se enciende cuando la salida de digital es 1, es decir, cuando detecte luz.

En cualquier de las prácticas anteriores podemos sustituir el sensor de umbral (LDR) por el sensor de IR y funcionará con la misma programación. Las conexiones son diferentes, puedes consultar la página siguiente para ver cómo hacerlo correctamente.





Cuando un objeto se sitúa frente al LDR impide el paso de la luz, el sensor detecta oscuridad y la salida digital es uno.



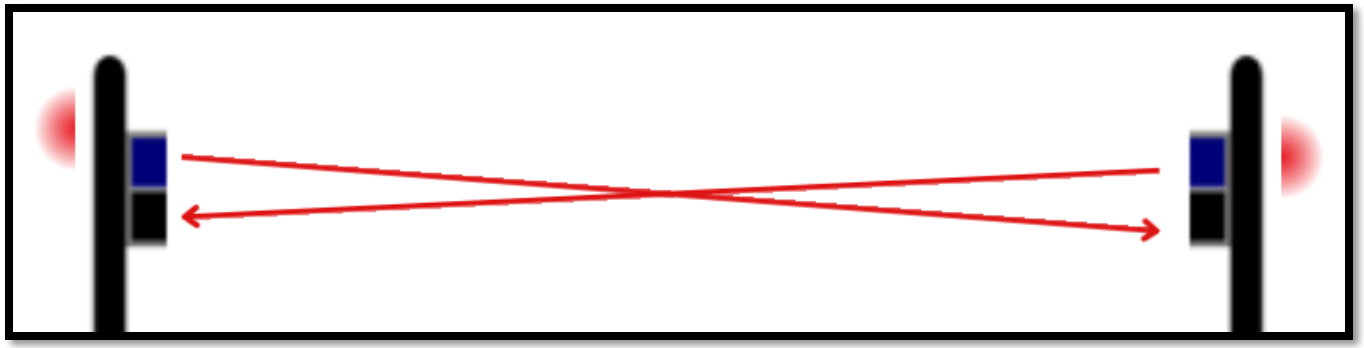
Cuando un objeto se sitúa frente al sensor IR la luz se refleja y es detectada por el receptor. La salida digital es uno.

Los dos sensores dan uno al detectar presencia, pero tienen lógicas diferentes:

- **El sensor de umbral (LDR) da uno cuando no detecta luz.**
- **El sensor de IR da uno cuando detecte luz**

Servomotor con zumbador y dos sensores IR (barrera).

Ahora ampliaremos el rango de detección de presencia a algo más del doble (unos 12 cm) enfrentando dos sensores IR de manera que detecten la luz que emite el otro.



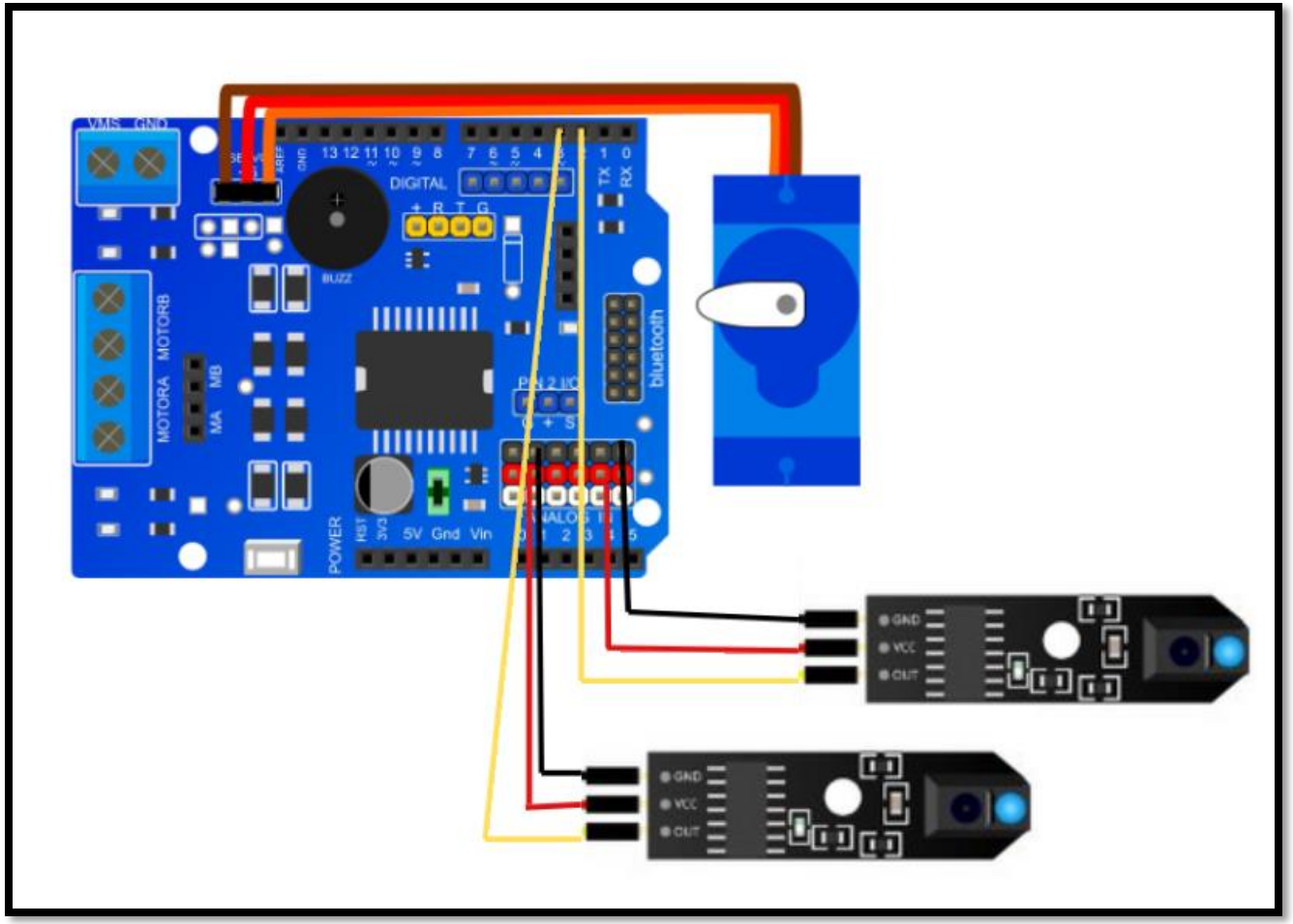
Asegúrate que los LEDs traseros de los sensores están iluminados, es decir ambos detectan la luz emitida por el otro.

Este sistema o similar es ampliamente utilizado (puertas de garaje, ascensores,..) y probablemente lo hayas visto en la vida real o en películas.

Fíjate bien en las conexiones del sensor IR, no son iguales que las del sensor de LDR.

GND
VCC (+ V)
Datos.

Servomotor con zumbador y dos sensores IR (barrera). Montaje y desafío.



A la izquierda tienes el esquema de conexiones. Recuerda que los sensores deben estar enfrentados como se muestra en la página anterior.

Desafío 1:

Realiza el programa teniendo en cuenta:

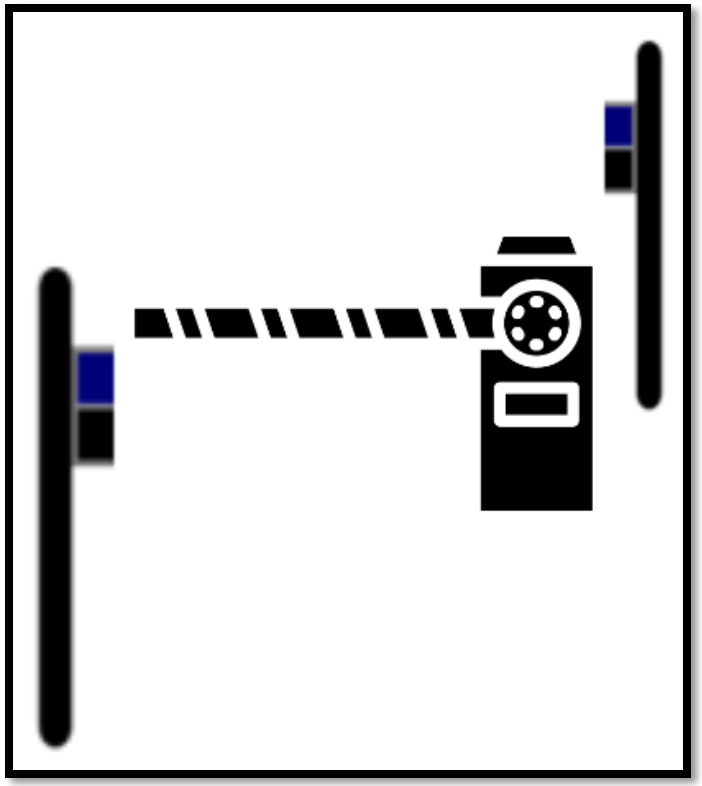
- **Ahora hay dos sensores: pin 2 y pin 3.**
- **Detectan presencia cuando la lectura de cualquiera de ellos sea cero (LOW).**
- **Puedes usar dos condicionales (if), uno para cada sensor.**

Servomotor con zumbador y dos sensores IR (barrera). Solución.

```
#include <Servo.h>
#define pin_Servo 9
#define buzzer 4
#define IF1  2
#define IF2  3

Servo miServo;//abrimos instancia miServo
void setup() {
pinMode(buzzer, OUTPUT);
pinMode(IF1, INPUT); // sensor IR con salida digital
pinMode(IF2, INPUT); // sensor IR con salida digital

miServo.attach(pin_Servo);//miServo conectado pin_Servo
}

void loop() {
digitalWrite(buzzer, LOW); // Apagado (silencio)
miServo.write(0);

if (digitalRead(IF1) == LOW) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(2000); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(200); // Espera medio segundo
}
if (digitalRead(IF2) == LOW) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(2000); // Espera medio segundo
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(200); // Espera medio segundo
}
}
```

Servomotor con zumbador y dos sensores IR (barrera). Entrada y salida. Desafío.

Hasta el ahora hemos dado un tiempo definido para que se baje la barrera. En está práctica utilizaremos el mismo montaje que en la anterior para hacer que uno de los sensores IR sea el de entrada (levanta la barrera cuando detecta presencia) y el otro la baje (después de haber atravesado la barrera)



Desafío 1:

Realiza el programa teniendo en cuenta:

- **Ahora hay dos sensores: pin 2 y pin 3.**
- **Detectan presencia cuando la lectura de sea uno (HIGH).**
- **Puedes usar dos condicionales (if), uno para cada sensor.**
- **Dado que la barrera no vuelve se baja por si misma a la posición cerrada hay que poner en el void setup que el servo se ponga en cero grados, después ya son los sensores los que determinarán la posición de la barrera.**
- **No hace falta que emita varios pitidos, con uno largo es suficiente.**

Servomotor con zumbador y dos sensores IR (barrera). Entrada y salida. Solución.

```
#include <Servo.h>

#define pin_Servo 9
#define buzzer 4
#define IF1 2
#define IF2 3
Servo miServo;//abrimos instancia miServo

void setup() {
pinMode(buzzer, OUTPUT);
pinMode(IF1, INPUT); // sensor IR con salida digital
pinMode(IF2, INPUT); // sensor IR con salida digital
miServo.attach(pin_Servo);//miServo conectado pin_Servo
miServo.write(0);
}
void loop() {
if (digitalRead(IF1) == HIGH) { // Si se pulsa el botón
miServo.write(90); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(2000); // Espera medio segundo
}
if (digitalRead(IF2) == HIGH) { // Si se pulsa el botón
miServo.write(0); //mueve el servo a 90º
digitalWrite(buzzer, HIGH); // Encendido (emite pitido)
delay(700); // Espera medio segundo
digitalWrite(buzzer, LOW); // Apagado (silencio)
delay(2000); // Espera medio segundo
}
}
```

Conclusiones. Próximos pasos.

En los programas de esta presentación se ha intentado simplificar la programación todo lo posible para trabajar con alumnado que tiene poca o ninguna experiencia trabajando con Arduino.

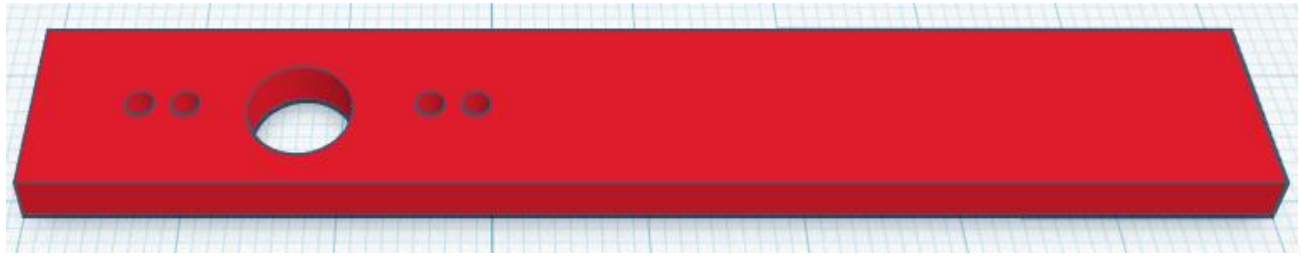
Se pueden utilizar estos montajes y desafíos para introducir nuevos conceptos como:

- **Bucle for:** se puede utilizar para poder controlar mejor los pitidos intermitentes, por ejemplo.
- **Or (||):** En el último programa se puede utilizar esta función en lugar de poner dos condicionales.
- **And (&&):** se puede añadir otro servomotor en el último montaje y programar Arduino para que suba la barrera que corresponde según los datos leídos por cada sensor.
- **Introducir variables:** se pueden usar para contar el número de veces que se abre la barrera y, por ejemplo, indicar si hay plazas libres en un parking.

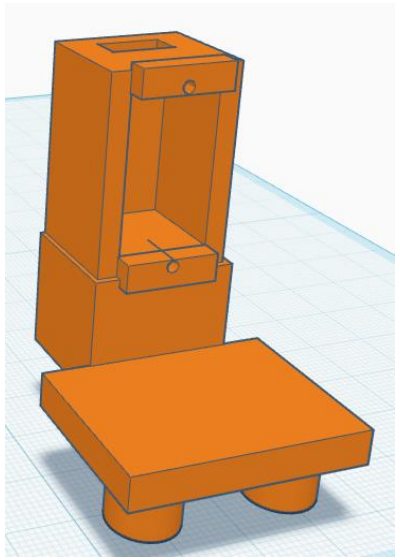
Todas las prácticas se pueden realizar en su totalidad usando exclusivamente el material del kit de robótica para Arduino de C.E. 4.0. Si se dispone de otro material se pueden completar las prácticas con LEDs, pulsadores,... e incluso introducir la función de mapeo con un potenciómetro.

Material para imprimir 3D.

A continuación, tenéis unos enlaces para soportes, barreras,... que os pueden venir bien para realizar estas prácticas u otras parecidas.

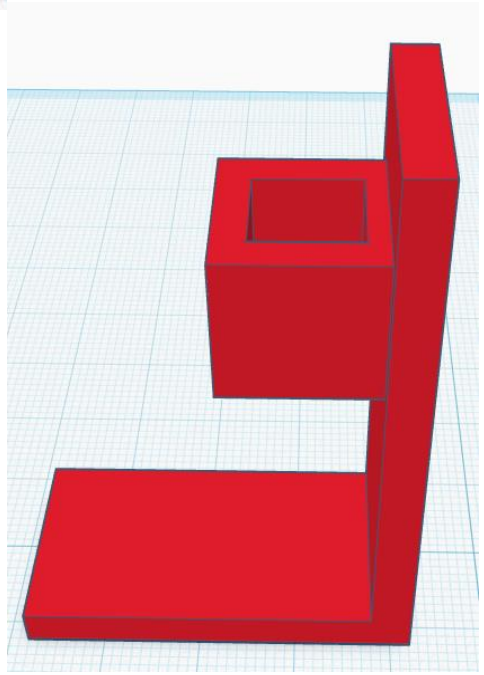


Barrera



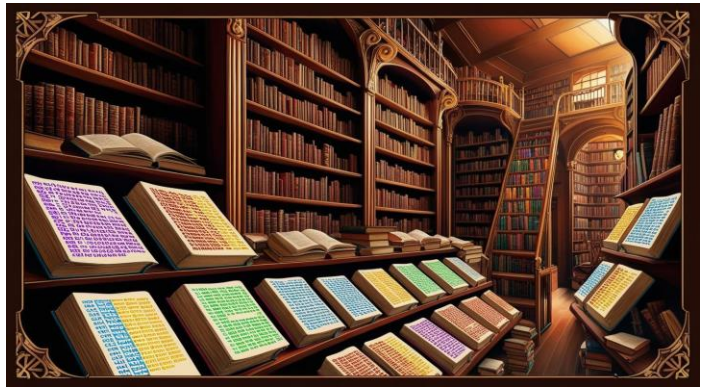
Soporte Servomotor

Soporte Sensores



Para aprender más I. Librerías.

Un **lenguaje de programación** siempre va a tener una serie de **funciones básicas** implementadas que el programador puede utilizar.



Por ejemplo, tanto Scratch como MakeCode tienen funciones para sumar, restar, comparar,...

En algunas ocasiones podemos necesitar funciones más complejas o específicas para nuestro programa. Por ejemplo, hacer integrales si estamos haciendo cálculos.

Cuando necesitamos funciones que no son las básicas utilizamos las librerías

Una **librería informática** nos va a permitir añadir ciertas funciones específicas a nuestro lenguaje de programación.

Estas librerías no se cargan por defecto porque lo haría mucho más pesado y poco funcional, también a veces puede haber problemas de compatibilidad. El éxito de un lenguaje de programación está vinculado a las librerías que tiene desarrolladas ya que va a determinar qué programas podemos realizar con él.

Para aprender más II. Instancias.

Una **instancia** es un objeto concreto creado a partir de una clase. La clase define las propiedades y métodos, mientras que la instancia es el elemento real que usamos en el programa. Podemos entender la clase como un molde a partir del cual crearemos los objetos necesarios. En Arduino, esto se aplica cuando trabajamos con librerías que definen clases, como la librería **Servo.h**, que permite controlar servomotores.

¿Para qué se utiliza?

Crear una instancia nos permite manejar un componente específico con sus propias características. Por ejemplo, si tenemos dos servomotores, podemos crear dos instancias de la clase `Servo` para controlarlos de manera independiente. Cada instancia tendrá sus propios métodos para escribir ángulos, adjuntar pines, etc.
Al crear una instancia nombramos el objeto como deseemos. En este caso hemos nombrado a los servos como servo1 y servo2.

```
#include <Servo.h>

// Crear instancias de la clase Servo
Servo servo1;
Servo servo2;

void setup() {
  // Asociar cada instancia a un pin
  servo1.attach(9);
  servo2.attach(10);
}

void loop() {
  // Mover servo1 de 0° a 180°
    servo1.write(0);
    servo2.write(180); // servo2 hace el movimiento inverso
    delay(500);
    servo1.write(180);
    servo2.write(0);
    delay(500);
```

Para aprender más III. PWM.

PWM (Pulse Width Modulation) es una técnica que consiste en enviar una señal digital que alterna entre encendido y apagado a una frecuencia constante, pero variando el **ancho del pulso** (el tiempo que permanece en alto). Esta variación controla la cantidad de energía entregada a un dispositivo, simulando un voltaje analógico usando una señal digital.

¿Para qué se utiliza?

Por ejemplo, para enviar al servomotor la información del ángulo mediante una señal digital ("0" y "1")
El porcentaje de tiempo que se encuentra a nivel alto ("1") indica el valor del ángulo.

- 360º: todo el tiempo "1"
- 180º: mitad del tiempo "1" mitad "0"
- 0º: todo el tiempo "0"