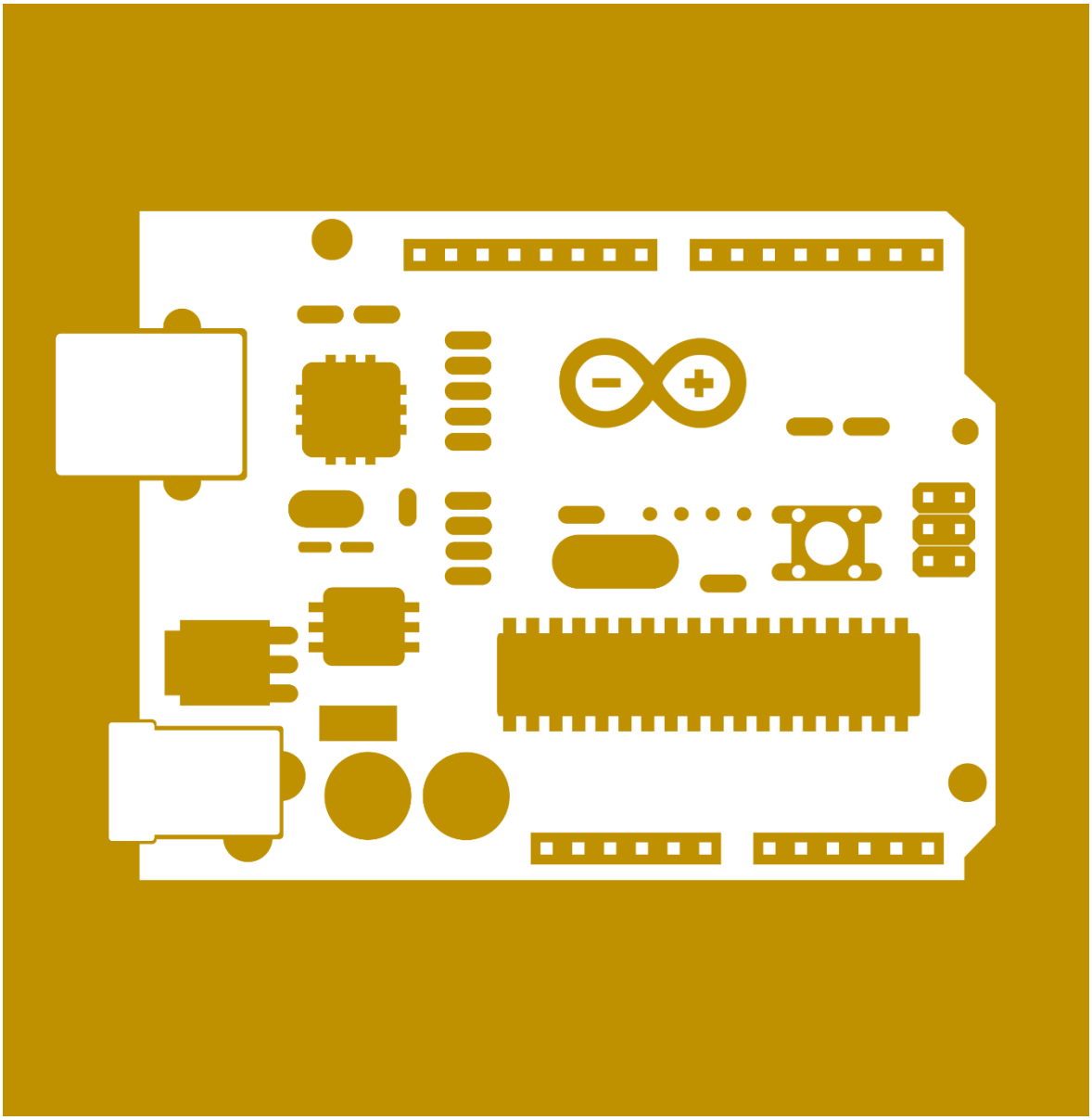


# INICIACIÓN A ARDUINO

Curso 2024-2025
Provincia de Burgos

PROYECTO DE INNOVACIÓN EDUCATIVA
Código Escuela 4.0

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
Código Escuela 4.0
Junta de Castilla y León
Consejería de Educación

Programa financiado por el Ministerio de Educación, Formación Profesional y Deportes

ÍNDICE

# PARTE I. Componentes Arduino



# PARTE II. Lenguaje Arduino

# PARTE I. Componentes Arduino

## 1. CONCEPTOS BÁSICOS

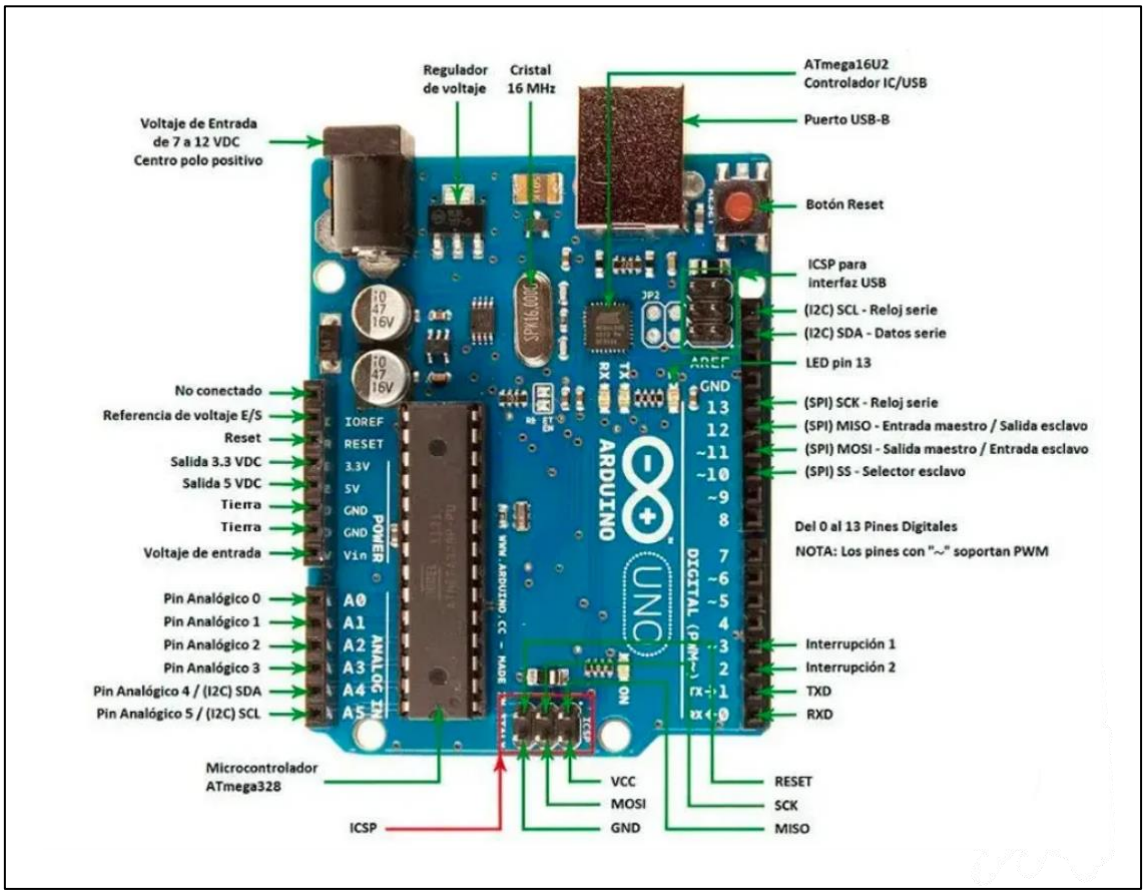
El entorno Arduino es una plataforma electrónica formada por un conjunto de **hardware y software libre**, de código abierto.

En la parte de hardware, tenemos principalmente una **placa**, a la que pueden darse una serie de instrucciones a través de un lenguaje de programación propio.

Tenemos, además, un conjunto de elementos accesorios que podemos conectar a la misma, como sensores y actuadores, así como una serie de shields que se conectan a la placa para facilitar o realizar conexiones específicas.

Dichos elementos se conectan a través de diversos pines, pudiendo hacerlo directamente o a través de elementos como cables de diversa tipología, recomendándose también el apoyo en placas ***protoboard***, que no solo organizan las diferentes conexiones, sino que facilitan también la detección de errores en las mismas.

Para gestionar la comunicación entre la placa y los elementos conectados, se hace uso de diferentes software, como Arduino IDE, basado en Processing, o algunas plataformas donde programar en bloques como **ArduinoBlocks**, o Tinkercad Circuits, pudiendo en ésta última simular también los circuitos.



## 2. LA PLACA ARDUINO

Existen diferentes tipologías y tamaños de placa, destinadas principalmente a usos varios y su adecuación a diferentes proyectos. A nivel educativo, en Castilla y León, la mayoría de los centros cuenta con "Arduino UNO", de la cual procedemos a hablar a continuación.

Para comenzar a comprender la multitud de elementos que componen esta placa, debemos comprender el funcionamiento mas básico de la misma: **recibe una información de entrada/realiza un proceso/emite una información de salida**.

Esto es muy similar al dia a día de un ser humano: **recibimos un estímulo/procesamos la información/realizamos una acción.**

Como veremos más adelante, tanto para la recepción de información del exterior como para la realización de acciones tras su procesamiento, hacemos uso de los sensores y actuadores, que pueden trabajar de forma digital o analógica.

Si se desea ampliar la información sobre las diferentes partes de la placa, se adjuntan dos enlaces (página de Arduino y vídeo de Youtube), sin embargo, en este documento, nos centraremos en los elementos indispensables para uso educativo.

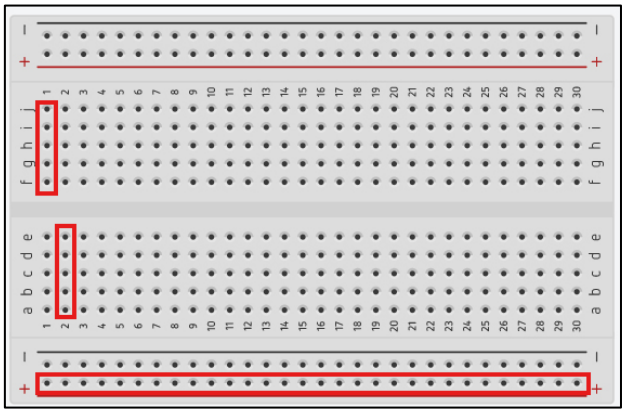
Principales elementos de la placa Arduino UNO:

- **Pines 3,3 y 5 Voltios**: salidas de voltaje que nos permiten alimentar componentes externos
- **Jack de alimentación**: permite conectar baterías (9V) en caso de necesitarse, dados proyectos demandantes de energía extra
- **USB**: comunica la placa con el ordenador, permitiendo cargar los programas, y le aporta 5 voltios, generalmente suficientes para montajes básicos.
- **GND**: pines de tierra, actúan como negativo para la alimentación.
- **Pines digitales**: las señales de entrada y de salida digitales solo tienen 2 estados, 0 o 1, 5V o 0V. Numerados del 0 al 13, algunos pueden funcionar como PWM si se desea
- **Pines PWM**: dado que no existen salidas analógicas, se emulan mediante este elemento, conocido como modulacion por ancho de pulso, comprendiendo valores entre 0 y 255.
- **Pines analógicos**: se trata de entradas únicamente, que detectan valores entre 0 y 5 V, que el procesador interpreta como valores entre 0 y 1023, teniendo por tanto un rango mucho más amplio, necesario para algunos actuadores y sensores que veremos más adelante.

## 3. LA PLACA PROTOBOARD

Se trata de un elemento auxiliar, no indispensable para el funcionamiento de los circuitos en la mayoría de los casos, pero si altamente recomendable dada la

organización que aporta a los montajes. Existiendo diferentes tamaños, todas siguen una estructura y lógica similar, que se detalla a continuación.



En este ejemplo vemos una protoboard dividida en dos partes (simétricas) por la zona gris oscuro central que la recorre horizontalmente. En cada una de las porciones encontramos otras dos zonas: la zona de positivo y negativo, y la zona de letras y números.

La línea **(+)** y la línea **(-)** se utilizan para conectar la placa a voltaje y a GND, estando todos los puntos de una misma línea conectados horizontalmente. Una vez tenemos esta conexión a la placa (donde los espacios son limitados), podemos conectar otros muchos elementos que esten conectados a la protoboard a voltaje y tierra sin preocuparnos por que no haya más pines disponibles en la placa.

La zona de letras y números sigue un patrón similar, pero con una salvedad. En esta ocasión cada línea numérica está conectada de modo horizontal, por ejemplo: f1, g1, h1, i1 y j1 están conectados entre sí, pero no conectados con la línea 2, ni con a1, b1, c1, d1 y e1. No obstante, podemos poner un cable entre dos puntos cualquiera consiguiendo así la conexión

## 4. SENSORES, ACTUADORES Y PERIFÉRICOS

### 4.1 SENSORES

Se trata de dispositivos que detectan variables ambientales de diferentes tipos, como puede ser la temperatura, y la transforma en una variable eléctrica. A grandes rasgos, nos encontramos con:

- **Sensores digitales**: detectan estados absolutos, 0 o 1, como puede ser un pulsador, que está pulsado o no está pulsado
- **Sensores analógicos**: detectan estados dentro de un rango, entre 0 y 1023, como puede ser un sensor de presión. Al igual que el pulsador, se ejerce una presión sobre una superficie, pero en este caso podemos saber ya no solo si se pulsa o no, si no cuan fuerte se está pulsando.

### 4.2 ACTUADORES

A modo general, un actuador transforma un determinado tipo de energía en una determinada acción, tarea o proceso. En el caso de Arduino, la energía utilizada es

la eléctrica, y encontramos entre otros actuadores electrónicos, eléctricos o diferentes motores.

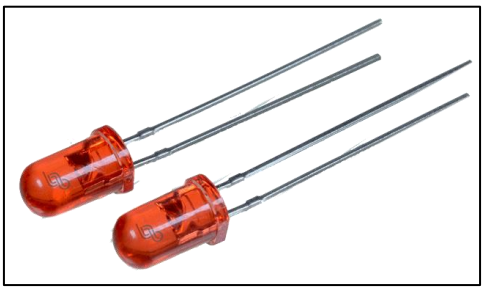
### 4.3 PERIFÉRICOS

Un tipo especial de actuadores son los periféricos. Permiten la comunicación de la placa Arduino con el exterior, como puede ser el caso de pantallas LCD, táctiles, *displays* numéricos u otros.

## 5. PRINCIPALES SENSORES Y ACTUADORES UTILIZADOS EN EDUCACIÓN

A continuación, se presentan una serie de elementos que encontramos a menudo en los proyectos educativos, junto a un código sencillo para entender su funcionamiento, tanto en bloques (realizado en Tinkercad) como en lenguaje Arduino, y una simulación de su montaje que permita observar las conexiones, las cuales también se detallan.

### 5.1 LED



El led, o diodo emisor de luz, es un componente eléctrico con dos electrodos que permite el paso de la electricidad unidireccionalmente, generando luz. Consta de dos patillas, una más larga, el Ánodo, y otra más corta, el Cátodo. Con un funcionamiento y conexión similar al que explicaremos, podríamos también programar una práctica sencilla para hacer funcionar un zumbador y un motor de corriente continua.

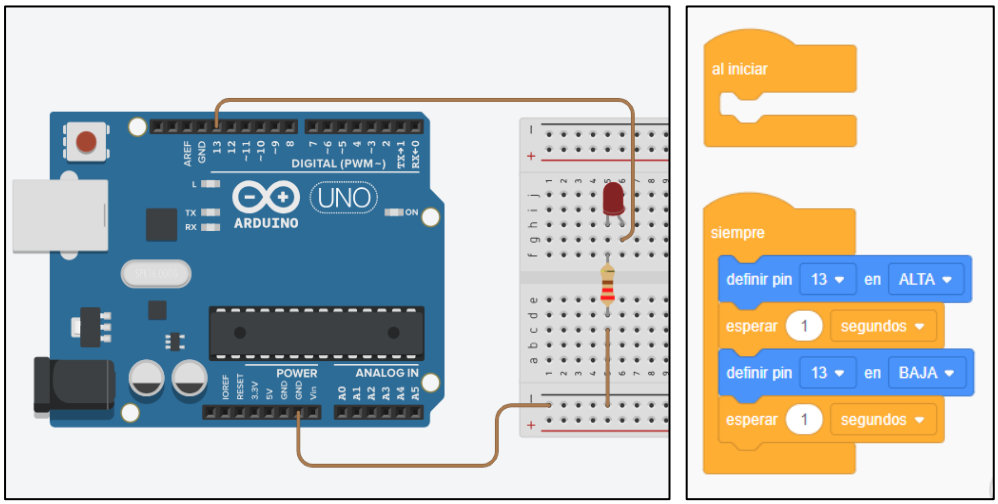
*Materiales necesarios para la práctica:*

- Arduino Uno (o cualquier otro modelo).
- LED.
- Resistencia de 220 ohmios (aproximadamente).
- Cable jumper.

*Conexión y código en bloques:*

1. Conecta el cátodo (pata más corta) del LED al pin GND (tierra) del Arduino.

2. Conecta el ánodo (pata más larga) del LED a través de una resistencia de 220 ohmios al pin digital 13 del Arduino.




En cuanto a los bloques, vemos el apartado "Al iniciar" vacío ya que, a diferencia del lenguaje en código, no debemos declarar los pines como entrada o salida, ya que el simulador lo detecta automáticamente.

Ya en el apartado siempre, tenemos la acción que se quiere realizar. Se aporta energía al pin 13 (alta), por lo que el led luce durante 1 segundo, hasta que introducimos el bloque "definir pin 13 en baja", donde deja de recibir energía y se apaga, manteniéndose apagado durante otro segundo. Al estar repitiéndose constantemente estas instrucciones, obtenemos un led que parpadea.

*Código en lenguaje Arduino:*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
int ledPin = 13;    // Introduzco la variable led en el pin 13

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

void loop() {
  digitalWrite(ledPin, HIGH);  // Encender el LED
  delay(1000);                  // Esperar 1 segundo (1000
milisegundos)

  digitalWrite(ledPin, LOW);   // Apagar el LED
  delay(1000);                  // Esperar 1 segundo (1000
milisegundos)
}
```

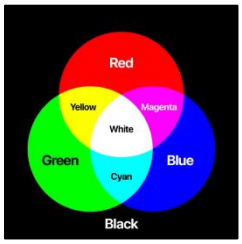
Siendo la misma estructura que en la programación de bloques, tenemos algunas variaciones. Debemos crear una variable, que es ledPin, y decirle a que pin corresponderá, en este caso al 13. Para introducir una variable usamos la función `int`.

Si recordamos, en bloques teníamos el espacio "al iniciar" vacío, pero ahora vemos su equivalente (`void setup`) conteniendo una información que nos indica que el led conectado al pin 13 funcionará como una salida. Para ello se utiliza la función `pinMode`.

Ya en el apartado `void loop`, equivalente al "siempre" de bloques, tenemos la información que se repetirá "en bucle".

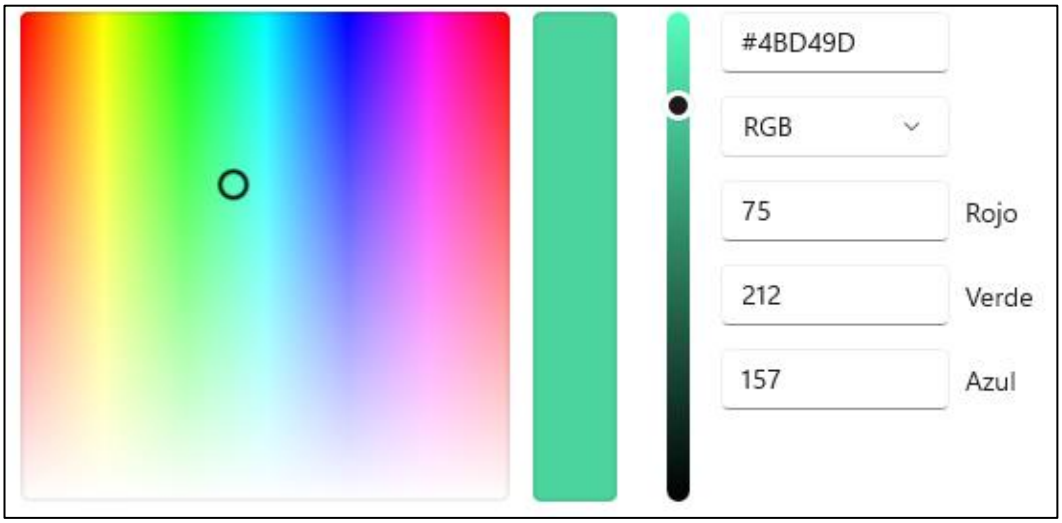
La estructura es similar a los bloques. Ponemos el led en alta (High) con la función `digitalWrite`, esperamos 1 segundo (1000ms) con la función `delay`, apagamos el led (`LOW`) con `digitalWrite` e introducimos otra espera de 1 segundo.

## 5.2 LED RGB



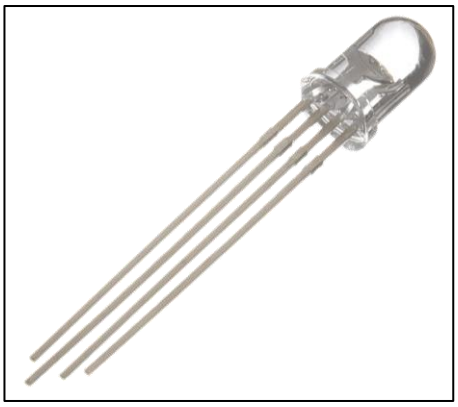

El funcionamiento de este actuador se basa en la teoría de que un color se obtiene a partir de la **suma de intensidades de los colores rojo, verde y azul**. Estos son los conocidos como colores primarios, y a partir de diferentes combinaciones de ellos obtenemos todos los demás.

Vemos en la imagen como a partir de la unión de dos colores primarios (rojo y verde) se forma un color complementario (amarillo). Si unimos los 3 primarios en la misma proporción se crea blanco, mientras que la ausencia de todos ellos genera negro.




No obstante, no solo podemos combinarlos en proporciones iguales, sino que al variar la proporción de cada color primario en la mezcla damos lugar a 16777216 colores posibles, tal y como puede observarse debajo.

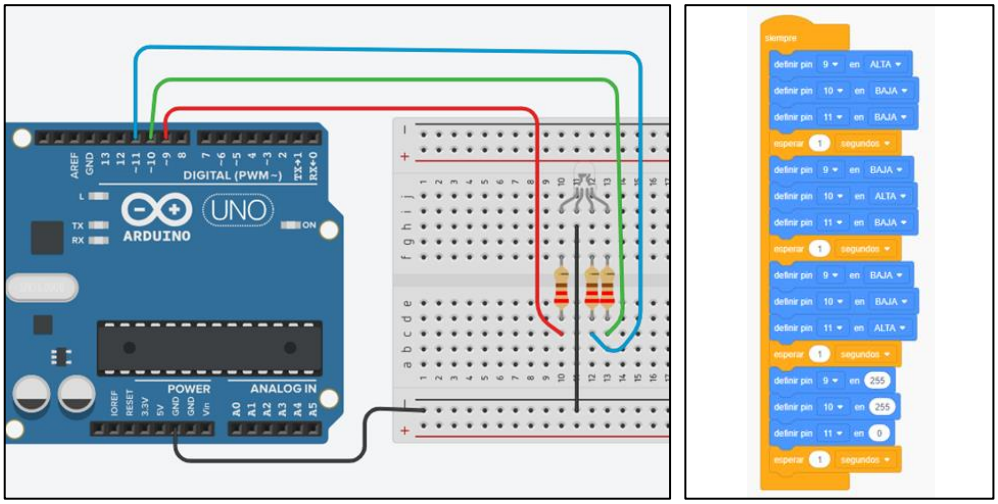
El led RGB sigue un funcionamiento similar. Como vemos en la imagen, consta de 4 patillas. La más larga es un ánodo o cátodo, dependiendo de la tipología, mientras que las otras 3 son las relativas a los colores primarios rojo, verde y azul. Si los conectamos a una salida PWM, podremos obtener hasta 255 valores para cada uno de ellos, combinándolos asi para conseguir los diferentes colores.



*Materiales necesarios:*

- Arduino Uno (o cualquier otro modelo).
- LED tricolor (RGB).
- Resistencias de 220 ohmios (aproximadamente) para cada color del LED (R, G, B).
- Cables jumper.

*Conexión y código en bloques:*




3. Conecta cada uno de los cátodos (patas más cortas) de los LED rojo (R), verde (G) y azul (B) a un pin GND (tierra) del Arduino.
4. Conecta cada uno de los ánodos (patas más largas) de los LED a través de resistencias de 220 ohmios a los pines digitales 9 (R), 10 (G) y 11 (B) del Arduino, siendo pines PWM.

Vemos en el código como en los 3 primeros grupos de bloques azules, consecutivamente tengo solo el pin 9 en alta, luego solo el 10, y el 11. Por tanto, solo el pin 9 aporta rojo en primer lugar, luego solo el 10 (verde) y posteriormente el 11 (azul). Por último, se consigue el color amarillo, pero de diferente manera, a partir del rojo y el verde. En este caso, escojo un bloque que me permite seleccionar el valor 255 (máximo), seria equivalente la instrucción "ALTA". En caso de querer un color del resto de toda la gama (que no sea color primario o complementario), bastaría con seleccionar valores entre 0 y 255 en esa ultima tipología de bloque para cada uno de los 3 pines (colores).

*Código en lenguaje Arduino:*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
int redPin = 9;    // Conectado al ánodo del LED rojo
int greenPin = 10; // Conectado al ánodo del LED verde
int bluePin = 11;  // Conectado al ánodo del LED azul
void setup() {
  pinMode(redPin, OUTPUT);    // Configurar como salida
  pinMode(greenPin, OUTPUT);  // Configurar como salida
  pinMode(bluePin, OUTPUT);   // Configurar como salida
}
void loop() {
  // Encender rojo, apagar verde y azul (para obtener rojo)
  digitalWrite(redPin, HIGH);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, LOW);
  delay(1000);  // Esperar 1 segundo

  // Encender verde, apagar rojo y azul (para obtener verde)
  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, HIGH);
  digitalWrite(bluePin, LOW);
  delay(1000);  // Esperar 1 segundo

  // Encender azul, apagar rojo y verde (para obtener azul)
  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, HIGH);
  delay(1000);  // Esperar 1 segundo

  // Encender rojo y verde, apagar azul (para obtener amarillo)
  digitalWrite(redPin, HIGH);
  digitalWrite(greenPin, HIGH);
  digitalWrite(bluePin, LOW);
  delay(1000);  // Esperar 1 segundo
}
```

La estructura del código es bastante similar. En primer lugar, declaro las variables en el pin correspondiente (debe ser PWM).

En `void setup` defino los 3 pines de los colores como salidas.

Ya en el bucle, repito la misma secuencia que en el código en bloques con las instrucciones ya conocidas `digitalWrite` y `delay`.

## 5.3 PULSADOR

Un pulsador es un botón que se utiliza en para permitir la interacción del usuario con el sistema. Al presionar el pulsador, se cierra un circuito y envía una señal al Arduino, que puede desencadenar diversas acciones, como encender un LED, activar un motor, o contar el número de veces que se ha presionado el botón. Es una forma sencilla de iniciar eventos y controlar dispositivos en proyectos de Arduino.

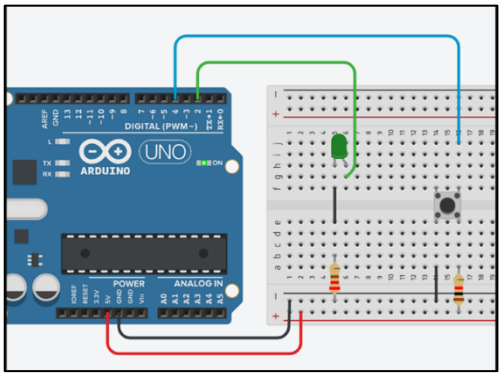
Existen dos tipos principales de conexiones para un pulsador: pull-up y pull-down. En una conexión pull-up, el pin está normalmente en estado ALTO (5V) y cambia a BAJO (0V) al presionar el pulsador. Por ejemplo, en un circuito simple con un led, el led estaría encendido puesto que el circuito está cerrado, hasta que pulso, abriendo el circuito, manteniéndose apagado el led hasta que dejo de pulsar.

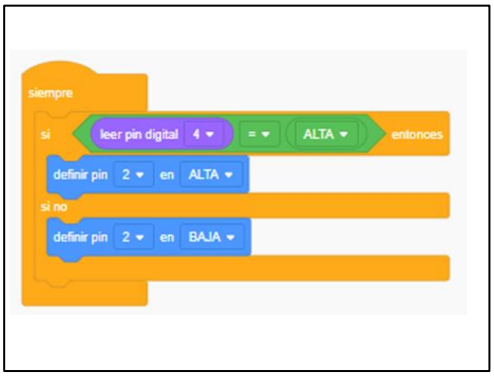
En una conexión pull-down, el pin está en estado BAJO (0V) y cambia a ALTO (5V) cuando se presiona el pulsador. En el ejemplo anterior, el led estaría apagado hasta que se pulsa el pulsador, que cierra el circuito y hace que la corriente llegue al led, encendiéndose éste.

*Materiales necesarios:*

- Arduino Uno (o cualquier otro modelo).
- LED verde.
- 1 resistencia de 220 ohmios para el LED.
- 1 resistencia de 1k ohmios para el pulsador.
- 1 pulsador.
- Cables jumper.
- Protoboard.

*Conexión y código en bloques:*






1. Conecta el ánodo (pata más larga) del LED verde al pin digital 2 del Arduino a través de una resistencia de 220 ohmios.
2. Conecta el cátodo (pata más corta) del LED a un pin GND (tierra) del Arduino.
3. Conecta un extremo del pulsador al pin digital 4 del Arduino a través de una resistencia de 1k ohmios.
4. Conecta el otro extremo del pulsador a un pin GND del Arduino.
5. Conecta el otro pin del pulsador (sin resistencia) al pin de 5V del Arduino.

No se adjunta captura del apartado “al iniciar”, ya que, al igual que en el caso anterior y en los siguientes, como dijimos, no es necesario declarar las variables utilizadas. En cuanto al código en sí, utilizamos un condicional si/entonces si no/entonces, para detectar si esta accionado el pulsador (cuando se pulsa, la lectura del pin 4 es ALTA, cuando no se pulsa, es BAJA).

A partir de aquí, cuando se detecta como pulsado (ALTA) se define el LED conectado en el pin 2 como ALTA, encendiéndose, estando apagado cuando no se detecta el pulsador accionado (BAJA).

*Código en lenguaje Arduino:*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
int pinVerde = 2;           // Pin donde está conectado el LED verde
int pulsador = 4;           // Pin donde está conectado el pulsador
int estadoDelPulsador = 0;  // Variable para almacenar el estado del
pulsador

void setup() {
  pinMode(pinVerde, OUTPUT);  // Configura el pin del LED como salida
  pinMode(pinVerde, INPUT);   // Configura el pin del pulsador como
entrada
}
```

```
void loop() {
  // Leer el estado del pulsador
  estadoDelPulsador = digitalRead(pulsador);

  // Si el pulsador está presionado, enciende el LED
  if (estadoDelPulsador == HIGH) {
    digitalWrite(pinVerde, HIGH);   // Enciende el LED
  } else {
    digitalWrite(pinVerde, LOW);    // Apaga el LED
  }
}
```

Como novedad respecto a la práctica anterior, tenemos una variable creada para el estado del pulsador, donde almacenaremos (con `digitalrEAD`) si está pulsado (sería 1 o `HIGH`) o no (sería 0 o `LOW`). Además, se utiliza `if/else` (si/si no) para encender el led (con `digitalWrite`) si el pulsador está o no pulsado. Utilizamos además el operador ==, equivalente a igual. Si el estado es igual a `HIGH` (1), encendemos el led con `digitalWrite` (pinVerde, `HIGH`), y si no, no se enciende, con `digitalWrite` (pinVerde, `LOW`).

## 5.4 SERVOMOTOR


Se trata de un motor cuyo ángulo de giro es entre -90º y 90º, lo que suponen 180º efectivos, permitiendo ajustar su posición de manera precisa. Utiliza fuerza para mantener o modificar su posición, por lo que nos permite mover o sostener objetos. A él se conectan 3 cables, uno a **voltaje**, otro a **tierra**, y el tercero a un **pin PWM** (modulacion por ancho de pulsos).

La duración del pulso (en microsegundos) determina el ángulo hacia el que se mueve el motor.

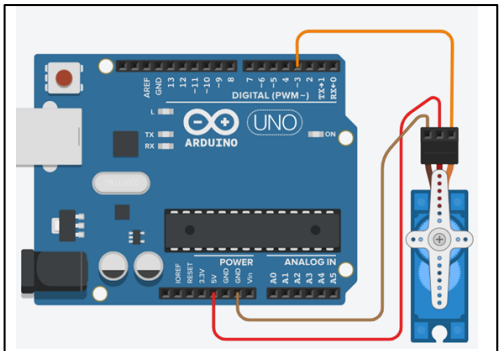
- Un pulso corto (alrededor de 1 ms) mueve el motor a 0°.
- Un pulso largo (alrededor de 2 ms) lo mueve a 180°.
- Un pulso intermedio lo posiciona en algún punto entre 0° y 180°.

*Materiales necesarios:*

- Arduino Uno (u otro modelo compatible).
- Servomotor (cualquier modelo estándar de servomotor servirá).

- Fuente de alimentación (opcional, dependiendo de la carga del servomotor).

*Conexión y código en bloques:*





1. Conecta la señal (generalmente el cable naranja o amarillo) del servomotor al pin 9 del Arduino.
2. Conecta el cable de alimentación del servomotor al pin de alimentación del Arduino (5V).
3. Conecta el cable de tierra (generalmente negro o marrón) del servomotor al pin GND del Arduino.

El funcionamiento es muy simple, al estar conectado a un pin PWM nos permite seleccionar directamente el grado en el que queremos posicionarlo.

Cuando programamos en bloques (como en plataformas educativas tipo Scratch o Tinkercad), el control del servomotor es mucho más fácil, ya que no necesitamos preocuparnos por detalles técnicos como convertir los valores del PWM (0-255) a grados (0-180). Esto es porque el bloque ya tiene esa conversión predefinida. Así, solo hay que elegir el ángulo deseado (por ejemplo, 10 grados) y el programa hace todo lo necesario para mover el servomotor.

En lenguaje Arduino puro, en como veremos ahora, necesitamos incluir algunas instrucciones más.

*Código en lenguaje Arduino:*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
#include <Servo.h> // Incluimos la librería Servo

Servo servomotor; // Declaramos el servomotor
```

```
void setup() {
  servomotor.attach(3); // Conectamos el servomotor al pin 3
}

void loop() {
  servomotor.write(10); // Mover el servomotor a 10 grados
  delay(1000);          // Esperar 1 segundo (1000 milisegundos)

  servomotor.write(100); // Mover el servomotor a 100 grados
  delay(1000);           // Esperar 1 segundo
}
```

En este tipo de programación necesitamos incluir una librería para trabajar con el servomotor, la cual nos ayudará a suplir, en parte, el trabajo que ya viene predefinido en las plataformas de bloques.

Una librería, en programación, es como un conjunto de herramientas ya hechas, que nos ahorran mucho trabajo. En lugar de escribir todo el código desde cero, simplemente llamamos a la librería y usamos sus funciones. Por ejemplo, en el caso de los servomotores, usamos la librería `<Servo.h>`, que se encarga, entre otras tareas, de:

Convertir automáticamente los valores de PWM que usa Arduino (entre 0 y 255) a ángulos de servomotor (entre 0° y 180°).

Controlar el servo con comandos sencillos como `servomotor.write(90)`, donde el número representa los grados del servo (en este caso 90º).

Esto significa que no tenemos que programar cómo se convierte el PWM a grados, porque la librería ya lo tiene todo resuelto. Solo incluimos la línea `#include <Servo.h>` al inicio del código y ya podemos mover el servomotor fácilmente.

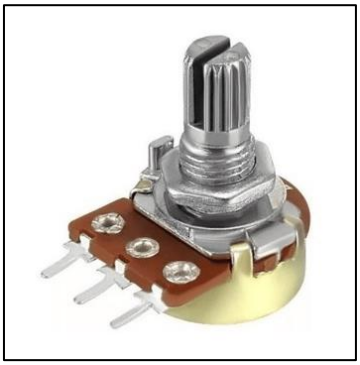
Posteriormente, observamos la instrucción “Servo servomotor”. Esta crea un objeto llamado servomotor de la clase `servo`. La clase `servo`, al igual que la librería, contiene instrucciones para controlar servos. El “servomotor” será el nombre que usaremos para controlar nuestro motor específico.

Entrando ya en `void setup()`, nos encontramos con la instrucción `servomotor.attach(3)`, que nos sirve para decir a que pin va conectado el servomotor. Servomotor es el nombre que le hemos puesto, y 3 es el pin al que va conectado. Si hubiésemos nombrado al servo como “miservo”, con la instrucción Servo miservo, deberíamos declarar el pin asi: `miservo.attach(3)`.

En cuando al `void loop()`, encontramos la instrucción `servomotor.write(90)`, que nos permite elegir el ángulo (10º en este caso) al que se desplaza el servomotor. Nuevamente, si nuestro servomotor se llamase “miservo”, la instrucción sería `miservo.write(90);`.

El funcionamiento es simple: un bucle en el que se desplaza a 10º, espera 1 segundo, se desplaza a 100º, espera 1 segundo.

## 5.5 POTENCIÓMETRO



Se trata de una resistencia variable, que propone una dificultad mayor o menor al paso de la corriente eléctrica, permitiendo controlar la cantidad de la misma que pasa por un circuito. Para entenderlo, es como un control de volumen de una radio, o el control de la luz que emite la lampara del salón en los casos que cuentan con un mando en forma de ruleta que puedes girar.

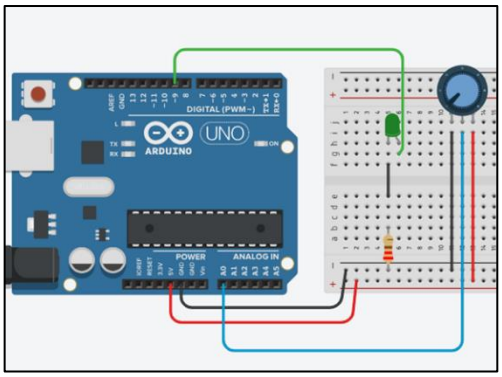
Una vez detectamos el grado de giro que se ha hecho, Arduino lee el voltaje a través de un pin analógico, y puede usarse ese dato.

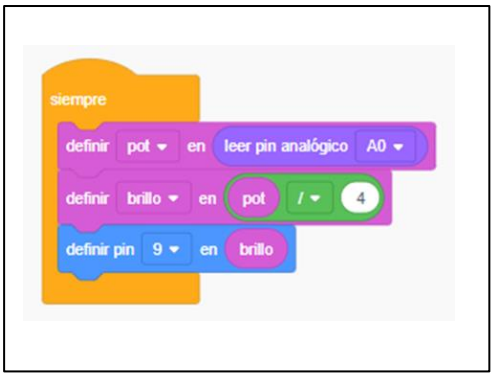
*Materiales necesarios:*

- Arduino Uno (u otro modelo compatible).
- LED.
- Resistencia de 220 ohmios (aproximadamente).
- Potenciómetro de 10k ohmios.
- Cables jumper.

*Conexión y código en bloques:*

1. Conecta un extremo del potenciómetro a 5V del Arduino.
2. Conecta el otro extremo del potenciómetro a GND del Arduino.
3. Conecta el terminal central (wiper) del potenciómetro al pin A0 del Arduino.
4. Conecta el LED a través de la resistencia de 220 ohmios al pin 9 del Arduino (o cualquier otro pin digital que prefieras).






En primer lugar se crean dos variables, pot y brillo. Pot almacena el valor leído por el pin A0, al que está conectado el potenciómetro. Brillo almacena el valor de la lectura del A0 (valores entre 0 y 1023) dividido por 4 (para obtener el rango 0-255=.

Ese brillo con valores entre 0 y 255 es el valor que puede tomar el led (puesto que esta conectado a un pin PWM). De este modo, un led, que en un pin digital normal solo puede estar en estados de 0 (apagado) o 1 (máximo brillo), al conectarse a un pin PWM puede tener brillos con hasta 256 valores, incluidos el 0 (apagado) y el 255 (brillo máximo).

*Código en lenguaje Arduino*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
int pot = A0;       // Pin analógico conectado al potenciómetro
int led = 9;        // Pin digital conectado al LED
int brillo = 0;     // Variable para almacenar el brillo del LED

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

void loop() {
  // Leer el valor del potenciómetro (entre 0 y 1023)
  brillo = analogRead(pot) / 4;  // Convertir el valor a un rango de 0
a 255

  // Establecer el brillo del LED según el valor leído del
potenciómetro
  analogWrite(led, brillo);

  // Pequeño retardo para estabilizar el valor leído y evitar cambios
rápidos
  delay(10);
}
```

El código comienza definiendo las variables `pot` y `led`, en el pin A0 (analógico) y el pin 9 (digital PWM) respectivamente. Estos serán los elementos físicos conectados a la placa.

Por otro lado, creo la variable brillo, a la que podríamos asignar cualquier valor (de momento) aunque comúnmente se usa 0.

Ya en `void setup()`, declaramos el led como una entrada.

En `void loop()`, comenzamos realizando dos acciones en el mismo tiempo con la instrucción `brillo = analogRead(pot) / 4`.

Leemos el `pot` (pin A0 analogico) con `analogRead` obteniendo un valor entre 0-1023.

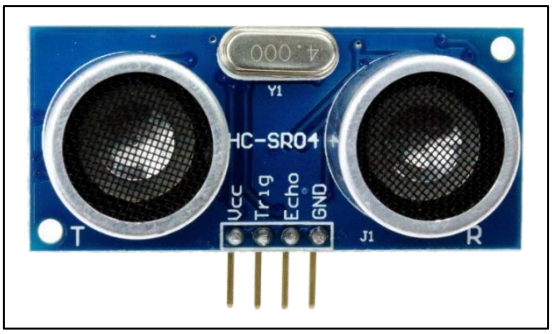
Dividimos por 4 para transformar 0-1023 del analógico al 0-255 del PWM

Seguidamente, le decimos al led el valor del brillo que debe mostrar, con `analogWrite(led, brillo)`, donde led es el pin 9 y brillo es el valor 0-255 almacenado en la variable.

Por último, un pequeño `delay` de 10 milisegundos para ir actualizando el brillo conforme cambia la lectura del potenciómetro.

## 5.6 SENSOR ULTRASÓNICO HC-SR04

Un dispositivo que puede medir qué tan lejos está un objeto. Funciona enviando ondas sonoras (como las de un murciélago) y luego mide cuánto tiempo tarda en regresar el eco de esas ondas cuando rebotan en un objeto, pudiendo saber a qué distancia se encuentra este.




La lógica detrás de este sensor es la siguiente:

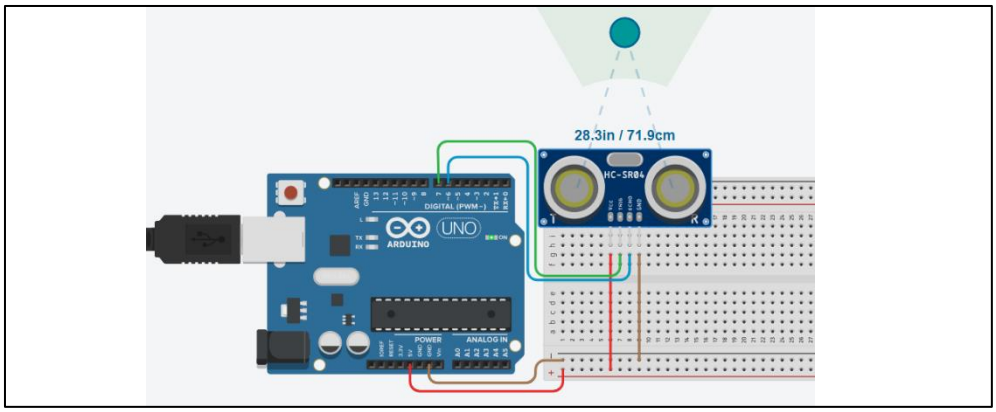
- Envía un sonido: El sensor tiene dos "ojos". Uno de esos ojos, llamado "Trig" (de "trigger"), envía una onda sonora muy rápida que nosotros no podemos escuchar porque es un sonido "ultrasónico".
- Escucha el eco: El otro ojo, llamado "Echo", está esperando a que esa onda rebote en un objeto y regrese. Cuando la onda vuelve, el sensor "escucha" el eco.
- Calcula la distancia: El sensor mide cuánto tiempo tardó el eco en volver. Como sabe a qué velocidad viajan las ondas sonoras, puede calcular la distancia del objeto. Es como cuando gritas en una cueva y esperas a escuchar el eco para saber cuán lejos está la pared.

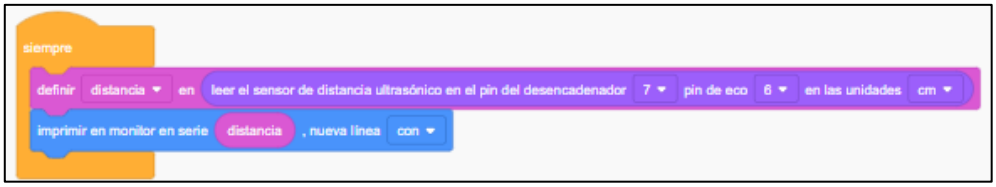
*Materiales necesarios:*

- Arduino Uno (u otro modelo compatible).
- Sensor de ultrasonido HC-SR04.
- Cables jumper.

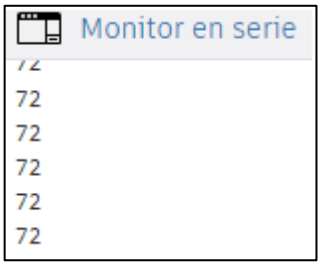
*Conexión y código en bloques:*

1. Conecta el pin VCC del sensor HC-SR04 a 5V del Arduino.
2. Conecta el pin GND del sensor HC-SR04 a GND del Arduino.
3. Conecta el pin TRIG del sensor HC-SR04 al pin 7 del Arduino.
4. Conecta el pin ECHO del sensor HC-SR04 al pin 6 del Arduino.




En este código crearemos una variable llamada "distancia", y la haremos corresponder a la lectura que nos aporte el bloque morado. Es importante que en primer lugar se introduzca el pin del TRIG (en nuestro ejemplo es el pin 7) y en el segundo selector del bloque morado se introduzca el pin correspondiente al ECHO (en este caso el 6).




Para visualizar la distancia (en el simulador es el punto verde), podemos verla en la propia simulación o pedir al programa que la imprima en el "monitor serie", que es una especie de pantalla donde podemos ver los datos que previamente hayamos mandado mostrar.

*Código en lenguaje Arduino:*

Nota: para comprender los conceptos de este apartado es conveniente leer la Parte II de este documento (Lenguaje Arduino).

```
int trigPin = 7;  // Pin TRIG del sensor conectado al pin 7 del Arduino
int echoPin = 6;  // Pin ECHO del sensor conectado al pin 6 del Arduino
long duracion;    // Variable para almacenar la duración del pulso
(usamos "long" porque el valor que devuelve pulseIn puede ser mayor que
lo que puede almacenar un int)
int distancia;    // Variable para almacenar la distancia medida

void setup() {
  Serial.begin(9600);  // Iniciar comunicación serial a 9600 baudios
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  // Generar un pulso corto en el pin TRIG
  digitalWrite(trigPin, LOW);
  delay(2);                // Esperar 2 milisegundos
  digitalWrite(trigPin, HIGH);
  delay(10);               // Esperar 10 milisegundos
  digitalWrite(trigPin, LOW);

  // Medir la duración del pulso ECHO
  duracion = pulseIn(echoPin, HIGH);

  // Calcular la distancia en centímetros (la velocidad del sonido es
343 m/s)
  distancia = duracion * 0.0343 / 2;

  // Mostrar la distancia medida en el monitor serial
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");

  delay(1000);  //
potenciómetro
  analogWrite(led, brillo);

  // Pequeño retardo para estabilizar el valor leído y evitar cambios
rápidos
  delay(10);
}
```

Al igual que sucedía en otros montajes, el código escrito se complica un poco, pero son aspectos que responden a la lógica.

En primer lugar, asignamos las variables `echoPin` y `trigPin` a los pines 6 y 7. Creamos también dos variables que necesitaremos más adelante, distancia (con `int`) y duración (con `long` pues permite almacenar valores mayores*).

Ya en `void setup()`, iniciamos el puerto serie (para luego mandar datos al monitor serie) con `Serial.begin(9600)`, siendo esos 9600 baudios la velocidad a la que se transmiten los datos (será siempre así para este sensor). Declaramos también el trigger como salida (manda el eco) y el echo como entrada (pues recoge el eco).

Continuando en `void loop()`, necesitamos comenzar sin enviar ecos, por lo que el trigger estará apagado → `digitalWrite(trigPin, LOW)`. Tras una breve espera (`delay(2)`), mandamos un eco `digitalWrite(trigPin, HIGH)` y tras otra breve espera (`delay(2)`), lo apagamos de nuevo `digitalWrite(trigPin, LOW)`. Se ha enviado por tanto el eco, ahora habrá que ver cuanto tarda en rebotar en el objeto y regresar al sensor.

Para ello hacemos uso de la formula matemática que relaciona la velocidad, la distancia y el tiempo. Como sabemos la velocidad (en este caso nos la da el fabricante del sensor, y es ese valor que vemos de `0.0343 / 2`, únicamente hay que multiplicarla por la duración (que es el tiempo que el sensor mide que ha tardado en ir y volver la onda) para obtener la distancia.

Una vez calculada, únicamente faltaría enviarla al monitor serie para que este la muestre, para lo que usamos los siguientes comandos:

`Serial.print("Distancia: ")` imprime literalmente la palabra distancia:

`Serial.print(distancia)` imprime el valor almacenado en la variable distancia.

`Serial.println("cm")` imprime literalmente la palabra centímetro, precedido de un espacio, en un salto de línea (pues ha usado `println` en lugar de `print`).

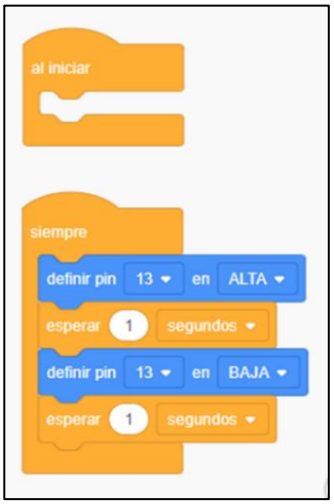
Por último, un delay de un segundo hasta que se realice la próxima medición. Por tanto, en el monitor serie aparecerá un dato que actualice la distancia del objeto cada segundo.

# PARTE II. Lenguaje Arduino

## 6. ESTRUCTURA BÁSICA DE UN PROGRAMA EN ARDUINO

Se compone de, como mínimo, dos partes. Estas partes contienen líneas de texto o bloques que contienen instrucciones, declaraciones o funciones.

```
1   int ledPin=13;
2   void setup()
3   {
4     pinMode(13, OUTPUT);
5   }
6
7   void loop()
8   {
9     digitalWrite(13, HIGH);
10    delay(1000); // Wait for 1000 millisecond(s)
11    digitalWrite(13, LOW);
12    delay(1000); // Wait for 1000 millisecond(s)
13  }
```

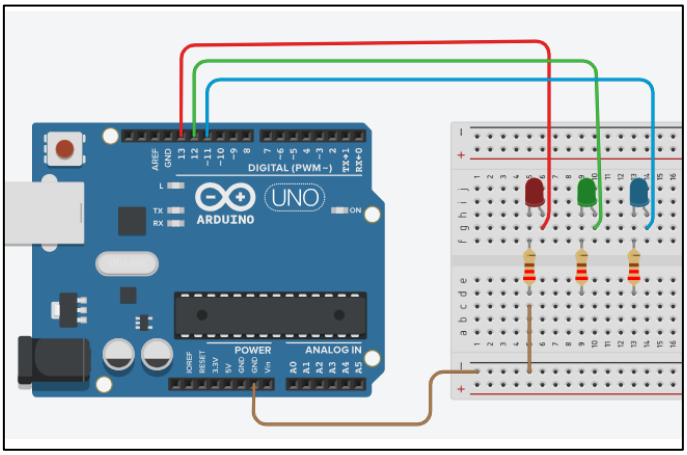



**Void setup()** , al iniciar, es la parte que recoge la información necesaria. Por ejemplo, se declaran los pines como entrada **(INPUT)** o salida **(OUTPUT).** En el caso del ejemplo, tenemos un led conectado al pin 13, el cual declaramos como salida **(OUTPUT).**

**Void loop()** es la parte que recoge el programa en si, que se ejecuta cíclicamente (loop=bucle). En este bucle vemos como se repiten instrucciones para dar corriente al pin 13 (**alta**), y que se encienda el led, y retirar la corriente, apagándose (**baja**). El resultado es un parpadeo del led.

Si bien las funciones anteriores son indispensables, la **función de configuración** (líneas antes del void setup, en este caso la 1) debe contener la declaración de las variables. En este caso, no es estrictamente necesario, ya que puedo decir en el loop que el pin 13 se ponga como **LOW** o como **HIGH**.

No obstante, imagina que tienes varios sensores/actuadores conectados, por ejemplo, un led rojo, uno azul y otro verde en los pines 13, 12 y 11. En este caso, es mas fácil declarar variables, antes del void setup. Por ejemplo, **ledRojo** podría ser el pin 13, **ledVerde** el pin 12 y **ledAzul** el pin 11.



Para crear esas variables, se utiliza la función **int+nombre=pin**, por ejemplo, **int ledRojo=13**.

Así, en el void setup, en lugar de tener **pinMode(13,OUTPUT)** podría tener **pinMode(ledRojo, OUTPUT)**, lo que ayuda a comprender el código y evitar confusiones.

Del mismo modo sucede en el void loop, podría tener **digitalWrite (ledRojo, HIGH)** en lugar de **digitalWrite (13, HIGH)**.

## 7. INSTRUCCIONES HABITUALES EN ARDUINO (NIVEL INICIAL)

**Punto y coma (;)** se utiliza para separar instrucciones dentro de los grupos de configuración, void setup y void loop. Omitirlas dará lugar a un error en el programa.

```
1   int ledPin=13;
```

**Llaves { }** sirven para marcar el inicio y el final de un bloque de instrucciones, ya sean void setup y void loop o conjuntos de instrucciones que van dentro de los mismos. Para cada llave que abre un conjunto de instrucciones, debe haber otra que lo cierra, de lo contrario dará lugar a error

```
2   void setup()
3   {
4     pinMode(13, OUTPUT);
5   }
```

**Comentarios //** sirve para aclarar una línea de código sin que varie el programa.

```
delay(1000); // Wait for 1000 millisecond(s)
```

**Int motor=2** crea una variable de valor exacto

**Float motor=2.2** crea una variable de valor decimal

**pinMode (13, INPUT)** define el pin 13 como una entrada, se declara en void setup.

**pinMode (13, OUTPUT)** define el pin 13 como una salida, se declara en void setup.

**digitalWrite (13, HIGH)** hace pasar la corriente por el pin 13, se incluye en void loop.

**digitalWrite (13, LOW)** retira la corriente del pin 13, se incluye en void loop.

**delay (1000)** introduce una espera de 1000 milisegundos, equivalente a 1 segundo. Los tiempos dentro del paréntesis deben ir expresados en milisegundos. se incluye en void loop.

```
delay(1000); // Wait for 1000 millisecond(s)
```

**valor=digitalRead(13)** asigna a la variable valor la lectura digital obtenida del pin 13. se incluye en void loop. Imaginemos que en ese pin 13 tenemos un pulsador. La variable estado pulsador, que tendrá solo valores 0 y 1, dependerá de si este está pulsado o no, por tanto tendremos: estadopulsador=digitalRead(13), lo que almacenaría en la variable "estadopulsador" el valor 0 o 1.

**valor=analogRead(13)** asigna a la variable valor la lectura analógica obtenida del pin 13. se incluye en void loop. Imaginemos que en el pin 13 tenemos un sensor de temperatura. La variable temperatura, que tendrá un rango de valores, se considera una

entrada analógica, por lo que se utiliza esta función, por ejemplo: temperatura=analogRead(13), almacenando en la variable "temperatura" valores dentro del rango medido.

## IMPRESIÓN EN PUERTO SERIE

**Serial.print** permite escribir caracteres en el monitor serie (herramientas>monitor serie), que es una especie de monitor con el que cuentan algunos softwares como Arduino IDE. El texto requerido debe ir entre comillas si es un texto estático, se incluye en void loop.

```
Serial.print("mensaje"); //muestra en el monitor serie la palabra mensaje
```

No obstante, también permite mostrar el valor de una variable. Imaginemos que tenemos almacenado en la variable temperatura el valor 10.

```
temperatura=10;
Serial.print("temperatura"); muestra temperatura
```

Si queremos mostrar un texto compuesto para decir "la temperatura es X", donde X es un valor que varia, podemos introducir:

```
temperatura=10;
Serial.print("la temperatura es");
Serial.print(temperatura);
```

En este caso, el mensaje que obtenemos es "la temperatura es 10".

**Serial.println** nos permite introducir mensajes en una nueva línea colocada debajo de la anterior, o incluso introducir una línea en blanco. se incluye en void loop. En el caso de la derecha el mensaje que se obtiene es:

```
Serial.print(23);
Serial.println(24);
```

23

24

**Serial.begin** indispensable para poder usar **Serial.Print** y **Serial.println**. Inicia la comunicación con el puerto serie, siendo necesario indicar también la velocidad de transmisión de datos, frecuentemente 9600 baudios (símbolos por segundo). Debe inicializarse en el apartado void setup.

```
void setup()
{
 Serial.begin(9600);
}
void loop()
{
    Serial.print("Hola Mundo");
}
}
```

## 8. INSTRUCCIONES HABITUALES EN ARDUINO (NIVEL MEDIO)

### BUCLE IF

Se utiliza para probar si una determinada condición se ha cumplido, y en consecuencia, ejecutar una instrucción. Se incluye dentro de void loop. Podemos aquí utilizar diferentes **operadores**, < menor que, > mayor que, == igual, != no igual.

```
if (unaVariable operador valor)
{
 ejecutaInstrucciones;
 }
```

En un ejemplo del termómetro anterior, si la variable temperatura supera los 20 grados, encienda un led rojo.

```
if (temperatura > 20)
{
 digitalWrite (ledRojo, HIGH);
 }
```

### BUCLE IF/ELSE

Partiendo de la idea anterior, permite declarar instrucciones si no se cumple la condición. Si se cumple, se ejecutarán las condiciones bajo "if", mientras que si no se cumplen se ejecutarán las condiciones bajo el "else". Se incluye dentro de void loop.

```
if (inputPin == HIGH)
{
 instruccionesA;
}
Else
{
 instruccionesB;
}
```

Siguiendo el ejemplo anterior de la temperatura en el led, podría encender un led verde si la temperatura es igual o inferior a 20 grados

```
if (temperatura > 20)
{
 digitalWrite (ledRojo, HIGH);
 }
else
{
 digitalWrite (ledVerde, HIGH);
}
```

Else puede ir precedido de otra condición de manera que se pueden establecer varios "else". Sin embargo sólo uno se llevará a cabo dependiendo de la condición. Podria usarse por ejemplo para establecer 4 rangos diferentes de temperatura en los ejemplos anteriores, encendiendo 4 colores distintos de led.

```
if (inputPin < 500)
{
instruccionesA;
}
else if (inputPin >= 1000)
{
instruccionesB;
}
 else
{
instruccionesC;
}
```

## BUCLE WHILE

Se trata de un bucle que se ejecuta constantemente mientras que se cumpla la condición colocada entre paréntesis en la cabecera del bucle, y sólo se abandonará el bucle cuando esta deje de cumplirse. Se incluye dentro de void loop.

```
while (unaVariable ?? valor)
{
Instrucciones;
}
```

Imaginemos un ejemplo donde queremos pedir un numero al usuario, y este debe ser positivo, no pudiendo seguir el programa hasta que el número introducido sea positivo. La condición que crearemos es "mientras que el numero introducido sea menor o igual a 0, volver a pedir el número"

```
num=número introducido por el usuario;
while (num<=0)
{
num=introduce un número positivo;
}
```

## BUCLE FOR

Se utiliza para repetir un bloque de instrucciones un numero determinado de veces, volviendo a comprobarse la instrucción cada vez que se ejecuta. Muy útil, por ejemplo, en contadores.

Debe introducirse un valor inicial, un valor final o condición, y una expresión que explica como avanzará el contador. Imaginemos un contador cuyo valor inicial es 0, su valor final o condición es "hasta que llegue a 20", y avance de 1 en 1, es decir, se va a ejecutar 20 veces la

```
for (inicialización; condición;
incremento)
{
ejecutaInstrucciones;
}
```

instrucción que va dentro del bloque. Básicamente, el funcionamiento es muy similar al “repetir X veces” de scratch.

En el caso del incremento, las mas utilizadas son las siguientes:

- i++ incrementa 1
- i--decrementa 1

A modo de ejemplo, se muestra un código donde tenemos un zumbador (pin 13) que va a emitir 10 pitidos discontinuamente.

```
for (int i=0; i<20; i++)
{
digitalWrite(13, HIGH);
delay(500);
digitalWrite(13, LOW);
delay(500);
}
```