Canarias avanza con Europa

Gobierno de Canarias

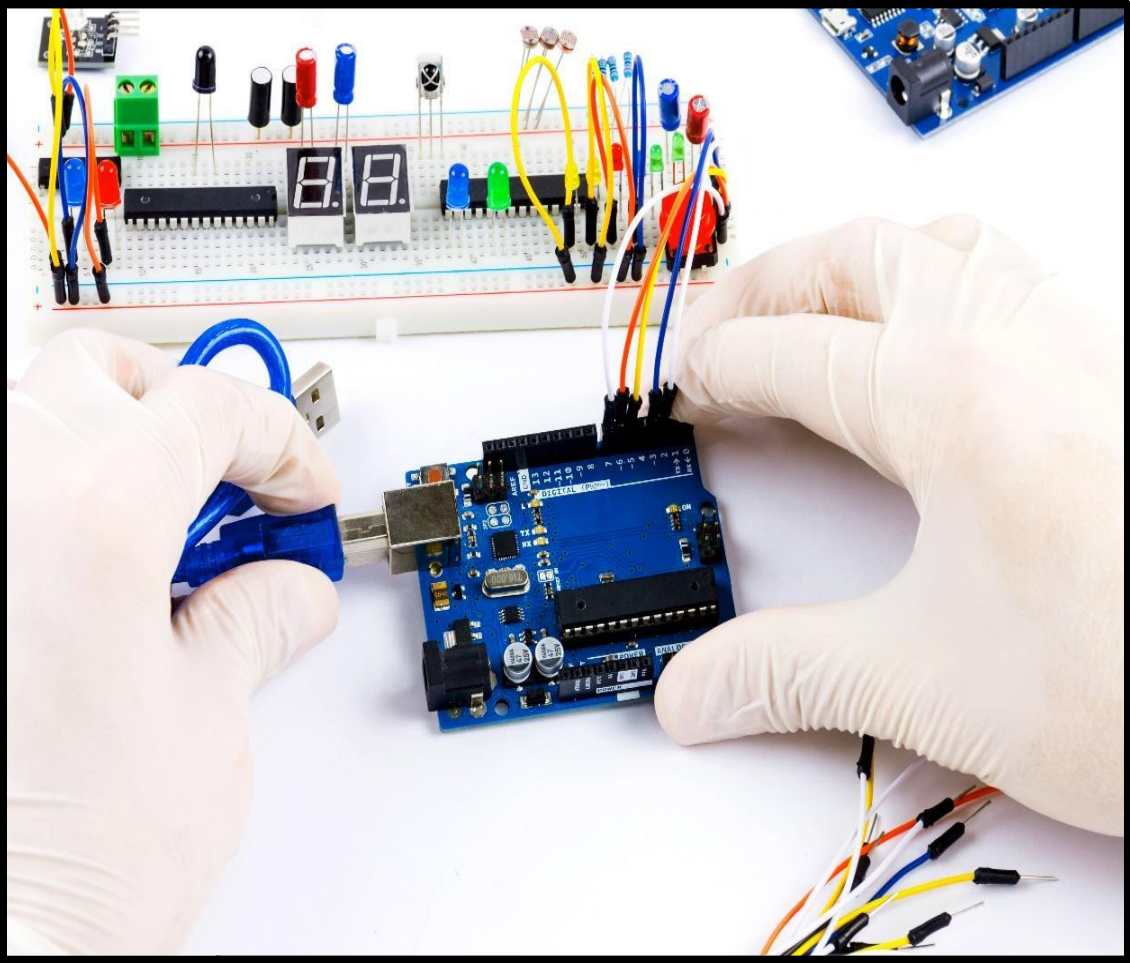
Consejería de Economía, Conocimiento y Empleo
Agencia Canaria de Investigación, Innovación y Sociedad de la Información

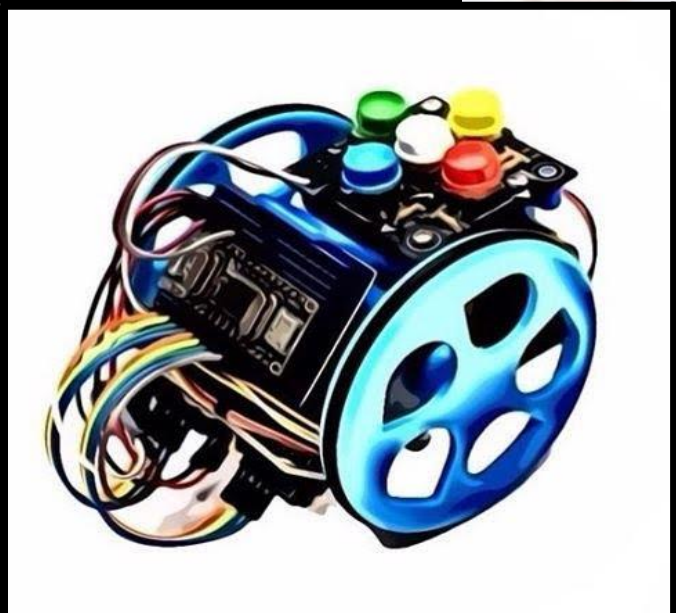
Fondo Europeo de Desarrollo Regional

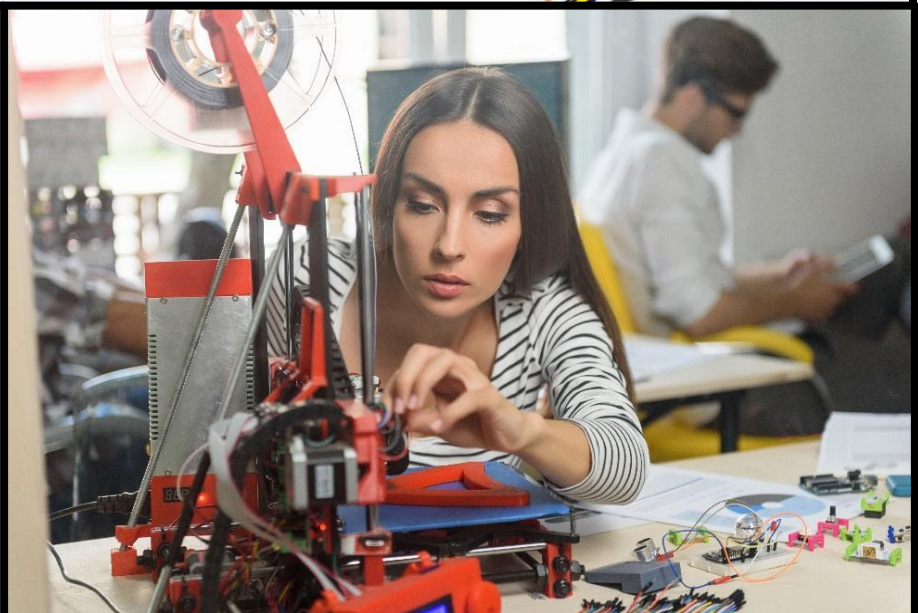
Unión Europea

VALLE STEAM

Fundamentos de Arduino







Índice

1. Datos generales

- Título del curso: Fundamentos de Arduino (EAP-01)
- Organiza: Asociación Educativa y Tecnológica Valle_Steam
- Docente: Augusto Samuel Hernández Martín
- Modalidad: Presencial
- Duración total: 6 horas
- Fecha: 3/12/2021 y 9/12/2021
- Mínimo de plazas: 20

2. Fundamentos de Arduino

2.1. ¿Qué es Arduino?

Arduino es una marca de placas de desarrollo Open Source y Open Hardware. La primera placa de Arduino fue desarrollada en el año 2005 en el Instituto IVREA (IDII), en Ivrea (Italia), con el propósito de crear una herramienta de bajo costo y fácil de usar, que permitiera a alumnos y personas sin altos conocimientos técnicos desarrollar todo tipo de proyectos de electrónica, robótica y programación. La placa alberga un microcontrolador programable y los circuitos necesarios para su correcto funcionamiento y programación. En la siguiente imagen se observan diversos modelos de placas Arduino, ordenados según tamaño y potencia.

2.2. ¿Qué es un microcontrolador

Un microcontrolador es un circuito integrado que se puede programar, es decir, que puede ejecutar órdenes que estén almacenadas en su memoria. Su funcionamiento se puede asemejar a una CPU de un ordenador y cuenta con partes similares: Unidad Central de Procesos, memoria y dispositivos de Entrada y Salida.

Existen diversos fabricantes de microcontroladores como Atmel, PIC, etc. Arduino monta en sus principales modelos (Arduino UNO y NANO) el microcontrolador de la familia ATmega 328 del fabricante Atmel.

2.3. ¿Por qué es tan popular?

Arduino es una de las tecnologías que más ha hecho posible el acercamiento de ciertas materias complejas como la electrónica y la programación de microcontroladores a personas de todas las edades y sin conocimientos específicos, de forma que lo que antes solo podía estar en mano de ingenieros, investigadores y en general de empresas y profesionales cualificados, ahora ha pasado a ser de dominio público.

Las características más relevantes de Arduino que han sido causa de su alta popularidad e influencia, se resumen en los siguientes atributos:

1. **FÁCIL DE APRENDER**
2. **FÁCIL DE USAR**
3. **OPEN-SOURCE**
4. **COMPATIBLE CON MUCHAS TECNOLOGÍAS**
5. **COMUNIDAD INMENSA**
6. **POSIBILIDADES INFINITAS**
7. **DIVERTIDO**
8. **MUY BARATO**

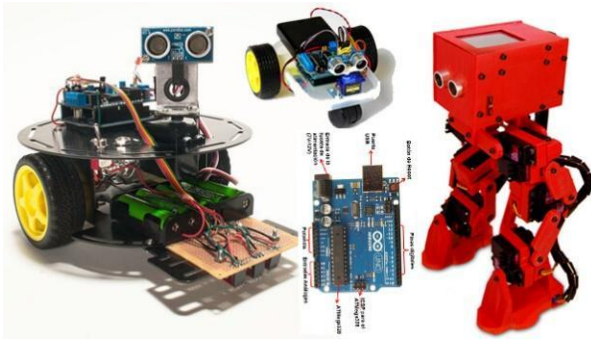
Por todo esto, Arduino ha logrado atraer una gran avalancha de usuarios interesados por el desarrollo de proyectos y la creación de tecnología: aficionados, makers, profesores, jóvenes, niños, e incluso empresas y profesionales que han visto en Arduino

y otras tecnologías paralelas, la oportunidad de desarrollar nuevos productos de forma mucho más rápida y económica.

2.4. ¿Qué tipo de proyectos se pueden hacer?

Tal vez sea mucho más rápido explicar lo que no se puede hacer que lo que sí se puede hacer con Arduino, dado que las posibilidades de esta placa de desarrollo son literalmente infinitas, especialmente teniendo en cuenta la infinidad de dispositivos y módulos electrónicos con los que Arduino es compatible y con los que podemos desarrollar proyecto para aplicaciones muy diversas.

No obstante y para comprender mejor los usos y aplicaciones de esta herramienta, de manera muy resumida podríamos agrupar los diferentes tipos de proyectos más populares de Arduino en las siguientes temáticas:

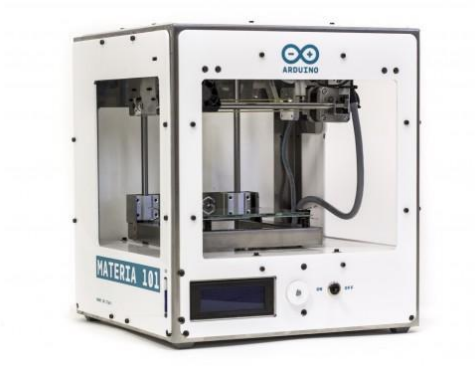


ROBÓTICA MÓVIL

ROBÓTICA INDUSTRIAL

DOMÓTICA

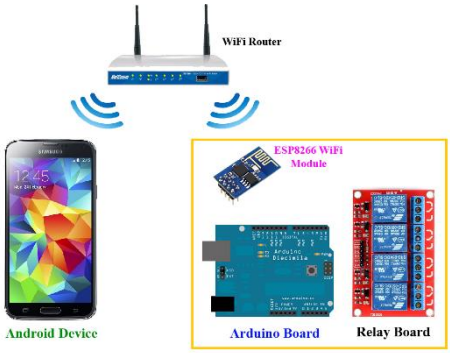
DRONES



FABRICACIÓN DIGITAL

ROPA INTELIGENTE

IMAGEN Y SONIDO

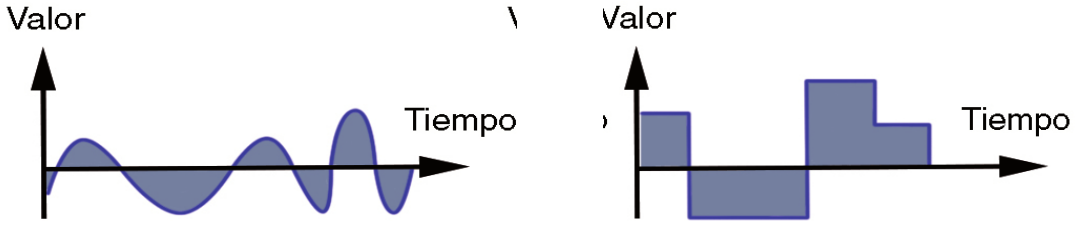


COMUNICACIONES

2.5. Tipos de Entradas y Salidas de Arduino

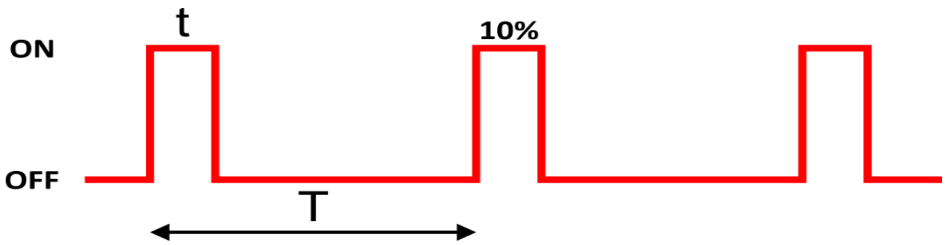
Mediante los conectores de Arduino podemos comunicar nuestros programas con el *exterior*, pudiendo leer valores de sensores o activando/desactivando diferentes tipos de actuadores (válvulas, luces, etc.).

Las entradas pueden ser de tipo **analógico** o **digital** (posteriormente se explicará con mayor detenimiento las diferencias entre estos dos tipos de señales). Las **digitales** se emplean para la lectura de dispositivos que solo puedan tener dos estados: activo o inactivo, como podrían ser los interruptores o pulsadores. Por el contrario, las entradas **analógicas** son empleadas para la lectura de la gran mayoría de sensores, como pueden ser sensores de temperatura, humedad, luminosidad, etc. Mediante diferentes circuitos acondicionadores es posible corresponder un valor concreto de tensión con una medida concreta de la magnitud física.



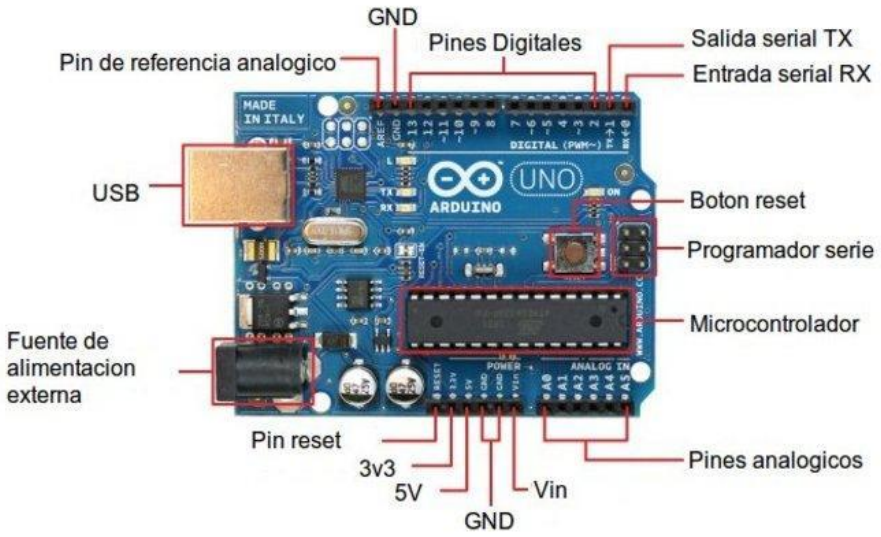
a. Señal analógica b. Señal digital

En cuanto a las **salidas**, únicamente son de tipo digital teniendo como estados '1' o '0' según la salida se quiera activar o desactivar. No obstante, es posible obtener una salida similar a la analógica, con la que controlar por ejemplo la velocidad de un motor o la intensidad luminosa de un LED, mediante una **salida PWM** (señalizadas en la placa de desarrollo con la etiqueta *PWM(~)*). Esta modulación por anchura de pulso proporciona una onda cuadrada con un cierto tiempo de la señal en alta y otro en baja.



2.6. Arduino UNO - Características

Arduino UNO es el modelo estándar de placas Arduino, normalmente la más utilizada cuando se comienza a trabajar en los primeros proyectos. En la siguiente imagen se muestra una placa Arduino UNO destacando sus principales componentes:

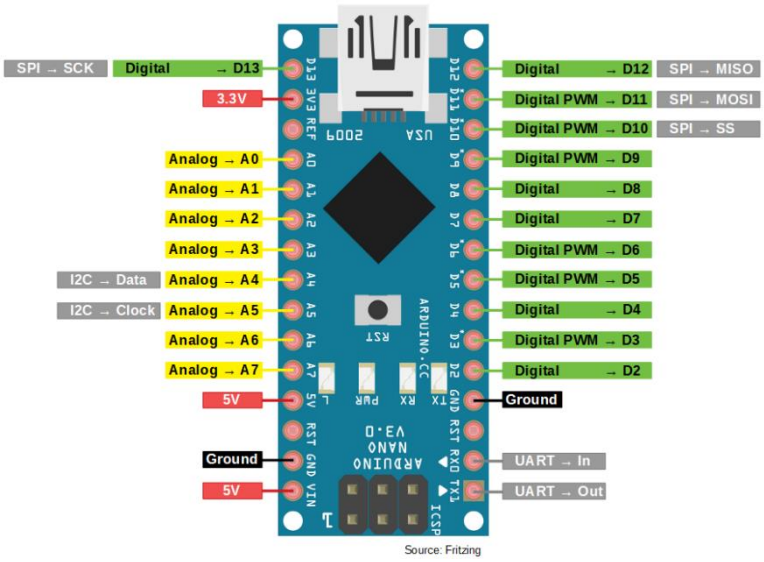


Las principales características del Arduino UNO respecto a otras versiones de Arduino son las siguientes:

- Microcontrolador ATmega328
- Voltaje de entrada 7-12 V
- Tensión de funcionamiento 5 V
- 14 pines digitales de I/O (6 salidas PWM)
- 6 entradas analógicas
- 32k de memoria Flash
- Reloj de 16MHz de velocidad
- Dimensiones 68,6 x 53,4 mm

2.7. Arduino NANO - Características

Se trata de otra versión de Arduino igual de sencillo de manejar que Arduino UNO y formada por el mismo microcontrolador (ATmega 328).



Las principales características del Arduino NANO respecto a otras versiones de Arduino son las siguientes:

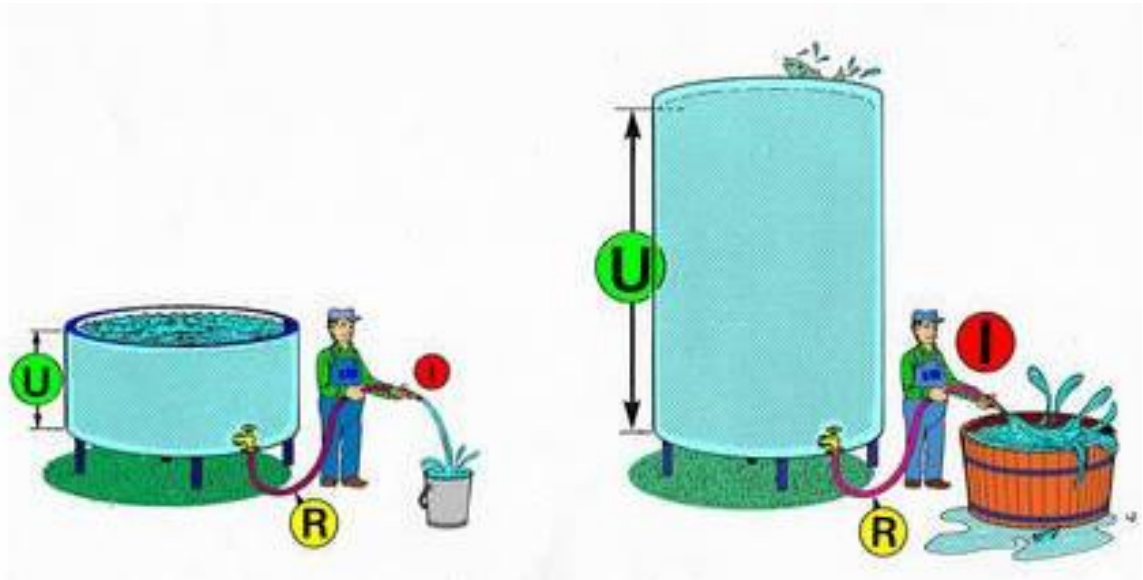
- Microcontrolador ATmega328
- Voltaje de entrada 7-12 V
- Tensión de funcionamiento 5 V
- 14 pines digitales de I/O (6 salidas PWM)
- 8 entradas analógicas
- 32kB de memoria Flash
- Reloj de 16MHz de velocidad
- Dimensiones 18,5 x 43,2 mm

3. Electrónica básica

3.1. La Ley de Ohm

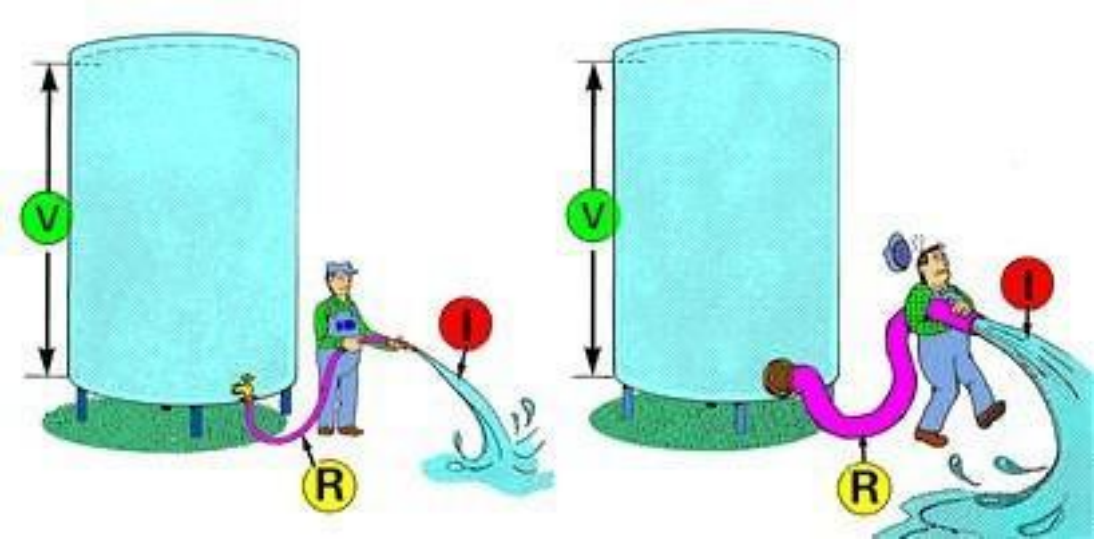
En electrónica y electricidad existe una ley fundamental, la **Ley de Ohm**, que relaciona las tres magnitudes básicas más importantes de un circuito: **Tensión** (**Voltaje**), **Corriente** (**Intensidad**) y **Resistencia**.

Tratar de comprender directamente el significado de estas magnitudes y su relación puede resultar un poco complicado, pero es bastante sencillo de entender si se hace una analogía entre un circuito eléctrico y un circuito de agua.

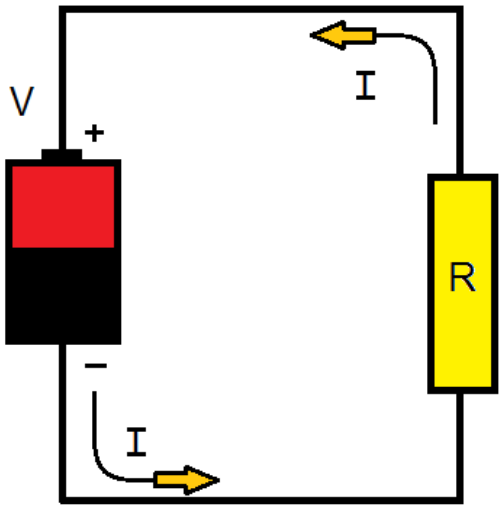


$$V = I \cdot R$$

- **Tensión** (**Voltaje**): La altura del tanque que aporta mayor o menor presión de agua, equivale a la tensión o voltaje que aporta una pila o fuente de alimentación en un circuito.
- **Corriente** (**Intensidad**): El flujo de agua a través de la manguera equivale a la corriente eléctrica que circula a través de los cables.
- **Resistencia**: La resistencia eléctrica que regula el paso de la corriente equivale al diámetro de la tubería que permite que pase mayor o menor caudal de agua.

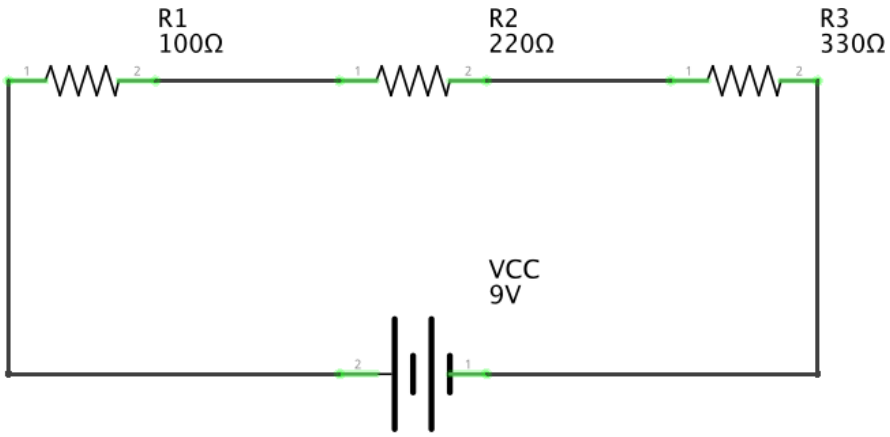


El circuito eléctrico más sencillo que se puede realizar es el que se muestra a continuación, y equivale directamente a la analogía con el agua explicada en los esquemas anteriores:



Mediante esta relación es posible determinar corrientes que pasan por diferentes resistencias o las tensiones existentes en estas. Para ello únicamente hay que tener en cuenta las dos formas de conexión de los componentes resistivos: serie y paralelo:

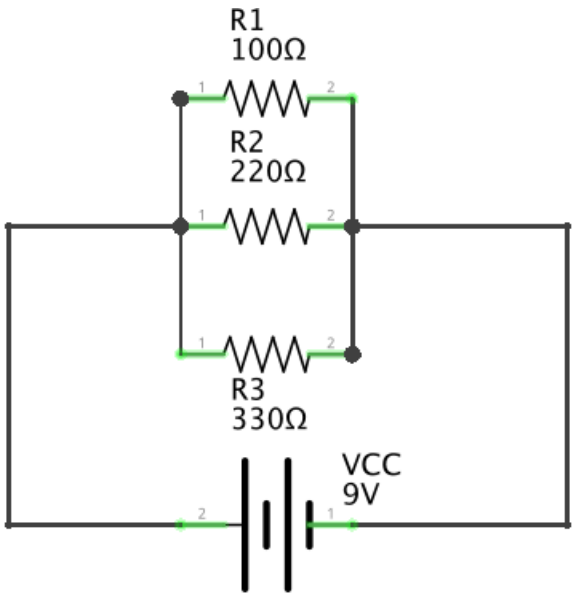
- Conexión serie: Las resistencias se conectan en cascada, es decir, una resistencia tras otra. De este modo, la corriente atraviesa todas las resistencias.



En esta configuración la resistencia total se obtiene como la suma aritmética de las resistencias, es decir:

$$R_{total} = R1 + R2 + R3$$

- Conexión en paralelo: En esta configuración todas las resistencias están a la misma tensión, de modo que por cada una pasa una parte de la corriente total, proporcional a la resistencia existente.

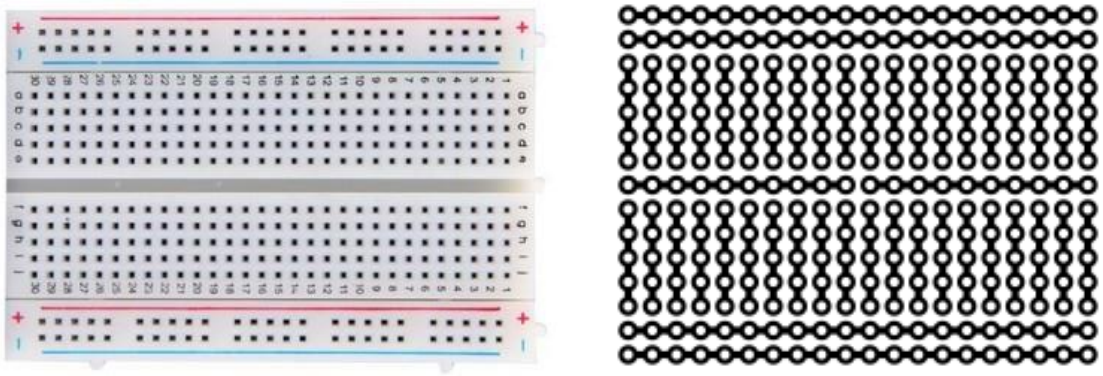


En este caso, la resistencia total se obtiene como la inversa de la suma fraccionaria de los valores, es decir:

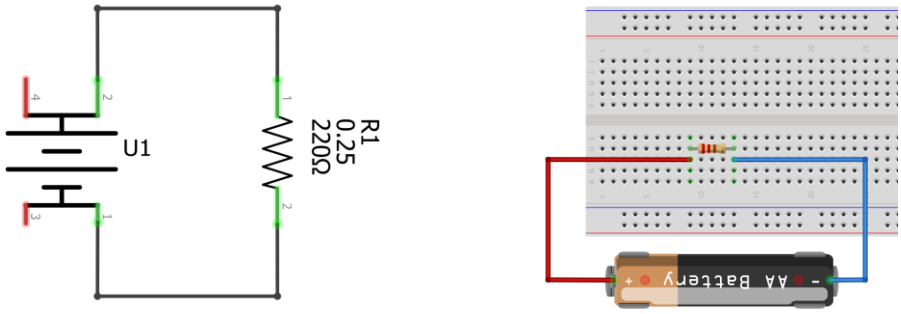
$$\frac{1}{R_{total}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} \quad \rightarrow \quad R_{total} = \frac{R_1 \cdot R_2 \cdot R_3}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3}$$

3.2. La Protoboard

Una placa de pruebas o protoboard es un tablero con múltiples agujeros conectados entre sí, preparado para insertar y conectar componentes electrónicos y cables sin necesidad de soldar, con el fin de construir y prototipar circuitos electrónicos de forma rápida y sencilla.



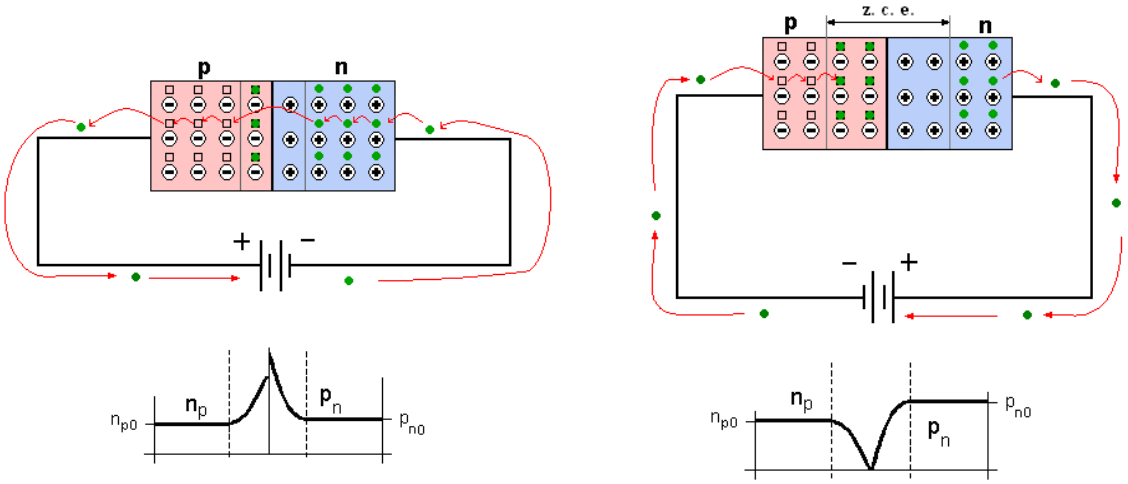
De las dos imágenes anteriores, la de la izquierda muestra el aspecto de una *protoboard*, mientras que la de la derecha muestra el patrón típico de disposición de las láminas de cobre interiores que conectan los agujeros de la *protoboard*. Se puede observar que las líneas horizontales de la imagen están unidas eléctricamente, de igual forma que sucede con cada línea vertical.



Esquema básico de conexión en Protoboard.

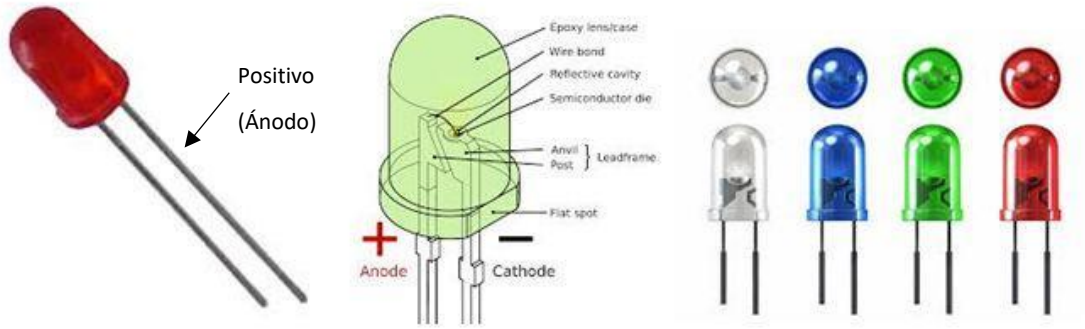
3.3. El LED

Un LED (Diodo Emisor de Luz) es un tipo de diodo con la particular característica de que es capaz de emitir luz. Un diodo es un componente electrónico que permite la circulación de corriente en un solo sentido. Está formado internamente por un material semiconductor con unión tipo p-n, por tanto, cuando se polariza en directa, es decir, con mayor tensión en el ánodo que en el cátodo, el diodo conduce electricidad y cuando se polariza en inversa bloquea el paso de corriente. Esta manera de funcionar del diodo hace que en cierta forma se parezca a un interruptor controlado por tensión.



En el caso particular del LED, se trata de un diodo que al conducir la electricidad su material semiconductor libera energía en forma de fotones, lo que hace que se ilumine.

Veamos ejemplos reales de diodos led:

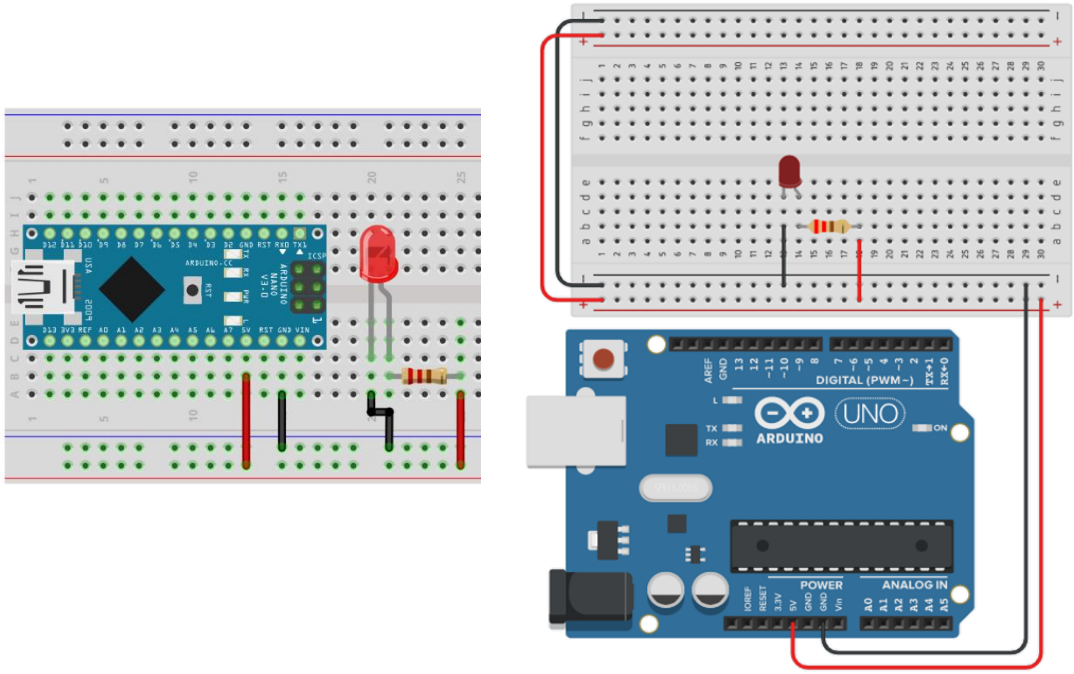


Veamos el símbolo de diodo led:

Nota importante: Cuando un LED conduce electricidad ofrece muy poca resistencia al paso de corriente, por lo que la fuente de alimentación va a suministrar toda la corriente posible de forma descontrolada, lo que equivale a un cortocircuito que como poco supondrá un alto riesgo de quemar el LED. Para evitar esto, siempre debemos conectar una resistencia limitadores de valor adecuado (220 ohm normalmente para tensiones de 5 voltios) en serie con el LED. Así controlamos el paso de corriente y evitamos que nada se rompa.

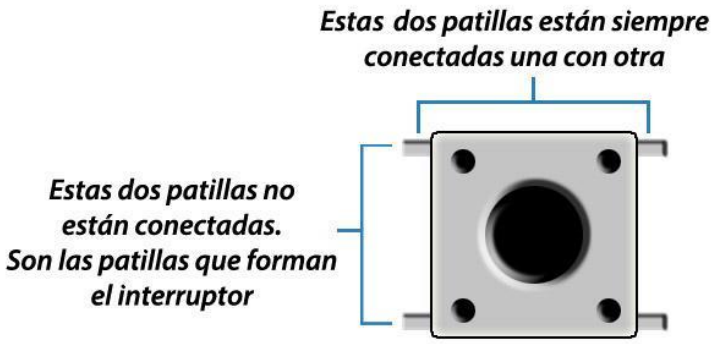
3.4. Actividad 0: Primer circuito con LED

Esta primera actividad consistirá en conectar un LED de forma directa a la salida de 5V del Arduino haciendo uso de la resistencia limitadora de 220 Ohmios.



3.5. Pulsadores

Un botón o pulsador en un componente básico en electrónica, que sirve para permitir o interrumpir de forma mecánica el paso de la corriente en un circuito cuando se acciona de forma manual, con un dedo por lo general.

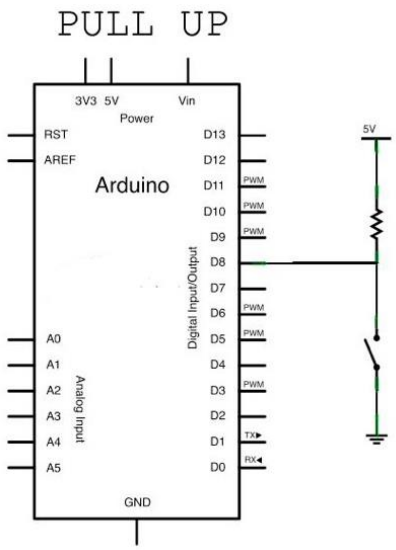


A diferencia del interruptor, el botón o pulsador cuenta con una única posición de reposo, lo que significa que cuando se deja de accionar el contacto vuelve por sí solo a su posición original.

En función de cuál sea su posición de reposo existen dos tipos de pulsadores, con contacto normalmente abierto (NA) o con contacto normalmente cerrado (NC).

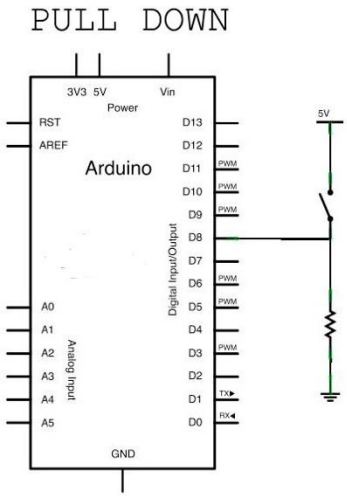
Para la lectura de los estados de los interruptores y pulsadores mediante un microcontrolador es posible emplear dos configuraciones: *Pull-up* o *Pull-down*, según la empleada se fijará como nivel lógico por defecto el nivel alto o bajo.

- **<u>Conexión mediante resistencia Pull-up</u>**: Esta configuración, con una resistencia conectada a la alimentación y la lectura de la entrada en la unión entre la resistencia y el pulsador, establece un nivel lógico de 1 en el estado de reposo del circuito (con el interruptor abierto) y un nivel de 0 cuando se pulsa.



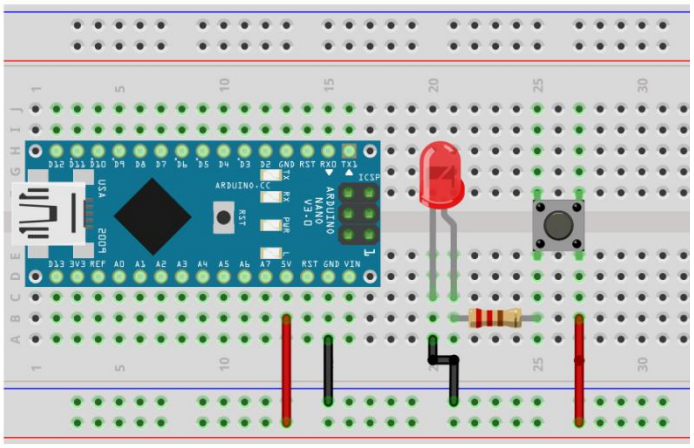
Esta **resistencia de *pull-up* puede omitirse** si se decide **emplear la que incorpora Arduino** en sus entradas digitales, para ello se debe indicar durante la programación que la entrada es *pull-up* y se conectará directamente el interruptor a la placa de desarrollo.

- <u>Conexión mediante resistencia Pull-down:</u> Con esta configuración se establece un nivel lógico de 0 en reposo y de 1 al pulsar el interruptor.



3.6. Actividad 1: LED con pulsador

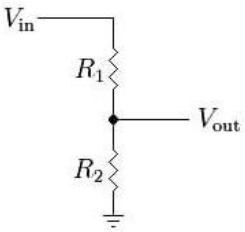
Esta segunda actividad consiste en emplear un pulsador para encender el LED. En este caso, el pulsador servirá de interruptor, de modo que cuando esté pulsado circulará corriente por el LED y este se encenderá. No es necesario emplear programación para realizar esta actividad.



3.7. Potenciómetro

Un potenciómetro es una resistencia variable mecánicamente. Dispone de tres terminales, de forma que manipulando el potenciómetro se obtiene entre la pata central (cursor o salida) y cualquiera de las patas de los extremos, una fracción de la diferencia de potencial total existente entre los extremos (entrada y VIN):

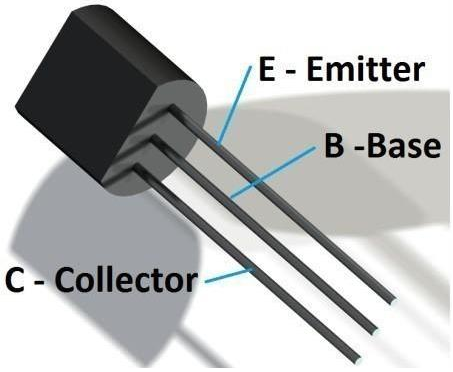
El circuito de un potenciómetro equivale al de un divisor de tensión. Entre los extremos del potenciómetro tenemos una diferencia de potencial total Vin y una resistencia total constante igual a R1+R2, no obstante los valores de R1 y R2 dependen de la posición del cursor y con ello el valor de Vout, que siempre tendrá un valor comprendido entre 0V y Vin.



$$Vout = \frac{R2}{R1 + R2} \cdot Vin$$

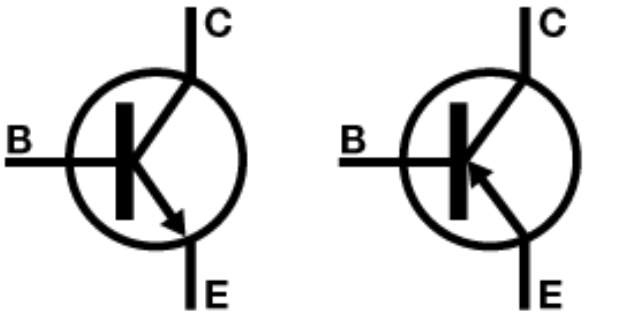
3.8. El transistor

Muchos de los circuitos que se controlarán desde el Arduino necesitan una corriente mayor a la que puede proporcionar esta placa. Para ello se recurre a un componente llamado transistor que permite amplificar la señal de salida o ser empleado como un interruptor controlador electrónicamente.



Está formado por materiales semiconductores P y N como los que forman el diodo, aunque en este caso consta de tres capas: Dos capas N y una P si es de tipo NPN, o dos P y una N si es un transistor PNP. Por tanto, consta de tres terminales:

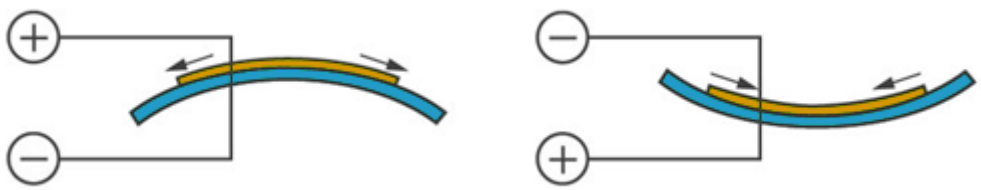
- **Colector:** Es el pin que generalmente se usa para entrada de corriente.
- **Base:** Es un pin de entrada de corriente al igual que el colector. Mediante éste se regula el funcionamiento del transistor.
- **Emisor:** Se trata de un pin de salida de corriente.



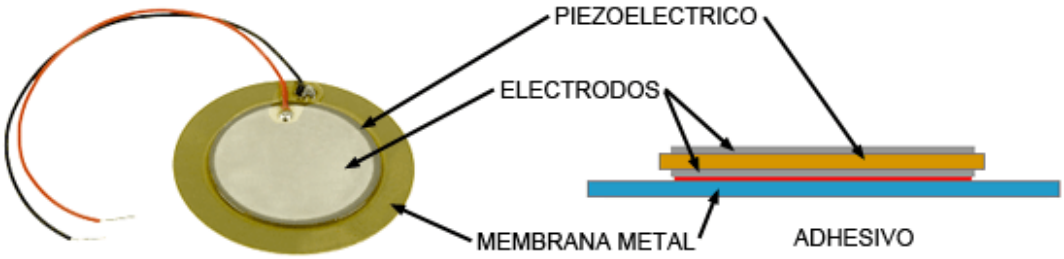
Transistor NPN Transistor PNP

3.9. El Zumbador

El zumbador es un componente sonoro que basa su principio de funcionamiento en la piezoelectricidad, un fenómeno que sucede con determinados cristales como el cuarzo, que al ser sometidos a una tensión mecánica generan una tensión eléctrica, y también lo contrario, cuando son sometidos a una tensión eléctrica se deforman variando su volumen, volviendo a su posición de reposo cuando se les deja de aplicar tensión.



Si sometemos un material piezoeléctrico a una tensión eléctrica variable se produce una vibración de igual frecuencia que la señal eléctrica. El zumbador aprovecha este fenómeno para hacer vibrar una membrana de metal a la que va adherido el cristal. Si la frecuencia de vibración está dentro de la frecuencia audible (20 Hz a 20 KHz aproximadamente) se produce una onda sonora a la misma frecuencia.



Los zumbadores tienen la ventaja de ser componentes muy baratos y fáciles de manejar, por lo que son ideales para proyectos que requieran de sonido o incluso melodías mediante tonos simples, como por ejemplo alarmas o tonos para pulsaciones. Sin embargo, los zumbadores no son válidos para reproducir pistas musicales, no porque no sean capaces de hacerlo sino porque la calidad del sonido es muy mala, por lo que para este tipo de aplicaciones lo más adecuado es emplear altavoces con un pequeño amplificador de audio.

Podemos distinguir principalmente entre dos tipos de zumbadores:

- **Zumbador pasivo**: Sin oscilador interno, lo que implica que debe alimentarse con corriente alterna para poder funcionar. Tienen la desventaja de ser un poco más difíciles de utilizar al tener que ser alimentados directamente a través de una señal eléctrica de frecuencia concreta para producir el tono deseado. Sin embargo, existen librerías en Arduino que nos facilitan mucho el control de zumbadores pasivos. Tienen la gran ventaja de que podemos generar melodías variando la frecuencia de la señal. Se suele distinguir fácilmente por no tener generalmente pegatina en la parte superior, y por tener terminales de igual longitud.

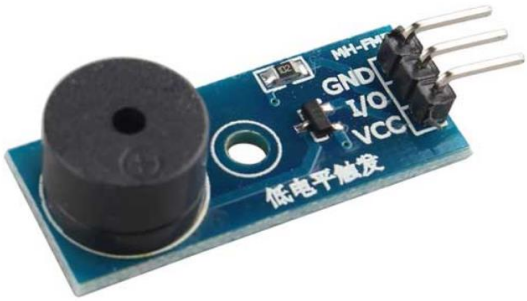
- **Zumbador activo**: Dispone de un oscilador interno, lo que implica que funciona con corriente continua, por lo que resulta más sencillo de utilizar que el zumbador pasivo, ya que simplemente se controla con una señal digital en alta o baja para producir o no sonido. La desventaja es que no puede generar melodías, ya que no se puede variar la frecuencia de la señal para producir diferentes tonos. Se distingue fácilmente porque suele tener pegatina en la parte superior, además de tener el terminal positivo más largo que el negativo.

Una forma útil de distinguir un zumbador activo de un zumbador pasivo, es alimentándolo a con corriente continua (generalmente 5V). Si el zumbador suena es activo, de lo contrario es pasivo.

Zumbador pasivo

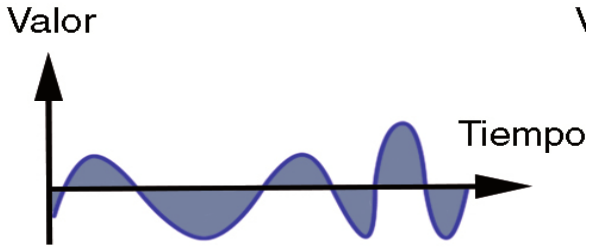
Zumbador activo

También están los zumbadores en formato módulo, aún más sencillos de utilizar, ya que incorporan los componentes electrónicos adicionales necesarios, generalmente una resistencia y un transistor, por lo que ya vienen preparados para poder ser conectados a un microcontrolador y funcionar. La necesidad del transistor es debido a que el zumbador generalmente consume más corriente de la que pueden suministrar los pines digitales de los microcontroladores. En formato módulo, también podemos encontrar zumbadores tanto activos como pasivos.



3.10. Señales analógicas y digitales

- **Señal analógica:** Es aquella que presenta una variación continua en el tiempo por lo que puede tomar infinitos valores en un intervalo de tiempo dado. Este tipo de señales son las que podemos encontrar de forma natural en el medio físico, como por ejemplo el sonido, variaciones en la temperatura, etc.



a. Señal analógica

- **Señal digital:** Es aquella que presenta una variación discontinua en el tiempo y que solo puede tomar valores discretos, 0-1 en el caso de señales digitales de un bit, 0-1-2-3 en el caso de señales de dos bits, etc. Estas señales no se encuentran en el mundo físico de forma natural, sino en electrónica y sistemas de comunicaciones digitales creados por el hombre.

b. Señal digital

3.11. Señales PWM

Antes de nada recordar que existen dos tipos de señales, las señales digitales y las señales analógicas.

Las señales digitales varían de forma discontinua y pueden tomar valores discretos, comúnmente dos valores (0 y 1), utilizados muchas veces para activar o desactivar algo, como por ejemplo encender y apagar un LED.

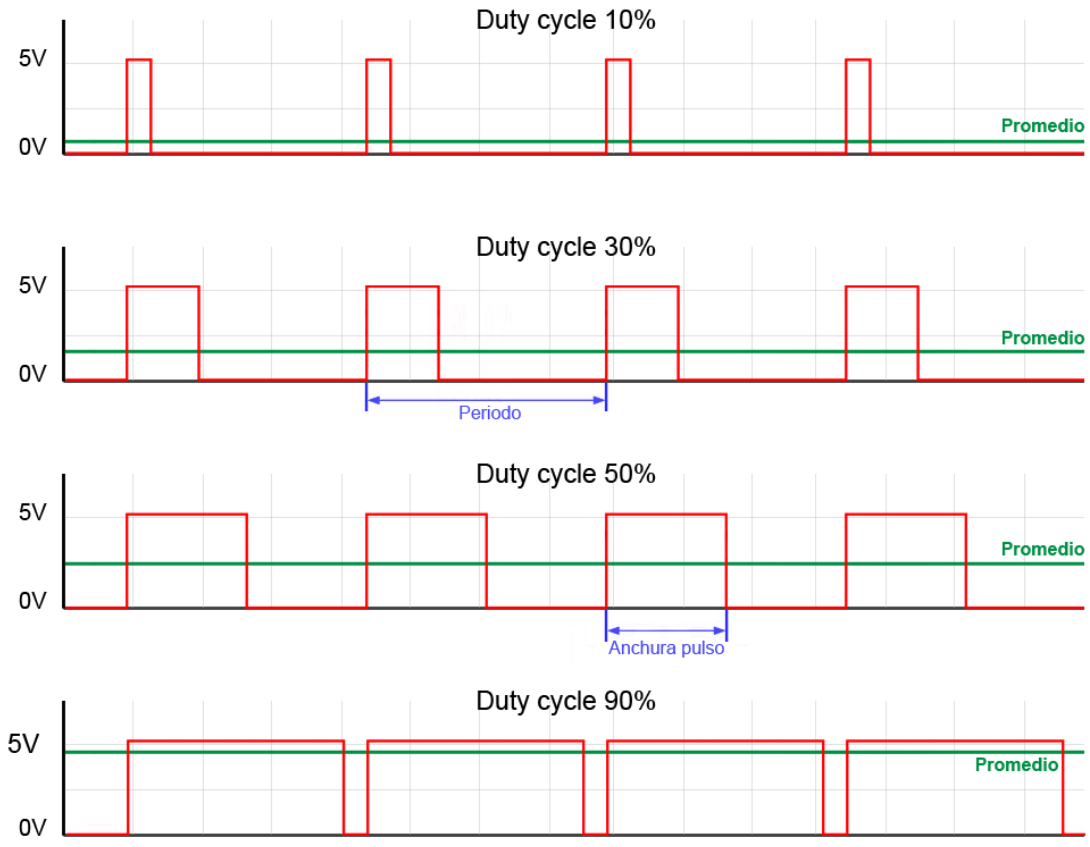
Las señales analógicas varían de forma continua y pueden tomar infinitos valores en un rango determinado. Como salida, se utilizan normalmente para regular en intensidad alguna magnitud, como por ejemplo la intensidad de luz de una bombilla o la velocidad de un motor, a diferencia del “todo o nada” de las señales digitales.

Sin embargo, al igual que la mayoría de las placas de control programables, Arduino dispone de entradas analógicas pero no de salidas analógicas, y en estos casos recurrimos a las señales PWM, un “truquito” que nos permite emular señales analógicas a partir de señales digitales.

Una señal PWM (Pulse-Width Modulation o Modulación por Ancho de Pulso), es un tipo de señal digital que toma valores entre un mínimo y un máximo (por ejemplo 0 y 5 V) y

que, manteniendo constante su frecuencia, mantiene activa la señal digital durante un tiempo y la desactiva durante el tiempo restante del periodo.

El promedio de la señal será un valor analógico que podemos controlar modificando a conveniencia el ancho del pulso de la señal PWM.



La proporción de tiempo del ancho del pulso de la señal PWM respecto del periodo se denomina "**Duty Cycle**" y se expresa en tanto por ciento.

Esto no significa que realmente exista una señal promedio. El dispositivo que vayamos a controlar se conecta directamente a la señal PWM. Lo que ocurre, siempre que la frecuencia de la señal sea lo suficientemente alta, es que el dispositivo conectado, por ejemplo un LED, no es capaz de seguir la alta frecuencia de conmutación de la señal PWM, por lo que al ojo humano muestra un nivel de intensidad de luz estable, proporcional al promedio de la señal.

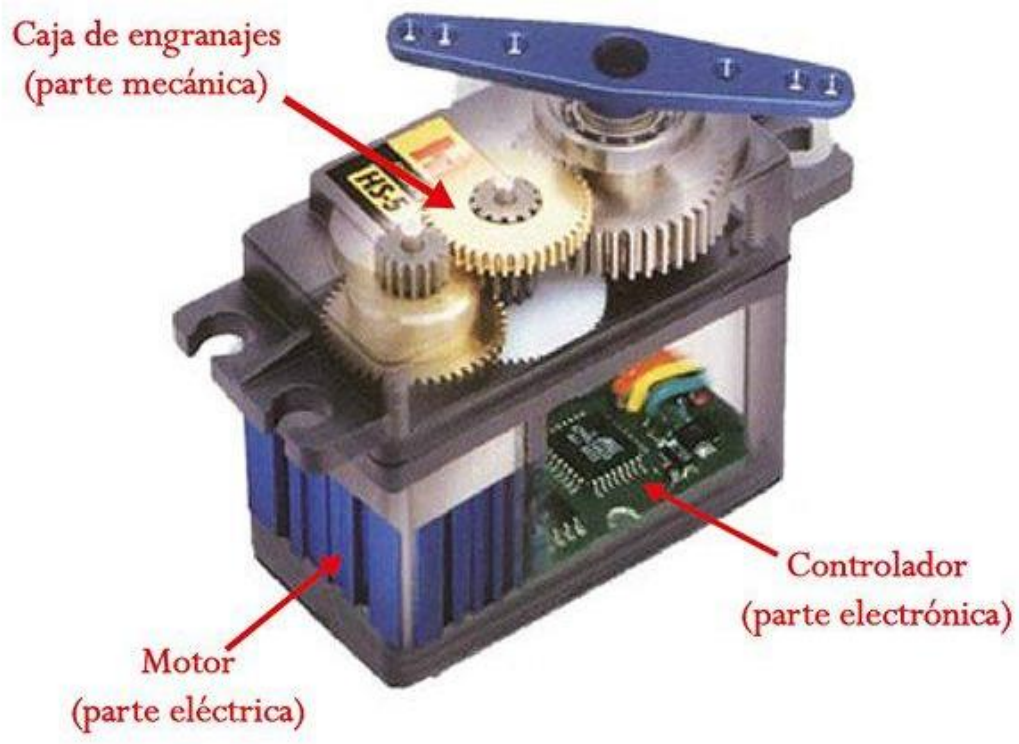
Como ejemplo, comentar que los servomotores llevan tres cables, VCC y GND para alimentación, y un tercer cable para control que emplea precisamente una señal de tipo PWM.

Los pines digitales del Arduino que a su vez están preparados para funcionar como salidas PWM aparecen identificados en la placa con el símbolo "~" junto al número del pin. En el caso del Arduino UNO, Nano y Mini, tenemos 6 salidas PWM disponibles con resolución de 8 bits, en los pines 3, 5, 6, 9, 10 y 11.

Una vez llegados a este punto, con ayuda de las explicaciones del *Apartado 7 "Fundamentos de Arduino"*, deberías ser capaz de hacer las Actividades 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 y 13 del Apartado 8 (Actividades).

4. Servomotor

Un servomotor es un tipo de motor de corriente continua, que en lugar de estar diseñado para girar de forma continua, se construye para ser capaz de girar un ángulo determinado a una velocidad determinada, en función de una señal de control, y mantenerse estable en dicha posición. Los servomotores por tanto, son motores con los que podemos controlar fácilmente la posición y la velocidad angular.



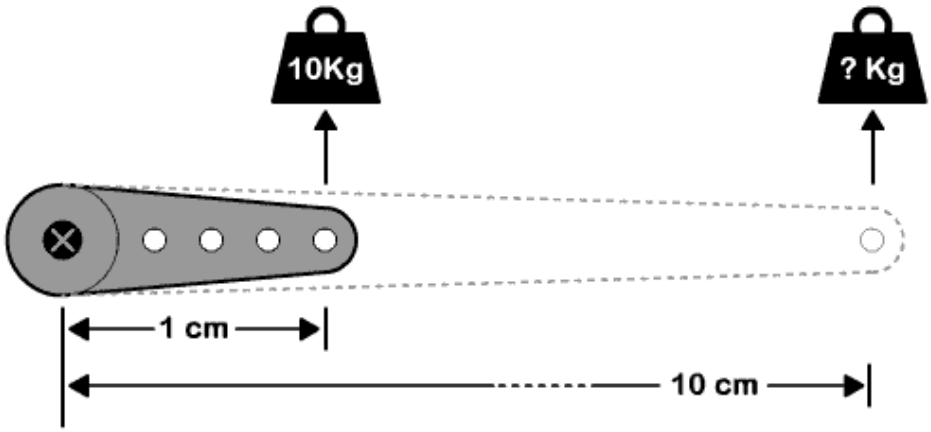
La mayoría de los servomotores tienen un determinado ángulo de operación, es decir que son capaces de girar entre 0º y un ángulo dado, que por lo general es 180º, aunque también hay servomotores capaces de operar entre 0º y 360º.

Los servomotores constan de tres cables, dos para alimentación (VCC y GND) y un tercer cable de control, a través del cual se le indica mediante una señal la posición angular.

4.1 Características de los servomotores

Las características más importantes en las que nos debemos fijar a la hora de elegir un servomotor son las siguientes:

- **Tensión de alimentación**: Como todo, los servomotores se alimentan a un determinado nivel de tensión o rango de tensión que hay que conocer y respetar.
- **Fuerza**: Se mide en Kg/cm y se refiere al peso que es capaz de levantar el servo a una distancia de 1 cm respecto del eje.



- **Velocidad**: Se expresa en seg/60º.
- **Rango de operación**: Ángulo de rotación permitido por el servomotor.
- **Material de los engranajes**: plástico o metal.
- **Peso**: Una característica especialmente importante en determinadas aplicaciones.
- **Dimensiones**: En un proyecto se debe tener en cuenta las dimensiones de todo.

4.2 El servomotor SG-90

Para nuestras actividades vamos a utilizar servomotores SG-90, cuyas características son las siguientes:

- **Tensión de alimentación**: 4,8 V – 7,2 V
- **Fuerza**: 1,2 Kg/cm (a 4,8 V)
- **Velocidad**: 0,12 seg/60º (a 4,8 V)
- **Rango de operación**: 0º a 180º
- **Material de los engranajes**: Plástico
- **Peso**: 9 gr
- **Dimensiones**: 22.0 x 11.5 x 27 mm (L x W x H)

Arduino UNO y Arduino Nano disponen de 13 pines digitales, de los cuales 6 están implementados para funcionar como salidas PWM de 8 bits (pines 3, 5, 6, 9, 10 y 11), que podemos reconocer fácilmente por el símbolo “ ~ “.

La frecuencia de funcionamiento de las señales PWM que aportan estas salidas es de aproximadamente 500 Hz, lo que equivale a un periodo de 20 ms.

Controlar la posición de un servomotor a partir de una señal PWM desde cero, implicaría desarrollar un programa que relacione el ancho del pulso de la señal con el ángulo de giro deseado.

Afortunadamente y como también ocurre con otros miles de componentes, existen librerías ya desarrolladas por la comunidad Arduino que simplifican enormemente este trabajo, y que nos permiten hacer uso de los motores de una forma muy sencilla.

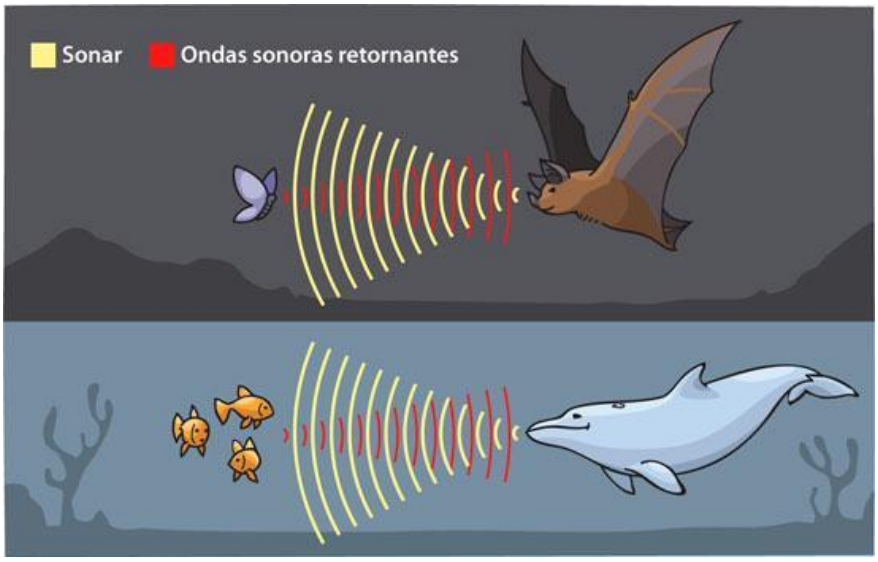
En el caso de IDE, tenemos la librería “Servo.h”, que además ya nos viene instalada por defecto. Las sentencias básicas que podemos usar con esta librería son las siguientes:

Sentencia	Descripción
#include <Servo.h>	Para usar la librería “Servo.h” agregamos esta línea de código al inicio del programa.
Servo servo1	Para utilizar la librería “Servo.h” debemos declarar un objeto tipo Servo al que le podemos dar cualquier nombre. En este caso le hemos llamado “servo1”
servo1.attach(9)	En el “Setup” hay que incluir esta sentencia que determina el pin digital PWM al que irá conectado el objeto creado, “servo1”. Como ejemplo se ha puesto el pin 9.
servo1.write(angulo)	Hecho lo anterior ya podemos controlar el servo dentro del “Loop”. Esta sentencia determina la posición angular del servo, indicado el “angulo” con números enteros.

5. Sensor de proximidad por ultrasonidos

5.1 La ecolocalización

La ecolocalización es una capacidad que poseen algunos animales, como el murciélago y el delfín, de orientarse y reconocer su entorno mediante la emisión de sonidos, analizando el eco recibido cuando la onda sonora choca y rebota contra un obstáculo. Estos animales pueden conocer la distancia a la que se encuentran los objetos midiendo la diferencia de tiempo entre la señal emitida y la recibida, incluso pueden ser capaces de captar información más específica, como la posición espacial del objeto, su tamaño y otras características.

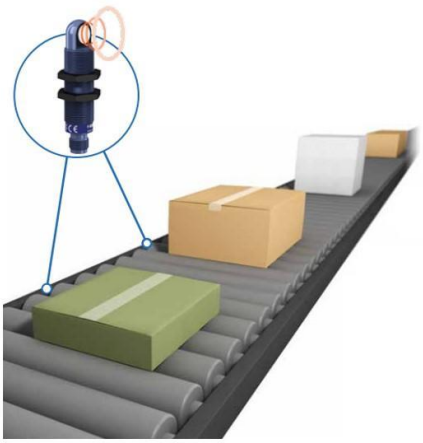


5.2 Ultrasonidos

Los ultrasonidos son ondas sonoras de frecuencia superior a la perceptible por el oído humano, es decir, superiores a los 20.000 Hz. Recordemos que el oído humano solo es capaz de percibir el sonido de frecuencias comprendidas entre los 20 Hz y los 20.000 Hz.

En la práctica los ultrasonidos son muy útiles en diversos campos como por ejemplo la medicina. Por lo general a todos nos suena el término ecografía. Se trata de imágenes creadas a partir del "eco". Las máquinas utilizadas para generar ecografías, permiten crear imágenes de órganos internos gracias a los ultrasonidos, ondas sonoras que tras rebotar contra las estructuras corporales retornan al dispositivo, el cual es capaz de crear las imágenes a partir del análisis de la intensidad y tiempo de retorno de las ondas ultrasónicas.

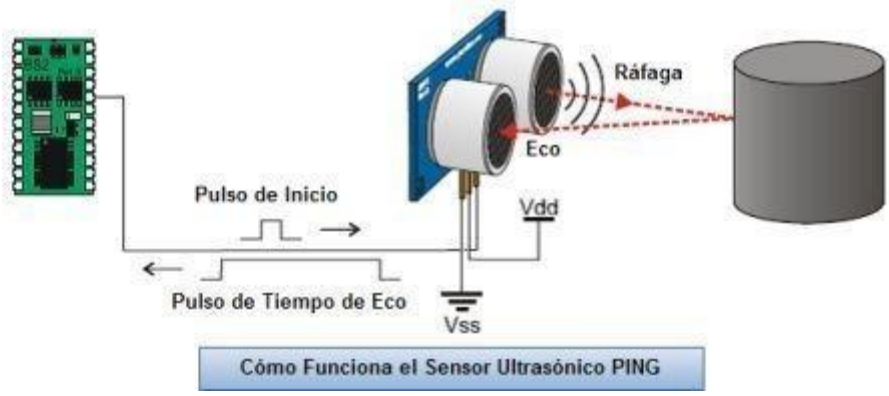
En el sector industrial, los ultrasonidos se emplean comúnmente como sensores para medir distancias o detectar obstáculos, así como para limpieza de determinadas piezas y componentes electrónicos por medio de bañeras de ultrasonidos.



5.3 Sensor de proximidad HC-SR04

El HC-SR04 es un sensor de proximidad por ultrasonidos que sirve para medir distancias. Su modo de operar se basa en todo lo que ya hemos descrito anteriormente acerca de las ondas ultrasónicas.

El dispositivo cuenta con un elemento transmisor y un elemento receptor. El transmisor emite pulsos ultrasónicos que rebotan contra los objetos cercanos y son recibidos por el receptor. Conociendo la velocidad del sonido y midiendo el tiempo de retorno, somos capaces de calcular la distancia a la que se encuentra el objeto que reflejó la onda sonora.



Las principales características del sensor son las siguientes:

- Alimentación a 5 V
- Rango de medición: de 2 cm a 400 cm (en la práctica se queda en torno a 200 cm)
- Frecuencia del pulso: 40 KHz
- Apertura del pulso: 15 grados
- Señal de disparo: 10 us
- Dimensiones del módulo: 45x20x15 mm

En cuanto a las funciones de sus entradas y salidas tenemos lo siguiente:

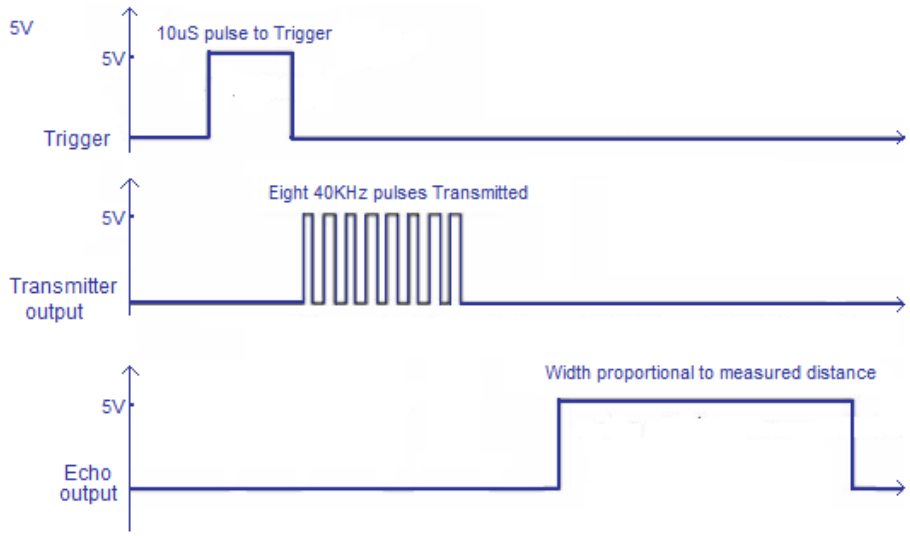
Nombre	E/S	Descripción
VCC	IN	Entrada de alimentación 5 V
Trig	IN	Señal de Trigger. Cuando recibe un pulso de 10 us el transmisor emite un sonido de 8 pulsos a 40 KHz
Echo	OUT	Señal de Eco. Emite una pulso digital de duración igual al tiempo que la onda emitida tardó en llegar al receptor
GND	IN	Entrada de tierra

5.4 Algoritmo y funcionamiento del sensor

Para medir una distancia con el sensor de ultrasonidos HC-SR04 el funcionamiento sería el siguiente:

- Desde cualquier pin digital de Arduino se envía un único pulso de 5V y 10us de duración a la entrada “Trig” del sensor.
- Tras recibir dicha señal, el transmisor del sensor indicado mediante la letra “T” emitirá una onda sonora ultrasónica compuesta de 8 pulsos y frecuencia de 40 KHz.
- La onda sonora rebotará contra un objeto del medio físico y regresará al sensor a través del elemento receptor indicado con la letra “R”.

- Después que el receptor ha recibido los 8 pulsos de la onda sonora, el sensor mide el tiempo que la onda ha tardado en regresar, y comunica dicho dato generando una señal de un único pulso de duración igual al tiempo calculado y la envía a través de la salida "Echo".



- De vuelta al Arduino, para conocer la duración del pulso emitido por la señal "Echo", recurrimos a la función "pulseIn()" que explicaremos más adelante en el apartado de programación.
- Ahora que ya tenemos el tiempo de propagación de la onda, podemos calcular fácilmente la distancia recorrida por la misma y por tanto la distancia entre el sensor y el objeto. Sabiendo que la velocidad del sonido es de 343 m/s, el cálculo sería el siguiente:

  Primero aplicamos la siguiente conversión de unidades:

$$343\ \frac{m}{s} = 34300\,\frac{cm}{s} = 0{,}0343\,\frac{cm}{\mu s} = \frac{1\ cm}{29{,}2\ \mu s}$$

  Lo que significa que la velocidad del sonido recorre 1 cm en 29,2 μs. El espacio recorrido por la onda sonora es el doble de la distancia entre el sensor y el objeto puesto que la onda ha de ir y volver, por tanto la fórmula para calcular dicha distancia será la siguiente:

$$2\ x\ Distancia = Tiempo \cdot Velocidad$$

$$Distancia = \frac{Tiempo \cdot Velocidad}{2}$$

$$Distancia = \frac{Tiempo}{2\ x\ 29{,}2}$$

6. Alimentación de circuitos y dispositivos

6.1 Tensión de alimentación

Lo primero a tener en cuenta antes de alimentar un circuito, un módulo electrónico y en general cualquier dispositivo o aparato, es su tensión de funcionamiento, así como el tipo de corriente utilizada (alterna o continua), si bien de esto último no nos preocuparemos de momento puesto que la mayoría de aplicaciones con Arduino funcionan con corriente continua.

Todos los componentes están diseñados para funcionar a un determinado nivel de voltaje o bien dentro de un rango de tensiones posibles. Superar el nivel de tensión máximo permitido por el componente puede suponer la rotura del mismo. Por otro lado, un nivel de tensión inferior al mínimo indicado impedirá al componente funcionar de forma correcta.

Veamos algunos ejemplos conocidos:

Bombilla Incandescente 220 VAC

Ordenador Portátil 19 VDC

6.2 Consumo del circuito o aparato receptor

El segundo aspecto importante a tener en cuenta a la hora de escoger una fuente de alimentación adecuada para nuestro proyecto, es el consumo de todos y cada uno de los componentes que lo conforman, lo cual dependerá del nivel de corriente que demanden tales componentes, medido en amperios.

Para calcular el consumo total debemos primeramente medir o averiguar el consumo aproximado de cada componente, y sumarlos todos. Una fuente adecuada será aquella con capacidad de suministrar **como mínimo** el nivel de amperios demandado por el proyecto.

Para conocer el consumo de cada módulo o componente electrónico, debemos consultar la ficha técnica del fabricante para cada componente, y a menudo también

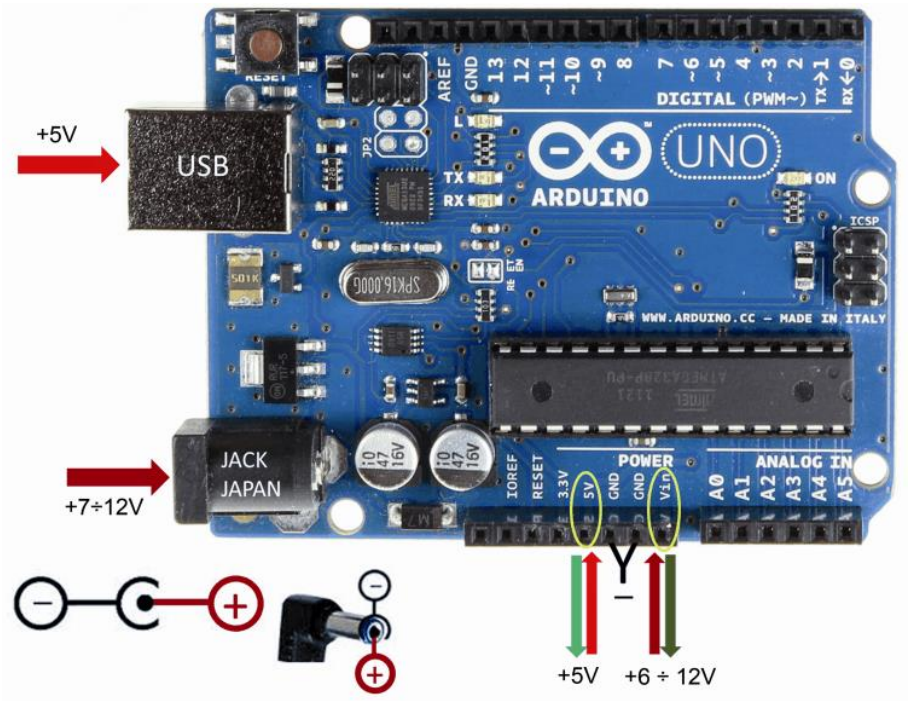
podremos consultar esta información buscando por internet a través de diversos artículos que traten sobre el tema. A la hora de realizar la estimación, también debemos tener en cuenta que los componentes pueden no tener un consumo fijo, sino variable en función del nivel de tensión aplicado y del modo de uso. A continuación se muestran las tensiones de alimentación y los consumos típicos de algunos componentes a modo de ejemplo:

Componente	Tensión	Consumo
Sensor ultrasonido HC-SR04	5 V	15 mA
Servomotor SG90	5 - 6 V	220 - 450 mA
Diodo LED 5mm	2 - 5 V	20 - 30 mA

6.3 Maneras de alimentar Arduino

Existen diversas formas de alimentar nuestra placa Arduino, las cuales pueden variar un poco de un tipo de placa a otra, aunque en este curso nos centraremos en las placas Arduino Uno y Arduino Nano.

Decantarnos por una u otra forma de alimentar la placa, dependerá principalmente de los requisitos del proyecto.



- A través del puerto USB:

La forma más sencilla de alimentar Arduino es a través del puerto USB. Esta entrada admite 5V y se emplea normalmente para alimentar el Arduino conectado al ordenador, aunque también se pueden emplear otros dispositivos como un powerbank o cualquier otro que disponga de puerto USB, siempre que la tensión sea de 5V estables.

Este tipo de alimentación suministra corriente directamente a la placa Arduino sin pasar por el regulador de voltaje.

- A través del conector Jack:

Esta opción está disponible en el Arduino Uno pero no en el Arduino Nano. El Arduino Uno dispone de un conector Jack que podemos utilizar para alimentar la placa conectando una fuente de alimentación que emplee un conector macho del mismo tipo. Esta entrada admite un rango de tensiones de 6-20 V, aunque lo más recomendable es utilizar un voltaje de entre 7-12 V.

Esta forma de alimentación emplea el regulador de voltaje del Arduino. Voltajes inferiores a 7 V pueden causar que el regulador no funcione correctamente y voltajes superiores a 12 V que el regulador se sobrecaliente en exceso.

Resulta muy adecuado en este caso utilizar fuentes de alimentación de 9V 1A, o baterías de Li-ion 7,4 V.

- A través del pin VIN:

El pin VIN funciona igual que el conector Jack, solo cambia el tipo de conector. Si lo utilizamos como entrada para alimentar la placa Arduino debemos aplicar un nivel de tensión recomendado de entre 6-12 V, dado que el pin VIN también emplea el regulador de voltaje.

A diferencia de la entrada Jack, con VIN podemos utilizar perfectamente un voltaje mínimo de 6V, ya que no pasa por el diodo de protección contra polaridad inversa conectado a la entrada Jack, que supone una caída de tensión de casi 1V. Esto hace que la pin VIN sea especialmente recomendable cuando necesitamos emplear como fuente de alimentación cuatro pilas o baterías AA conectadas en serie de 1,5V cada una.

En el caso de Arduino Uno, si alimentamos la placa a través del conector Jack podemos utilizar el pin VIN como salida en lugar de como entrada, para sacar el mismo valor de voltaje que estemos aplicando a través del conector Jack (descontando el voltio que se pierde por el diodo de protección contra polaridad inversa), y utilizar este voltaje para alimentar otras partes del circuito.

- A través del pin 5V:

Esta forma es equivalente al del puerto USB, pero a través del pin 5V. Tampoco emplea el regulador de voltaje, de ahí que la tensión de entrada deba ser exactamente 5V estables, lo que implica un mayor riesgo si no disponemos de una fuente de

alimentación adecuada para usar esta entrada, lo que hace de este método el menos recomendable a menos que el proyecto lo exija.

Además, por lo general suele ser mucho más interesante utilizar el pin 5V como salida y no como entrada, para alimentar otros componentes del proyecto a través de este pin.

- Resumen:

FORMAS DE ALIMENTAR ARDUINO (UNO Y NANO)			
Tipo de entrada	**Tensión**	**Regulador de tensión**	**Fuente preferida**
Puerto USB	5 V	No	Ordenador Powerbank
Conector Jack	7-12 V	Si	Cargador 9V Baterías Li-Ion 7,4 V
Pin VIN	6-12 V	Si	4 Baterías AA 1,5 V
Pin 5V	5 V	No	Fuente regulada 5 V

6.4 Alimentar componentes desde Arduino

En el apartado anterior vimos las distintas formas que existen de alimentar Arduino, así como los tipos de fuentes de alimentación más recomendables para cada caso.

Ahora veremos las distintas formas de utilizar la placa Arduino Uno y Nano para alimentar los circuitos y componentes conectados a la placa, cuyas características y limitaciones dependerán en gran medida del tipo de alimentación escogida para la placa y que ya consideramos en el apartado anterior.

En general vamos a tener dos situaciones diferentes:

- Arduino alimentado desde puerto USB:

En este caso tendremos disponible el pin 5V para alimentar componentes, con una limitación teórica de 500 mA en total. La razón de esta limitación es el polifusible de protección que tiene Arduino para proteger de sobrecargas y cortocircuitos al ordenador o fuente conectada al puerto USB, que actúa en caso de sobrepasarse el límite indicado.

Esta limitación es teórica ya que no tiene en cuenta el consumo del propio Arduino de aproximadamente 50 mA, lo que significa que nos quedan alrededor de 450 mA reales para alimentar componentes conectados al pin 5V.

También tenemos los pines digitales, los cuales tienen una capacidad máxima de 40 mA cada uno (aunque se recomienda no sobrepasar los 20 mA) y de 200 mA para todos los pines conectados simultáneamente, por lo que son utilizados principalmente para enviar y recibir señales de control, así como para alimentar pequeñas cargas como diodos LEDs.

Considerando las dos restricciones anteriores, tenemos que si alimentamos la placa Arduino por USB, el límite total será de 450 mA a repartir entre componentes conectados al pin de 5V y los conectados a los pines digitales.

CUANDO ARDUINO ES ALIMENTADO POR USB				
Tipo de salida	**I max (teórica)**	**I max (real o recomendada)**	**I max (todos los pines)**	**I max (total placa)**
Pin 5V	500 mA	450 mA	450 mA	450 mA
Pines digitales	40 mA	20 mA	200 mA	

- Arduino alimentado desde Jack o Vin:

En este caso tenemos igualmente disponibles el pin de 5V y los pines digitales para alimentar componentes conectados a la placa Arduino.

La limitación de los pines digitales es exactamente la misma que la descrita anteriormente, pero sin embargo cambia la situación del pin 5V. Alimentando el Arduino a través del Jack o el pin Vin, la corriente máxima que el Arduino puede suministrar a través del pin 5V ya no se encuentra limitada por el polifusible que afecta únicamente al puerto USB, sino por la corriente máxima que puede entregar el regulador de voltaje.

La corriente máxima que puede soportar el regulador de voltaje es de 1A teóricamente, aunque hay que aclarar que dicho valor depende del nivel de tensión que estemos usando para alimentar el Arduino. Aplicando menor voltaje (respetando siempre el mínimo de 7V) permitimos un mayor margen para subir la corriente sin que el regulador se queme, pudiendo llegar al máximo teórico de 1A. Sin embargo, si aumentamos la tensión de alimentación, este límite teórico se verá bastante reducido.

CUANDO ARDUINO ES ALIMENTADO POR JACK O VIN				
Tipo de salida	**I max (teórica)**	**I max (real o recomendada)**	**I max (todos los pines)**	**I max (total placa)**
Pin 5V	1000 mA (si Vin=7-9V)	900 mA	900 mA	1000 mA
Pines digitales	40 mA	20 mA	200 mA	

7 Fundamentos de programación Arduino

Como ya sabemos, Arduino es una placa de desarrollo que dispone de un microcontrolador en cuya memoria podemos cargar un programa definido por el usuario, que permitirá al Arduino realizar una tarea o función específica, comunicándose con otros dispositivos conectados a sus entradas y salidas.

Para programar una placa Arduino podemos utilizar diferentes lenguajes y entornos de programación que podemos clasificar en dos categorías:

1) Lenguajes de programación visual o por bloques:

- Scratch 4 Arduino
- Ardublock
- Visualino
- mBlock
- Tinkercad

2) Lenguaje de programación textual:

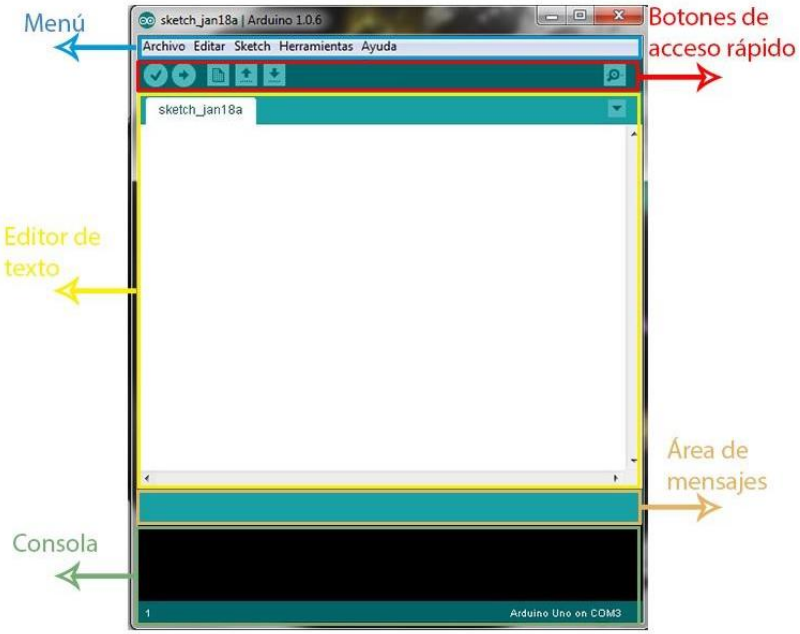
- C++ (Arduino IDE)

7.1 Descripción de Arduino IDE

El software Arduino IDE es un entorno de desarrollo integrado, que como ya hemos adelantado nos permite programar placas Arduino empleando un tipo de lenguaje de programación textual. Las funciones que dispone este software son las siguientes:

- **Editor de texto**: Entorno en el que se desarrolla el código.
- **Compilador**: Traduce el código desarrollado con lenguaje de alto nivel a lenguaje máquina.
- **Depurador**: Detección de errores de sintaxis en el código durante la compilación.
- **Subida de código**: Cargar el código compilado en la memoria flash del microcontrolador del Arduino.
- **Monitor Serie**: Arduino se comunica con el ordenador por puerto serie mediante USB. El monitor serie es una ventana que nos permite visualizar los datos recibidos y enviados entre el Arduino y el ordenador.

El entorno de trabajo del software IDE es el siguiente:



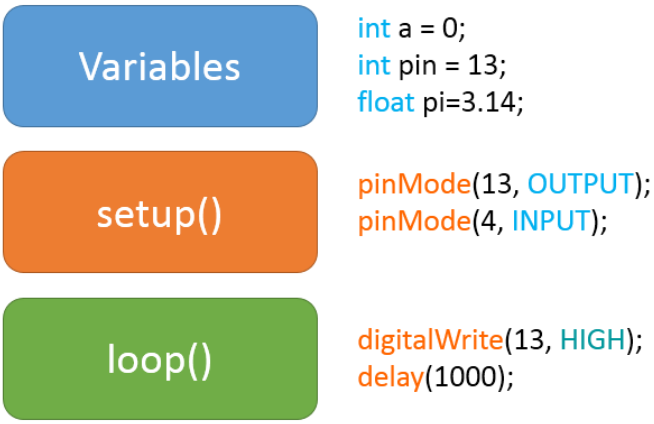
Las herramientas más importantes del software IDE que aprenderemos a manejar son las siguientes:

Símbolo	Descripción
	Crear nuevo proyecto
	Abrir un proyecto
	Guardar proyecto
	Compilar y depurar código
	Cargar programa en la placa de Arduino tras compilar
	Abrir la ventana del monitor serie
-	Selección de placa de desarrollo
-	Gestión de librerías
-	Comentar / Descomentar código

-	Aumentar / Disminuir sangría en código

7.2 Estructura de un programa en Arduino IDE

Como cualquier lenguaje de programación, Arduino IDE tiene su propia estructura básica que es necesario entender y respetar para evitar errores derivados de una falta de orden o conocimiento sobre las características de las diferentes partes que conforman dicha estructura, y que podemos resumir en las tres siguientes:



- **Variables:** En este apartado se definen los tipos de datos que se van a manipular y el nombre que se le va a dar. Según el tipo de dato, se reservará una cantidad diferente de memoria en el microcontrolador.

- **Void setup():** La función *setup()* se invoca una sola vez al comienzo del programa. Permite inicializar los modos de trabajo de los pines, el puerto serie, puesta de un sistema a un estado conocido, etc.

- **Void loop():** Esta función se ejecuta al finalizar la llamada a *setup()*. Se ejecuta de forma cíclica, de modo que en esta parte del código es donde se define el funcionamiento del programa principal.

En el entorno de programación, por defecto, al comenzar un nuevo sketch, se mostrarán estos dos grupos de funciones ya escritas.

7.3 Tipos de datos en Arduino

Como se comentaba anteriormente, las constantes y variables pueden ser de diferentes tipos según el tipo de dato y su tamaño. Generalmente se emplean las siguientes:

Tipo de variable	Descripción
bool	Dato booleano. Puede tomar valor '0' o '1'
int	Dato de tipo entero. Almacena valores desde -32.767 hasta 32.767 (16 bits)
unsigned int	Dato de tipo entero sin signo. Permite almacenar valores positivos desde 0 hasta 65535 (8 bits)
long	Dato de tipo entero extendido. Permite almacenar números enteros en un rango mucho mayor (32 bits)
unsigned long	Similar a *long* pero solo puede almacenar número enteros positivos
float	Dato de punto flotante. Se emplea para almacenar números decimales
Array	Permite definir vectores de datos. Para ello se debe definir el tipo de dato, el nombre y la extensión. *int miArray[5]*

7.4 Operadores en Arduino IDE

Como todo lenguaje de programación, Arduino presenta una serie de operadores tanto de comparación, como lógicos y aritméticos, que se describen a continuación.

Operadores Aritméticos	
Operador	**Descripción**
+/-	Operadores de suma o resta
*****	Operador de multiplicación
/	Operador de división
%	Operador que devuelve el resto de una división entera (módulo)
=	Operador de asignación, permite asignar a la variable que está a la izquierda el valor de la derecha del operador
Operadores de comparación	
!=	Comparación de desigualdad. (Si A es distinto de B)
</>	Operadores de comparación menor o mayor que
<=	Operador de comparación menor o igual que
==	Operador de comparación de igual. (Si A es igual a B)
Operadores booleanos	
!	Negación lógica
&&	Y lógica (AND)
||	O lógica (OR)

Escalado de valores	
map (var, var_min, var_max, desde, hasta)	Escala el valor de la variable a un nuevo rango determinado

7.5 Sintaxis de Arduino

Este lenguaje presenta un conjunto de símbolos e instrucciones que resultan muy útiles.

Símbolo	Descripción
/* */	Permite realizar un comentario de varias líneas de extensión
//	Realiza un comentario que finaliza al acabar la línea
;	Indica el final de una función. En obligatorio escribirlo al final de cada instrucción o asignación de valores.
#define	Se escribe antes del *setup()* y permite definir constantes antes de la compilación del programa, sin consumo de memoria
#include	Al igual que *#define*, se indica antes de comenzar el programa y permite incluir las librerías que contienen las funciones necesarias para nuestro código.

7.6 Principales funciones en Arduino

Antes de nada, debemos comenzar definiendo qué es una función: se trata de un fragmento de código que puede ser invocado desde otras partes del programa tantas veces como se desee. Por regla general necesita una serie de parámetros de entrada y devolverá algún tipo de dato o realizará alguna acción en el Hardware conectado a la placa.

Por defecto, el lenguaje propio de estos microcontroladores provee algunas. Se explicarán las más destacadas, clasificándolas según su uso.

7.6.1 Funciones para las entradas y salidas digitales

Función	Descripción
pinMode(*pin*, *mode*)	Configura el pin especificado para comportarse como entrada o como salida. Mediante el primer parámetro se indica el número de pin a configurar y el segundo de ellos el tipo de entrada o salida, pudiendo ser: INPUT, INPUT_PULLUP o OUTPUT.
digitalWrite(*pin*, *value*)	Escribe en la salida digital especificada un valor lógico de estado alto (HIGH) o bajo (LOW).
digitalRead(*pin*)	Lee el valor de la entrada digital especificada. Valor lógico HIGH si entrada 5V o LOW si entrada 0V.

Véase la Actividad 2, 3 y 4 para poner en práctica estos conceptos.

7.6.2 Funciones para las entradas y salidas analógicas

Función	Descripción
analogRead(*input*)	Lee el valor de tensión entre 0 y 5 voltios de la entrada especificada. Su lectura se realiza en la escala de 0 a 1023 (10 bits).
analogWrite(*pin*, *value*)	Carga en el pin seleccionado una señal PWM con un tiempo en alta especificado entre 0 (0%) y 255 (100%).

7.6.3 Funciones de temporización

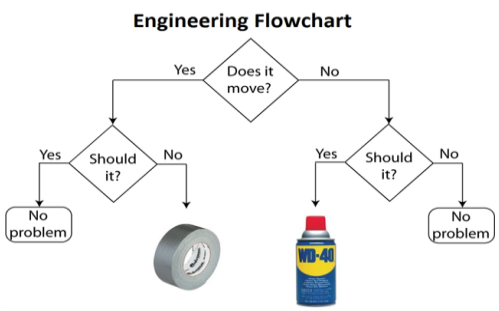
Función	Descripción
delay (*ms*)	Una función de tiempo. Sirve para pausar la ejecución del programa el tiempo especificado (milisegundos)
delayMicroseconds (*us*)	Función similar a *delay()* pero el tiempo de pausa es de microsegundos.

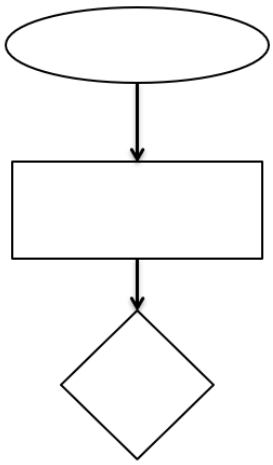
Al emplear este tipo de funciones de *delay()*, se debe de tener cuidado dado que pausa completamente la ejecución del programa. Es útil cuando por ejemplo se desea mantener un LED un determinado tiempo encendido o apagado y no hay que realizar otras comprobaciones, sin embargo, no resulta conveniente si simultáneamente se deben ejecutar otras acciones, leer otras entradas, etc.

7.7 Flujograma o Diagrama de Flujo

El diagrama de flujo o también diagrama de actividades es una manera de representar gráficamente un algoritmo o un proceso de alguna naturaleza, a través de una serie de pasos estructurados y vinculados que permiten su revisión como un todo.

La representación gráfica de estos procesos emplea, en los diagramas de flujo, una serie determinada de figuras geométricas que representan cada paso puntual del proceso que está siendo evaluado. Estas formas definidas de antemano se conectan entre sí a través de flechas y líneas que marcan la dirección del flujo y establecen el recorrido del proceso, como si de un mapa se tratara.



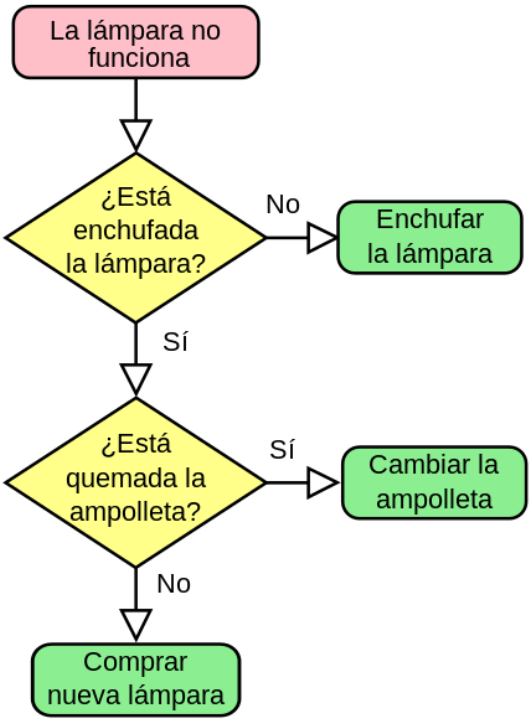


Óvalo o Elipse: Inicio y Final (Abre y cierra el diagrama).

Rectángulo: Actividad (Representa la ejecución de una o más actividades o procedimientos).

Rombo: Decisión (Formula una pregunta o cuestión).

EJEMPLO: Realizar un Diagrama de Flujo para Reparar una lámpara que no enciende.



7.8 Estructuras de control del Arduino IDE

Para controlar la ejecución de un programa no solo es necesario conocer las funciones básicas, también se deben emplear una serie de estructuras que permiten controlar la ejecución del código.

La siguiente tabla muestra las estructuras básicas de programación:

Sentencia	Descripción
If (*condición*) / else	Un tipo de control de flujo que ofrece dos o más posibles caminos a tomar por el programa en función de si se cumplen o no las condiciones especificadas.
for(*inicio*, *condición*, *incremento*)	Permite ejecutar un fragmento de código un número determinado de veces de forma cíclica. Se ejecutará mientras la condición no se cumpla e irá incrementándose de la forma que se indique en incremento.
while(*condición*)	Ejecuta de forma cíclica el conjunto de sentencias especificadas mientras la condición no sea falsa. En el interior del bucle, deberá existir alguna condición que cambie y permita la salida del mismo.
switch… case	Permite ejecutar diferentes secciones de código según el valor de la condición. Cuando el valor coincide con alguno de los casos establecidos, se ejecutará esa parte del código. Al final de cada caso, deberá existir una sentencia de salida *break*.

A continuación, se muestra un ejemplo de cada estructura:

if/else	

if (lectura == HIGH){
digitalWrite(9, HIGH);
}
else{
digitalWrite(9,LOW);
}
	Si el valor de la variable lectura es HIGH, se pondrá a 1 la salida del pin 9. En caso contrario, la salida permanecerá desactivada.

for	

for (int i=0; i<255 ; i++){
  analogWrite(10, i);
  delay(10);
}
	Se realizarán 255 iteraciones, incrementando en cada una de ellas en una unidad el valor de i. En cada una, ese valor se cargará como señal PWM, de modo que el LED que está conectado, irá aumentando su intensidad cada 10 ms.

while	

var = 0;
while (var < 50){
  var ++;
}
	Se resetea la variable *var* a 0. Mientras su valor sea menor que 50, se incrementará su valor en una unidad. Cuando este llegue a 50, el bucle *while* no se ejecutará y la variable será reseteada nuevamente a 0.

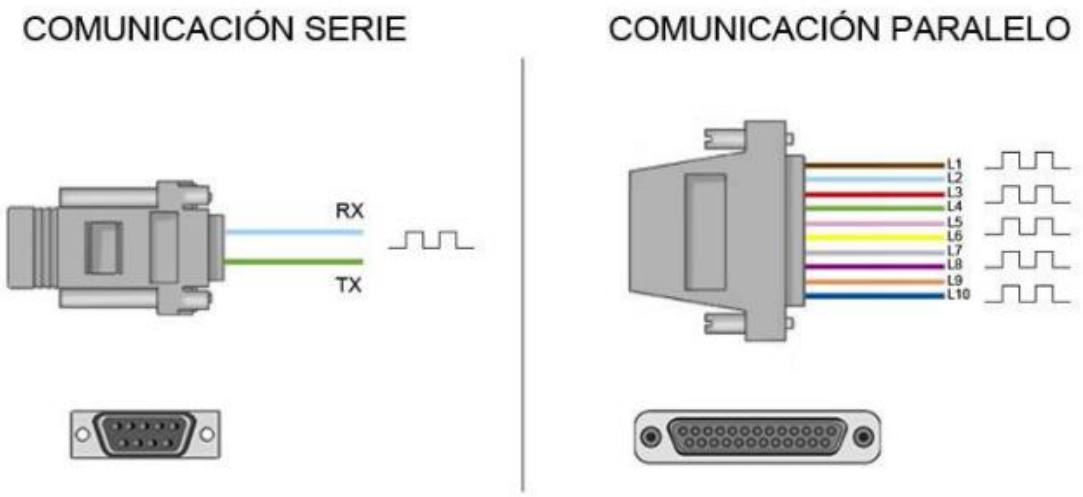
switch … case	

switch (var){
  case 1:
  //Sentencias
  break;
  case 2:
  //Sentencias
  break;
  default:
  //Sentencias
  break;
}
	Si la variable *var*, toma el valor 1 se ejecutarán las sentencias escritas entre el *case* y *break*. De igual forma sucede con el resto de *case*.
	En caso de que la variable no se corresponda con ningún valor se ejecutarán las sentencias del *default*.

7.9 Comunicación por puerto serie

La comunicación por puerto serie es la forma más común de comunicación entre dispositivos, y la principal manera en que el Arduino se comunica con el ordenador y otros componentes.

Un puerto serie envía y recibe información mediante una secuencia de bits de uno a uno a través de un canal de comunicación. A diferencia de este, un puerto paralelo envía la información a través de múltiples canales.



Existen muchos tipos de puertos serie. El más conocido sin duda es el puerto USB (Universal Serial Bus), aunque también podemos encontrar el antiguo puerto serie RS-232, y otros muchos protocolos de comunicación serie como el RS-485, I2C, SPI, etc…

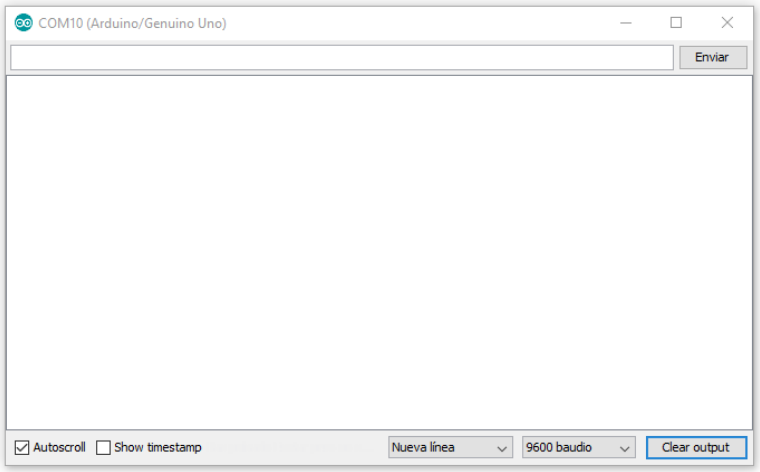
La comunicación serie es la base de las comunicaciones en los microcontroladores, y todos disponen de al menos una unidad UART (Universally Asynchronous Receiver/Transmitter), un tipo de puerto serie que consta de dos canales, uno de transmisión (TX) y otro de recepción (RX).

Las placas Arduino disponen de al menos una unidad UART. En el caso de Arduino UNO, los puertos serie están físicamente conectados a los pines digitales 0 (RX) y 1 (TX), que no podremos utilizar como entradas y salidas digitales corrientes mientras el puerto serie esté en uso.

Podemos comunicar la placa Arduino con el ordenador a través del bus serie utilizando el puerto USB, el mismo puerto que utilizamos para programarlo, estableciendo así un canal de comunicación bidireccional que nos permita enviar y recibir datos.

La interfaz que nos permite visualizar los datos enviados desde el Arduino, así como enviar datos desde el PC al Arduino, es el **Monitor Serial** que viene integrado en Arduino IDE, y que podremos encontrar en la esquina superior derecha del software con el siguiente símbolo:

Una vez abierto, el Monitor Serial de Arduino IDE tiene el siguiente aspecto:



Como puedes ver se trata de una interfaz muy básica y sencilla de utilizar, con los siguientes elementos a destacar:

- **Envío de datos**: Desde el espacio en blanco superior podemos introducir y enviar datos desde el PC al Arduino.
- **Recepción de datos**: Desde el espacio en blanco que ocupa la gran parte de la ventana se muestran los datos recibidos.
- **Autoscroll**: Para efectuar o no el scroll automático de la ventana de recepción de datos.
- **Show timestamp**: Para mostrar la hora exacta de envío de cada dato.
- **Ajuste de línea**: Para añadir un tipo de ajuste de línea al final del mensaje que se desea enviar.
- **Baud Rate o Tasa de baudios**: Velocidad de transmisión medida en tasa de baudios, lo que significa número de baudios por segundo. Un baudio es un símbolo que puede contener varios bits de información. Es muy importante que la tasa de baudios del receptor coincida con la del emisor.

A continuación se indican los nuevos tipos de sentencias que necesitaremos para trabajar la comunicación por puerto serie en Arduino IDE a nivel básico:

Sentencia	Descripción
Serial.begin(9600)	Línea de configuración de la comunicación serie indicando el baudrate o tasa de baudios. Como ejemplo se ha puesto la velocidad típica de 9600 baudios/seg.
Serial.print(val, format)	Se envía por puerto serie el contenido 'val' de la sentencia 'print', con el formato indicado (BIN, OCT, HEX,…), sin salto de línea.

Serial.println(val, format)	Se envía por puerto serie el contenido 'val' de la sentencia 'print', con el formato indicado (BIN, OCT, HEX,…), con salto de línea.
#include "SoftwareSerial.h"	También podemos utilizar esta librería para cuando necesitamos agregar por software un puerto serie adicional haciendo uso de dos pines digitales cualesquiera.
SoftwareSerial miPuertoSerie(2, 4);	Después de declarar la librería, lo primero es crear un objeto de tipo SoftwareSerial con cualquier nombre, indicando los pines digitales a utilizar para el puerto serie. En el ejemplo mostrado el pin 2 es para RX y el pin 4 para TX.
miPuertoSerie.begin(9600)	Para configurar la tasa de baudios cuando usamos la librería SoftwareSerial.h. En el ejemplo se indica la velocidad típica de 9600.

7.10 Funciones para el zumbador

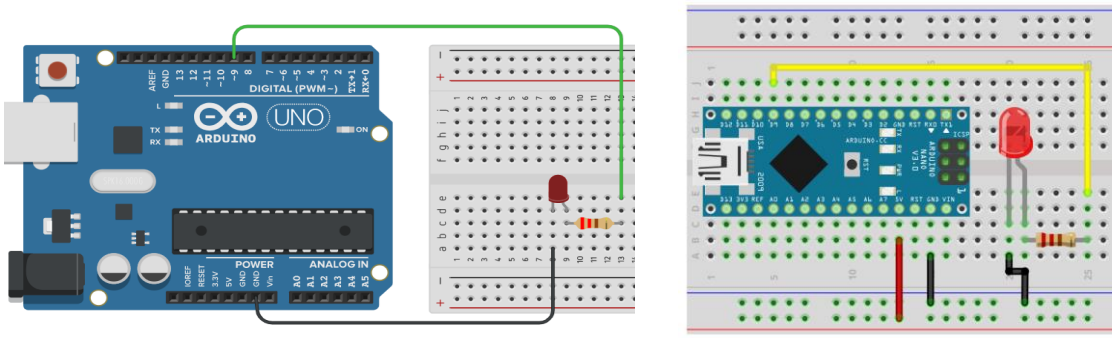
A continuación se indican los nuevos tipos de sentencias que necesitaremos para trabajar con zumbadores de tipo pasivo, para generar diferentes tonos a partir de distintas frecuencias:

Sentencia	Descripción
tone(pin, frecuencia)	Activa un tono de frecuencia determinada en un pin dado
noTone(pin)	Detiene el tono en el pin

8 Ejemplos básicos en Arduino

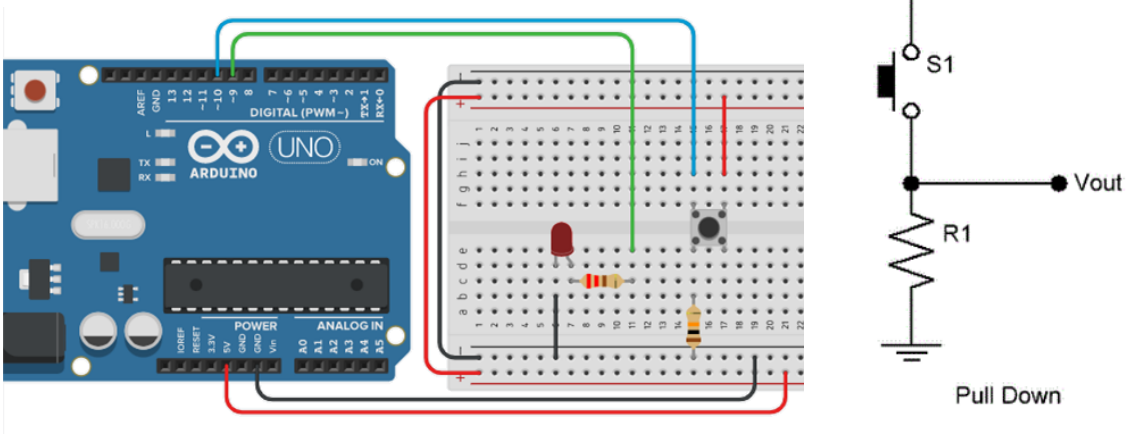
8.1 Actividad 2: LED programado con Arduino IDE

Esta actividad consistirá en realizar un programa básico desde Arduino IDE que logre encender y apagar un LED según el intervalo de tiempo programado, conectado a una de las salidas digitales de la placa Arduino, tal y como se muestra en el siguiente circuito:



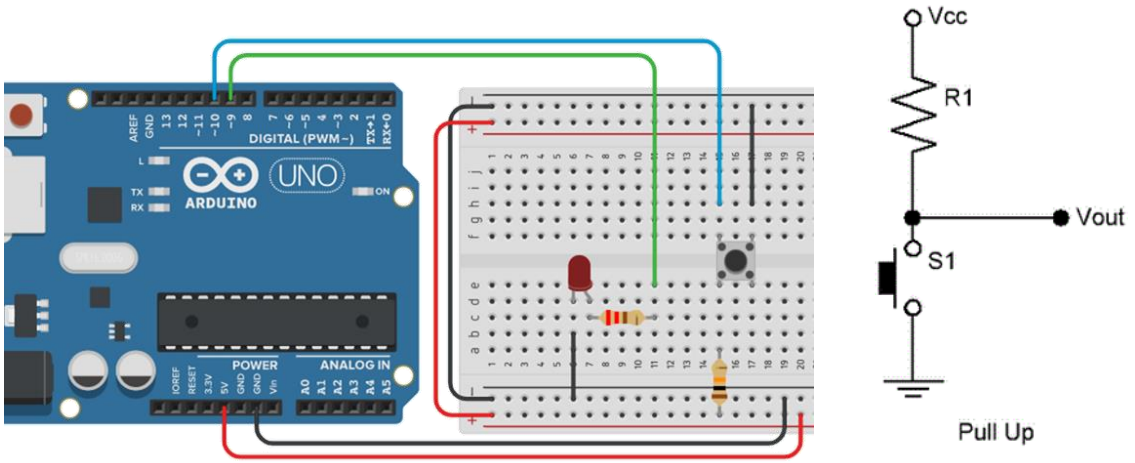
8.2 Actividad 3: LED + Pull-down

Esta actividad consiste en añadir un pulsador en modo Pull-Down, conectado a uno de los pines digitales del Arduino configurado como entrada. Realiza el programa de forma que al mantener pulsado el botón, el LED se mantenga encendido, y que al dejar de pulsar, el LED se mantenga apagado:



8.3 Actividad 4: LED + Pull-Up

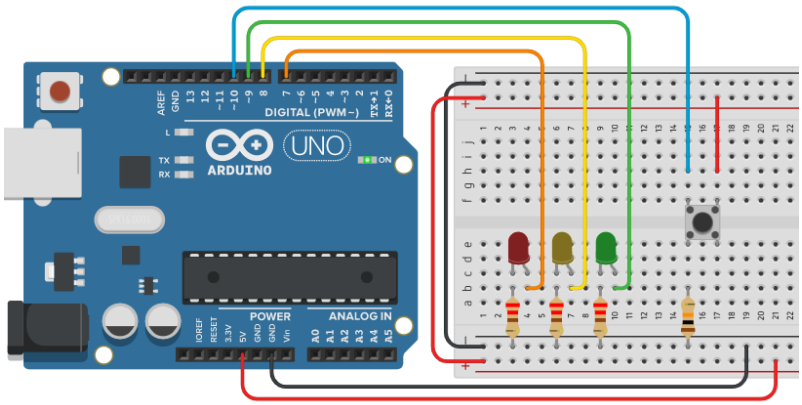
Este circuito funcionaría igual que el Pull-Down, pero conectando el pulsador en modo Pull-Up, de modo que ahora mientras pulsamos el pulsador el led se mantiene apagado y si lo dejamos de pulsar se mantiene encendido:



8.4 Actividad 5: Tres LEDs intermitentes secuencia única

En la siguiente actividad emplearemos un pulsador en configuración pull-down y tres leds conectados de la forma que se muestra en el montaje de la figura. Con cada pulsación, el programa deberá ejecutar una secuencia de encendido y apagado de los leds de forma ordenada, como por ejemplo:

- Orden encendido: verde, amarillo, rojo
- Orden apagado: rojo, amarillo, verde



Manteniendo los mismos intervalos de tiempo entre secuencias de encendido y apagado.

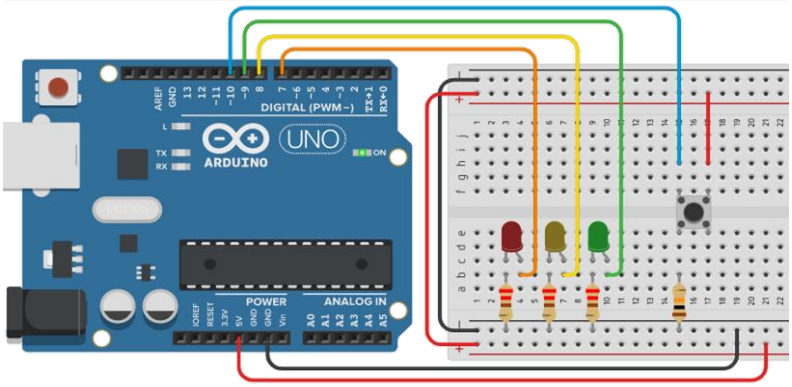
Utilizar para este ejemplo el condicional “if”. Definir también una constante de tipo entero para controlar de forma más sencilla los tiempos de encendido y apagado.

8.5 Actividad 6: Tres LEDs intermitentes con contador

En la siguiente actividad emplearemos un pulsador y tres leds conectados de la forma que se muestra en el montaje de la figura. Con cada pulsación, el programa deberá iniciar una secuencia de encendido y apagado de los leds de forma ordenada, con iguales tiempos de espera entre leds, aplicando el siguiente orden:

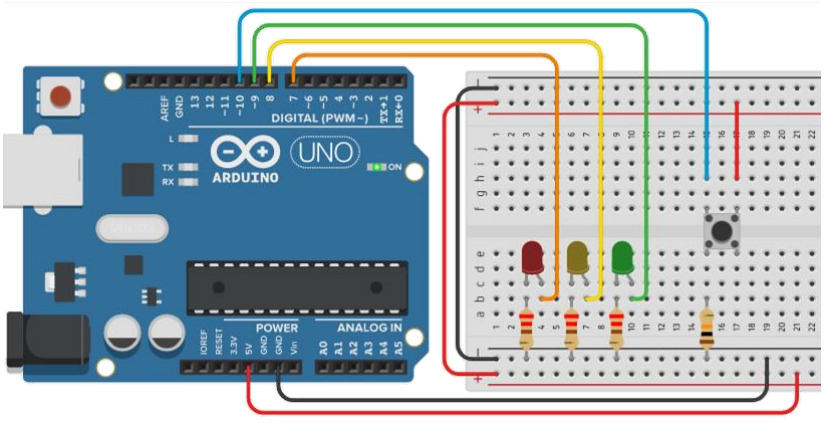
- Orden encendido: verde, amarillo, rojo
- Orden apagado: rojo, amarillo, verde

La secuencia anterior se repetirá un número determinado de veces por medio de un contador que definiremos mediante la sentencia “for”.



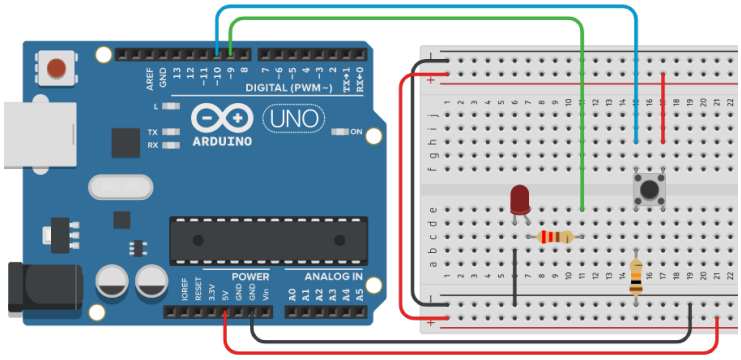
8.6 Actividad 7: Tres LEDs intermitentes secuencia infinita

En esta ocasión vamos a realizar un programa que ejecute una secuencia de encendido y apagado de tres LEDs, en el orden que sea, y que se repita de forma indefinida hasta que se accione una vez el pulsador, en cuyo caso deberá detenerse la secuencia, no pudiendo volver a ejecutarse a menos que se pulse el botón de reset del Arduino.



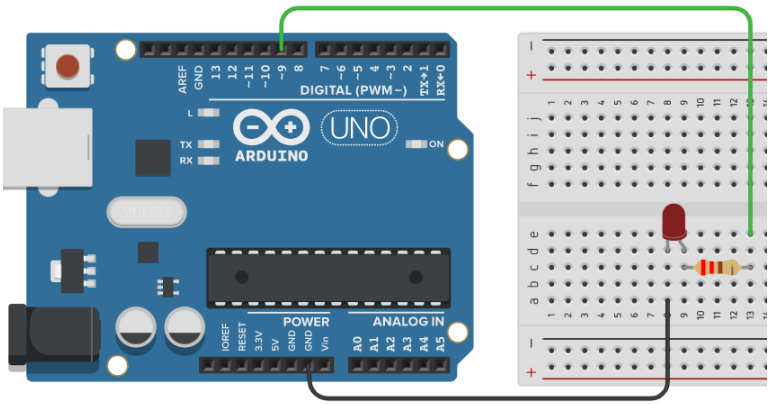
8.7 Actividad 8: Pulsador como interruptor programado con Arduino IDE

En esta actividad emplearemos el mismo circuito simple de LED con lectura de pulsador en configuración Pull_Down, programando en este caso el Arduino para que el pulsador funcione como un interruptor. De este modo el LED cambiará de estado cada vez que accionemos el pulsador.



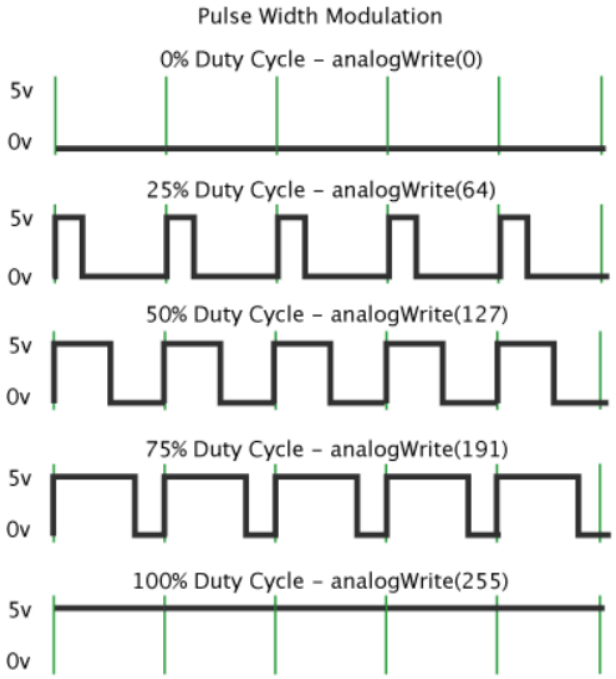
8.8 Actividad 9: Regulación de la intensidad de un led (PWM)

En esta actividad el circuito será muy sencillo, igual que el que utilizamos para controlar el encendido y apagado de un led simple, solo que en este caso haremos uso de señales de tipo PWM para regular el nivel de intensidad de la luz, en lugar de efectuar un mero encendido y apagado del led.



Véase el apartado de señales PWM para comprender las características de este tipo de señales.

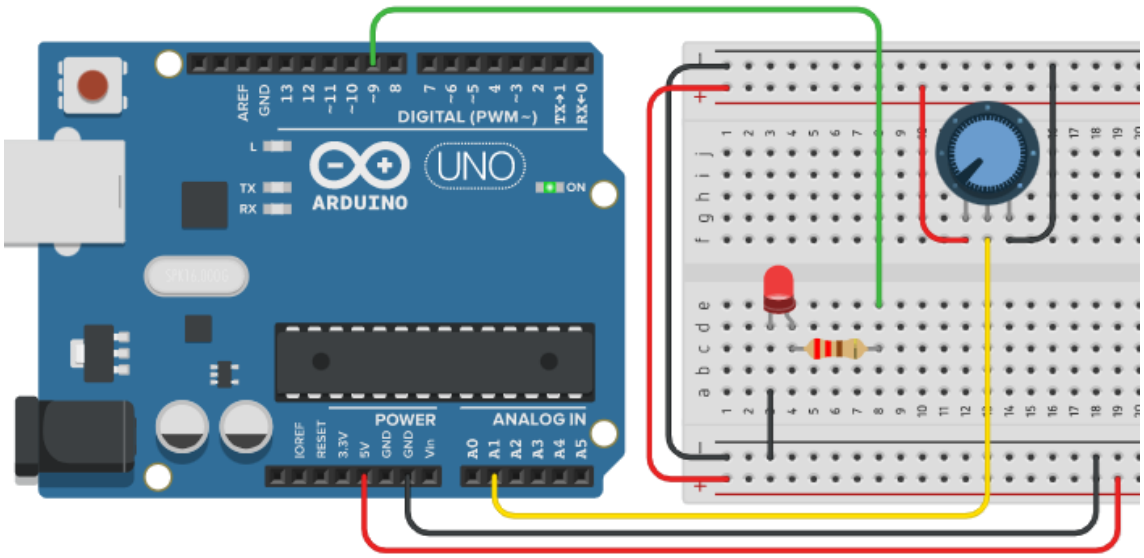
Las salidas PWM que implementa el Arduino tienen resolución de 8 bits, lo que significa que podemos configurar el “duty cycle” de la señal PWM con hasta 256 niveles desde 0% hasta 100%:



Prueba a realizar un código que regule progresivamente la intensidad de un led desde un valor mínimo hasta un valor máximo y viceversa, de forma cíclica. Por ejemplo, incrementando de 1 en 1 cada 10 ms y reduciendo de igual forma.

8.9 Actividad 10: Regulación de la intensidad de un led con un potenciómetro

En esta actividad se realizará la lectura de un potenciómetro (resistencia variable) y se usará ese valor para generar una PWM y así regular la intensidad del LED, con el montaje que a continuación se muestra:



Dependiendo de la posición angular del potenciómetro, la entrada analógica del Arduino recibirá un nivel de tensión de entre 0 y 5V. El Arduino realiza la conversión analógico digital con resolución de 10 bits, mapeando y discretizando los valores analógicos, en valores enteros de 0 a 1023.

Por tanto al realizar un "analogRead()" de la entrada analógica, lo que obtendremos será un valor entre 0 y 1023, en función del valor de tensión a la entrada del pin analógico.

NOTA: Puede añadirse la lectura del puerto Serie para ver el valor analógico medido en el pin A0.

8.10 Actividad 11: Recibir datos desde el Arduino por puerto serie

En este ejemplo vamos a ver las instrucciones básicas para poder visualizar datos enviados desde el Arduino al PC, utilizando como herramienta el monitor serie del Arduino IDE.

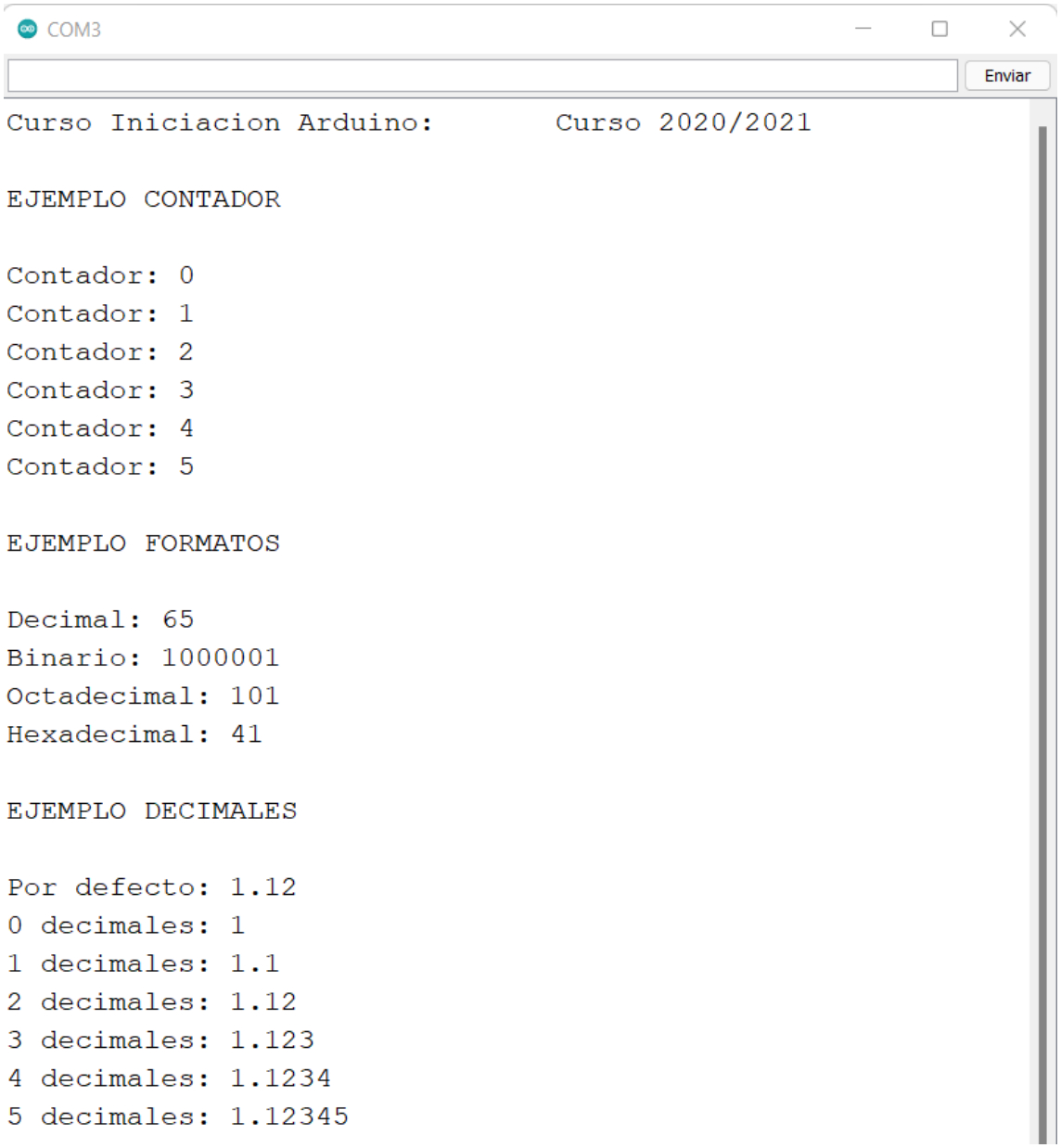
Realizar por tanto un código que imprima texto por puerto serie empleando las sentencias 'Serial.print()' y 'Serial.println()'. Emplear también '\t', '\n' y '\r', para imprimir tabulación, salto de línea y retorno de carro respectivamente.

Implementar también un código que imprima por puerto serie un contador del 1 al 5 a intervalos de tiempo de medio segundo entre cuentas.

Prueba también a imprimir un número en distintos formatos: DEC, BIN, OCT Y HEX.

Finalmente, prueba a imprimir un número de cinco decimales, determinando el número de decimales a mostrar.

Completado el código, el resultado debería ser algo así:



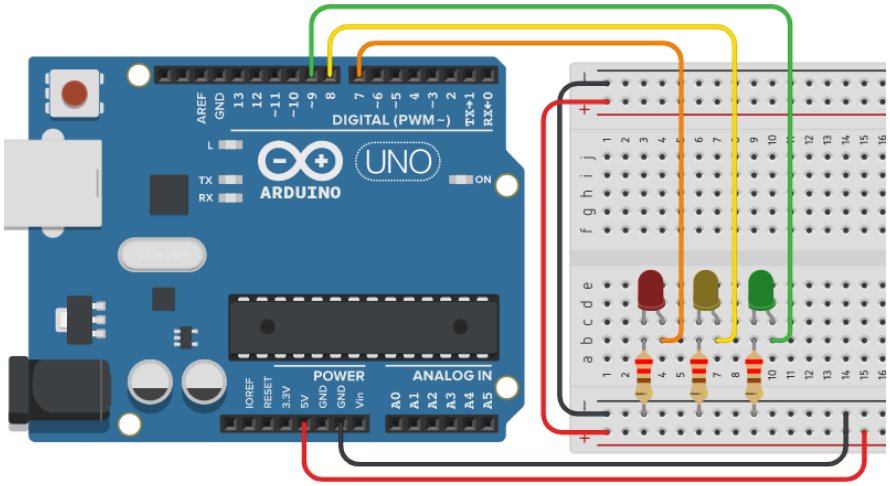
Si se realizase esta actividad en mBlock, hay que tener en cuenta que la versión mBlock 3 dispone de monitor serie, pero no la versión más reciente mBlock 5. Sin embargo, en este último caso, podemos desarrollar el código en mBlock 5 y hacer uso del monitor serie de Arduino IDE.

Otra consideración a tener en cuenta, es que mBlock 5 nos permite enviar texto y valores por puerto serie, pero no podemos determinar el formato ni la resolución de decimales.

8.11 Actividad 12: Enviar datos al Arduino por puerto serie

Con este ejemplo vamos a ver cómo enviar datos desde el PC al Arduino. Conecta tres LEDs a la placa Arduino y realiza un código que te permita encender el LED que quieras mediante instrucciones de teclado, haciendo uso del monitor serie del Arduino IDE.

Por ejemplo, que el LED rojo se encienda enviando el carácter 'r', el amarillo con el carácter 'a', el verde con el carácter 'v', que se enciendan todos con el carácter 't' y que se apaguen todos con el carácter 'x'.

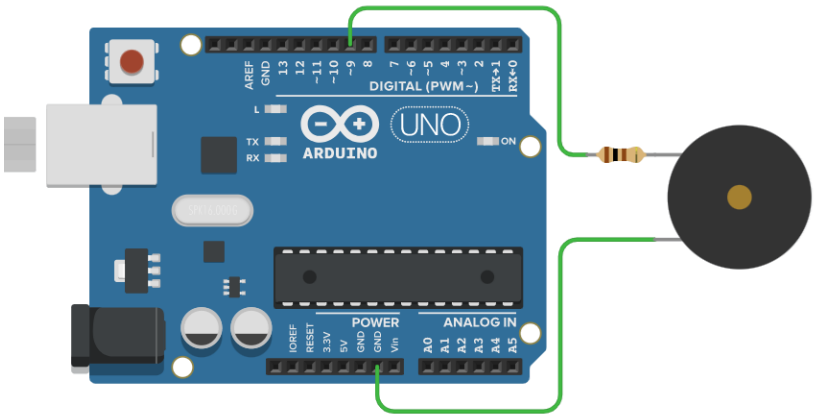


8.12 Actividad 13: Zumbador activo

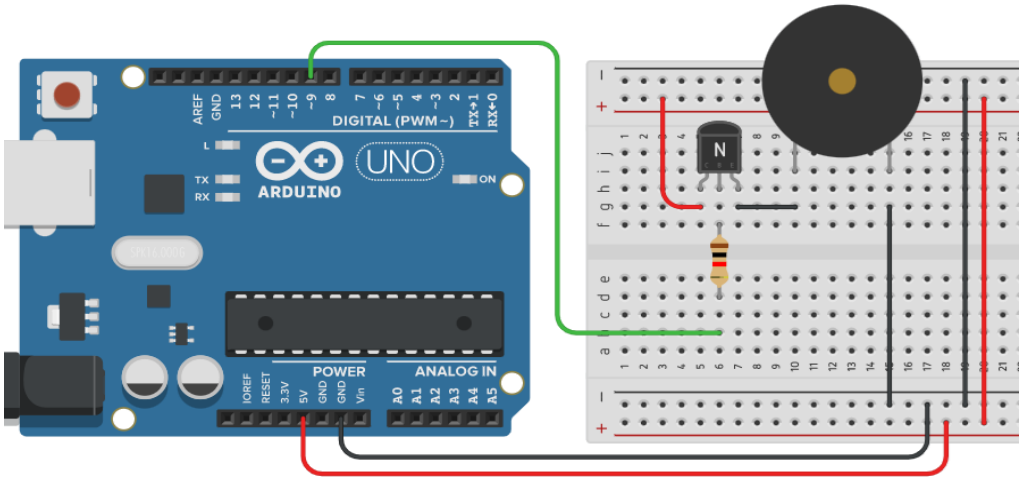
En este caso vamos a ver cómo utilizar un zumbador. Realiza un programa en Arduino capaz de activar y desactivar un zumbador activo de manera constante, con tiempos de encendido y apagado que tu prefieras.

El montaje más básico para realizar esto sería alimentar directamente el zumbador desde un pin digital, lo cual solo será válido para zumbadores de bajo consumo. Recuerda que los pines digitales del Arduino Uno pueden suministrar como máximo 40 mA.

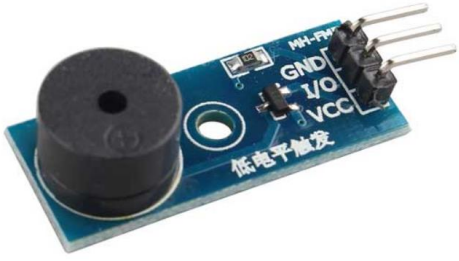
Recuerda también que los zumbadores activos, al tener oscilador, funcionan a frecuencia constante, por lo que para su encendido y apagado basta con usar señales digitales a alta y baja.



Para alimentar zumbadores de mayor consumo debemos hacer uso de un transistor, con circuitos como el que se muestra en la siguiente figura:



Los zumbadores con PCB integrada suelen incluir un circuito con transistor como el de la figura anterior, para soportar niveles de corriente mayores que los que puede suministrar una salida digital estándar. El siguiente módulo por ejemplo viene con su circuito preparado para poder conectar los tres pines (VCC, GND e I/O) del módulo directamente a la placa Arduino:



8.13 Actividad 14: Arrays

En esta actividad vamos a ver como declarar y utilizar 'arrays' en Arduino. Realiza un código que te permita declarar y visualizar por puerto serie diferentes 'arrays' con distintos tipos de datos: enteros, decimales, caracteres, cadenas de texto, etc.

Para este ejemplo no es necesario montar ningún circuito. Solo necesitas la placa Arduino.

8.14 Actividad 15: Zumbador pasivo

Ahora vamos a ver como programar un zumbador de tipo pasivo. Recordemos que a diferencia de activo, estos zumbadores no tienen oscilador integrado, por lo que deben ser alimentados con corriente alterna. El tono emitido por el zumbador vendrá dado por la frecuencia de la señal de alimentación, lo cual da a este tipo de zumbadores la gran ventaja de poder generar melodías.

Para esta actividad, realiza un código que permita producir una melodía. Para este tipo de programas es muy recomendable almacenar las notas musicales en constantes, relacionando cada nota a la frecuencia correspondiente, y definir las notas de una melodía concreta a través de un vector, así como los tiempos de las notas en un segundo vector.

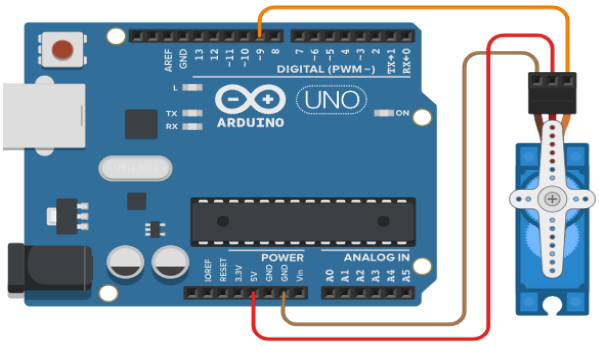
Finalmente y de forma muy sencilla podemos recorrer ambos vectores con un contador (for), empleando las funciones 'Tone()' y 'noTone()' para generar los tonos de la melodía de forma directa.

El circuito a utilizar puede ser cualquiera de los vistos para el zumbador activo.

8.15 Actividad 16: Servomotor

En este caso práctico vamos a programar Arduino para controlar un único servomotor. Para ello podemos utilizar el servo SG-90 que es servomotor económico y sencillo de utilizar. El objetivo será hacer un programa para que el servo barra continuamente su ángulo máximo de operación, oscilando en este caso de 0 a 180 grados y viceversa.

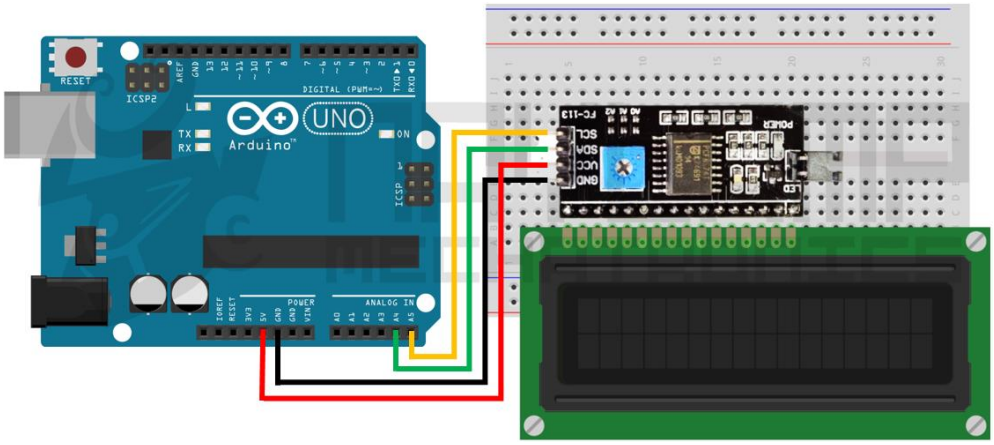
El circuito sería el siguiente:



8.16 Actividad 17: Pantalla LCD

La siguiente actividad será conectar una pantalla LCD I2C 16x2 y mostrar en ella un mensaje personalizado.

Para realizar este montaje debemos emplear los pines I2C del Arduino y la librería LiquidCrystal_I2C. La dirección por defecto de la pantalla es 0x27.



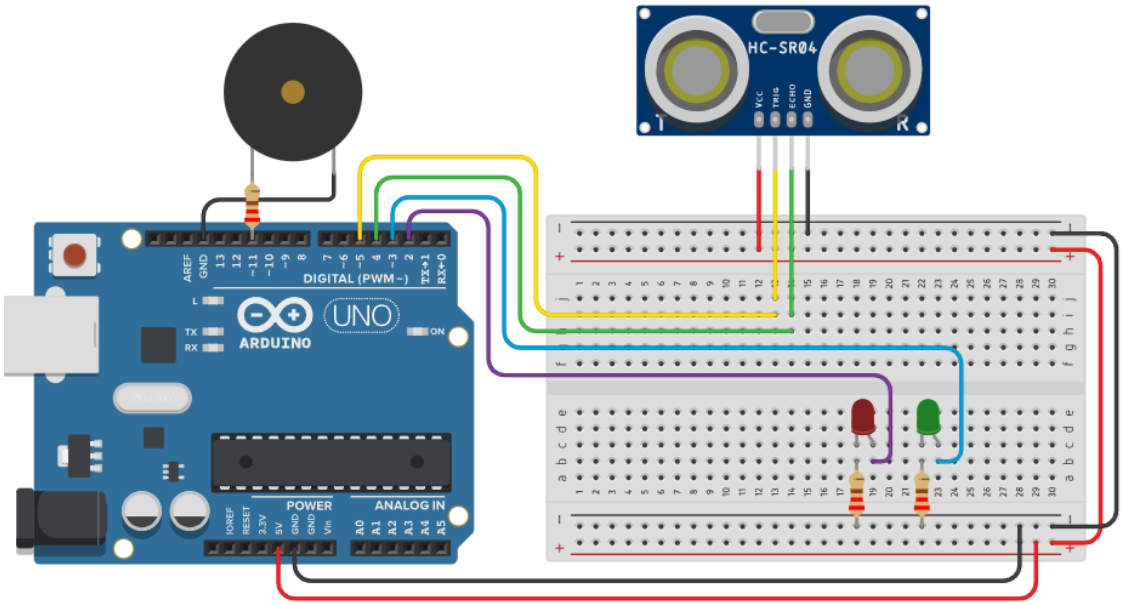
8.17 Actividad 18: Sensor de proximidad por ultrasonidos

Para familiarizarnos con el sensor de ultrasonidos, su esquema de conexión y el algoritmo básico de su funcionamiento, vamos a realizar un programa sencillo.

Por tanto, en este primer ejercicio el objetivo será programar un sensor de proximidad:

- Seleccionar un objeto cualquiera con tamaño suficiente para ser detectable dentro del ángulo de visión del sensor. Si no ves nada a mano, la mano también te vale.
- Utilizar una variable para almacenar la distancia de detección.
- Si el objeto se encuentra dentro de la distancia de detección, se activará una alarma constante por medio de un zumbador, y a la vez se encenderá un LED rojo.
- Si el objeto se encuentra fuera de la distancia de detección, la alarma dejará de sonar y se mantendrá encendido un LED verde.
- Probaremos con diferentes distancias y comprobaremos si éstas son correctas o no.

El esquema electrónico del circuito es el siguiente:



8.18 Actividad 19: Práctica final: sensor de aparcamiento con barrera

Como actividad final, vamos a proceder a desarrollar un circuito en el que vamos a combinar algunos de los elementos que ya hemos visto para desarrollar una barrera de un aparcamiento con indicadores luminosos y un aviso sonoro. Este circuito de cumplir las siguientes características:

- Dispondrá de 3 leds: verde, amarillo y rojo.
- Contará con un servotomor que representará la barrera del aparcamiento
- Tendrá un zumbador que emitirá sonidos.

Se espera del circuito lo siguiente:

- Mientras no haya objetos delante del sensor permanecerá el led verde encendido y la barrera bajada (0º)
- Cuando se acerque un objeto a una distancia inferior a 60 cm, se apagará el led verde y se encenderá el amarillo y se subirá la barrera (90º).
- Si la distancia es inferior a 30 cm, es que el objeto ha llegado al final, por lo que se encenderá el led rojo y se apagará el verde. En ese momento se bajará la barrera.

Como guía:

- Conectaremos los leds a los pines 2, 3 y 4.
- Los pines 5 y 6 los dejaremos para el sensor de ultrasonidos
- En el pin 11 conectaremos el zumbador pasivo.

El esquema electrónico del circuito es el siguiente:

Canarias avanza con Europa
Gobierno de Canarias
Consejería de Economía, Conocimiento y Empleo
Agencia Canaria de Investigación, Innovación y Sociedad de la Información
Fondo Europeo de Desarrollo Regional
Unión Europea