

# Sensores y actuadores



[https://commons.wikimedia.org/wiki/File:Arduino\\_Uno\\_-\\_R3.jpg](https://commons.wikimedia.org/wiki/File:Arduino_Uno_-_R3.jpg)

Práctica 2 – Teoría (v1.7.3 septiembre 2022)

## Software para robots

Cristian González García

[gonzalezcristian@uniovi.es](mailto:gonzalezcristian@uniovi.es)

Basado en el material original de Jordán Pascual Espada

# Índice

1.	Introducción .....	2
2.	Sensores .....	3
	Sensor de luz .....	3
	Sensor PIR .....	3
	Sensor de vibración.....	4
	Sensor de proximidad por infrarrojos.....	5
	Sensor de ultrasonidos HC-SR04.....	6
	Cálculo de la distancia.....	6
	Teclado .....	8
3.	Actuadores .....	8
	Pantalla de 7 segmentos y 4 dígitos.....	8
	Servomotor 180º.....	9
4.	Arduino.....	10
	Shield.....	10
	Toma de tiempos .....	10
	Pin 13 del Arudino.....	11
	Importar librería.....	12
	Gestor de librerías.....	12
	Manualmente.....	13
5.	Ejemplos .....	16
	Detección de luz.....	16
	Detección de movimiento.....	17
	Otros sensores digitales: sensor de vibración, infrarrojos, ... ..	19
	Control de distancia mínima .....	19
	Pantalla de segmentos.....	21
	LED RGB.....	22
	Servomotor 180º .....	24
	Teclado .....	25

## 1. Introducción

El objetivo principal de esta práctica es utilizar las entradas y salidas de Arduino para controlar otro tipo de elementos más complejos (sensores, actuadores y elementos de interfaz de usuario).

Los **sensores** nos permiten obtener información del entorno. Para utilizar un sensor desde Arduino se suele utilizar al menos una entrada digital o analógica, dependiendo del tipo de sensor. Algunos sensores pueden requerir el uso de varias salidas, o de varias entradas, para enviar parámetros que configuren su funcionamiento o sean parte de él. Un ejemplo es el de ultrasonidos, que necesita primero enviar una onda, que será la que utilice para detectar la distancia.

La mayor parte de **actuadores** se gestionan mediante el uso de salidas. Los actuadores pueden ser: pantallas, motores, LEDs, etc. Estos pueden utilizar más de una salida, analógicos o digitales, dependiendo de la complejidad del actuador y el modelo.

Todos los ejercicios se pueden realizar utilizando el *Shield* de sensores o sin él, utilizando los pines equivalentes del Arduino. El *Shield* lo que proporciona es una pequeña «facilidad» en la conexión de sensores y actuadores y proporciona más pines GND y V.

## 2. Sensores

### Sensor de luz



**Sensor de luz:** este tipo de sensores deben conectarse a un **pin analógico**, pues nos devolverá más de 2 valores.

Retorna un valor que suele oscilar entre 0 – 620: a mayor valor retornado mayor cantidad de luz está detectando.

En este modelo:

- el **pin negro va a GND**, pues es el negativo.
- el **pin central y rojo a V**, a la potencia.
- el **pin restante y de otro color** (azul o blanco, depende del modelo) **a un pin analógico de lectura**. Es decir, a un pin analógico entre A0 y A5 en el Arduino UNO.

### Sensor PIR



**Sensor de movimiento pasivo, PIR, o pasivo infrarrojo:** permite detectar movimiento cercano ya que detecta la diferencia de calor emitido por las fuentes de energía, respecto al espacio de alrededor. Por ejemplo, detecta el calor emitido por el cuerpo humano o el de los animales. Es pasivo debido a que recibe radiaciones y no las emite. El sensor está recubierto por una lente de Fresnel que le ayuda a incrementar el área de funcionamiento.

Cuando se instalan, necesitan «aclimatarse» a las radiaciones del ambiente, pues reciben radiaciones de todo su alrededor. Una vez estén «acostumbrados» son capaces de detectar las variaciones en ese ambiente, como cuando una persona entra en la habitación, ya que al entrar se modifica la radiación infrarroja del ambiente.

Cuando detecta movimiento mantiene una salida de 5V, mientras que cuando no hay movimiento mantiene una salida de 0V (algunas marcas funcionan de la forma contraria). Para detectar movimiento, el sensor utiliza un detector de niveles de radiación infrarroja debido a que la mayor parte de cosas que se mueven generan un pequeño nivel de radiación infrarroja (calor).

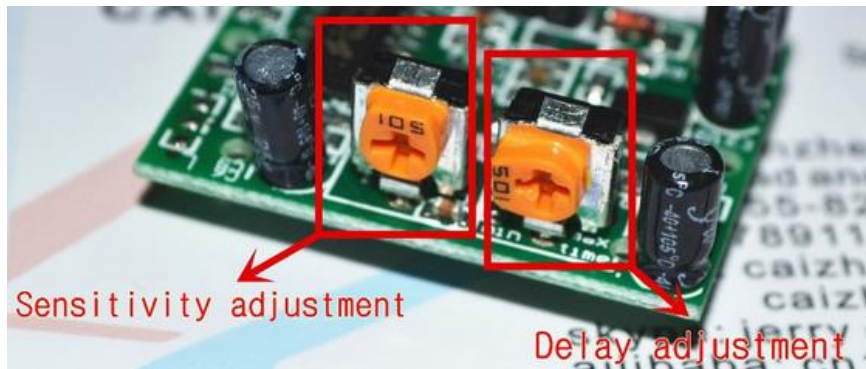
Dependiendo del tipo de sensor, este puede tener diferente distancia de detección y diferente ángulo de detección. Unos valores típicos para un sensor de este tamaño son 7 metros y 110°.

En este modelo:

- el **pin GND** es el negativo.
- el **pin VCC**, a la potencia.

- el **pin OUT** va a un **pin digital de lectura**. Es decir, a un pin digital entre 2 y 12 en el Arduino UNO.

Algunos sensores como este incluyen cierta capacidad de calibración. En este caso, tenemos dos tornillos en la parte inferior que nos permiten ajustar el grado de sensibilidad del sensor y el retardo entre detecciones.

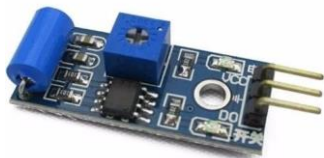


#### Notas importantes:

Los reflejos o pequeños movimientos de la mesa pueden hacer que detecte movimiento. Recordad, que el sensor tiene que «aclimatarse» al ambiente.

Puede ser que la lente Fresnel se caiga del sensor, pues el acoplamiento entre ambos es bastante bajo. Simplemente hay que volver a colocarla haciendo que encaje y se asiente correctamente sobre la palca del sensor.

#### Sensor de vibración



**Sensor de vibración SW-400:** permite detectar vibraciones por encima de un umbral. El umbral se puede configurar de forma física ajustando el tornillo.

Se suelen utilizar en aplicaciones industriales para medir la fiabilidad de motores, engranajes, bombas, máquinas, etc. De esta manera, se pueda tener información sobre un mal comportamiento para repararlo cuanto antes.

Cuando no detecta vibración retorna una salida de 0V (LOW) y cuando la detecta retorna 5V (HIGH).

La patilla **DO** se conecta a un pin digital, **VCC** a un voltaje, y **GND** que es la patilla central a la toma tierra/negativo.

**Ajuste:** si giramos el tornillo hasta su límite en sentido antihorario, lo colocamos en su mayor nivel de sensibilidad y podrá detectar vibraciones muy leves. Cuanto más giremos el tornillo de forma horaria más reduciremos la sensibilidad y más fuertes deberán ser las vibraciones para ser detectadas.



## Sensor de proximidad por infrarrojos



**Sensor de proximidad por infrarrojos:** permite detectar elementos colocados delante del sensor. La distancia de detección se puede configurar de forma física ajustando el tornillo.

Está compuesto por un emisor de infrarrojos que emite pulsos de forma continua y por un detector de infrarrojos que detecta cuando la luz infrarroja rebota contra un objeto, captando ese rebote. De esta forma sabemos que hay algo delante del sensor. En función del ajuste, detectará que hay objetos delante de él a mayor o menor distancia.

Cuando no detecta ningún elemento retorna una salida de 5V (HIGH), y cuando detecta un elemento retorna 0V (LOW). Este es el **funcionamiento opuesto de los sensores anteriores**, cuidado.

El LED que está cerca de la salida **OUT** se enciende en rojo cuando detecta algo.

La patilla **OUT** se conecta a un pin digital, la **VCC** a un voltaje y la patilla **GND** a una tierra (G) del Arduino.

**Ajuste:** si giramos el tornillo hasta su límite en sentido antihorario, lo colocamos en su menor nivel de detección y no detectará absolutamente nada. Fijaros que, en este caso, es totalmente al contrario que el sensor de vibración. Esto depende del fabricante del sensor.

Cuanto más giremos el tornillo de forma horaria, más aumentaremos el nivel de detección en centímetros. Primero, detectará elementos que estén únicamente a 0-2 cm. Según vayamos girando el tornillo podrá llegar a detectar a 0-8 cm, aproximadamente.



## Sensor de ultrasonidos HC-SR04



**Sensor de ultrasonidos SR04:** detecta la distancia a la que se encuentra un objeto por medio del uso de ultrasonidos.

Este sensor emite y recibe el rebote de un ultrasonido. Luego, si calculamos el tiempo transcurrido entre estas dos acciones es posible saber si la onda colisionó con algún objeto y su distancia.

Dependiendo del fabricante y el tipo del sensor, estos pueden tener más o menos precisión y alcance: esta versión **permite medir distancias de 2cm y 4m con una precisión aproximada de 3mm**, teóricamente, y **dependiente del modelo**. No obstante, en la práctica, cuanto más cerca de los extremos se quiera medir, más problemas dará, aunque esto varía según el sensor.

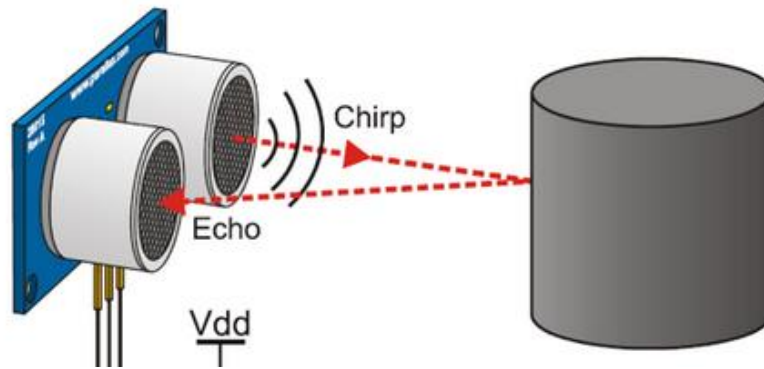
El sensor tiene cuatro pines:

- **VCC** va a la potencia.
- **GND** va al negativo.
- **Trig** se conecta a un pin digital y se encargará de enviar el pulso de ultrasonidos.
- **Echo** se conecta a otro pin digital y se encargará de recibir el eco de dicho pulso.

**Entre pulsos hay que esperar al menos 10-20 microsegundos para evitar problemas.** Además, **en el primer disparo hay que ponerlo a LOW y esperar al menos 5 microsegundos para asegurar realizar un «disparo limpio».**

### Cálculo de la distancia

1. Para saber a qué distancia está un objeto, deberemos calcularla. Para ello, sabemos que un sonido se mueve a 0,034 cm/microsegundo:
  - a. Velocidad del sonido:  $343,2 \text{ m/s} \rightarrow 343 \text{ m/s} * 100\text{cm/m} * 1\text{s}/1000000 \text{ ms} = 0,03432 \text{ cm/ms}$
2. Hay que tener en cuenta, que la onda recorre el doble de distancia de a la que realmente está el objeto, pues la onda la enviamos, choca con el objeto, y vuelve. Así pues, hay que dividir entre 2 el cálculo:
  - a.  $0,03432 / 2 = 0,01716$
3. Problemas:
  - a. Podría pasar que se false la lectura por los rebotes que pueda dar el pulso contra varios objetos o contra el propio objeto si no impacta de forma directa o tiene ángulos.
  - b. Si enviamos **dos pulsos muy seguidos** que recibamos el primero después de enviar el segundo, lo que dará una lectura errónea. Para esto, hay que tener en cuenta que el sonido se desplaza a 0,03432 cm/ms (**1 segundo son 34,32 cm**), así que a distancias más largas necesitará mayor tiempo de espera.



### Notas importantes:

En ocasiones el sensor de sonidos deja de emitir mediciones o solo emite mediciones erróneas. En ese caso, hay que desconectar el USB y volver a conectarlo.

Algunas veces, algunos sensores no bajan de 5 cm de distancia mínima, o bien, devuelven siempre 0 en la primera petición. Esto puede ser problemas del propio sensor. Una posible solución es desconectar el USB y volver a conectarlo. Si sigue dando ese problema hay que hacer un «apaño» por software.

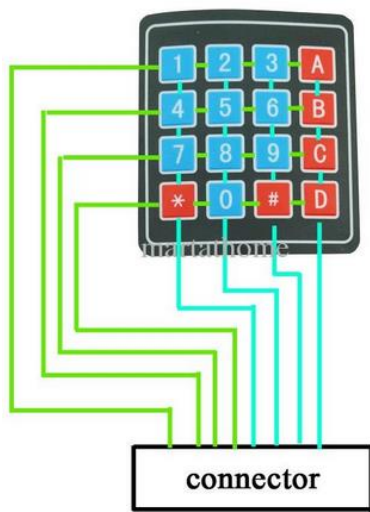
Hay q tener cuidado con los valores del tiempo de la función `millis()`, pues pueden llegar a ser muy grandes. En este caso, si se ponen como enteros, podría haber un *overflow* y pasar a ser negativos o muy bajos, lo que puede ocasionar problemas. Utilizad entonces el tipo de variable *float* o *double*.

Algunos modelos de sensor de ultrasonidos introducen demasiado «ruido». En estos casos, puede ser una buena estrategia realizar un conjunto de mediciones muy seguidas y tomar como válida la mediana (<http://playground.arduino.cc/Main/Average>).

Hay que tener en cuenta que, dependiendo de la orientación, la onda puede rebotar sin retornar al lector de ultrasonidos. En este caso, el sensor pensará que no hay nada delante. Es justo por eso por lo que los sensores de ultrasonidos suelen colocarse en servomotores, sobre todo en los robots móviles, y así poder lanzar la onda en varias direcciones y tener mayores posibilidades de éxito.



## Teclado



**Teclado matricial:** los teclados matriciales permiten capturar pulsaciones en una matriz de botones utilizando el indicador de la fila y la columna pulsada.

Para detectar la pulsación se conectan cuatro pines como entradas a las columnas y cuatro pines como salidas a las filas, o viceversa.

El modo de conexión más común es utilizar 8 pines digitales para conectarlo, pero esto, en un Arduino UNO nos dejaría solo con 3 pines digitales usables libres para el resto de los elementos que quisiéramos utilizar. Por eso, para evitar este problema, podemos optar por conectar cuatro cables, por ejemplo, los primeros de la izquierda (verdes) a los pines digitales (2 - 5) y los otros cuatro cables (azules) a

los pines analógicos (A0 - A3). Si los conectáis del revés, cambiara la asignación de la matriz de símbolos y sería otra diferente.

## 3. Actuadores

### Pantalla de 7 segmentos y 4 dígitos



**Pantalla de 7 segmentos y 4 dígitos:** permite representar valores de cuatro dígitos.

Este componente incluye un chip que permite controlar la pantalla haciendo uso únicamente de una entrada y una salida digital.

Este modelo también permite regular el brillo de la luz emitida.

Las conexiones necesarias son:

- **CLK (Pin Digital PWM).**
- **DIO (Data IO, Pin Digital).**
- **VCC (5V).**
- **GND (GND).**

Librería:

- TM1637
- <https://github.com/avishorp/TM1637>

## Servomotor 180º



**Servomotor 180º:** es un pequeño motor de corriente continua que incluye en su interior una especie de potenciómetro que le permite saber su posición exacta y controlarla.



Gracias a esta particularidad, podemos hacer que el motor se mueva a un ángulo exacto. Existen servomotores de diferentes tipos: unos permiten rotar 360° (rotación continua) y otros 180° o menos, como pueden los de 60°, por ejemplo.

A parte del ángulo de rotación soportado, los servomotores tienen otras características interesantes como el peso que pueden mover o la velocidad a la que se mueven.

El servomotor tiene tres pines:

- **Rojo** que se conecta a **5V** (aunque podría ser más).
- **Negro o marrón** que se conecta a **GND**.
- **Blanco o naranja** que se conecta a un **pin digital PWM**.

Más información:

- <https://www.arduino.cc/en/Reference/Servo>
- <https://github.com/arduino-libraries/Servo/blob/master/src/Servo.h>
-

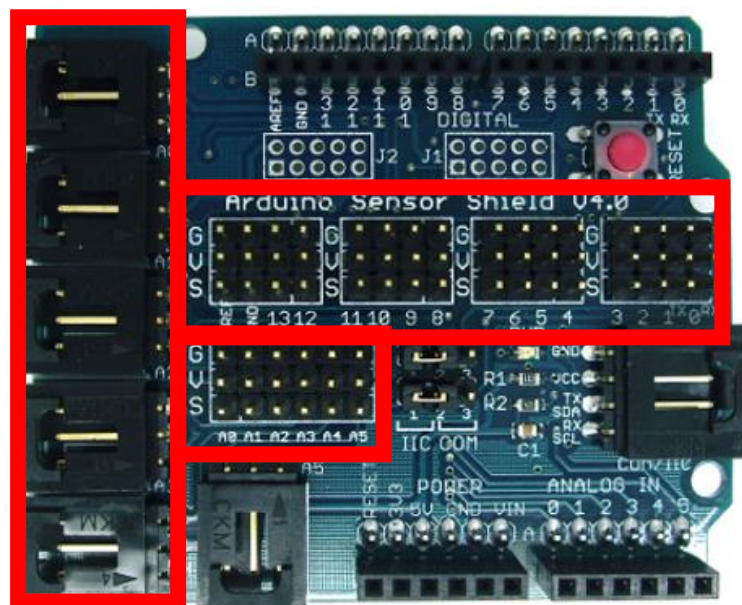
## 4. Arduino

### Shield

Este es el **Arduino Sensor Shield v4.0**. Este *shield* nos permite extender la funcionalidad del Arduino y que nos sea más fácil conectar sensores, sobre todo si estos son los que vienen con un cable entrecruzado inseparable de 3 pines.

Para acoplarlo a la placa Arduino, los pines machos del *shield* deben coincidir perfectamente con los mismos pines hembra del Arduino. Es decir, el pin A0 del *shield* debe enchufarse en el pin A0 del Arduino, y así sucesivamente.

La característica más destacada de este *shield* es la existencia de un pin de tierra (G) y uno positivo (V) por cada pin digital (S superior) y analógico (S inferior). Además del tipo de entrada del lateral izquierdo para los pines analógicos, siendo esta diferente a las del Arduino. El resto del funcionamiento es exactamente igual que en el Arduino.



### Toma de tiempos

Como en otros lenguajes de programación, Arduino permite tomar tiempos. Para ello, hay que utilizar la función `millis()`;

Cuando se almacena el valor de esta función en una variable hay que tener en cuenta el tipo de variable, pues es posible que sea muy grande y provoque un *overflow*. Esto sucede si se utilizan variables de tipo `int`, haciendo que se desborde y el número pase a ser negativo y pase de un valor alto a otro bajo rápidamente, ocasionando un mal funcionamiento del programa. Se recomienda utilizar el tipo `long` o `double`.

Incluso, podemos utilizar la versión sin signo de algunos tipos de datos, que se declara escribiendo la palabra reservada `unsigned` delante del tipo de datos. Tenemos dos: `unsigned int` y `unsigned long`. Esto permite usar la capacidad de ese tipo de dato, pero solo para números positivos, doblando así la capacidad de números positivos del tipo de dato.

Además, si se deja funcionando de seguido sobre aproximadamente 50 días, la función hará *overflow* y volverá a cero.

Más información en:  
<https://www.arduino.cc/reference/en/language/functions/time/millis/>

Ejemplo de su uso:

```
int led13 = 13;
double inicioCuentaTiempo;

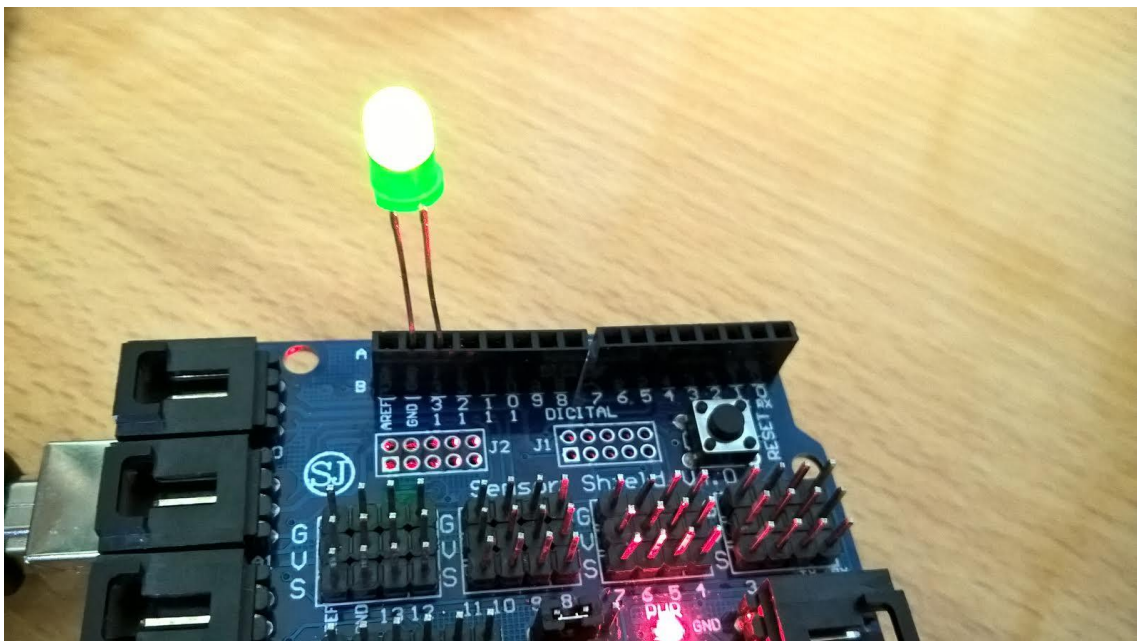
void setup() {
  Serial.begin(9600); // Iniciar el Serial
  pinMode(led13, OUTPUT);

  // Instante de tiempo que empieza a contar!
  inicioCuentaTiempo = millis();
}

void loop() {
  if (millis() - inicioCuentaTiempo >= 5000){
    // Tiempo actual - inicio Cuenta tiempo >= 5000
    // Han pasado 5 segundos, enciendiendo el led
    digitalWrite(led13, HIGH);
  }
}
```

### Pin 13 del Arudino

El pin 13 del Arudino tiene una resistencia de  $220\Omega$  integrada, así como un pequeño LED cerca de este pin. También, dispone de un pin GND justo a su izquierda, por lo que podemos conectar un led de forma rápida para probar si el Arduino está encendido y funcionando, o si el *Shield* que hemos puesto está bien conectado, o probar si un LED funciona o no. Recordad, la patilla larga del LED es el positivo, así que irá al pin 13.



## Importar librería

Hay dos formas de importar una librería en Arduino.

### Gestor de librerías

La primera de todas es ver si existe en el repositorio de Arduino. Para ello, en el IDE, deberemos ir a Herramientas -> Administrar bibliotecas... (Ilustración 1).

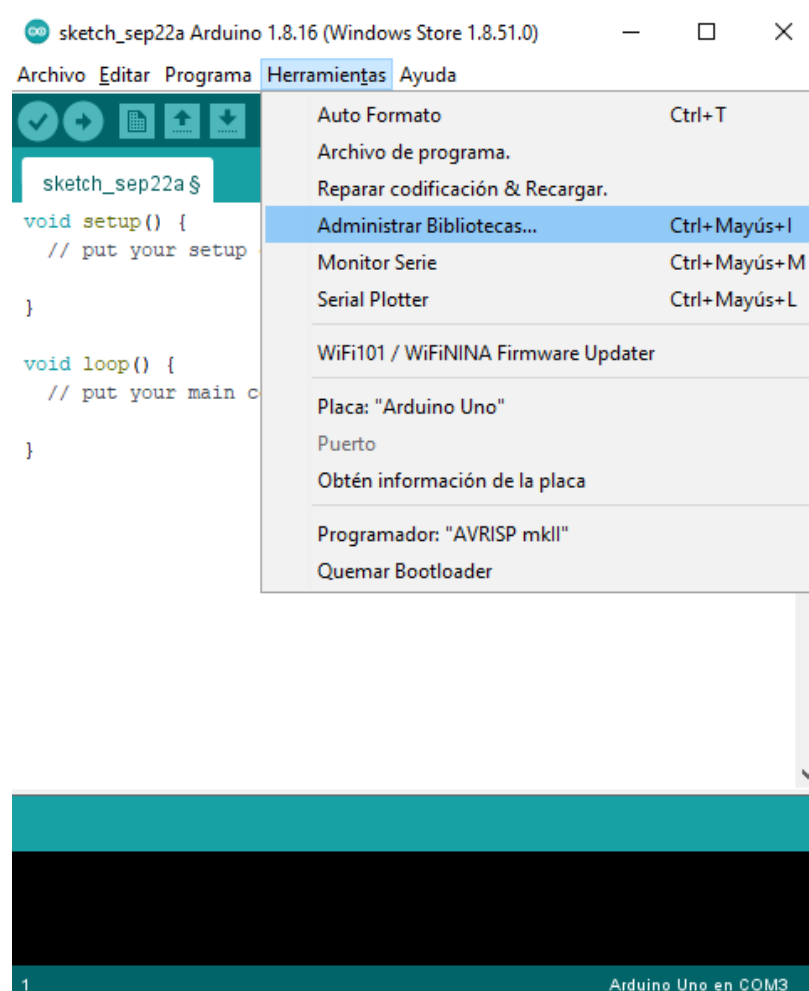


Ilustración 1 Administración de bibliotecas en el IDE de Arduino

Una vez en el gestor (Ilustración 2), tendremos acceso a todas y podremos realizar búsquedas por:

- Tipo: actualizable, instalado, Arduino, compañero, recomendado, contribución, retirado.
- Tema: comunicación, procesando datos, datos almacenados, control de dispositivos, pantalla, otro, sensores, señal entrada/salida, temporizador, sin categoría.
- Palabra: búsqueda a partir de la palabra introducida.

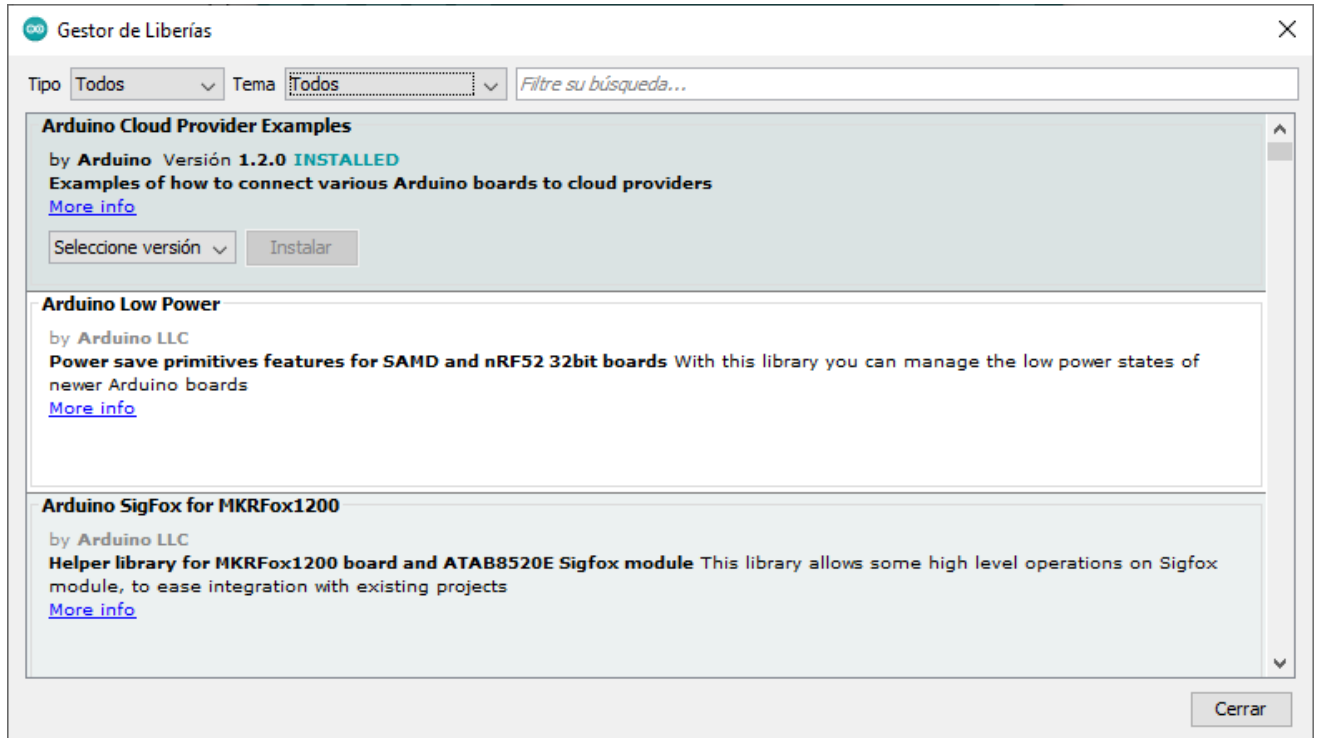


Ilustración 2 Gestor de bibliotecas

Además, permite instalar una versión en concreto si se requiere, la última, actualizar las que ya se tienen instaladas, etc.

### Manualmente

En el caso de que la librería no exista en el gestor, podemos añadir de forma manual. Por ejemplo, si quisiéramos añadir la librería para manejar el teclado (KeyPad), tendremos que descargarla de: [https://cdn.shopify.com/s/files/1/0557/2945/files/KeyPad\\_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262](https://cdn.shopify.com/s/files/1/0557/2945/files/KeyPad_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262)

Una vez descargada, vamos a **Programa -> Include Library -> Add. Zip Library** (Ilustración 3).

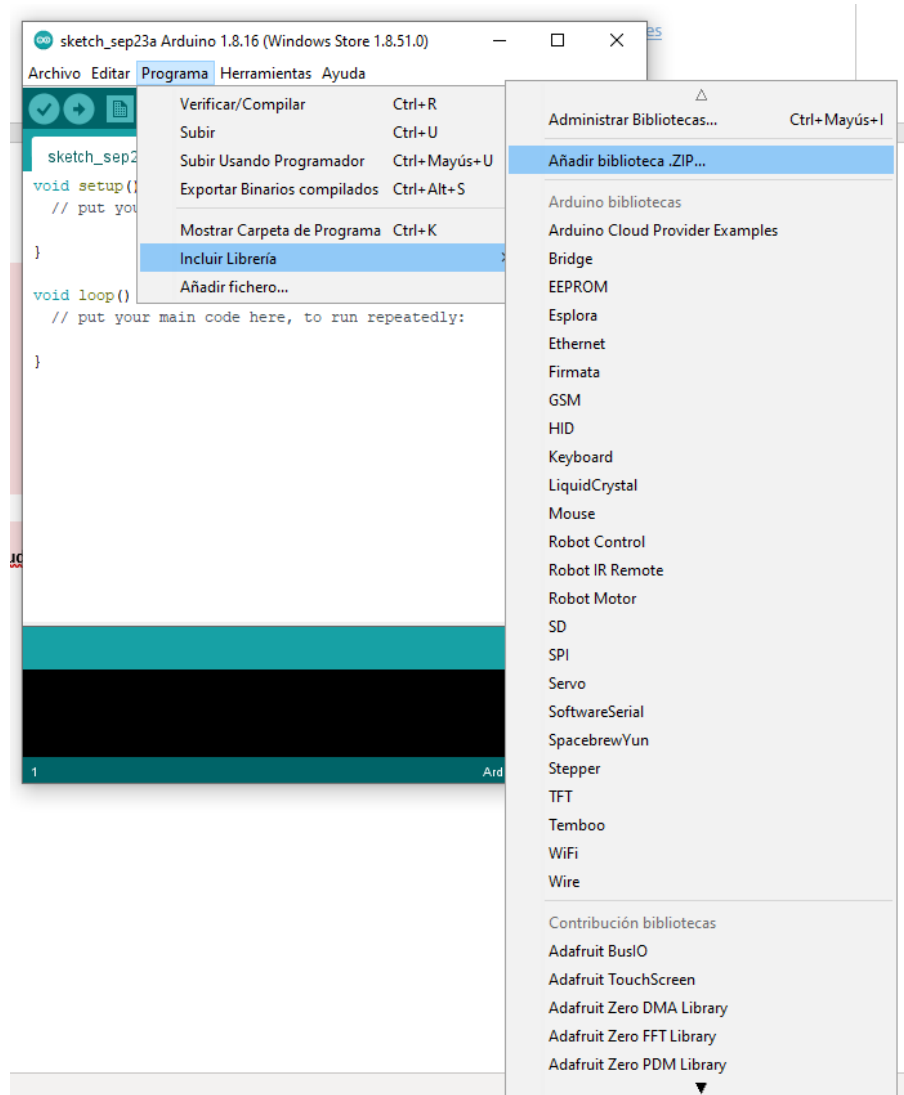


Ilustración 3 Añadir librería manualmente

A partir de este momento la librería estará disponible en la lista de librerías: **Programa -> Include Library -> Keypad**

Si ya existiera os dará un error como muestra la Ilustración 4.

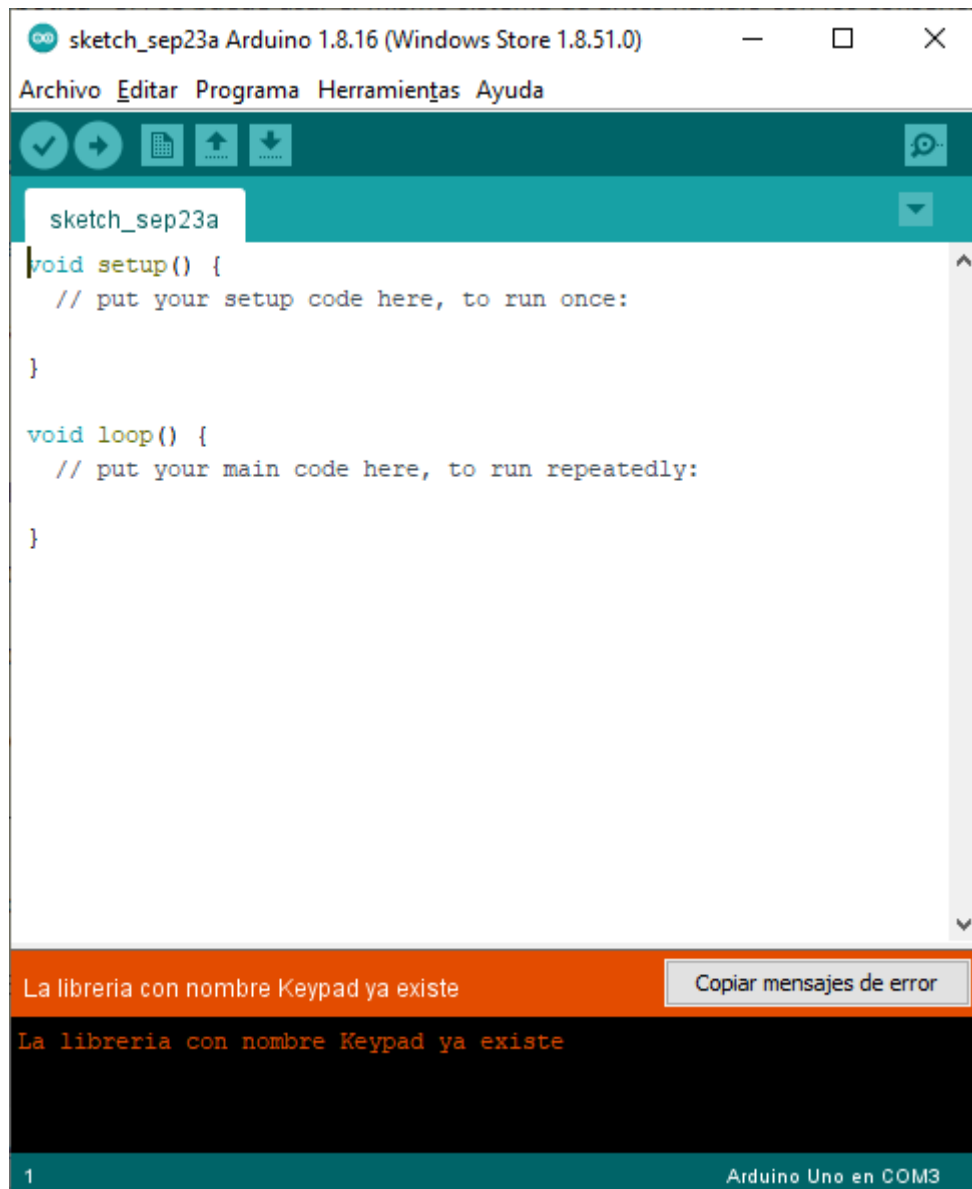


Ilustración 4 Error al cargar librería ya existente



## 5. Ejemplos

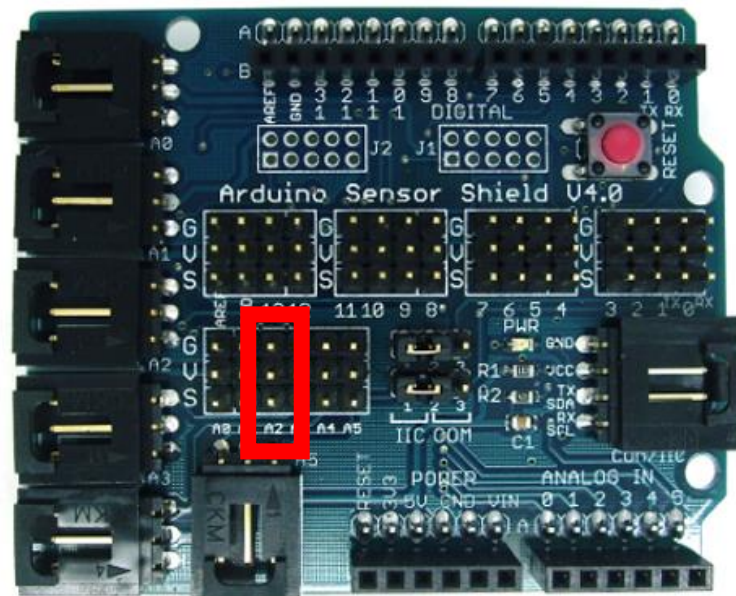
### Detección de luz

En este ejemplo vamos a utilizar un sensor de luz para monitorizar el nivel de luz de una habitación, en el caso de que el nivel de luz sea bajo encenderemos un LED.

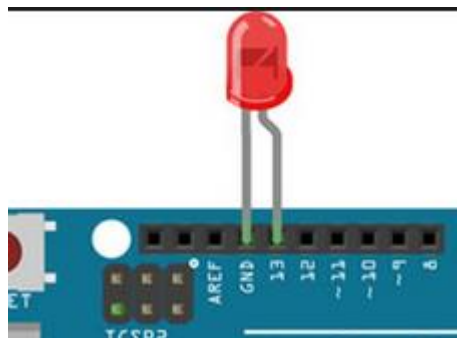
#### Construcción del circuito:

##### 1. Montaje

- Opción 1:** protoboard + Arduino: conectamos el sensor de luz a: **GND (negro), 5V (Rojo) y A2 (azul).**
- Opción 2:** sensor Shield. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores.



- Conectamos directamente sobre la placa un LED al Pin 13 y GND. El pin 13 incluye una resistencia, por lo que no hace falta colocar una. Realmente el pin 13 ya incluye un pequeño LED (justo debajo), pero conectamos nuestro LED para darle mayor visibilidad.



#### Programación de Arduino:

- Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos el pin digital 13 como salida.

2. En cada interacción del método `loop()` obtenemos el valor del sensor de luz leyendo directamente la entrada analógica.
3. Evaluamos el valor retornado por el sensor. En caso de que este valor sea bajo, encendemos el led.
4. Comprobamos el funcionamiento del programa: podemos tapar la fotorresistencia con la mano para obtener valores de luz bajos.

```
void setup(){
  //1.- No hace falta declarar las lecturas analógicas
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop(){
  // Leer lectura analógica
  int lightValue = analogRead(A2);
  //2.-
  Serial.println("Valor: "+String(lightValue));

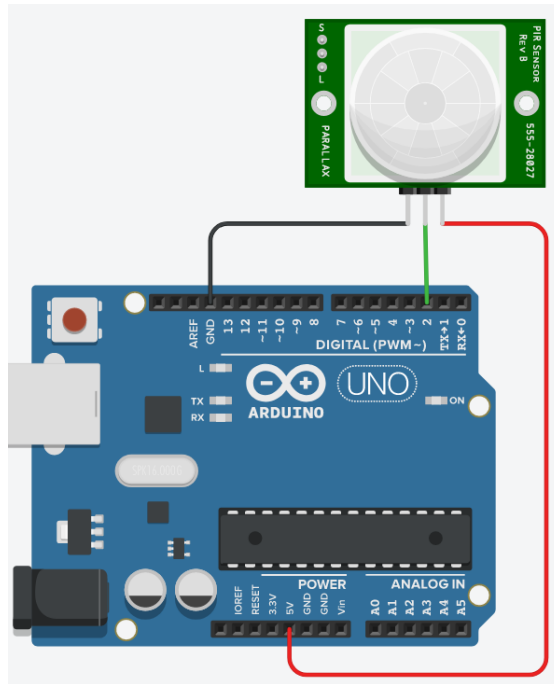
  //3.-
  if (lightValue < 200){
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

## Detección de movimiento

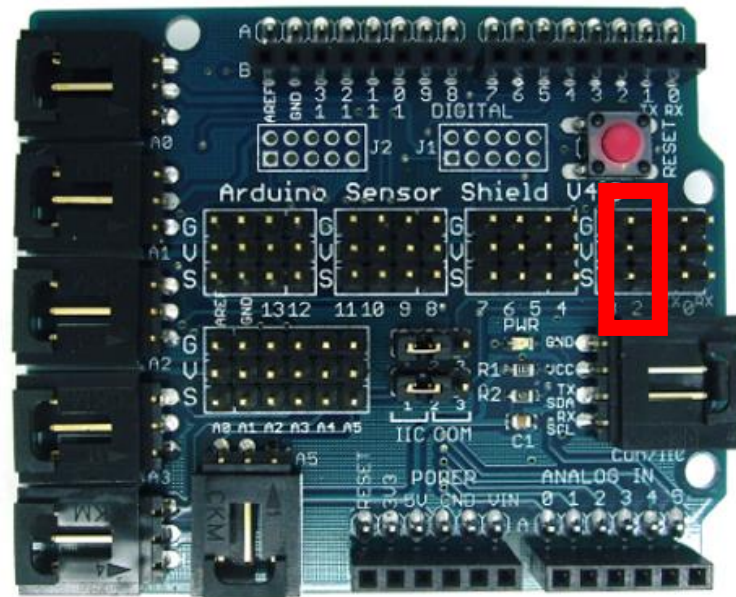
En este ejemplo utilizamos el sensor de movimiento de infrarrojos para detectar movimiento cerca del sensor.

### Construcción del circuito:

1. Montaje
  - a. **Opción 1:** conectamos el sensor PIR a: **GND, pin 2 y 5V**



- b. **Opción 2:** sensor Shield. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores:



- Conectamos directamente sobre la placa un LED al Pin 13 y GND. El pin 13 incluye una resistencia, por lo que no hace falta colocar una.

#### Programación de Arduino:

- Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos el pin digital 2 como entrada e inicializamos el puerto Serial para imprimir por consola.
- En cada iteración del método `loop()` obtenemos el valor del sensor leyendo la entrada digital.

3. Evaluamos el valor retornado y hacemos una pequeña pausa entre mediciones. **Tarda en responder por consola y hay que ajustar la sensibilidad a la deseada, además de enfocarlo al sitio correcto, pues los reflejos o pequeños movimientos de la mesa harán que detecte movimiento.**
4. Código de ejemplo:

```
int pinSensor = 2;

void setup() {
  //1.-
  Serial.begin(9600);
  pinMode(pinSensor, INPUT);
}

void loop() {
  //2.-
  int pirValue = digitalRead(pinSensor);
  Serial.println("Valor: "+String(pirValue));

  //3.-
  if(pirValue == HIGH) {
    Serial.println("Detectado movimiento");
  } else {
    Serial.println("- - - -");
  }

  delay(50);
}
```

### Otros sensores digitales: sensor de vibración, infrarrojos, ...

El funcionamiento del sensor de vibración es prácticamente igual que el del sensor PIR. Sin embargo, **las patillas del sensor cambian, cuidado.**

### Control de distancia mínima

En este ejemplo utilizamos el sensor de ultrasonidos para detectar la distancia a la que colocamos un objeto.

#### Construcción del circuito:

1. Conectamos el sensor de ultrasonido a: **V5, Pin digital 9 (Trig), Pin digital 8 (Echo) y GND**

#### Programación de Arduino:

1. Creamos las variables globales que utilizaremos a lo largo de todo el programa: distance, responseTime, pinTrig, y pinEcho.
2. Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos los 2 pines, uno como salida y otro como entrada, e inicializamos el puerto Serial para imprimir por consola.
3. En cada interacción del método `loop()` colocamos el pin Trig a LOW y realizamos una breve pausa.
4. Emitimos la señal HIGH por el pin Trig y realizamos otra breve pausa.

5. Ya podemos leer el resultado del pin Echo. Para ello, utilizamos el método especial `pulseIn()`. Este método nos da los microsegundos que tarda el PIN en cambiar su estado HIGH / LOW.
  - a. <https://www.arduino.cc/en/Reference/pulseIn>
6. Una vez tenemos los microsegundos que tarda el ultrasonido en impactar contra el objeto y volver, podemos calcular la distancia.
  - a. La velocidad del sonido es de 343,2 m/s, que si lo transformamos en cm/s sería:  $343,2 \text{ m/s} * 100\text{cm/m} * 1\text{s}/1000000 \text{ ms} = 0,03432 \text{ cm/ms}$ . No obstante, como solo nos interesa lo que tarda en llegar, podemos dividir el tiempo de respuesta o la velocidad entre 2, pues sino estaríamos calculando lo que tarda en ir y volver. Esto nos da 0,01716 exactamente.
7. Finalmente añadimos una pequeña espera (quizá sea mejor probar con esperas más grandes).
8. Código de ejemplo:

```
//1.-
long distance;
long responseTime;

int pinTrig = 9;
int pinEcho = 8;

void setup(){
  //2.-
  Serial.begin(9600);
  pinMode(pinTrig, OUTPUT); /* Trig envía el pulso ultrasónico
*/
  pinMode(pinEcho, INPUT); /* Echo capta el rebote del pulso
ultrasónico*/
}

void loop(){
  //3.-
  digitalWrite(pinTrig, LOW); /* Por seguridad volvemos a poner
el Trig a LOW*/
  delayMicroseconds(5);

  //4.-
  digitalWrite(pinTrig, HIGH); /* Emitimos el pulso ultrasónico
*/
  delayMicroseconds(10);

  //5.-
  responseTime = pulseIn(pinEcho, HIGH); /* Medimos la longitud
del
pulso entrante Cuanto tiempo tarda la entrada en pasar de HIGH
a LOW
  retorna microsegundos */
  Serial.println("Tiempo "+ String(responseTime)+"
microsegundos");

  //6.-
  distance = int(0.01716*responseTime); /* Calcular la distancia
conociendo la velocidad */

  Serial.println("Distancia "+ String(distance)+"cm");
```

```
//7.-  
delay(100);  
}
```

## Pantalla de segmentos

En este ejemplo vamos a utilizar una pantalla de segmentos de cuatro dígitos para imprimir un número por pantalla. Vamos a mostrar el valor de un entero en la pantalla.

### Construcción del circuito:

1. Partiendo del circuito anterior conectamos la pantalla a: **CLK (~3), DIG (4), VCC (5V) y GND (GND).**

### Programación de Arduino:

En primer lugar, para poder manejar la pantalla, necesitamos descargar e importar una nueva librería: **DigitalTube TM1637:**  
<https://github.com/reedstudio/libraries/tree/master/DigitalTube>

Para agregar la librería, descargamos el archivo comprimido en zip y en el IDE pulsamos sobre **Programa -> Include Library -> Add. Zip Library.**

1. Al incluir la librería veremos que se genera un **#include** al principio de nuestro programa de la librería TM137.
2. Creamos las variables globales para los pines CLK y DIO. Declaramos un nuevo objeto de tipo TM1637 al cual le indicamos los pines a utilizar. La librería se encargará de su configuración.
3. En el método `setup()` inicializamos la pantalla y le establecemos el brillo. Posteriormente hay que realizar una pausa para darle tiempo a que se inicie. El método `init()` de TM1637 se encarga de iniciar todo lo necesario, incluido las salidas digitales. Con el método `set()` podremos establecer el brillo que deseemos entre 0 como el más oscuro y 7 el más claro.
4. En el interior del método `loop()`, una vez obtenida la distancia, la mostramos por pantalla. Para ello, usamos el método `display(índice de dígito, valor[0-9])`. Hay que tener en cuenta que debemos dividir la distancia en dígitos: unidades, decenas, centenas, millares. Para esto, tendremos que calcular en que posición va cada dígito antes de imprimirlo por la pantalla. Para ello, «jugaremos» con el tipo entero para quedarnos solo con la parte que nos importa del número y que el multiplicarlo de nuevo, nos quede solo esa unidad: 1,245 en un entero es 1, por 1.000 es 1.000, no 1245, y así sucesivamente.
5. `ClearDisplay()`: el método **clearDisplay** sirve para limpiar los datos enviados anteriormente a la pantalla de segmentos y así poder imprimir menos de 4 números en la pantalla, dejando alguno o varios sin utilizar, evitando que salga «basura» o los números previos en los que deseamos dejar en blanco.
6. Código de ejemplo:

```
//1.-  
#include <TM1637.h>
```

```

long distance;

//2.-
int pinClk = 3;
int pinDio = 4;

TM1637 screen(pinClk, pinDio);

void setup(){
    Serial.begin(9600);

    //3.-
    screen.init();
    screen.set(BRIGHT_TYPICAL);
    //BRIGHT_TYPICAL = 2; BRIGHT_DARKEST = 0; BRIGHTEST = 7;

    delay(1500); // Esperamos a que se inicie la pantalla
}

void loop(){
    distance = 1245;

    //4.-
    int digit0 = distance/1000;
    Serial.println("Digit0 "+ String(digit0));

    int digit1 = (distance - digit0*1000)/100;
    Serial.println("Digit1 "+ String(digit1));

    int digit2 = (distance - (digit0*1000 + digit1*100))/10;
    Serial.println("Digit2 "+ String(digit2));

    int digit3 = distance - (digit0*1000 + digit1*100 +
digit2*10);
    Serial.println("Digit3 "+ String(digit3));

    screen.display(0, digit0);
    screen.display(1, digit1);
    screen.display(2, digit2);
    screen.display(3, digit3);

    delay(2000);

    //5.-
    screen.clearDisplay();
    screen.display(1, digit0);
    screen.display(2, digit1);

    delay(2000);
}

```

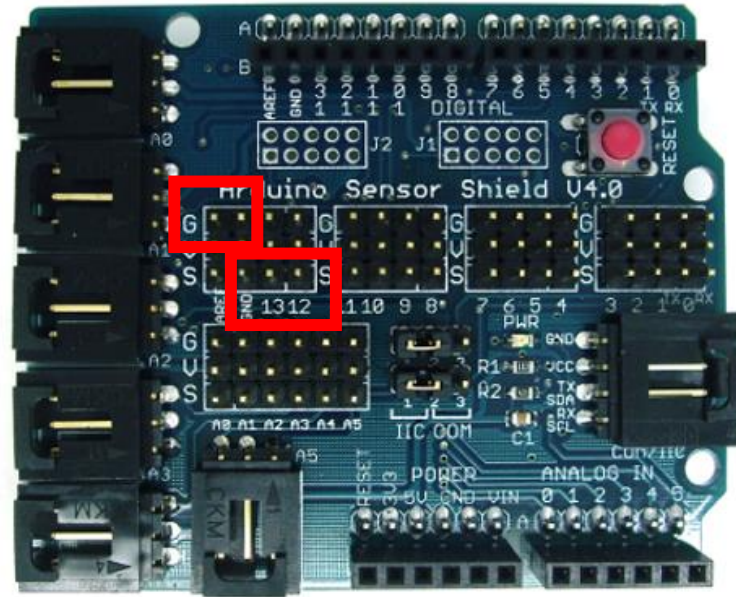
## LED RGB

En este ejemplo vamos a explorar el funcionamiento de un LED RGB.

### Construcción del circuito:



1. Conectamos las patillas de LED utilizando el Shield de sensores a: **R al S9, G al S10 y B al S11, - a cualquier G.**



#### Programación de Arduino:

En el siguiente ejemplo controlamos los tres colores del LED con salidas digitales. Podemos colocar y establecer el color deseado en el LED RGB según la salida que pongamos a HIGH (enciende) o LOW (apaga).

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  digitalWrite(redPin, HIGH);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, LOW);
  delay(500);

  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, HIGH);
  digitalWrite(bluePin, LOW);
  delay(500);

  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, HIGH);
  delay(500);
}
```

En este otro ejemplo controlamos el LED con salidas digitales PWM, que nos permiten enviar señales analógicas y podemos establecer cada color de salida a un valor entre [ 0 – 255].



Esto nos proporciona el poder crear una gama más amplia de colores según las mezclas que realicemos.

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

int scaleColour = 0;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  analogWrite(redPin, 255 - scaleColour);
  analogWrite(greenPin, scaleColour);
  analogWrite(bluePin, scaleColour);
  delay(15);

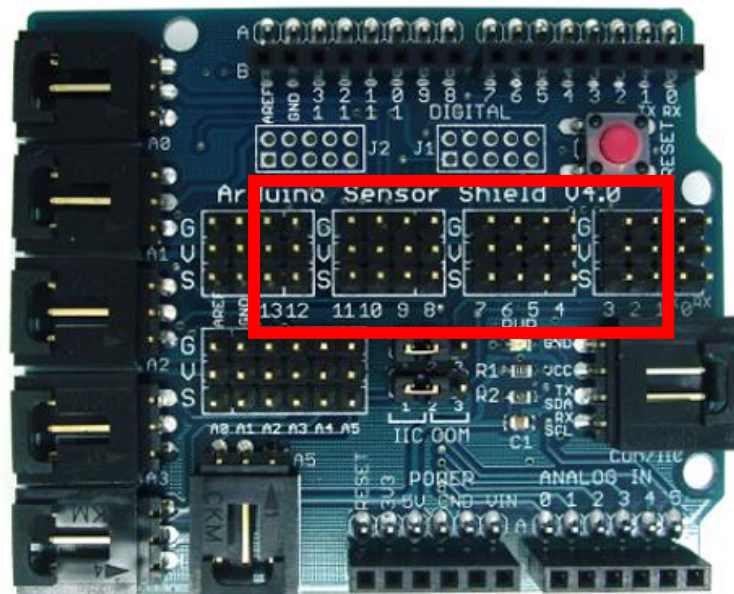
  scaleColour++;
  if (scaleColour > 255)
    scaleColour = 0;
}
```

## Servomotor 180°

En este ejemplo vamos a explorar el funcionamiento de un miniservomotor.

### Construcción del circuito:

1. Conectamos el servomotor a: **V5 (rojo), Pin digital 8 (naranja) y GND (negro/marrón).**
2. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores:



### Programación de Arduino:

1. En primer lugar, importamos la librería que Arduino nos ofrece para controlar el servomotor. Después de importarla, observaremos que se ha incluido un nuevo `#include` en nuestro programa de la librería `Servo`.
2. Creamos la variable global de tipo `Servo`.
3. Inicializamos el puerto `Serial` para imprimir por consola y el puerto para registrar la salida en el método `setup()` y asociamos el pin 8 al servo. No hace falta especificar que el pin 8 será una salida digital, ya que se encarga la librería de hacerlo por nosotros.
4. En el método `loop()` movemos el servo a diferentes grados utilizando el método `write(grados)`. Después de cada movimiento realizamos una pausa.
5. Código de ejemplo:

```
//1.-
#include <Servo.h>

//2.-
Servo servol;

void setup() {
  //3.-
  Serial.begin(9600);
  servol.attach(8);
}

void loop() {
  //4.-
  servol.write(180);
  Serial.println("180");
  delay(3000);
  servol.write(90);
  Serial.println("90");
  delay(3000);
  servol.write(0);
  Serial.println("0");
  delay(3000);
}
```

**No desmontéis el circuito**, pues se utilizará en el siguiente ejemplo.

## Teclado

En este ejemplo vamos a utilizar un teclado matricial para introducir el número de grados que debe girar un mini servomotor.

### Construcción del circuito:

1. Partiendo del circuito anterior, el del servomotor, conectamos el teclado a: **2 - 5 y A0 - A3** de forma ordenada empezando por el conector de la izquierda.

### Programación de Arduino:

Con el nuevo IDE; parece no ser necesaria incluirla. Si se trabaja con el IDE antiguo, tal vez hubiera que bajar, en algunos casos, la siguiente librería para poder manejar el teclado, necesitamos importar una nueva librería, la **KeyPad**:

[https://cdn.shopify.com/s/files/1/0557/2945/files/Keypad\\_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262](https://cdn.shopify.com/s/files/1/0557/2945/files/Keypad_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262)

Para agregar la librería descargamos el Zip y pulsamos sobre **Programa -> Include Library**  
-> **Add. Zip Library**

1. Continuamos modificando el programa anterior. Lo primero es incluir la librería del teclado matricial. Aquí veremos que se genera un **#include** al principio de nuestro programa de la librería Keypad (la K es mayúscula).
2. Creamos las variables globales para el número de filas y de columnas, y la matriz con todas las teclas que tiene nuestro teclado.
3. Creamos dos arrays indicando los pines que vamos a utilizar para las filas y las columnas.
4. Creamos un objeto de tipo Keypad, que recibe un KeyMap con las teclas, los pines de las filas, los pines de las columnas y el número de filas y columnas.
5. Finalmente, añadimos un buffer de lectura en el que iremos almacenando todas las teclas que pulsa el usuario.
6. En el método `loop()` pedimos que nos devuelva la tecla obtenida. Si el usuario no pulsa nada en cada iteración del bucle `loop` devolverá vacío (`"\0"`). Obtenemos la tecla pulsada con el método `getKey()` y comprobamos que no es vacía antes de continuar con el procesamiento. En caso de que no sea vacía la mostramos por consola.
7. Vamos a definir que cuando el usuario pulse la tecla '#' ya ha dejado de escribir. Cualquier tecla distinta a '#' la agregamos al buffer. Cuando el usuario pulsa '#' pasamos todo el contenido del buffer a un entero y lo vaciamos.
8. Utilizamos ese «valorNumerico» para indicarle al servomotor la posición que debe tomar.

```
//1.-
#include <Keypad.h>
#include <Servo.h>

Servo servol;

//2.-
const byte nfilas = 4;
const byte ncolumnas = 4;
char teclas[nfilas][ncolumnas] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

//3.-
byte pfilas[nfilas] = {2,3,4,5}; // Filas
byte pcolumnas[ncolumnas] = {A0,A1,A2,A3}; //Columnas

//4.-
Keypad teclado = Keypad(makeKeymap(teclas), pfilas, pcolumnas,
nfilas, ncolumnas);

//5.-
String bufferLectura = "";
```

```

void setup(){
    Serial.begin (9600);
    servol.attach(8);
    Serial.println("Setup()");
}

void loop(){
    //6.-
    char tecla = teclado.getKey();
    if (tecla != '\0'){
        Serial.println("tecla pulsada: "+String(tecla));

        if (isdigit(tecla)){
            bufferLectura = bufferLectura + tecla;
            Serial.println("buffer: "+bufferLectura);

            //7.-
        } else if (tecla == '#'){
            // es # transformamos a entero
            int valorNumerico = bufferLectura.toInt();
            if (valorNumerico > 180){
                valorNumerico = 180;
            }
            Serial.println("Aceptado: "+String(valorNumerico));
            //8.-
            servol.write(valorNumerico);
            bufferLectura = ""; // reinicio
        } else {
            Serial.println(";Letras no!");
        }
    }
}

```

# Introducción a Arduino



[https://commons.wikimedia.org/wiki/File:Arduino\\_Uno\\_-\\_R3.jpg](https://commons.wikimedia.org/wiki/File:Arduino_Uno_-_R3.jpg)

Práctica 1 – Teoría (v1.6.2 septiembre 2022)

## Software para robots

Cristian González García

[gonzalezcristian@uniovi.es](mailto:gonzalezcristian@uniovi.es)

## Índice

Introducción .....	2
Instalación y configuración .....	2
Componentes eléctricos .....	9
Resistencia .....	9
Cables .....	10
Sensores .....	10
Pulsador/Botón .....	10
Potenciómetro .....	11
Actuadores .....	12
Diodo LED .....	12
Altavoz/Zumbador/Speaker .....	12
LED RGB .....	16
Ejemplos de circuitos .....	17
Circuito básico .....	17
Circuito en serie .....	20
Circuito en paralelo .....	21
Circuitos electrónicos y programación .....	21
Salidas digitales - Leds que parpadean .....	22
Entradas digitales - Controlando los leds con un botón .....	24
Lectura de entrada analógica .....	30
Emulador de Arduino: Tinkercad - Circuits .....	32
Arduino .....	37
Generar números aleatorios .....	37
Estructuras de datos .....	40
Arrays .....	40
Errores comunes .....	41
avrdude : stk500_getsync() attempt 10 of 10: not in sync: resp=0x00 .....	41

## Introducción

El objetivo de esta práctica es adquirir los conocimientos básicos necesarios para comenzar a manejar la plataforma Arduino. Para realizar esta práctica cada alumno dispondrá de un kit Arduino.

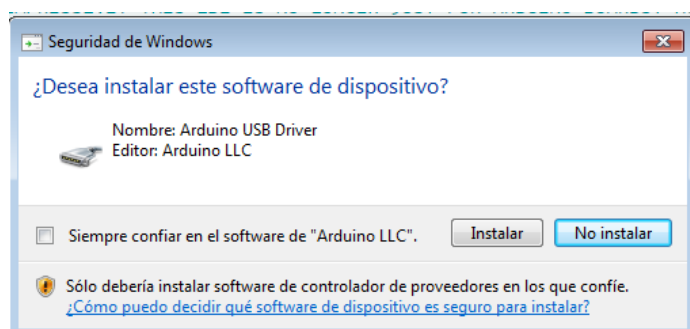
Se recomienda también el uso del entorno de emulación <https://www.tinkercad.com> para realizar diferentes pruebas o probar los ejercicios en casa si no se dispone de un Arduino y así en clase solamente tener que probar lo realizado en casa y grabar el vídeo. Funciona con cuenta Autodesk. Hay que hacerse una cuenta Autodesk, en caso de que no se tenga. Después, seleccionar «Circuits». En este apartado se tienen diferentes tutoriales y el editor gráfico y simulador de la placa Arduino. También se puede introducir el código en este simulador y así probarlo en este simulador como funciona antes de subirlo al Arduino. Su uso básico se explica en: Emulador de Arduino: Tinkercad - Circuits.

## Instalación y configuración

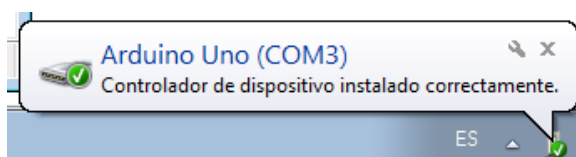
En los ordenadores del laboratorio, los drivers de Arduino y su IDE ya deberían de estar. Para comprobar si todo está correctamente funcionando, enchufad el Arduino al puerto USB del ordenador y tratad de subir a él un programa vacío desde la placa.

En caso de necesitar instalarlo, hay que seguir los siguientes pasos:

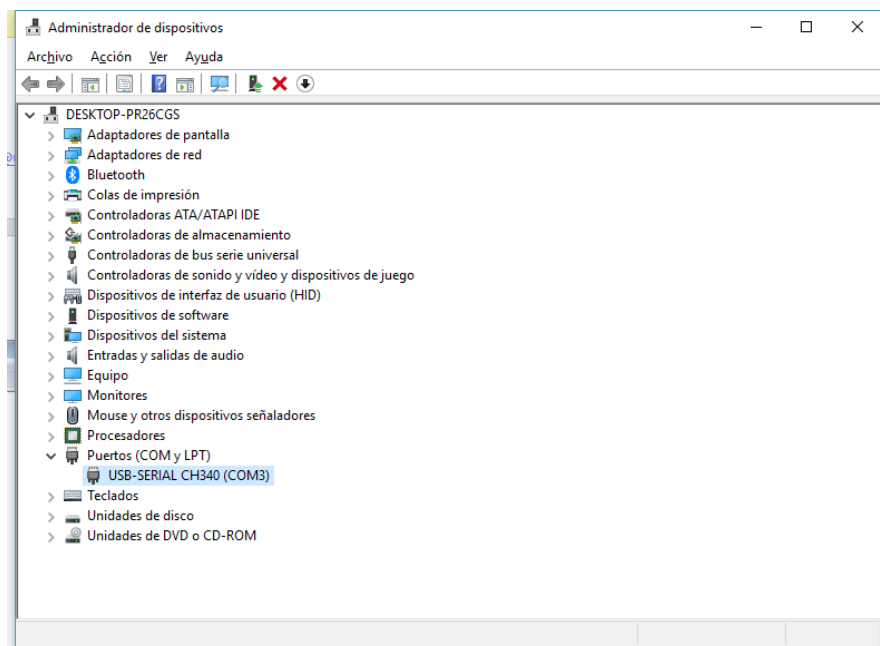
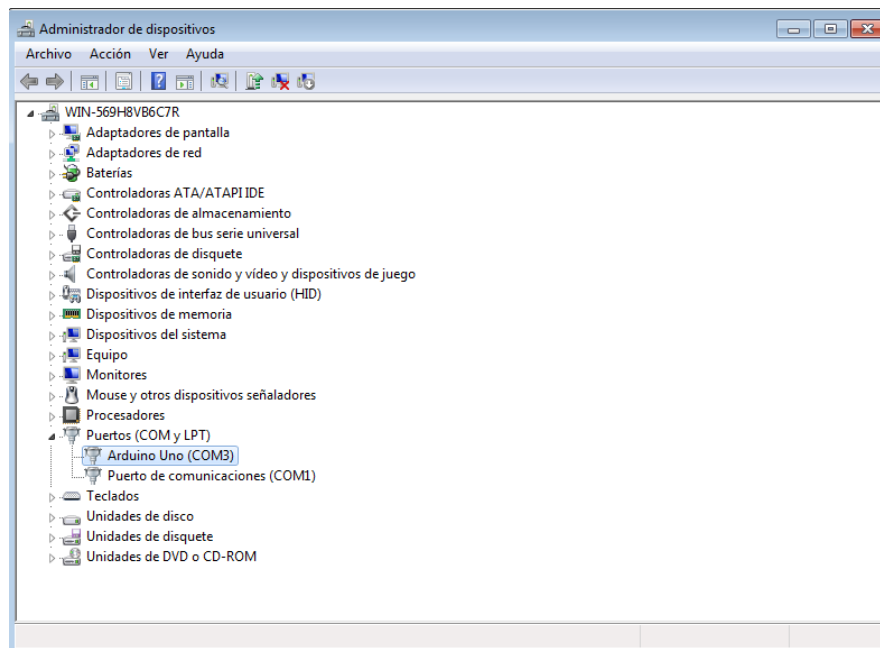
1. Descargar y ejecutar el Arduino IDE de su página oficial, siempre teniendo en cuenta nuestro sistema operativo (Windows 7, Windows 10, Linux, ...): <https://www.arduino.cc/en/Main/Software>. La última versión es la 1.8.19, o la 2 (liberada el 14/09/2022).
2. Durante el proceso, puede que nos pregunte si queremos instalar varios drivers. Así que los aceptaremos. Si nos salta una pantalla del firewall en Windows, le damos permiso de conexión en redes privadas para que baje actualizaciones y librerías. En caso de que no detectase e instalase estos drivers, esto mismo suele ocurrir si se conecta la placa Arduino y no se están instalados los drivers.



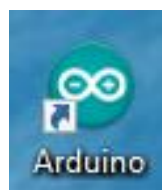
3. Conectamos la placa Arduino al equipo por medio del puerto USB, el cuál debería reconocerlo correctamente.



4. Abrimos el **administrador de dispositivos** y vamos a la categoría **Puertos**. Debería aparecernos el dispositivo **Arduino Uno** con uno de los puertos COM asignados, o bien el nombre **CH340** junto a su puerto COM.



5. Ejecutamos el software de Arduino que instalamos previamente.



*Ilustración 1 Icono de Arduino*



6. Desde la opción «herramientas» comprobamos que la placa seleccionada es **Arduino Uno**, y que el **puerto** con el que estamos trabajando es realmente el puerto del Arduino (en este caso, COM3).
- a. **En caso de que no funcione** un determinado puerto, como suele ser habitual con los delanteros tras estar avanzado el semestre, **probar con los puertos traseros**. Esto suele suceder cuando se estropea el controlador del USB de tanto usarlo. La **solución** a este problema es **usar otro USB o desinstalar el controlador** que da problemas y que Windows lo reinstale de nuevo.

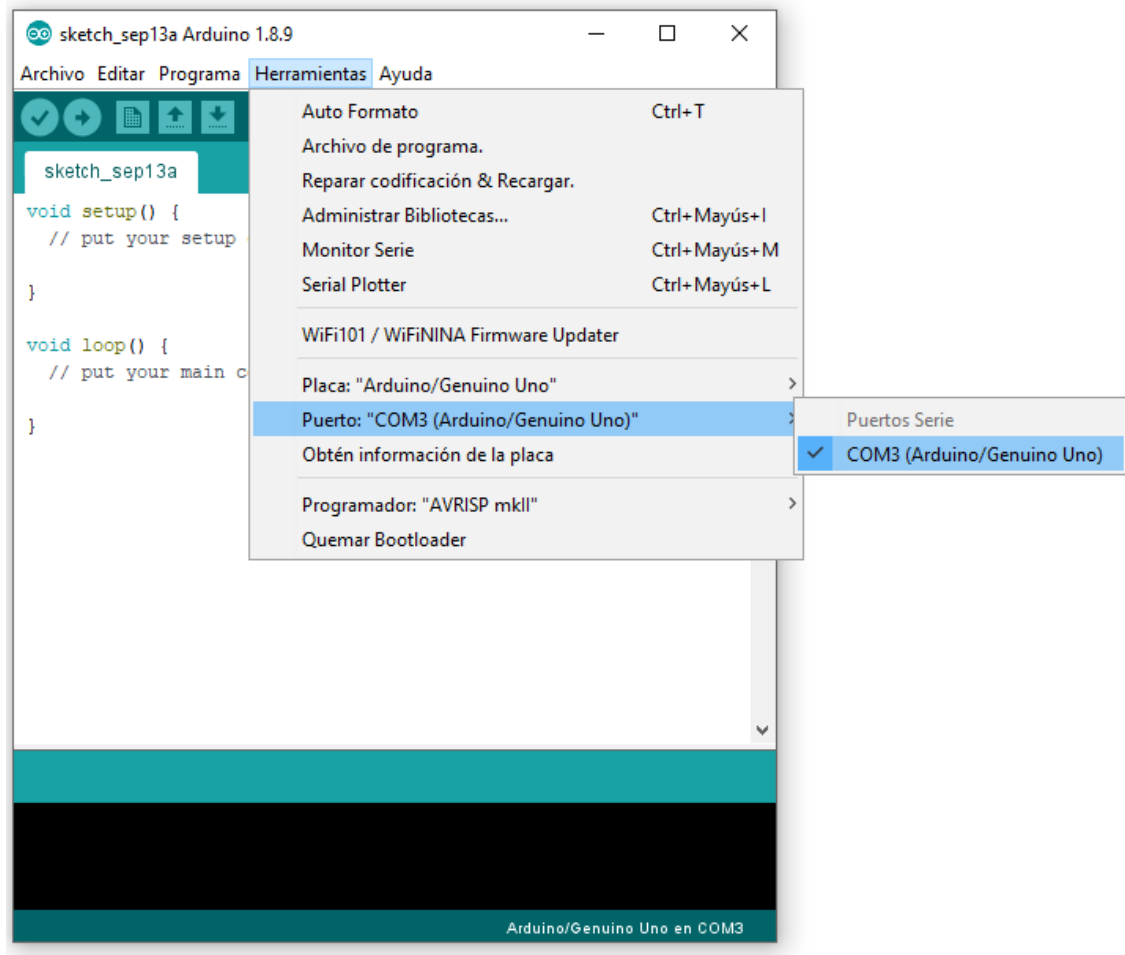


Ilustración 2 Selección de puerto COM

7. **Tipo de placa:** estamos utilizando los modelos **Arduino Uno** (Ilustración 3), **Arduino BT** (Ilustración 4), o **BQ Zum** (Ilustración 3). Es importante seleccionar el modelo que se utiliza correctamente.



*Ilustración 3 Placa Arduino Uno*

\* La placa **Arduino BT** tiene un interruptor de encendido en el lateral.



*Ilustración 4 Placa Arduino BT*

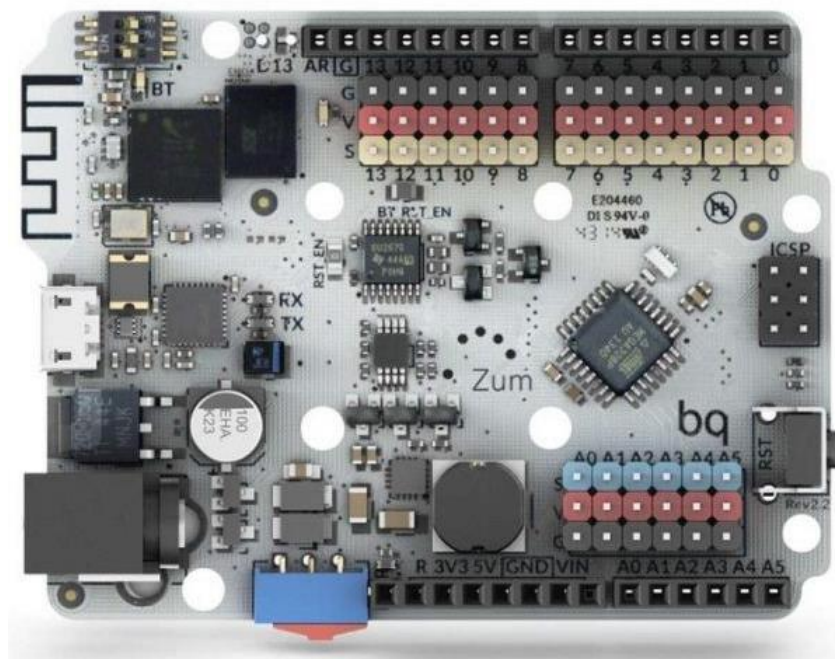


Ilustración 5 Placa BQ Zum

8. Seleccionamos el tipo de placa en el IDE. Si fuera la BQ Zum seleccionaremos Arduino Uno.

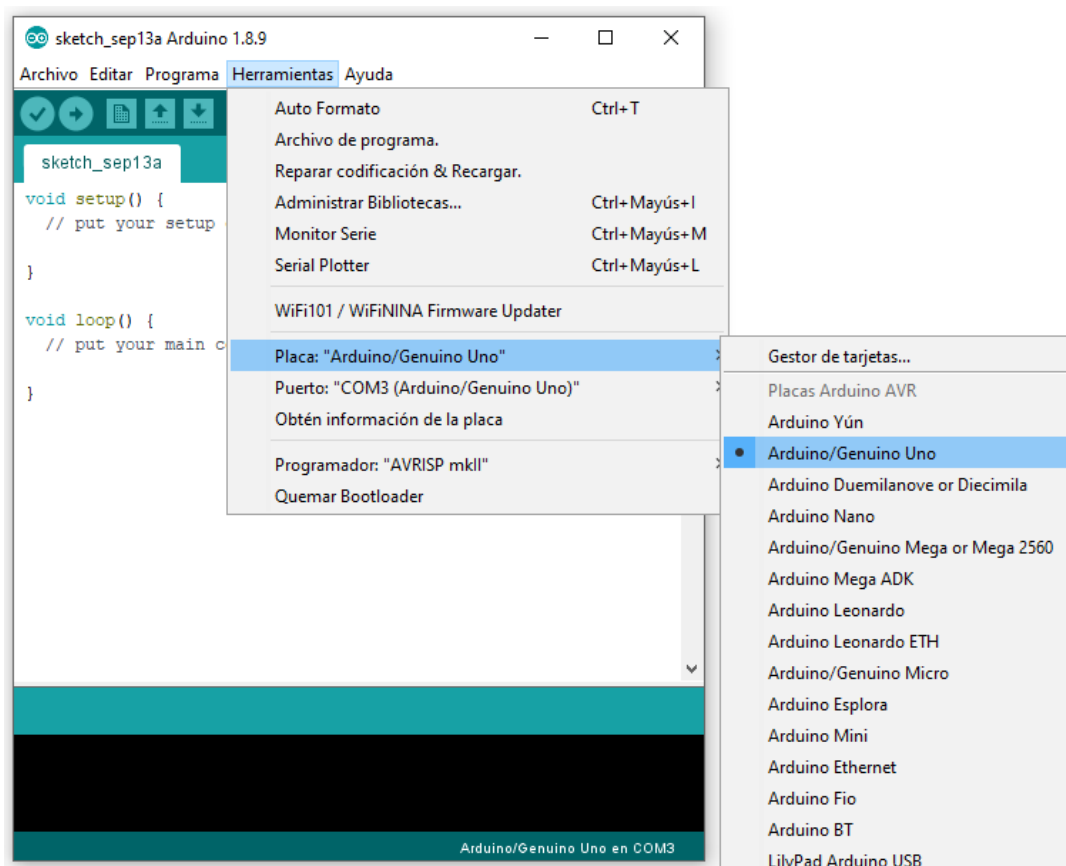
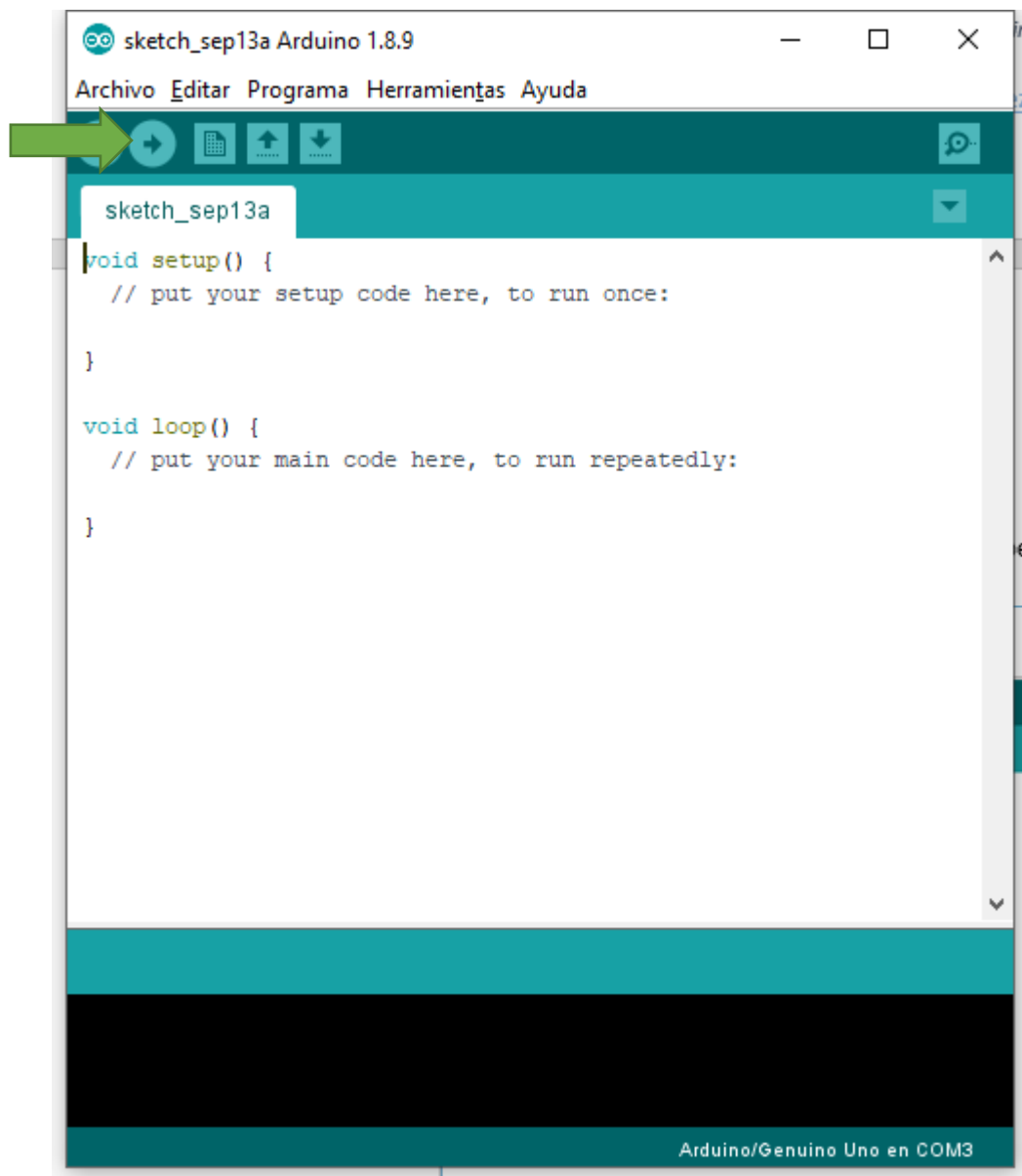


Ilustración 6 Selección de microcontrolador Arduino

9. Si todo es correcto, al pulsar en subir el programa la acción debería realizarse con éxito.



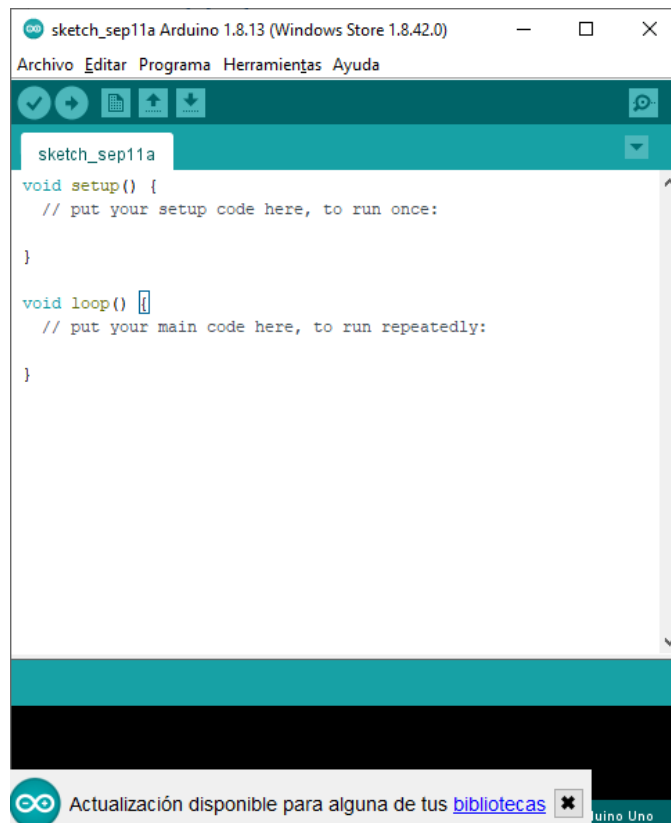
*Ilustración 7 Subir programa*

**NOTA:**

En ocasiones estos valores (COM y Placa) se desconfiguran solos provocando errores en la compilación.

En caso de errores de compilación tendréis que revisar que la configuración sea correcta.

A veces, puede salirnos un popup en la parte inferior del IDE para indicarnos que hay nuevas versiones de algunas bibliotecas (Ilustración 8).



*Ilustración 8 Nuevas bibliotecas*

Si pinchamos en bibliotecas, o bien, vamos de forma manual a **Herramientas -> Ayuda** (Ilustración 9), podremos buscar nuevas bibliotecas, y actualizar las que ya tenemos. **Se recomienda trabajar con las últimas versiones estables siempre.**

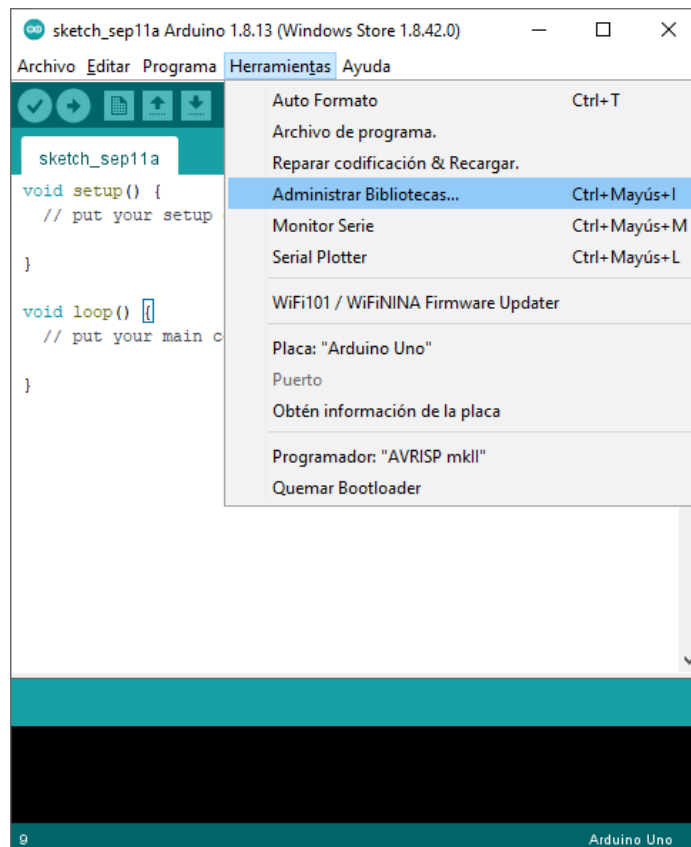


Ilustración 9 Administrar bibliotecas

## Componentes eléctricos

### Resistencia



**Resistencia:** agrega una oposición a los electrones. No tiene polaridad. Podemos conocer su valor gracias al código de colores.

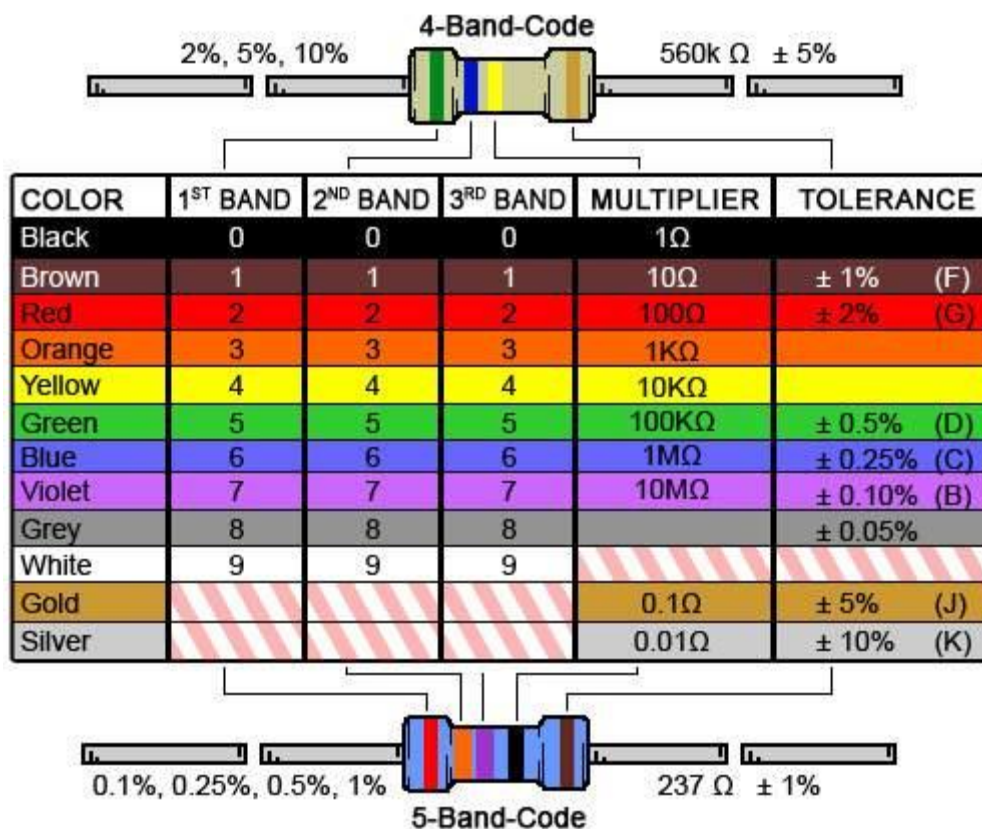
Para hacer que el LED rojo reciba la potencia adecuada hay que utilizar una resistencia de  $220\Omega$ . Salvo que se utilice el pin 13 del Arduino, pues este pin ya incluye una resistencia.



La segunda resistencia de la izquierda es de  $10k\Omega$ .

Calculadora de resistencias:  
<https://www.digikey.es/es/resources/conversion-calculators/conversion-calculator-resistor-color-code-5-band>

A continuación, se muestra una imagen que explica como leer el valor de una resistencia según sus colores.



1

## Cables

Siempre utilizaremos el cable rojo para el positivo (ánodo) y el negro para el GND (también llamado negativo, -, cátodo, o tierra). No obstante, se llama así por el tema positivo-negativo, pero no tiene polaridad negativa.

## Sensores

### Pulsador/Botón



**Pulsador:** cuando está abierto rompe el circuito eléctrico, cuando está cerrado activa el circuito. Sirve simplemente para cortar o no la corriente y no tiene ninguna resistencia.

Por este motivo, siempre necesita una resistencia el circuito que lleve un interruptor, salvo que los elementos utilizados cuenten con una o hagan suficiente resistencia al paso de la corriente. Esto es debido a que el pulsador actúa, cuando se pulsa, como si fuera un cable. Luego, conecta positivo y negativo directamente, lo que provoca un cortocircuito, salvo que haya una resistencia. El cortocircuito podría estropear el Arduino y otros elementos del circuito. Así pues, si se pone un pulsador con un elemento que no tenga resistencia apenas, debería de usarse para una salida de 5 V (si es de 3,3 V serviría una más pequeña), al menos, una resistencia de 250  $\Omega$ ,

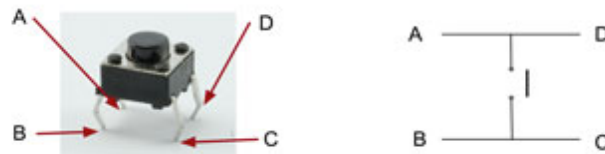
<sup>1</sup> <https://repository.polibatam.ac.id/uploads/215207-20170811070840.pdf>



pues el Arduino trabaja en cada pin con 0,02 A. Hay gente que usa resistencias de 470  $\Omega$ . Ambas son válidas, pero la primera está al límite.

Este modelo de pulsador tiene cuatro patillas, por lo que podría ser utilizado para situaciones «complejas».

En nuestro caso, vamos a utilizarlo como un pulsador simple, haciendo uso únicamente de dos patillas A - B o D - C.

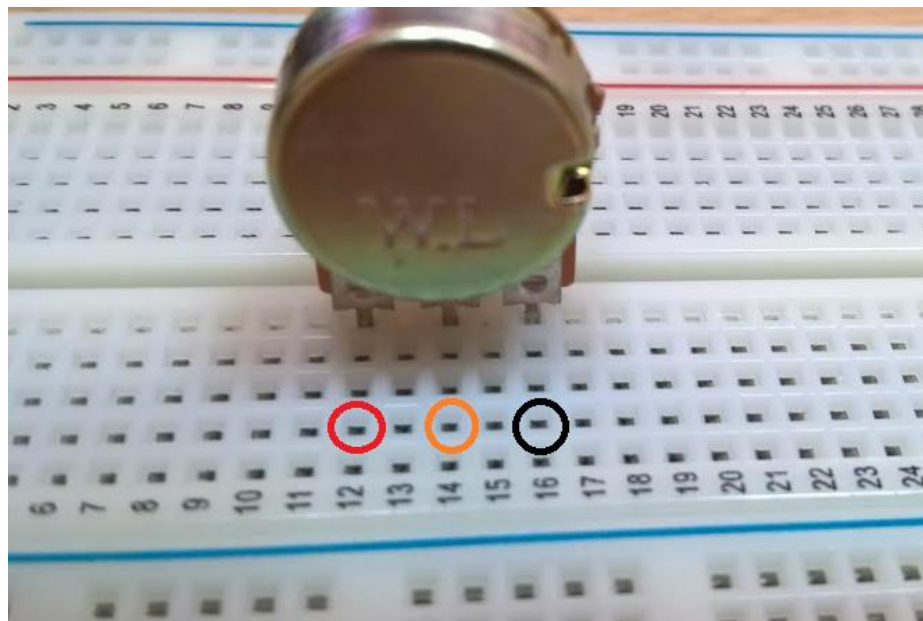


## Potenciómetro



**Potenciómetro de 10K  $\Omega$ :** es una resistencia variable que sirve para variar la intensidad de la resistencia girando la rueda.

**Nota:** si se conectan del revés el positivo y el negativo al **potenciómetro**, entonces, el potenciómetro, **funcionaría del revés**.



*Ilustración 10 Conexión de potenciómetro en ProtoBoard. Rojo a 5V, Naranja a pin A0 y Negro a GND. Si + y - se conectan del revés, funcionaría el giro del revés.*



## Actuadores

### Diodo LED



**Diodo LED:** es un componente electrónico que solo permite pasar la corriente en una dirección. Siguiendo la dirección del positivo al negativo (la parte ancha del triángulo). La pata positiva del LED es la más larga.

En este caso para hacer que el Led rojo reciba la potencia adecuada utilizaremos una resistencia de  $220\Omega$  (si queremos una intensidad de 20mA para que tenga un brillo fuerte, o una resistencia mayor para que brille menos). La resistencia puede ir tanto en el lado positivo como negativo. Salvo que se utilice el pin 13 del Arduino, pues este pin ya incluye una resistencia. Si al pin 13 le conectáramos una resistencia entre este y el LED, entonces el LED no funcionaría, o apenas alumbraría.

Calculadora de resistencias para LEDs: [http://gzalo.com/resistencias\\_led/](http://gzalo.com/resistencias_led/)

Color	Caída de voltaje (Aproximada)	Resistencia (Ohmios) con 5v aprox. y 0,015A
Infrarrojo	1,4 V	270
Rojo (alto)	1,8 V a 2,2 V	220
Naranja	2,1 V a 2,2 V	200
Amarillo	2,1 V a 2,4 V	200
Verde	2 V a 3,5 V	150
Azul	3,5 V a 3,8 V	100
Violeta	3,6 V	100
Blanco	3,8 V	100

**NOTA:** si se enchufa a un pin digital del Arduino y el LED no alumbrá o alumbrá muy poco y el código fuente es correcto, hay que revisar a ver si se tiene puesto el pin que se usa para el LED marcado como salida (OUTPUT). Si no es así, el LED no funcionará o iluminará muy poco.

### Altavoz/Zumbador/Speaker/Piezo



**Zumbador:** capaz de producir varios sonidos. Se suele utilizar en alarmas y sonidos característicos del sistema.

Tiene una patilla positiva **VCC**, una negativa **GND** y una patilla **I/O** que se conecta a un pin PWM, como el ~11, capaz de emular escrituras analógicas. Esto nos permitirá especificar los tonos. Dependiendo de la señal que le enviemos por el pin emitirá un tono u otro.

Hay dos formas de usarlo. La primera es con la función «analogWrite» (<https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>). Esta función se puede utilizar a la vez en varios pines. No obstante, siempre emitirá el mismo tono y solo admite valores entre 0 y 255. Luego, es un altavoz un poco limitado y con un tono no muy agradable.

```
int pinbuzzer = 11;

int song[] = {261, 349, 392, 440, 392, 330, -10, 261, 349, 392,
440, 392, -10, -10, 261, 349, 392, 440, 392, 330, -10, 330, 349, 330,
261, 261};

void setup()
{
  Serial.begin(9600);
  pinMode(pinbuzzer, OUTPUT);
}

void loop()
{
  for (int i = 0; i < sizeof(song)/sizeof(int); i++)
  {
    analogWrite(pinbuzzer, song[i]);
    delay(500);
  }
}
```

La segunda forma es utilizando la API de Arduino que nos provee de dos métodos específicos para manejar el zumbador:

- Tone: <https://www.arduino.cc/en/pmwiki.php?n=Reference/Tone>
- NoTone: <https://www.arduino.cc/reference/en/language/functions/advanced-io/notone/>

La función «tone» recibe como parámetro el pin del altavoz y la frecuencia de la nota. Como tercer parámetro opcional recibe la duración. La función «notone» recibe un pin y lo que hace es silenciarlo.

Dependiendo del pin y de la placa (Uno o Mega), tiene más rango de frecuencias y más pines disponibles. No obstante, la función «tone» solo funciona en un pin a la vez. En el caso de usar varios pines, solo funcionará en el primer pin que se llame con dicha función.

Cada frecuencia da una «nota musical diferente». Las frecuencias van desde 16 hasta 4.000 en placas de 8 MHz y de 31 a 8.000 en placas de 16 MHz. El último es el Arduino Uno Rev. 3.

También se puede utilizar el ejemplo de tipo digital llamado «tonemelody» que proporciona el IDE de Arduino. Este tiene en un fichero .h definidos diferentes sonidos que se utilizan en diferentes ejemplos de Internet.

El -10 hace «silencio».

```
#include "pitches.h"
```

```

int pinbuzzer = 11;

int song[] = {261, 349, 392, 440, 392, 330, -10, 261, 349, 392, 440,
392, -10, -10, 261, 349, 392, 440, 392, 330, -10, 330, 349, 330, 261,
261};
//int song[] = {NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0,
NOTE_B3, NOTE_C4};

//int tones[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440};
//          mid C  C#   D    D#    E     F      F#    G     G#    A

void setup(){
  Serial.begin(9600);
  pinMode(pinbuzzer, OUTPUT);
}

void loop(){
  for (int i = 0; i < sizeof(song)/sizeof(int); i++) {
    tone(pinbuzzer, song[i]);
    delay(500);
  }
}

```

### Pitches.h

```

/*****
 * Public Constants
 *****/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175

```

```

#define NOTE_FS3 185
#define NOTE_G3 196
#define NOTE_GS3 208
#define NOTE_A3 220
#define NOTE_AS3 233
#define NOTE_B3 247
#define NOTE_C4 262
#define NOTE_CS4 277
#define NOTE_D4 294
#define NOTE_DS4 311
#define NOTE_E4 330
#define NOTE_F4 349
#define NOTE_FS4 370
#define NOTE_G4 392
#define NOTE_GS4 415
#define NOTE_A4 440
#define NOTE_AS4 466
#define NOTE_B4 494
#define NOTE_C5 523
#define NOTE_CS5 554
#define NOTE_D5 587
#define NOTE_DS5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_FS5 740
#define NOTE_G5 784
#define NOTE_GS5 831
#define NOTE_A5 880
#define NOTE_AS5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_CS6 1109
#define NOTE_D6 1175
#define NOTE_DS6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_FS6 1480
#define NOTE_G6 1568
#define NOTE_GS6 1661
#define NOTE_A6 1760
#define NOTE_AS6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
#define NOTE_CS7 2217
#define NOTE_D7 2349
#define NOTE_DS7 2489
#define NOTE_E7 2637
#define NOTE_F7 2794
#define NOTE_FS7 2960
#define NOTE_G7 3136
#define NOTE_GS7 3322
#define NOTE_A7 3520
#define NOTE_AS7 3729
#define NOTE_B7 3951
#define NOTE_C8 4186
#define NOTE_CS8 4435
#define NOTE_D8 4699
#define NOTE_DS8 4978

```

## LED RGB



**Led RGB:** circuito integrado con un LED RGB que se compone de 3 LEDs, cada uno con un color primario (Rojo, azul y verde), y permite alumbrar el LED en cualquiera de estos 3 colores o realizar combinaciones entre todos estos ellos.

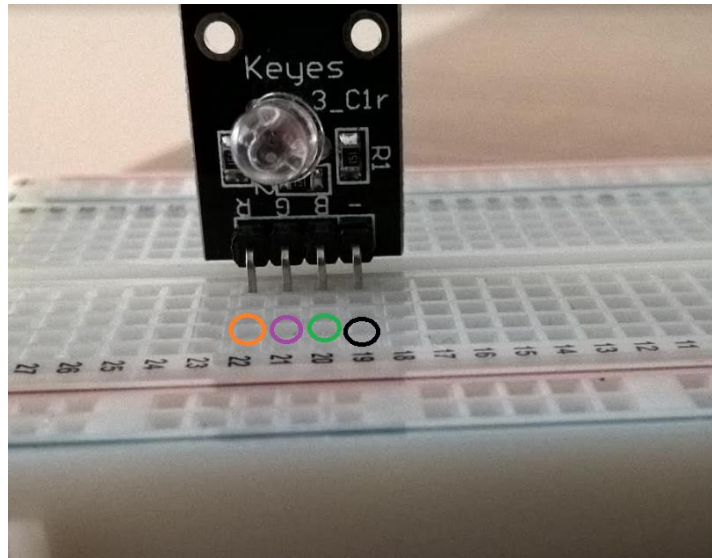
El LED tiene 4 pines:

- **R** que se conecta a un pin digital (PWM o normal) para controlar el nivel de luz roja
- **G** que se conecta a un pin digital (PWM o normal) para controlar el nivel de luz verde
- **B** que se conecta a un pin digital (PWM o normal) para controlar el nivel de luz azul
- - que se conecta a **GND**. ¡Hay que conectarlo siempre!

El LED RGB **no requiere utilizar una resistencia externa, pues la lleva integrada**, salvo que se usara corrientes con un voltaje mayor. En el caso de que no viniera todo integrado en una placa, lo recomendado sería que cada color tuviera su propia resistencia para así igual el brillo de todos los colores. Si se usa una resistencia para los 3 LED, o una con el mismo color en los tres colores, el rojo brillaría más al requerir menor voltaje o incluso podría sobrecargarse. Salvo que el LED RGB ya venga preparado para solo utilizar una única resistencia.

Dispone de una patilla negativa (que se debe conectar al GND) y de tres patillas que deben ser conectadas a pines digitales, cada una de las patillas se corresponde con un color R, G, B. El color se iluminará si enviamos voltaje como salida por el pin correspondiente. Podemos utilizar solo una patilla de color R, G, B o varias. Al enviar voltaje de forma simultánea por varias patillas podemos realizar combinaciones de colores.

Luz RGB	Placa
<b>R</b>	Pin digital
<b>G</b>	Pin digital
<b>B</b>	Pin digital
-	GND



*Ilustración 11 Conexión. VCC a 5V (Rojo) GND a GND (Negro) Out a un pin digital (Verde)*

Podemos utilizar la función `digitalWrite(<pin>, LOW/HIGH)` si queremos **alumbrar al máximo** de la capacidad del LED o apagarlo.

Si quisiéramos hacer **combinaciones** más complejas **de colores**, por ejemplo, un poco de rojo y mucho verde, podemos utilizar el `analogWrite(<pin>, [0-255])`, pero en este caso tenemos que estar seguros de **usar pines PWM (los señalados con ~)**.

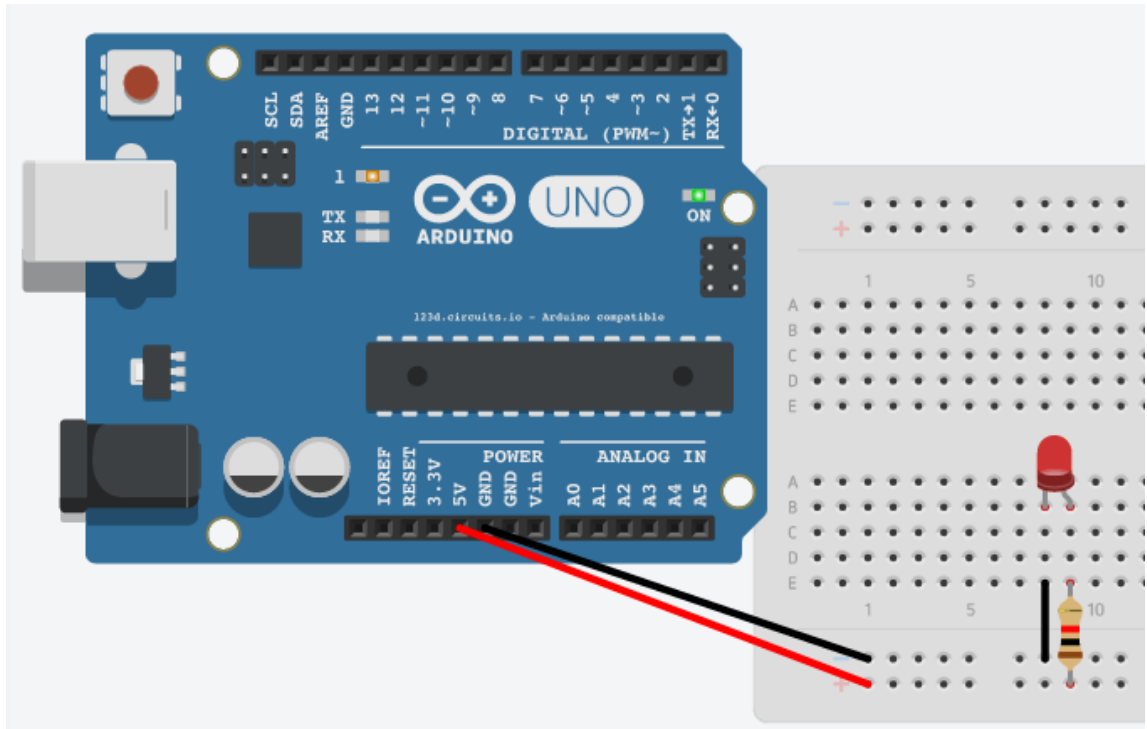
Si por ejemplo, solo queremos utilizar el LED como rojo y verde, basta con conectar los pines digitales a la patilla R y G, y el GND a GND, pudiendo dejar la B sin conexión.

## Ejemplos de circuitos

### Circuito básico

Vamos a crear un circuito muy básico para conectar un Led.

Para ello, utilizaremos los siguientes componentes: 3 LEDs, 1 pulsador, y 3 resistencias.



Construcción del circuito:

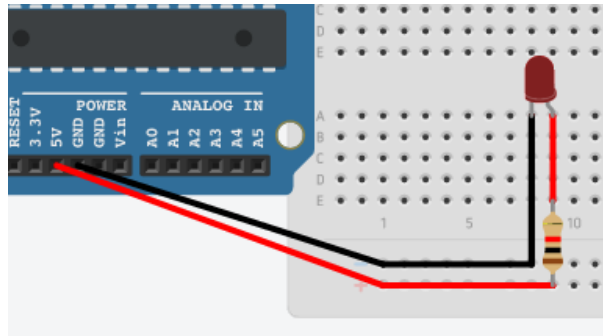
1. Conectamos el pin **5V** al carril positivo de la ProtoBoard (Rojo).
2. Conectamos el pin **GND** al carril negativo del ProtoBoard (Azul).
3. Colocamos el Led rojo en la ProtoBoard. Cuidado con la polaridad del Led.
4. Utilizamos la resistencia para abrir un carril positivo que se conecte con la parte positiva del Led (Extremo largo). Mirar que resistencia es la adecuada.

Los LEDs apenas presentan resistencia, luego, si toda la corriente pasa por el LED este se quemará. Para evitar esto, debemos colocar primero una resistencia de  $220\text{-}330\ \Omega$ , depende de la luminosidad que queramos.

5. Cerramos el circuito uniendo el carril de la pata negativa del led con el carril negativo del circuito.

¿Cómo sería este circuito si no estuviésemos utilizando la ProtoBoard?

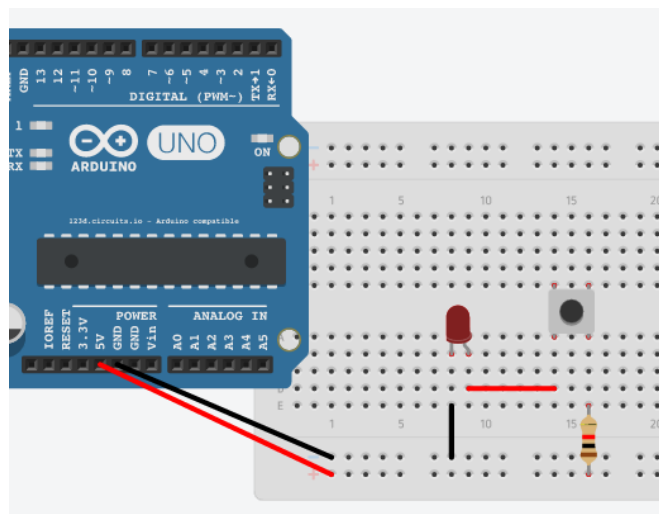
Sería algo similar a lo siguiente, con un cable en la zona de polaridad y un cable en cada carril.



Una vez comprobamos que el circuito funciona correctamente vamos a extenderlo, agregando un pulsador para cortar / abrir la corriente.

Incluir un pulsador para cortar / abrir el circuito

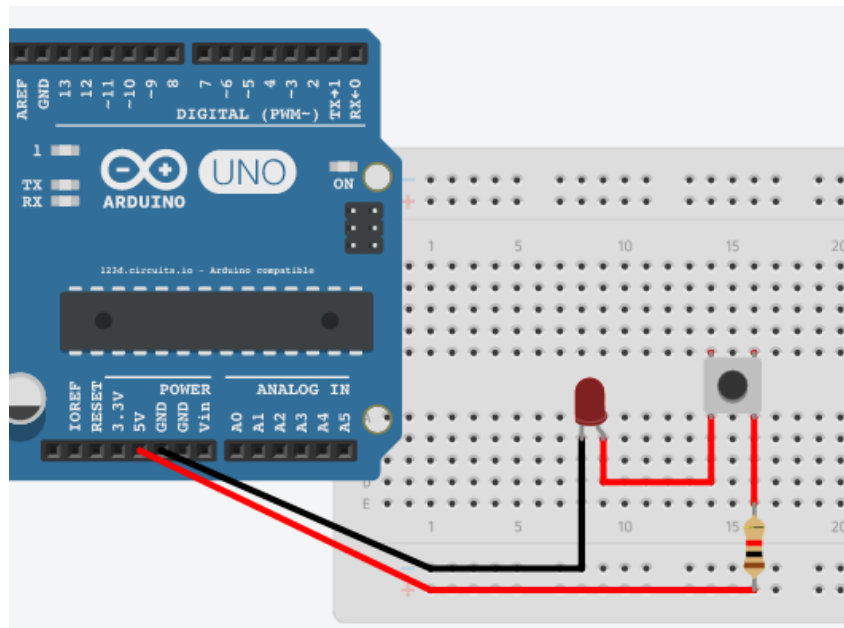
Si conectamos una patilla del pulsador a cada carril este hará pasar la electricidad de uno a otro cuando esta pulsado, ya que, al pulsarlo, se cierra el circuito. Si no, solo pasaría la electricidad a la patilla del mismo lado.



¿Cómo sería este circuito si no estuviésemos utilizando la ProtoBoard?

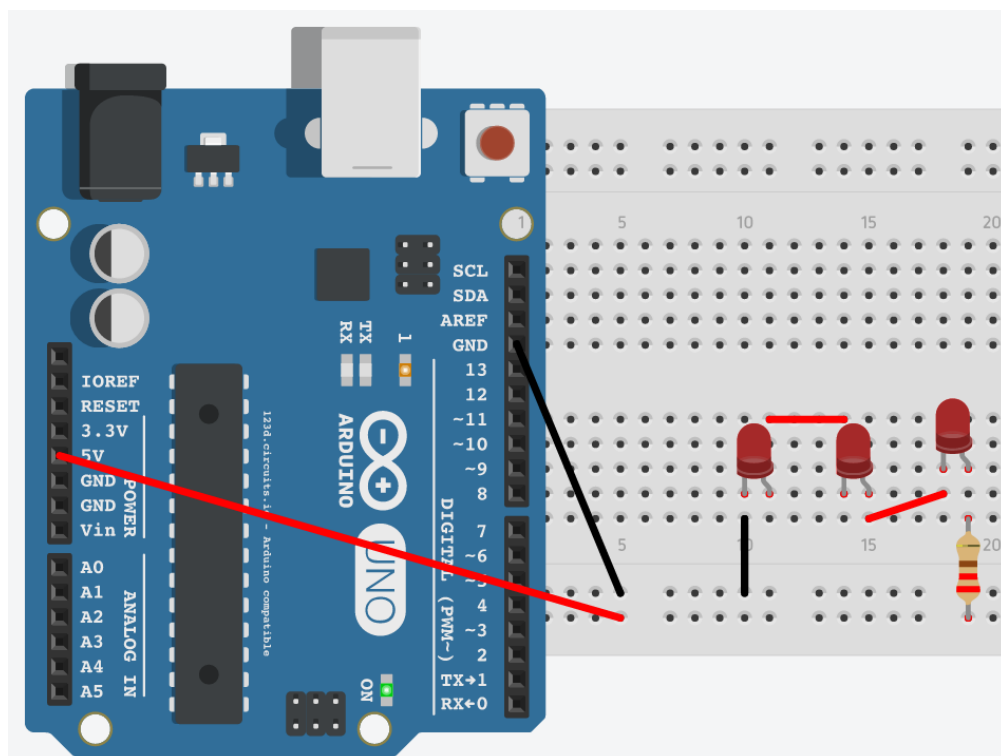
Sería algo similar a lo siguiente, con un cable en la zona de polaridad y un cable en cada carril.





### Circuito en serie

Vamos a probar a conectar varios leds en serie para ver el efecto de la corriente eléctrica.

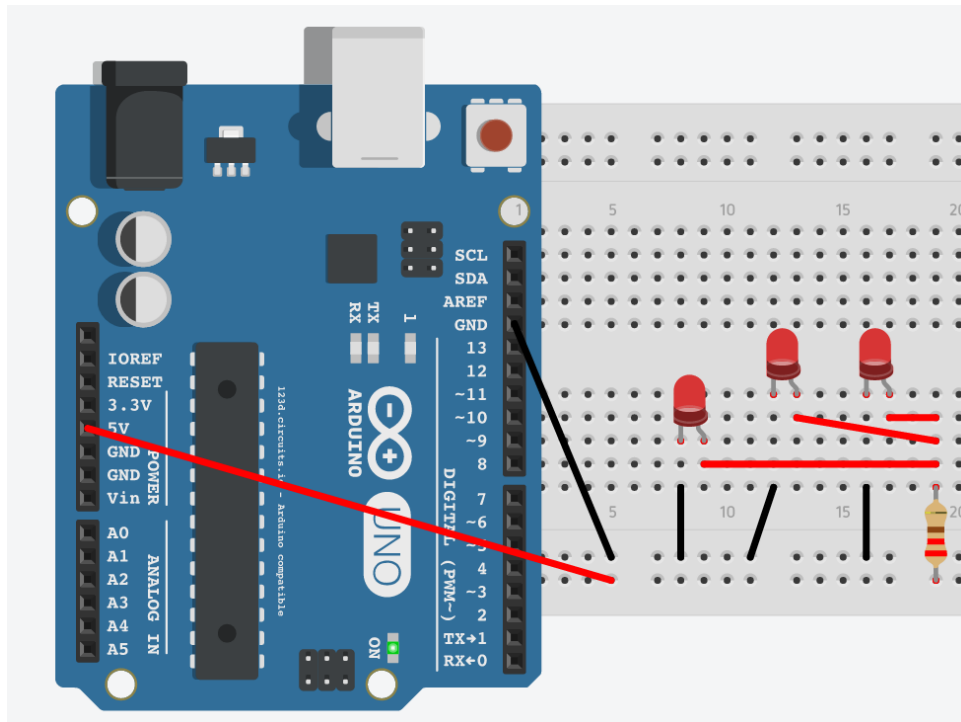


Como comentamos anteriormente, un Led hace caer el voltaje, luego, si colocamos varios leds en serie las caídas de voltaje se suman. **Esta mayor caída de voltaje unida a la resistencia hace que los leds apenas se iluminen o ni se iluminen.**

- Este circuito requeriría un mayor voltaje para encender todos los Leds, o una resistencia más baja.
- Si un Led se rompe, el circuito dejará de funcionar y todos los Leds se apagarán.

- La intensidad del circuito se mantiene respecto al primer ejemplo, por lo que consume la misma electricidad.

### Circuito en paralelo



Cuando conectamos los Leds en paralelo, todas las ramas reciben el mismo voltaje: el voltaje original menos la caída de un único led.

Los caminos con menor resistencia son los que mayor intensidad reciben (en este caso todos tienen la misma). En este caso, **todos los caminos reciben la misma intensidad provocando que los Leds se iluminen igual que antes, lo que aumenta aquí es el gasto de electricidad.**

- La intensidad que necesita este circuito en paralelo es mayor que la que vimos anteriormente en el circuito en serie.
- Incluso con un voltaje pequeño podríamos hacer funcionar muchos leds.
- Si un led se funde no le pasará nada al resto.

### Circuitos electrónicos y programación

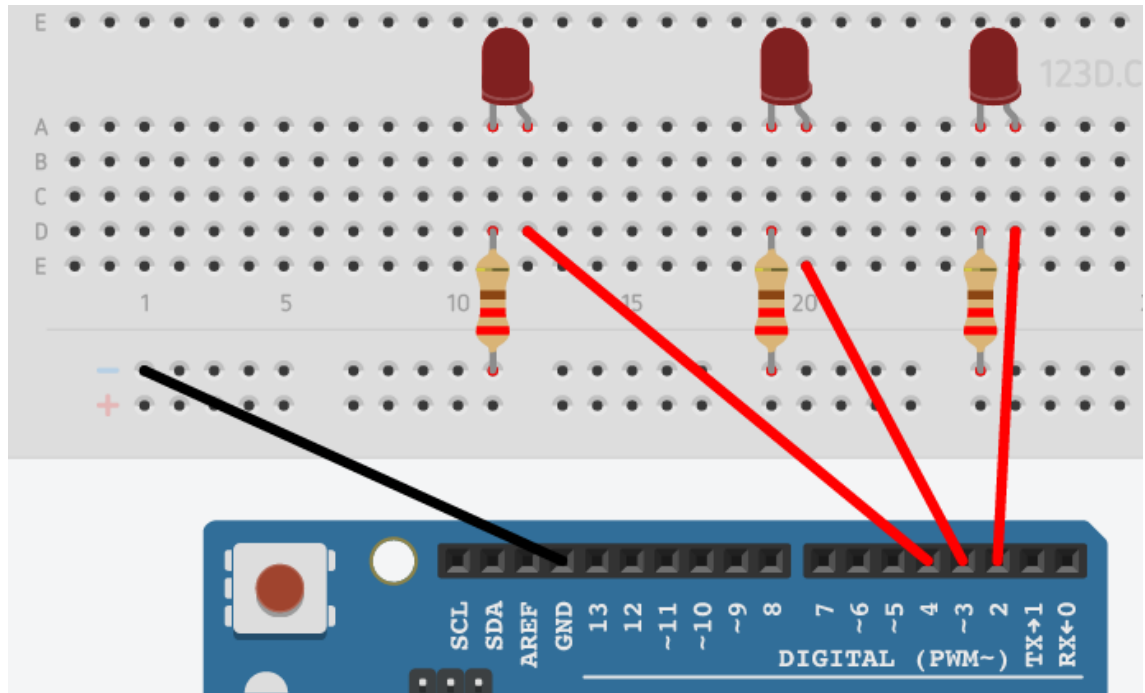
En nuestros proyectos futuros primará la parte de la programación del microcontrolador frente a la construcción de circuitos electrónicos.

La placa Arduino se utilizará para conectar varios elementos (sensores y actuadores), formando cada uno de ellos un pequeño circuito muy simple. La programación de las entradas y las salidas será la parte encargada de coordinar los elementos electrónicos.

### Salidas digitales - Leds que parpadean

Vamos a crear un sistema con 3 LEDs que se enciendan y se apaguen de forma ordenada, primero el A, luego el B y finalmente el C. Una vez terminada la secuencia está se repetirá de forma indefinida.

Para ello utilizaremos los siguientes componentes: LED, resistencia



En este sistema tenemos tres pequeños circuitos:

- Led 1 (Pin 2 - [salida 0V/5V] - Led 1 - Resistencia - GND).
- Led 2 (Pin 3 - [salida 0V/5V] - Led 2 - Resistencia - GND).
- Led 3 (Pin 4 - [salida 0V/5V] - Led 3 - Resistencia - GND).

Los tres circuitos tienen conectado un pin de Arduino, que en este caso servirá para abrir/cerrar la alimentación eléctrica de los leds. Desde un programa ejecutado en Arduino podremos controlar cuando abrir o cerrar la alimentación de cada pin.

Construcción del circuito:

1. Colocamos 3 leds en la ProtoBoard.
2. Conectamos el pin digital 2 a la parte positiva del primer led, el pin digital 3 a la parte positiva del segundo led y el pin digital 4 a la parte positiva del tercer led.
3. Conectamos el pin GND al carril negativo de la ProtoBoard.
4. Ahora debemos cerrar el circuito de cada uno de los tres leds. No podemos hacerlo directamente ya debemos incluir una resistencia para evitar que los leds se quemen. Añadimos la resistencia, en este caso en la parte negativa del led y aprovechamos la propia resistencia para conectar el led al carril negativo y así cerrar el circuito.

Programación de Arduino:

1. Inicializamos tres variables con los pins digitales que vamos a utilizar.

```
// Pins
int led1 = 2;
int led2 = 3;
int led3 = 4;
```

2. Dentro del método `setup()` declaramos los pins digitales que vamos a utilizar para darle electricidad a los leds.

```
void setup() {
  // Inicializamos los pins digitales como salida
  // Queremos que alimenten electricamente los leds
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
}
```

3. Dentro del `loop()` comenzamos a enviar las salidas digitales con el método `digitalWrite`.

- a. Esta función recibe el número del pin y el voltaje que se enviara (HIGH o LOW). En caso de HIGH se envían 5 voltios y de LOW 0 voltios, es decir, actúa como GND.
- b. <https://www.arduino.cc/en/pmwiki.php?n=Reference/DigitalWrite>

4. Encendemos el led 1 dentro de la función `loop`

```
digitalWrite(led1, HIGH); // Voltaje a - HIGH para el led 1.
delay(500);
```

5. Apagamos el led 1 y encendemos el led2

```
digitalWrite(led1, LOW); // Voltaje a - LOW para el led 1.
digitalWrite(led2, HIGH); // Voltaje a - HIGH para el led 2.
delay(500);
```

6. Apagamos el led2 y encendemos el led3

```
digitalWrite(led2, LOW); // Voltaje a - LOW para el led 1.
digitalWrite(led3, HIGH); // Voltaje a - HIGH para el led 2.
delay(500);
```

7. Apagamos el led3

```
digitalWrite(led3, LOW); // Voltaje a - LOW para el led 3.
```

```
// Pins
int led1 = 2;
```

```

int led2 = 3;
int led3 = 4;

void setup() {
  // Inicializamos los pins digitales como salida
  // Queremos que alimenten electricamente los leds
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
}

// El loop es llamado una y otra vez sin parar
void loop() {
  digitalWrite(led1, HIGH); // Voltaje a - HIGH para el led 1.
  delay(500);

  digitalWrite(led1, LOW); // Voltaje a - LOW para el led 1.
  digitalWrite(led2, HIGH); // Voltaje a - HIGH para el led 2.
  delay(500);

  digitalWrite(led2, LOW); // Voltaje a - LOW para el led 1.
  digitalWrite(led3, HIGH); // Voltaje a - HIGH para el led 2.
  delay(500);

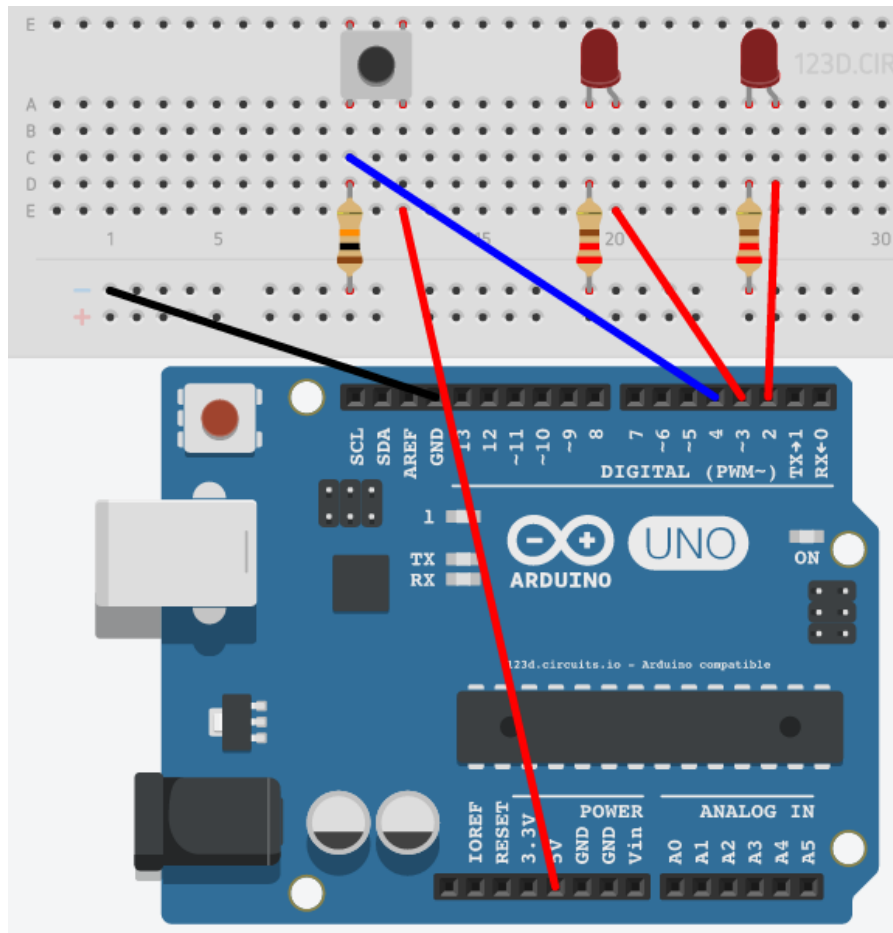
  digitalWrite(led3, LOW); // Voltaje a - LOW para el led 3.
}

```

### Entradas digitales - Controlando los leds con un botón

Vamos a crear un sistema con 2 LEDs y un botón. Cuando se pulse el botón se encenderá el LED 1 y cuando el botón se vuelva a pulsar se encenderá el led 2. Si se vuelve a pulsar el botón, se apagarán los dos leds y se volverá al estado inicial.

Para ello utilizaremos los siguientes componentes: 1 LED, 2 resistencias de  $220\ \Omega$  y una de  $10\ k\Omega$ , 1 pulsador.



En este sistema tenemos varios pequeños circuitos:

- Botón (5V - Botón - Pin 4 [entrada 0V/5V]- Resistencia - GND).
- Led 1 (Pin 3 - [salida 0V/5V]- Led 1 - Resistencia - GND).
- Led 2 (Pin 2 - [salida 0V/5V]- Led 2 - Resistencia - GND).

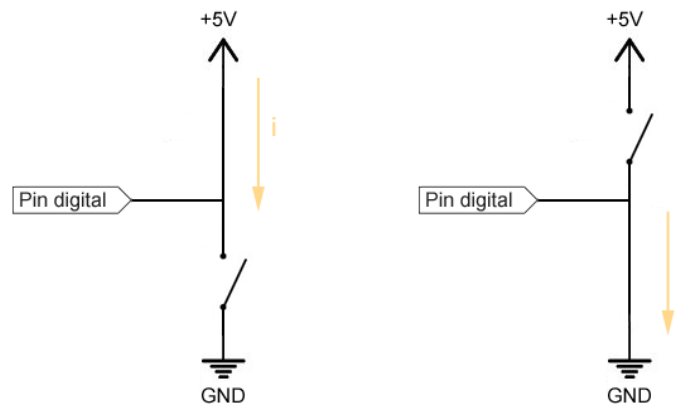
Los tres circuitos tienen conectado un pin de Arduino, que sirven, en el primer caso, para saber si el circuito del botón está abierto/cerrado, y en los otros dos casos para abrir/cerrar la alimentación eléctrica de los leds.

El Arduino es el elemento central que está sirviendo para manejar el funcionamiento de esos tres circuitos.

Construcción del circuito:

1. Colocamos 2 leds en la ProtoBoard.
2. Conectamos el pin digital 2 a la parte positiva del primer led, el pin digital 3 a la parte positiva del segundo LED.
3. Conectamos el pin GND al carril negativo de la ProtoBoard.
4. Ahora debemos cerrar el circuito de cada uno de los dos leds. No podemos hacerlo directamente **debemos incluir una resistencia para evitar que los leds se quemen**. Añadimos la resistencia, en este caso, en la parte negativa del led y aprovechamos la propia resistencia para conectar el led al carril negativo y así cerrar el circuito.

5. Colocamos el pulsador en la ProtoBoard y conectamos a una patilla del pulsador el cable de alimentación de 5V.
6. Conectamos a la otra patilla del pulsador el pin digital 4. Desde este pin podremos leer el voltaje de este pequeño circuito.
7. Ahora deberíamos cerrar el circuito, pero **no podemos conectar el interruptor directamente al carril negativo ya que se produciría un cortocircuito.**



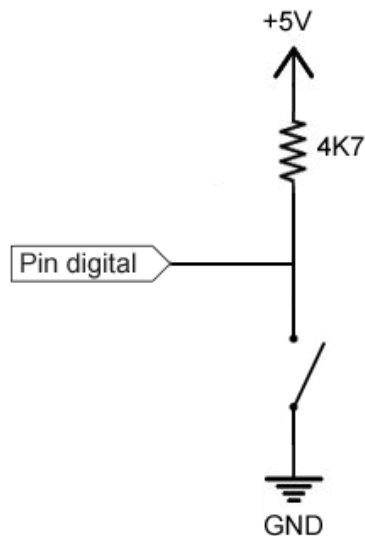
**Nunca debemos conectar directamente 5V a GND o provocaremos un cortocircuito, hay que incluir una resistencia de por medio siempre y cuando los elementos que utilicemos no tengan resistencias incorporadas o no hagan apenas resistencia a la electricidad, como ocurre con el pulsador y el LED.**

Si conectamos directamente 5V y GND (0V) causaremos un cortocircuito por el elevado paso de corriente, por lo que produciría un calentamiento de los elementos, pudiendo llegar incluso al incendio.

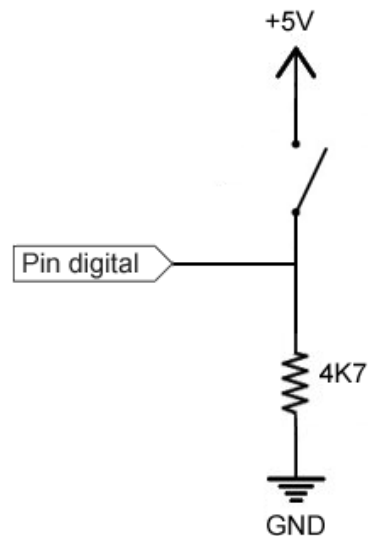
El pin digital que tiene como misión leer el voltaje de esa conexión tampoco funcionaría correctamente, ya que estaría conectado simultáneamente a: 5V y GND (0V).

Para solucionar esta situación debemos colocar una resistencia **Pull-Down o Pull-Up**. Este tipo de resistencias se colocan entre los extremos **5V y GND** para forzar el valor de la tensión.

### RESISTENCIA PULL UP



### RESISTENCIA PULL DOWN



El esquema **Pull-Up** fuerza a HIGH el valor cuando el pulsador está abierto (Si está cerrado el valor será LOW).

El esquema **Pul-Down** fuerza a LOW el valor cuando el pulsador está abierto (Si está cerrado el valor será HIGH).

¿Qué resistencia hay que colocar?

- Una resistencia muy pequeña deja pasar mucha corriente (0 o cerca de 5), por lo que las mediciones del voltaje del circuito resultan bastante precisas, pero supone un mayor consumo y calentamiento.
- Una resistencia muy grade, deja pasar muy poca corriente (0 o lejos de 5), por lo que las mediciones de voltaje del circuito son más propensas a mediciones incorrectas (ruido).

Para la tensión de Arduino 0-5V se suelen colocar resistencias de 4k-10k, pues suponen un consumo de electricidad de 1mA y 0,5mA cuando el circuito está activo.

Así que utilizamos la resistencia de 10k para conectar el pulsador a GND siguiendo el esquema Pull-Down.

Programación de Arduino:

```
// Pins
int led1 = 2;
int led2 = 3;
int button = 4;
int buttonState= 0;
// 0: todo apagado
// 1: led1 encendido
// 2: led2 encendido

void setup() {
```



```

Serial.begin(9600); // Iniciar el Serial
Serial.println("Setup");

// Inicializamos los pines digitales como salida
// Queremos que alimenten electricamente los leds
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
// Queremos leer la entrada
pinMode(button, INPUT);
}

// La funcion loop se llama una y otra vez hasta el infinito
void loop() {
    int read = digitalRead(button);

    if(read == HIGH){
        Serial.println("Valor HIGH");
        if(buttonState == 2){
            buttonState = 0;
        } else {
            buttonState++;
        }

        switch(buttonState){
            case 0:
                digitalWrite(led1, LOW);
                digitalWrite(led2, LOW);
                break;
            case 1:
                digitalWrite(led1, HIGH);
                digitalWrite(led2, LOW);
                break;
            case 2:
                digitalWrite(led1, LOW);
                digitalWrite(led2, HIGH);
                break;
        }
    }
}
}

```

Probamos el programa y detectamos un problema

Cuando pulsamos el botón, los LEDs comienzan a parpadear, y al soltarlo, los LEDs se quedan en un estado «aleatorio».

¿Porque está funcionando de esta forma? Mientras mantenemos el botón pulsado el método `loop()` se sigue ejecutando, por lo que probablemente se haya ejecutado cientos de veces y el estado de los leds habrá cambiado en cada una de esas ejecuciones.

Tenemos dos estrategias para mejorar el sistema:

**Estrategia 1:** la más simple. Cada vez que se produce un cambio de estado en los LEDs colocar un «`delay(1000);`». De esta forma nos aseguramos que el botón tenga que estar pulsado al menos 1 segundo para realizar un nuevo cambio en el estado de las luces.

```

case 2:
    digitalWrite(led1, LOW);

```

```

        digitalWrite(led2, HIGH);
        break;
    }
    delay(1000);
}
}

```

**Estrategia 2:** creamos una variable que controle que la lógica del botón no se ejecute más de una vez, a no ser que el usuario levante el dedo del botón, en cuyo caso si se podrá volver a ejecutar.

```

...
int pushed = 0;
// 0: botón sin pulsar
// 1: botón pulsado

void loop() {
    ...

    // Si han pulsado el botón y estaba sin pulsar ->
    if(read == HIGH && pushed == 0){
        // botón pulsado!
        pushed = 1;

        ...

        // Si han quitado el dedo del botón cambio su estado
    } else if (read == LOW && pushed == 1){
        Serial.println("Valor LOW");
        pushed = 0;
    }
}

```

#### Monitor de serie:

El monitor de serie nos permite enviar y recibir datos desde Arduino. Para comenzar a utilizarlo lo inicializamos en el método **setup()**.

```
Serial.begin(9600); // Iniciar el serial
```

A partir del momento de su inicialización podemos escribir mensajes que se mostrarán por el monitor de serie.

```
Serial.println("Mensaje");
```

Para ver los mensajes debemos abrir el monitor de serie. **Solo se puede abrir si la placa está conectada.**



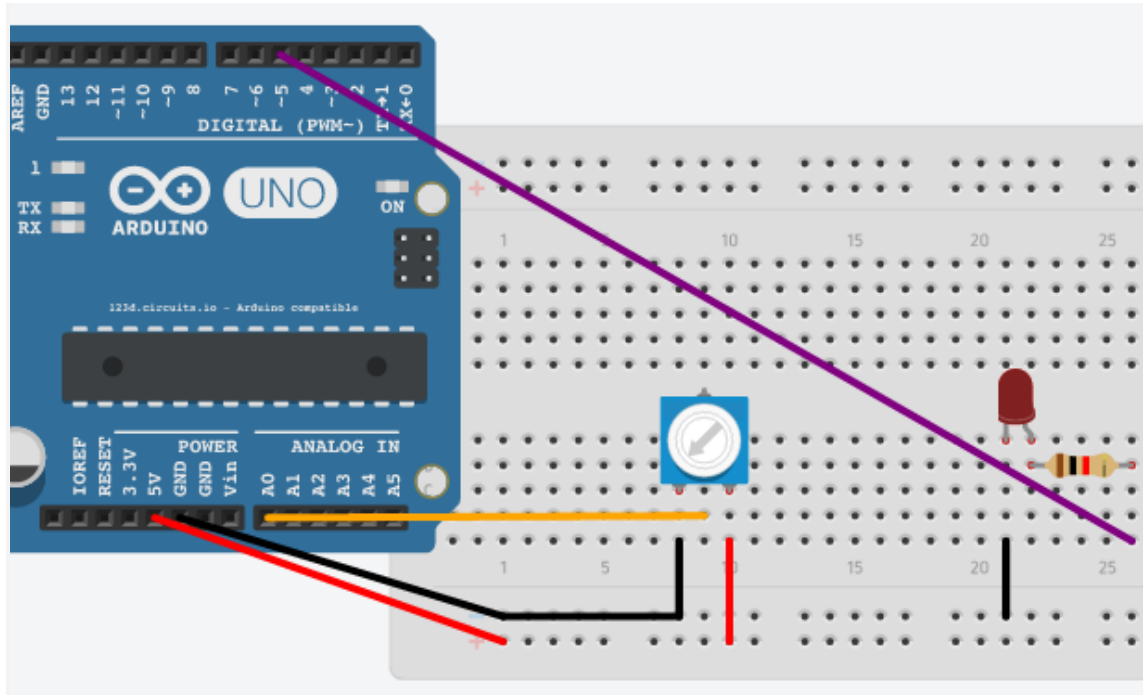
El método `println()` no realiza **conversiones automáticas de tipos de datos**. Si queremos incluir un valor no String en la cadena debemos convertirlo previamente. Ejemplo:

```
Serial.println("Valor de la entrada: "+ String(value));
```

### Lectura de entrada analógica

Vamos a crear un sistema que regule la intensidad de un diodo LED utilizando un potenciómetro. Al mover la rueda del potenciómetro cambiaremos la intensidad de la luz del LED.

Para ello utilizaremos los siguientes componentes: 1 LED, 1 resistencia de  $220\Omega$ , 1 potenciómetro de  $10k\Omega$ .



En este sistema tenemos dos pequeños circuitos

- Led (Pin 5 - [salida analógica entre: 0V - 5V]- Resistencia - Led - GND).
- Potenciómetro 3 (5V - Potenciómetro – GND + Pin A0 [entrada analógica]).

Construcción del circuito:

1. Colocamos el potenciómetro en la ProtoBoard, conectamos su pin derecho e izquierdo a los carriles positivo y negativo.
2. Conectamos el pin central del potenciómetro a la entrada analógica A0.
3. Colocamos el Led en la ProtoBoard, en su polo positivo conectamos una resistencia de  $220\Omega$ .
4. Conectamos el inicio de la resistencia al pin analógico ~5, que además de ser una fuente de voltaje, este pin es capaz de «regularlo», pues permite una escritura analógica.
  - a. <https://playground.arduino.cc/Learning/Pins>
  - b. <https://www.arduino.cc/en/Tutorial/DigitalPins>
5. Para cerrar el circuito conectamos el polo negativo del LED al carril negativo.

Programación de Arduino:

1. Inicializamos dentro del método `setup()` el puerto serial para poder escribir mensajes.

```
Serial.begin(9600); // Iniciar el Serial
```

2. También declaramos los pines digitales que vamos a utilizar. En este caso el pin 5 como salida para darle electricidad al Led.

```
// Los pins digitales se declaran - LED  
pinMode(5, OUTPUT); // Salida
```

3. Creamos dos variables globales para almacenar el valor del potenciómetro y el valor que le daremos a la salida analógica.

4. Dentro del `loop()` leemos la entrada analógica A0 donde está conectado el potenciómetro.

- a. <https://www.arduino.cc/en/Reference/analogRead>

```
valuePotenciometer = analogRead(A0); // 0 - 1024
```

5. La función `analogRead()` devuelve valores entre 0 - 1024, pero la salida analógica admite valores de 0 - 255. Por ello, realizamos una conversión antes de escribir la salida.

- a. <https://www.arduino.cc/en/Reference/Map>

```
// Transformar de 0 a 1023 -> 0 a 255  
valueAnalogPin = map(valuePotenciometer, 0, 1023, 0, 255);
```

6. Escribimos el valor convertido en la salida analógica 5.

```
// Escribimos el valor en el pin 5 - LED  
analogWrite(5, valueAnalogPin);
```

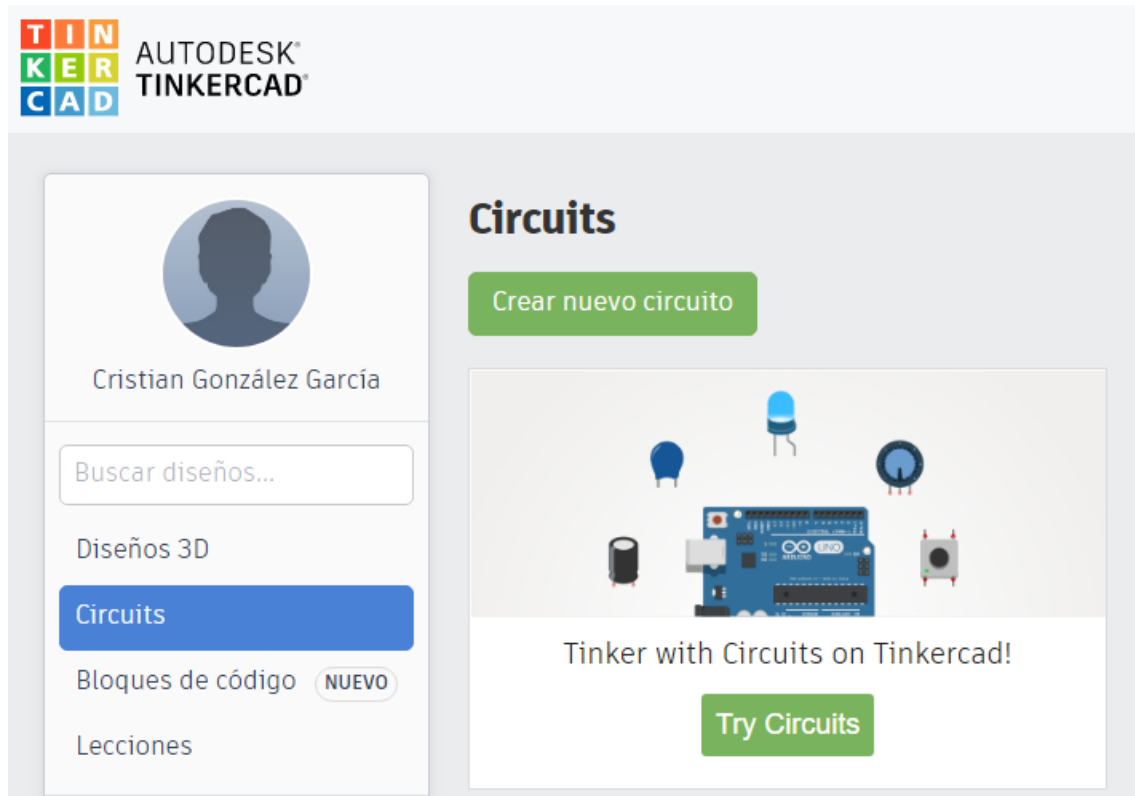
7. Código de ejemplo:

```
int valuePotenciometer;  
int valueAnalogPin;  
  
void setup() {  
  Serial.begin(9600); // Iniciar el Serial  
  Serial.println("Setup");  
  
  // Los pins digitales se declaran - LED  
  pinMode(5, OUTPUT); // Salida  
}  
  
void loop() {  
  // Leemos el valor del pin A0 - Potenciómetro  
  valuePotenciometer = analogRead(A0); // 0 - 1024  
  Serial.println("valor: "+String(valuePotenciometer));  
  
  // Transformar de 0 a 1023 -> 0 a 255  
  valueAnalogPin = map(valuePotenciometer, 0, 1023, 0, 255);  
  // Escribimos el valor en el pin 5 - LED  
  analogWrite(5, valueAnalogPin);  
}
```

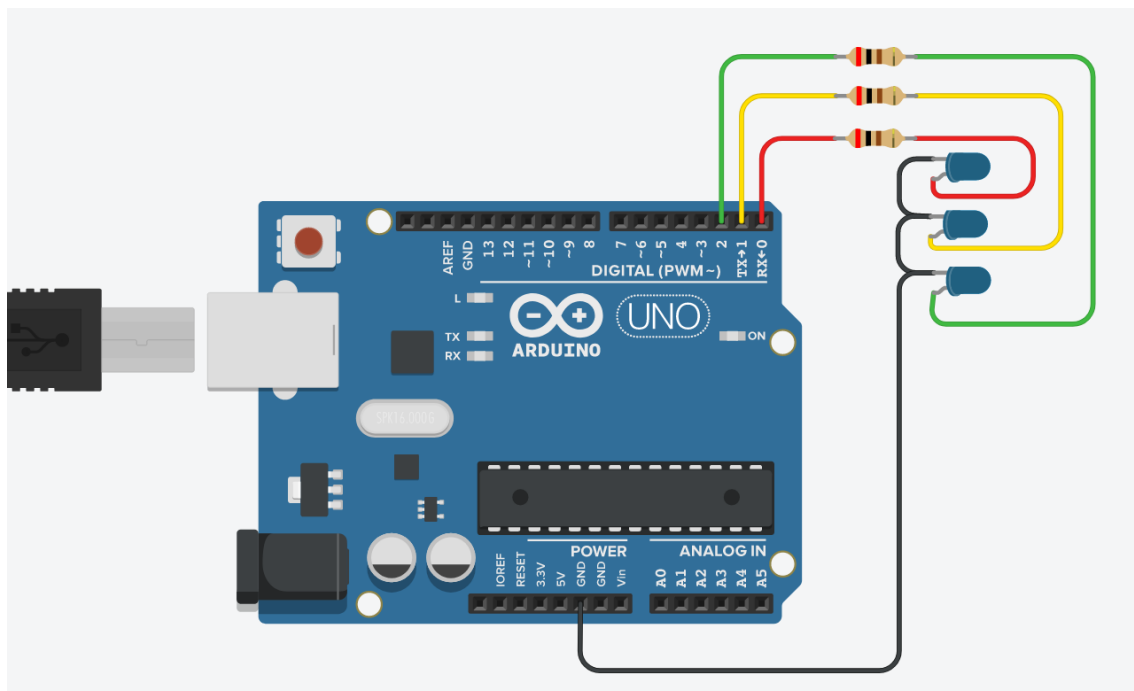
Ampliación:

## Emulador de Arduino: Tinkercad - Circuits

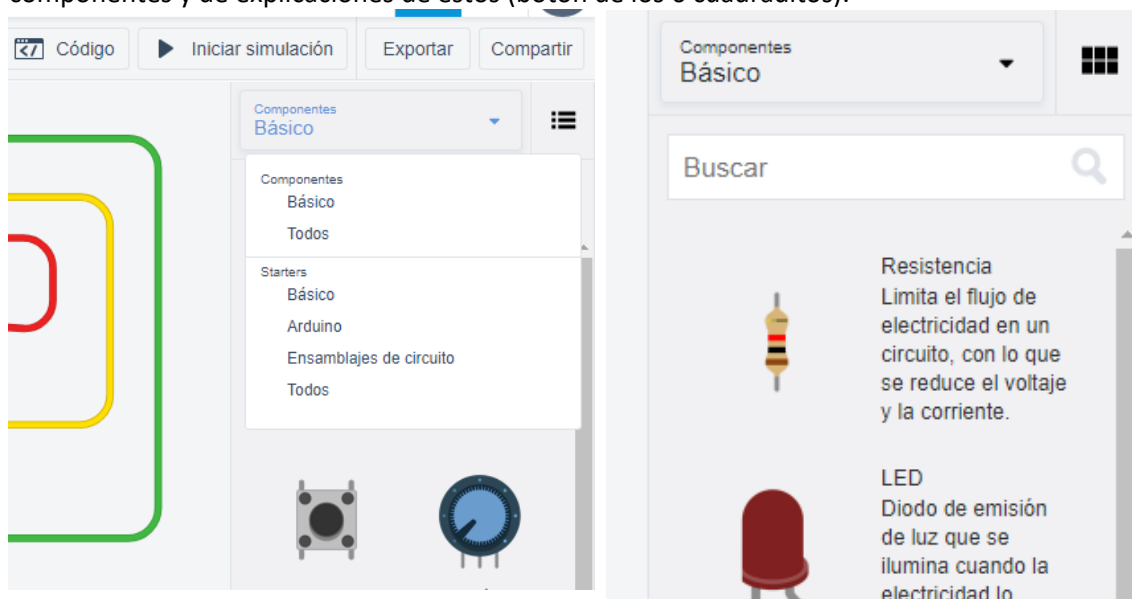
Es una plataforma web que permite crear circuitos utilizando Arduino. El emulador soporta muchos sensores y actuadores diferentes. Además, permite simular el circuito para comprobar si funciona o no. Por ejemplo, si usamos LED, estos se encenderán o no según lo que hayamos programado y según hayamos montado el circuito.



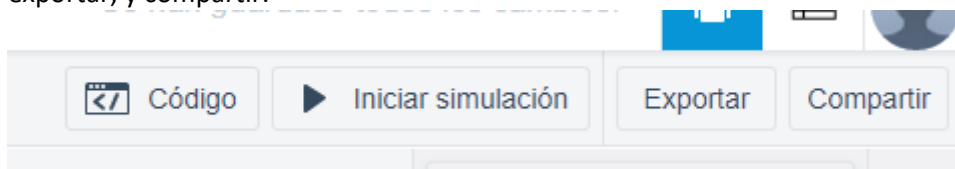
Esta plataforma está muy bien para probar nuestros circuitos en caso de no disponer una placa Arduino, o si no sabemos si funcionará bien y pueda haber peligro de estropear la placa. Además, debido a que es un simulador, puede ser más fácil en el caso de que haya muchos cables. También se pueden compartir los diseños y probar los de otras personas.



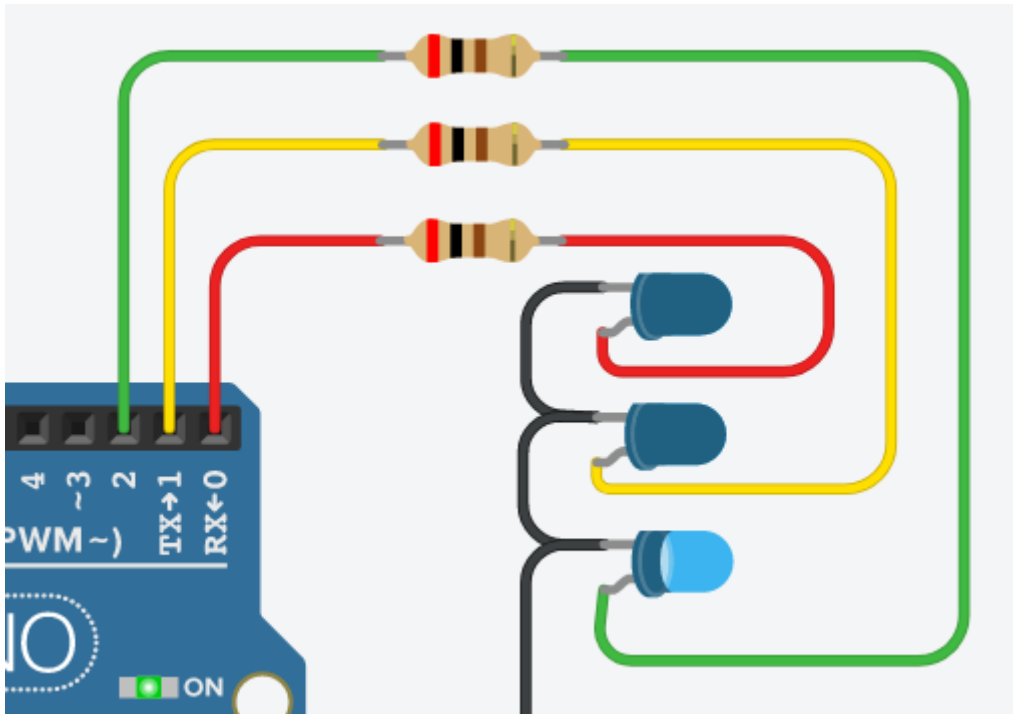
El emulador es de tipo «Drag and Drop» y permite cambiar las opciones del elemento al hacer clic sobre él (colores, tipo de resistencia, etc.). Dispone de varias opciones en las que podemos ver los componentes básicos o todos los existentes en el programa. Dispone de buscadores de componentes y de explicaciones de estos (botón de los 6 cuadraditos).



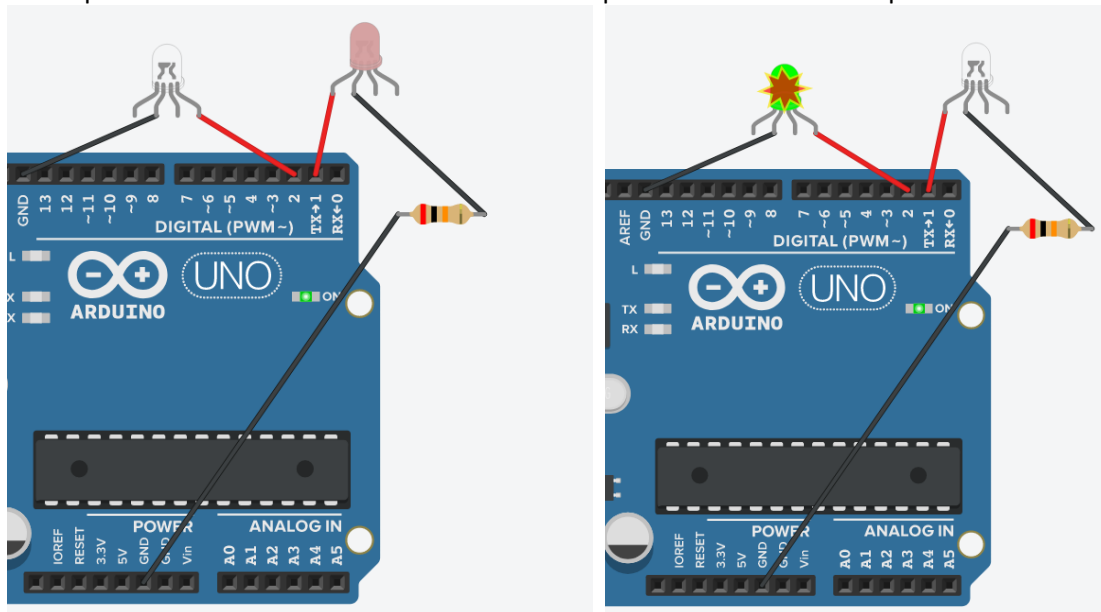
Arriba a la derecha de la pantalla, podéis ver el menú de opciones: código, Iniciar simulación, exportar, y compartir.



Si se inicia una simulación, se puede ver como los LEDs se iluminan y la placa está encendida.

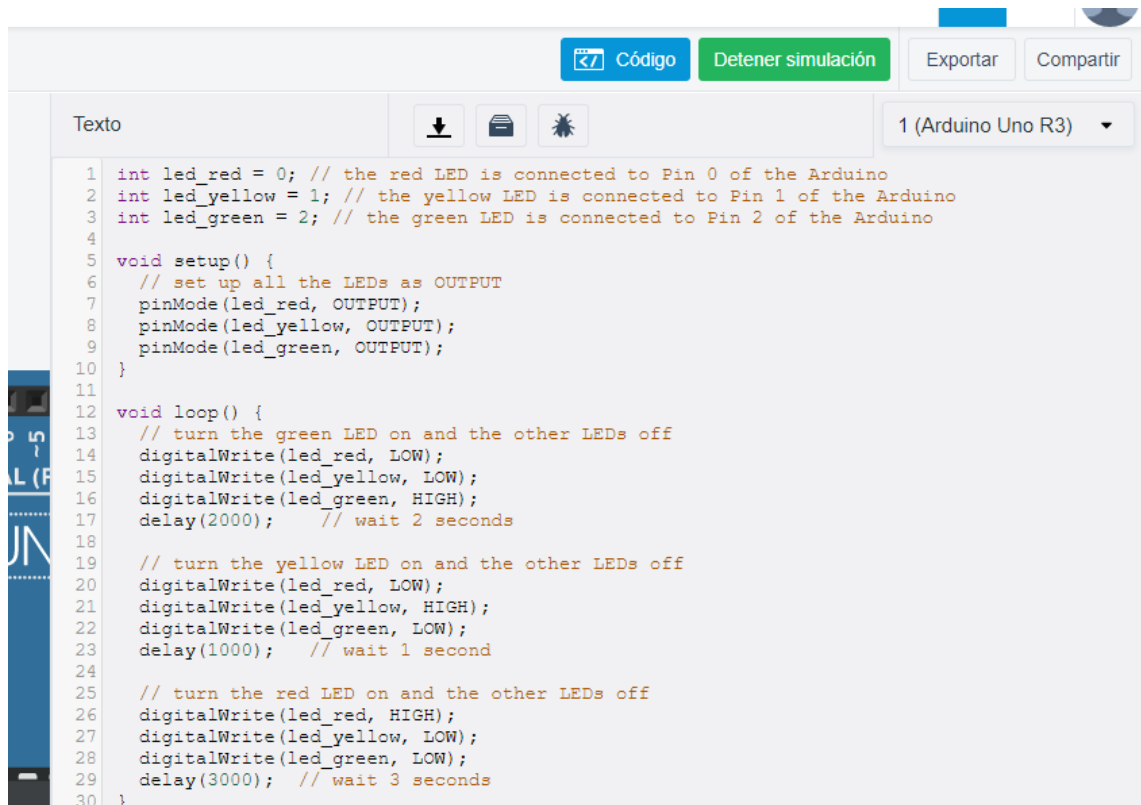


En cambio, por ejemplo, el circuito montado estuviera mal, este nos lo indicaría. En la Ilustración 12 vemos, como al faltar una resistencia, el LED tiene una imagen imitando la explosión. No obstante, no con todo ocurre este aviso, por ejemplo, con un pulsador mal conectado no avisa, ni tampoco si se conecta la toma tierra de un componente a una toma de potencia.



*Ilustración 12 Circuito incorrecto por falta de resistencia*

Si se hace clic en código, nos la pestaña en dónde se escribe el código fuente y en dónde se debe programar. Además, cuenta con otras tres opciones: descarga, bibliotecas, y depurador.

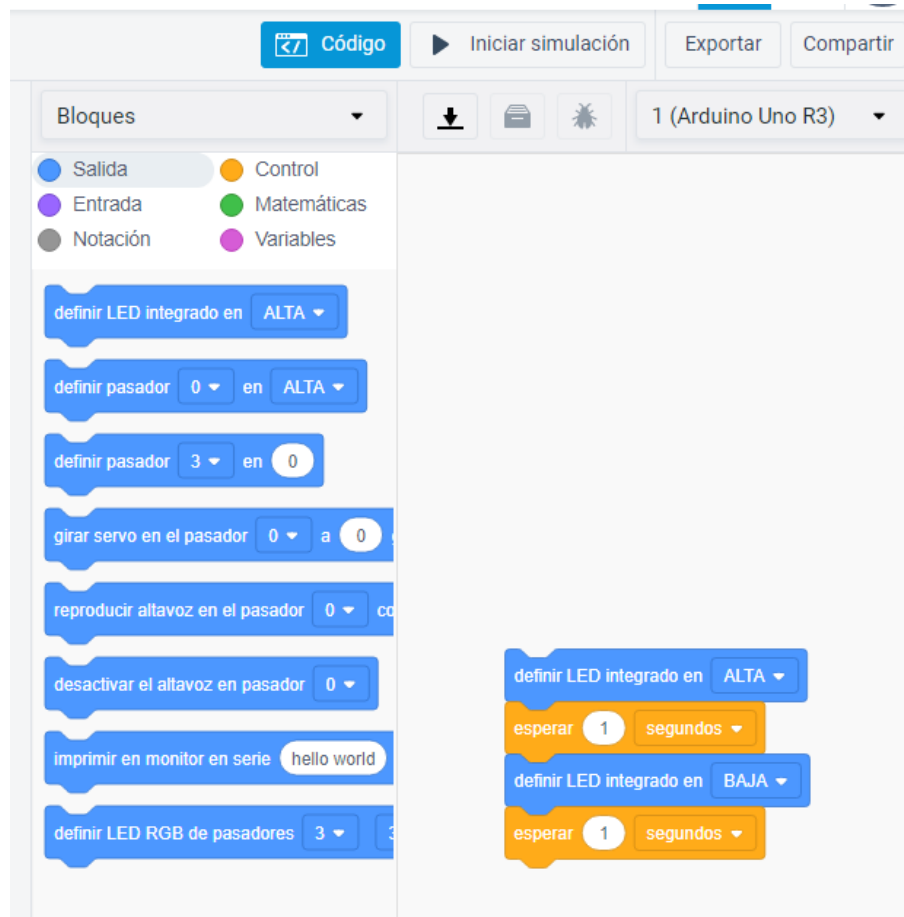


```
1 int led_red = 0; // the red LED is connected to Pin 0 of the Arduino
2 int led_yellow = 1; // the yellow LED is connected to Pin 1 of the Arduino
3 int led_green = 2; // the green LED is connected to Pin 2 of the Arduino
4
5 void setup() {
6   // set up all the LEDs as OUTPUT
7   pinMode(led_red, OUTPUT);
8   pinMode(led_yellow, OUTPUT);
9   pinMode(led_green, OUTPUT);
10 }
11
12 void loop() {
13   // turn the green LED on and the other LEDs off
14   digitalWrite(led_red, LOW);
15   digitalWrite(led_yellow, LOW);
16   digitalWrite(led_green, HIGH);
17   delay(2000); // wait 2 seconds
18
19   // turn the yellow LED on and the other LEDs off
20   digitalWrite(led_red, LOW);
21   digitalWrite(led_yellow, HIGH);
22   digitalWrite(led_green, LOW);
23   delay(1000); // wait 1 second
24
25   // turn the red LED on and the other LEDs off
26   digitalWrite(led_red, HIGH);
27   digitalWrite(led_yellow, LOW);
28   digitalWrite(led_green, LOW);
29   delay(3000); // wait 3 seconds
30 }
```

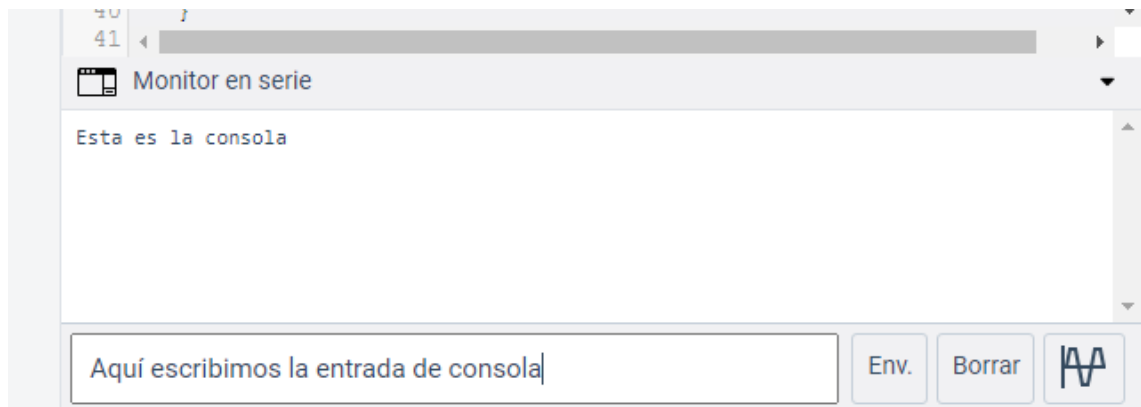
También dispone de una opción de crear el código fuente utilizando piezas de puzle. Este modo sirve para que la gente que no tiene conocimientos de programación pueda aprender y realizar cosas de una manera mucho más sencilla.

Además, si en el desplegable de debajo de código, que dice bloques, haces clic, podréis elegir entre utilizar el editor de bloques, bloques+texto, o solo texto.





En la parte inferior derecha del programa tenéis el botón «Monitor en serie». Al hacer clic en él se abrirá la consola. En ella podemos imprimir y recibir valores y textos.



### Notas sobre Tinkercad

- El cronómetro de Tinkercad puede ralentizarse algunas veces. Cuidado cuando se utilice no fiarse de él.

## Arduino

### Proyecto

La **extensión** del archivo es «**ino**». Además, este **debe ir en una carpeta que se llama igual** que el nombre del archivo sin la extensión. Este nombre **no puede llevar espacios**, pues sino dará un error.

### Puertos digitales

**Los puertos digitales 0 (RX) y 1 (TX) del Arduino UNO comparten interfaz con el USB.** El pin 0 o RX es el que lee la información y el pin 1 o TX es el que la transmite. Luego, si se usan estos pines mientras se está utilizando el Serial del Arduino, es decir, si se utiliza el puerto USB. **Si se usan ambos a la vez, la comunicación tendrá interferencias**, habrá problemas de subida de datos y puede que hagan cosas raras y q no obedezcan. Si se quisieran utilizar cuando no se conecte el Arduino a un USB, lo mejor es hacer comprobaciones y parar la lógica necesaria que funcione con los pines 0 y 1 para poder subir datos o usarlo mientras se tiene enchufado al USB-

Esto no ocurre con el Arduino Mega, pues dispone de 4 Serial, o de otras versiones de Arduino, que disponen de más Serial o usan otros pines diferentes.

Más información en:

- <https://www.arduino.cc/reference/en/language/functions/communication/serial/>

### Puerto serial: Serial.begin()

Para establecer una **comunicación** entre el **Arduino** y la consola del **ordenador**, pudiendo así visualizar cosas que imprimamos en el Arduino y/o enviarle datos desde la consola, deberemos **utilizar el objeto Begin y su método Serial**.

Este, se suele establecer en el método setup ya que solo se suele necesitar ejecutarlo una vez, en su inicialización. Como parámetro recibe los baudios. Normalmente, en la mayoría d ellos sitios se usa 9600, aunque se puede poner más o menos dentro de un rango de valores predefinido del Arduino. Cuantos más baudios, mayor velocidad de escritura por segundo en el puerto serial, pues es la medida que representa el número de símbolos por segundo en un medio de comunicación

```
void setup() {  
  Serial.begin(9600);  
}
```

Más información:

- <https://www.arduino.cc/en/Serial/Begin>

### Imprimir por pantalla/consola

Como en otros lenguajes de programación, podemos imprimir por pantalla/consola información. Esta información puede ser datos fijos, como una cadena de texto o una notificación, o datos dinámicos que genere nuestro programa o que recojamos de sensores.

Se utiliza el método **print** del objeto **Serial**. Este método recibe el dato a imprimir, que puede ser una cadena de texto o un valor numérico. También disponemos del método **println** que realizará un salto de línea tras la impresión.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  // Imprimimos por pantalla el string "Imprimimos el número 13:"
  Serial.print("Imprimimos el número 13: ");
  // Imprimimos por pantalla el int 13
  Serial.println(13);
}
```

Se pueden imprimir tabulaciones ("**\t**"), saltos de línea ("**\n**" o **println()**), etc.

Se puede **castear** un tipo **String** en otros tipos como **Int**, **Double**, etc., o **int/float** a otros tipos o como **String**. Este es el uso de diferentes funciones:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  String example = "5.1";
  float number = 5.1;

  Serial.println(example.toInt()); // Imprime 5
  Serial.println(example.toDouble()); // Imprime 5.10
  Serial.println(example.toFloat()); // Imprime 5.10
  //Serial.println(example.toString()); // No existe

  //Serial.println(int(example)); // Invalid cast
  //Serial.println(double(example)); // Invalid cast
  //Serial.println(float(example)); // Invalid cast
  Serial.println(String(example)); // Imprime 5.1

  Serial.println(int(number)); // Imprime 5
  Serial.println(double(number)); // Imprime 5.10
  Serial.println(float(number)); // Imprime 5.10
  Serial.println(String(number)); // Imprime 5
}
```

**Se recomienda** usar alguna de estas, sobre todo **para pasar un número a String** para evitar posibles problemas que puedan surgir alguna rara vez.

También se puede invocar a los métodos **print** y **println** con un segundo parámetro, dónde:

- 1er parámetro: número en base decimal, 2º parámetro: base (BIN, OCT, DEC, HEX) -> pasa el primer número a la base deseada.
- 1er parámetro: número con decimales, 2º parámetro: número de decimales (0, 1, 2, 3, ...) -> recorta los decimales a los indicados.

```
void setup() {
```

```

    Serial.begin(9600);
}

void loop() {
    Serial.println(13, BIN); // 1101
    Serial.println(13, DEC); // 13
    Serial.println(13, OCT); // 15
    Serial.println(13, HEX); // D
    Serial.println(3.141592, 0); // 3
    Serial.println(3.141592, 1); // 3.1
    Serial.println(3.141592, 2); // 3.14
    Serial.println(3.141592, 3); // 3.142
    Serial.println(3.141592, 4); // 3.1216 <- redondea
    Serial.println(3.141592, 5); // 3.12159
}

```

## Generar números aleatorios

Arduino tiene una función llamada «random» (<https://www.arduino.cc/reference/en/language/functions/random-numbers/random/>) que permite generar números aleatorios, y puede recibir un parámetro (0 a máximo+1) o dos (mínimo y máximo+1).. No obstante, para generar aleatorios, esta función necesita una semilla. Si no introdujéramos esta semilla, el Arduino nos generaría siempre la misma secuencia tras cada reinicio. Esto puede resultar útil en caso de que necesitemos siempre el mismo resultado, pero no si queremos obtener números aleatorios.

```

int pinForRandom = A0;

void setup() {
    Serial.begin(9600);
}

void loop() {
    // Imprimimos un número aleatorio entre 0 y 50
    Serial.print("Entre 0 y 50: ");
    Serial.println(random(51));

    // Imprimimos números aleatorios entre 7 y 12
    Serial.print("Entre 7 y 12: ");
    Serial.println(random(7, 13));

    delay(1000);
}

```

**Para generar un aleatorio** hay una guía sencilla en la página de Arduino: <https://www.arduino.cc/reference/en/language/functions/random-numbers/random/>. Básicamente, consiste en leer un pin analógico al aire (vacío, sin conectar) y usar la función «random», a la que previamente se le introduce una semilla usando la función «randomSeed». De esta manera, al usar un pin al aire, lo que conseguimos es que le pase como número a esa semilla el ruido que se lee de ese pin, que es muy variable al no tener algo conectado.

```

int pinForRandom = A0;

void setup() {
    Serial.begin(9600);

    /* Si usamos el pin 0 para generar aleatorios,
     * este pin deberá de estar vacío, es decir, no puede

```

```

    * tener nada conectado.
    * Le pasaremos este pin al generador de semillas
    */
    randomSeed(analogRead(pinForRandom));
}

void loop() {
    // Imprimimos un número aleatorio entre 0 y 50
    Serial.print("Entre 0 y 50: ");
    Serial.println(random(51));

    // Imprimimos números aleatorios entre 7 y 12
    Serial.print("Entre 7 y 12: ");
    Serial.println(random(7, 12));

    delay(1000);
}

```

## Estructuras de datos

### Arrays

Los arrays funcionan muy parecidos a otros lenguajes. La mayor diferencia es a la hora de mirar su longitud, debido a que no hay una función específica y hay que utilizar la función «sizeof» (<https://www.arduino.cc/reference/en/language/variables/utilities/sizeof/>) para ello. La longitud es igual al tamaño del array dividido entre el tamaño del tipo, es decir:

**length = sizeof(myArray)/sizeof([int/String/char/...])**

Cuidado, pues si se selecciona mal el tipo, la longitud saldrá mal y puede que no lo recorra entero, o intente recorrer de más.

```

int array1[5];
int array2[] = {0, 1, 2, 3, 4};
int array3[5] = {4, 3, 2, 1, 0};
char array4[12] = "hello world";

// Solo recoge hello
char array5[6] = "hello world";

void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    Serial.println(array4); // Hello World

    Serial.println(array2[3]); // 2;

    array2[3] = 13;
    Serial.println(array2[3]); // 13;

    int aux = array2[3];
    Serial.println(aux); // 13;
}

void loop() {
    for(int i = 0, len = sizeof(array4)/sizeof(char); i < len; i++){
        Serial.print(array4[i]);
    }
    Serial.println();
}

```

```
Serial.println("-----");  
  
delay(1000);  
}
```

## Errores comunes

avrdude : stk500\_getsync() attempt 10 of 10: not in sync: resp=0x00

Puede ser que esté seleccionada la placa incorrecta.

Posible solución: Herramientas -> Placa -> Seleccionar la placa correcta