

# 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.

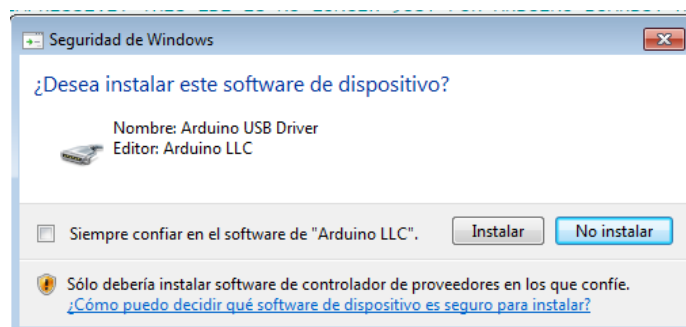
Se recomienda también el uso del entorno de emulación <https://www.tinkercad.com> para realizar diferentes pruebas o probar los ejercicios en casa si no se dispone de un Arduino y así en clase solamente tener que probar lo realizado en casa y grabar el vídeo. Funciona con cuenta Autodesk. Hay que hacerse 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

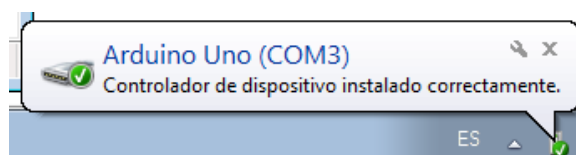
En los ordenadores del laboratorio, los drivers de Arduino y su IDE ya deberían de estar. Para comprobar si todo está correctamente funcionando, enchufad el Arduino al puerto USB del ordenador y tratad de subir a él un programa vacío desde la placa.

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

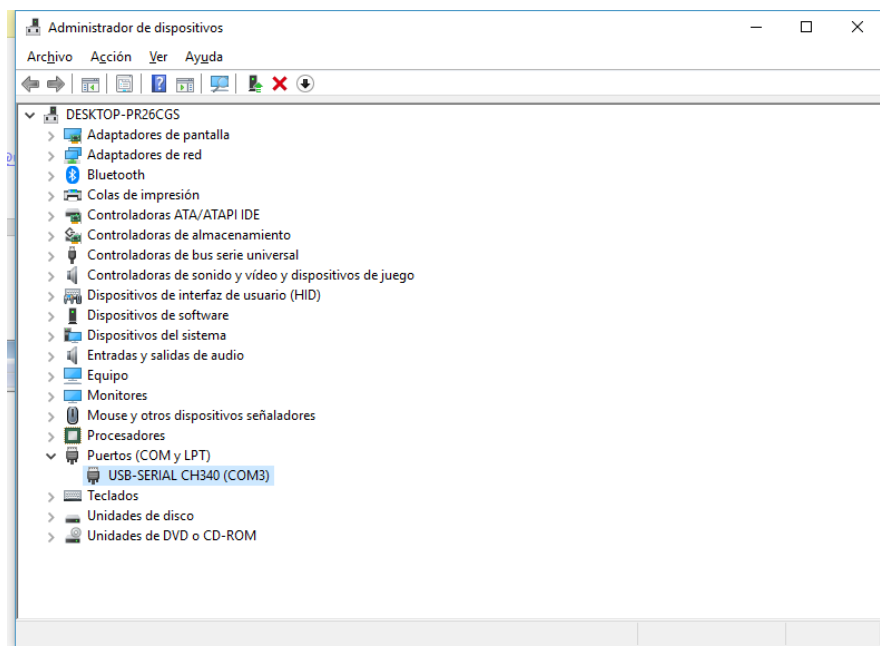
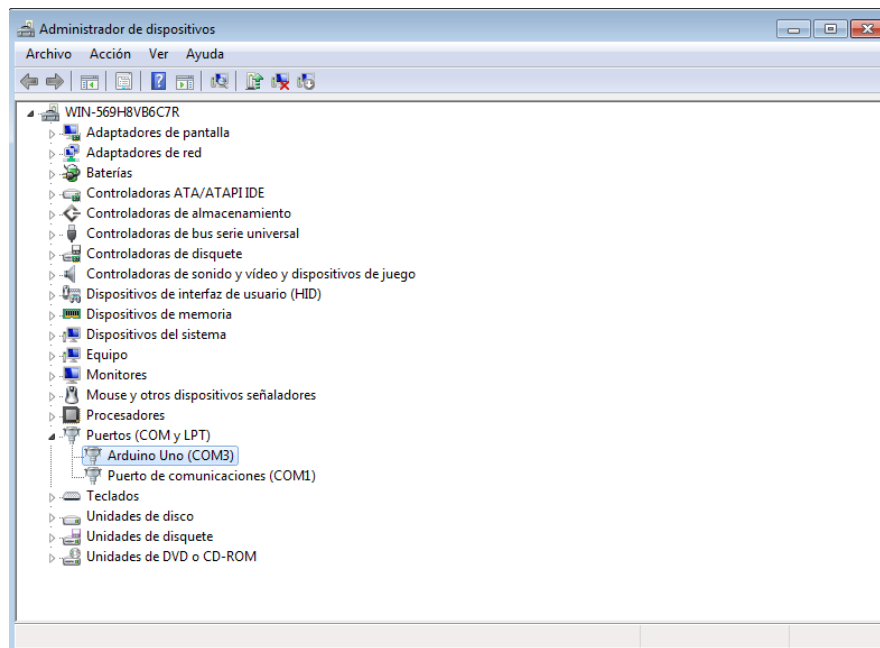
1. Descargar y ejecutar el Arduino IDE de su página oficial, siempre teniendo en cuenta nuestro sistema operativo (Windows 7, Windows 10, Linux, ...): <https://www.arduino.cc/en/Main/Software>. La última versión es la 1.8.19, o la 2 (liberada el 14/09/2022).
2. Durante el proceso, puede que nos pregunte 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.



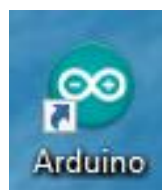
3. Conectamos la placa Arduino al equipo por medio del puerto USB, el cuál debería reconocerlo correctamente.



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.

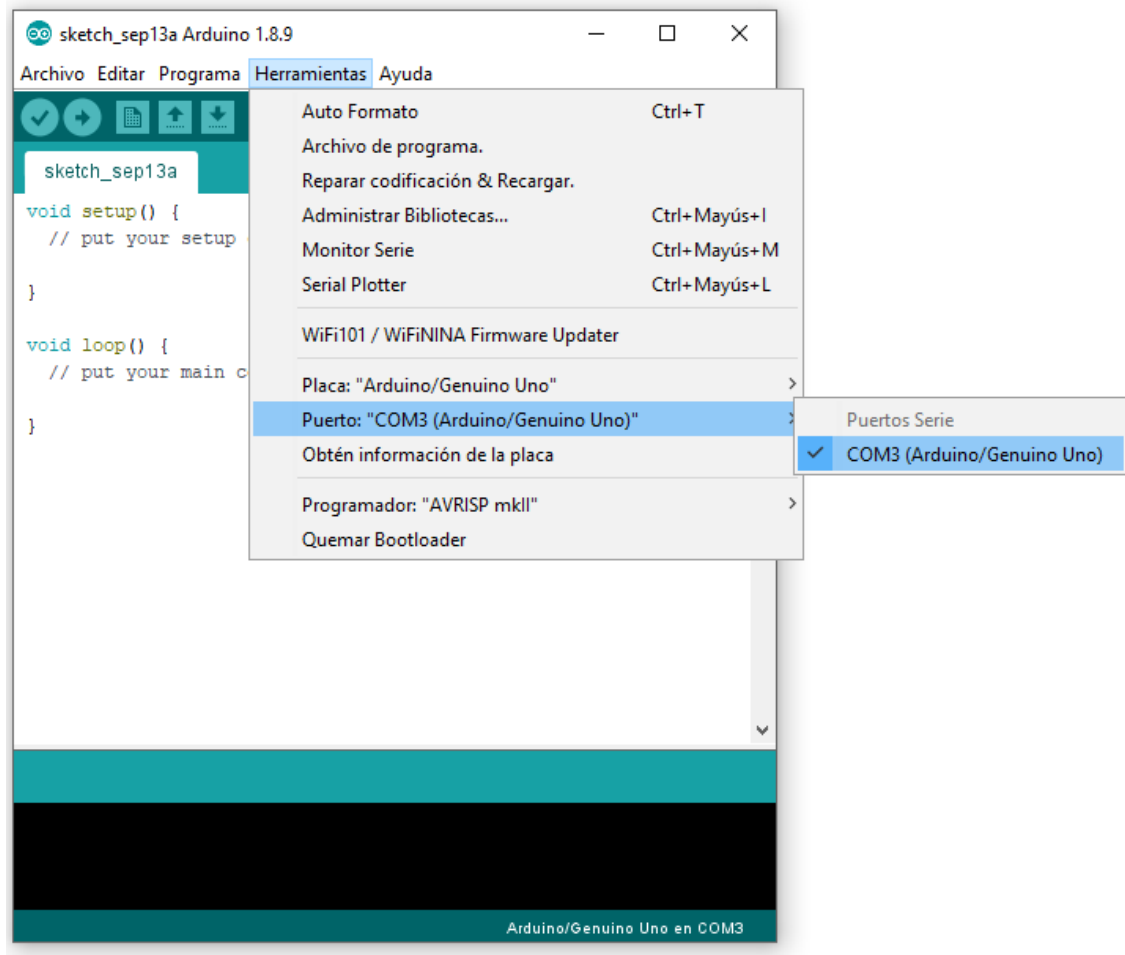


Ilustración 2 Selección de puerto COM

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*

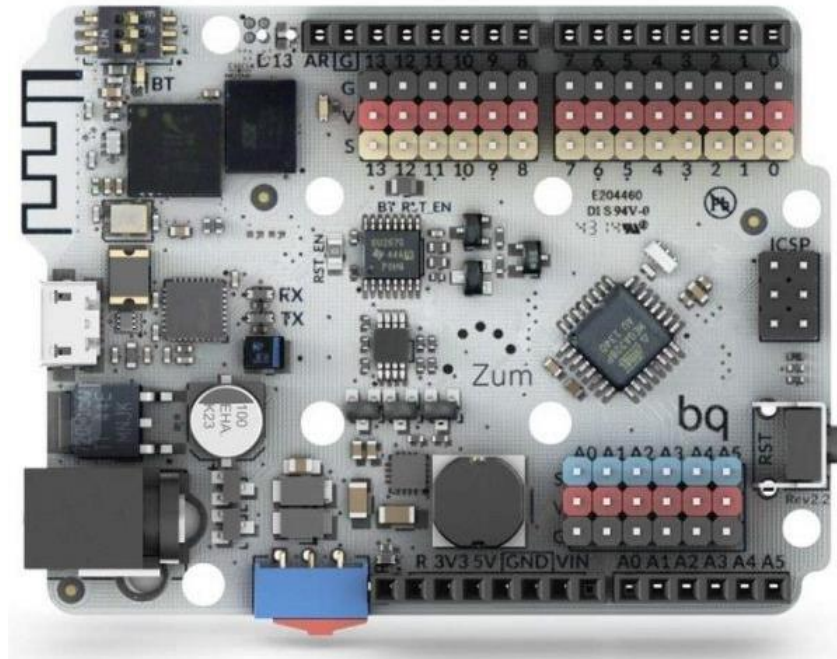


Ilustración 5 Placa BQ Zum

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

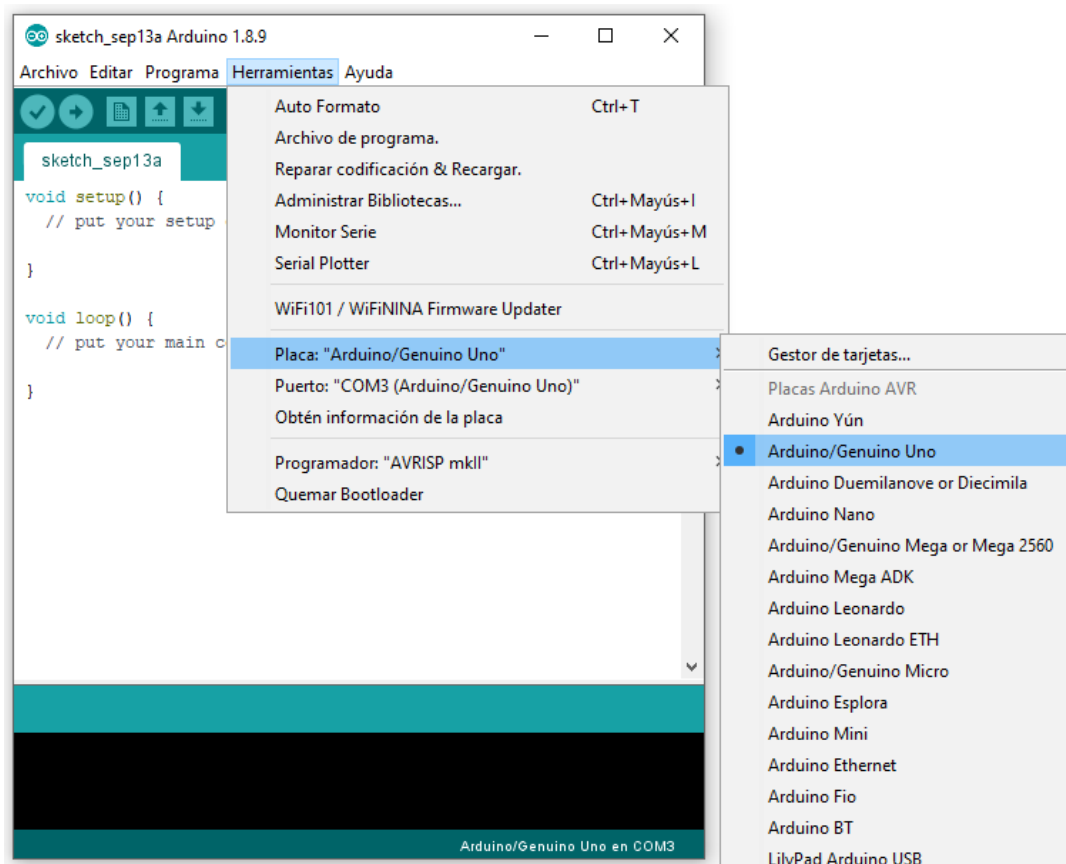
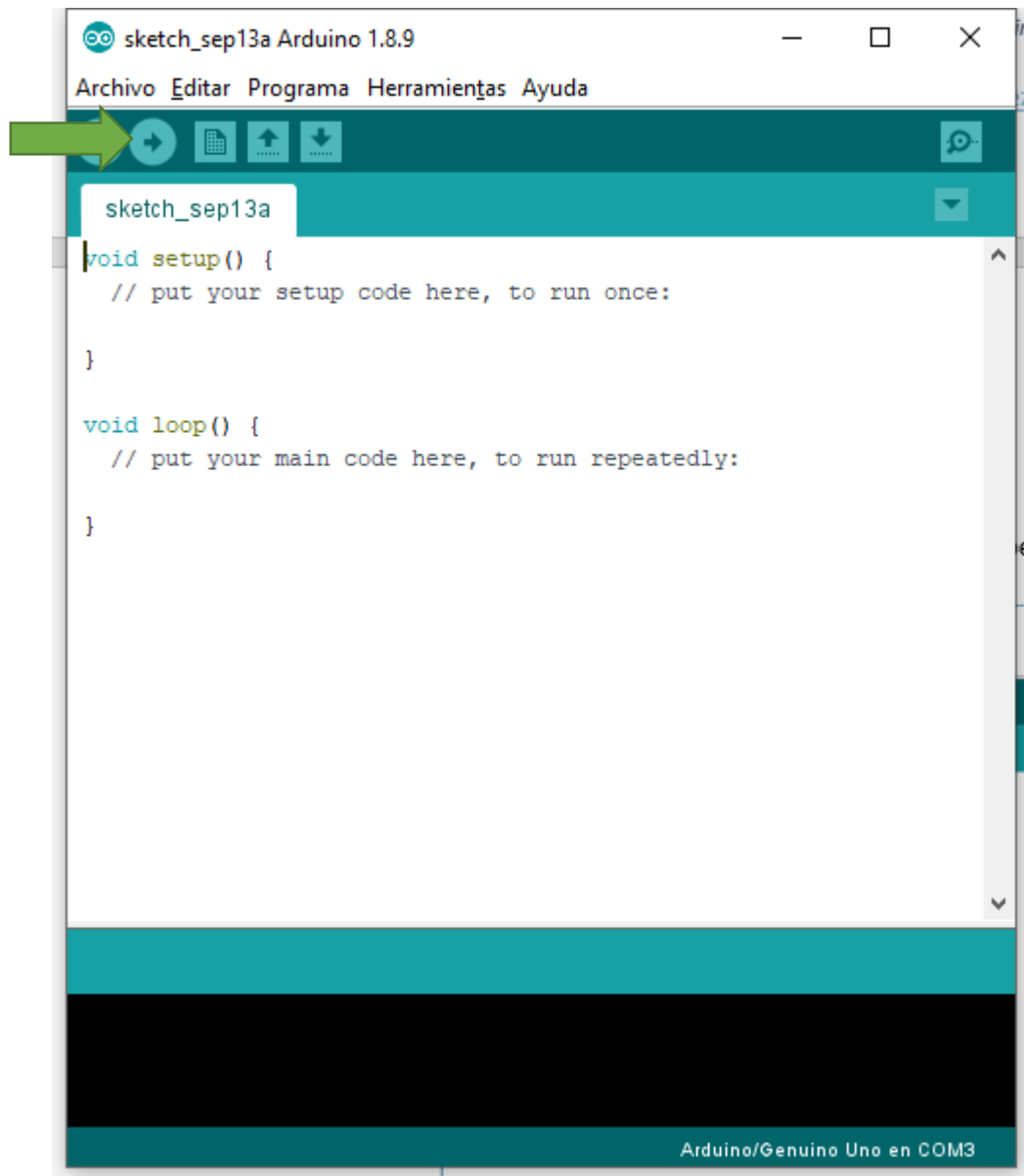


Ilustración 6 Selección de microcontrolador Arduino



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



*Ilustración 7 Subir programa*

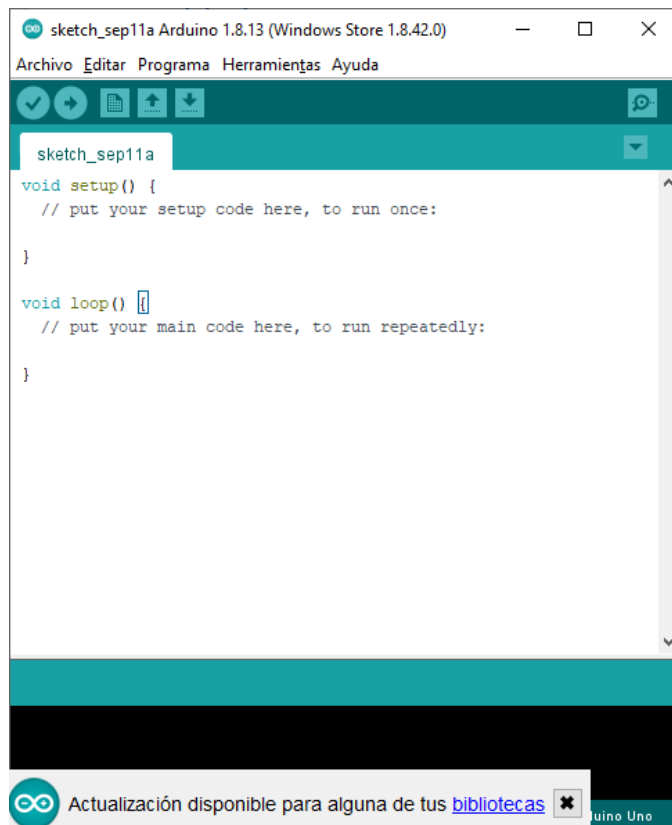
**NOTA:**

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.**

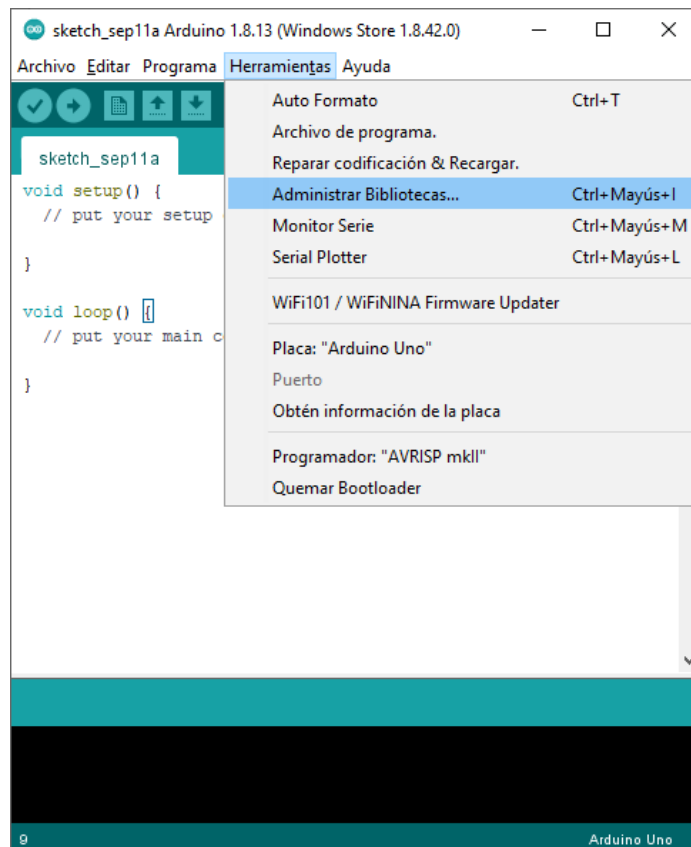


Ilustración 9 Administrar bibliotecas

## 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.

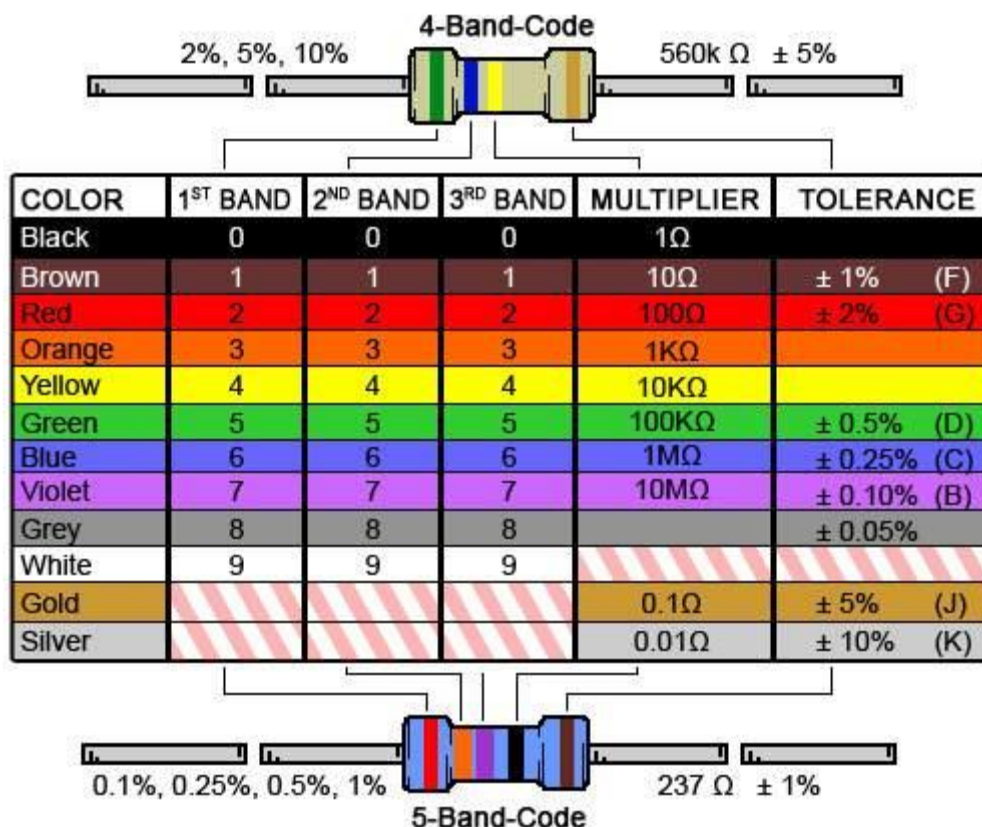


La segunda resistencia de la izquierda es de  $10k\Omega$ .

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.

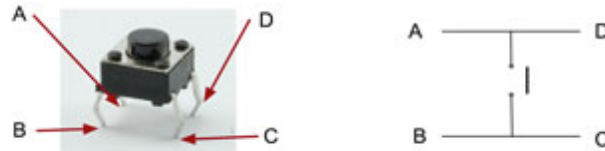
Por este motivo, siempre necesita una resistencia el circuito que lleve un interruptor, salvo que los elementos utilizados cuenten con una o hagan suficiente resistencia al paso de la corriente. Esto es debido a que el pulsador actúa, cuando se pulsa, como si fuera un cable. Luego, conecta positivo y 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.

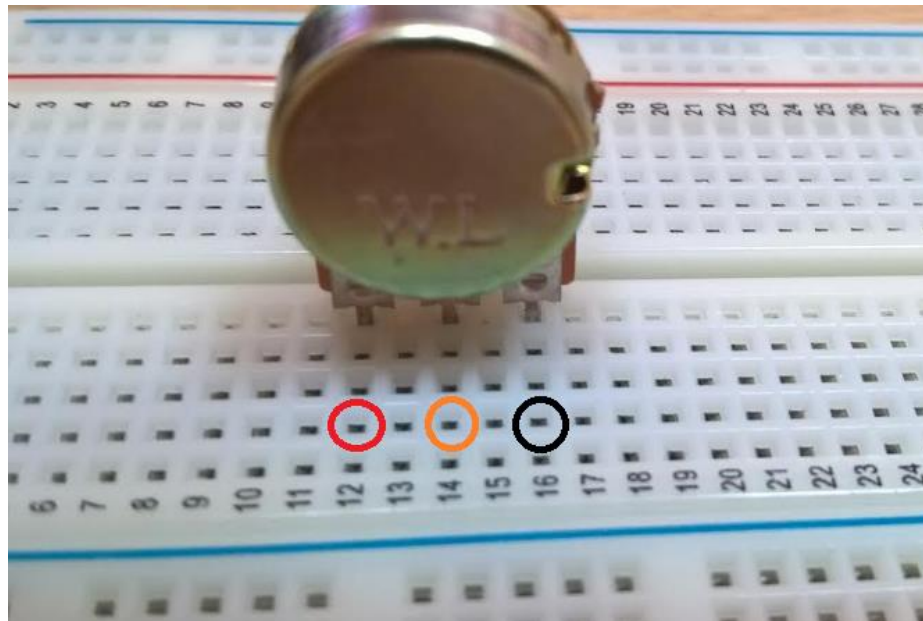


## Potenciómetro



**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.

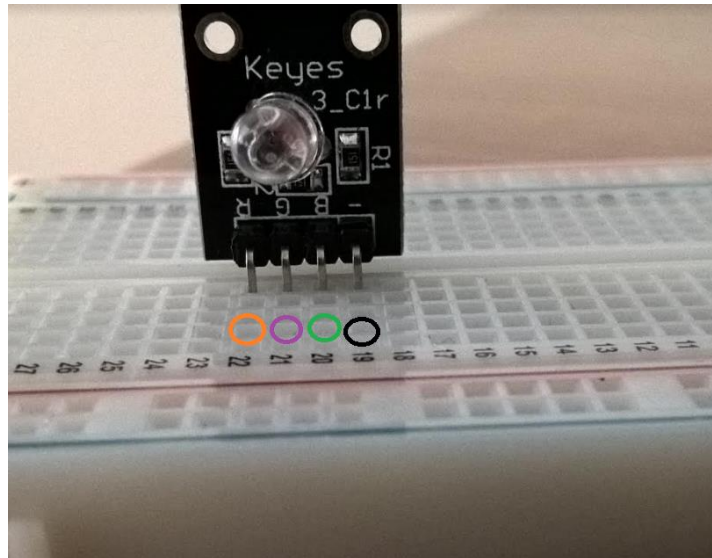
El LED tiene 4 pines:

- **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 ~)**.

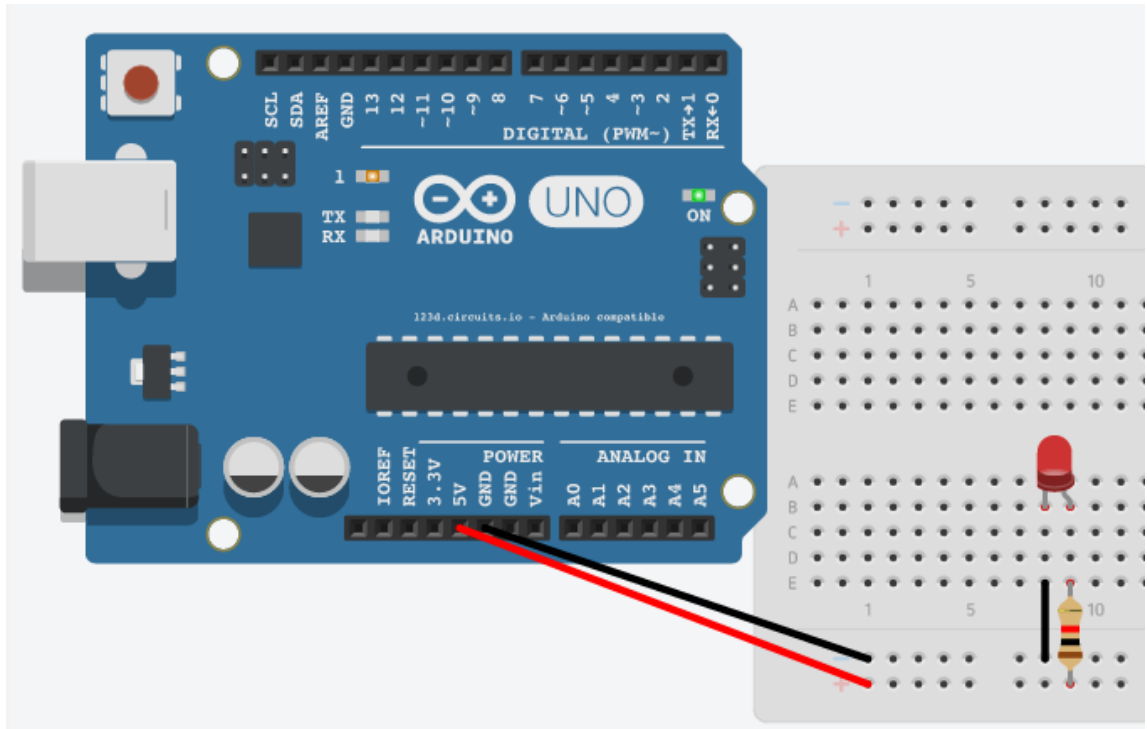
Si por ejemplo, solo queremos utilizar el LED como rojo y verde, basta con conectar los pines digitales a la patilla R y G, y el GND a GND, pudiendo dejar la B sin conexión.

## 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:

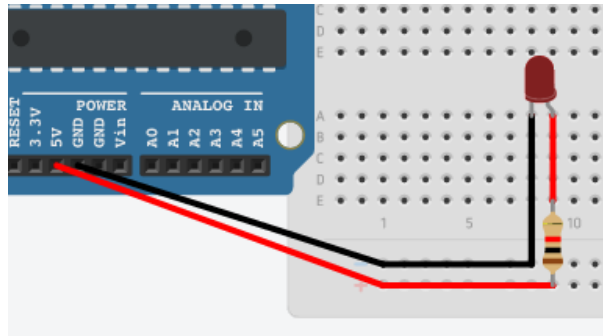
1. Conectamos el pin **5V** al carril positivo de la ProtoBoard (Rojo).
2. Conectamos el pin **GND** al carril negativo del ProtoBoard (Azul).
3. Colocamos el Led rojo en la ProtoBoard. Cuidado con la polaridad del Led.
4. Utilizamos la resistencia para abrir un carril positivo que se conecte con la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

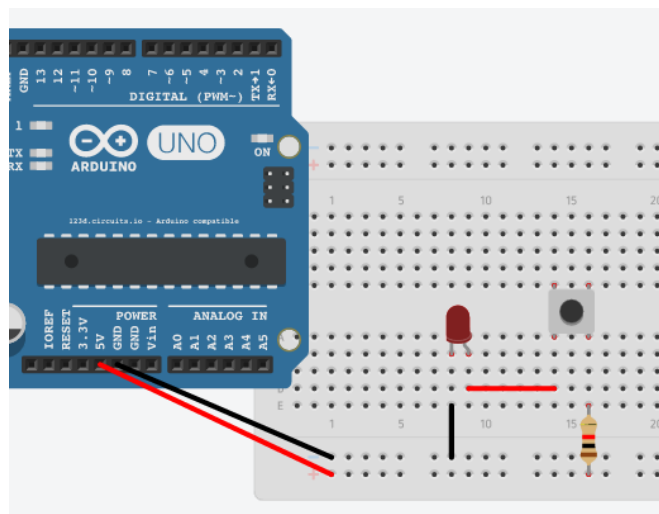
Sería algo similar a lo siguiente, con un cable en la zona de polaridad y un cable en cada carril.



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

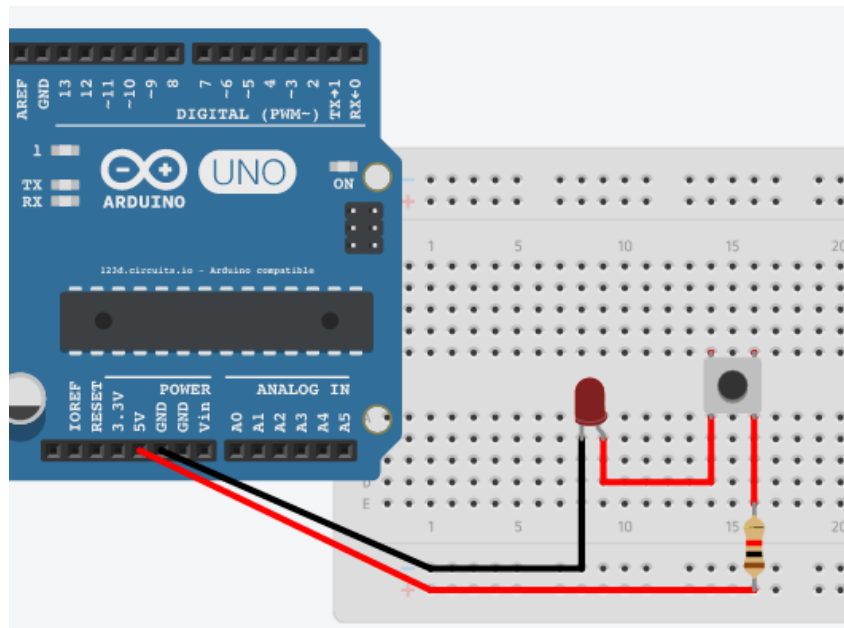
Incluir un pulsador para cortar / abrir el circuito

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.



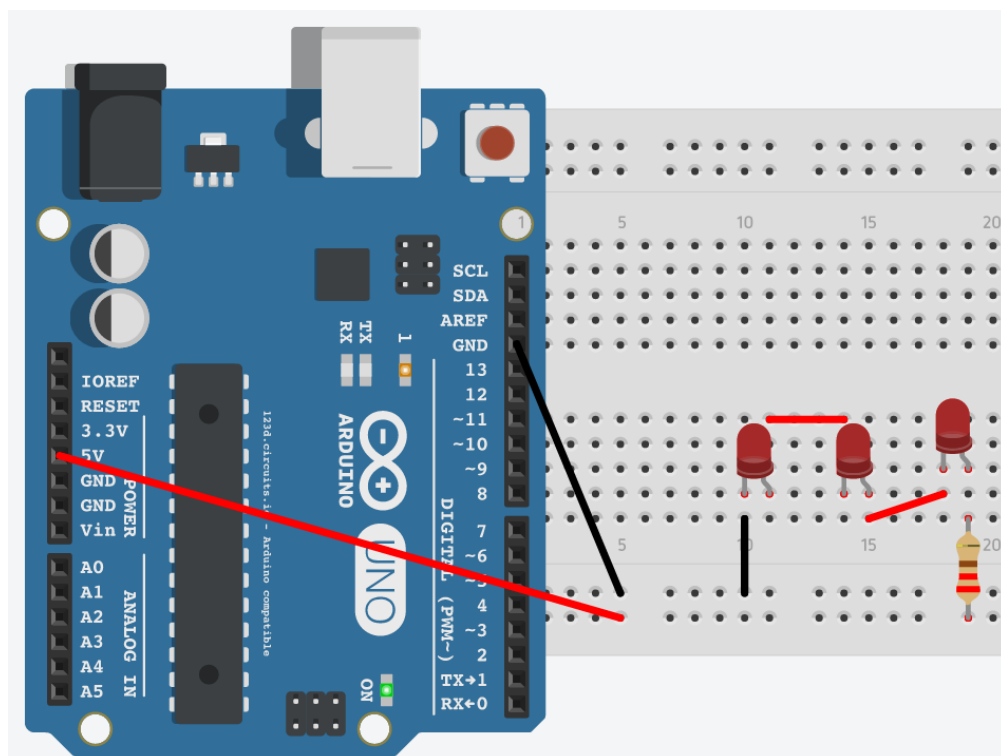
¿Cómo sería este circuito si no estuviésemos utilizando la ProtoBoard?

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.

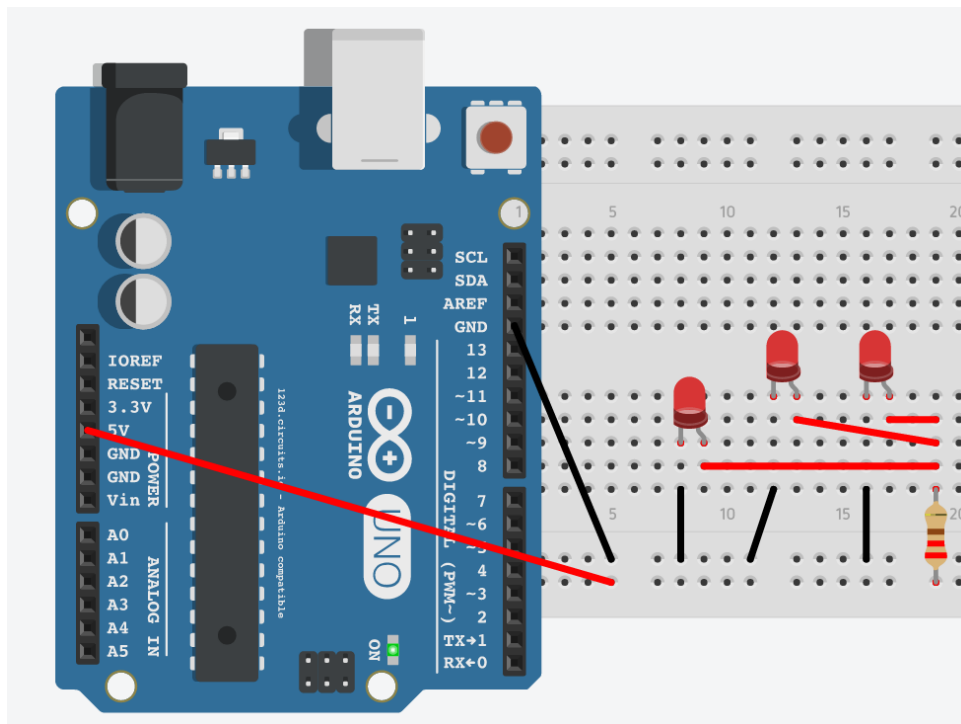


Como comentamos anteriormente, un Led hace caer el voltaje, luego, si colocamos varios leds en serie las caídas de voltaje se suman. **Esta mayor caída de voltaje unida a la resistencia hace que los leds apenas se iluminen o ni se iluminen.**

- 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.

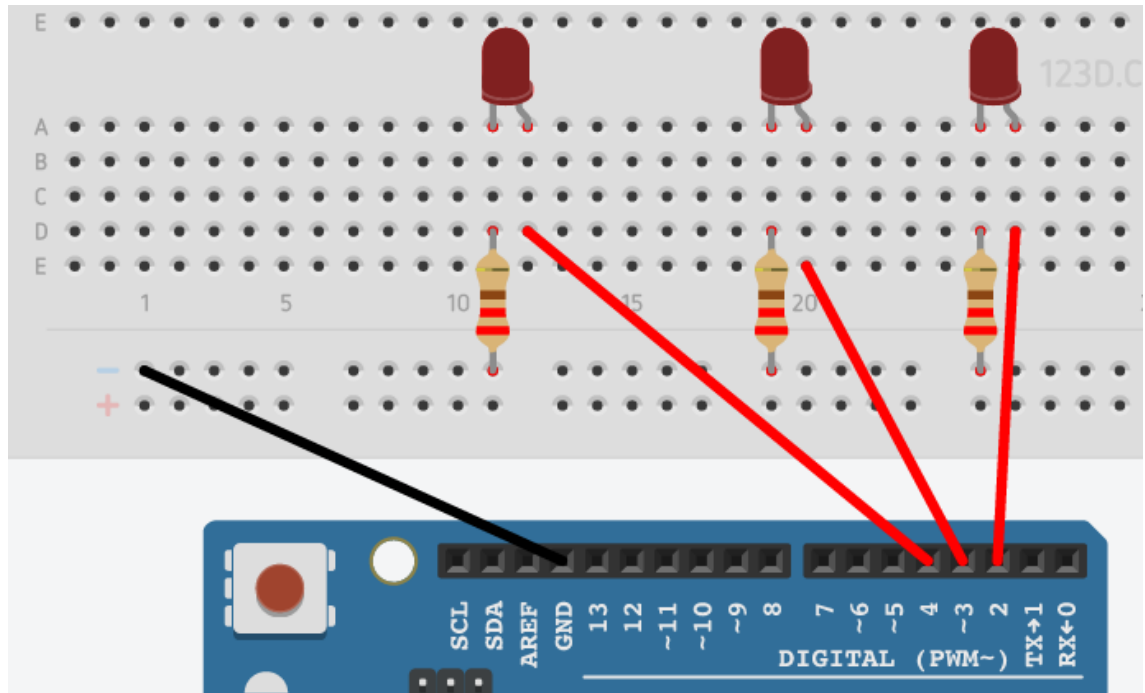
La placa Arduino se utilizará para conectar varios elementos (sensores y actuadores), formando cada uno de ellos un pequeño circuito muy simple. La programación de las entradas y las salidas será la parte encargada de coordinar los elementos electrónicos.



### 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);

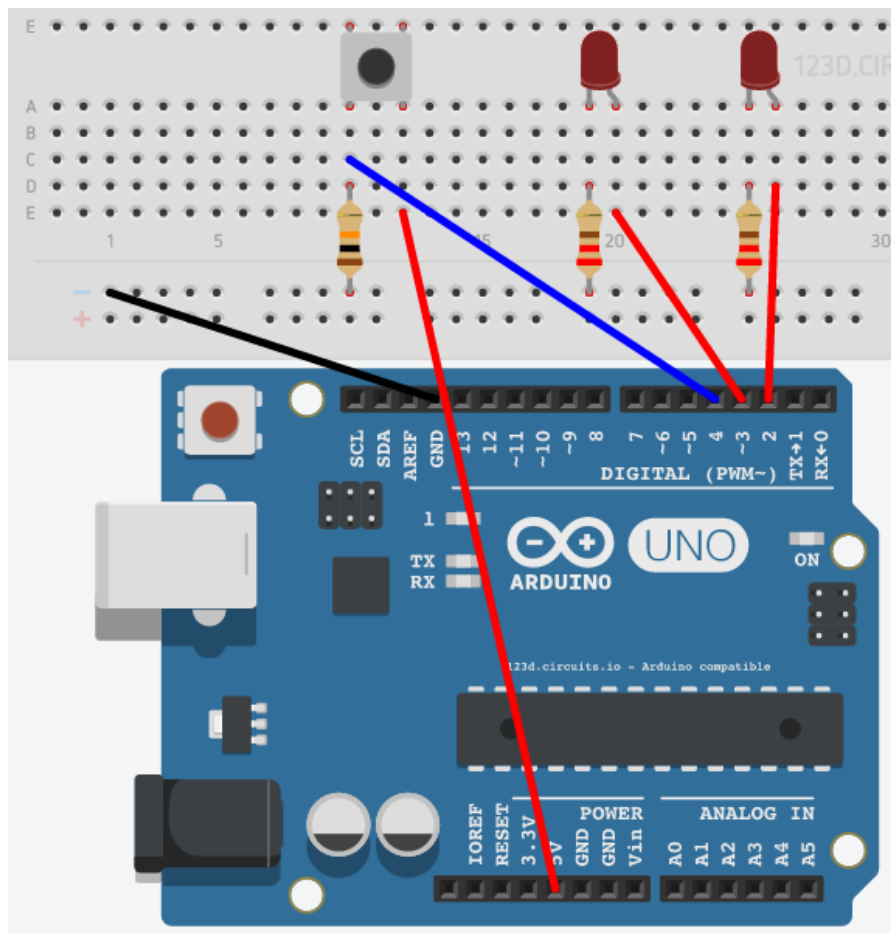
  digitalWrite(led3, LOW); // Voltaje a - LOW para el led 3.
}

```

### 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).

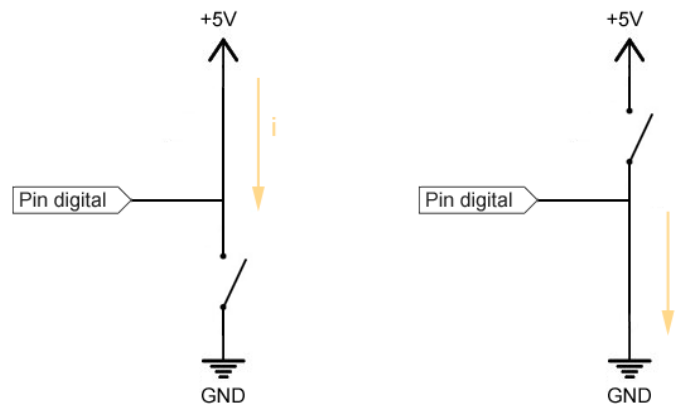
Los tres circuitos tienen conectado un pin de Arduino, que sirven, en el primer caso, para saber si el circuito del botón está abierto/cerrado, y en los otros dos casos para abrir/cerrar la alimentación eléctrica de los leds.

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.**



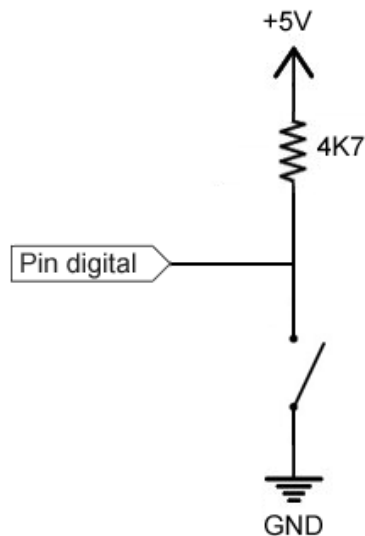
**Nunca debemos conectar directamente 5V a GND o provocaremos un cortocircuito, hay que incluir una resistencia de por medio siempre y cuando los elementos que utilizemos no tengan resistencias incorporadas o no hagan apenas resistencia a la electricidad, como ocurre con el pulsador y el LED.**

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.

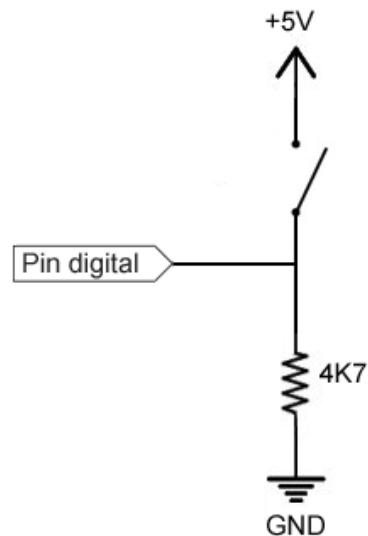
El pin digital que tiene como misión leer el voltaje de esa conexión tampoco funcionaría correctamente, ya que estaría conectado simultáneamente a: 5V y GND (0V).

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.**



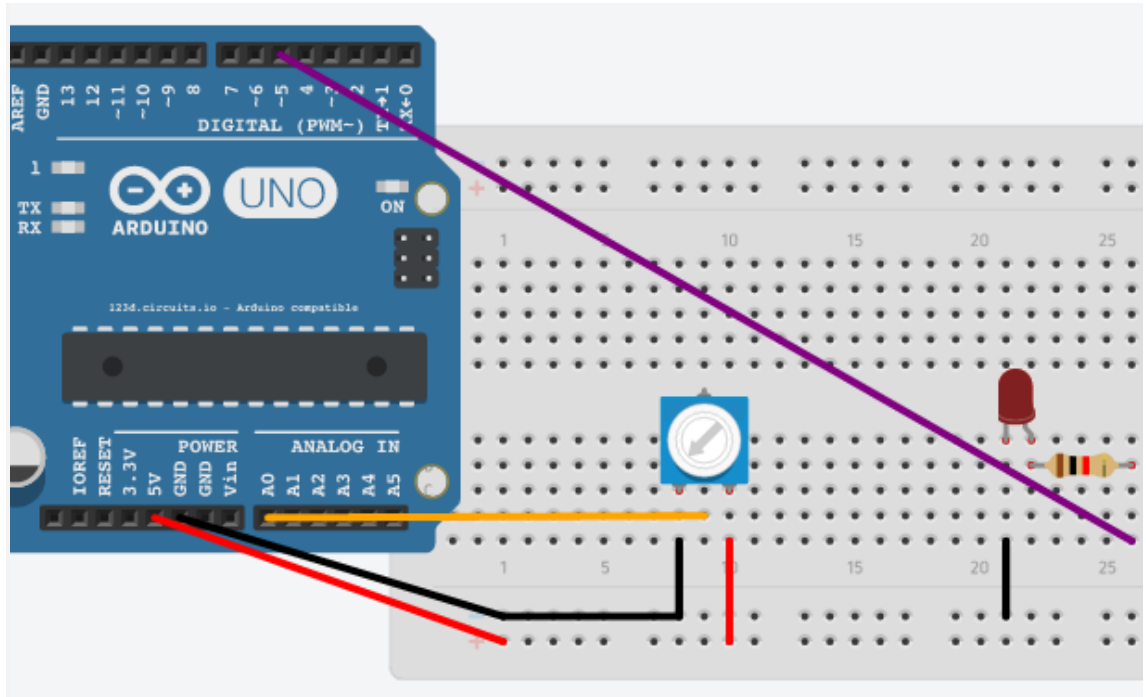
El método `println()` no realiza **conversiones automáticas de tipos de datos**. Si queremos incluir un valor no String en la cadena debemos convertirlo previamente. Ejemplo:

```
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 pines 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);
```

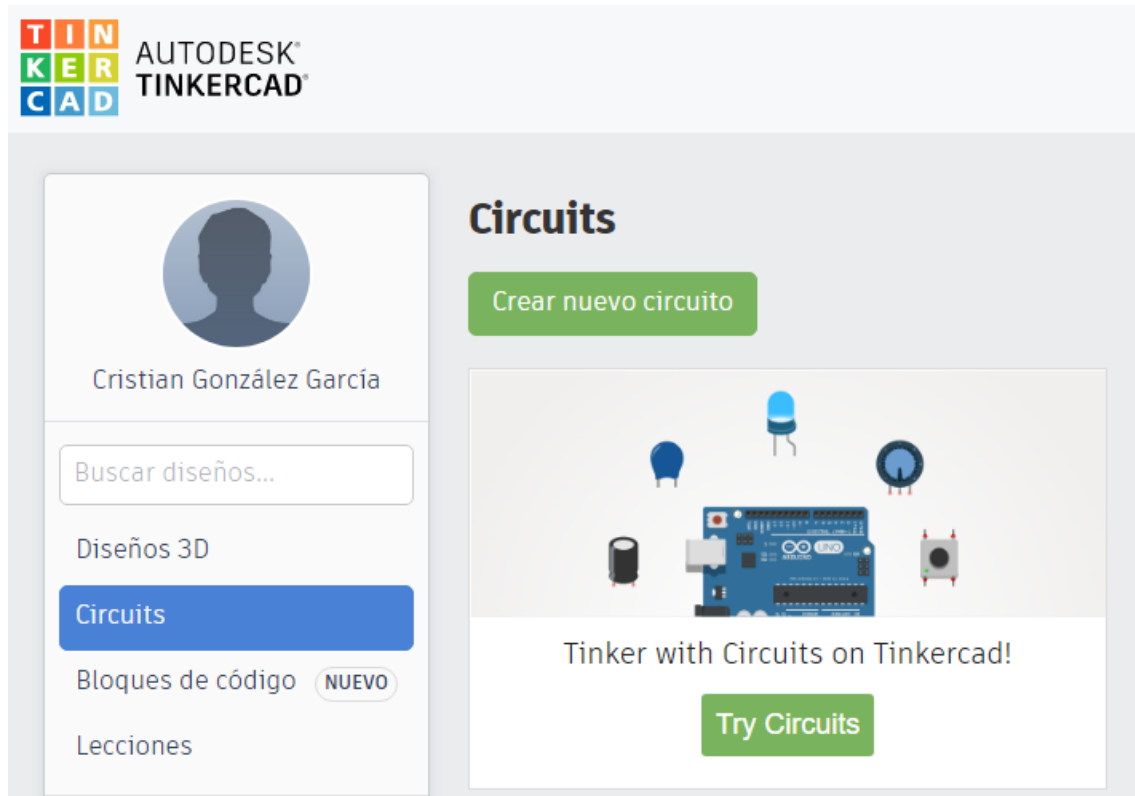
7. Código de ejemplo:

```
int valuePotenciometer;  
int valueAnalogPin;  
  
void setup() {  
  Serial.begin(9600); // Iniciar el Serial  
  Serial.println("Setup");  
  
  // Los pines 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);  
}
```

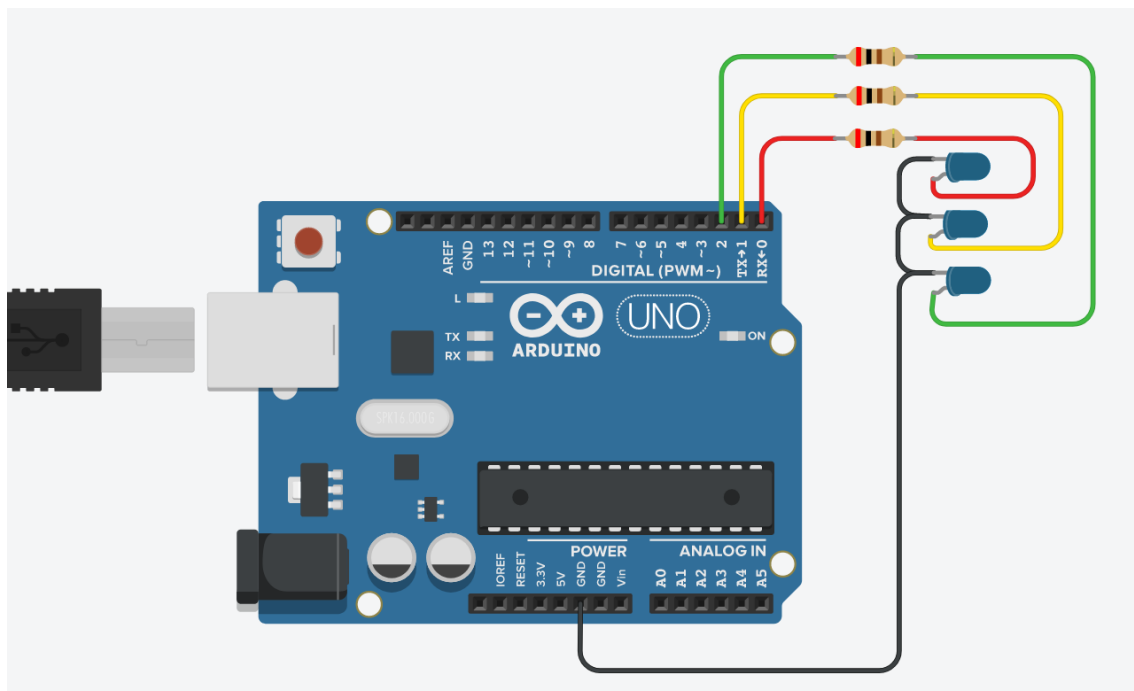
Ampliación:

## Emulador de Arduino: Tinkercad - Circuits

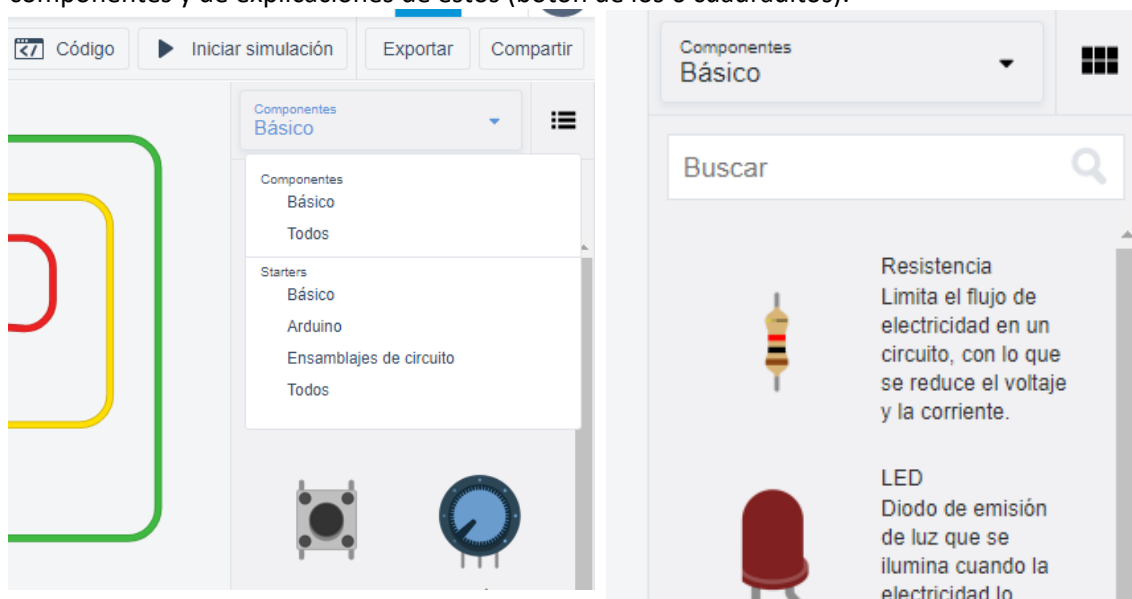
Es una plataforma web que permite crear circuitos utilizando Arduino. El emulador soporta muchos sensores y actuadores diferentes. Además, permite simular el circuito para comprobar si funciona o no. Por ejemplo, si usamos LED, estos se encenderán o no según lo que hayamos programado y según hayamos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



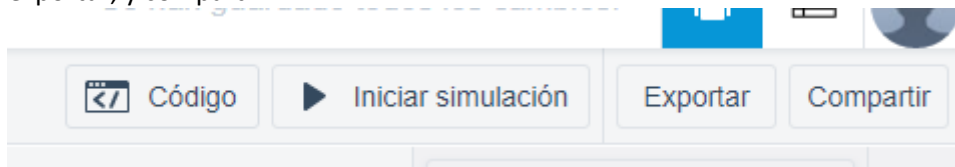
Esta plataforma está muy bien para probar nuestros circuitos en caso de no disponer una placa Arduino, o si no sabemos si funcionará bien y pueda haber peligro de estropear la placa. Además, debido a que es un simulador, puede ser más fácil en el caso de que haya muchos cables. También se pueden compartir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



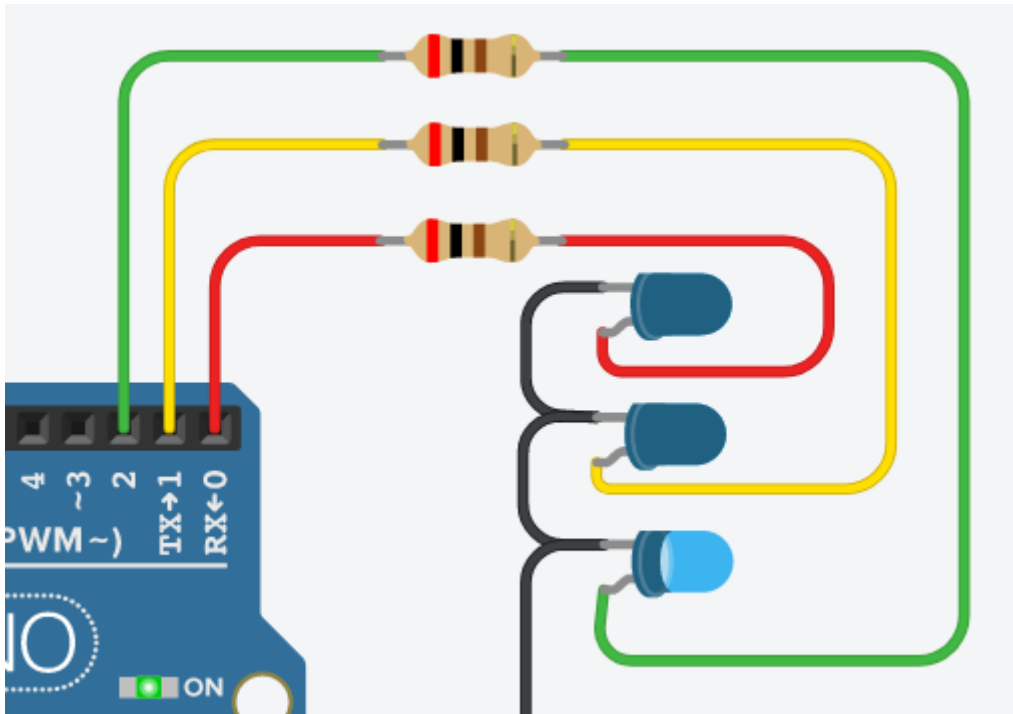
El emulador es de tipo «Drag and Drop» y permite cambiar las opciones del elemento al hacer clic sobre él (colores, tipo de resistencia, etc.). Dispone de varias opciones en las que podemos ver los componentes básicos o todos los existentes en el programa. Dispone de buscadores de componentes y de explicaciones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



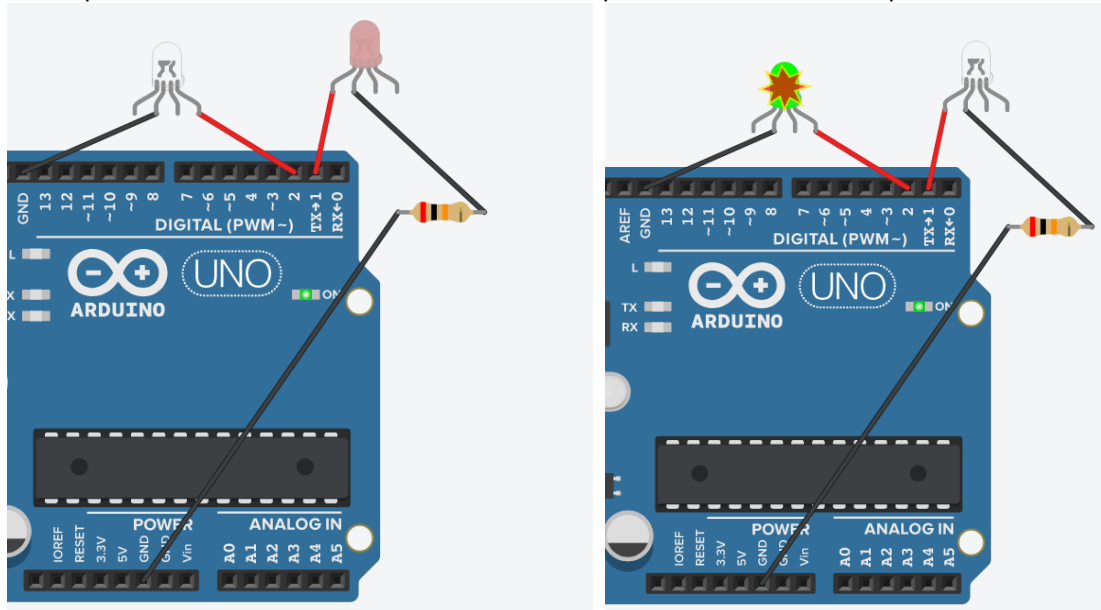
Arriba a la derecha de la pantalla, podéis ver el menú de opciones: código, Iniciar simulación, exportar, y compartir.



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

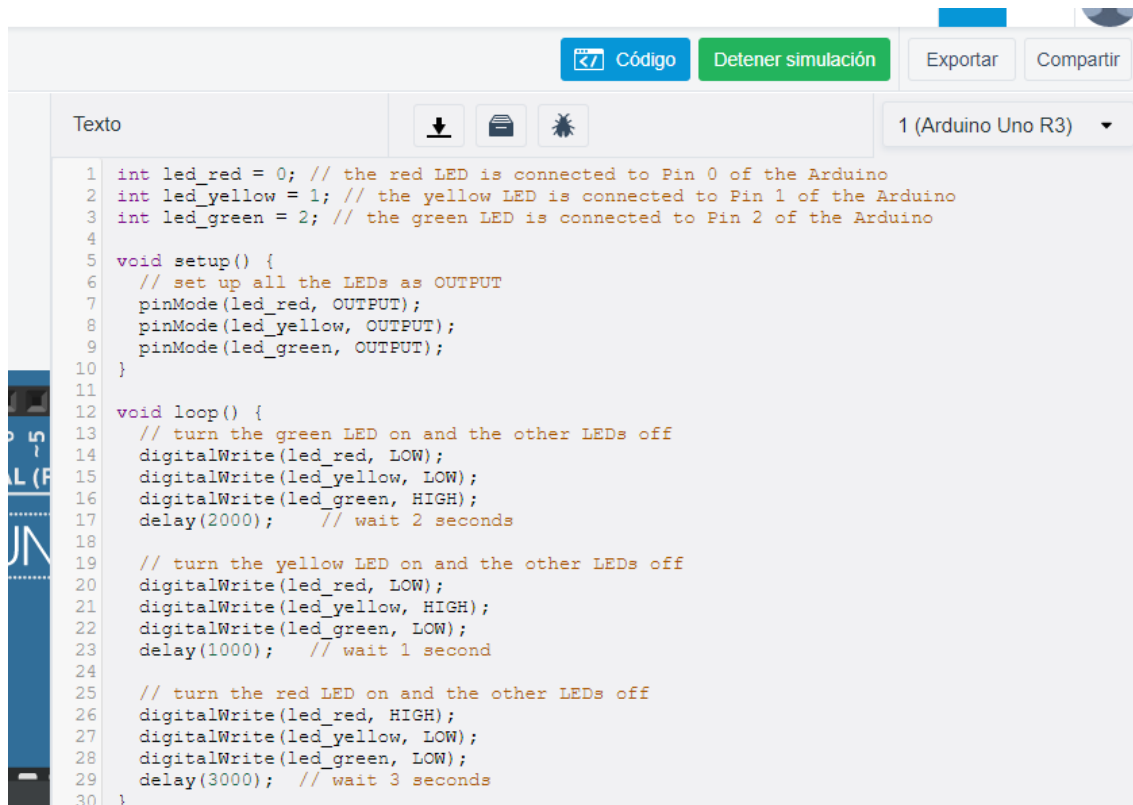


En cambio, por ejemplo, el circuito montado estuviera mal, este nos lo indicaría. En la Ilustración 12 vemos, como al faltar una resistencia, el LED tiene una imagen imitando la explosión. No obstante, no con todo ocurre este aviso, por ejemplo, con un pulsador mal conectado no avisa, ni tampoco si se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

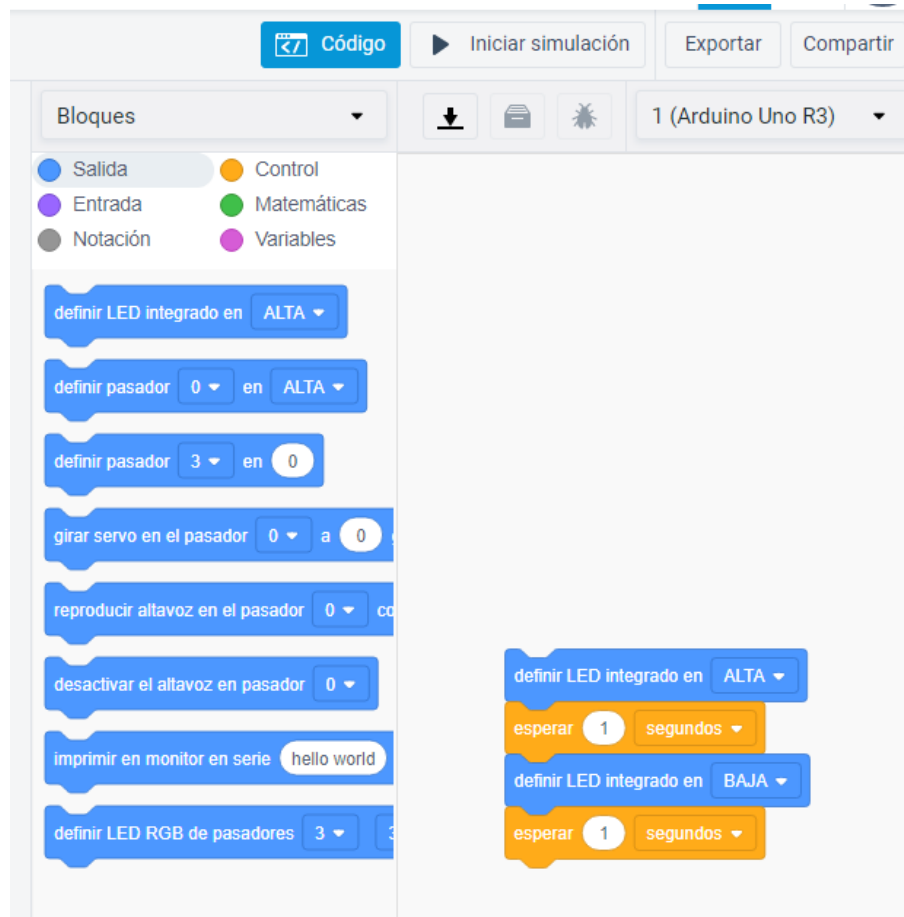


```
1 int led_red = 0; // the red LED is connected to Pin 0 of the Arduino
2 int led_yellow = 1; // the yellow LED is connected to Pin 1 of the Arduino
3 int led_green = 2; // the green LED is connected to Pin 2 of the Arduino
4
5 void setup() {
6   // set up all the LEDs as OUTPUT
7   pinMode(led_red, 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 }
```

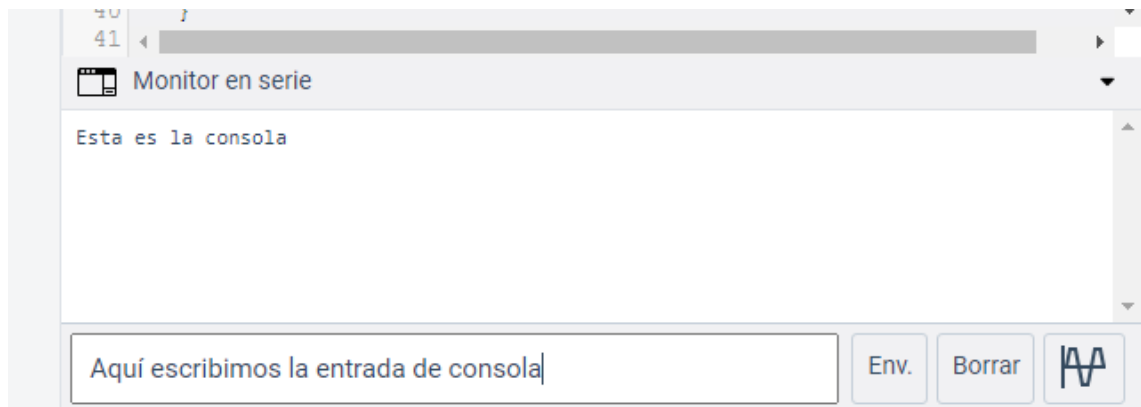
También dispone de una opción de crear el código fuente utilizando piezas de puzle. Este modo sirve para que la gente que no tiene conocimientos de programación pueda aprender y realizar cosas de una manera mucho más sencilla.

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