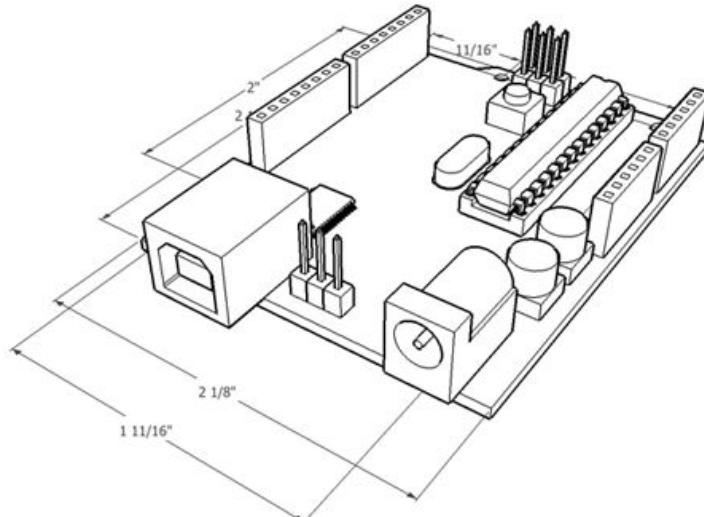


TALLER: Introducción a la Robótica libre con ARDUINO

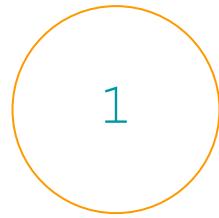


Innova Pilas 2019
#InnovaPilas19
<http://innova.pilas.es/es/>

José Pujol
@jo_pujol
<https://tecnopujol.wordpress.com/>

Índice

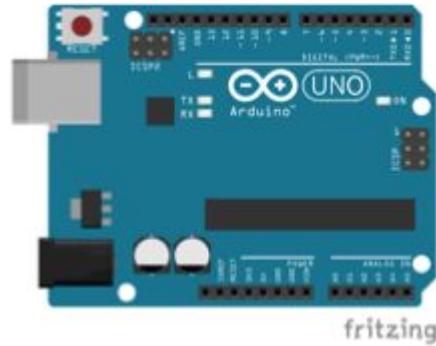
1. INTRODUCCIÓN A ARDUINO
2. CONCEPTOS BÁSICOS
3. PRÁCTICAS CON ARDUINO
 - P1: Hola Mundo
 - P2: Salidas Digitales
 - P3: Entradas Digitales
 - P4: Entradas Analógicas. Puerto Serie
 - P5: Salidas Analógicas



Introducción a Arduino

¿Qué es Arduino?

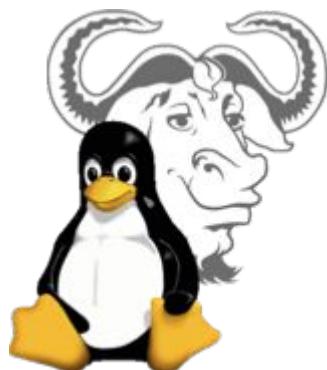
- Plataforma de prototipado de electrónica abierta
- Se basa en software y hardware libre
- Es flexible y de fácil uso



Software libre

Aquel software que nos da **libertad** para:

1. Ejecutarlo con cualquier propósito
2. Estudiar cómo está hecho y modificarlo
3. Redistribuir copias
4. Distribuir copias modificadas à Licencia GPL



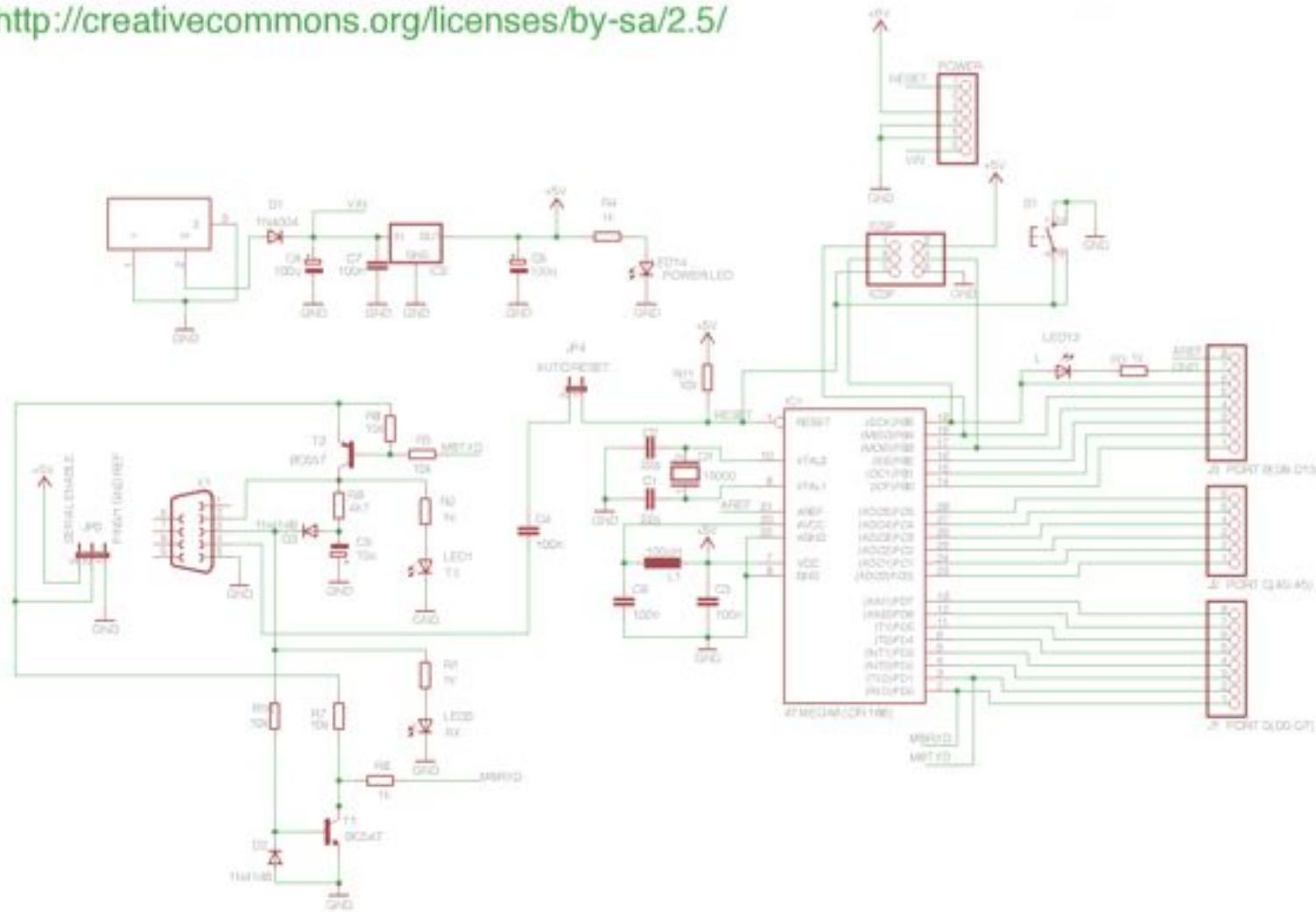
GNU Linux



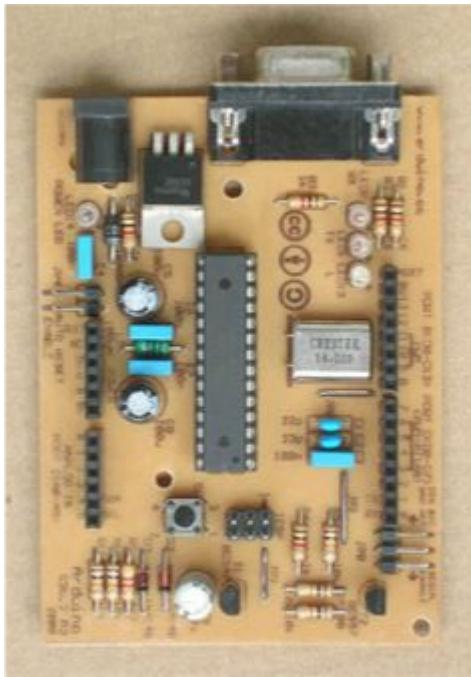
Richard Stallman

Open Source Hardware

Released under the Creative Commons Attribution Share-Alike 2.5 License
<http://creativecommons.org/licenses/by-sa/2.5/>



Open Source Hardware



Ecosistema de Arduino

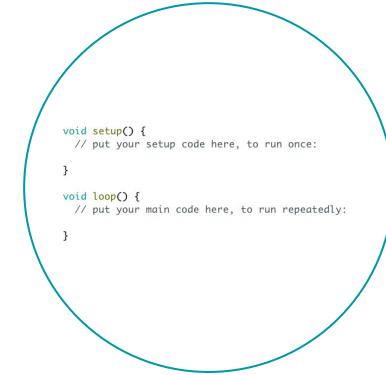
Hardware



Entorno de desarrollo



Lenguaje Programación



Soporte online



Hardware



Placa microcontrolada de entradas y salidas

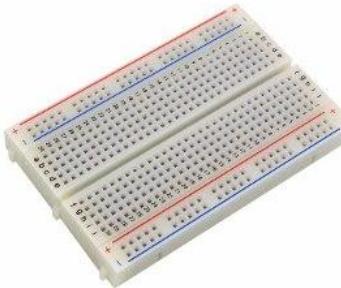
Arduino UNO es el modelo más popular

Componentes y módulos

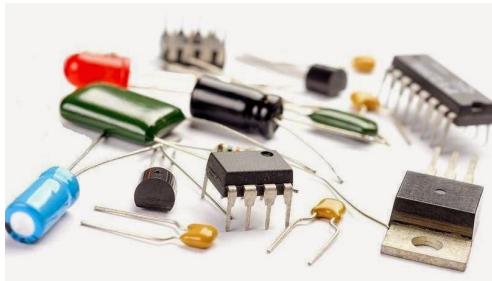
Componentes de electrónica discreta y módulos en placas presoldadas que conectan los pines.
Ejemplo: pantalla LCD y módulo Bluetooth.



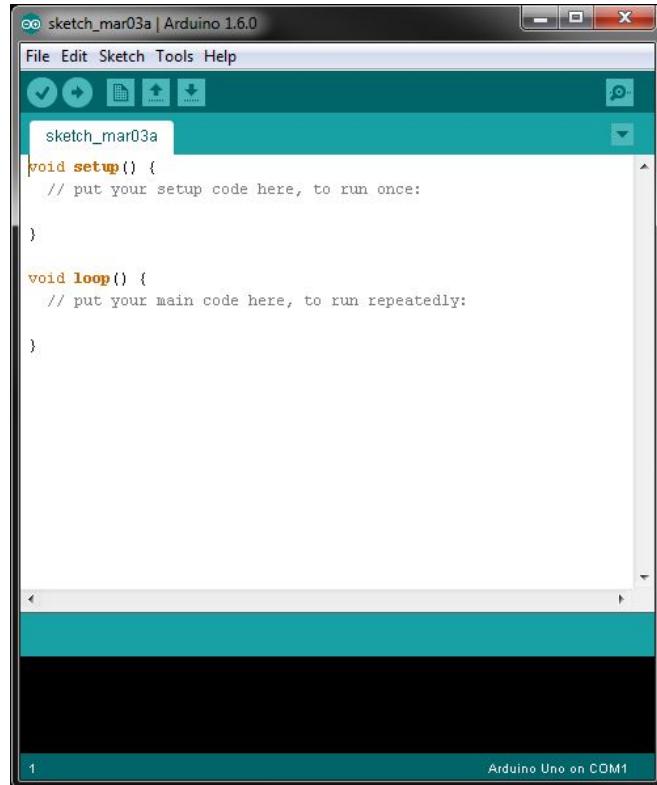
Componentes



Módulos



IDE: Entorno de Desarrollo



- Es de software libre
- Proviene del entorno de Processing

Funcionalidades:

- Escribir código
- Depurarlo
- Compilarlo y cargarlo en la placa
- Comunicarnos con la placa

Software

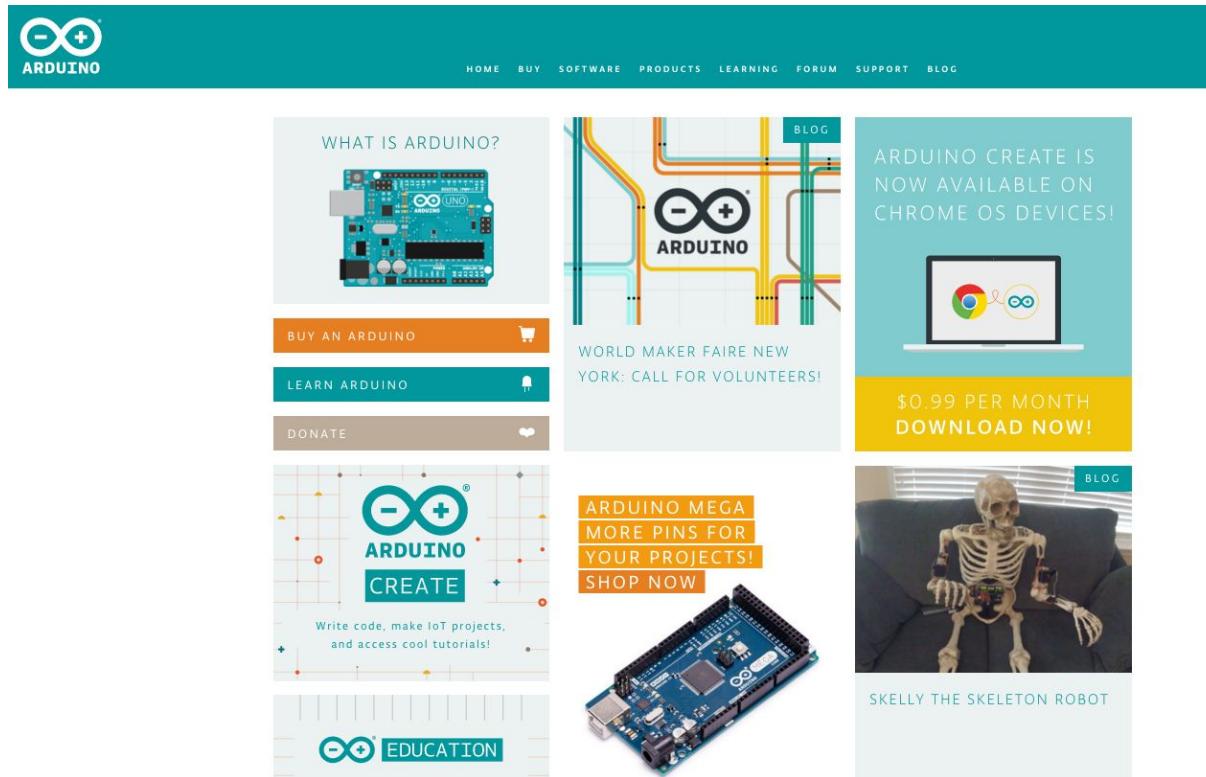
- Lenguaje con el que se programan las placas de entradas y salidas.
- Basado en **C++**, programación orientada a objetos con dos estructuras fundamentales: **setup** y **loop**.
- Gran cantidad de **librerías** que aumentan las funcionalidades y permiten controlar los dispositivos de forma sencilla.

```
// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
}

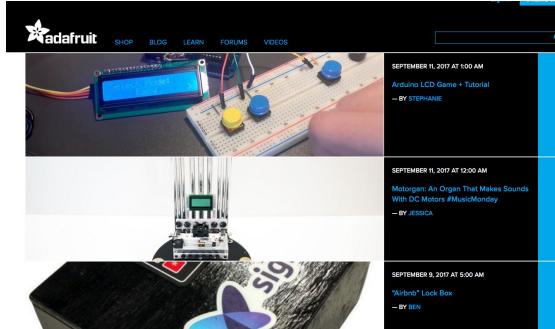
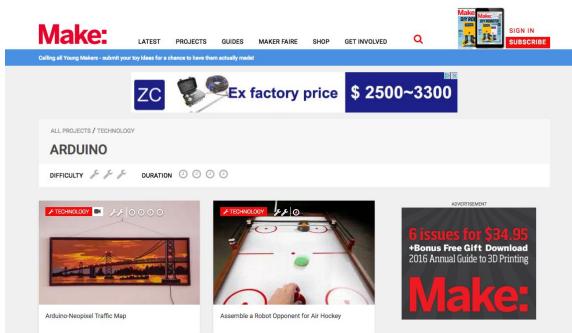
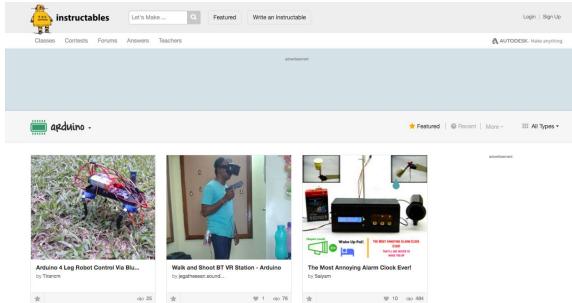
// the loop function runs over and over again forever
void loop() {
    digitalWrite(13, HIGH);    // turn the LED on (HIGH is the voltage level)
    delay(1000);              // wait for a second
    digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
    delay(1000);              // wait for a second
}
```

Soporte en red

La Web de Arduino nos muestra gran cantidad de información:
lenguaje de programación, proyectos, guías, hardware...



Soporte en red



Especial atención a **blogs y canales de Youtube** de usuarios de Arduino. Ejemplo de webs interesantes:

Instructables

Make

Adafruit

Intro Arduino: Reflexiones

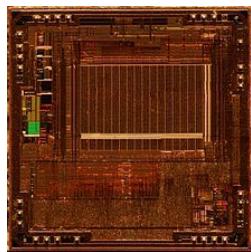
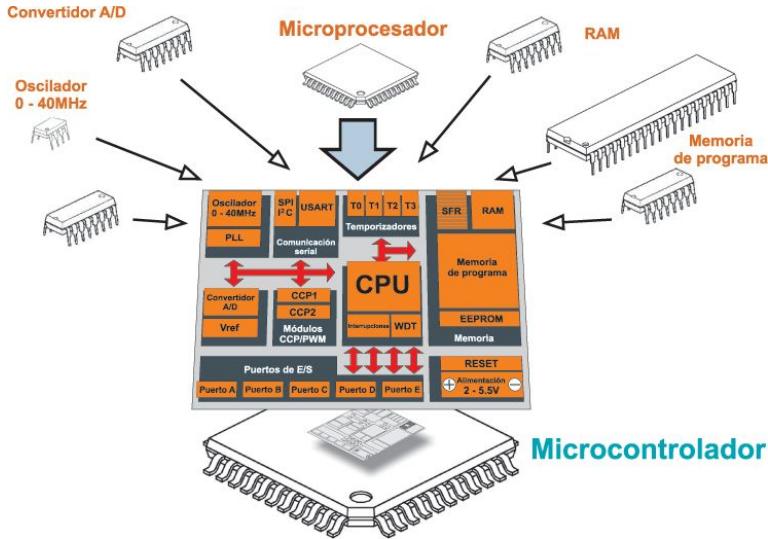
1. Pon ejemplos de Software libre
2. Que significa Hardware Open Source
3. ¿Cuales son las ventajas de usar estos sistemas?
4. Realiza una búsqueda en Google y en Youtube con la palabra Arduino ¿cuántos resultados se registran?



2

Conceptos básicos sobre Arduino

¿Qué es un microcontrolador?



- Circuito integrado y programable.
- Posee todos los componentes de un ordenador: CPU, memoria, almacenamiento y dispositivos de entrada y salida.
- Arduino utiliza un ATmega 328, que cuenta con 300 millones de transistores.

¿Por qué son importantes?



- Económico
- Flexibilidad: facilidad de programación
- Velocidad de operación

Ha dado lugar a una revolución tecnológica, propiciando que se incluyan en gran cantidad de dispositivos.

Funcionamiento



Lectura de entradas >> procesado de la información >> decisión de la actuación >> actuación sobre las salidas

Entradas y salidas

Analogía entre Arduino y el cuerpo humano:

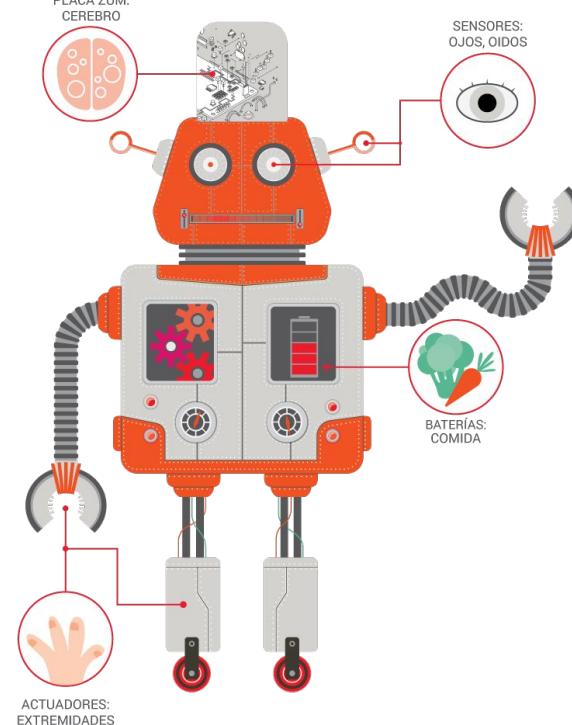
Entradas: información

Cuerpo humano

- Sentidos

Arduino

- Pulsador
- Sensor



Salidas: actuación

Cuerpo humano

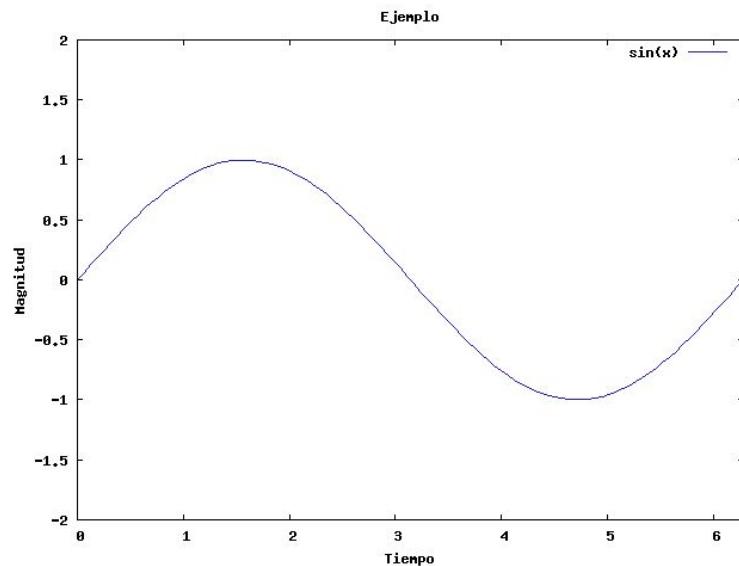
- Brazos
- Piernas
- Voz

Arduino

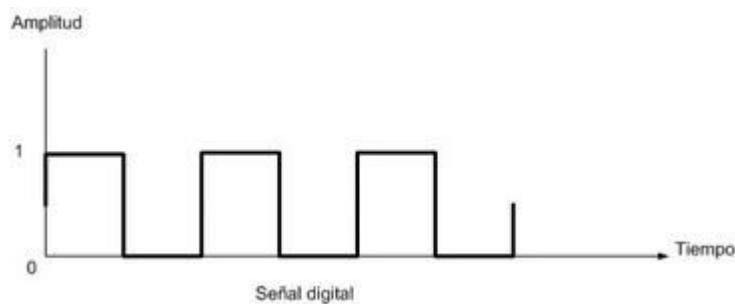
- Motores
- Leds
- Altavoz

Señales analógicas

Una señal es analógica cuando varía de forma continua entre un mínimo y un máximo, pudiendo tomar todos los valores intermedios.



Señales digitales



0: 0v: Falso

1: 5v: Verdadero

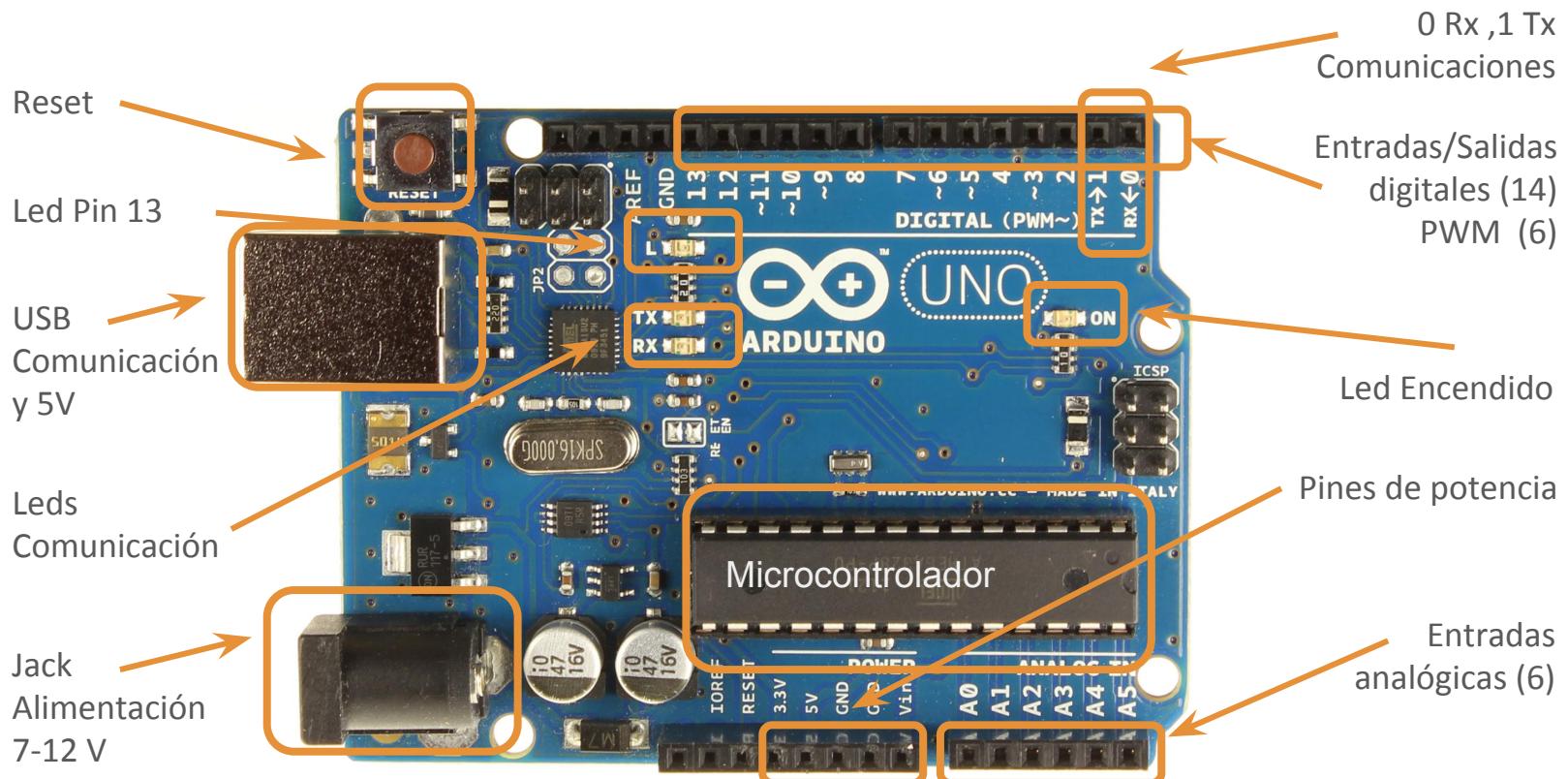
Una señal digital es aquella que solamente toma dos valores:
uno mínimo y uno máximo.

Ventajas sistemas digitales

- Rapidez de procesamiento y transmisión sencilla de los datos
- Fácilmente programables con lenguajes de alto nivel
- Almacenamiento sencillo de información y sin límites
- Menos susceptibles al ruido, pues pueden detectarse valores ruidosos
- Menor costo
- Capacidad de integración dado su pequeño tamaño
- Diseño modular

Desventaja: nuestro mundo es analógico por lo que esos valores deben convertirse en digitales

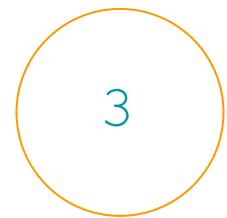
Arduino UNO: Conexiones



CB Arduino: Reflexiones

1. ¿Qué es un microcontrolador? ¿Cómo funcionan?
2. Por qué se ha impuesto el uso de estas tecnologías
3. Cosas que funcionen con un microcontrolador





PRÁCTICAS CON ARDUINO

P1 :Hola mundo
Arduino

Hola mundo

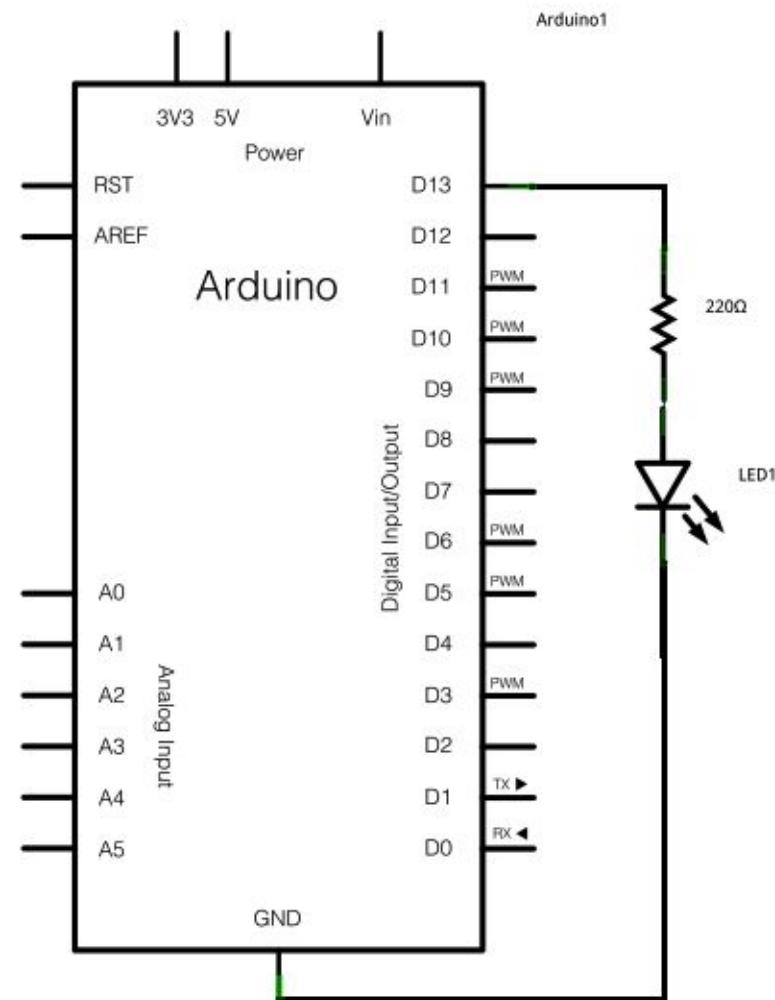
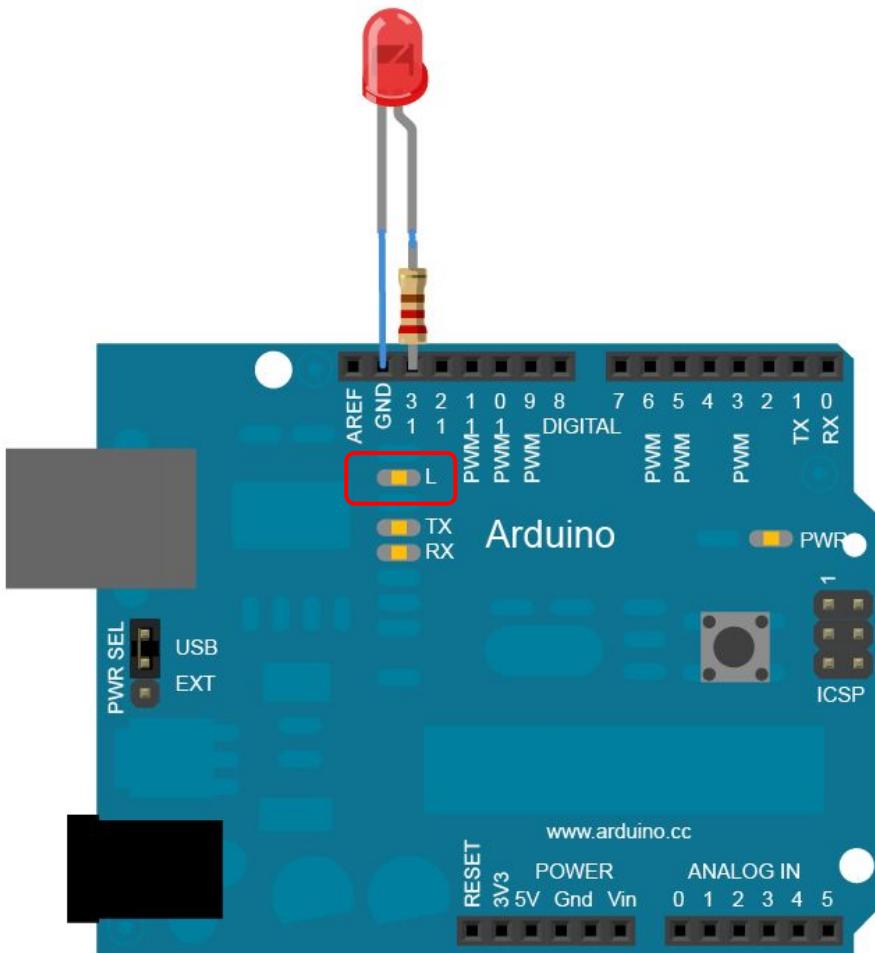
- Nos introduce en la estructura del código de Arduino.
- Es un recurso al que podemos recurrir para comprobar que el sistema compila y carga correctamente en la placa.
- Nos iniciamos en este lenguaje desde la perspectiva de “aprender haciendo”.

Finalidad

La finalidad de nuestro sistema es encender y apagar un LED de forma repetitiva o cíclica.



Hardware-Esquema



Software

El código de Arduino se divide en tres partes:

- La primera, se encarga de las Variables Globales.
- La segunda, el Setup, posee la característica de que se ejecuta una sola vez. Se utilizará para realizar la configuración del sistema.
- La tercera parte es el Loop, que se ejecuta de forma cíclica y donde colocaremos el cuerpo de nuestro programa.

Global
Variables

Setup()

Loop()

Software

Diagrama de flujo del programa que vamos a realizar, que consta de los siguientes elementos:



Código Arduino

- La primera parte suele estar dedicada a **comentarios explicativos** del funcionamiento del programa. Para hacer comentarios de tipo párrafo utilizaremos: /* */
Los comentarios de línea irán precedidos de: //
- A continuación, encontraremos la declaración de **Variables Globales**. En el siguiente caso, hemos declarado la variable ledPin y le hemos asociado el Pin 13.
- Le sigue la función **Setup**. En el caso que se muestra a continuación, al utilizar una salida digital, debemos configurarla, ya que los elementos digitales pueden ser tanto de entrada como de salida. Para ello nos valdremos de la instrucción pinMode. Nos pide el Pin y asociar un valor de salida o entrada.
- Pasamos al **Void Loop**, el cuerpo de nuestro programa. Para establecer 5v, utilizaremos la función digitalWrite, con el Pin del LED y los valores alto o bajo. Para esperar un segundo utilizaremos la función delay (en milisegundos). Volvemos a utilizar digitalWrite para establecer 0v, en este caso con la constante LOW. Para volver a esperar un segundo volvemos a utilizar delay. Con esto terminaría la función loop, que se volvería a ejecutar.

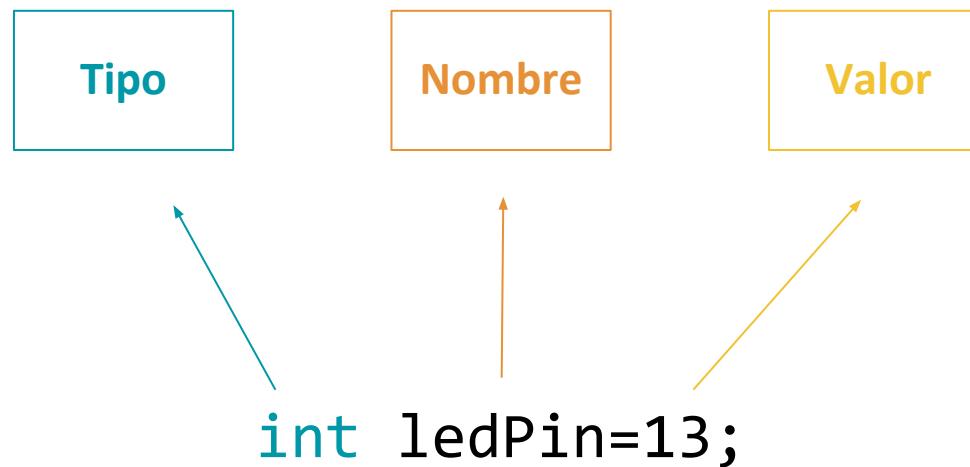
Código Arduino

```
8 // declaracion de variables
9 int ledPin=13; // LED asociado al pin 13
10
11 // la funcion setup se ejecuta una sola vez
12 void setup() {
13   pinMode(ledPin, OUTPUT); // inicializa el pin 13 como una salida digital
14 }
15
16 // la funcion loop se ejecuta repetidamente de forma infinita
17 void loop() {
18   digitalWrite(ledPin, HIGH); // establece 5v en el pin del LED
19   delay(1000); // espera un segundo
20   digitalWrite(ledPin, LOW); // establece 0v en el pin del LED
21   delay(1000); // espera un segundo
22 }
```

[Enlace al código](#)

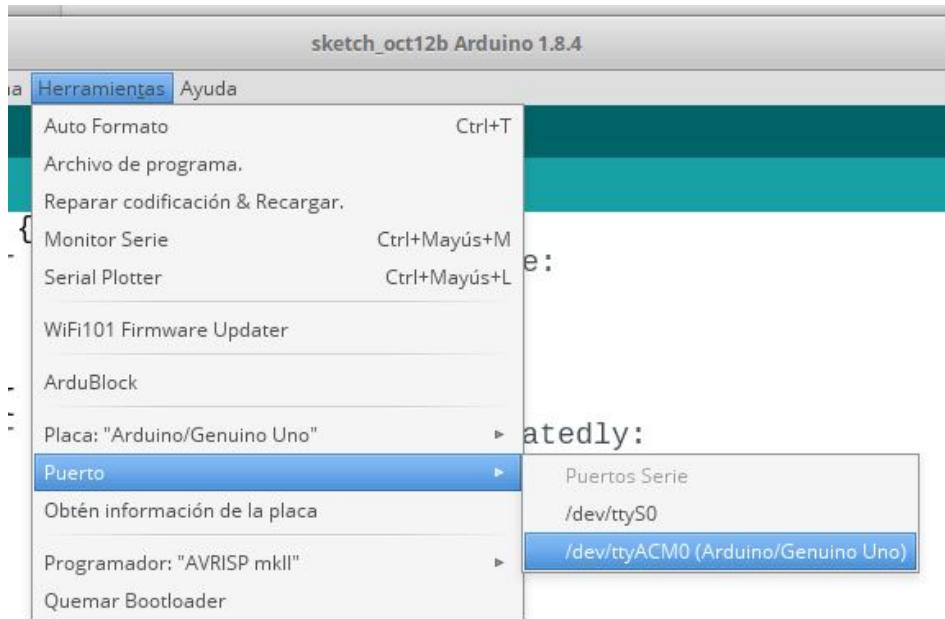
Variable

Es un espacio de memoria en el que almacenamos un valor. Las variables contribuyen a la legibilidad del código. Por lo tanto, deben elegirse nombres significativos. En el caso de que cambiemos los pines de conexión, nos servirá para realizar los cambios fácilmente.



Las variables están definidas por el tipo, nombre y valor que le asociamos.

Configuración y carga de programas



Subir el programa a la placa

- Seleccionamos el modelo de placa: Herramientas → Placa → Arduino UNO
- Seleccionamos el puerto de conexión: Herramientas → Puerto /dev/ttyACM0
- La placa debe estar conectada
- El nombre del puerto varía según SO y modelo de placa

Código Arduino-Recordar

- Utilizar nombres de variables significativas
- Terminar cada una de las instrucciones en punto y coma (;)
- Abrir y cerrar los bloques con una llave ({ })
- Utilizar la tabulación para facilitar la lectura del programa

Propuestas de actividad

- Cambiar los tiempos de parpadeo
- Que parpadee al ritmo del corazón
- ¿A qué velocidad deja de verse el parpadeo del LED?
- ¿Cómo podemos hacer que solo parpadee una vez?
- Crear variables para el tiempo de parpadeo
- Decrementar el tiempo de parpadeo en cada iteración.

P2: Semáforo.
Secuencias y
repeticiones

SALIDAS DIGITALES

Finalidad

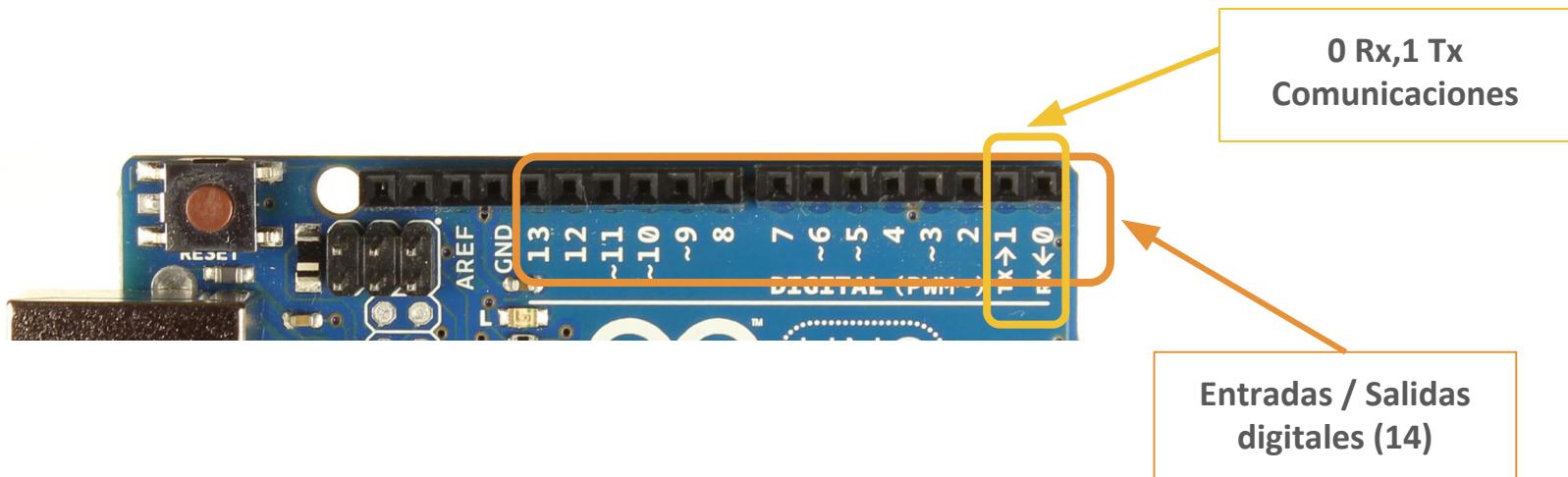
La Finalidad de nuestro sistema es hacer una secuencia de luces que simule un semáforo de coches



Salidas Digitales

Disponemos de los pines digitales (entradas y salidas), que van del 0 al 13.

Las entradas analógicas (6) son configurables como digitales,
si fuera necesario.



Configuración de Entrada o Salida

Los pines deben ser configurados en el setup mediante la instrucción **pinMode**, indicando el número del pin al que vamos a conectar nuestro componente y el valor.

El valor puede tomar la constante **OUTPUT** o **INPUT**.

Por defecto, están configurados como entradas.

pinMode(pin, Valor)

OUTPUT

activamos el pin digital como una salida

INPUT

activamos el pin digital como una entrada

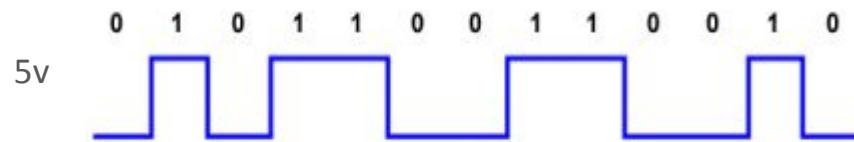
DigitalWrite: activando/desactivando la salida

Para establecer un valor de 0v o 5v utilizamos la instrucción **digitalWrite**, indicando una vez más el pin y el valor, que puede tomar las constantes **LOW** y **HIGH**.

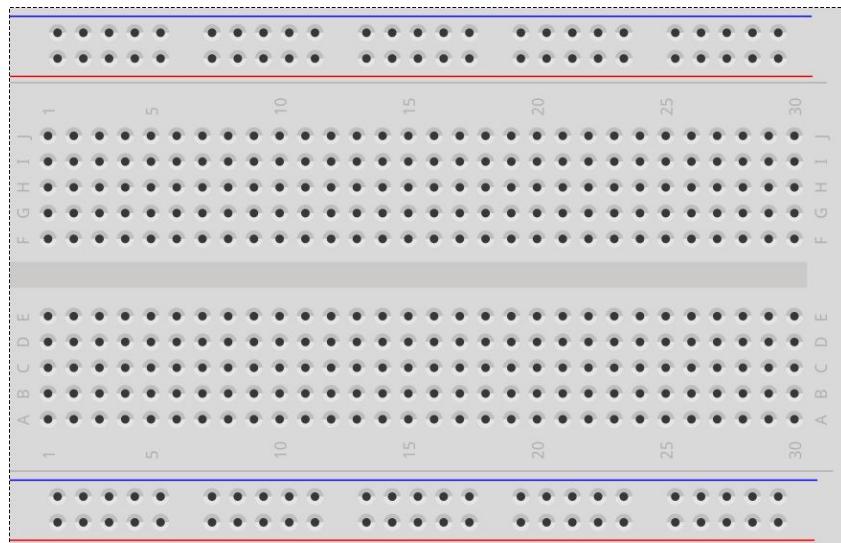
digitalWrite(pin, Valor)

digitalWrite(pin, HIGH)
digitalWrite(pin, LOW)

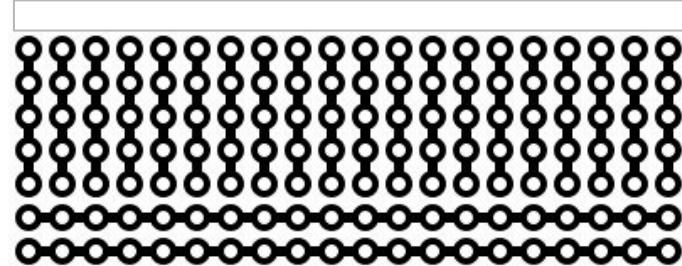
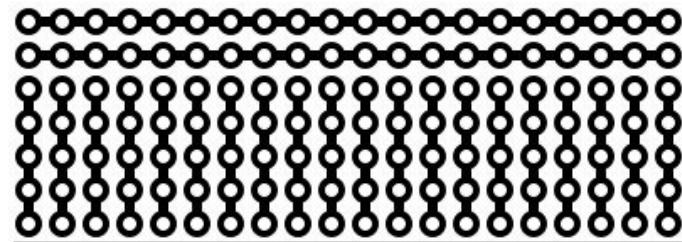
Nombre	Constante	Valor Lógico	Voltaje
Bajo	LOW	0	0V (GND)
Alto	HIGH	1	5V



Placa de prototipos



fritzing



Cables de conexión

Rojo=5v



Colores pines



Negro=0v



Resistencias



220Ω

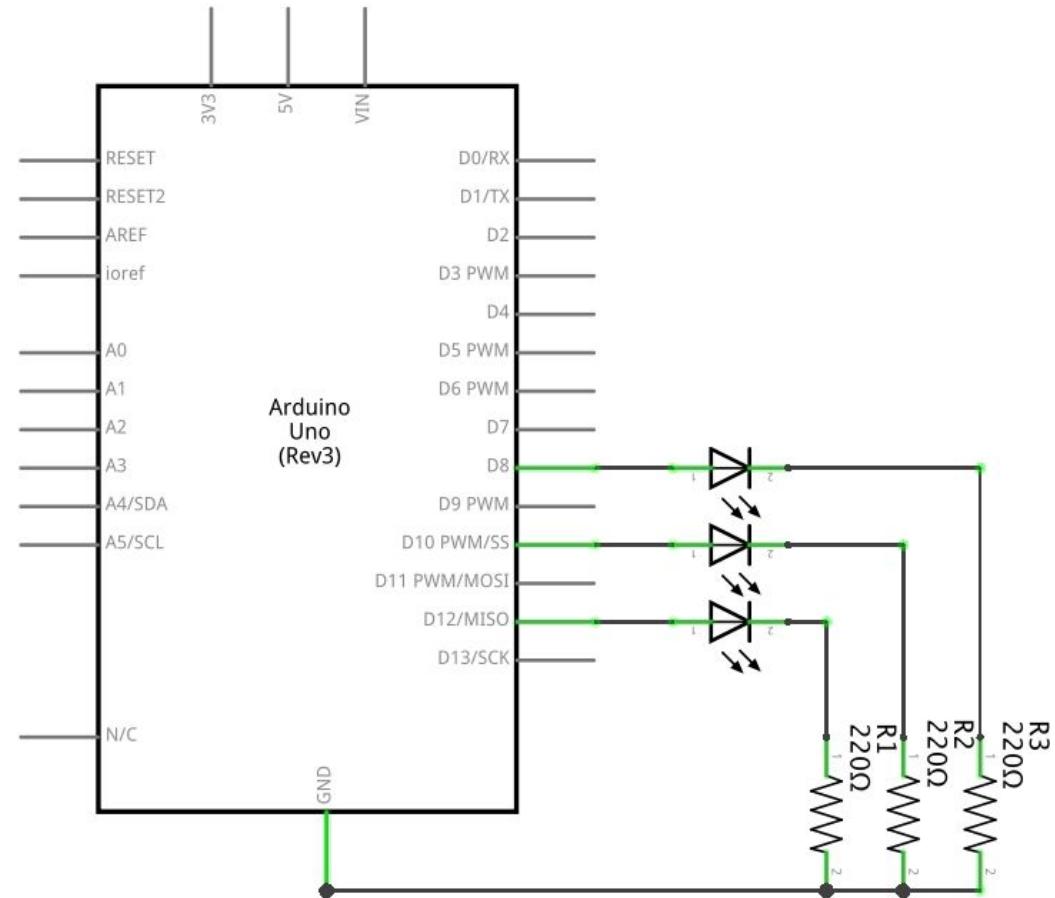
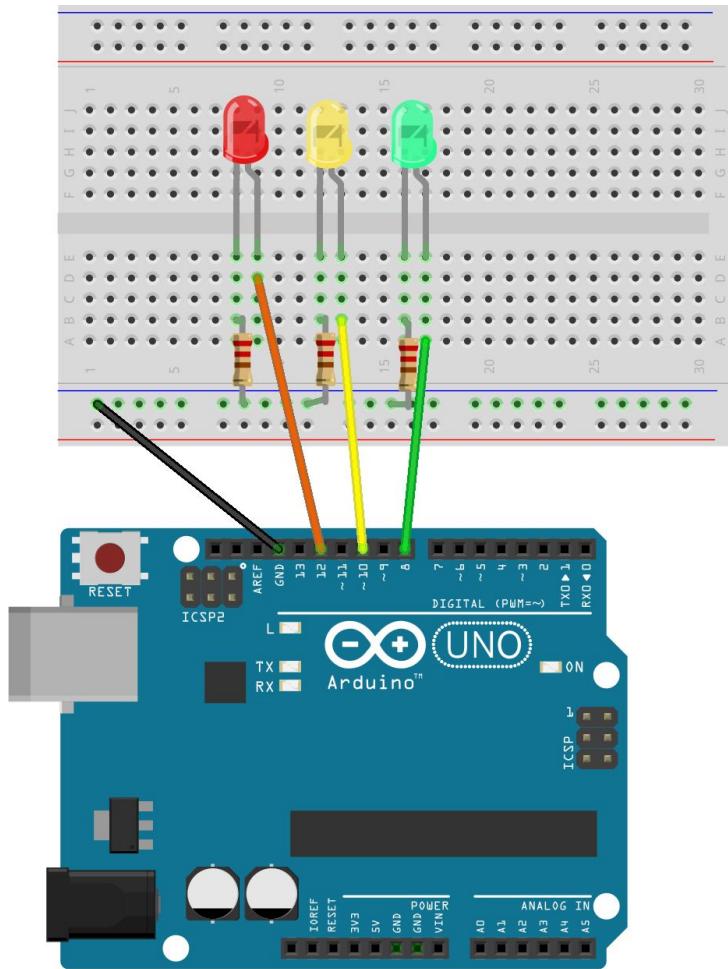


$1 \text{ K}\Omega$



$10 \text{ K}\Omega$

Hardware-Esquema



fritzing

Software

```
// Declaracion de variables de tipo constante entero
const int ledRPin = 12; // LED rojo asociado al pin 12
const int ledAPin = 10; // LED amarillo asociado al pin 10
const int ledVPin = 8; // LED verde asociado al pin 8

// la funcion set up se ejecuta una sola vez
void setup() {
    // inicializa los pines como una salida digital
    pinMode(ledRPin, OUTPUT);
    pinMode(ledAPin, OUTPUT);
    pinMode(ledVPin, OUTPUT);
}

// la funcion loop se ejecuta repetidamente de forma infinita
void loop() {
    // ESTADO SEMAFORO VERDE
    digitalWrite(ledRPin, LOW); // establece 0v en el pin del LED rojo
    digitalWrite(ledAPin, LOW); // establece 0v en el pin del LED amarillo
    digitalWrite(ledVPin, HIGH); // establece 5v en el pin del LED verde
    delay(6000); // espera seis segundos
    // ESTADO SEMAFORO AMARILLO
                // establece 5v en el pin del LED verde
                // establece 0v en el pin del LED rojo
                // establece 0v en el pin del LED verde
    delay(2000); // espera dos segundos
    // ESTADO SEMAFORO ROJO

    delay(4000); // espera cuatro segundos
}
```

[Enlace al código](#)

Propuestas de actividad

- Completar el código
- Simplificar el código eliminando instrucciones innecesarias
- Introducir un parpadeo en el LED amarillo
- Usa la instrucción for para hacer parpadear el LED amarillo
- Acelerar el parpadeo del LED

Sentencia de repetición For en Arduino

```
for (inicialización; condición de repetición; incremento) {  
    //sentencia(s);  
}
```

```
for (int i=0;i<20;i=i+1) {  
    //sentencia(s);  
}
```

P3: Pulsadores

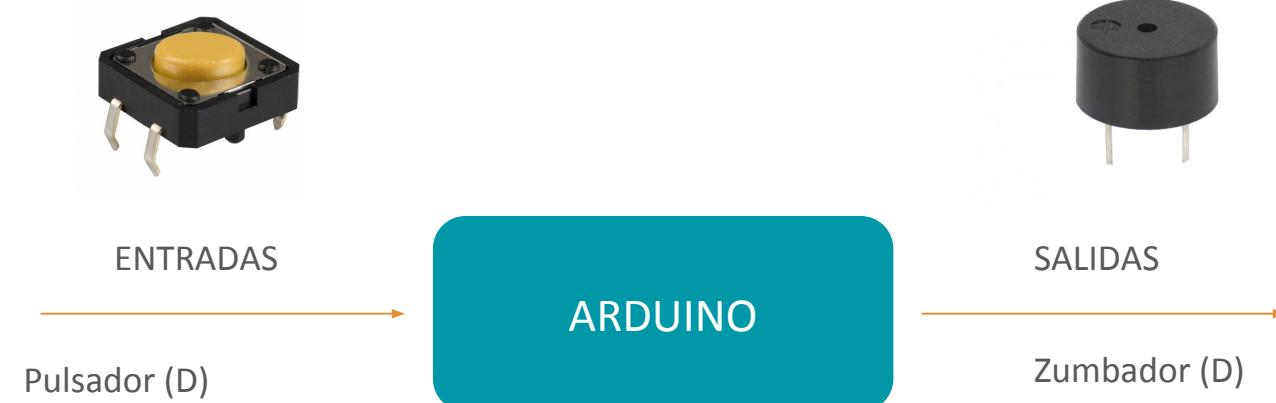
3A Pulsador-Zumbador

3B Pulsador con Memoria

ENTRADAS DIGITALES

Finalidad

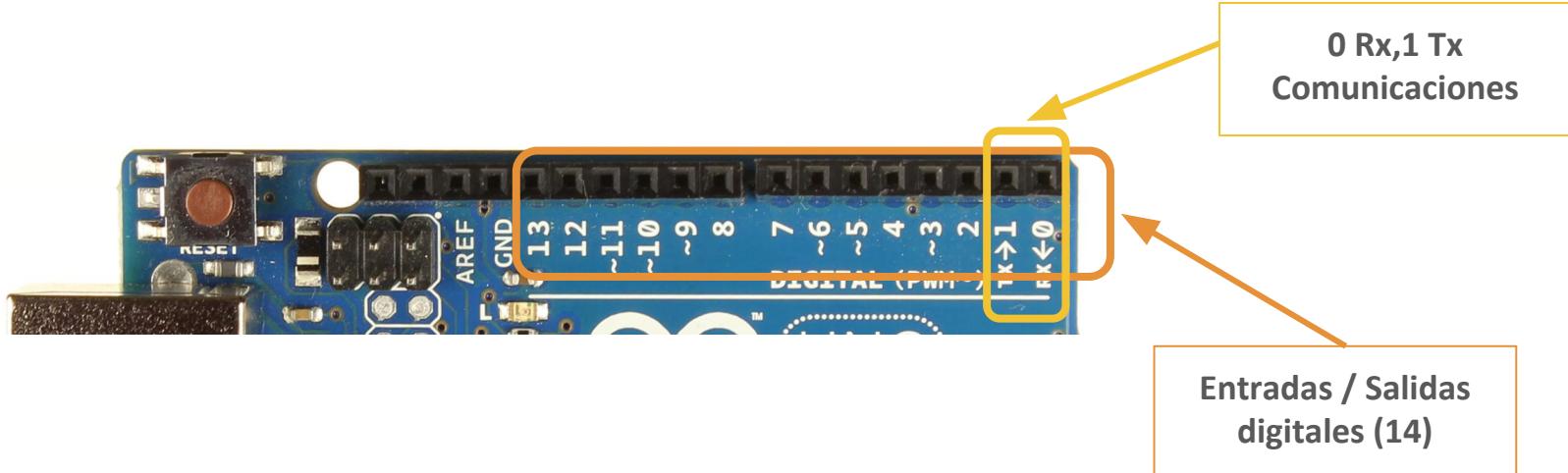
La Finalidad de nuestro sistema es controlar un zumbador con un pulsador de forma que este suene si está presionado el pulsador



Entradas Digitales

Disponemos de los pines digitales (entradas y salidas), que van del 0 al 13.

Las entradas analógicas (6) son configurables como digitales,
si fuera necesario.



Configuración de Entrada o Salida

Los pines deben ser configurados en el setup mediante la instrucción **pinMode**, indicando el número del pin al que vamos a conectar nuestro componente y el valor.
El valor puede tomar la constante **OUTPUT** o **INPUT**.
Por defecto, están configurados como entradas.

pinMode(pin, Valor)

OUTPUT

activamos el pin digital como una salida

INPUT

activamos el pin digital como una entrada

Lectura de Entradas Digitales

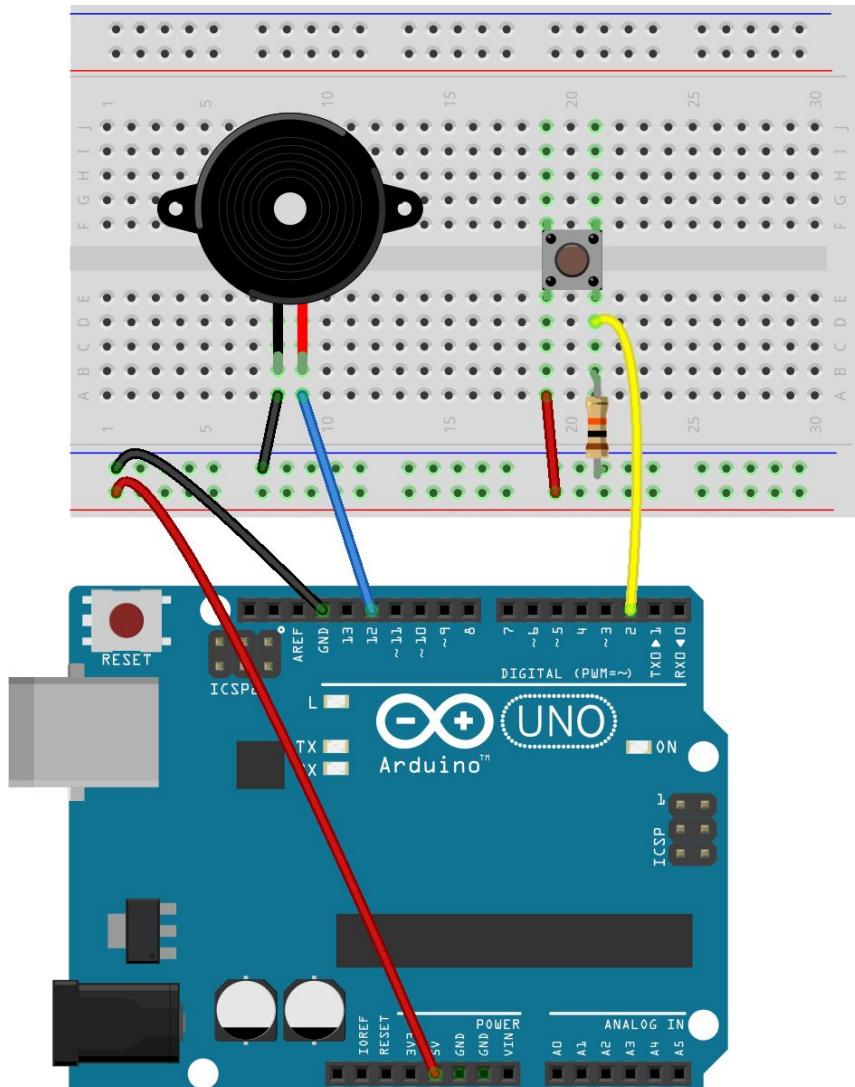
Para leer el valor digital usaremos:

```
int Lectura = digitalRead(pin);
```

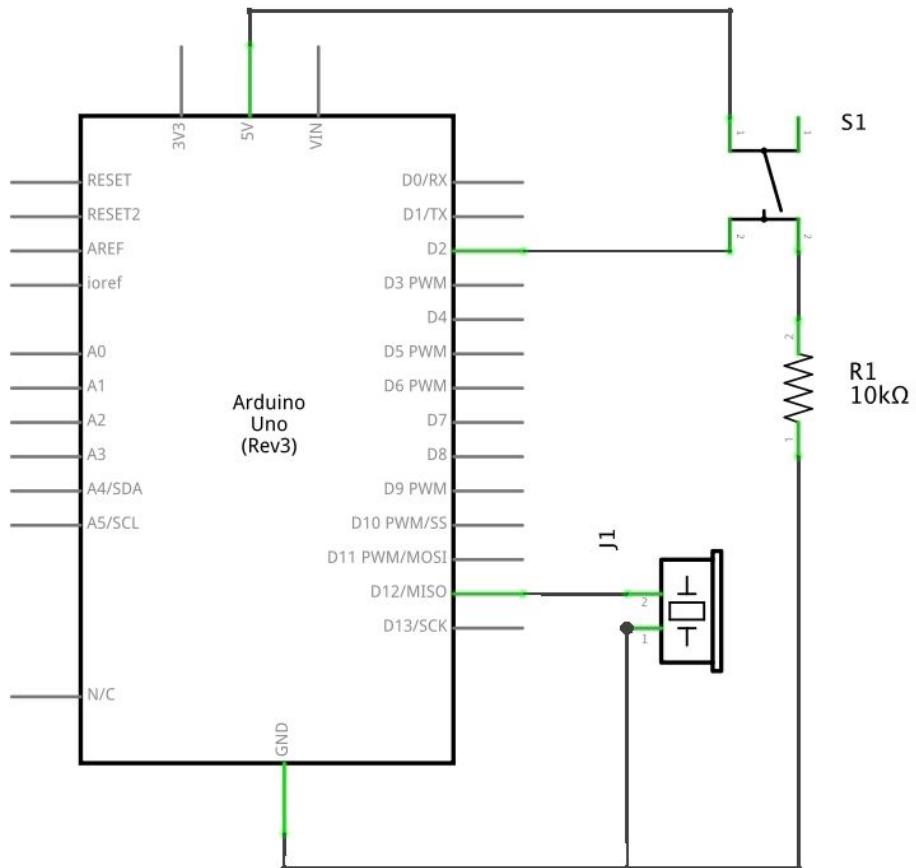
Lectura valdrá:

LOW	0v	0	Bajo
HIGH	5v	1	Alto

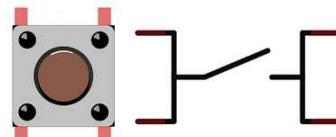
Hardware-Esquema



Parte 1

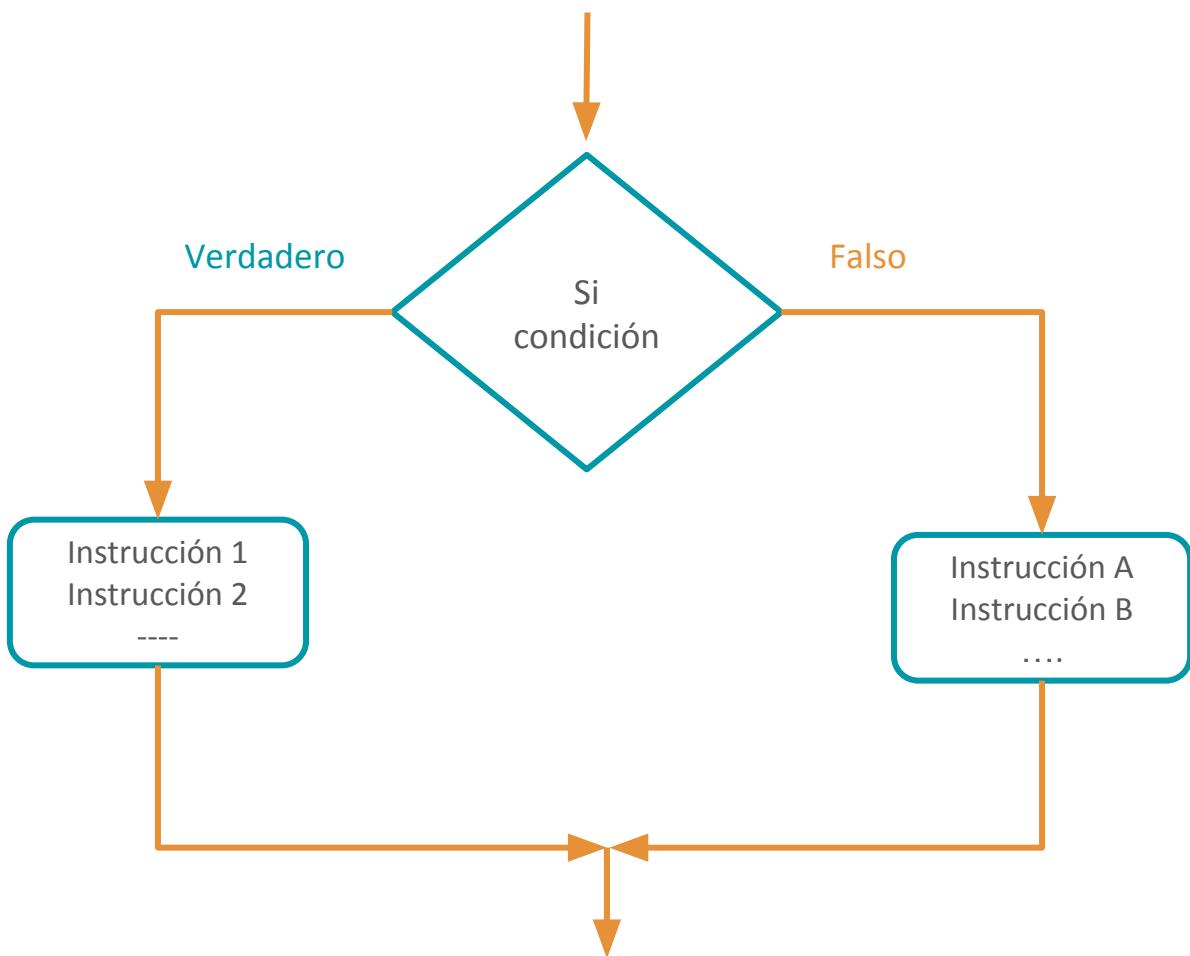


fritzing



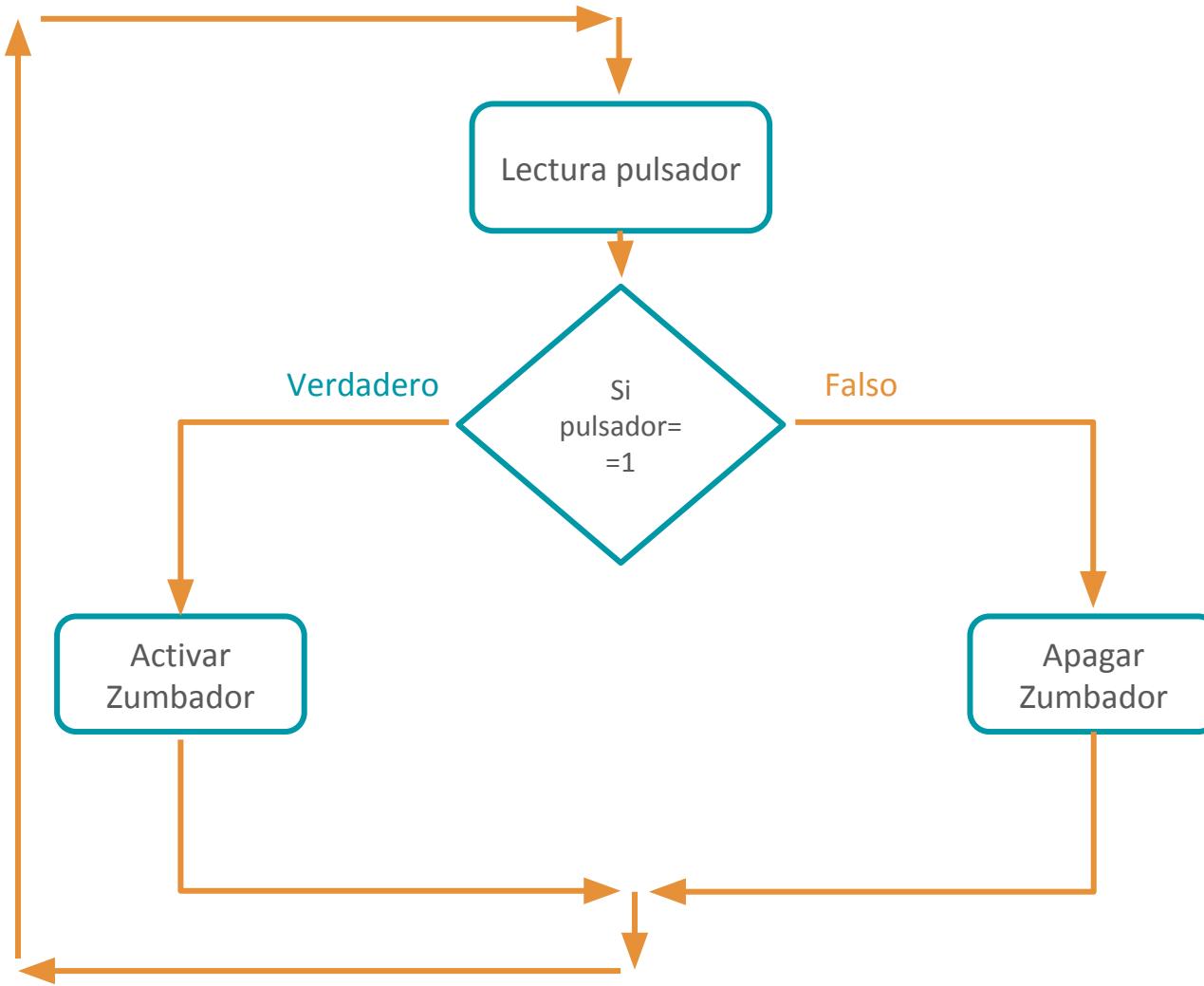
fritzing

Estructura Si/Sino



```
if (condicion) {  
    // Instrucción 1  
    //Instrucción 2  
}  
  
else {  
    // Instrucción A  
    //Instrucción B  
}
```

Diagrama de Flujo



Código Arduino

```
8 // constantes que no cambian se usan para establecer los pines
9 const int buttonPin = 2;      // el numero del pin del pulsador
10 const int buzzerPin = 12;     // el numero de pin del zumbador
11
12 // variables que cambian
13 int buttonState = 0;          // variable para almacenar el estado del pulsador
14
15 void setup() {
16   // inicializa el pulsador como entrada
17   pinMode(buttonPin, INPUT);
18   // inicializa el zumbador como salida
19   pinMode(buzzerPin, OUTPUT);
20 }
```

Debemos indicarle al microcontrolador si van a ser entradas (INPUT) o salidas(OUTPUT)

Empezamos declarando los pines donde vamos a conectar los componentes del sistema (2 para el pulsador y 12 para el zumbador). Hemos declarado las variables de tipo constante y entero

Código Arduino

```
void loop() {  
  
    // lee el estado del pulsador y lo almacena en la variable  
    buttonState = digitalRead(buttonPin);  
  
    // revisa si el pulsador esta presionado  
    if (buttonState == HIGH) {  
        digitalWrite(buzzerPin, HIGH);          // activa el zumbador  
    }  
    // si no esta presionado  
    else {  
        digitalWrite(buzzerPin, LOW);          // apaga el pulsador  
    }  
}
```

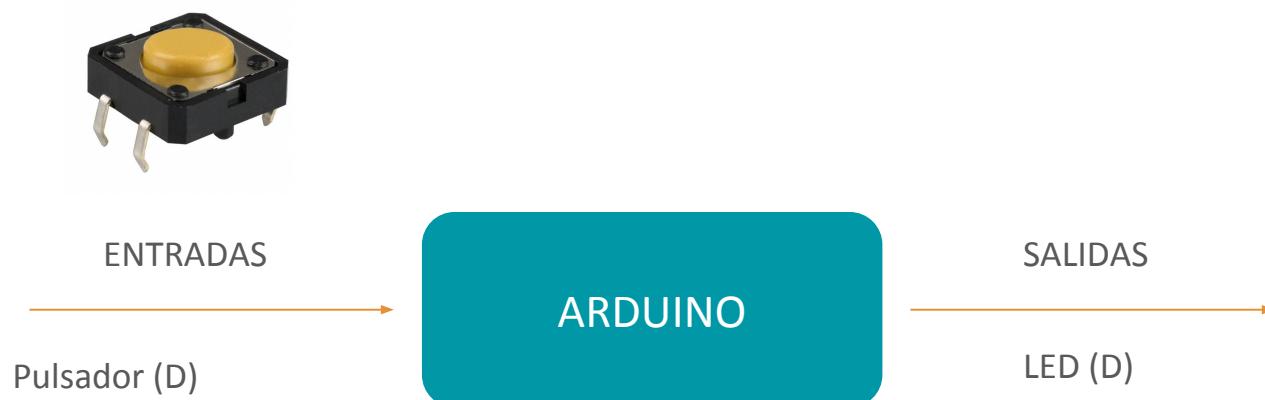
[Enlace al código](#)

Propuesta de actividad

Añadir dos LEDS (verde y rojo) de forma que cuando el zumbador esté accionado el LED verde esté encendido y cuando el zumbador esté apagado se encienda el LED rojo.

Finalidad 3B

Simulamos con un pulsador el funcionamiento de un interruptor.
El objetivo de esta práctica es poder conectar y desconectar sistemas e introducir la memoria de estado.



Hardware-Esquema

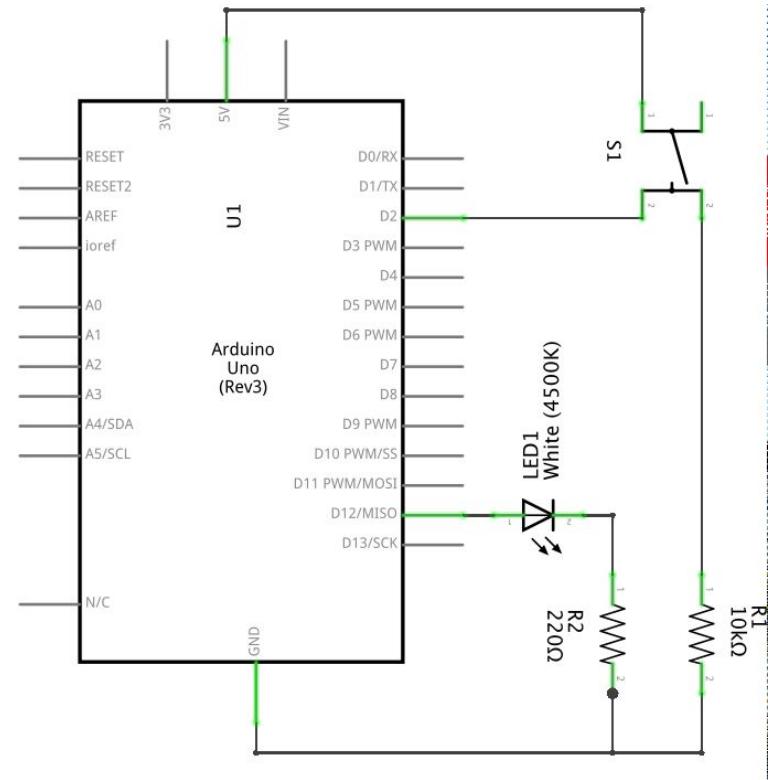
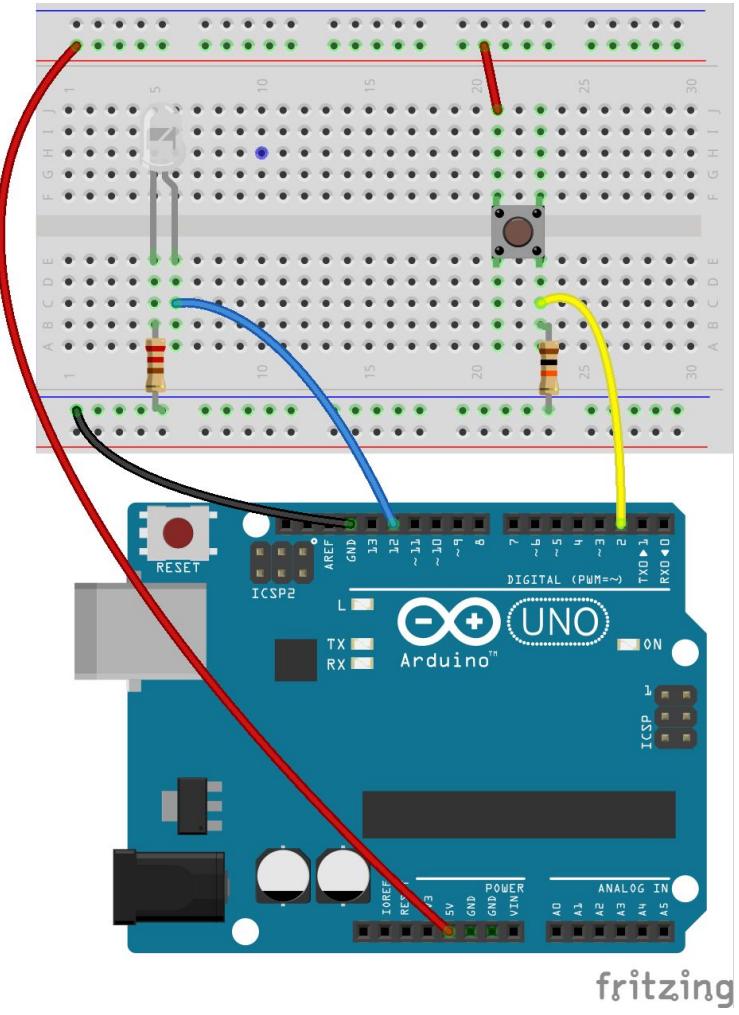
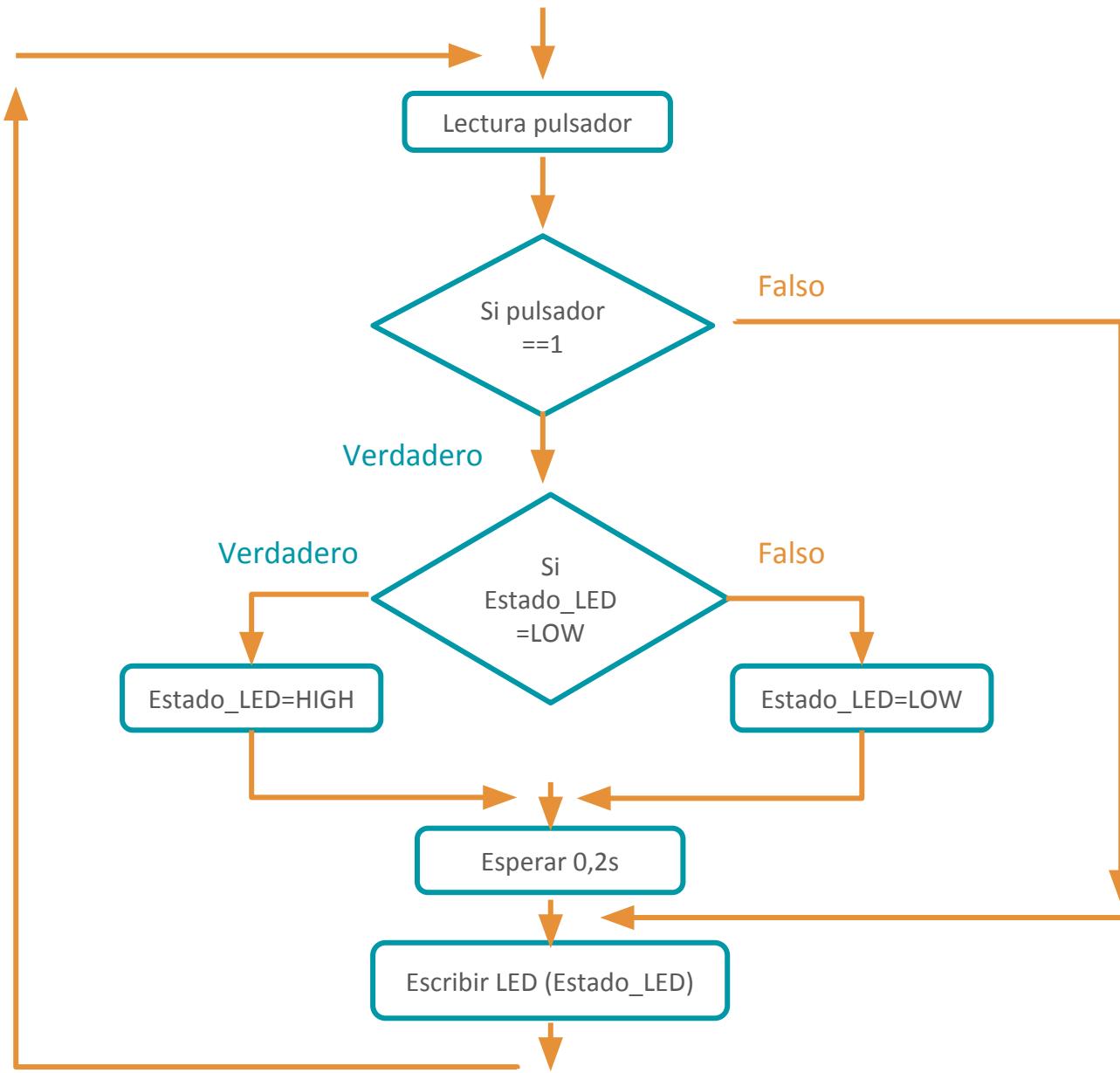


Diagrama de Flujo



Código Arduino

```
6 // constantes que no cambian se usan para establecer los pines
7 const int buttonPin = 2;      // el numero del pin del pulsador
8 const int ledPin = 12;        // el numero de pin del led
9
10 // variables que cambian
11 int ledState = LOW;          // Estado actual del led apagado
12 int buttonState = 0;          // variable para almacenar el estado del pulsador
13
14 void setup() {
15     // inicializa los pines como entradas y salidas
16     pinMode(buttonPin, INPUT);
17     pinMode(ledPin, OUTPUT);
18 }
19
```

Código Arduino

```
20 void loop() {  
21 // lee el estado del pulsador y lo almacena en variable  
22 buttonState = digitalRead(buttonPin);  
23  
24 // revisa si el pulsador esta presionado  
25 if (buttonState == HIGH) {  
26 // si el estado del led es bajo  
27 if (ledState == LOW) {  
28 ledState = HIGH; // lo establece como alto  
29 }  
30 // en caso contrario lo establece como bajo  
31 else {  
32 ledState = LOW;  
33 }  
34 // tiempo que evita que el programa siga cumpliendo la condicion  
35 delay(200);  
36 }  
37 // establece el estado del LED:  
38 digitalWrite(ledPin, ledState);  
39 }
```

Si anidado

[Enlace al código](#)

Propuesta de actividad

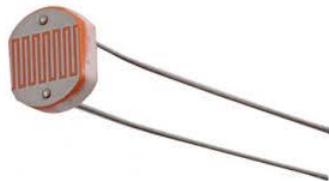
- Probar el código Debounce. Archivo → Ejemplos → Digital → Debounce
- Probar a realizar el código usando While (mientras el pulsador esté accionado)
- Realizar un sistema que incluye el funcionamiento de las dos actividades, con un pulsador que active un zumbador y un pulsador con memoria que active un LED.

P4: Interruptor Crepúscular

ENTRADAS ANALÓGICAS

Finalidad

El objetivo es medir la realidad a través de un sensor analógico, para encender y apagar las luces en función de la intensidad luminosa. Para ello tenemos que incorporar la comunicación serie para conocer el estado de la medición del sensor de luz y poder tomar decisiones. Es, por tanto, un sistema inteligente que controla el encendido de las luces.

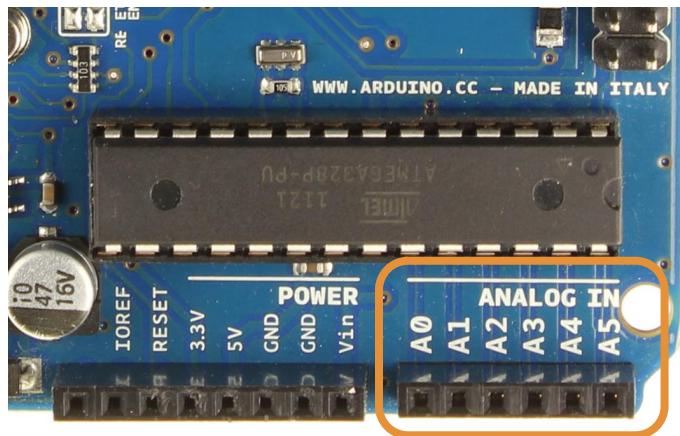


ENTRADAS
LDR (A)



SALIDAS
LED (D)

Entradas Analógicas



Entradas analógicas (6)

Desde A0 hasta A5

No necesitan configuración

Arduino usa 10 bits =
 $2^{10} = 1024$ niveles

Lectura de entradas analógicas

Para leer el valor de una entrada analógica usaremos:

```
int valor=analogRead(pin);
```

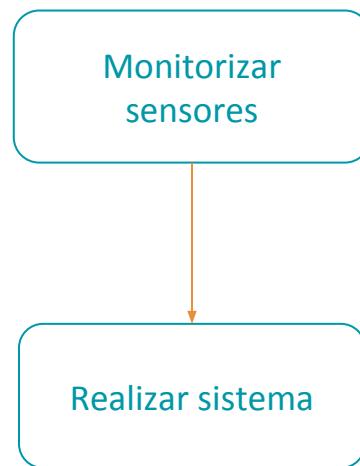
pin entre A0 y A5

valor:

Valor	Voltaje
0	0v
1023	5v

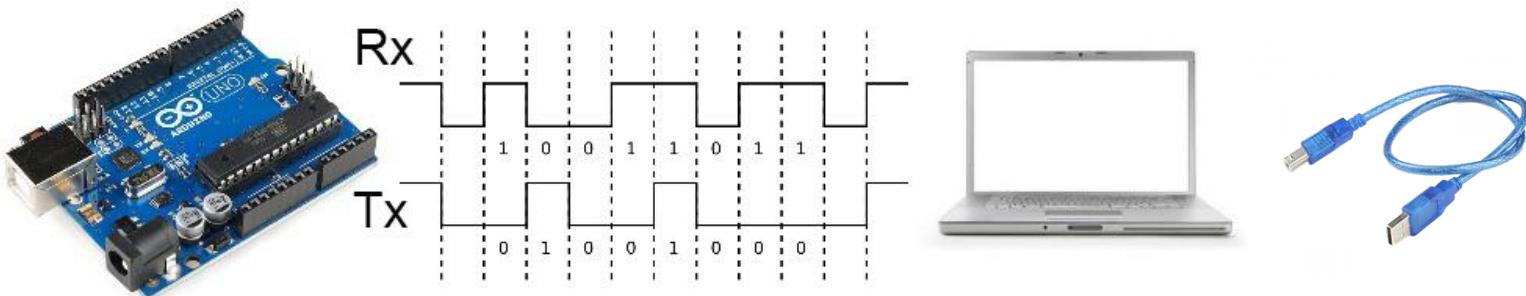
Software: Fases

En la 1^a fase nos planteamos qué valores nos da la LDR según las condiciones de luz. Una vez monitorizado el sensor, pasaremos a la 2^a fase.



Comunicación Serie

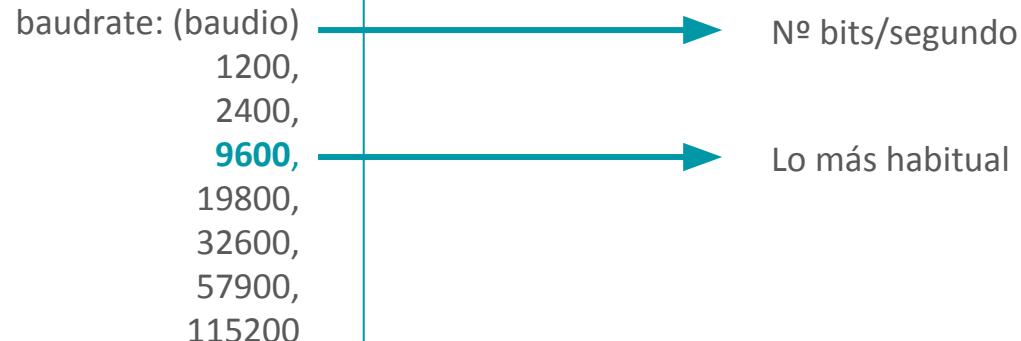
- Se realiza a través de los pines Rx (recepción de datos) y Tx (transmisión de datos)
- Número bits: 1 byte= 8 bits
- Velocidad: Baudios
- Comunicación a través de USB (Universal Serial Bus)



Configuración comunicación serie

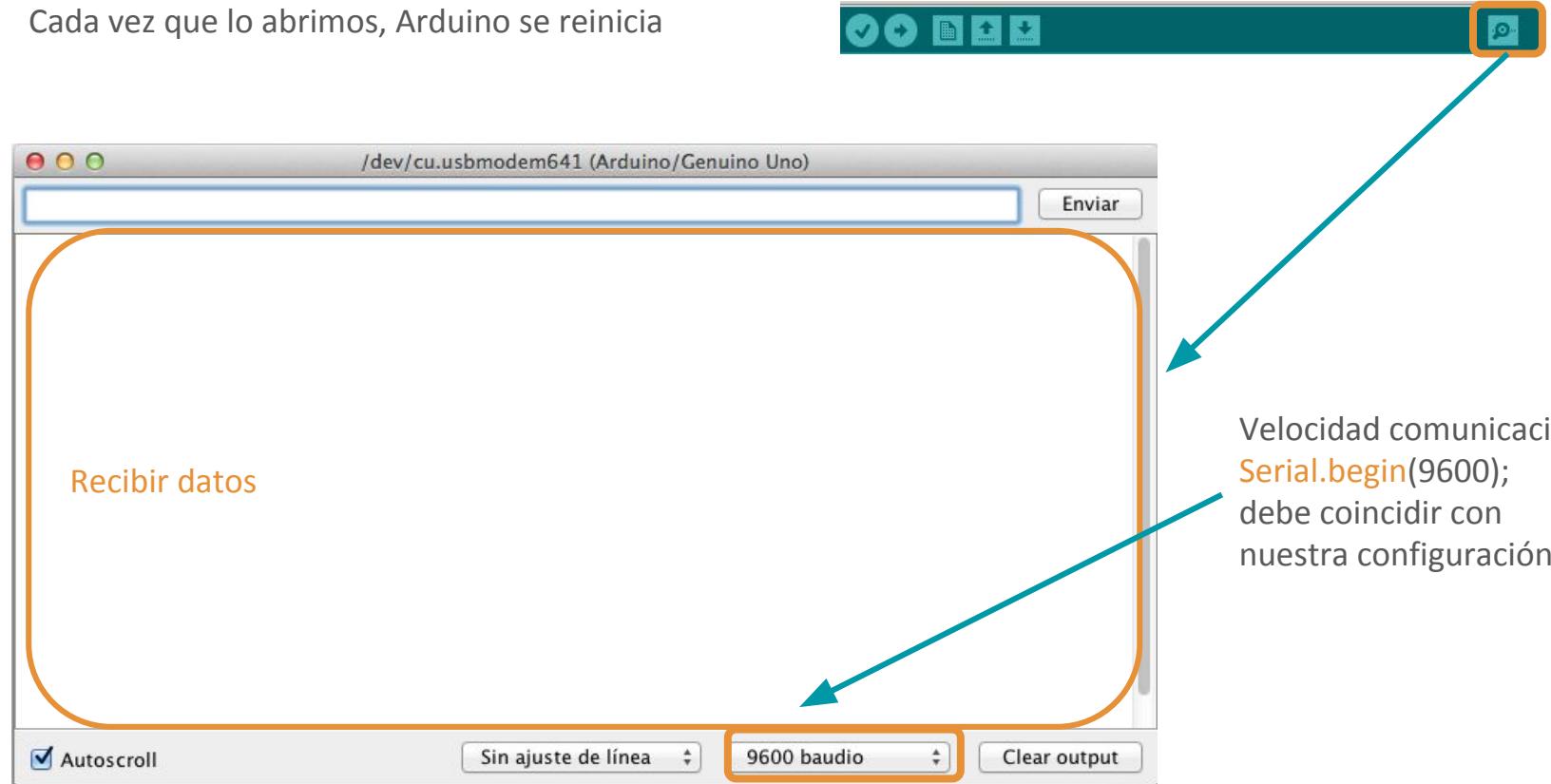
Necesitamos configurarlo en el **setup** mediante la siguiente instrucción:

```
Serial.begin(baudrate);
```



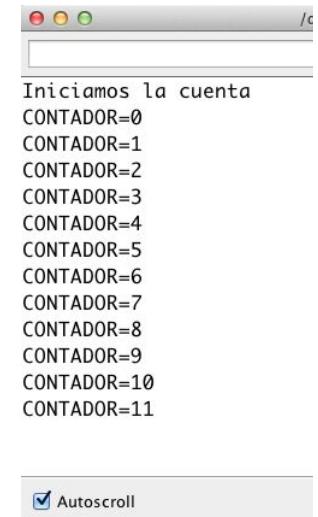
Monitor Serie

Cada vez que lo abrimos, Arduino se reinicia



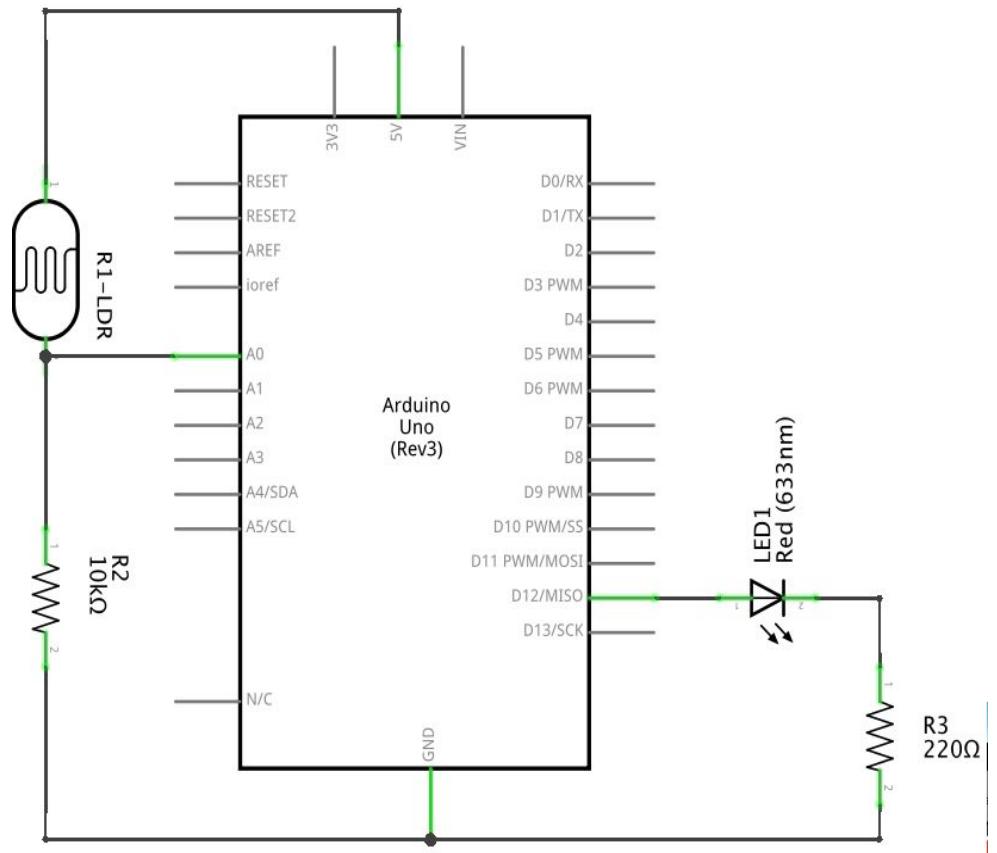
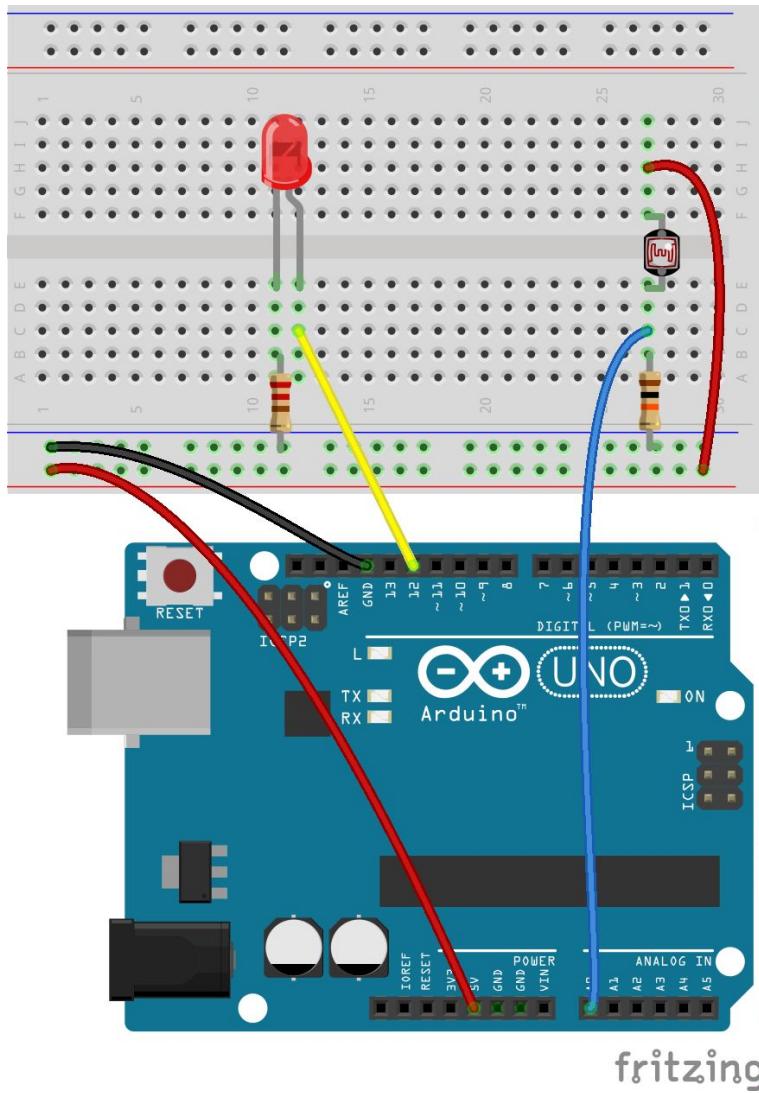
Realización de un contador

```
9 int contador = 0; // variable para almacenar el valor del contador
10
11 void setup() {
12     // abre el puerto serie
13     // y establece la velocidad de conexión en baudios
14     Serial.begin(9600);
15     Serial.println("Iniciamos la cuenta"); // Imprime texto
16 }
17
18 void loop() {
19
20     Serial.print("CONTADOR="); // Imprime texto
21     Serial.println(contador); // Imprime el valor de la variable contador
22     delay(1000); // tiempo de espera de 1s
23     contador++; // se incrementa el valor del contador en 1
24 }
```



[Enlace al código](#)

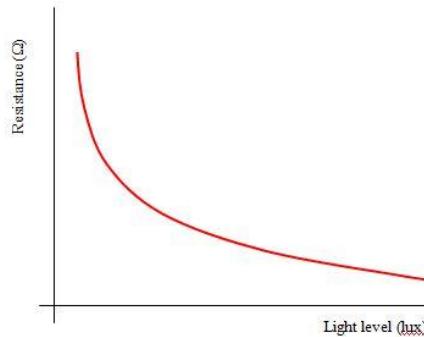
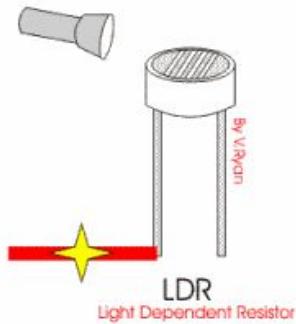
Hardware-Esquema



fritzing

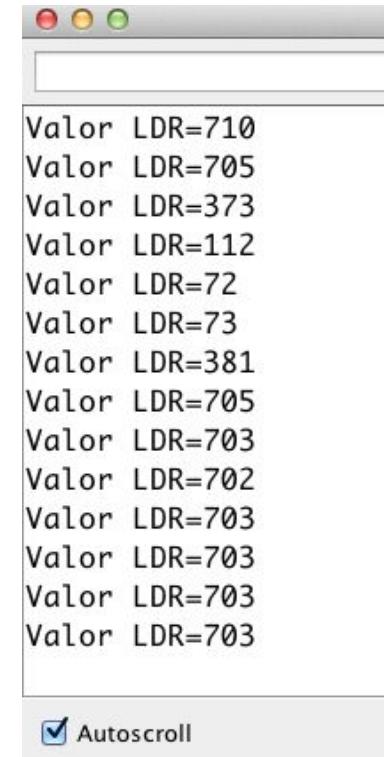
Funcionamiento LDR

- A medida que iluminamos, se produce un cambio en la resistencia.
- La resistencia disminuye de forma exponencial a medida que aumentamos la luz.
- El montaje adecuado es un divisor de tensión donde la resistencia está cerca de la alimentación. Hay otra resistencia en un formato pull down, montada como un divisor de tensión. Nos permite obtener una lógica positiva. Es decir, a más luz, más voltaje.



Código monitorizar sensor

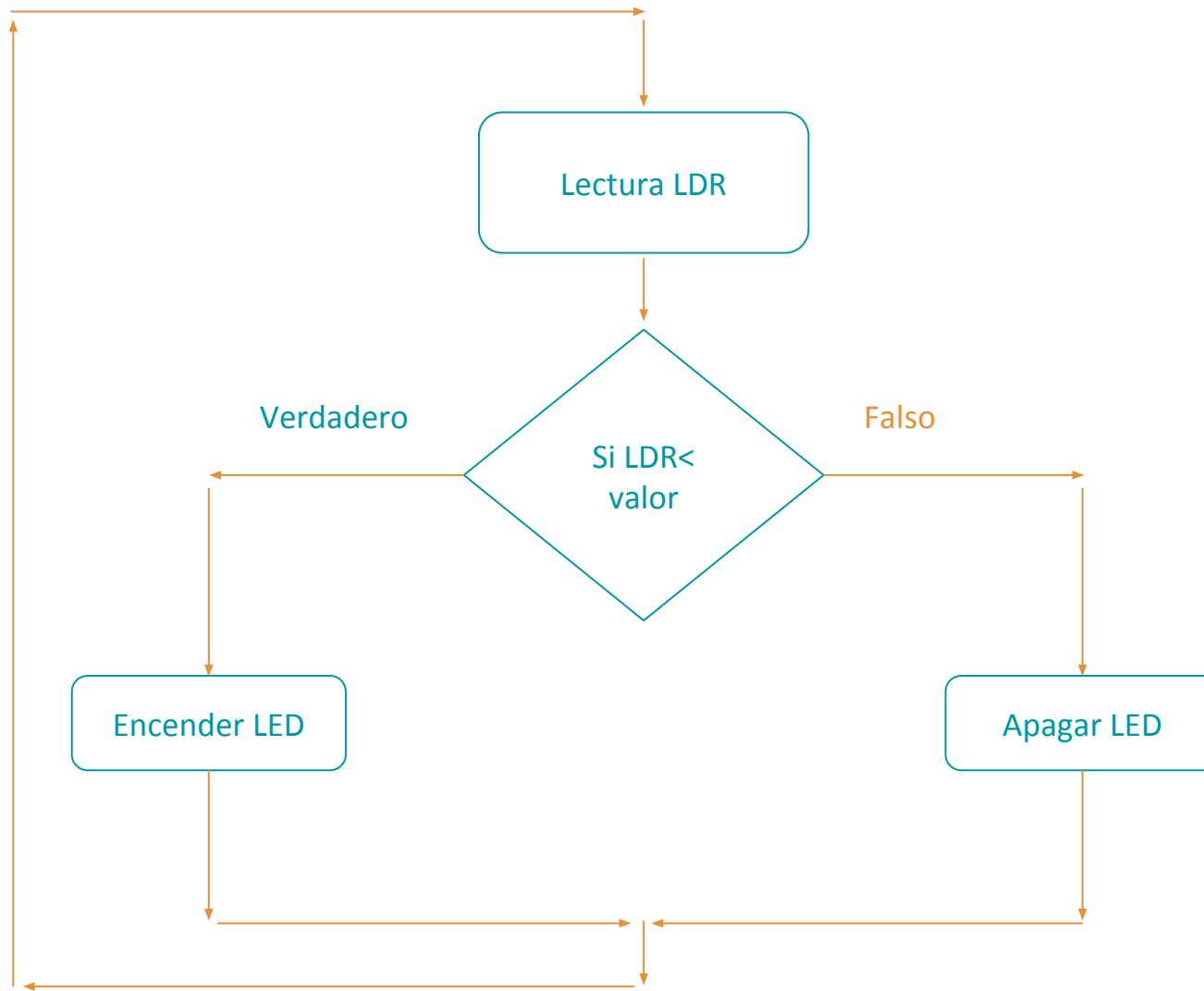
```
7 // variables constantes para los pines
8 const int ldrPin = A0;      // establece el pin de la LDR
9
10 // variable para almacenar el valor del sensor
11 int ldrValue = 0;    // variable para almacenar el valor del sensor
12
13 void setup() {
14     // abre el puerto serie
15     // y establece la velocidad de conexión en baudios
16     Serial.begin(9600);
17 }
18
19 void loop() {
20     // lee el valor del sensor
21     ldrValue = analogRead(ldrPin);
22     // Imprime un texto
23     Serial.print("Valor LDR=");
24     // Imprime el valor de la variable por el puerto serie
25     Serial.println(ldrValue);
26     // tiempo de espera para visibilidad
27     delay(1000);
28 }
```



Una vez hemos comprobado qué valores toma nuestra LDR según las condiciones de luz, anotaremos en qué punto fijamos el umbral a partir del cual el LED está encendido o apagado.

[Enlace al código](#)

Diagrama de flujo



Software

```
8 // variables constantes para los pines
9 const int ldrPin = A0;      // establece el pin de la LDR
10 const int ledPin = 12; // establece el pin del LED rojo
11
12 // variables que pueden cambiar
13 int ldrValue = 0; // variable para almacenar el valor de la LDR
14 int ldrlevel=200; // nivel de cambio del valor de la LDR
15
16 void setup() {
17     // establece el LED como una salida
18     pinMode(ledPin, OUTPUT);
19 }
20
21 void loop() {
22     // lee el valor del sensor
23     ldrValue = analogRead(ldrPin);
24     // si el valor es menor enciende los LEDs
25     if (ldrValue < ldrlevel) {
26         digitalWrite(ledPin, HIGH);
27     }
28     // sino los apaga
29     else {
30         digitalWrite(ledPin, LOW);
31     }
32 }
```

[Enlace al código](#)

Propuesta de ampliación

Realizar un sistema que muestre la intensidad luminosa por medio de 4 LEDS, de forma que a baja intensidad luminosa solo se ilumine un LED y a medida que va subiendo la intensidad se iluminen más leds.

Pistas- Operadores booleanos

```
if (ldrValue<100)
{
    // Instruccion A
}
if (ldrValue>=100 && ldrValue<300)
{
    // Instruccion B
}
```

Si el valor de la LDR es menor de 100 se encendería uno de los LEDS

Si el valor de la LDR es mayor o igual que 100 y menor de 300, se encenderían 2 LEDS

P5: Control del
brillo de un LED

SALIDAS ANALÓGICAS

Finalidad



Controlar el encendido de un LED pudiendo variar su intensidad luminosa.
Se realizará en dos fases: en la 1^a el control se realizará mediante el tiempo
y en la 2^a mediante un potenciómetro.

Relacionamos por primera vez entradas y salidas analógicas.
Posible aplicación de este sistema: control de la velocidad de un motor de corriente continua.

Salidas PWM



Salidas PWM (6) marcadas con ~

Las salidas PWM no necesitan configuración

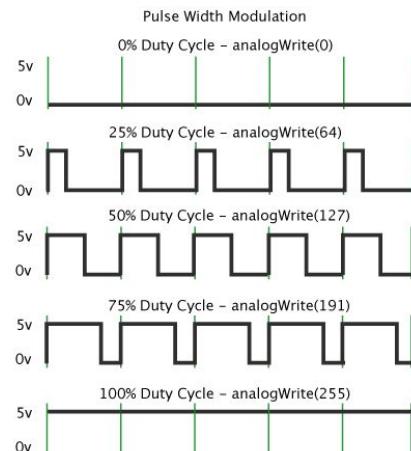
`analogWrite(pin,valor);`

3, 5, 6, 9, 10 y 11

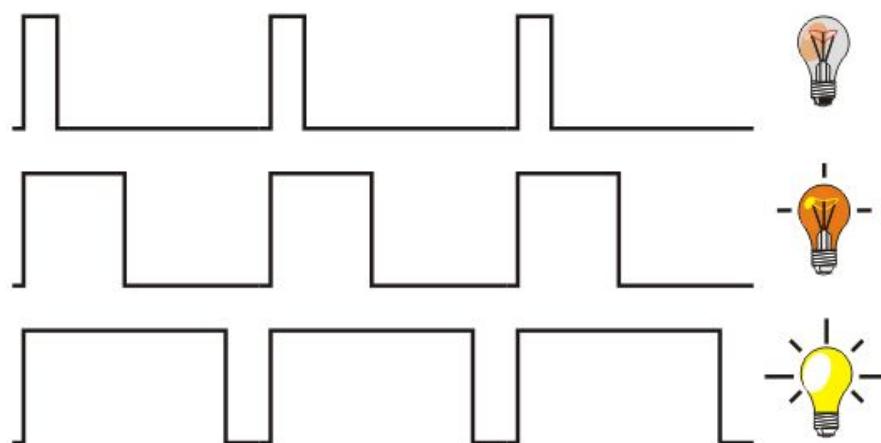
Valor	Duty Cycle
0	0
255	100

Salidas analógicas: PWM

PWM: modulación de ancho de pulso (Pulse Width Modulation). Vamos a hacer que determinados pulsos que enviamos a los dispositivos tengan una duración diferente, consiguiendo que la transferencia de energía sea mayor (en el caso en que el pulso sea más ancho) o menor. El resultado de aplicar señales PWM sobre dispositivos analógicos (como un LED) será que se consigue un mayor nivel de brillo a medida que el ancho del pulso (Duty Cycle) es mayor.

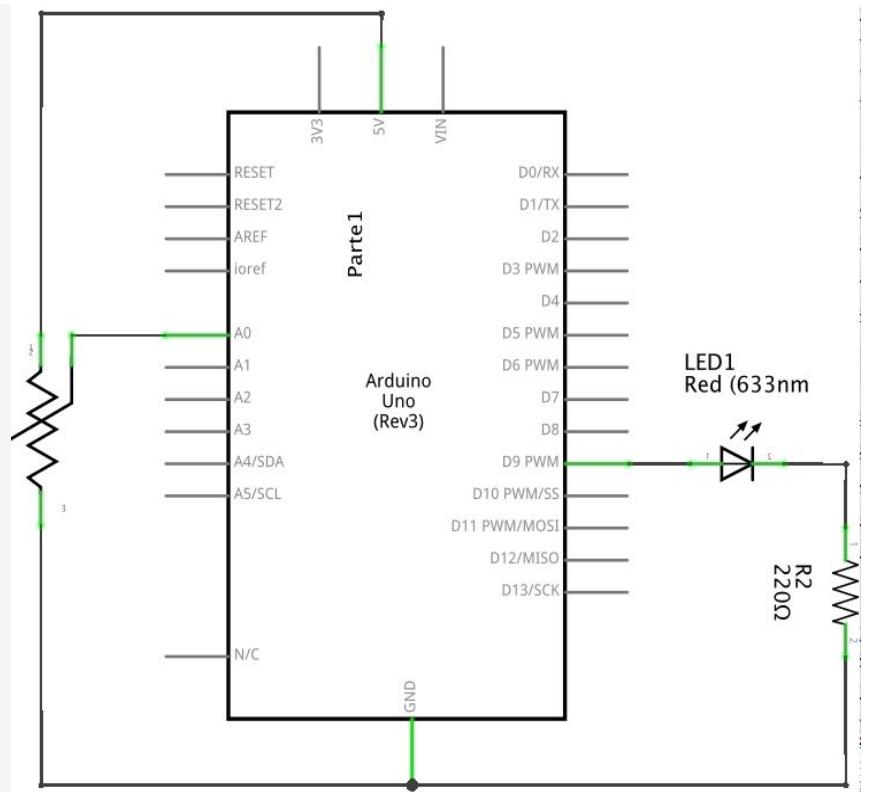
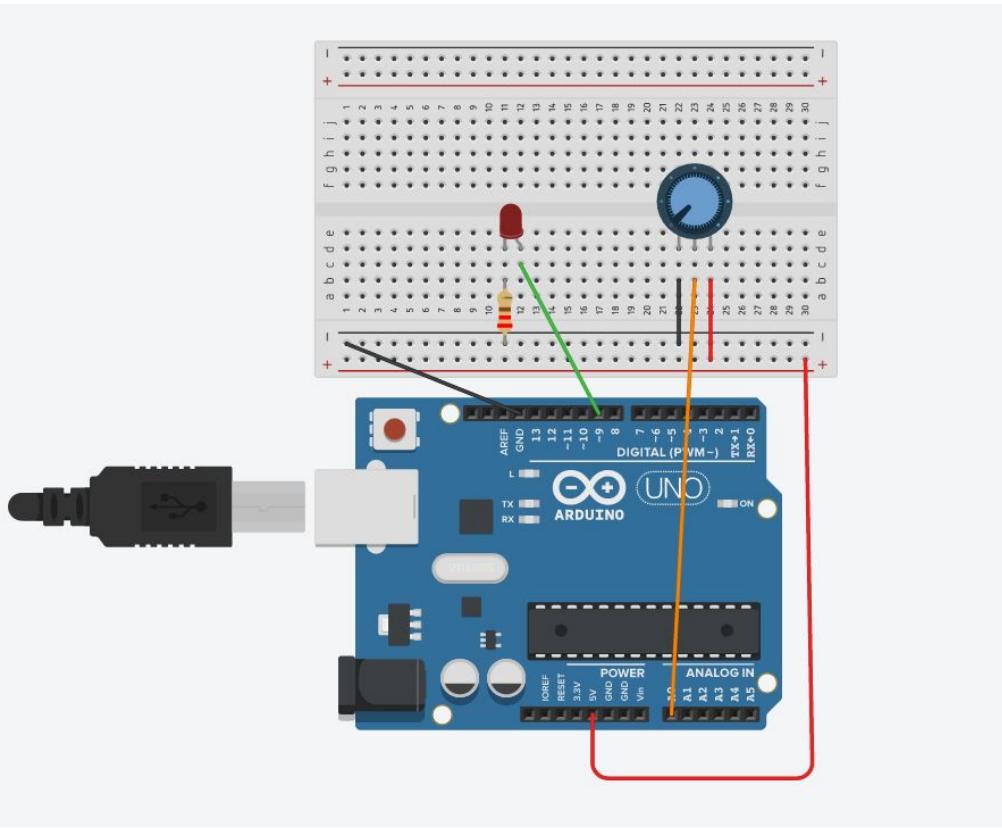


8 bits = $2^8 = 256$ niveles



Frecuencia en Arduino UNO: 490Hz

Hardware - Esquema



fritzing

Software

S1 Control LED función tiempo

Código Arduino S1

```
10 // Variables globales constantes
11 const int ledPin = 9; // LED conectado a Pin 9
12
13 void setup() {
14     // nada que declarar en el set up
15 }
16 void loop() {
17     // Encendido gradual del LED en 2,5s
18     for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
19         analogWrite(ledPin, fadeValue); // establece el valor de 0 a 255
20         delay(50); // espera de 50 ms
21     }
22     // Apagado gradual del LED en 2,5s
23     for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
24         analogWrite(ledPin, fadeValue); // establece el valor de 255 a 0
25         delay(50); // espera de 50 ms
26     }
27 }
```

Pues se trata de salidas analógicas.

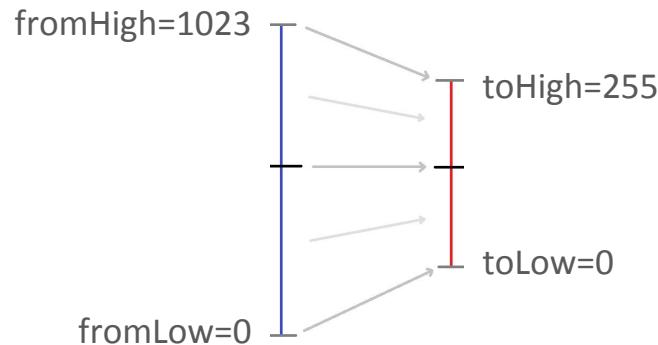
Dos bucles for: uno de subida y otro de bajada.

[Enlace al código](#)

Software

S2 Control LED función potenciómetro

Conceptos teóricos software: Función map



Escala una variable de un rango a otro
map(value; fromLow; fromHigh; toLow; toHigh);

value | variable a escalar

fromLow | valor inferior de la variable a escalar

fromHigh | valor superior de la variable a escalar

toLow | valor inferior de la variable escalada

toHigh | valor superior de la variable escalada

Ejemplo:

```
sensorMapvalue = map(sensorValue; 0; 1023; 0; 255);
```

Código Arduino S2

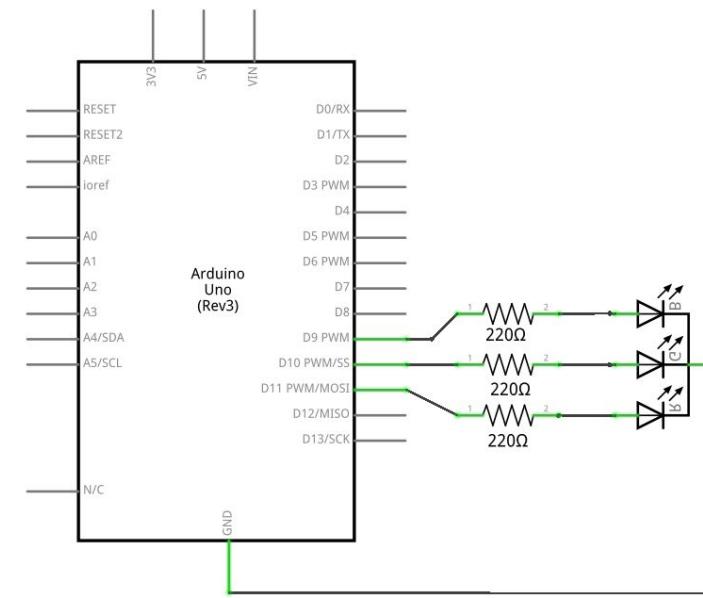
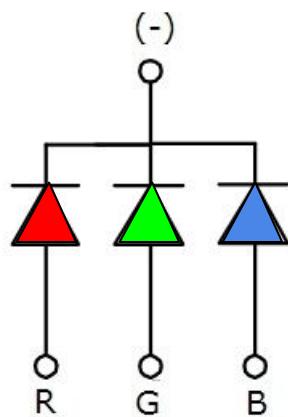
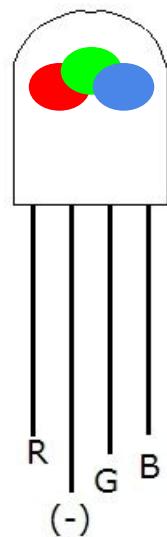
```
5 // Variables globales
6 // Variables constantes para los pines
7 const int potPin = A0; // selecciona el pin de entrada para el potenciómetro
8 const int ledPin = 9; // selecciona pin 9 para el LED
9
10 // variables que almacenan los datos
11 int potValue = 0; // variable para almacenar los valores del potenciómetro 0-1023
12 int bright = 0; // variable que almacena el valor del sensor mapeado 0-255
13
14 void setup() {
15     // nada que declarar en el setup
16 }
17
18 void loop() {
19     // lee el valor del sensor 0-1023
20     potValue = analogRead(potPin);
21     // mapea el valor a escala 0-255
22     bright = map(potValue, 0, 1023, 0, 255);
23     // escribimos el valor en analogico en el LED
24     analogWrite(ledPin, bright);
25 }
```

[Enlace al código](#)

Propuesta de ampliación

Controlar el encendido de un LED RGB

Common
Cathode (-)



fritzing

Autoría y Licencia

Esta presentación es obra de José Pujol

Ha sido realizada a partir del curso “The Tech Project: Arduino en el aula” cuyos contenidos fueron realizados por Jose Antonio Vacas y José Pujol, la dirección del curso la realizó avante.es

Los contenidos se distribuyen bajo Licencia Reconocimiento- Compartir Igual Creative Commons 4.0

