

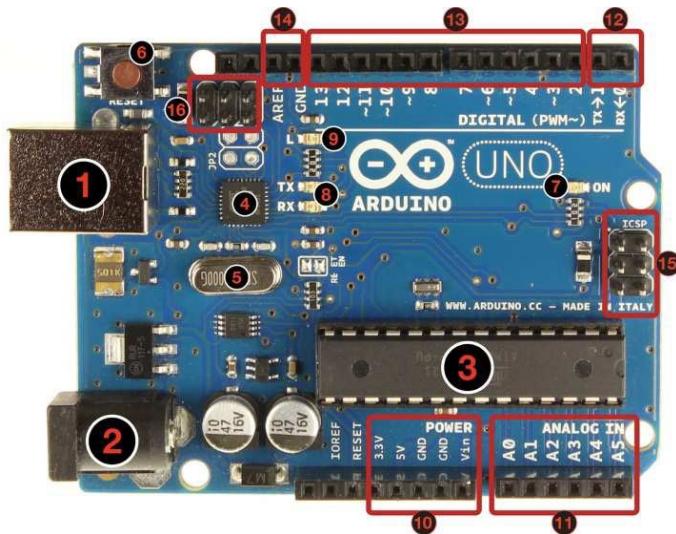
## Desarrollo de aplicaciones con Arduino

### 1 – Qué es la placa controladora Arduino y para qué sirve

Arduino es una plataforma de prototipado electrónico de código y hardware abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede “sentir” el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el “Arduino Programming Language” (basado en Wiring) y el “Arduino Development Environment” (basado en Processing). Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con Flash, Processing, MaxMSP, etc.).

Las placas se pueden ensamblar a mano o encargarlas preensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.



1 - Conexión USB

2 - Alimentación

3 - Microcontrolador ATmega328

4 - Atmega18U2

5 - Oscilador de cuarzo 16MHz

6 - Botón de reset

7 - LED de encendido

8 - LEDs de comunicación

9 - LED pin 13

10 - Pines de alimentación

11 - Entradas analógicas

12 - Pines de conexión serie

13 - E/S digitales

14 - Pin de referencia analógica y pin GND

15 - Pines para programación serie (In Circuit Serial Programming)

### Alimentación de la placa

El Arduino UNO puede ser alimentado a través de la conexión USB o con un suministro de energía externo. La alimentación externa (no USB) puede venir o desde un adaptador AC/DC o desde una batería. El adaptador puede ser conectado mediante un enchufe centro-positivo en el conector de alimentación de la placa. Los cables de la batería pueden insertarse en las cabeceras de los pines Gnd y Vin del conector POWER. Un regulador de bajo abandono proporciona eficiencia energética mejorada.

La placa puede operar con un suministro externo de 6 a 20 voltios. Sin embargo, si es suministrada con menos de 7 V el pin de 5 V puede que suministre menos de 5 V y la placa podría ser inestable. Si usa más de 12 V, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

**VIN.** La entrada de tensión a la placa Arduino cuando se está usando una fuente de alimentación externa (al contrario de los 5 voltios de la conexión USB u otra fuente de alimentación regulada). Puedes suministrar tensión a través de este pin, o, si suministra tensión a través del conector de alimentación, acceder a él a través de este pin.

**5V.** El suministro regulado de energía usado para alimentar al microcontrolador y otros componentes de la placa. Este puede venir o desde VIN a través de un regulador en la placa, o ser suministrado por USB u otro suministro regulado de 5V.

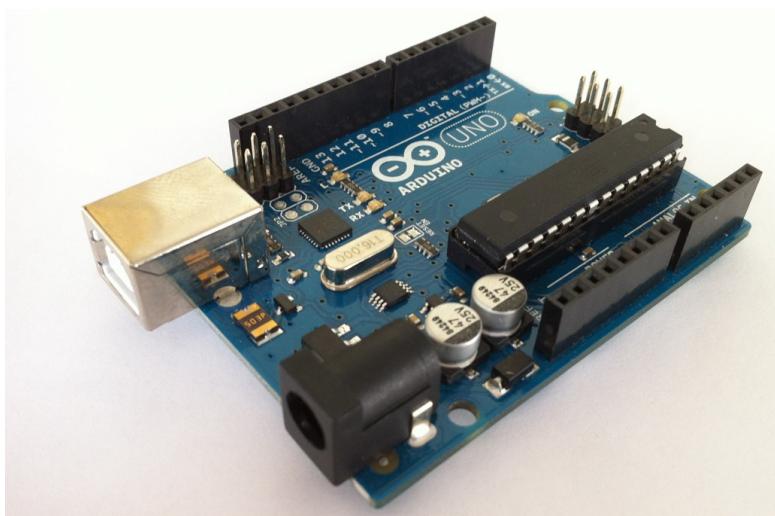
**3,3V.** Un suministro de 3.3 V generado por el chip FTDI de la placa. La corriente máxima es de 50 mA.

### Memoria

El ATmega328 tiene 32 KB de memoria Flash para almacenar código (de los cuales 0.5 KB se usan para el bootloader). Tiene 2 KB de SRAM y 1 KB de EEPROM (que puede ser leída y escrita con la librería EEPROM).

## Desarrollo de aplicaciones con Arduino

Característica	Descripción
Microcontrolador	Atmega 328
Voltaje de operación	5V
Voltaje de entrada (recomendado)	7V - 12V
Voltaje de entrada (límites)	6V - 20V
E/S digitales	14 de las cuales 6 proveen salidas PWM
Entradas analógicas	6
Corriente DC E/S digitales (por pin)	40 mA
Corriente DC para salida 3.3V	50mA
Memoria FLASH	32 KB
Memoria SRAM	2KB
EEPROM	1KB
Velocidad de reloj	16MHz



### Entradas y salidas

Cada uno de los 14 pines digitales del UNO puede ser usado como entrada o salida, usando funciones pinMode(), digitalWrite() y digitalRead(). Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna “pull-up” (desconectada por defecto y que se puede activar mediante software) de 20-50 Kohms. Además, algunos pines tienen funciones especiales:

Serial: 0 (Rx) y 1 (Tx). Usados para recibir (Rx) y transmitir (Tx) datos TTL en serie. Estos pines están conectados a los pines correspondientes del chip FTDI USB-a-TTL Serie.

Interruptores externos: 2 y 3. Estos pines pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. Mirar la función attachInterrup().

PWM: 3, 5, 6, 9, 10 y 11. Proporcionan salida PWM de 8 bits con la función analogWrite().

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines soportan comunicación SPI, la cual, aunque proporcionada por el hardware subyacente, no está actualmente incluida en el lenguaje Arduino.

I2C/TWI: Se puede comunicar arduino mediante I2C/TWI. El Arduino Due tiene dos interfaces I2C / TWI SDA1 y SCL1 que están cerca del pin AREF y los adicionales en los pines 20 y 21. En el Arduino Uno los pines I2C son los que están a continuación de AREF, Pin 16(SDA) y 17(SCL) y en el Mega son el 20 (SDA) y el 21 (SCL).

Para comunicar con este protocolo es necesaria la librería Wire.

LED: 13. Hay un LED empotrado conectado al pin digital 13. Cuando el pin está a valor HIGH, el LED está encendido, cuando el pin está a LOW, está apagado.

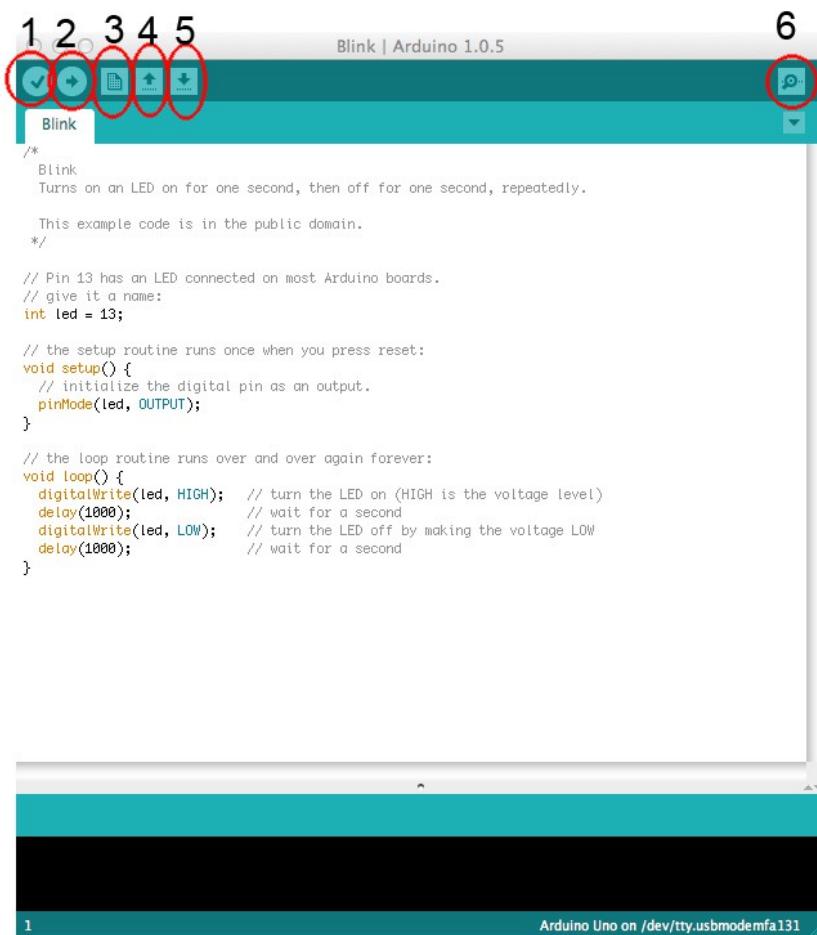
El UNO tiene 6 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (por ejemplo 1024 valores diferentes). Por defecto miden 5 voltios desde tierra, aunque es posible cambiar el valor más alto de su rango usando el pin AREF y algún código de bajo nivel.

Además, algunos pines tienen funcionalidad especializada:

- AREF. Voltaje de referencia para las entradas analógicas. Usado con analogReference().
- Reset. Pone esta linea a LOW para resetear el microcontrolador. Típicamente usada para añadir un botón de reset a dispositivos que bloquean a la placa principal.

### 2 – El IDE de Arduino

El IDE (Integrated Development Environment) es un programa instalado en tu ordenador que te permite escribir código para que lo ejecute el microcontrolador. Está basado en processing y por eso su interfaz es muy parecido. El IDE se descarga desde la página de Arduino en internet, [wwwarduino.cc](http://wwwarduino.cc)



1. Verifica/Compila el código

2. Compila y carga el código en la placa

3. Nuevo programa

4. Abre la carpeta donde están nuestros programas

5. Guarda el programa

6. Abre el Serial monitor

#### 2.1 Menús

Sketch

- Verify/Compile: Comprueba tu rutina para errores.
- Import Library: Utiliza una librería en tu rutina. Trabaja añadiendo #include en la cima de tu código. Esto añade funcionalidad extra a tu rutina, pero incrementa su tamaño. Para parar de usar una librería, elimina el #include apropiado de la cima de tu rutina.

## Desarrollo de aplicaciones con Arduino

- Show Sketch Folder: Abre la carpeta de rutinas en tu escritorio.
- Add File... : Añade otro fichero fuente a la rutina. El nuevo archivo aparece en una nueva pestaña en la ventana de la rutina. Esto facilita y agranda proyectos con múltiples archivos fuente. Los archivos pueden ser eliminados de una rutina usando el Tab Menu.

### Tools

- Auto Format: Esto formatea tu código amigablemente.
- Copy for Discourse: Copia el código de tu rutina al portapapeles de forma conveniente para postear en un foro, completa con resaltado de sintaxis.
- Board: Selecciona la placa que estas usando. Esto controla la forma en que tu rutina es compilada y cargada así como el comportamiento de los elementos del menú Burn Bootloader.
- Serial Port: Este menú contiene todos los dispositivos serie (reales o virtuales) de tu máquina. Debería actualizarse automáticamente cada vez que abres el nivel superior del menú Tools. Antes de subir tu rutina, necesitas seleccionar el elemento de este menú que representa a tu placa Arduino. En el Mac, esto es probablemente algo como /dev/tty.usbserial-1B1 (para la placa USB), o /dev/tty.USA19QW1b1P1.1 (para una placa Serie conectada con un adaptador USB-a-Serie Keyspan). En Windows, es probablemente COM1 o COM2 (para una placa Serie) o COM4, COM5, COM7 o superior (para una placa USB) – para descubrirlo, busca USB serial device en la sección puertos del “Gestor de dispositivos de Windows”.
- Burn Bootloader: Los elementos en este menú te permiten grabar un bootloader en tu placa con una variedad de programadores. Esto no es necesario para uso normal de una placa Arduino, pero puede ser útil si encargas ATmegas adicionales o estás construyendo una placa por tu cuenta. Asegurate que has seleccionado la placa correcta del menú Boards de antemano. Para grabar un bootloader con el AVR ISP, necesitas seleccionar el elemento que corresponde a tu programador del menú Serial Port.

### 3 Control del encendido y apagado de leds.

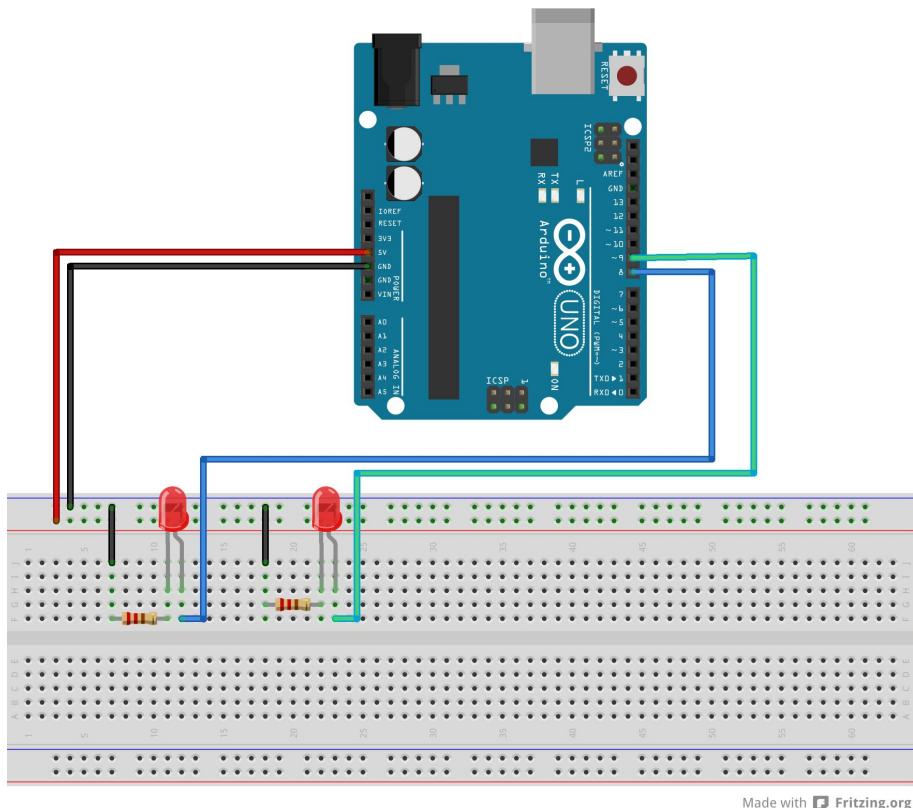
#### Práctica nº 1

Vamos a realizar una sencilla práctica en la que controlaremos el encendido y apagado de dos leds mediante la programación de un sketch en Arduino.

#### Necesitamos:

- 1 arduino
- 2 leds
- 2 resistencias de 220 Ω

#### Este es el circuito a realizar:



### El código:

```
/*
Intermitencia
Enciende un LED durante medio segundo y lo apaga
al tiempo que enciende un segundo LED durante medio segundo y lo apaga.

*/
int led1 = 8; // creamos una variable que almacenará el pin al que se conecta el led1
int led2 = 9; // creamos una variable que almacenará el pin al que se conecta el led2

void setup()
{
    pinMode(led1, OUTPUT); // declaramos el pin led1 como OUTPUT
    pinMode(led2, OUTPUT); // declaramos el pin led como OUTPUT
}

void loop()
{
    digitalWrite(led1, HIGH); // enciende el LED 1
    delay(500); // espera durante medio segundo
    digitalWrite(led1, LOW); // apaga el LED 1
    digitalWrite(led2, HIGH);
    delay(500);
    digitalWrite(led2, LOW);
}
```

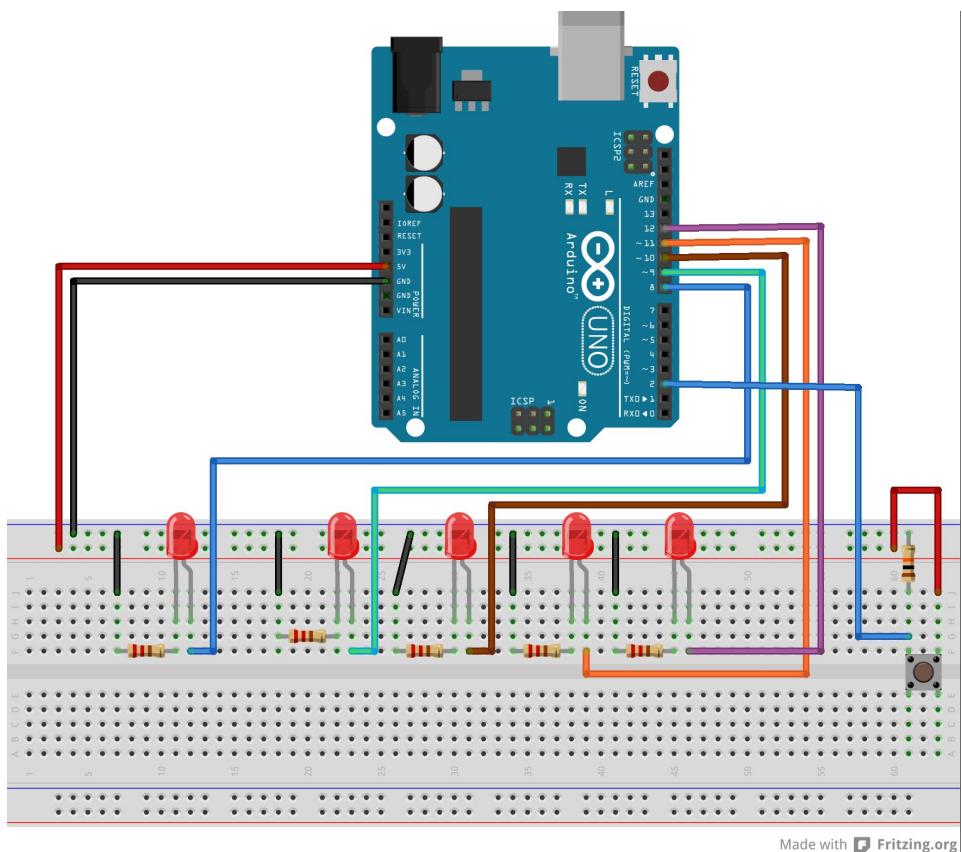
### Práctica nº 2

Vamos a añadir 3 leds más a la práctica anterior y crearemos combinaciones de intermitencias que serán controladas por un pulsador.

#### Necesitamos:

- 1 arduino
- 5 leds
- 5 resistencias de  $220\ \Omega$ , 1 resistencia de  $10K\ \Omega$
- 1 pulsador

**Realizar el siguiente circuito:**



### El código:

```
/*
Intermitencia
Enciende un LED durante medio segundo y lo apaga. Repite la accion en los 5 leds
*/
int ledPin [] = {8, 9, 10, 11, 12};
int temps = 500;
int botoPin = 7;
int boto = 0;

void setup()
{
    for (int i = 0; i < 5; i++)
    {
        pinMode(ledPin[i], OUTPUT);
    }
    pinMode(botoPin, INPUT);

    for (int i = 0; i < 5; i++)
    {
        digitalWrite(ledPin[i], LOW);
    }
}

void loop()
{
    boto = digitalRead(botoPin);
    if(boto == 1)
    {
        secuencia();
    }else{
        for (int i = 0; i < 5; i++)
        {
            digitalWrite(ledPin[i], LOW);
        }
    }
}
```

```

        }
    }
}

void secuencia()
{
    for(int i = 0; i < 5; i++)
    {
        digitalWrite(ledPin[i], HIGH);
        delay(tempo);
        digitalWrite(ledPin[i], LOW);
    }
}

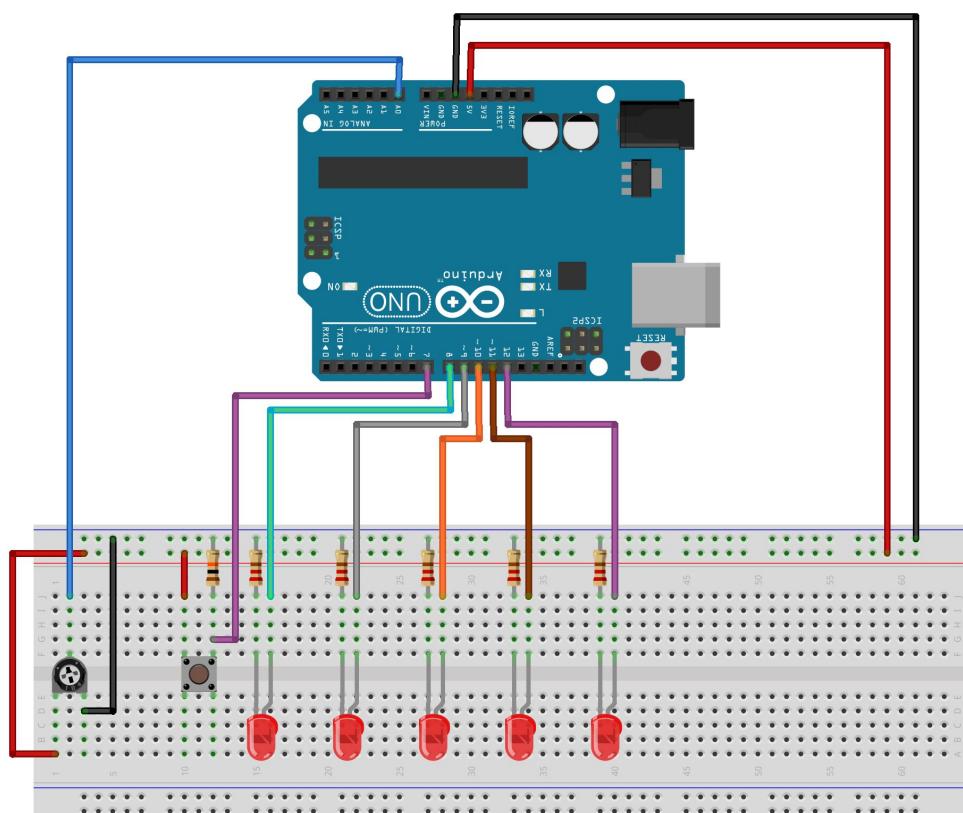
} //Fin del código

```

### Práctica nº 3

Siguiendo con las prácticas anteriores vamos a mejorar el código y además implantar un potenciómetro para poder ajustar manualmente la velocidad del apagado y encendido de los leds. El pulsador ahora funcionará como un interruptor. Cuando se pulsa se encenderán los leds y seguirán encendidos hasta que se vuelva a pulsar.

**Realizar el siguiente circuito:**



### El código:

```
/*
Intermitencia
Enciende un LED durante medio segundo y lo apaga. Repite la accion en los 5 leds
Implementamos un potenciómetro para ajustar la velocidad y ampliamos el c'digo para que
el pulsador funcione como un interruptor.

*/
int ledPin [] = {8, 9, 10, 11, 12};
int temps = 500;
int botoPin = 7;
int boto = 0;
int botoAntes = 0;
int pot = A0;
int estado = 0;

void setup()
{
    for (int i = 0; i < 5; i++)
    {
        pinMode(ledPin[i], OUTPUT);
    }
    pinMode(botoPin, INPUT);

    for (int i = 0; i < 5; i++)
    {
        digitalWrite(ledPin[i], LOW);
    }
}

void loop()
{
    boto = digitalRead(botoPin);
    temps = (analogRead(pot))/4;
```

```
if((boto == HIGH) && (botoAntes == LOW))
{
    estado = 1 - estado;
    delay(10);
}
botoAntes = boto;

if(estado == 1)
{
    secuencia();
}else
{
    for (int i = 0; i < 5; i++)
    {
        digitalWrite(ledPin[i], LOW);
    }
}
void secuencia(){
    for(int i = 0; i < 5; i++)
    {
        digitalWrite(ledPin[i], HIGH);
        delay(temp);
        digitalWrite(ledPin[i], LOW);
    }
}

} // Fin del código
```

### 4 – Entradas analógicas del Arduino

El Arduino dispone de 6 entradas analógicas que son las nombradas como A0, A1, A2, A3, A4 y A5 . Mientras que con las entradas digitales solo podemos obtener dos datos, (encendido o apagado) con las entradas analógicas podemos obtener multitud de datos. (entre un apagado y un encendido pueden haber “infinitos” estados intermedios) De todas formas, para que esos datos nos sean útiles hay que cuantificar los valores, es decir, hay que hacer una conversión de un valor analógico continuo a una representación finita. El Arduino dispone de un conversor analógico-digital de 10 bits de resolución. Esto quiere decir que de un continuo analógico podemos obtener un valor entre 0 a 1023.

Por ello, las entradas analógicas van a ser, normalmente, las utilizadas para conectar sensores.

#### 4.1 Los sensores

Un sensor es un dispositivo capaz de transformar un parámetro físico en una variable que se puede medir. La salida de un sensor habitualmente nos genera:

- Una diferencia de potencial
- Resistencia
- Anchura de pulso

Podemos distinguir dos grandes tipos de sensores:

##### Sensores binarios

También conocidos como “sensores digitales”. Proporcionan dos estados de información (abierto – cerrado) Por tanto se conectarán a una entrada digital y esencialmente son de tipo mecánico como el Pulsador, Interruptor, Relé Reed, Final de carrera...

##### Sensores continuos

También conocidos como “sensores analógicos”. Proporcionan información continua dentro de un rango y se conectan a una entrada analógica. Los más populares son de tipo resistivo, aunque hay una gran variedad como potenciómetros, LDR, NTC, Termopar...

## Desarrollo de aplicaciones con Arduino

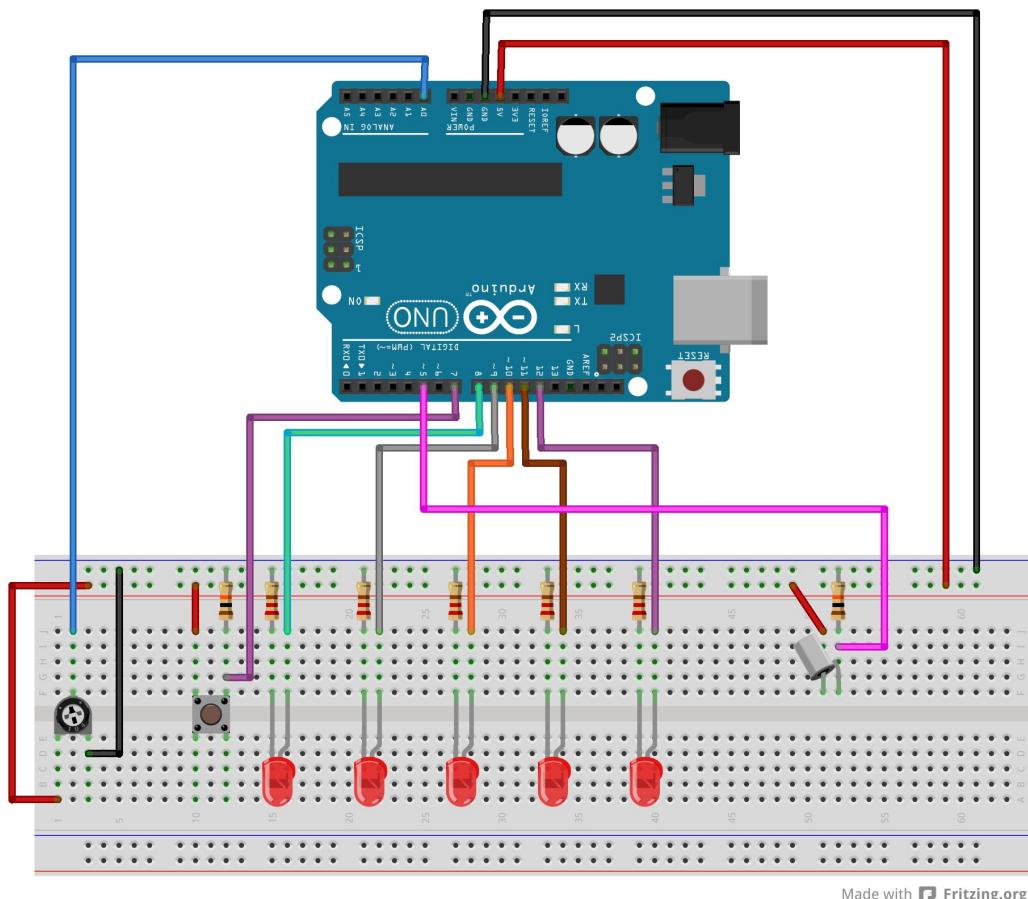
### Práctica nº 5

Siguiendo con las prácticas anteriores vamos a introducir un “sensor digital” para que sea este el que nos haga cambiar de sentido la dirección de la luz de los leds. Usaremos un sensor Tilt para detectar la inclinación de la protoboard.

#### Necesitamos:

- 1 arduino
- 5 leds
- 5 resistencias de  $220\ \Omega$
- 2 resistencias de  $10K\ \Omega$
- 1 pulsador
- 1 potenciómetro de  $10K\Omega$
- 1 sensor Tilt

#### Realizar el siguiente circuito:



Made with Fritzing.org

### El código:

```
/*
Intermitencia
Enciende un LED durante medio segundo y lo apaga. Repite la accion en los 5 leds
Implementamos un potenciómetro para ajustar la velocidad y ampliamos el c'digo para que
el pulsador funcione como un interruptor.
Añadimos un pulsador para cambiar de dirección el sentido de la luz
*/
```

```
int ledPin [] = {8, 9, 10, 11, 12};
int temps = 500;
int botoPin = 7;
int boto = 0;
int botoAntes = 0;
int pot = A0;
int estado = 0;
int sensorTilt = 5;

void setup() {

    for (int i = 0; i < 5; i++){
        pinMode(ledPin[i], OUTPUT);
    }
    pinMode(botoPin, INPUT);

    for (int i = 0; i < 5; i++){
        digitalWrite(ledPin[i], LOW);
    }
}

void loop() {
    boto = digitalRead(botoPin);
    temps = (analogRead(pot))/4;
```

```
if((boto == HIGH) && (botoAntes == LOW)){
    estado = 1 - estado;
    delay(10);
}

botoAntes = boto;

if(estado == 1){
    boolean golpe = digitalRead(sensorTilt);
    if (golpe == false){
        secuencialInv();
    }else{
        secuencia();
    }
}else{
    for (int i = 0; i < 5; i++){
        digitalWrite(ledPin[i], LOW);
    }
}
}

void secuencia(){
for(int i = 0; i < 5; i++){
    digitalWrite(ledPin[i], HIGH);
    delay(tempo);
    digitalWrite(ledPin[i], LOW);
}
}

void secuencialInv(){
for(int i = 5; i >= 0; i--){
    digitalWrite(ledPin[i], HIGH);
    delay(tempo);
    digitalWrite(ledPin[i], LOW);
}
}
```

### Práctica nº 6 – Mejorando el código

Si observamos el funcionamiento de las prácticas anteriores, podemos observar que cuando pulsamos el pulsador el cambio tarda en producirse si nos encontramos a mitad de secuencia. Hay que esperar a que termine para que tenga efecto el cambio. Esto es debido al uso de la función `delay()` que interrumpe la ejecución de código. Si pulsamos un pulsador mientras el Arduino está en `delay` no puede detectar el cambio realizado. Por este motivo, no es recomendable el uso de la función `delay()`. En su caso, es mejor utilizar otras funciones como `millis()`. En la próxima práctica realizamos el mismo programa pero utilizando la función `millis()` en vez de `delay()`

### El código mejorado con el uso de la función `millis()` en vez de `delay()`

```
/*
```

El mismo código de la práctica 5 pero implementando la función `millis()` y descartando el uso de `delay()`

```
*/
```

```
int ledPin [] = {8, 9, 10, 11, 12};  
const int pot = A0;  
const int botoPin = 7;  
  
int boto = 0;  
int botoAntes = 0;  
int temps = 0;  
int estado = 0;  
int sensorTilt = 5;  
  
boolean secuenciaCompleta = false;  
unsigned long tiempo;  
  
void setup(){  
    for (int i = 0; i < 5; i++){  
        pinMode(ledPin[i], OUTPUT);  
    }  
    pinMode(botoPin, INPUT);  
    tiempo = millis();  
}
```

```
void loop(){

    boto = digitalRead(botoPin);
    temps = (analogRead(pot))/4;

    if((boto == HIGH) && (botoAntes == LOW)){
        estado = 1 - estado;
        delay(10);
    }
    botoAntes = boto;

    if(estado == 1){

        boolean golpe = digitalRead(sensorTilt);
        if (golpe == false){

            for (int i = 0; i < 5; i++){
                digitalWrite(ledPin[i], LOW);
            }
            secuenciaInv();
            if(secuenciaCompleta == true){

                tiempo = millis();
            }
        }else{

            for (int i = 0; i < 5; i++){

                digitalWrite(ledPin[i], LOW);
            }
            secuencia();
            if(secuenciaCompleta == true){

                tiempo = millis();
            }
        }
    }

}else{

    for (int i = 0; i < 5; i++){

        digitalWrite(ledPin[i], LOW);
    }
}
```

## Desarrollo de aplicaciones con Arduino

```

        }
    }

void secuencia(){
    if(millis()-tiempo < (tempo * 1)){
        secuenciaCompleta = false;
        digitalWrite(ledPin[0], HIGH);
    }else{
        if(millis()-tiempo > (tempo * 1)){
            digitalWrite(ledPin[0], LOW);
        }
    }
    if((millis()-tiempo < (tempo * 2)) && (millis()-tiempo > (tempo * 1))){
        digitalWrite(ledPin[1], HIGH);
    }else{
        if(millis()-tiempo > (tempo * 2)){
            digitalWrite(ledPin[1], LOW);
        }
    }
    if((millis()-tiempo < (tempo * 3)) && (millis()-tiempo > (tempo * 2))){
        digitalWrite(ledPin[2], HIGH);
    }else{
        if(millis()-tiempo > (tempo * 3)){
            digitalWrite(ledPin[2], LOW);
        }
    }
    if((millis()-tiempo < (tempo * 4)) && (millis()-tiempo > (tempo * 3))){
        digitalWrite(ledPin[3], HIGH);
    }else{
        if(millis()-tiempo > (tempo * 4)){
            digitalWrite(ledPin[3], LOW);
        }
    }
    if((millis()-tiempo < (tempo * 5)) && (millis()-tiempo > (tempo * 4))){
        digitalWrite(ledPin[4], HIGH);
    }else{

```

## Desarrollo de aplicaciones con Arduino

```

if(millis()-tiempo > (temps * 5)){
    digitalWrite(ledPin[4], LOW);
    secuenciaCompleta = true;
}
}

void secuencialInv(){
    if(millis()-tiempo < (temps * 1)){
        secuenciaCompleta = false;
        digitalWrite(ledPin[4], HIGH);
    }else{
        if(millis()-tiempo > (temps * 1)){
            digitalWrite(ledPin[4], LOW);
        }
    }

    if((millis()-tiempo < (temps * 2)) && (millis()-tiempo > (temps * 1))){
        digitalWrite(ledPin[3], HIGH);
    }else{
        if(millis()-tiempo > (temps * 2)){
            digitalWrite(ledPin[3], LOW);
        }
    }

    if((millis()-tiempo < (temps * 3)) && (millis()-tiempo > (temps * 2))){
        digitalWrite(ledPin[2], HIGH);
    }else{
        if(millis()-tiempo > (temps * 3)){
            digitalWrite(ledPin[2], LOW);
        }
    }

    if((millis()-tiempo < (temps * 4)) && (millis()-tiempo > (temps * 3))){
        digitalWrite(ledPin[1], HIGH);
    }else{
        if(millis()-tiempo > (temps * 4)){
            digitalWrite(ledPin[1], LOW);
        }
    }
}

```

## Desarrollo de aplicaciones con Arduino

```
}

if((millis()-tiempo < (tempo * 5)) && (millis()-tiempo > (tempo * 4))){
    digitalWrite(ledPin[0], HIGH);
}else{
    if(millis()-tiempo > (tempo * 5)){
        digitalWrite(ledPin[0], LOW);
        secuenciaCompleta = true;
    }
}
}
```

### 5 - Arduino digital: Control de entradas y salidas digitales con Arduino

El Arduino dispone de 14 pines digitales que pueden ser configurados como entradas o como salidas a través de la función pinMode()

El Atmega que utiliza Arduino tiene los pines configurados por defecto como Inputs, así que, si los vas a usar como inputs no hace falta configurarlos mediante pinMode(). Los pines configurados como inputs se dice que están en un estado de alta impedancia. Por tanto, si estos no están conectados a nada, pueden presentar cambios aleatorios dependiendo de otros factores (ruido eléctrico).

Hay una resistencia de 20K conectada en pull-up dentro de cada pin del chip Atmega que se puede activar mediante software. Se puede activar mediante la función pinMode() escribiendo en vez de INPUT, INPUT\_PULLUP. Con este modo se invierte el comportamiento lógico del pin y HIGH significa que el sensor está apagado y LOW que está encendido. Es decir, si conectamos un interruptor, un conector va al pin y el otro va a tierra, cuando el interruptor esté abierto el valor en el pin será HIGH y cuando esté cerrado será LOW.

Hay que tener en cuenta que el pin 13 tiene, en muchas placas Arduino, una resistencia y un led conectados en serie. Si activamos el INPUT\_PULLUP la tensión será menor de lo esperado. Unos 1,7 v. en vez de 5v.

Los pines que configuramos como OUTPUT a través de la función pinMode(), se dice que están en un estado de baja impedancia y por tanto pueden suministrar una cantidad importante de corriente a otros dispositivos. No obstante, la corriente que pueden suministrar es de 40 mA como máximo. Esto puede ser suficiente para iluminar un led (con su correspondiente resistencia) pero no para hacer mover un motor de corriente continua. No tener esto en cuenta puede destruir el pin, o incluso el chip Atmega.

Una vez configurados los pines como INPUT, OUTPUT o INPUT\_PULL\_UP podemos cambiar su estado o leerlo mediante el uso de funciones como digitalWrite() y digitalRead(), cambiando sus parámetros de HIGH a LOW y viceversa.

#### 5.1 – PWM Modulación por ancho de pulso

Una forma de utilizar una salida o entrada digital para emular un comportamiento analógico es la modulación por ancho de pulso que se conoce por sus siglas en inglés PWM (Pulse Width Modulation)

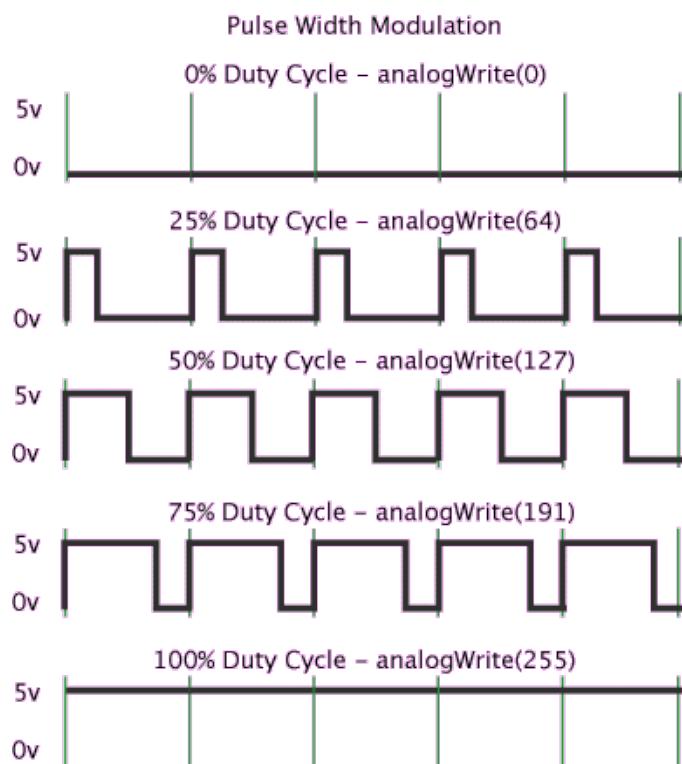
El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado.

A la duración del tiempo de encendido (ON) se le llama Ancho de Pulso (pulse width). Para variar el valor analógico cambiamos, o modulamos, ese ancho de pulso. Si repetimos este patrón de encendido-apagado lo suficientemente rápido, por ejemplo con un LED, el resultado es como si la señal variara entre 0 y 5 voltios y podríamos controlar el brillo del LED.

## Desarrollo de aplicaciones con Arduino

En el gráfico de abajo, las líneas verdes representan un periodo regular. Esta duración o periodo es la inversa de la frecuencia del PWM. En otras palabras, con el Arduino, la frecuencia PWM es bastante próxima a 500Hz lo que equivale a períodos de 2 milisegundos cada uno.

La llamada a la función `analogWrite()` debe ser en la escala desde 0 a 255, siendo 255 el 100% de ciclo (siempre encendido), el valor 127 será el 50% del ciclo (la mitad del tiempo encendido), etc.



Hay que tener en cuenta que el chip Atmega 328 del Arduino tiene 3 timers para poder modular la señal, llamados Timer 0, Timer 1 y Timer 2. Cada timer funciona a la misma frecuencia.

El Timer 0 controla los pines 5 y 6 en modo Fast PWM y funciona a casi 1KHz

El Timer 1 controla los pines 9 y 10 en modo phase-correct a casi 500Hz

El Timer 2 controla los pines 11 y 3 en modo phase-correct a casi 500Hz

## Desarrollo de aplicaciones con Arduino

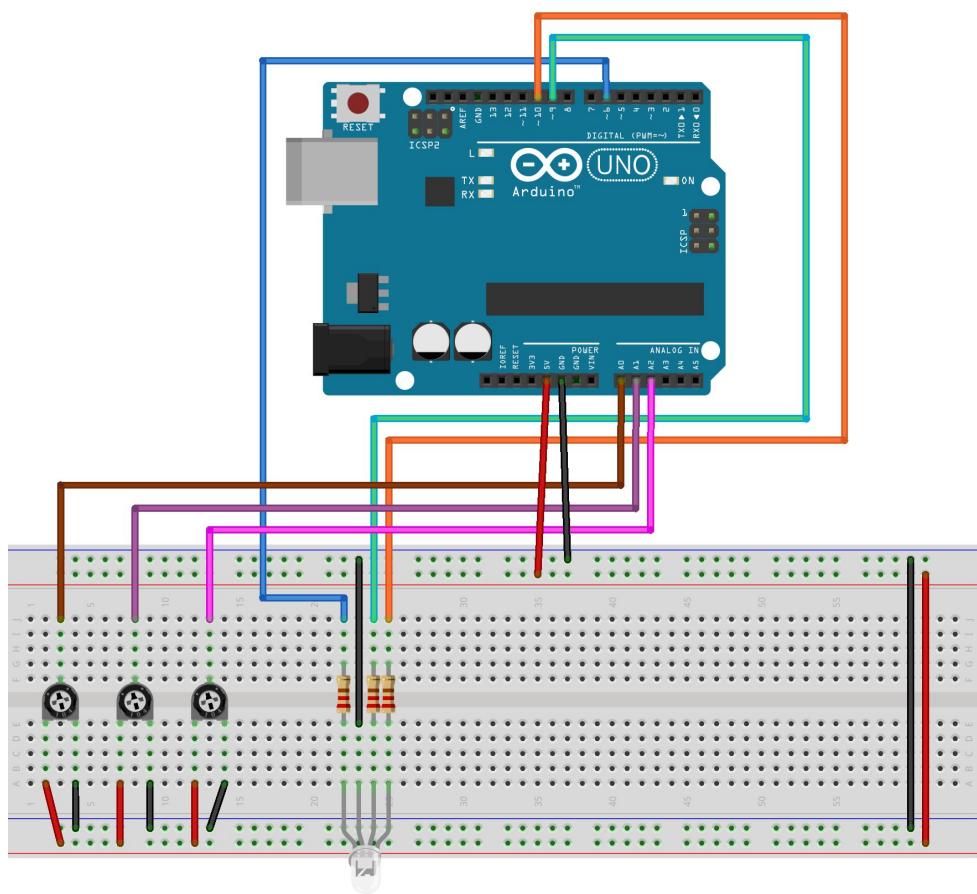
### Práctica nº 7

Entre las muchas aplicaciones que podemos tener con el uso de PWM hay una muy utilizada que es la de aumentar o disminuir la luminosidad de un Led. Pero si en vez de utilizar un Led normal usamos un led RGB, al aumentar o disminuir la luminosidad de cada componente RGB obtendremos cambios de color. Para realizar esas variaciones usaremos tres potenciómetros de 10KΩ

#### Necesitamos:

- 1 arduino
- 1 Led RGB
- 3 resistencias de 220 Ω
- 3 potenciómetros de 10KΩ

Montamos el siguiente circuito:



### El código de la práctica 7

```
/*
Control del color emitido por un led RGB mediante tres potenciómetros
que controlan la componente RGB del Led independientemente.

*/
int red = 6;
int green = 10;
int blue = 9;

int valorRed;
int valorGreen;
int valorBlue;

const int pot1 = A0;
const int pot2 = A1;
const int pot3 = A2;

void setup(){

    pinMode(red, OUTPUT);
    pinMode(green, OUTPUT);
    pinMode(blue, OUTPUT);
}

void loop(){

    valorRed = map(analogRead(pot1),0,1023,0,255);
    valorGreen = map(analogRead(pot2),0,1023,0,255);
    valorBlue = map(analogRead(pot3),0,1023,0,255);

    analogWrite(red, valorRed);
    analogWrite(green, valorGreen);
    analogWrite(blue, valorBlue);

}
```

## Desarrollo de aplicaciones con Arduino

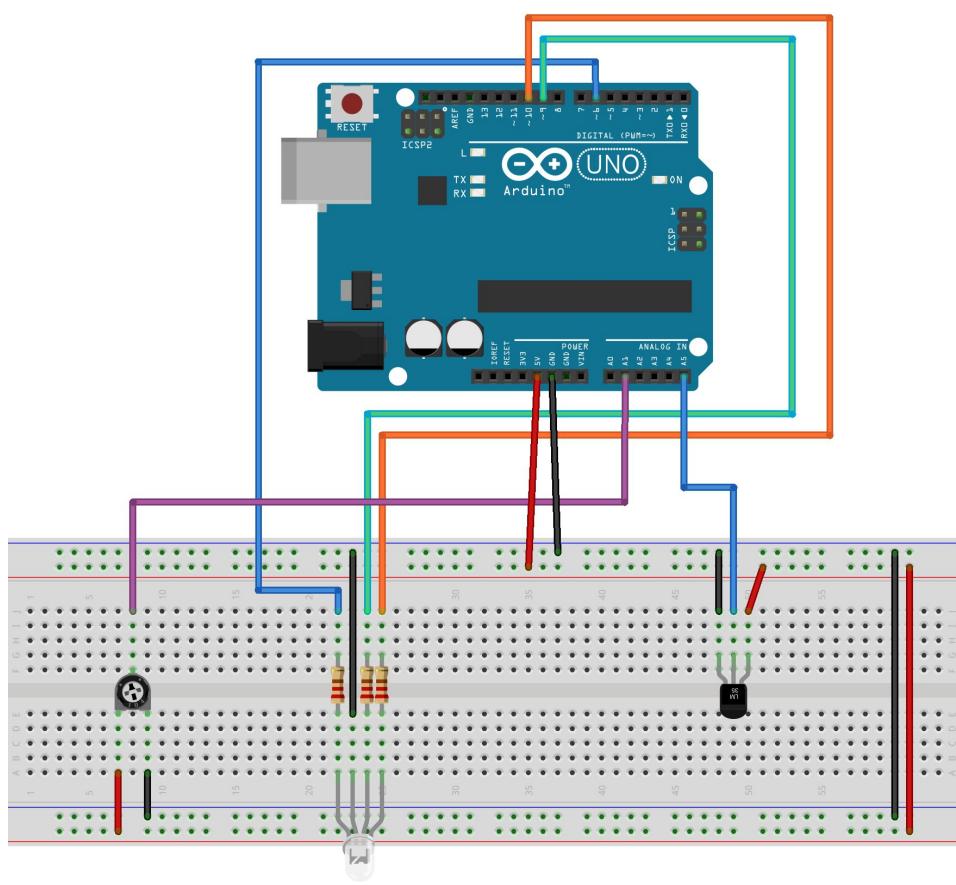
### Práctica nº 8

Vamos a incorporar a la práctica anterior un sensor de temperatura. El TEMP35 puede convertir variaciones de temperatura de 1 °C en variaciones de tensión de 10 milivoltios. Conectamos la salida del sensor a la entrada analógica A5 y hacemos que los cambios de color estén en relación a los cambios de temperatura.

#### Necesitamos:

- 1 arduino
- 1 Led RGB
- 3 resistencias de 220 Ω
- 1 potenciómetro de 10KΩ
- 1 sensor TEMP35

#### Montamos el siguiente circuito:



Made with Fritzing.org

### El código de la práctica 8

```
/*
Control del color emitido por un led RGB mediante los valores que registra
el sensor de temperatura TEMP35.

*/
int red = 6;
int green = 10;
int blue = 9;

int valorRed;
int valorGreen;
int valorBlue;

const int pot2 = A1;
const int tempPin= A5;
const float baseTemp = 26.00;

void setup(){
    Serial.begin(9600);
    pinMode(red, OUTPUT);
    pinMode(green, OUTPUT);
    pinMode(blue, OUTPUT);
}

void loop(){
    valorGreen = map(analogRead(pot2),0,1023,0,255);

    int tempVal = analogRead(tempPin);
    float voltage = (tempVal/1024.0)*5.0;
    float temperatura = (voltage - .5)* 100;

    int temp = temperatura;
    valorRed = map(temp,22,30,0,255);
    valorBlue = map(temp,22,30,255,0);

    analogWrite(red, valorRed);
    analogWrite(green, valorGreen);
    analogWrite(blue, valorBlue);
    delay(100);

    Serial.print("Valor del sensor: ");
    Serial.print(tempVal);
    Serial.print(" voltatge: ");
    Serial.print(voltage);
    Serial.print(" Temperatura: ");
    Serial.println(temp);

}
```

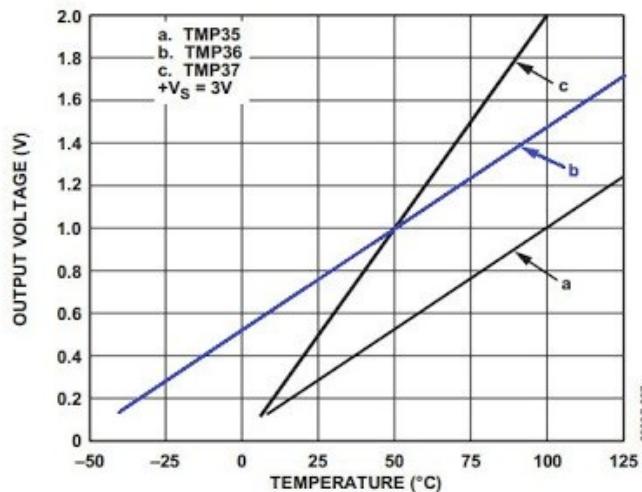
### El sensor TMP36:

Si miramos el datasheet del componente electrónico podemos observar que el TMP36 es un circuito integrado que actúa como un sensor de temperatura calibrado directamente en grados Celsius, que se alimenta entre 2.7V y 5.5V y que por lo tanto, es ideal para usarlo con la placa Arduino.

Proporciona una salida de voltaje directamente proporcional a la temperatura en grados Celsius y es muy parecido al clásico LM335A. Algunas características son:

- Rango de temperatura: -40°C to 150°C / -40°F to 302°F
- Factor de escala 10 mV/°C
- Precisión de ±2°C
- Linealidad de ±0.5°
- Alimentación: Entre 2.7 y 5.5V.

Remarquemos del datasheet sobretodo la curva Vout vs Temp:



**Una variación de 10 milivoltios supone una variación de un grado centígrado.**

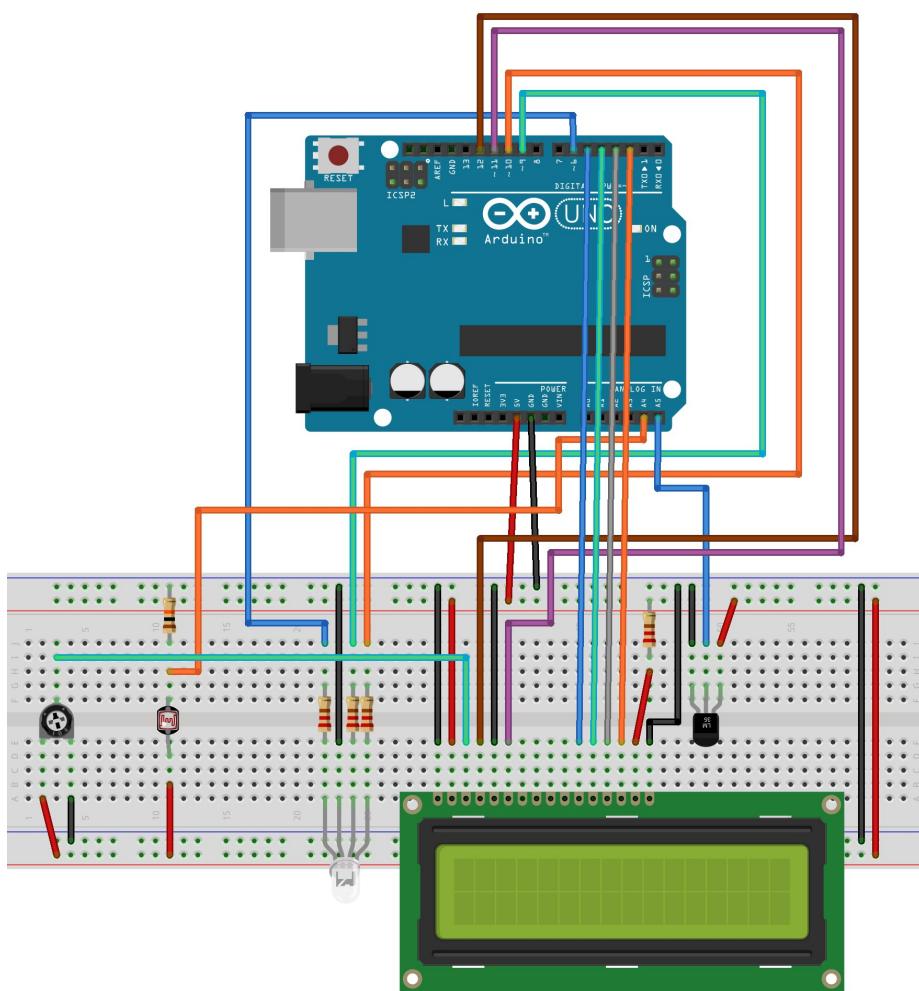
### Práctica nº 9

Para seguir aprendiendo con el uso de sensores, vamos a introducir una resistencia LDR que es capaz de variar su resistividad en función de la cantidad de luz que incide sobre ella. Además, incorporamos una pantalla LCD para visualizar los datos de luz y temperatura en su display.

#### Necesitamos:

- 1 arduino
- 1 Led RGB
- 3 resistencias de 220 Ω
- 1 potenciómetro de 10KΩ
- 1 sensor TEMP35
- 1 resistencia LDR
- 1 display LCD

**Montamos el siguiente circuito:**



Made with Fritzing.org

### El código a realizar:

```
/*
Control del color emitido por un led RGB mediante los valores de
temperatura que registra el sensor TEMP35. Instalamos una LCD para
visualizar datos de temperatura y sensacion luminica mediante una LDR

*/
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int rangoTemp=22;

int red = 6;
int green = 10;
int blue = 9;

int valorRed;
int valorGreen;
int valorBlue;

const int tempPin= A5;
const int sensorLlum = A0;
const float baseTemp = 26.00;

void setup(){
    pinMode(red, OUTPUT);
    pinMode(green, OUTPUT);
    pinMode(blue, OUTPUT);

    lcd.begin(16, 2);
    lcd.print("Temp: ");

}

void loop(){

    int tempPrint = medirTemp();
    int llumPrint = medirLlum();

    if(tempPrint < rangoTemp){
        analogWrite(red, 0);
        analogWrite(green, 150);
        analogWrite(blue, 255);
    }

    if(tempPrint > rangoTemp + 2){
        analogWrite(red, 150);
        analogWrite(green, 150);
        analogWrite(blue, 150);
    }
}
```

## Desarrollo de aplicaciones con Arduino

```
}

if(tempPrint > rangoTemp + 4){
    analogWrite(red, 255);
    analogWrite(green, 150);
    analogWrite(blue, 100);
}

if(tempPrint > rangoTemp + 6){
    analogWrite(red, 255);
    analogWrite(green, 0);
    analogWrite(blue, 0);
}

lcd.setCursor(7,0);
lcd.print(tempPrint);
lcd.setCursor(9,0);
lcd.print("C.");
lcd.setCursor(0,1);
lcd.print("Sens. Ilum:");
lcd.setCursor(12,1);
lcd.print(lumPrint);

}

float medirTemp(){

    int tempVal = analogRead(tempPin);
    float voltage = (tempVal/1024.0)*5.0;
    float temperatura = (voltage - .5)* 100;

    delay(400);
    return temperatura;
}

int medirLlum(){
    int llum = analogRead(sensorLlum)/10;
    return llum;
}
```

### 6 – Controlar un servo de rotación continua mediante un potenciómetro y Arduino

Con Arduino, no solo podemos recibir datos del entorno mediante los sensores, sino que también podemos “actuar” sobre el entorno mediante el control de “actuadores”. Con Arduino podemos manejar motores DC, servos, solenoides, motores PAP...

Empezamos con el control de un servo. Vamos a necesitar aprender el uso de librerías externas que nos ayudan con la programación.

#### Práctica nº 10

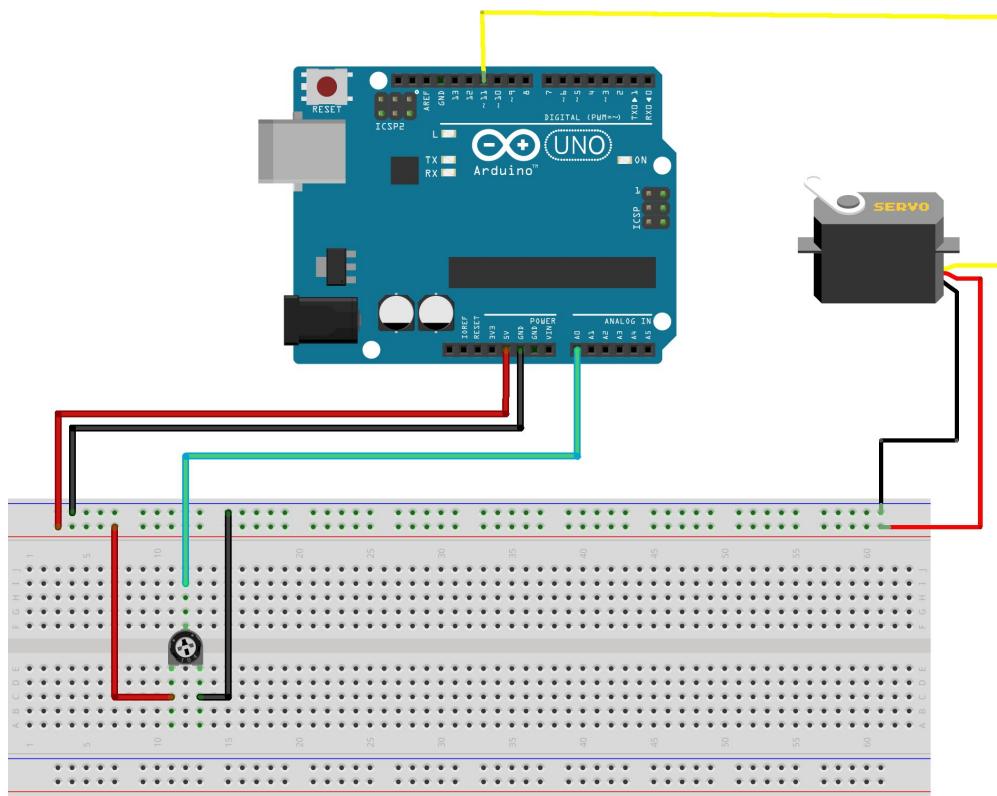
##### Necesitamos:

- 1 arduino
- 1 servomotor
- 1 potenciómetro de 10 KΩ

##### El circuito a realizar:

Vamos a conectar en la protoboard un potenciómetro. Conectamos una línea de 5 v. a la fase del potenciómetro (normalmente de color rojo) y la otra a GND (normalmente de color negro). A su vez, la tercera (es la de señal y puede ser de cualquier otro color), la llevamos a la entrada analógica A0 del arduino.

Luego conectamos la fase positiva del servo a 5 V. Y la negativa a GND. El cable de señal al pin 11 del Arduino. Es conveniente instalar un condensador en paralelo al motor dependiendo del tipo de servo que utilicemos.



Made with  Fritzing.org

**El código para poder controlar el servo es el siguiente:**

```
#include <Servo.h>

Servo myservo;          // Crea el objeto servo para controlarlo
int potpin=A0;           // usamos el pin analógico 0 para leer el potenciómetro
int val;                 // variable para leer los datos del pin A0

void setup()
{
    Serial.begin(9600);
    myservo.attach(11); // enlazamos el pin 11 al objeto servo
}

void loop()
{
    val= analogRead(potpin);      // lee el valor del potenciómetro (valor entre
                                // 0 y 1023)
    val = map(val,0,1023,0,179); // lo convierte a un valor entre 0 y 179

    Serial.print("Valor: ");
    Serial.println(val);         // imprime el dato en la ventana serial del
                                // ordenador
    myservo.write(val);         // envia al servo el nuevo valor
    delay(20);                // espera 15 milisegundos
}
```

Con **Serial.begin (9600)** abrimos el puerto serie a 9600 bits por segundo para que el arduino pueda comunicarse con el ordenador. Podemos abrir una ventana Serie y visualizar los datos que manda el arduino.

**int valor = analogRead(0);** Declaramos una variable llamada *valor* y almacenamos en ella la lectura que obtenemos del pin analógico 0

Con **Serial.print()** imprimimos los datos en la ventana serial del ordenador.

Con la función **map()** podemos convertir un valor incluido en un intervalo determinado a otro valor equivalente incluido en otro intervalo diferente. En este caso, la lectura del pin analógico da valores entre 0 y 1023. Sin embargo, para controlar un servo necesitamos enviarle valores entre 0 y 179 . Por eso, necesitamos convertir el valor recibido por el pin A0 a su valor correspondiente para el intervalo de 0 a 179. Este trabajo lo hace por nosotros la función **map()**

### Qué es un servo

Un **servomotor** (también llamado **servo**) es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.

Un servomotor es un motor eléctrico que puede ser controlado tanto en velocidad como en posición.

Es posible modificar un servomotor para obtener un motor de corriente continua que, si bien ya no tiene la capacidad de control del servo, conserva la fuerza, velocidad y baja inercia que caracteriza a estos dispositivos.

Está conformado por un motor, una caja reductora y un circuito de control. Un servomotor es un motor especial al que se ha añadido un sistema de control (tarjeta electrónica), un potenciómetro y un conjunto de engranajes. Con anterioridad los servomotores no permitían que el motor girara 360 grados, solo aproximadamente 180; sin embargo, hoy en día existen servomotores en los que puede ser controlada su posición y velocidad en los 360 grados. Los servomotores son comúnmente usados en modelismo como aviones, barcos, helicópteros y trenes para controlar de manera eficaz los sistemas motores y los de dirección.

Los servomotores hacen uso de la modulación por ancho de pulsos (PWM) para controlar la dirección o posición de los [motores de corriente continua](#). La mayoría trabaja en la frecuencia de los cincuenta [hercios](#), así las señales PWM tendrán un periodo de veinte [milisegundos](#).

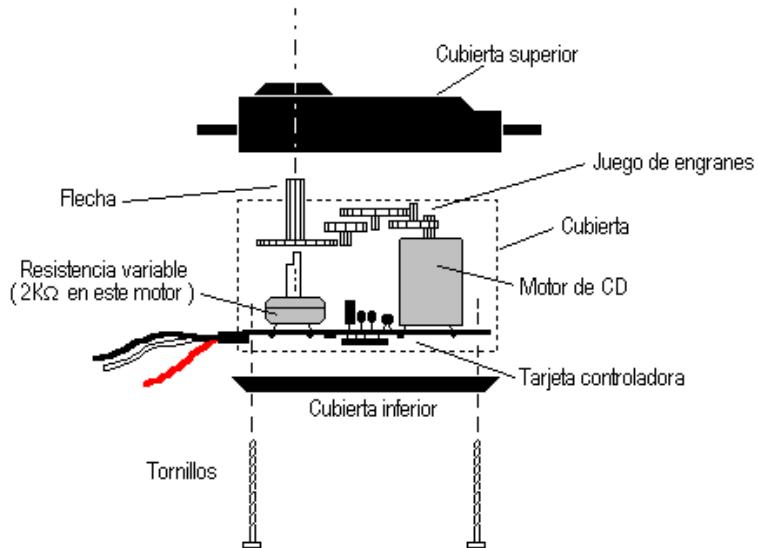
La electrónica dentro del servomotor responderá al ancho de la señal modulada. Si los circuitos dentro del servomotor reciben una señal de entre 0,5 a 1,4 milisegundos, éste se moverá en sentido horario; entre 1,6 a 2 milisegundos moverá el servomotor en sentido antihorario; 1,5 milisegundos representa un estado neutro para los servomotores estándares.

## Desarrollo de aplicaciones con Arduino

Las duraciones del pulso pueden cambiar en función del modelo del servomotor.

Duración del pulso	Posición del eje	Rango de duración del pulso de control: 900 µs – 2100 µs
900 µs	0°	
1200 µs	45°	
1500 µs	90°	
1800 µs	135°	
2100 µs	180°	

$$\text{Posición} = \frac{3 \cdot \text{duración} - 2700}{20}$$



### 7 – Creación de sonidos mediante la función tone()

Mientras que con PWM podemos variar el ancho del pulso eléctrico, pero no su frecuencia, con la función tone() lo que hacemos es variar la frecuencia. Esta función nos sirve para crear sonidos de diferentes frecuencias con el Arduino.

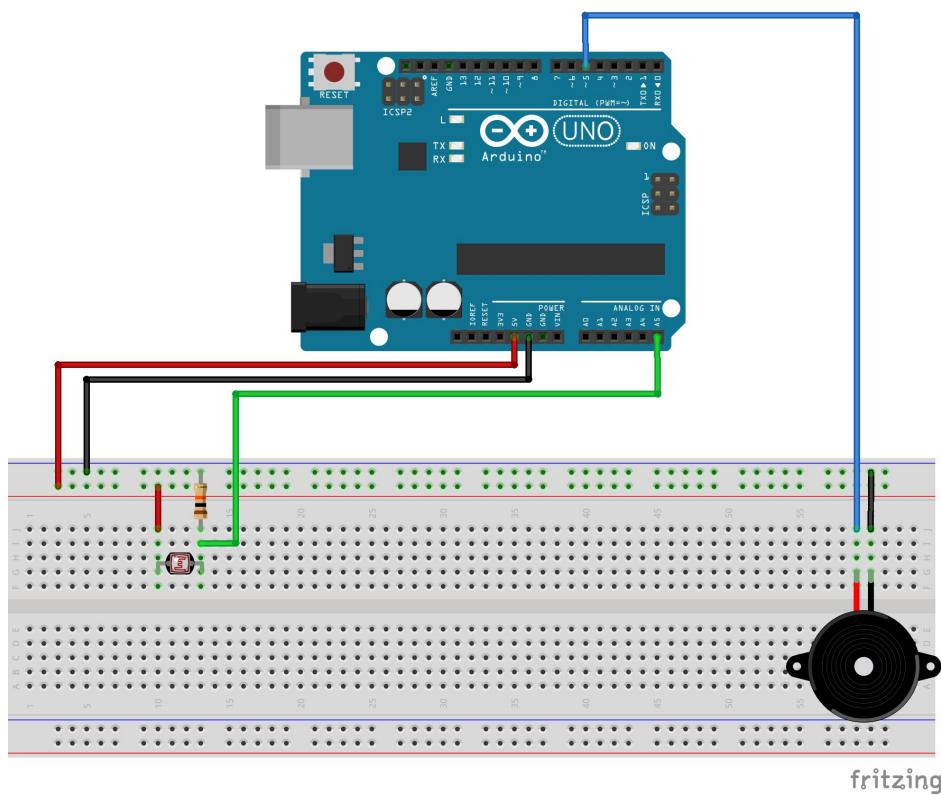
La función tone() admite tres parámetros: el pin a usar, la frecuencia y la duración. Este último es opcional.

#### Práctica nº 11

##### Necesitamos:

- 1 arduino
- 1 piezo eléctrico
- 1 resistencia LDR
- 1 resistencia de 10K

#### El circuito a realizar:



### El código

```
/*
 Creación de sonido mediante la función tone() y una LDR
 */

int luz;      //será el valor de entrada de la LDR
int sonido;   //será el valor de la frecuencia del altavoz
int ldr = A5; //conectaremos la LDR al pin A5
int altavoz = 5; //conectaremos el altavoz al pin 5

void setup()
{
    pinMode(altavoz,OUTPUT);
    Serial.begin(9600); //interesa ver los valores de la LDR
}

void loop()
{
    luz=analogRead(ldr);
    Serial.print("luz recibida: ");
    Serial.print(luz);
    Serial.print("\t");
    sonido=map(luz,200,450,3000,200);
    sonido=constrain(sonido,200,3000); //obligamos a que no salga de esos valores
    Serial.print("frecuencia sonido: ");
    Serial.println(sonido);
    tone(altavoz,sonido,200);
    delay(250); //damos tiempo a que suene la nota y a una pequeña pausa
}
```

### 8 – Controlar un motor DC mediante un transistor y Arduino

El Arduino, a través de sus pines digitales, puede suministrar hasta 5 v. con una intensidad de corriente de 40 mA. como máximo. Sin embargo, es muy habitual encontrarnos con dispositivos electrónicos que necesiten más tensión y/o intensidad de corriente (Amperios). En estos casos no podemos alimentar los dispositivos directamente desde el Arduino.

Tenemos que proporcionar una fuente de alimentación alternativa para estos componentes. Con el fin de poderlos controlar desde el Arduino necesitamos un tercer elemento que sí pueda recibir corriente eléctrica desde el Arduino y, a su vez, controlar el paso de corriente de mayor voltaje o intensidad.

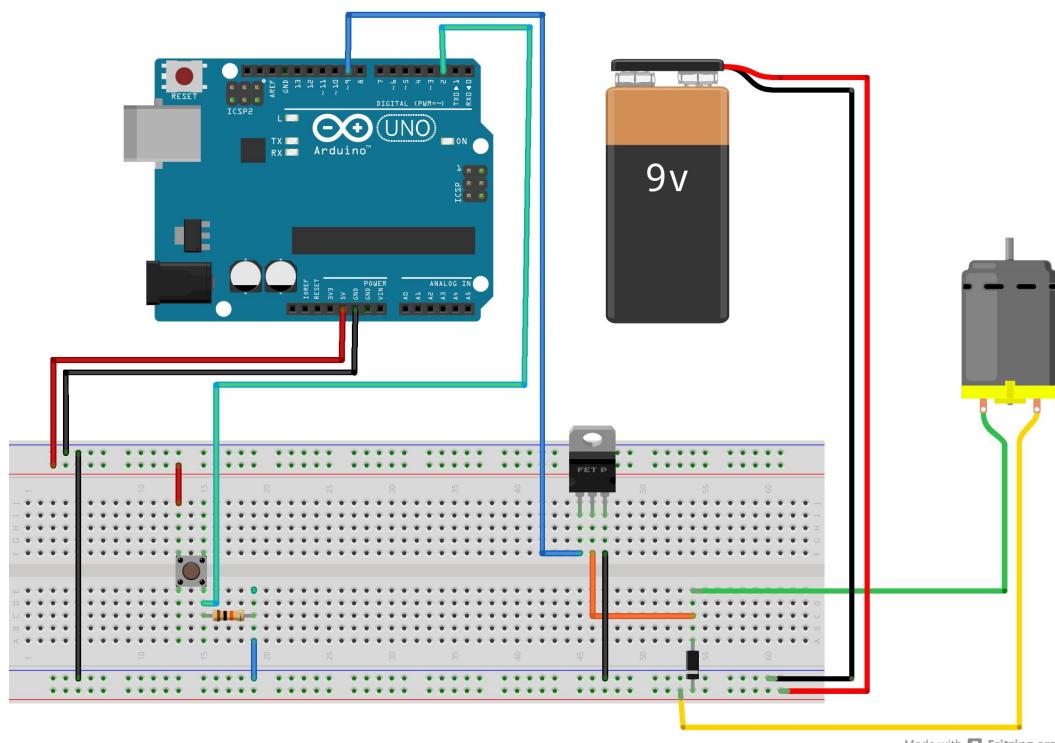
En nuestro ejercicio, este tercer elemento será un transistor que recibirá corriente eléctrica del Arduino y, en función de dicha corriente, podrá cerrar o abrir el otro circuito que alimentará al motor.

#### Práctica nº 12

##### Necesitamos:

- 1 arduino
- 1 motor de Corriente continua
- 1 transistor (MOSFET)
- 1 pulsador
- 1 resistencia de 10 KΩ
- 1 pila de 9V
- 1 diodo

##### El circuito a realizar:



Made with Fritzing.org

### El código para controlar el motor DC:

```
const int pulsadorPin = 2;           //declaramos una constante y le asignamos el pin 2
const int motorPin = 9;             //declaramos una constante y le asignamos el pin 9
int pulsadorEstado = 0;             // declaramos una variable para leer el estado del pulsador

void setup()
{
    pinMode(motorPin, OUTPUT);
    pinMode(pulsadorPin, INPUT);
}

void loop()
{
    pulsadorEstado = digitalRead(pulsadorPin);      // leemos el estado del pin 2
                                                       // y lo almacenamos en la variable
                                                       // pulsadorEstado

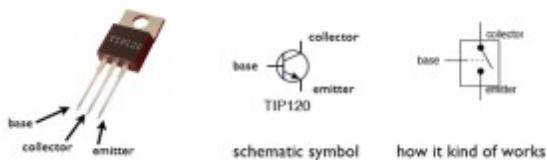
    if (pulsadorEstado == HIGH){ // si el pulsador está pulsado, activamos el pin del motor
        digitalWrite(motorPin, HIGH);
    } else {
        digitalWrite(motorPin, LOW);
    }
}
```

### Los transistores

Hasta ahora los dispositivos que hemos utilizado son de baja potencia. Arduino trabaja con un rango de 3 a 5 voltios y aproximadamente 20 mA de corriente, esto es suficiente para controlar dispositivos pequeños, pero cuando usamos algún actuador que utiliza mas de 20 mA de corriente o requiera una tensión mas alta no podemos hacerlo funcionar conectándolo directamente a los puertos E/S de Arduino, tenemos que implementar un circuito intermedio.

Aquí utilizamos un transistor TIP120. Un transistor se define como: un dispositivo electrónico semiconductor que cumple funciones de amplificador, oscilador, conmutador o rectificador. El término «transistor» es la contracción en inglés de transfer resistor («resistencia de transferencia»).

En este ejemplo el motor requiere mas de 20 mA para funcionar. Aquí implementamos como interfaz entre Arduino y el Motor CC un transistor npn TIP120. Este transistor esencialmente funciona como amplificador o como un interruptor electrónico (comutador o switch).



¿Cómo funciona esto?

Básicamente, tiene una entrada llamada el Colector, una salida llamada el Emisor, y un control denominado Base. Cuando se envía una señal de ALTO a la base (B, pin de control), el transistor cambia y permite que la corriente fluya desde el colector (C) para el emisor (E).

### 8 – Control completo de un motor DC mediante un “puente en H”

#### Práctica nº 13

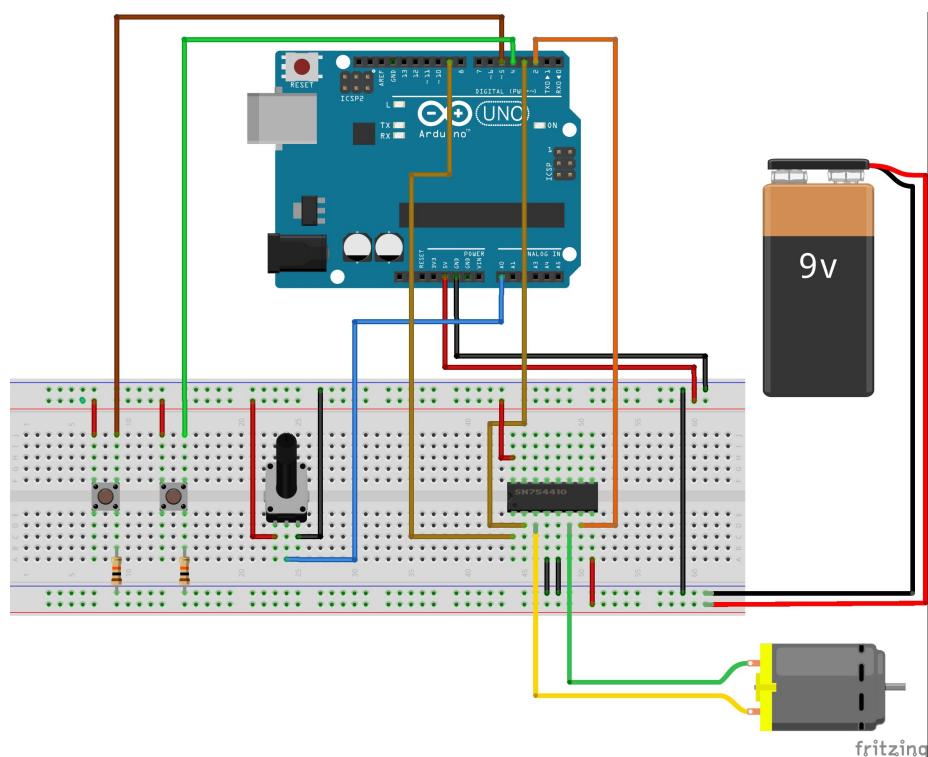
##### Necesitamos:

- 1 arduino
- 1 motor de Corriente continua
- 2 pulsadores
- 2 resistencias de 10 KΩ
- 1 potenciómetro de 10 KΩ
- 1 pila de 9V
- 1 integrado L293D

##### El circuito a realizar:

Vamos a realizar un circuito para poder tener un control total de un motor de corriente continua. Implementaremos un pulsador para poner en marcha el motor, otro pulsador para variar el sentido de rotación y un potenciómetro para regular la velocidad.

Para controlar de esta manera el motor usaremos el integrado **L293D**. Este circuito integrado nos dará la posibilidad de alimentar el motor con una fuente de alimentación externa al **Arduino** y, además, nos permitirá cambiar la polaridad del circuito para que cambie el sentido de rotación del motor.



### El código para controlar el motor DC:

```
const int controlPin1 = 2;
const int controlPin2 = 3;
const int enablePin = 9;
const int directionSwitchPin = 4;
const int onOffSwitchStateSwitchPin = 5;
const int potPin = A0;

int onOffSwitchState = 0;
int previousOnOffSwitchState = 0;
int directionSwitchState = 0;
int previousDirectionSwitchState = 0;
int motorEnabled = 0;
int motorSpeed = 0;
int motorDirection = 1;

void setup () {
    pinMode(directionSwitchPin, INPUT);
    pinMode(onOffSwitchStateSwitchPin, INPUT);
    pinMode(controlPin1, OUTPUT);
    pinMode(controlPin2, OUTPUT);
    pinMode(enablePin, OUTPUT);
    digitalWrite(enablePin, LOW);
}

void loop () {
    onOffSwitchState = digitalRead(onOffSwitchStateSwitchPin);
    delay(1);
    directionSwitchState = digitalRead(directionSwitchPin);
    motorSpeed = analogRead(potPin)/4;

    if (onOffSwitchState != previousOnOffSwitchState){
        if (onOffSwitchState == HIGH){
```

```
motorEnabled = !motorEnabled;
}

}

if (directionSwitchState != previousDirectionSwitchState){
    if (directionSwitchState == HIGH) {
        motorDirection = !motorDirection;
    }
}

if (motorDirection == 1){
    digitalWrite(controlPin1, HIGH);
    digitalWrite(controlPin2, LOW);
}else{
    digitalWrite(controlPin1, LOW);
    digitalWrite(controlPin2, HIGH);
}

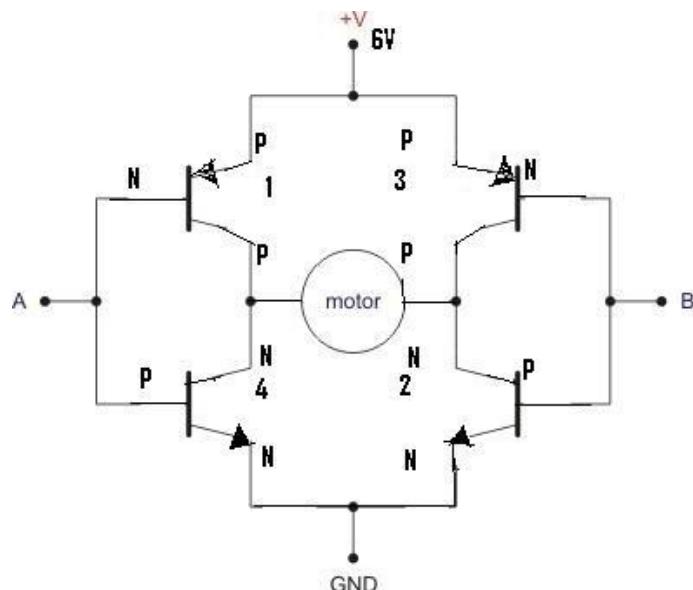
if (motorEnabled == 1) {
    analogWrite(enablePin, motorSpeed);
}else{
    analogWrite(enablePin, 0);
}

previousDirectionSwitchState = directionSwitchState;
previousOnOffSwitchState = onOffSwitchState;
}
```

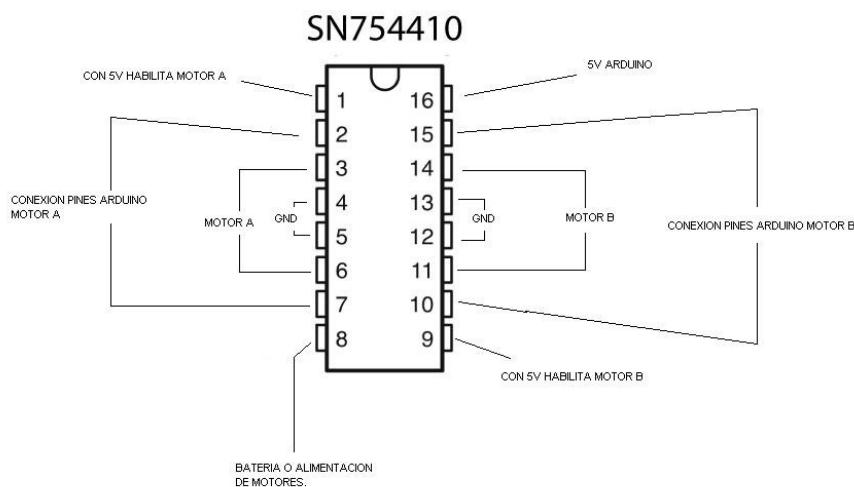
### Qué es un puente en H y para qué sirve

El puente en H es un circuito formado por cuatro transistores que, mediante la combinación entre ellos, permite invertir la polaridad de la corriente que pasa por un motor DC y con ello, cambiar el sentido de la rotación del motor.

El nombre de puente en H viene dado por la semejanza de su esquema con la letra H.



Existen circuitos integrados que llevan en su interior un puente en H, e incluso dos, para poder controlar a la vez dos motores. Los integrados dobles tienen 16 patillas mientras que los anteriores solo ocho. Estos son muy útiles en robótica porque es muy habitual que los sistemas de tracción necesiten de dos motores para poder girar.



### 9 Arduino + Processing

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.

En nuestro caso trataremos de realizar aplicaciones en las que el programa que controla y monitoriza los datos de Arduino este en el IDE Processing.

Existen dos métodos para controlar Arduino desde Processing:

1. Mediante la Librería Arduino para Processing
2. Mediante la lectura/escritura de datos a través del puerto serie.

#### Método 1: Librería Arduino para Processing

Antes de nada debemos realizar los siguientes pasos para acondicionar el entorno Processing:

##### A) CARGAR LA LIBRERIA ARDUINO EN EL IDE DE PROCESSING.

No debemos olvidarnos antes de nada de cargar el firmware correspondiente en Arduino. El fichero de la librería Arduino para Processing está en el archivo processing-arduino o arduino-processing-e231 que se encuentra en la página de Arduino. Dentro de ellos hay una carpeta que se llama Arduino y contiene la librería. \processing-arduino\arduino o \arduino-processing-e231\arduino.

<http://www.arduino.cc/playground/uploads/Interfacing/processing-arduino.zip>

Para que Processing pueda trabajar con la librería de Arduino debemos incluir la carpeta Arduino dentro de la carpeta librerías del IDE Processing:  
\processing-0138\libraries

##### B) CONFIGURAR ARDUINO PARA QUE PROCESSING PUEDA DIALOGAR CON ÉL.

Para cargar el firmware en Arduino nos vamos a la librería processign-arduino y en la carpeta \arduino\firmware\Standard\_Firmata se encuentra el fichero Standard\_Firmata.pde que debemos cargar con el IDE Arduino y lo debemos descargar sobre Arduino. En este fichero están las funciones que luego se utilizarán desde el IDE Processing para poder dialogar con Arduino.

#### Método 2: Mediante intercambio de datos a través del puerto serie

Se puede controlar Arduino desde Processing sin necesidad de incluir la librería Arduino en Processing, en este caso se trata de recoger datos del puerto que la tarjeta Arduino envía al puerto serie.

Procedimiento:

- 1.- Se carga en la tarjeta Arduino el programa que se encargue de escribir en el puerto el dato que después leerá Processing y lo incorporará en el programa que este ejecutando.
- 2.- Cargar y ejecutar el programa en el IDE Processing que recogerá los datos que Arduino le envía por el puerto serie.

Las siguientes funciones se encuentran en la Librería Arduino para Processing y comunican (a partir de Processing) con un Arduino, una vez que el Firmata se ha instalado en la tarjeta.

**Arduino.list()**: devuelve una lista con los dispositivos serie (puertos serie) disponibles. Si la tarjeta Arduino está conectada a la computadora cuando se llama a esta función, su dispositivo estará en la lista.

**Arduino(parent, name, rate)**: crea un “objeto” Arduino (objeto a nivel de elemento de programación). parent debe aparecer sin comillas; name es el nombre del dispositivo serie (es decir, uno de los nombres devueltos por Arduino.list()); rate es la velocidad de la conexión (57600 para la versión actual del de firmware).

**pinMode(pin, mode)**: pin configura un pin digital como entrada (input) o como salida (output) mode (Arduino.INPUT o Arduino.OUTPUT).

**digitalRead(pin)**: devuelve el valor leído de una de las entradas digitales, Arduino.LOW o bien Arduino.HIGH (el pin debe estar configurado como entrada).

**digitalWrite(pin, value)**: escribe Arduino.LOW o Arduino.HIGH en un pin digital.

**analogRead(pin)**: devuelve el valor de una entrada analógica leída (de 0 a 1023).

**analogWrite(pin, value)**: escribe un valor analógico (señal tipo PWM) en un pin digital que soporta salida analógica, los valores debes estar comprendidos entre 0 (equivalente a off) y 255 (equivalente a on).

### 9.1 Control de la intensidad de 2 leds mediante el movimiento del ratón

#### Práctica nº 14

Control de la intensidad de 2 leds mediante el movimiento del ratón

Para controlar la iluminación de los 2 leds crearemos un cuadrado con las siguientes dimensiones: 512 X 512 pxls.

Seleccionamos estas dimensiones para poder realizar una conversión más cómoda al trasladar las coordenadas del ratón a la escritura PWM del led. Si dividimos entre 2 las coordenadas del ratón, conoceremos el valor que hará encender ambos leds. Estos se iluminarán según el movimiento dentro del cuadrado, un led determinado por la posición vertical, mientras que el otro por la posición horizontal.

Si el puntero del ratón está en la posición X = 0 e Y = 512, entonces deberemos encender el LED1 a la máxima potencia:

### El código en el IDE de Processing

```
import processing.serial.*;
import cc.arduino.*;

int led1 = 9;
int led2 = 10;
Arduino arduino;

void setup()
{
    size(512, 512);
    arduino = new Arduino(this, Arduino.list()[0], 57600);
}

void draw()
{
    background(constrain(mouseX/2, 0, 255));
    arduino.analogWrite(led1, constrain(mouseX/2, 0, 255));
    arduino.analogWrite(led2, constrain(mouseX/2, 0, 255));
}
```

### 9.2 Control de 8 pines digitales de la placa Arduino

#### Práctica nº 15

En esta actividad vamos a representar y controlar 8 pines digitales de nuestra placa Arduino. Para ello crearemos una interfaz gráfica representando 8 pulsadores, cada uno de ellos va a controlar un pin de nuestra placa. De esta manera podremos activar y desactivar un pin al hacer click con el puntero del ratón.

En primer lugar vamos a establecer la interfaz gráfica que quedará como la figura siguiente:

Creamos un rectángulo de 290 x 100 pxls y 8 cuadrados de 20 x 20 pxls, cada uno de los cuales representa un pin digital (2 – 10). La separación entre cada cuadrado es de 10 pxls, y la

distancia del primero y último respecto el borde del rectángulo es de 30 pxls.

A partir de esta información, podemos decirle al puntero del ratón en qué posición debe

encender o apagar cada uno de los pines:

8 pines \* 20 pxls de anchura = 160 pxls

10 pxls \* 7 espacios de separación entre cuadrados = 70 pxls 290 (ancho rectángulo) – 70 – 160 = 60 pxls (extremos) Quedando la distancia entre cuadrados de la siguiente forma:

Utilizaremos la función mousePressed(), la cual contiene el evento que informará si se ha pulsado el botón del ratón, e implementaremos en su cuerpo la siguiente ecuación para saber si estamos dentro de un cuadrado y en cuál de ellos:

```
int pin = ((mouseX-270) / 30)+9;
```

En la variable pin almacenaremos un número entero positivo entre 2 y 8 (los 8 pines digitales).

### El código para el IDE de Processing

```
import processing.serial.*;
import cc.arduino.*;

color off = color(0, 0, 0);
color on = color(255, 0, 0);

int[] valores = {Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW,
Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW}; //vector con 10 posiciones

Arduino arduino;

void setup()
{
    size(290, 100);
    arduino = new Arduino(this, Arduino.list()[0], 57600);

    //pines del 2 al 9
    for(int i=2; i<10; i++)
    {
        arduino.pinMode(valores[i], Arduino.OUTPUT);
    }
}
```

## Desarrollo de aplicaciones con Arduino

```
void draw()
{
    background(off);
    stroke(on);
    int valor = 0;
    for(int i = 2; i< 10; i++)
    {
        if(valores[i] == Arduino.LOW)
        {
            fill(off);
            rect(valor * 30 + 30, 40, 20, 20); // (x, y, ancho, alto)
        }else
        {
            fill(on);
            rect(valor * 30 + 30, 40, 20, 20);
        }
        valor++;
    }
}

void mousePressed()
{
    //Determinamos la posición donde activaremos las salidas
    int pin = ((mouseX - 270)/30) + 9;
    if(mouseY < 40 || mouseY > 60 || mouseX > 260)
    {
        textSize(50);
        fill(255);
        text("ERROR", 60, 70);
    }else
    {
        if(valores[pin] == Arduino.LOW)
        {
            valores[pin] = Arduino.HIGH;
        }
    }
}
```

```
    arduino.digitalWrite(pin, Arduino.HIGH);  
}  
else  
{  
    valores[pin] = Arduino.LOW;  
    arduino.digitalWrite(pin, Arduino.LOW);  
}  
}  
}
```

### 9.3 Representación gráfica del valor procedente de un sensor

#### Práctica nº 16

En este ejercicio vamos a leer un valor de una señal analógica presente en la entrada analógica 0 y se va mostrar en la ventana de Processing su valor convertido (0-1024), su valor real en voltios (0-5) y su representación gráfica.

#### El código para el IDE de Processing

```
import processing.serial.*;  
import cc.arduino.*;  
int sensor = 0;  
int valor;  
  
int[] valores;  
  
Arduino arduino;  
  
void setup()  
{  
    size(256, 256);  
    arduino = new Arduino(this, Arduino.list()[0], 57600);  
    valores = new int[width];  
}
```

## Desarrollo de aplicaciones con Arduino

```
/* Calculamos el valor recibido en voltios*/
float calcularVoltaje(int bits)
{
    float v = 5.0 * (bits/255.0);
    return v/4;
}

/*Desplazamos los valores del array*/
void desplazarArray()
{
    for(int i = 1; i < width; i++)
    {
        valores[i-1] = valores[i];
    }
}

/*Representar valores*/
void representarValores(int v)
{
    textAlign(RIGHT);
    text(v/4, 200, 30); //bits
    text(calcularVoltaje(v) + "V", 200, 60);
}

/*Almacenamos el valor leido en la última posición del array*/
void almacenarValor(int val)
{
    val = arduino.analogRead(sensor);
    valores[width-1] = val/4;
    representarValores(val);
}
```

```
/*Dibujar array*/
void dibujarGrafico()
{
    stroke(255);
    for(int i=1; i<width; i++)
    {
        point(i, 255 - valores[i]);
    }
}

void draw()
{
    background(0);
    desplazarArray();
    almacenarValor(valor);
    dibujarGrafico();
}
```

### 9.4. Control de las salidas digitales mediante el teclado

#### Práctica nº 17

Vamos a controlar el encendido y apagado de tres LEDs conectados a las salidas digitales 12, 11 y 10 mediante las teclas “1”, “2” y “3” respectivamente actuando estas en modo biestable (una pulsación enciende la siguiente pulsación apaga).

En esta actividad vamos a realizar el intercambio de datos mediante el puerto serie

#### 9.4.1. Programa realizado en el IDE Arduino

```
int ledPin1 = 12;
int ledPin2 = 11;
int ledPin3 = 10;
int estado1 = HIGH;
int estado2 = HIGH;
int estado3 = HIGH;
int valor = 0;
```

## Desarrollo de aplicaciones con Arduino

```
void setup()
{
    Serial.begin(9600);
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
}

void loop()
{
    valor = Serial.read(); //lee el valor del puerto
    if(valor != -1)
    {
        switch(valor)
        {
            case'1':
                estado1 = !estado1;
                break;
            case'2':
                estado2 = !estado2;
                break;
            case'3':
                estado3 = !estado3;
                break;
        }
    }
    digitalWrite(ledPin1, estado1);
    digitalWrite(ledPin2, estado2);
    digitalWrite(ledPin3, estado3);
}
```

### 9.4.2. Programa realizado en el IDE Processing

```
import processing.serial.*;  
  
Serial port;  
  
void setup()  
{  
    size(255, 255);  
    port = new Serial(this, Serial.list()[0], 9600); //establecemos la misma velocidad en el Arduino  
}  
  
void draw()  
{  
    background(0);  
}  
  
void keyReleased()  
{  
    port.write(key);  
}
```