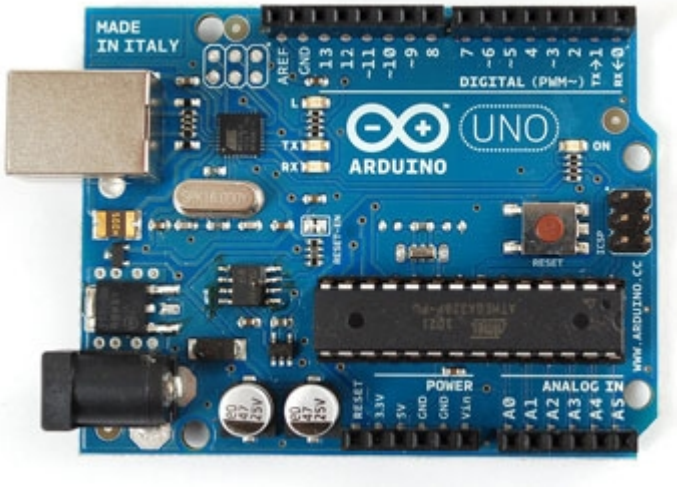
ARDUINO

Descubre el mundo de la electrónica de forma fácil

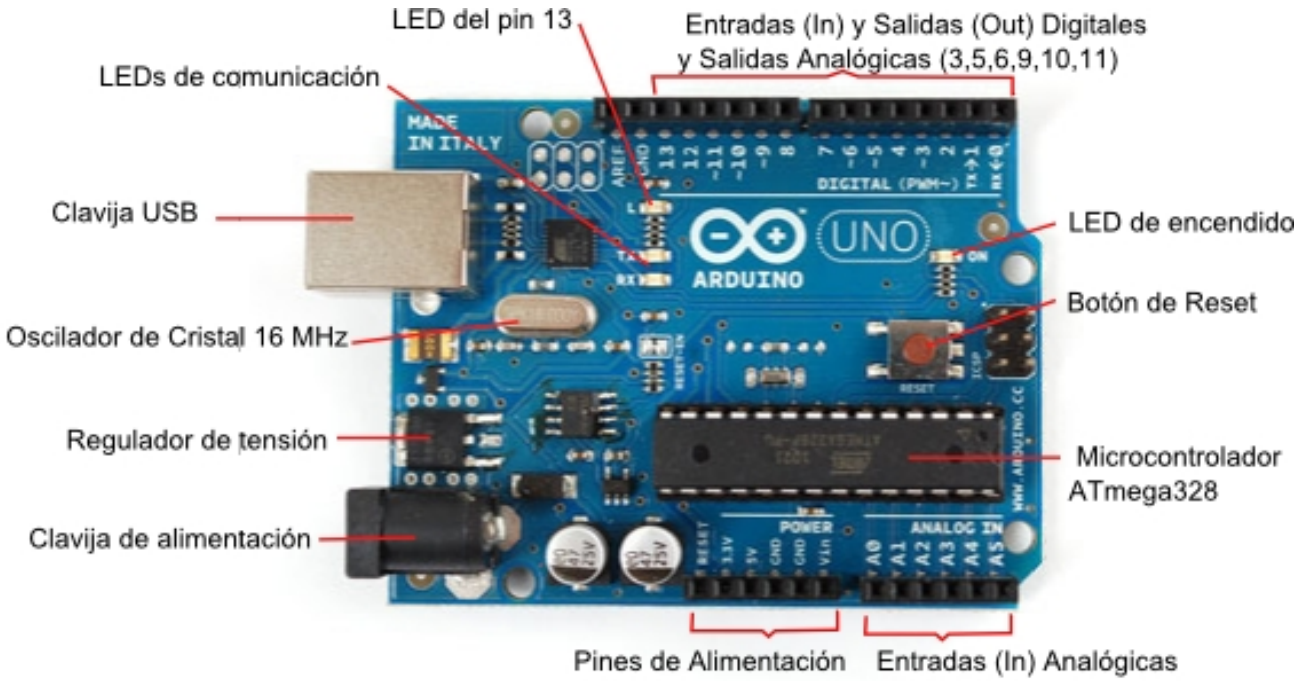


Apuntes de ARDUINO Nivel Pardillo

Daniel Gallardo García
Profesor de Tecnología del IES Laguna de Tollón

1. ¿QUÉ ES ARDUINO?

Arduino es una placa o tarjeta controladora, con una serie de entradas y salidas, y que se programa a través del ordenador mediante un lenguaje de programación. Veamos qué elementos componen una ***Arduino UNO***:



Alimentación: Arduino puede estar alimentado por dos vías:

- conexión **USB** (que proporciona 5 V).
- **jack** de alimentación (que normalmente será una pila de 9 V o fuente de alimentación, que se recomienda que esté entre 7 – 12 V).

Los **pines de alimentación** son para alimentar los circuitos la placa de prototipos o *breadboard* o *protoboard*:

- **3.3 V** proporciona una tensión de 3,3 V, y una intensidad máxima de 50 mA.
- **5 V** proporciona una tensión de 5 V, y una intensidad máxima de 300 mA.
- **GND** es la toma de tierra, o nivel 0 V de referencia.
- **Vin** proporciona la tensión máxima con la que está alimentado Arduino.

Valores de entrada y de salida: en función de cómo esté siendo utilizado en pin, tendremos:

- **Salida y entrada digital:** los valores de salida pueden ser o 0 V (`LOW`) o 5 V (`HIGH`), y se interpretará una entrada de entre 0 y 2 V como `LOW` y de entre 3 y 5 V como `HIGH`.
- **Salida analógica:** los valores de salida van desde 0 V a 5 V en un rango de **0** a **255** (precisión de 8 bits) valores intermedios.
- **Entrada analógica:** los valores de entrada van desde 0 V a 5 V en un rango de **0** a **1023** (precisión de 10 bits) valores intermedios.

La intensidad máxima de todos estos pines es de 40 mA.

Normalmente, todo el circuito electrónico que Arduino controlará se monta sobre una placa de prototipos o *breadboard*, y el conexionado se realiza con cables tipo *jumper* (es importante utilizar este tipo de cables porque no suelen romperse en los zócalos):

2. PROGRAMANDO ARDUINO

Todo programa para Arduino presenta una **estructura básica**:

1ª parte	`int x=0;`	**Declarar** las **variables**.
2ª parte	`void setup() {…}`	**Configuración** de Arduino.
3ª parte	`void loop() {…}`	**Comandos** que regirán el comportamiento de Arduino.

1ª parte**: Declarar las variables** `int x=0;`

Una variable es un valor que Arduino puede almacenar en su memoria, y que posteriormente podrá ser utilizado o modificado.

Los **tipos de variables** más utilizados son:

- `int`: almacena un **número entero** entre -32769 y 32767 (2 bytes).
- `long`: almacena un **número entero** muy largo, entre -2147483648 y 2147483647 (4 bytes).
- `float`: almacena un **número decimal** con un rango entre $-3.4028235 \cdot 10^{38}$ y $3.4028235 \cdot 10^{38}$ (4 bytes).
- `const`: especifica que la variable definida no podrá ser cambiada durante el programa, siendo un siempre un **valor constante**:

```
const float pi=3.1415;
```

Es importante saber que es posible declarar una variable sin asignarle un valor inicial, y hacerlo posteriormente durante el transcurso del programa:

```
int x;
…
x=4;
```

Dominio de una variable: si declaro una variable al comienzo del programa, podré emplear dicha variable en cualquier momento (dentro de cualquier función o bloque de programa), pero si declaro una variable dentro de una función, sólo se podrá utilizar en dicha función.

Poner nombre a las variables: Por último, una última consideración: a la hora de poner un nombre a una variable es recomendable utilizar alguna palabra que nos ayude a reconocer qué se está almacenando en ella, y en caso de utilizar dos o más palabras se suele emplear la notación de **joroba de camello** (poner en mayúscula la primera letra de las siguientes palabras). Ejemplos son:

```
ledPin              estadoAnterior          cuentaPulsaciones
miVariable          lecturaSensor           ledPinAzul
```

2ª parte: Configuración de Arduino `void setup() {…}`

En este bloque habrá que especificar:

- Qué **pines** van a ser empleados como **entrada** y cuáles como **salida**.

```
pinMode(2,OUTPUT);   //utilizaré el pin 2 como salida Digital.
pinMode(3,OUTPUT);   //utilizaré el pin 3 como salida Digital o Analógica.
pinMode(8,INPUT);    //utilizaré el pin 10 como entrada Digital.
```

Las entradas analógicas no hacen falta incluirlas en el setup, puesto que esos pines (A0, A1, A2, A3, A4, A5) solo pueden ser entradas analógicas.

- Si vamos a querer establecer una **conexión** con el **ordenador**.

```
Serial.begin(9600);
```
/*hay que especificar los baudios (bits por segundo) a la que va a realizarse dicha comunicación Arduino-PC */

- Si vamos a querer utilizar **número aleatorios**.

```
randomSeed(0);
```
//se inicia la generación de número aleatorios.

3ª parte: Comandos que regirán el comportamiento de Arduino `void loop () {…}`

En este bloque se deberá escribir todas aquellas instrucciones, órdenes, primitivas, comandos o funciones necesarias para que Arduino funcione según nuestro deseo. Realmente, este bloque constituye un bucle infinito, ya que Arduino, mientras esté alimentada con energía, funcionará haciendo el programa **loop** una y otra vez.

Iremos viendo cuáles son estas funciones durante el desarrollo de estos apuntes.

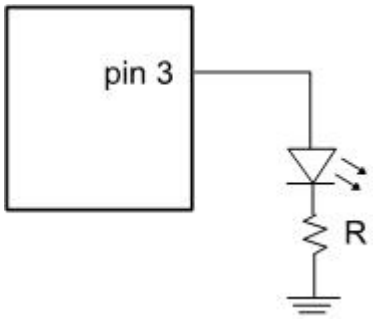
3. SALIDAS DIGITALES `digitalWrite(4,HIGH);`

Podemos indicar a Arduino que en un pin determinado coloque un "**0**" o un "**1**" lógico (que serán 0 V o 5 V) mediante los siguientes comandos, respectivamente:

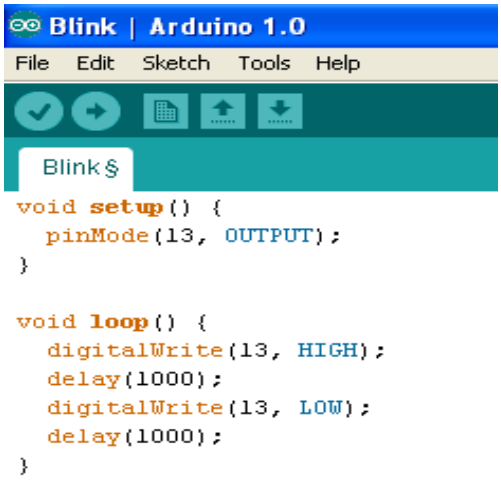
```
digitalWrite(12,LOW);          digitalWrite(12,HIGH);
```

Ejemplo 1. *Parpadeo de un LED*

Conectaremos un diodo LED en el pin 3. Siempre que conectemos un LED a una salida de Arduino debemos hacerlo en serie con una resistencia de valor comprendido entre 100 y 1K para evitar que una intensidad demasiado elevada destruya dicho LED:



El programa que utilizaremos será el siguiente:



Vemos que ha aparecido una nueva instrucción: **delay (**milisegundos**);**

```
delay(1000);          //realiza una pausa en el programa de 1000 ms
```

Es muy cómodo utilizar **variables** para luego modificar rápidamente los parámetros del programa, como por ejemplo el tiempo de espera o el pin donde conecto el LED:

```
int pinLed=3;                 //conectaré el LED en el pin 3
int t=1000;                   //t será el tiempo de espera

void setup() {
  pinMode(pinLed,OUTPUT);     /*configuro el pin 3
                              como pin de salida */
}

void loop() {
  digitalWrite(pinLed,HIGH);  //enciendo el LED
  delay(t);                   //espero un segundo
  digitalWrite(pinLed,LOW);   //apago el LED
  delay(t);                   //espero otro segundo
}
```

Asimismo, es muy útil emplear **comentarios** dentro de un programa, que facilitan mucho su lectura y comprensión. A la hora de escribir comentarios podemos emplear dos opciones:

`// …` permite escribir un comentario en una línea de código.

`/* … */` permite escribir un comentario en varias líneas de código.

Ejemplo 2. Luz que avanza a través de 5 LEDs

Para este montaje utilizaremos 5 LEDs, conectados en pines consecutivos (del 7 al 11, por ejemplo). Los LEDs se irán encendiendo consecutivamente, empezando por el 7 y terminado en el 11.

```
int i=7;

void setup() {
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
}

void loop() {
  for(i=7; i<=11; i++){
    digitalWrite(i,HIGH);     //enciendo el LED i
    digitalWrite(i-1,LOW);    //apago el LED anterior
    delay(1000);
  }
  digitalWrite(11,LOW);       /*apago el último LED puesto que
                              nunca se llega a i-1=11 */
}
```

Aquí ha aparecido una nueva estructura: **for (**incio**;** condición**;** incremento**) { …** **}**

La estructura `for` repite un número de veces las instrucciones que estén contenidas entre llaves, y la lógica que sigue es la siguiente:

- Emplea una variable (en este caso `i`) que se inicia (asignándole un valor inicial, en este ejemplo `i=7`).
- Dicha variable va incrementándose cada vez que se repite el `for`. El incremento puede expresarse así:

```
i=i+5     //el valor de i se incrementa en 5
i+=5      //el valor de i se incrementa en 5 (es otra forma)
i=i+1     //el valor de i se incrementa en 1
i+=1      //el valor de i se incrementa en 1 (es otra forma)
i++       //el valor de i se incrementa en 1 (sólo para incremento +1)
i=i-1     //el valor de i disminuye en 1
i-=1      //el valor de i disminuye en 1 (es otra forma)
i--       //el valor de i disminuye en 1 (sólo para incremento -1)
i=i*3     //el valor de i se multiplica por 3
i*=3      //el valor de i se multiplica por 3 (es otra forma)
i=i/2     //el valor de i se divide entre 2
i/=2      //el valor de i se divide entre 2 (es otra forma)
```

- El bucle `for` se repetirá siempre y cuando se siga cumpliendo la condición.

`for(i=7; i<=11; i ++) { … }` hará un primer ciclo con el valor `i=7`. Después de hacer todo lo que está entre las llaves, incrementará en 1 el valor de `i`. ¿Es `i` (ahora 8) menor o igual que 11? Como la respuesta es sí, volverá a hacer el bloque. Cuando termine el ciclo con valor `i=11`, la `i` se incrementará en 1 (valdrá 12). En ese momento ya no cumplirá la condición y el programa se saldrá del bucle `for`.

Otra observación es que, como la variable `i` solo se va a utilizar dentro del for (y en ninguna otra parte del programa) puedo declararla en ese mismo momento:

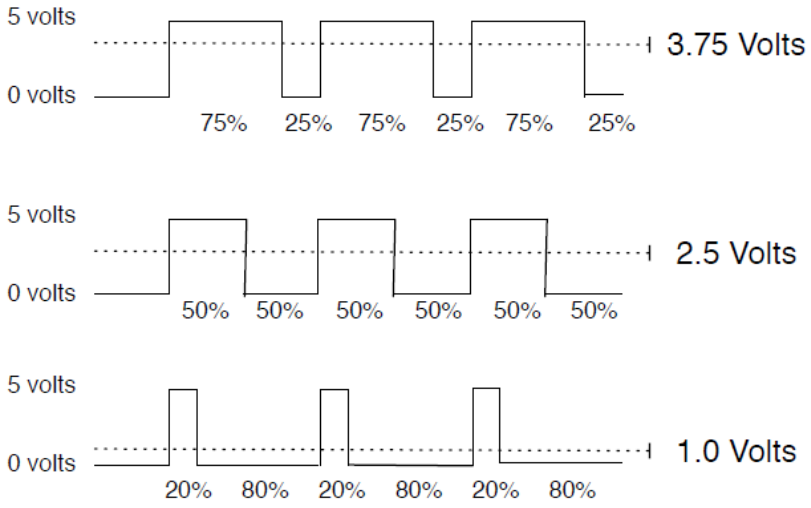
```
for(int i=7; i<=11; i++) { … }
```

4. SALIDAS ANALÓGICAS `analogWrite(5,128);`

Podemos indicar a Arduino que en un pin determinado coloque un valor de tensión comprendido entre 0 V y 5 V, pudiendo seleccionar entre 256 valores intermedios posibles (de **0** a **255**), empleando la siguiente orden:

```
analogWrite(11,214);      /* coloca en el pin 11 un valor de salida de
                             214, que equivale a unos 4,2 V */
```

Realmente, la señal analógica de salida no es analógica en sí, sino un PWM. Esto significa que la salida es una señal modulada por pulsos, formada por una serie de pulsos (de valor 5 V) repartidos durante un tiempo determinado de tal forma que el valor promedio de la señal de salida se hace coincidir con el de la señal analógica que se persigue imitar. Es importante tener esto presente para comprender el funcionamiento de Arduino y, por ejemplo, de los transistores conectados a ella.



Ejemplo 3. Parpadeo de un LED cuya intensidad será gradual

Se trata ahora de que el LED brille de forma gradual: de menos intensidad a más, y cuando alcance su máximo brillo comience a disminuir, y así sucesivamente.

El programa será el siguiente:

```
int brillo=0;       //declaro la variable brillo, con valor inicial=0
int incremento=5;   //declaro la variable incremento, con valor=5
int ledPin=9;

void setup() {
  pinMode(ledPin,OUTPUT);      //el pin 9 será la salida
  }

void loop() {
  analogWrite(ledPin, brillo);       //coloca el valor brillo en el pin 9
  brillo = brillo + incremento;      //brillo aumenta su valor en 5
  if(brillo==0 || brillo==255) {     //si brillo llega a sus límites...
    incremento = -incremento;        //...pasamos de ir creciendo a pasar...
                                     //...a ir decreciendo y al revés
  }
  delay(30);       /*hay que dar un pequeño tiempo entre valor y valor
                   de brillo para que la variación no sea instantánea */
}
```

Vemos que ha aparecido una nueva **estructura**, la **condicional**: **if (**condición**) {**...**}**

```
if(x>=5)  {…}
```

Entre los paréntesis pondremos una condición, y en caso de cumplirse, se ejecutarán los comandos que estén dentro de las llaves.

Si solamente se va a escribir un único comando, pueden omitirse las llaves:

```
if(x!=4) digitalWrite(pinLed,HIGH);
```

A la hora de comprobar si una condición se cumple o no, podemos utilizar los siguientes **operadores de comparación** dentro del paréntesis:

- `x == 3` si `x` es igual a 3
- `x != 3` si `x` es distinto de 3
- `x < 3` si `x` es menor que 3
- `x > 3` si `x` es mayor que 3
- `x <=3` si `x` es menor o igual que 3
- `x >= 3` si `x` es mayor o igual que 3
- `x < 2 || x > 5` si `x` es menor que 2 **o** mayor que 5
- `x == 4 && z <= 8` si `x` es igual a 4 **y** `z` es menor o igual que 8
- `! x > 0` si `x` **no** es mayor que 0 (si es menor o igual que 0)

¡Importante! No confundir **=** (que es para asignar) con **==** (que es para comparar).

Una variante a esta estructura es la formada por: **if (**condición**) {**…**} else {**…**}**

```
if (x>=5) {…}
else {…}          //correspondería al caso para x<5
```

Permite que el programa coja uno de los dos caminos: si se cumple la condición (que será lo que acompañe al `if`), o si no lo cumple (que será lo que acompañe a `else`).

Otra variante es la siguiente: **if (**cond1**) {**…**} else if (**cond2**) {**…**}** … **else {**…**}**

```
if (condición1) {…}
else if (condición2) {…}
else if (condición3) {…}
…
else if (condiciónN) {…}
else {…}
```

La única precaución a tomar en esta estructura es que no haya dos condiciones que sean compatibles, es decir: todas las condiciones deben ser excluyentes (en caso de haber más de una condición verdadera, Arduino ejecutará la primera que encuentre).

A la hora de escribir la condición dentro de los paréntesis de un `if` cabe una posibilidad de no utilizar operadores comparadores, y escribir únicamente una orden de lectura o una variable. En estos casos, la estructura `if` solo interpretará como condición de **no cumplimiento** un valor de return de dicho valor de `0`, **`LOW`** o **`FALSE`**:

`if(digitalRead(4)){…}` es lo mismo que `if(digitalRead(4)!=LOW)`

`if(!valor){…}` solo se ejecutará si la variable `valor` es `0`, `LOW` o `false` (porque así `!valor` dará un valor de `true`)

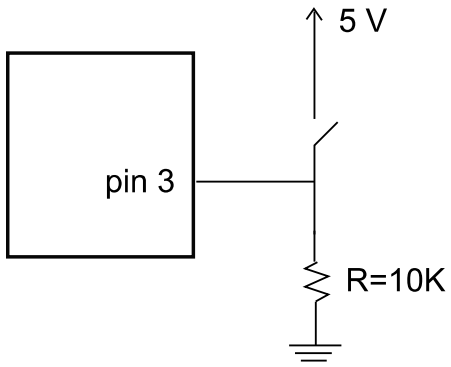
5. ENTRADAS DIGITALES `digitalRead(5);`

Debemos indicar a Arduino qué pin vamos a emplear como entrada digital. Dicha señal de entrada podrá tener dos valores: `LOW` o `HIGH`.

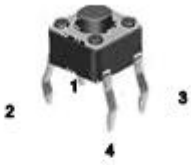
```
x = digitalRead(3);      //asigna a x el valor lógico que Arduino lee en el pin 3
```

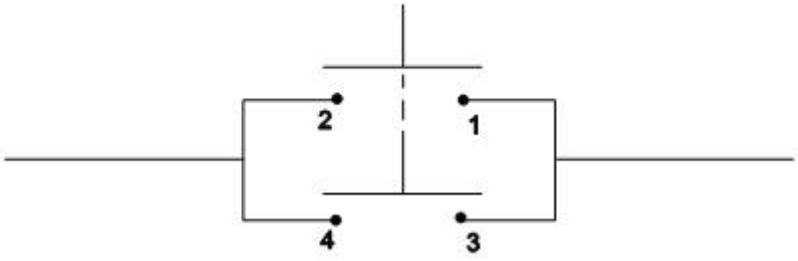
Cualquier sensor que tenga dos posibles valores distintos puede emplearse como entrada digital.

Podemos hacer una entrada digital con un **pulsador** o interruptor a través de una resistencia de drenaje (pull down), normalmente de 10K. Debemos tener en cuenta que Arduino va a percibir todo el transitorio (sobreoscilación o bouncing) cada vez que el pulsador sea accionado, en el cambio de 0 a 5 V. Eso significa que durante un muy breve periodo de tiempo, el valor en la entrada puede oscilar.



El esquema de un pulsador doble es el siguiente:





Ejemplo 4. ***Contador de pulsaciones con codificación binaria en 3 LEDs***

Este montaje consta de 3 LEDs, que reflejarán el número de veces que ha sido accionado un pulsador, codificando dicho número en binario, es decir, contará desde 0 a 7. El programa que controla este proyecto es el siguiente:

```
int val;       //no hace falta asignarle un valor inicial
int x=0;       //será el número de pulsaciones

void setup() {
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(13,INPUT);    //configuro el pin 13 como entrada digital
  }

void loop() {
  val=digitalRead(13);  //asigno a val el valor de lectura del pin 13
  if(val==HIGH) x=x+1;  //detecta una pulsación y la suma a la cuenta
  if(x==8) x=0;         //si llegamos a 8, hacemos que inicie la cuenta
  if(x==1 || x==3 || x==5 || x==7) digitalWrite(6,HIGH);
                        //condición para encender el tercer bit
  else {digitalWrite(6,LOW);}
  if(x==2 || x==3 || x==6 || x==7) digitalWrite(7,HIGH);
                        //condición para encender el segundo bit
  else {digitalWrite(7,LOW);}
  if(x==4 || x==5 || x==6 || x==7) digitalWrite(8,HIGH);
                        //condición para encender el primer bit
  else {digitalWrite(8,LOW);}
  delay(200);           /*espero 200ms para que no cuente más de una vez la
                        misma pulsación */
}
```

Como vemos, se suele utilizar estas entradas con la función `if` (condicionante). Recordemos que la lectura digital de una entrada sólo puede tomar dos valores: `LOW` y `HIGH`.

Ejemplo 5. Cambiar el estado de un LED (ON/OFF) con un pulsador

Un montaje muy sencillo sería controlar el encendido de LED con un pulsador si el funcionamiento fuera que mientras esté pulsado el botón, esté encendido el LED, y cuando se deje de accionar el LED se apagara. Pero este es otro ejercicio más complicado: sería utilizar el pulsador a modo de interruptor.

En este caso el LED pasará a estar encendido o a estar apagado cada vez que accionemos el pulsador. Para determinar esto, crearemos una variable (estado) que deberá cambiar cada vez que apretemos el pulsador, pero… ¿qué ocurre si estoy apretando el botón bastante tiempo? Hasta que no vuelva a apretar no debería cambiar el estado, y la solución no puede ser introducir un `delay(1000)` porque puede darse el caso de que se apriete dos veces el botón en un intervalo de tiempo muy pequeño y el programa no reconociera las dos pulsaciones.
La solución a esto es crear dos variables: `val` y `valAnterior`, que permitirán determinar si ha habido o no una acción de cambiar el estado del pulsador o no. El programa puede ser el siguiente:

```
int val=0;             //almacenará la lectura del pulsador: ON u OFF
int valAnterior=0;     //almacenará una lectura justamente anterior
int estado=0;          //¿qué toca ahora, encender o apagar el LED?
```

```
void setup(){
  pinMode(13,OUTPUT);
  pinMode(7,INPUT);
}

void loop(){
  val=digitalRead(7);
  if(val==HIGH && valAnterior==LOW) {     /*detecta el cambio del estado
                                          del pulsador de no estar pulsando
                                          a estar pulsando */
    estado = 1 - estado;      //si he pulsado, que cambie el estado del LED
    delay(20);                //elimino los efectos de bouncing
  }
  valAnterior=val;      //el valor del pulsador pasa a ser valor pasado
  if(estado==1) digitalWrite(13,HIGH);
                        //el LED encenderá cuando el estado sea 1
  else {
    digitalWrite(13,LOW);
  }
}
```

6. ENTRADAS ANALÓGICAS `analogRead(A0);`

Recordemos que las entradas analógicas en Arduino no hay que configurarlas en el setup. Dicha señal analógica de entrada podrá tener valores comprendidos entre **0** y **1023**, correspondientes a los valores intermedios de un rango de 0 a 5 V.

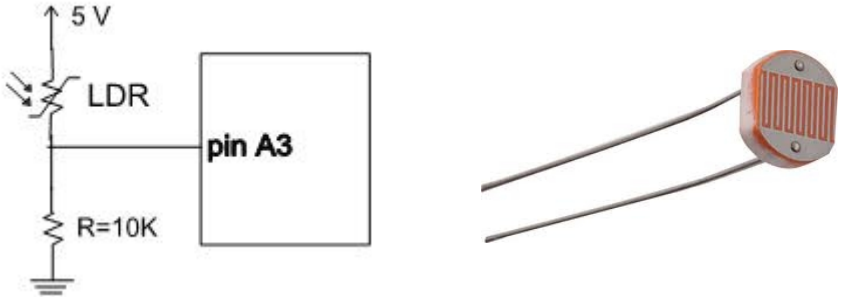
`x = analogRead(A3);` //asigna a `x` el valor analógico que Arduino lee en el pin `A3`

Como los únicos pines válidos para una entrada analógica son los A0...A5, no hay posibilidad de confusión, y puede omitirse la `A`: `x = analogRead(3);`

Evidentemente, para emplear una entrada analógica necesitamos un sensor analógico, es decir, que sus valores eléctricos varíen en un rango significativo, no limitándose a dos posibles valores. Sensores analógicos pueden ser: LDR (fotorresistencia), NTC (termoresistencia), potenciómetro, sensor de sonido (piezoresistencia), sensor de ultrasonido, etc…

Ejemplo 6. Cambiar el brillo de un LED en función de la luz ambiente recibida

Para este montaje necesitaremos un LED conectado a un pin de salida analógica (pues queremos controlar la intensidad de luz, y una **LDR** conectada a una entrada analógica. La forma de conectar una resistencia variable (como la LDR) a una entrada analógica de Arduino es a través de un divisor de tensión, obteniéndose el mayor rango de lectura si utilizamos una resistencia pull down con un valor semejante al valor máximo que puede adquirir la resistencia variable (para este dato puede ser muy útil un polímetro):



El programa para este circuito puede ser el siguiente:

```
int val;
int ledPin=9;

void setup(){
  pinMode(ledPin,OUTPUT);
}

void loop(){
  val=analogRead(A0);          //hago la lectura analógica en el pin A0
  val=val/4;                   //divido entre 4 dicho valor
  analogWrite(ledPin,val);     //pongo en la salida este último valor
  delay(100);
}
```

Es importante conocer el motivo de por qué se divide entre 4 el valor de la lectura antes de ponerlo en la salida: recordemos que un valor de entrada puede oscilar entre 0 y 1023 (es decir, 10 bits), pero la salida debemos limitarnos a un valor comprendido entre 0 y 255 (es decir, 8 bits).

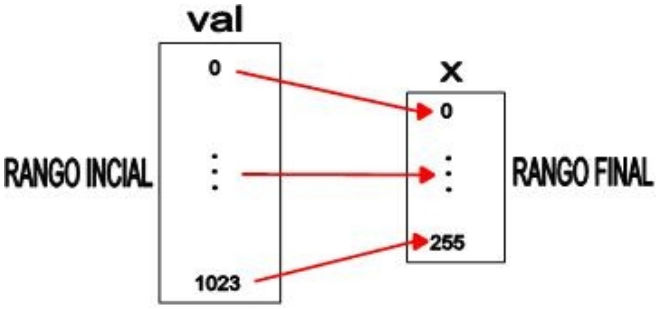
¿Qué ocurriría si no hago esta escala de valores? Pues que el valor de salida, una vez sobrepasado el valor 255, pasaría al otro extremo del intervalo, es decir, volvería al 0:

256=0, 257=2, 258=3, 259=4, 260=5, …, 510=254, 511=255.

De igual modo, una salida por debajo de cero volvería a dar la vuelta al intervalo:

-1=255, -2=254, -3=253, …, -256=0.

Otra forma de hacer este escalado (o mapeo) de valores, es a través de la función **map**, que mapea un rango inicial de valores a otro rango final de valores:

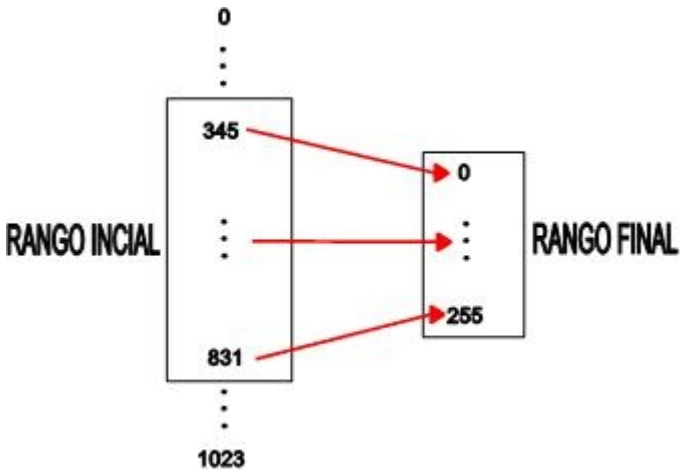



```
x = map(val,0,1023,0,255);
```

Si no quiero utilizar una nueva variable, puedo reasignar la variable val al nuevo valor mapeado:

```
val = map(val,0,1023,0,255);
```

Otro aspecto a tener en cuenta es que nuestra entrada, tal y como está construida (con un divisor de tensión) nunca alcanzaría los valores máximo y mínimo (quizás tenga un rango entre 345 y 831, por ejemplo). Si queremos que la intensidad de brillo de nuestro LED pueda oscilar entre todo su rango posible, se hace necesario la función map (no sería suficiente dividir entre 4):




```
val = map(val,345,831,0,255);
```

Debo tener presente que cuando realice un mapeo, y me salga fuera del rango de entrada, Arduino da la vuelta y aparece por el otro extremo, es decir, si considero el ejemplo:

```
val = map(val,50,70,0,255);
```

un valor de entrada de 45, daría el equivalente a una entrada de 65.

También puede ser muy útil, a la hora de evitar salirnos de los rangos habituales de uso, la función **constrain (**variable**,** valorMínimo**,** valorMáximo**)**:

```
x = constrain(val,a,b);
```

asigna a `x` el valor de `val` siempre y cuando `val` esté comprendido entre `a` y *b*. En el caso de que `val` salga de dicho intervalo, tomará los valores extremos de este:

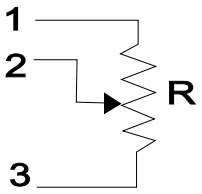
si `a<val<b`, entonces `x = val`
si `val<a`, entonces `x = a`
si `val>b`, entonces `x = b`

Ejemplo 7. La luz avanza por 4 LEDs en función de un potenciómetro

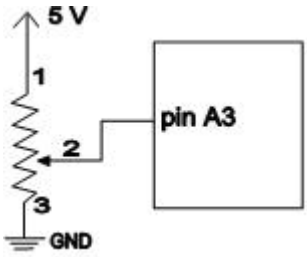
El potenciómetro es otro sensor de entrada muy usado. Consta de una resistencia variable, que dependerá de la posición de giro en que se encuentre su consola (o mango). Es muy importante conectar el potenciómetro de manera correcta para

evitar que éste se estropee (en caso de dejar pasar toda la corriente por una sección del potenciómetro de muy baja resistencia).

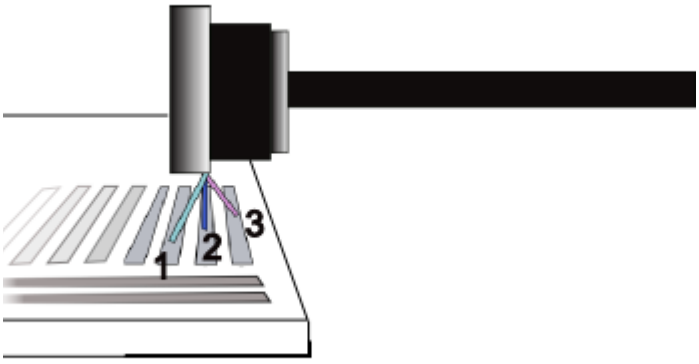
Un potenciómetro posee tres patas:



Debemos conectar la entrada analógica del potenciómetro de la siguiente manera:



También es importante recalcar la necesidad de insertar este tipo de potenciómetros de manera que las patas no dañen las hileras de las **breadboard** (porque una patilla demasiado ancha puede dañar irreparablemente la hilera de conexiones de la placa). Es conveniente montarla de la siguiente manera:



El truco para cablear dichas patillas es insertar el cable jumper para la patilla 2 previamente antes de intentar conectar el potenciómetro.

Otra forma más cómoda es utilizando conectores para placas breadboard (¡ojo!, deben ser de 5 mm)

El programa que controla este circuito es el siguiente:

```
int led1=5;
int led2=6;
int led3=9;
int led4=10;
int led5=11;
int val;
```

```
void setup(){
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
  pinMode(led5,OUTPUT);
}

void loop(){
  val=analogRead(A0);   //repartiré los 1023 valores posibles en 5 tramos
  if(val>=0 && val<204) {
    digitalWrite(led1,HIGH); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW);
  }
  if(val>=204 && val<408) {
    digitalWrite(led1,LOW); digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW);
  }
  if(val>=408 && val<612) {
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,HIGH); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW);
  }
  if(val>=612 && val<816) {
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,HIGH);
    digitalWrite(led4,LOW);
  }
  if(val>=816 && val<1023) {
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,HIGH);
  }
}
```

Como vemos, es un simple programa que divide el rango de entrada (que va desde 0 hasta 1023) en 5 posibilidades. En función del valor de lectura del potenciómetro, encenderá el LED correspondiente.

Otra estructura que se podría haber utilizado es: **switch**

```
switch(val) {          /*el programa cogerá un camino u otro en función
                         del valor de la variable val */
    case 3: …;         //en caso de que val==3 …
    break;             //provoca la salida del bloque switch (opcional)
    case 12: …;        //también se puede poner como case(val==12):
    break;
    …
    default: …;        /*en caso de no cumplirse ningún case, ejecutará
                         las funciones que se incluyan en el default */
}
```

En el caso de que tenga que **comparar la variable con un caracter**, deberé ponerlo de la siguiente forma:

```
case(val=='A'):        //es el caso en el que la variable es igual a la letra A
```

Veamos cómo quedaría el `void loop()` del proyecto anterior utilizando la estructura `switch`:

```
void loop(){
  val=analogRead(A0);
  val=map(val,0,1023,0,4);
  switch(val) {
  case 0:
    digitalWrite(led1,HIGH); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW); break;
  case 1:
    digitalWrite(led1,LOW); digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW); break;
  case 2:
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,HIGH); digitalWrite(led4,LOW);
    digitalWrite(led4,LOW); break;
  case 3:
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,HIGH);
    digitalWrite(led4,LOW); break;
  case 4:
    digitalWrite(led1,LOW); digitalWrite(led2,LOW);
    digitalWrite(led3,LOW); digitalWrite(led4,LOW);
    digitalWrite(led4,HIGH); break;
  }
}
```

7.COMUNICACIÓN ARDUINO-PC `Serial.begin(9600);`

En muchas ocasiones es muy útil poder visualizar a través del ordenador los valores de lectura en los pines de entrada y de salida de Arduino. Asimismo, también puede ser necesario mandar información a Arduino desde el teclado del PC. Veamos cómo poner en contacto ambos aparatos:

Antes de nada, debemos configurar en el `void setup()` que vamos a establecer dicha comunicación, utilizando la orden **Serial.begin( )**

```
void setup( ) {
 Serial.begin(9600);      //se especifica los baudios, normalmente 9600
}                           //recordemos que 1 baudio = 1 bit/segundo
```

Luego, dentro del `void loop()` podemos utilizar las siguientes funciones:

```
Serial.print(val);
```
//imprime el valor de la variable val

```
Serial.println(val);
```
//imprime el valor de val e inserta una línea nueva

```
Serial.print("hola, amigos");
```
//imprime el texto *hola, amigos*

```
Serial.print('\t');
```
//imprime una tabulación

```
Serial.print(val,BASE);
```
/*imprime el valor de la variable val pero pasando la variable en la base que le especifiquemos: `DEC`, `HEX`, `OCT`, `BIN`, `BYTE`, que corresponden a: *Decimal*, *Hexadecimal*, *Base 8*, *Binario*, y caracter del *código ASCII* respectivamente */

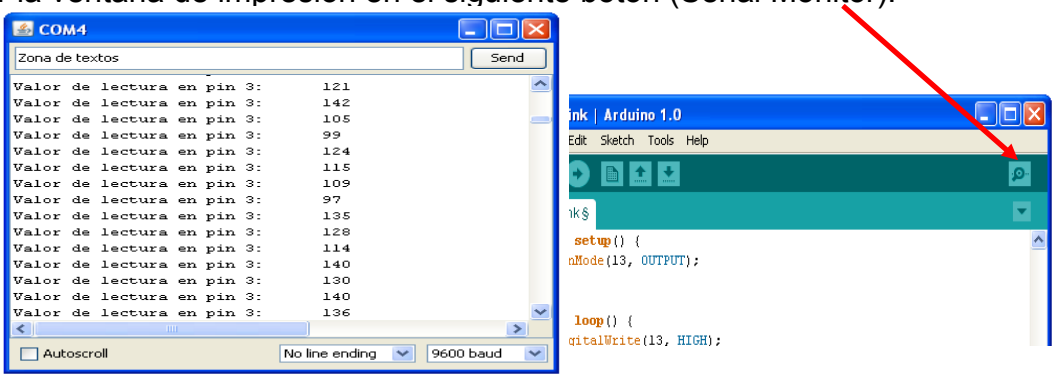
```
x = Serial.available();
```
/*asigna a *x* el número de bytes disponibles en el puerto serie que aún no han sido leídos. Después de haberlos leídos todos, la función Serial.available() devuelve un valor 0 hasta que lleguen nuevos datos al puerto serie */

```
y = Serial.read();
```
/*asigna a *y* el valor disponible en el puerto serie, que lo introducimos desde el teclado del ordenador en la zona de textos del *Serial Monitor* */

```
Serial.flush();
```
/*porque los datos pueden llegar al puerto serie a más velocidad que la del proceso del programa, Arduino puede guardar todos los datos de entrada en un buffer. Si es necesario limpiar el buffer para llenarlo de datos nuevos, debemos usar la función `flush();` */

Cuando Arduino establece comunicación con el PC necesita utilizar los pines 0 y 1 (**RX** y **TX**), por lo tanto no debemos utilizarlos como entradas o salidas para nuestro circuito.

Para poder visualizar en pantalla los datos que Arduino va imprimiendo, debemos abrir la ventana de impresión en el siguiente botón (Serial Monitor):



(Ver Ejemplo 8.)

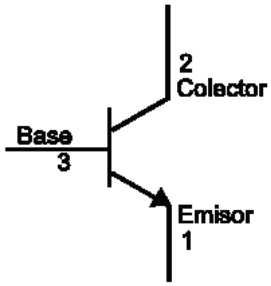
8. MOTORES DE CORRIENTE CONTINUA (DC motor)

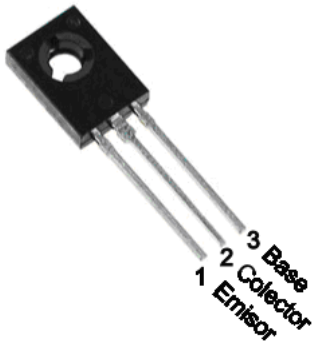
Un motor de corriente continua es un actuador que consume una intensidad relativamente elevada. La intensidad de corriente máxima que Arduino puede suministrar por un pin de salida es de 40 mA. Así pues, para poder gobernar a un DC motor, así como a cualquier otro actuador de considerable potencia, podemos utilizar:

- un **transistor**.
- un **relé**.
- un integrado **L293D** (consiste en un puente H de diodos, específico para el control de motores de corriente continua, inclusive el cambio de giro).

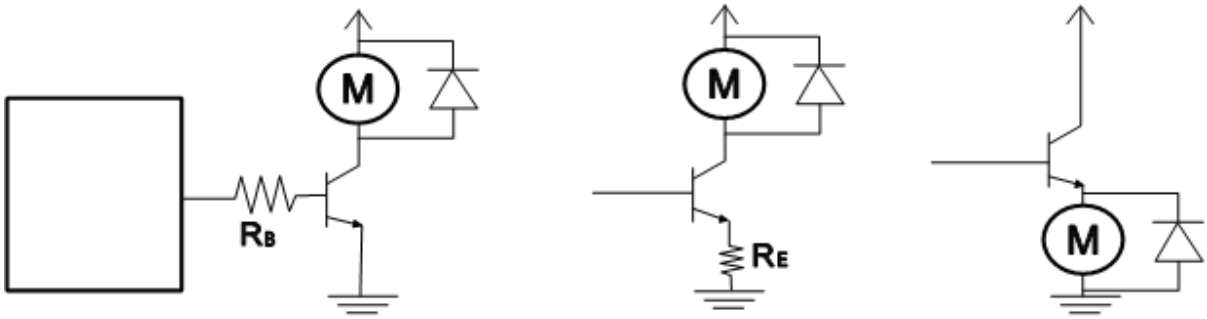
1. Transistor BD135

Éste es un transistor npn de mediana potencia, y a diferencia del BC547, puede soportar intensidades mayores. Debemos saber muy bien en qué patilla está situado el Emisor, el Colector y la Base (la palabra ***EsCoBa*** nos ayudará a recordarlo).



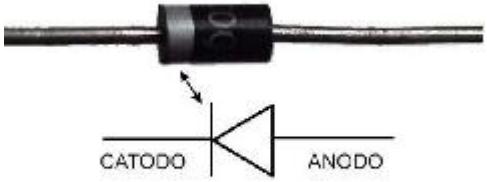


Podemos realizar las siguientes conexiones:



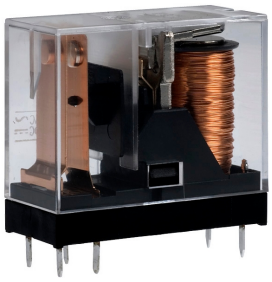
Quizás, la más recomendada sea la primera opción (utilizando una resistencia de base).

El diodo se utiliza para evitar las corrientes inducidas por las bobinas del motor, así que, en la medida de lo posible, debemos ponerlo (recordemos que la línea plateada corresponde al cátodo del diodo).



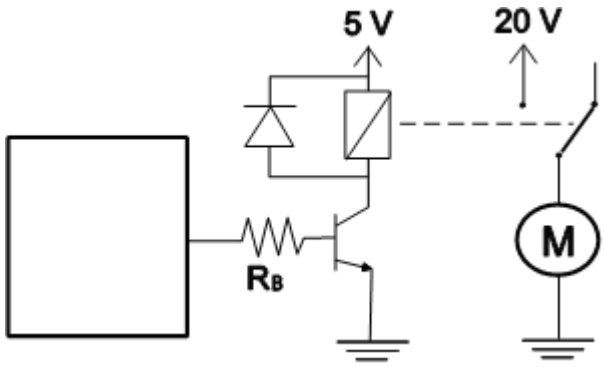
Con este montaje podemos **controlar**, tanto el **encendido** como el **apagado** del motor, así como su **velocidad** (utilizando un PWM como salida), y siempre **girando en el mismo sentido**. Recordemos que la salida modulada por pulsos es una imitación a una señal analógica formada por una serie de pulsos repartidos durante un tiempo determinado (los pulsos de salida son de 488 Hz). Esto supone que en este montaje del control de un motor a través de un transistor, si realmente se tratara de una salida analógica, o bien se coloca al transistor en corte o bien en saturación (no se podría controlar la velocidad del motor, sólo ponerlo en ON y en OFF); pero lo que ocurre realmente es que el "valor analógico" de la salida se transforma en una serie de pulsos (5V) repartidos durante cada pulso para que resulte un equivalente en su Vmedia, y durante esos picos, el transistor se satura e irá dando "impulsos de corriente" al motor, cuyo efecto final será el de girar a una velocidad controlada.

2. Relé



Un relé es un dispositivo electromecánico que permite gobernar el paso o no de corriente en un circuito independiente al circuito que utilizamos para el control.

La conexión debe realizarse de la siguiente forma:

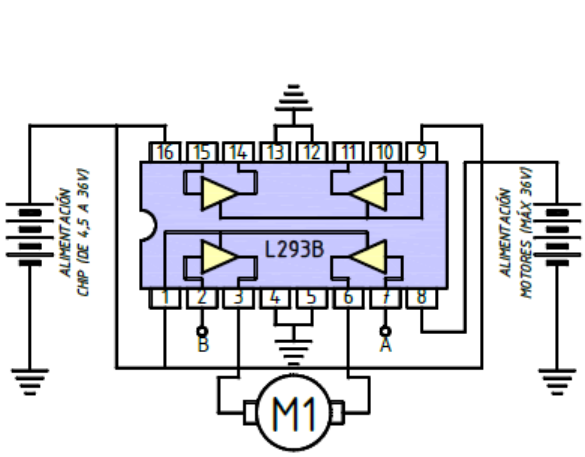


En esta ocasión, este montaje sólo permite **controlar** el **encendido** y **apagado** del motor, y siempre **girando en el mismo sentido** (no tiene sentido controlar la velocidad, pues el relé no tiene velocidad de cambio de posición de sus contactores como para "seguir" un PWM).

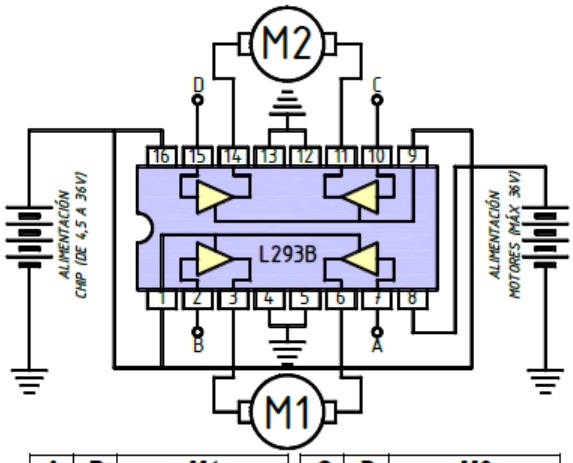
3. Integrado L293D

Con este integrado sí vamos a poder **gobernar el sentido de giro** del motor, así como su **velocidad,** pudiendo además un mismo integrado controlar dos motores simultáneamente.

El conexionado para este integrado es el siguiente:



A	B	M1
0	0	PARO
1	1	PARO
0	1	IZQUIERDA
1	0	DERECHA



A	B	M1
0	0	PARO
1	1	PARO
0	1	IZQUIERDA
1	0	DERECHA

C	D	M2
0	0	PARO
1	1	PARO
0	1	IZQUIERDA
1	0	DERECHA

Ejemplo 8. Control de un motor con el teclado

Con este programa vamos a poder modificar la velocidad de un motor, conectado con un transistor BD135, desde el teclado. Para ello, podremos ir introduciendo a través de la zona de textos del Serial Monitor números del 1 al 9, que corresponderán a los 9 niveles de potencia que se pueden aplicar al motor. Observar que Arduino manejará el valor de la entrada por teclado (1, 2, 3, … , 9) por sus códigos ASCII (que son, respectivamente, 49, 50, 51, … , 58).

El programa es el siguiente:

```
int valor;        //controlo la velocidad del motor por el teclado
int salida;
```

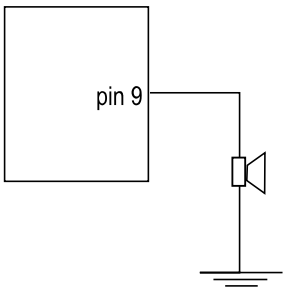
```
void setup(){
  pinMode(11,OUTPUT);
  Serial.begin(9600);
}

void loop(){
  if(Serial.available()) {    /*solo realizará el if si hay algo en la zona
                              de textos del Serial Monitor */
    valor=Serial.read();
    Serial.print(valor);
    Serial.print('\t'); //pongo una tabulación para hacer dos columnas
    salida=map(valor,49,58,0,255);
    Serial.println(salida);   //quiero ver el valor de salida
    analogWrite(11,salida);
  }
}
```

9. REPRODUCIR SONIDO `tone(3,2500);`

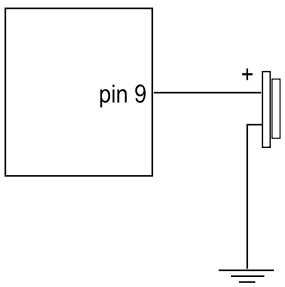
Otro dispositivo de salida muy utilizado puede ser un timbre piezoeléctrico, un zumbador o un altavoz. Estos dispositivos se pueden usar en salidas digitales (Ruido o NoRuido), pero presentan su máxima aplicabilidad en salidas analógicas, pues podremos reproducir distintas notas musicales.

Los dispositivos de salida más empleados son:

Altavoces



Transductores Piezos



Para ello utilizaremos la función **tone(**pin,frecuencia**);**

```
tone(6,1350);
```
//el altavoz conectado al pin 6 emitirá un sonido de 1350 Hz. Tras un `tone` se suele poner un `delay()` que permita escuchar esa frecuencia haciendo una pausa en el programa*/

Para que deje de sonar un *tone* debo utilizar **noTone(**pin**);**

```
noTone(6);        //el altavoz conectado al pin 6 dejará de sonar
```

Otra posibilidad es la de especificar la duración del tono:
tone(pin,frecuencia,duración**);**

```
tone(9,4500,200);   /*el altavoz conectado al pin 9 emitirá un sonido de
                    4500 Hz durante un tiempo de 200 milisegundos y
                    luego se para */
```

¡Ojo! Cuidado con esta opción porque sí es verdad que el sonido durará 200 ms pero el programa no se detiene para escucharlo, sino que sigue corriendo, y si antes de que pasen 200 ms el código vuelve a toparse con un tone, este último sonido pisará al anterior.

Las **frecuencias audibles** por el oído humano van de los **20 Hz** a los **20 KHz**.

Ejemplo 9. Instrumento musical invisible (Theremin)

Crearemos un instrumento musical que se controla con la posición de la mano (más o menos cerca de una LDR, que será empleada como sensor de proximidad). Cuanto más cerca esté la mano de la LDR, mayor oscuridad y menor será mayor será su resistencia, y le asociaremos una nota musical aguda; y cuanto más separada esté la mano, más grave será el sonido.

El programa puede ser el siguiente:

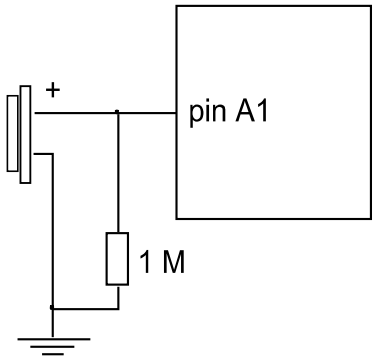
```
int luz;          //será el valor de entrada de la LDR
int sonido;       //será el valor de la frecuencia del altavoz
int ldr=A5;       //conectaré la LDR al pin A5
int altavoz=5;    //conectaré el altavoz al pin 5

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,20,500,3000,20);
  sonido=constrain(sonido,20,3000); //obligo a que no salga de esos valores
  Serial.print("frecuenca sonido: ");
  Serial.println(sonido);
  tone(altavoz,sonido,200);
  delay(250);     //doy tiempo a que suene la nota y a una pequeña pausa
}
```

Puede ser muy útil saber qué frecuencia, en hertzios, corresponde a cada nota musical por si queremos que Arduino nos recite alguna canción (se recuerda que para expresar los decimales se emplea el punto y no la coma). Es bueno saber que tampoco se va a notar mucho si redondeo estas frecuencias a números enteros.

Frecuencias (Hz) de las notas musicales								
♫	ESCALA							
NOTA	1	2	3	4	5	6	7	8
Do	65.406	130.813	261.626	523.251	1046.502	2093.005	4186.009	8372.018
Do#	69.296	138.591	277.183	554.365	1108.731	2217.461	4434.922	8869.844
Re	73.416	146.832	293.665	587.33	1174.659	2349.318	4698.636	9397.273
Re#	77.782	155.563	311.127	622.254	1244.508	2489.016	4978.032	9956.063
Mi	82.407	164.814	329.628	659.255	1318.51	2637.02	5274.041	10548.082
Fa	87.307	174.614	349.228	698.456	1396.913	2793.826	5587.652	11175.303
Fa#	92.499	184.997	369.994	739.989	1479.982	2959.955	5919.911	11839.822
Sol	97.999	195.998	391.995	783.991	1567.982	3135.963	6271.927	12543.854
Sol#	103.826	207.652	415.305	830.609	1661.219	3322.438	6644.875	13289.75
La	110	220	440	880	1760	3520	7040	14080
La#	116.541	233.082	466.164	932.328	1864.655	3729.31	7458.62	14917.24
Si	123.471	246.942	493.883	987.767	1975.533	3951.066	7902.133	15804.266

Sensor de vibración

El transductor piezo, además de utilizarse para reproducir sonidos también puede emplearse como sensor de vibración (como sensor de sonido no funciona bien, pues necesita que sus dos chapas vibren lo suficiente). Para ello, lo deberemos conectar de la siguiente manera:



10. AZAR `randomSeed(0);`

En muchas ocasiones es muy útil utilizar números aleatorios, seleccionados al azar. Comportamientos inesperados, suerte, espontaneidad,… son fruto de emplear números aleatorios. Veamos cómo trabajarlos con Arduino:

Antes de nada, debemos configurar en el setup que Arduino comience a generar números aleatorios, utilizando la orden **randomSeed( )**

```
void setup() {
 randomSeed(0);        //activa la generación de números semi-aleatorios
}
```

También es posible iniciar la generación de números aleatorios de una manera más impredecible, utilizando las señales recogidas en un pin flotante de entrada analógica, que recoja el ruido de fondo electromagnético (ondas de radio, rayos cósmicos, interferencias electromagnéticas de teléfonos móviles y luces fluorescentes, etc…). En este caso utilizaríamos:

```
randomSeed(analogRead(A5));  //randomiza usando el ruido del pin A5
```

Luego, dentro del void **loop**( ) podemos utilizar las siguientes funciones:

```
val = random(100,200);    /*asigna a la variable val un valor aleatorio
                            comprendido entre 100 y 199 */
val = random(200);        /*asigna a la variable val un valor aleatorio
                            comprendido entre 0 y 199 */
```

Ejemplo 10. LED imitando una candela

Utilizaremos números aleatorios para determinar la intensidad de brillo y la duración de dicho destello de un LED para conseguir el efecto de una candela (como por ejemplo se emplea en muchos portales de Belén). Sólo necesitamos montar un LED (naranja, mejor que rojo).

El programa es el siguiente:

```
int brillo;
int tiempo;

void setup(){
  prandomSeed(analogRead(A0));
  pinMode(11,OUTPUT);          //interesa ver los valores de la LDR
}

void loop(){
  brillo=arandom(100,256);     //el valor de salida oscila entre 100 y 255
  analogWrite(11,brillo);
  tiempo=random(50,151);       //el destello oscila entre 10 y 150 ms
  delay(tiempo);
}
```

11. OPERADORES MATEMÁTICOS

Los operadores aritméticos que se utilizan en Arduino son:

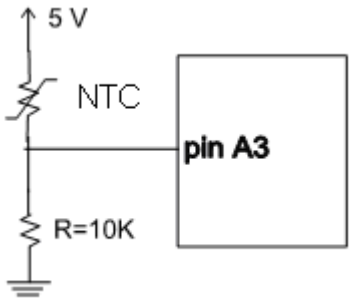
- asignación `=`
- suma `+`
- resta `-`
- multiplicación `*`
- división `/`
- resto o módulo `%` //no funciona con datos en coma flotante

Veamos los siguientes supuestos:

```
float x;
int y;
int z;
x = 7/2;   //x tomará el valor de 3.5 puesto que está declarado como float
y = 7/2;   //y tomará el valor de 3 puesto que no puede tener decimales
z = 7%2;   //z tomará el valor 1 puesto que es el resto de dividir 7 / 2
```

Ejemplo 11. Termómetro de luz

Construiremos un termómetro a través de 3 LEDs alineados, y a medida que la temperatura aumente, la luz irá subiendo por los distintos LEDs de manera gradual. Para convertir la temperatura en una señal eléctrica emplearemos una NTC, que se conectará de igual modo que se hizo con la LDR:



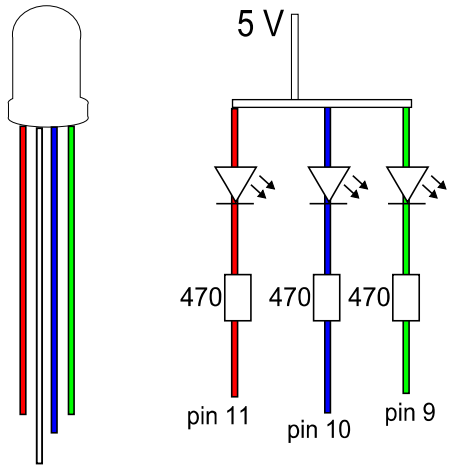
El programa será el siguiente:

```
void setup() {
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);          //utilizaré los pines 9,10 y 11 para los LEDs
  Serial.begin(9600);
}
```

```
void loop() {
  int lectura=analogRead(A0);
  Serial.println(lectura);    //me interesa conocer los valores máx y min
                              //para hacer bien el programa
  lectura=constrain(lectura,282,345); //impido que lectura salga del rango
  int x=map(lectura,282,345,0,30);
  int entero=x/10;      //solo podrá haber 4 casos: <1, <2, <3, =3
  int resto=x%10;       //podrá haber 10 casos: 0, 1, 2, …, 9
  int brillo=map(resto,0,9,0,255); //valores medios para el brillo del LED
  if(entero<1) {
    analogWrite(9,brillo);
    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
  }
  else if(entero<2){
    digitalWrite(9,HIGH);
    analogWrite(10,brillo);
    digitalWrite(11,LOW);
  }
  else if(entero<3){
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    analogWrite(11,brillo);
  }
  else {
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,HIGH);
  }
    delay(200);
}
```

Ejemplo 12. *Control del color y la intensidad de un LED RGB*

Un LED RGB (Red, Green, Blue) es un diodo LED que contiene 3 LED en sí mismo: uno rojo, otro verde y otro azul, y presenta 4 patas: un ánodo común (la parte del triángulo, que corresponde a la pata larga de un LED) y un cátodo (la parte del palo, que corresponde a la pata corta de un LED) para cada uno de los tres diodos. Conectaremos el ánodo común a 5 V, y los 3 cátodos a 3 respectivos pines de Arduino, de modo que un LED estará encendido cuando en estos pines haya menos de 5 V (un valor de `LOW` si se trata digitalmente, o un valor de menos de `255` para el caso analógico). Además, para este montaje utilizaremos 3 pulsadores: uno para el control del color y dos para variar la intensidad de brillo del LED.



El programa podría ser el siguiente:

```
int pinColor=7,pinBajar=6,pinSubir=5,rojo=11,verde=9,azul=10;
int contador=0,brillo=128; //al principio el LED iluminará a media potencia
int cambioColor,bajarLuminosidad,subirLuminosidad;
```

```
void setup(){
  for(int i=5;i<8;i++) pinMode(i,INPUT); //configuro los pines de entrada
  for(int j=9;j<12;j++) pinMode(j,OUTPUT); //...y los de salida
}

void loop(){
  cambioColor=digitalRead(pinColor);      //lecturas de los pulsadores
  bajarLuminosidad=digitalRead(pinBajar);
  subirLuminosidad=digitalRead(pinSubir);
  if(cambioColor==HIGH) contador++;       //contaré el número de pulsaciones
  if(bajarLuminosidad==HIGH) brillo=brillo-10; //disminuyo el brillo
  if(subirLuminosidad==HIGH) brillo=brillo+10; //aumento el brillo
  if(brillo<0) brillo=0;
  if(brillo>255) brillo=255;  //evito que brillo salga del rango 0 - 255
  for(int i=9;i<12;i++) digitalWrite(i,HIGH);   //apago los 3 LEDs
  if(contador%3==0) analogWrite(rojo,255-brillo);
      //un valor de brillo=255 debe traducirse a una salida de 0 V
  else if((contador+1)%3==0) analogWrite(verde,255-brillo);
  else analogWrite(azul,255-brillo);      /*como solamente quiero
      distinguir 3 opciones, consideraré que en 3 números consecutivos
      sólo hay uno que es múltiplo de 3, y dará resto=0 */
  delay(200);     /*debería haberlo hecho como en el ejemplo anterior, pero
      con esta espera, va bien. Si hubiera puesto menos tiempo, habría
      problemas, pues tardamos unos 200ms en apretar y soltar el pulsador*/
}
```

12. SERVOMOTORES `#include <Servo.h>`

Un servomotor es un motor que se caracteriza por su **precisión,** pues puede situarse en cualquier posición dentro de un rango de giro, normalmente **de 0º a 180º**. Así pues no son motores pensados para hacer mover un vehículo que recorra cierta distancia, sino para movimientos de precisión como pudiera ser el de un brazo robot, cuyo margen de maniobra no exceda dicho rango de giro.

Son tres los cables de que dispone el servomotor, y debemos conectarlos de manera correcta:

- Cable **rojo:** se conectará a la tensión de **5 V**.
- Cable **negro**: se conectará a tierra (**0 V**).
- Cable **blanco** o **naranja**: se conectará al **pin** de control (del 0 al 13).

Este dispositivo se controla habitualmente a través del envío pulsos de anchura (en el tiempo) determinada. Así, para nuestro servomotor tenemos que:

Duración del pulso	Posición del eje	Rango de duración del pulso de control: 900 µs – 2100 µs $Posición = \frac{3 \cdot duración - 2700}{20}$
900 µs	0º	
1200 µs	45º	
1500 µs	90º	
1800 µs	135º	
2100 µs	180º	

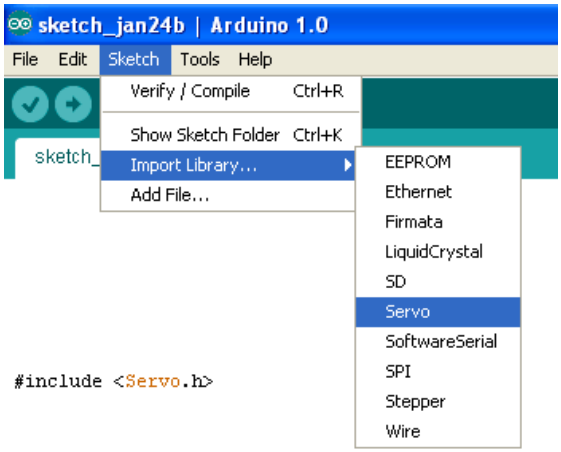
Las duraciones del pulso pueden cambiar en función del modelo del servomotor.

Aunque no es muy complicado sí hay que ser cuidadoso, y es relativamente fácil dañar al servomotor.

Para el control de los servomotores Arduino posee una **librería** específica. Una librería es una colección de funciones que están especialmente creadas para facilitar el manejo de ciertos dispositivos, y que no son cargadas por defecto a Arduino para ahorrar espacio en su memoria. Algunas de las librerías más utilizadas son las de control uso de: servomotores, motores paso a paso, pantallas de cristal líquido, matriz de LEDs, memorias SD, sensor de ultrasonido...

Veamos cómo controlar nuestro servomotor con Arduino:

- En primer lugar debemos abrir la librería `Servo.h`, a través de:
  Sketch / Import Library / Servo



- Luego debemos declarar, como si de una variable fuera, nuestro servomotor, asignándole un nombre. Y lo hacemos con el siguiente comando:

```
Servo nombreServo;
```

- Dentro del void setup() debemos configurar el pin de salida que Arduino va a utilizar para controlar el servomotor. Para ello se utiliza la función **attach(**pin**):**

```
nombreServo.attach(10);  // será controlado por el pin 10
```

Existe una variante en la que se especifica la duración mínima y máxima de los pulsos de control **attach(**pin,duraciónMín,duraciónMáx**):**

```
nombreServo.attach(10,900,2100);
```

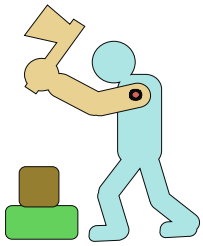
- Para controlar el servomotor, utilizaremos la función **write(**ángulo**)**:

```
nombreServo.write(45);// sitúa el eje del servo en la posición de 45º
```

¡Ojo! Entre posición y posición debemos poner alguna pausa para dar tiempo al servomotor a moverse.

Nota: el funcionamiento de los servomotores "low cost" puede que no se ajuste exactamente a lo esperado.

Ejemplo 13. Leñador de un portal de Belén



Utilizaremos nuestro servomotor para animar una figura de nuestro portal de Belén: se trata de un leñador que está cortando un tronco. Para ello el brazo articulado que sujeta el hacha deberá estar acoplado al servomotor. Lo programaremos de manera que el movimiento de bajada (el corte) sea rápido, y después de una pequeña pausa haga un retroceso lento.

El programa será el siguiente:

```
#include <Servo.h>

Servo brazo;

void setup(){
  brazo.attach(8);
}

void loop(){
  brazo.write(20);     //corresponde a la posición de hacha bajada
  delay(2000);
  for(int i=1;i<=120;i++) { //levanto el hacha lentamente
    brazo.write(i);
    delay(50);
  }
  delay(2000);         //tomo aliento
  brazo.write(160);     //cojo impulso antes de asestar el hachazo
  delay(500);
}
```

Bibliografía:
- *Curso de Arduino de Francis Perea.*
- *Getting Started With Arduino (Massimo Banzi).*
- *Beginning Arduino Programming (Brian Evans).*
- *30 Arduino Projects for he Evil Genios (Simon Monk).*