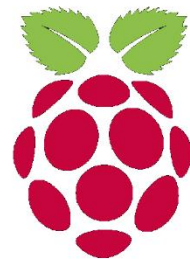
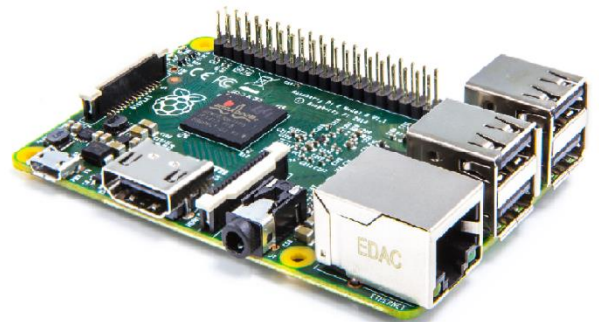
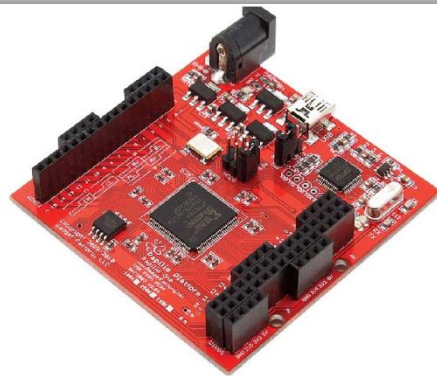


CURSO 2016-2017

# MEMORIA DE PRÁCTICAS LABORATORIO DE DESARROLLO DE HARDWARE



Autor: Carlos Crespo Jiménez

NIF: 28785141-C

Grado de Ingeniería de Computadores

Memoria de Prácticas de Laboratorio de  
Desarrollo de Hardware.

# Índice

<b>Plataforma Arduino .....</b>	<b>3</b>
1.1   Objetivos .....	3
1.2   Introducción .....	3
1.3   Instalación IDE Arduino, entorno de programación y configuración .....	4
1.4   Estructura de un programa en Arduino .....	7
1.5   Controlar el encendido-apagado del LED desde el PC vía puerto serie .....	7
1.6   Controlar el encendido-apagado de una bombilla por relé .....	9
1.7   Controlar la posición de un servo mediante un potenciómetro .....	11
1.8   Controlar la posición de un servo mediante el puerto serie .....	14
1.9   Controlar la velocidad de giro de un motor DC con un potenciómetro .....	15
1.10   Impresión en una pantalla LCD .....	19
1.11   Sensor de temperatura monitorizado .....	23
1.12   Contador en display 7-segmentos .....	25
1.13   Conclusión .....	32

## 1.1 Objetivos

Los objetivos mínimos en la plataforma arduino son:

- Conocer la plataforma arduino, sus características, sus variantes y sus modos de programación.
- Conocer una serie de componentes básicos de hardware típicos de aplicaciones de sistemas empujados.
- Preparar el PC para que funcione el entorno de desarrollo de arduino.
- Realizar ejemplos básicos de funcionamiento sobre arduino.
- Desarrollar otros ejemplos de uso de arduino manejando diversos componentes hardware.

## 1.2 Introducción



### ¿Qué es Arduino?

Arduino (en EEUU, Genuino a nivel internacional) es una compañía de hardware libre y una comunidad tecnológica que diseña y manufactura placas de desarrollo de hardware y software, compuesta respectivamente por circuitos impresos que integran un microcontrolador y un entorno de desarrollo (IDE), en donde se programa cada placa.

Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Toda la plataforma, tanto para sus componentes de hardware como de software, son liberados bajo licencia de código abierto que permite libertad de acceso a los mismos.

El hardware consiste en una placa de circuito impreso con un microcontrolador, usualmente Atmel AVR, puertos digitales y analógicos de entrada/salida, los cuales pueden conectarse a placas de expansión (shields), que amplían las características de funcionamiento de la placa Arduino. Asimismo, posee un puerto de conexión USB desde donde se puede alimentar la placa y establecer comunicación serial con el computador.

Por otro lado, el software consiste en un entorno de desarrollo (IDE) basado en el entorno de Processing y lenguaje de programación basado en Wiring, así como en el cargador de arranque (bootloader) que es ejecutado en la placa.<sup>4</sup> El microcontrolador de

la placa se programa a través de un computador, haciendo uso de comunicación serial mediante un convertidor de niveles RS-232 a TTL serial.

Las principales ventajas de utilizar Arduino son:

- ✓ Un entorno de desarrollo muy fácil de manejar (basado en processing).
- ✓ Un amplio conjunto de librerías de manejo de periféricos.
- ✓ Una comunidad de desarrolladores muy grande.

En la práctica vamos a utilizar la placa ATMEGA 328P, y sus características técnicas son:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g



**Ilustración 1.** Especificaciones técnicas placa ATMEGA 328P.







### 1.3 Instalación IDE Arduino, entorno de programación y configuración

Pasos a seguir para realizar la instalación del IDE de Arduino:

- 1) Descargamos la última versión del IDE de Arduino, en nuestro caso, arduino-1.6.12 en la siguiente url: <https://www.arduino.cc/en/Main/Donate>. Pulsamos "Just Download".
- 2) Descomprimos el archivo que nos hemos descargamos.
- 3) Posteriormente nos vamos al terminal de Linux (Ubuntu) y ejecutamos el archivo install.sh.

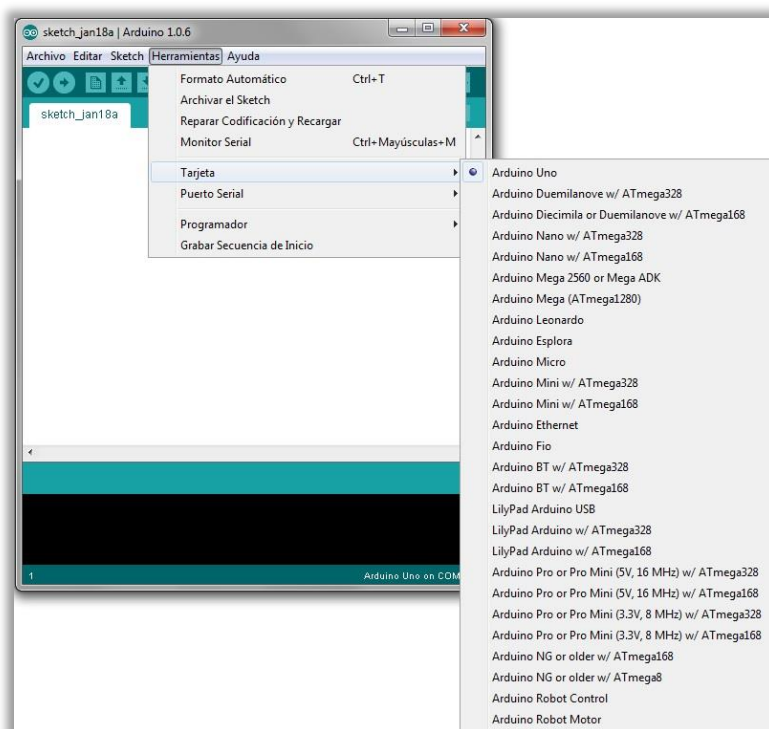


En los botones de acceso rápido tenemos los siguientes iconos:

-  Verifica si tu programa está bien escrito y puede funcionar.
-  Carga el programa a la placa de Arduino tras compilarlo.
-  Crea un programa nuevo.
-  Abre un programa.
-  Guarda el programa en el disco duro del ordenador.
-  (En la parte derecha de la barra de herramientas se encuentra el Monitor Serial) abre una ventana de comunicación con la placa Arduino en la que podemos ver las respuestas que nuestro Arduino nos está dando, siempre que tengamos el USB conectado.

En el cuadro del editor de texto escribiremos el código del programa que queramos que Arduino ejecute. Finalmente, en el área de mensajes y la consola Arduino nos irá dando información sobre si la consola está compilando, cargando...y sobre los fallos o errores que se produzcan tanto en el código como en el propio IDE.

El siguiente paso que realizaremos será configurar nuestro IDE para que se comunique con nuestra placa Arduino. Para ello conectaremos nuestro Arduino mediante el cable USB al PC y después de que el sistema operativo haya reconocido e instalado la tarjeta automáticamente, nos dirigimos a la zona de menú, pulsamos en Herramientas y después en Tarjeta. Ahí seleccionamos el modelo de tarjeta Arduino que tengamos, en nuestro caso "Arduino Uno/Genuino".



## 1.4 Estructura de un programa en Arduino

La estructura básica de programación de Arduino es simple y divide la ejecución en dos partes: setup y loop.

Setup() constituye la preparación del programa y loop() es la ejecución. La función Setup() incluye la declaración de variables y es la primera función que se ejecuta en el programa. Esta función se ejecuta una única vez y es empleada para configurar el pinMode (p. ej. si un determinado pin digital es de entrada o salida) e inicializar la comunicación serie. La función loop() incluye el código a ser ejecutado continuamente (leyendo las entradas de la placa, salidas).

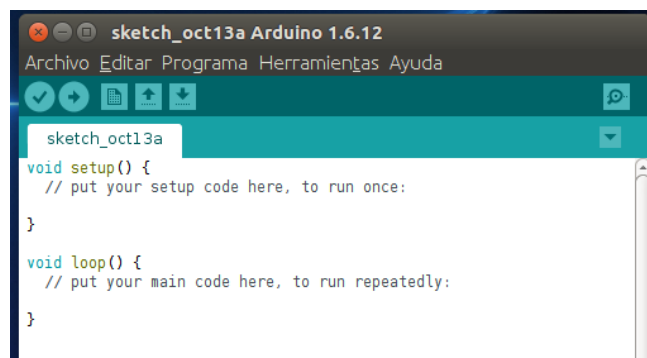


Ilustración 2. Estructura de un programa en Arduino.

## 1.5 Controlar el encendido-apagado del LED desde el PC vía puerto serie

### Desarrollo

En esta sección trataremos de manejar las salidas de nuestra placa Arduino UNO a través de nuestro PC comunicándonos con el mediante el puerto serie. Esta placa en concreto tiene dos puertos series, uno en el propio conector USB y otro en los pines digitales 0 y 1 (marcados con las letras 'TX' y 'RX').

Utilizaremos a nivel software la librería incluida en la aplicación de desarrollo de Arduino 'Serial', la cual tiene diversas funciones para la lectura y escritura del puerto serie de nuestra placa, entre ellas 'Serial.read' y 'Serial.write'.

Por último, implementaremos un pequeño programa en Python para controlar el encendido y el apagado del LED. Además, podremos manejar la salida digital de nuestro pin seleccionado (en nuestro caso el 13).

## Código en Arduino

```
int led = 13;
void setup () {
  pinMode(led, OUTPUT); //LED 13 como salida
  Serial.begin(9600); //Inicializo el puerto serial a 9600 baudios
}

void loop () {
  if (Serial.available()) { //Si está disponible
    char c = Serial.read(); //Guardamos la lectura en una variable char
    if (c == 'H') { //Si es una 'H', enciendo el LED
      digitalWrite(led, HIGH);
    } else if (c == 'L') { //Si es una 'L', apago el LED
      digitalWrite(led, LOW);
    }
  }
}
```

Ilustración 4. Código en Arduino para encender/apagar LED desde el PC vía puerto serie.

## Código en Python

```
import serial

arduino = serial.Serial('/dev/ttyACM0', 9600)

print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    if comando == 'H':
        print('LED ENCENDIDO')
    elif comando == 'L':
        print('LED APAGADO')

arduino.close() #Finalizamos la comunicacion
```

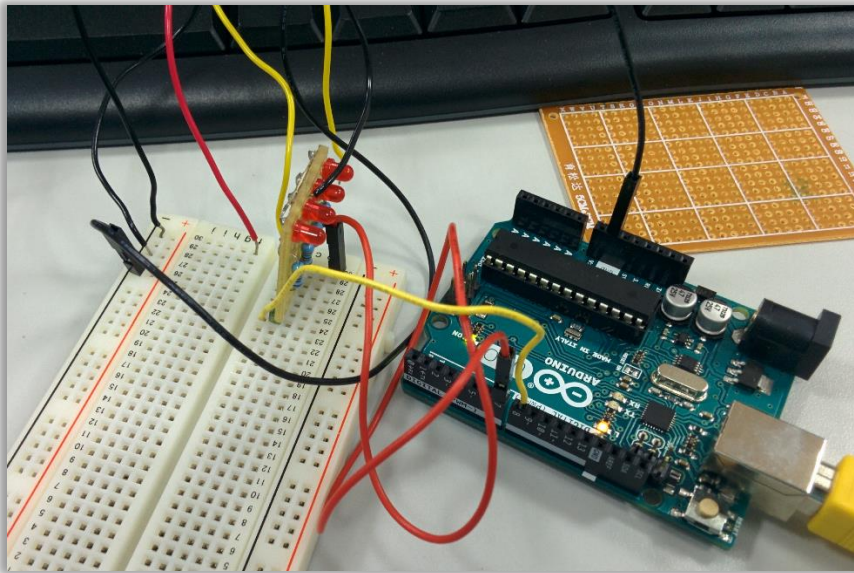
Ilustración 5. Código en Python para encender/apagar LED desde el PC vía puerto serie.

## Ejecución desde el terminal

```
practicass@mCR-89:~$ sudo python parte1.py
Starting!
Introduce un comando: H
LED ENCENDIDO
Introduce un comando: L
LED APAGADO
```



## Montaje final



### 1.6 Controlar el encendido-apagado de una bombilla por relé

#### 🤔 ¿Qué es un relé?

El relé o relevador es un dispositivo electromagnético. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.



Ilustración 6. Relé utilizado en esta práctica.

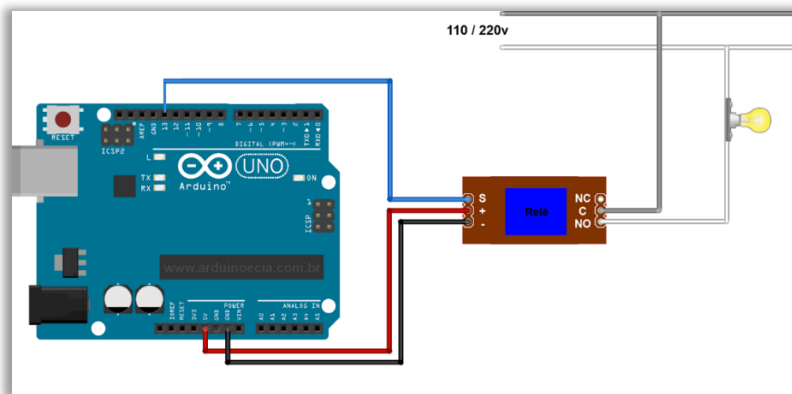
Con este dispositivo electrónico (relé) podemos manejar, con nuestra placa Arduino UNO, altas tensiones, muy útil para cualquier proyecto de domótica (tensiones de 220 V), como por ejemplo el caso de nuestro proyecto de encender o apagar una bombilla. Este ejemplo, por muy simple que parezca, nos abre una amplia variedad, mucho más cuando se le añaden dispositivos como son los sensores (de proximidad, de

luz, etc.), que nos permitirá activar cualquier aparato eléctrico que queramos y cuando queramos.

En definitiva, el relé amplía las tensiones manejadas por nuestra placa para hacer más grande nuestro campo de actuación con diversos aparatos eléctricos.

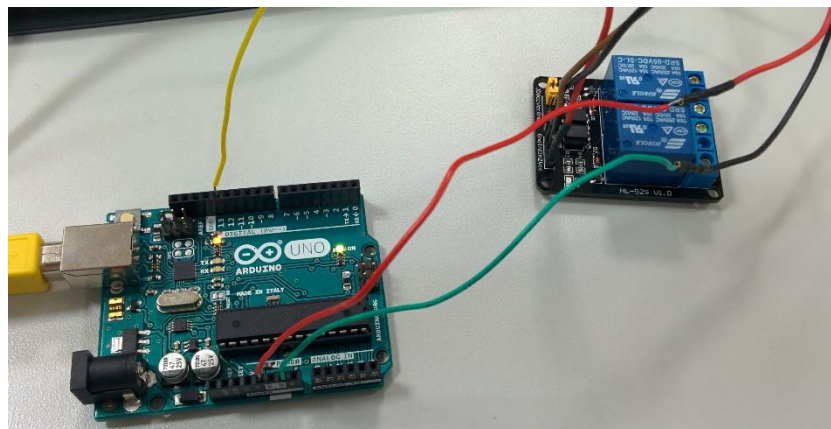
#### Desarrollo

Una vez que hemos comprobado que funciona el encendido y apagado del LED a través del puerto serie con nuestro programa en Python, la modificación será simplemente física, conectando nuestro pin a la patilla 'IN1' ó 'IN2', y en la salida del relé conectaremos la bombilla (tal y como aparece en la imagen). De este modo, de igual forma que encendíamos el LED en el ejercicio anterior lo haremos en este, mediante comando por consola ('H' para el encendido y 'L' para el apagado) ejecutando nuestro código escrito en Python. Los códigos son los aportados en el apartado anterior.



**Ilustración 7.** Conexión placa Arduino con relé y bombilla.

#### Montaje final

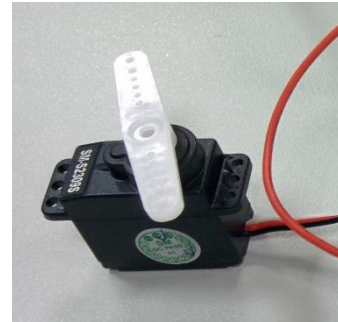


## 1.7 Controlar la posición de un servo mediante un potenciómetro



### ¿Qué es un servo?

Un 'servomotor de modelismo' (conocido generalmente como servo o servo de modelismo) es un dispositivo actuador que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y de mantenerse estable en dicha posición. Está formado por un motor de corriente continua, una caja reductora y un circuito de control, y su margen de funcionamiento generalmente es de menos de una vuelta completa.



Los servos de modelismo se utilizan frecuentemente en sistemas de radiocontrol y en robótica, pero su uso no está limitado a estos.

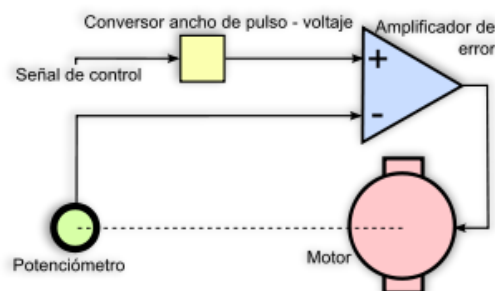
El componente principal de un servo es un motor de corriente continua, que realiza la función de actuador en el dispositivo: al aplicarse un voltaje entre sus dos terminales, el motor gira en un sentido a alta velocidad, pero produciendo un bajo par. Para aumentar el par del dispositivo, se utiliza una caja reductora, que transforma gran parte de la velocidad de giro en torsión.



### ¿Cómo controlamos la posición?

El dispositivo utiliza un circuito de control para realizar la ubicación del motor en un punto, consistente en un controlador proporcional.

El *punto de referencia* o *setpoint* (que es el valor de posición deseada para el motor) se indica mediante una señal de control cuadrada. El ancho de pulso de la señal indica el ángulo de posición: una señal con pulsos más anchos (es decir, de mayor duración) ubicará al motor en un ángulo mayor, y viceversa.



Inicialmente, un amplificador de error calcula el valor del error de posición, que es la diferencia entre la referencia y la posición en que se encuentra el motor. Un error de posición mayor significa que hay una diferencia mayor entre el valor deseado y el existente, de modo que el motor deberá rotar más rápido para alcanzarlo; uno menor, significa que la posición del motor está cerca de la deseada por el usuario, así que el motor tendrá que rotar más lentamente. Si el servo se encuentra en la posición deseada, el error será cero, y no habrá movimiento.

Para que el amplificador de error pueda calcular el error de posición, debe restar dos valores de voltaje analógicos. La señal de control PWM se convierte entonces en un

valor analógico de voltaje, mediante un convertidor de ancho de pulso a voltaje. El valor de la posición del motor se obtiene usando un potenciómetro de realimentación acoplado mecánicamente a la caja reductora del eje del motor: cuando el motor rote, el potenciómetro también lo hará, variando el voltaje que se introduce al amplificador de error. Una vez que se ha obtenido el error de posición, éste se amplifica con una ganancia, y posteriormente se aplica a los terminales del motor.

## 🤔 ¿Qué es un potenciómetro?

Un potenciómetro es uno de los dos usos que posee la resistencia o resistor variable mecánica (con cursor y de al menos tres terminales). Conectando los terminales extremos a la diferencia de potencial a regular (control de tensión), se obtiene entre el terminal central (cursor) y uno de los extremos una fracción de la diferencia de potencial total, se comporta como un divisor de tensión o voltaje.



Según la potencia que disipe en su funcionamiento, como regulador de tensión, así debe ser la potencia de la resistencia variable mecánica a utilizar.

## 🚦 Desarrollo

Con estos dos dispositivos (servo y potenciómetro) podremos controlar la posición de nuestro servo, la velocidad de rotación de nuestro motor de continua y por último la dirección en la que gira usando un puente H.

Para hacer esto, tendremos que ajustar los datos que vayamos recibiendo del potenciómetro para convertirlos en ángulos, es decir, tendremos como valor máximo 180° y como valor mínimo 0°, haciendo que los valores dentro del rango sean proporcionales.

Utilizaremos la función “map” de Arduino, la cual modifica nuestro rango de valores al que nosotros queramos, dándole los dos rangos. En nuestro ejercicio, el primer rango será la salida del potenciómetro digitalizada en 10 bits, es decir, de 0 a 1023, y el segundo rango será de 0 a 180 grados.

Para este ejercicio necesitaremos importar la librería “Servo.h” que viene por defecto en nuestro compilador. En ella usaremos las funciones “write” para escribir el valor que le pasemos como parámetro y “attach” para seleccionar el pin al que vamos a conectar el puerto serial (en nuestro caso, será el pin 9).

## 🚦 Código en Arduino

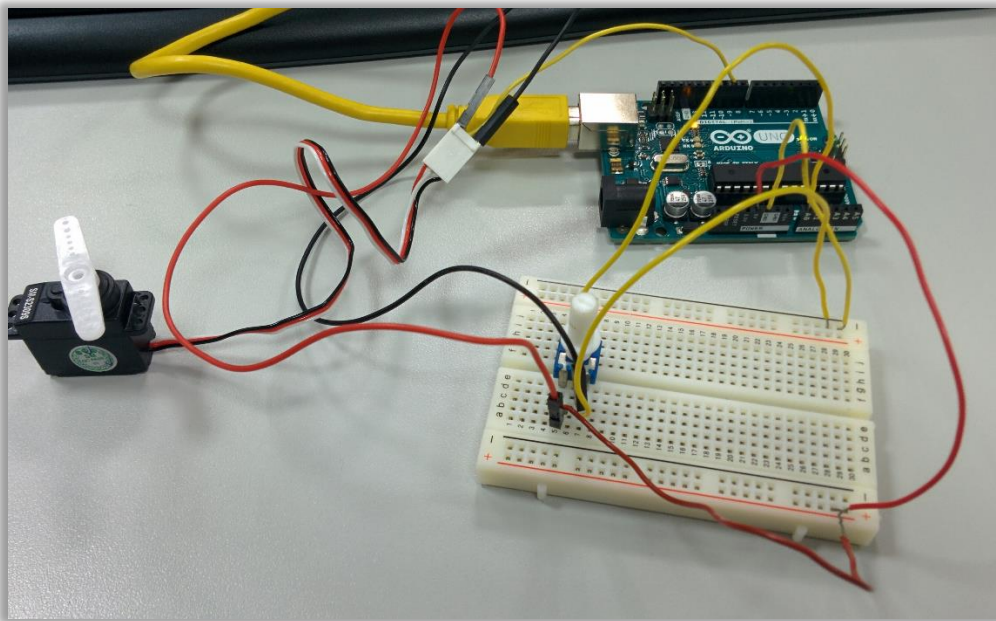
```
Servo servo; // Creamos un objeto Servo para controlar el motor

int potenciometro = 0; // seleccionamos el pin analogico que vamos a usar
int aux;               // variable auxiliar para almacenar los datos del potenciometro

void setup() {
  servo.attach(9); // seleccionamos el pin donde conectemos el servomotor
}

void loop() {
  aux = analogRead(potenciometro); // leemos el valor del potenciometro
  aux = map(aux, 0, 1023, 0, 180); // escalamos el valor para pasarlo al rango de angulos
  servo.write(aux);                // le pasamos al servo el valor ya mapeado antes
  delay(50);
}
```

## 🚦 Montaje final





## 1.8 Controlar la posición de un servo mediante el puerto serie

### Desarrollo

En este ejercicio no utilizaremos el potenciómetro ya que pasaremos por puerto serie el valor del ángulo y lo escribiremos directamente en el servo. En el circuito sólo desconectaremos el potenciómetro. En cuanto al código, hemos utilizado dentro del bucle la función “parseInt”, que pasa a número entero el dato leído por nuestro puerto serie.

### Código en Arduino

```
#include <Servo.h>

Servo servo; // Creamos un objeto Servo para controlar el motor

int potenciometro = 0; // seleccionamos el pin analogico que vamos a usar
int aux;                // variable auxiliar para almacenar los datos del potenciometro

void setup() {
  Serial.begin(9600);
  servo.attach(9); // seleccionamos el pin donde conectemos el servomotor
}

void loop() {
  if(Serial.available())
  {
    aux=Serial.parseInt(); //Leemos y pasamos el valor del puerto serie a entero
    servo.write(aux);      //Escribimos en nuestro servo el valor
  }
  delay(50);
}
```

### Código en Python

```
import serial

arduino = serial.Serial('/dev/ttyACM1', 9600)
print("Comenzando")
while True:
    comando = raw_input('Introduce angulo: ')
    arduino.write(comando)
    print "GIRADO: ", comando
arduino.close()
```

🚦 Ejecución desde el terminal

```
practicass@mCR-90:~/Escritorio$ python giro.py
Comenzando
Introduce angulo: 90
GIRADO: 90
Introduce angulo: 45
GIRADO: 45
Introduce angulo: 0
GIRADO: 0
Introduce angulo: 180
GIRADO: 180
Introduce angulo: 0
GIRADO: 0
Introduce angulo: 90
GIRADO: 90
```

## 1.9 Controlar la velocidad de giro de un motor DC con un potenciómetro

🤔 ¿Qué es un motor de continua?

El motor de corriente continua (denominado también motor de corriente directa, motor CC o motor DC) es una máquina que convierte la energía eléctrica en mecánica, provocando un movimiento rotatorio, gracias a la acción que se genera del campo magnético.



Una máquina de corriente continua (generador o motor) se compone principalmente de dos partes. El estator da soporte mecánico al aparato y contiene los devanados principales de la máquina, conocidos también con el nombre de polos, que pueden ser de imanes permanentes o devanados con hilo de cobre sobre un núcleo de hierro. El rotor es generalmente de forma cilíndrica, también devanado y con núcleo, alimentado con corriente directa mediante escobillas fijas (conocidas también como carbones).

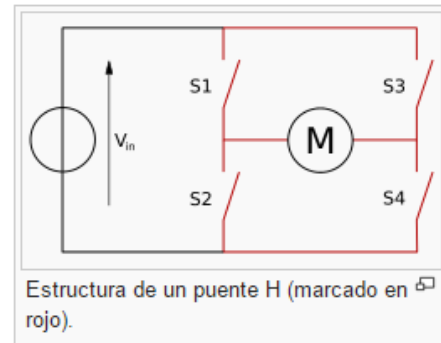
El principal inconveniente de estas máquinas es el mantenimiento, muy caro y laborioso, debido principalmente al desgaste que sufren las escobillas al entrar en contacto con las delgas.

Algunas aplicaciones especiales de estos motores son los motores lineales, cuando ejercen tracción sobre un riel, o bien los motores de imanes permanentes. Los motores de corriente continua (CC) también se utilizan en la construcción de servomotores y motores paso a paso. Además, existen motores de DC sin escobillas llamados brushless utilizados en el aeromodelismo por su bajo torque y su gran

velocidad. Es posible controlar la velocidad y el par de estos motores utilizando técnicas de control de motores CD.

### 🤔 ¿Qué es un puente en H?

Un Puente en H es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, avance y retroceso. Son ampliamente usados en robótica y como convertidores de potencia. Los puentes H están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos.



El término "puente H" proviene de la típica representación gráfica del circuito. Un puente H se construye con 4 interruptores (mecánicos o mediante transistores). Cuando los interruptores S1 y S4 (ver primera figura) están cerrados (y S2 y S3 abiertos) se aplica una tensión positiva en el motor, haciéndolo girar en un sentido. Abriendo los interruptores S1 y S4 (y cerrando S2 y S3), el voltaje se invierte, permitiendo el giro en sentido inverso del motor.

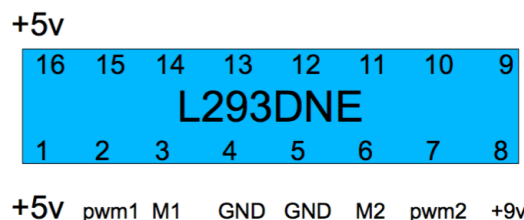
Con la nomenclatura que estamos usando, los interruptores S1 y S2 nunca podrán estar cerrados al mismo tiempo, porque esto cortocircuitaría la fuente de tensión. Lo mismo sucede con S3 y S4.



### Información sobre el chip L293DNE

Este chip contiene dos puentes en H, por lo que podemos alternar hasta dos motores independientes uno del otro.

Estas son las patas del chip que utilizaremos en nuestro sistema. Dicha placa estará alimentada con 9V. Las señales M1 y M2 son las conexiones con el motor de continua, pwm1 y pwm2 las salidas controladas por el potenciómetro.





## Desarrollo

En este ejercicio, cambiaremos de motor. El motor de continua es un motor que funciona a más revoluciones. En este ejercicio vamos a controlar la velocidad de giro con nuestro potenciómetro. Éste regulará la señal que alimenta a nuestro motor de continua. Para hacerlo más sencillo y útil utilizaremos el chip L293DNE, que en su interior alberga un puente en H que usaremos para cambiar el sentido de giro de nuestro motor. El sistema tendremos que alimentarlo con una pila de 9V que controle la velocidad de giro del motor de continua a través de nuestro potenciómetro.

Una vez tengamos todo el circuito montado, tendremos que comprobar que en el caso de que el potenciómetro esté situado en el valor medio, el motor de continua deberá de quedarse parado, ya que el valor de entrada de ambos polos sería nulo. Si giramos el potenciómetro hacia la derecha, el motor deberá empezar a girar con mayor velocidad hacia la derecha, y si giramos el potenciómetro hacia la izquierda, el motor deberá girar hacia la izquierda.

## Código en Arduino

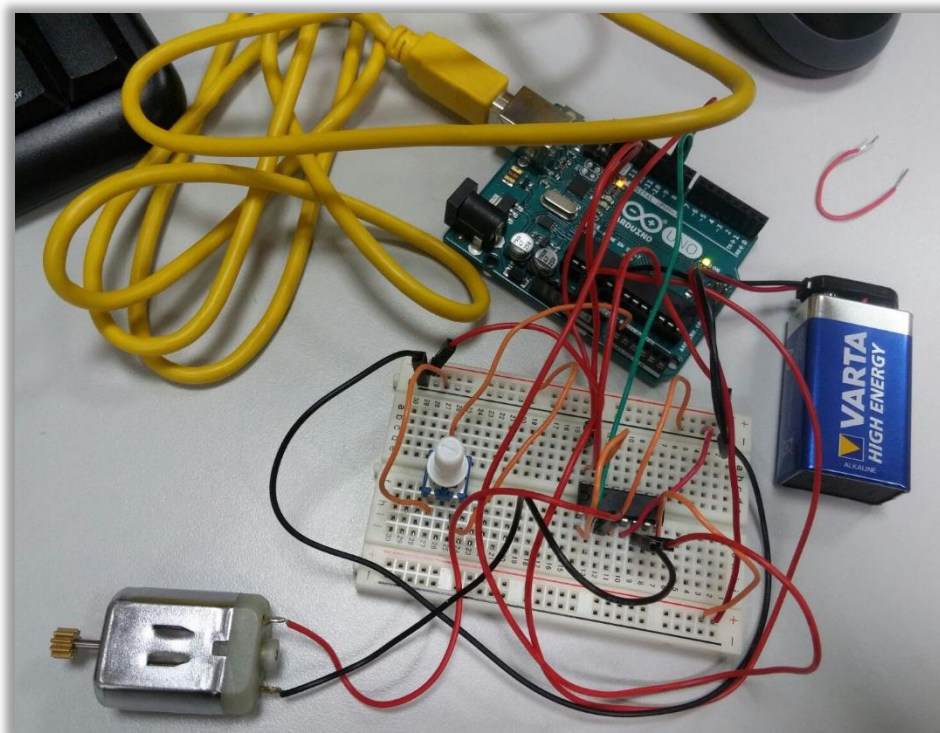
```
int pin2 = 9;
int pin7 = 10;
int pot = A0;
int valorpot;
int pwm1;
int pwm2;

void setup() {
  // put your setup code here, to run once:
  pinMode (pin2, OUTPUT);
  pinMode (pin7, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  valorpot = analogRead (pot);
  pwm1 = map (valorpot, 0, 1023, 0, 255);
  pwm2 = map (valorpot, 0, 1023, 255, 0);

  analogWrite (pin2, pwm1);
  analogWrite (pin7, pwm2);
}
```

## Montaje final



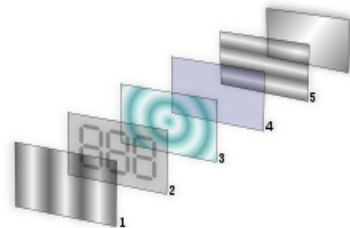
## 1.10 Impresión en una pantalla LCD

### 🤔 ¿Qué es una pantalla LCD?

Una pantalla de cristal líquido o LCD (sigla del inglés liquid crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.

### 🤔 ¿Cómo está hecha?

- 1) Filtro vertical que polariza la luz que entra.
- 2) Sustrato de vidrio con electrodos de Óxido de Indio ITO. Las formas de los electrodos determinan los dígitos que aparecen en pantalla.
- 3) Cristales líquidos.
- 4) Sustrato de vidrio con film electrodo común con los cantos horizontales para alinearse con el filtro horizontal.
- 5) Film de filtro horizontal para bloquear/permitir el paso de luz.
- 6) Fuente luminosa (en la de nuestro caso).



### 🤔 ¿Cómo funciona una pantalla LCD?

Una pantalla LCD puede escribir dos líneas con 16 caracteres cada una. Cada carácter consiste en 5x8 o 5x11 píxeles. En nuestro caso lo cubre un visualizador de 5x8 píxeles. Dispone de tres bloques de memoria:

- DDRAM Display Data RAM (RAM de datos de visualización).
- CGRAM Character Generator RAM (generador de caracteres RAM).
- CGROM Character Generator ROM (generador de caracteres ROM).

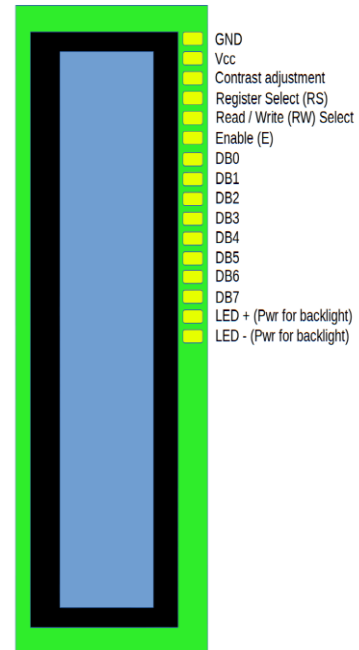
La memoria DDRAM se utiliza para almacenar los caracteres a visualizar. Tiene capacidad para almacenar 80 caracteres, aunque algunas localidades de memoria están directamente conectadas a los caracteres en el visualizador, la GCROM guarda los caracteres estándar en una matriz, y la GCGRAM guarda caracteres propios que no estén definidos.



## Información sobre la pantalla LCM1602C

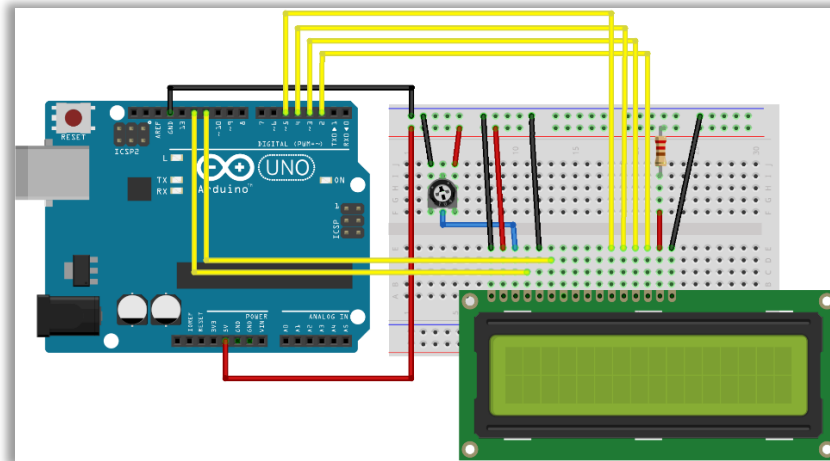
Vamos a explicar qué significa cada pin del LCD LCM1602C:

- ✓ **RS (Register Select):** es el pin que controla la memoria del LCD, e indica que registro de la memoria se lee o se escribe.
- ✓ **RW (Read/Write):** es el pin de lectura y escritura, que dirá si se escribe en memoria o si se lee en cada momento.
- ✓ **E (Enable):** pin que habilita los registros.
- ✓ **DB0-DB7:** son los pines de datos, que escribiremos en los registros. En el caso de la pantalla que tenemos los designa como DB0-DB7 y corresponden a la numeración 7-14, pero esto puede variar en función del fabricante o modelo de la LCD.
- ✓ **Vo:** es el pin de contraste del display, con el podremos modificar el contraste de la pantalla.
- ✓ **Vdd:** es el pin de alimentación, por el que entra la tensión que hará funcionar al LCD. Normalmente es de +5v.
- ✓ **GND:** complementa al anterior pin, siendo el otro pin de alimentación que se conectará a tierra.
- ✓ **BL1 y BL2:** son los pines de retro iluminación. Controlan la luminosidad del LCD. En algunas pantallas puede aparecer con los símbolos Bklt+ y Bklt-. Las siglas provienen de "Back Light".



### Desarrollo

En esta ilustración podemos ver el cableado que es necesario para conectar nuestro LCD a la placa Arduino, controlando la luminosidad de nuestra pantalla por un potenciómetro.



Una vez que tenemos todo bien conectado, nos iremos a nuestro compilador Arduino e incluiremos la librería para los displays LCD (“LiquidCrystal.h”).

De esta librería usaremos funciones como “setCursor”, que te posiciona dentro de la matriz de la pantalla para que podamos escribir donde queramos en el display.

Lo que vamos a hacer en este ejercicio es mandar por comando desde el PC (conectado por puerto serie con el Arduino) un texto cualquiera, y el display LCD lo escribirá en pantalla. Para mandar el texto escribiremos un pequeño programa escrito en lenguaje Python.

#### 🚦 Código en Arduino

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Inicializamos una variable LC con pines elegidos

void setup() {
  lcd.begin(16, 2); // seleccionamos numero de filas y columnas
  Serial.begin(9600); // Iniciamos comunicacion por puerto serie a 9600 baudios
}

void loop()
{
  if (Serial.available()) { // mientras haya algo que leer en el puerto serie
    delay(100);
    lcd.clear(); // borramos la pantalla
    while (Serial.available() > 0) //mientras haya digitos por leer
    {
      lcd.write(Serial.read()); // escribimos caracter a caracter lo que le mandamos
    }
  }
}
```

## 🚦 Código en Python

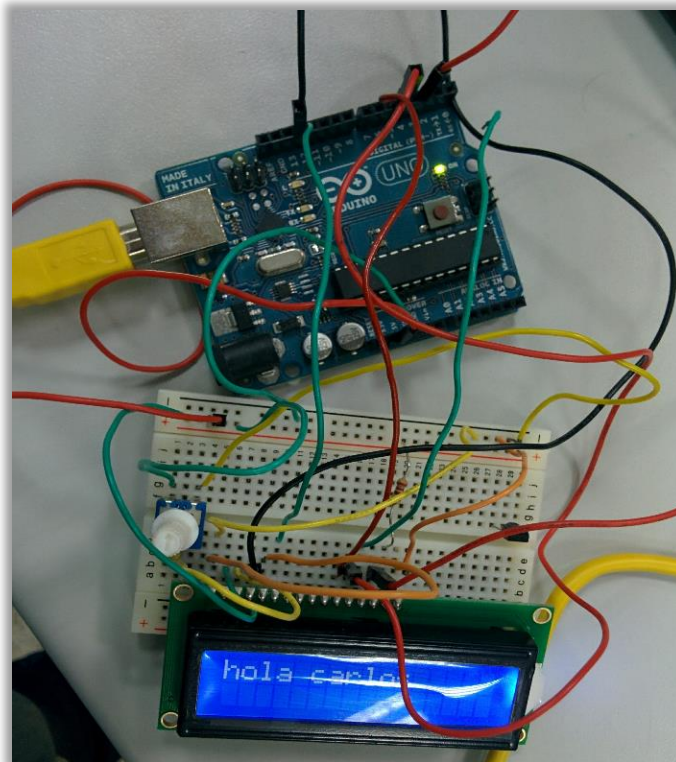
```
import serial

arduino = serial.Serial('/dev/ttyACM0', 9600)

print("COMENZANDO")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    print " TEXTO : ",comando
arduino.close() #Finalizamos la comunicacion
```

## 🚦 Resultado final





## 1.11 Sensor de temperatura monitorizado



### Información sobre el sensor de temperatura TMP36GZ

El TMP36 es el sustituto del LM335A y es un sensor de temperatura analógico muy popular y sencillo de utilizar. Funciona como un diodo Zener con un voltaje de corte proporcional a la temperatura absoluta con un rango de 10mV/°K. Conecta una resistencia desde 5V a GND y el sensor te dará en su salida un voltaje que podrás medir con el ADC de tu microcontrolador favorito y también con Arduino. La salida del sensor es lineal por lo que no tendrás que hacer cálculos de conversión. Puede operar de -40°C a 100°C.

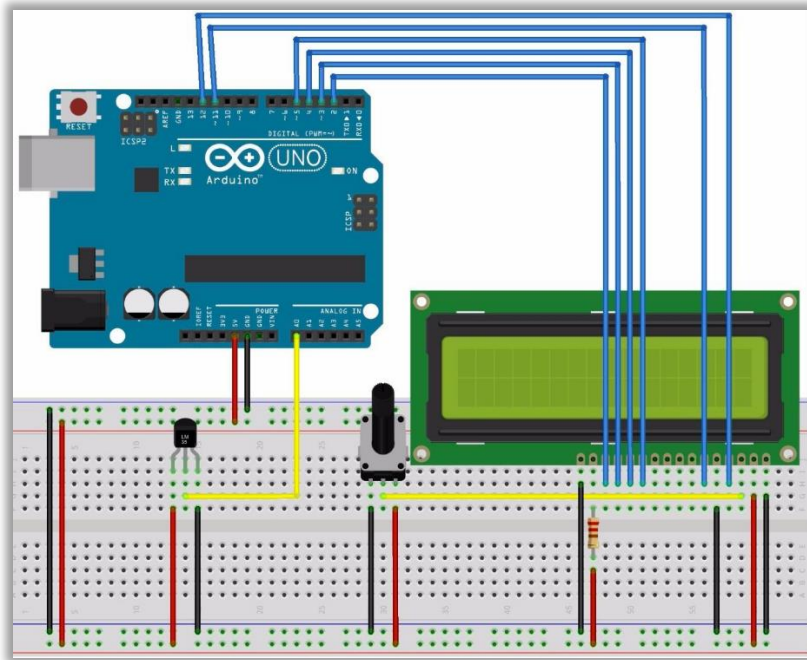


### Desarrollo

En este ejercicio vamos a utilizar el sensor de temperatura TMP36GZ del que recibiremos los datos (en grados centígrados) y lo imprimiremos en la pantalla LCD. Éste sensor consta de 3 pines, de izquierda a derecha: fuente (+5V), salida analógica del sensor y tierra (GND).

En el ejercicio anterior aprendimos a enviar un texto a la pantalla LCD por nuestro puerto serie. En este ejercicio, la complejidad será mayor ya que no recibiremos una cadena de texto. Nuestro sensor de temperatura nos enviará los datos a través de una señal analógica por lo que tendremos que realizar algunas operaciones para que dado el valor recibido por el pin analógico del sensor (el 0 en nuestro caso), que se encuentra en el rango 0-1023, se convierta a grados centígrados (°C). Para ello, nos crearemos una pequeña función (la llamaremos `valorGradosCent()`) que convierta el valor del sensor, recibirá el valor del pin analógico 0 y transformará el valor a grados centígrados.

Para finalizar el ejercicio, mostraremos las conexiones necesarias en nuestra placa de Arduino para que el programa funcione correctamente.



#### Código en Arduino

```

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int sensorPin = A0; // Pin A0;

float valorGradosCent() {
  int dato;
  float resultado;
  dato= analogRead(sensorPin);
  resultado=(5000.0 * dato)/1024;
  resultado=(resultado-500)/10;
  return resultado;
}

void setup(){
  Serial.begin(9600);
  lcd.begin(16,2);
}

void loop() {
  lcd.setCursor(0,0);
  int sensorVal = analogRead(sensorPin);
  Serial.print("Sensor Value: ");
  Serial.print(sensorVal);
  float voltage = (sensorVal/1024.0) * 5.0;
  Serial.print(", Voltios: ");
  Serial.print(voltage);
  lcd.setCursor(0,1);
  lcd.write("Temperatura: ");
  float temperature = (voltage - .5) * 100;
  lcd.print(temperature);
  lcd.setCursor(5,1);
  lcd.write((char)223);
  lcd.setCursor(6,1);
  lcd.write("C");
  delay(2000);
}

```



## 1.12 Contador en display 7-segmentos



### Información sobre los display 7-Segmento

El visualizador de siete segmentos (llamado también *display* en inglés) es una forma de representar caracteres en equipos electrónicos. Está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea.

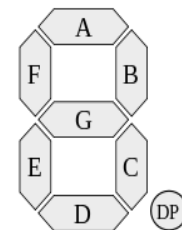


### ¿Cómo funciona un display 7-Segmento?

El visualizador de 7 segmentos es un componente que se utiliza para la representación de caracteres (normalmente números) en muchos dispositivos electrónicos, debido en gran medida a su simplicidad. Aunque externamente su forma difiere considerablemente de un led típico, internamente están constituidos por una serie de ledes con unas determinadas conexiones internas, estratégicamente ubicados de tal forma que forme un número '8'.

Cada uno de los segmentos que forman la pantalla están marcados con siete primeras letras del alfabeto ('a'-'g'), y se montan de forma que permiten activar cada segmento por separado, consiguiendo formar cualquier dígito numérico. A continuación, se muestran algunos ejemplos:

- Si se activan o encienden todos los segmentos se forma el número "8".
- Si se activan sólo los segmentos: "a, b, c, d, e, f," se forma el número "0".
- Si se activan sólo los segmentos: "a, b, g, e, d," se forma el número "2".
- Si se activan sólo los segmentos: "b, c, f, g," se forma el número "4".



Muchas veces aparece un octavo segmento denominado *dp*. (del inglés *decimal point*, punto decimal).

Los ledes trabajan a baja tensión y con pequeña potencia, por tanto, podrán excitarse directamente con puertas lógicas. Normalmente se utiliza un codificador (en nuestro caso decimal/BCD) que, activando una sola pata de la entrada del codificador, activa las salidas correspondientes mostrando el número deseado. Recordar también que existen pantallas alfanuméricas de 16 segmentos e incluso de una matriz de 7\*5 (35 bits).

Los hay de dos tipos: ánodo común y cátodo común. En los de tipo de ánodo común, todos los ánodos de los ledes o segmentos están unidos internamente a una patilla común que debe ser conectada a potencial positivo (nivel “1”). El encendido de cada segmento individual se realiza aplicando potencial negativo (nivel “0”) por la patilla correspondiente a través de una resistencia que límite el paso de la corriente.

En los de tipo de cátodo común, todos los cátodos de los ledes o segmentos están unidos internamente a una patilla común que debe ser conectada a potencial negativo (nivel “0”). El encendido de cada segmento individual se realiza aplicando potencial positivo (nivel “1”) por la patilla correspondiente a través de una resistencia que límite el paso de la corriente.

Los segmentos pueden ser de diversos colores, aunque el visualizador más comúnmente utilizado es el de color rojo, por su facilidad de visualización. También existen pantallas alfanuméricas de 14 segmentos que permiten representar tanto letras como números. El visualizador de 14 segmentos tuvo éxito reducido y solo existe de forma marginal debido a la competencia de la matriz de 5 x 7 puntos.

Si bien hoy este tipo de visualizadores parecen antiguos u obsoletos, ya que en la actualidad es muy común el uso de pantallas gráficas basadas en píxeles, el visualizador de 7 segmentos sigue siendo una excelente opción en ciertas situaciones en las que se requiera mayor poder lumínico y trabajo en áreas hostiles, donde las pantallas de píxeles podrían verse afectadas por condiciones ambientales adversas. Aún no se ha creado otro dispositivo de señalización que reúna características como este en cuanto a potencia lumínica, visualización a distancia, facilidad de implementación, bajo costo y robustez.

## Desarrollo

A nivel lógico tendremos que implementar una serie de funciones para poder comunicarnos con nuestro display, una función para que nos transforme nuestro valor digital en valor hexadecimal y otra función para escribir ese valor en él.

A la primera función la llamaremos *‘write\_data’*. Es un switchcase en el que por cada valor decimal le atribuiremos su valor hexadecimal.

## Código en Arduino

```
void write_data (int arg) {
  switch (arg) {
    case 0:
      write7seg(0x7e);
      break;
    case 1:
      write7seg(0x30);
      break;
    case 2:
      write7seg(0x6d);
      break;
    case 3:
      write7seg(0x79);
      break;
    case 4:
      write7seg(0x33);
      break;
    case 5:
      write7seg(0x5b);
      break;
    case 6:
      write7seg(0x1f);
      break;
    case 7:
      write7seg(0x70);
      break;
    case 8:
      write7seg(0x7f);
      break;
    case 9:
      write7seg(0x73);
      break;
  }
}
```

En cuanto a la función para escribirlo en el display la llamaremos '*write7seg*', y cuyo código tenemos a continuación.

```
void write7seg (unsigned char arg) {
  unsigned char segmen = 0x01;
  unsigned char display1;
  display1 = arg;

  for (int i = 0; i < 8; i++) {
    if ((display1 & segmen) == 0x00)
      digitalWrite(i, LOW);
    else
      digitalWrite(i, HIGH);
    segmen <<= 1; }
}
```

En resumen, la función anterior usa una máscara selectora de un solo bit en la que el parámetro '*segmen*' lo vamos desplazando a la izquierda para que obtener el bit que queremos, es decir, en el pin 4 necesitaríamos el valor que hay en el cuarto dígito del valor de entrada, así hasta llegar al último bit (recordemos que 8 bits es el tamaño del dato que le damos). Después hacemos un if-else, que escribe un 0 lógico si es 0 la máscara, y un alto lógico si no lo es.

Ya sólo nos queda realizar una función que nos vaya seleccionando alternadamente los dos displays de modo que no escribamos las decenas y las unidades a la vez en el mismo.

Para ello necesitaremos una frecuencia adecuada que cumpla que los cambios se hagan cada segundo, que podamos escribir en los dos displays sin notar nada raro y que evitemos los parpadeos. Esta función será nuestro '*refresh*':

```
void refresh( int data1, int data0) {
    int j=40; //se toma este valor para crearnos una frecuencia que evite el parpadeo
    int tiempo_refresco = tiempototal/(2*j);
    int i;
    for(i=0;i<j;i++)
    {
        delay(tiempo_refresco);
        digitalWrite(dis1, 1);
        digitalWrite(dis2, 0);
        write_data(data1);
        delay(tiempo_refresco);
        digitalWrite(dis1, 0);
        digitalWrite(dis2, 1);
        write_data(data0);
    }
}
```

Para terminar nuestro código, nos quedaría inicializar las variables en el '*setup*' y escribir en '*loop*' un bucle para que nos genere los datos que queremos. Nuestro código definitivo quedará de la siguiente manera:

```
int segPins[] = { 0, 1, 2, 3, 4, 5, 6, 7};
int tiempototal= 1000;
int j = 40;
int disp1 =8;
int disp2= 9;
int dat1 = 0;
int dat0 = 0;
int tiempoMax = 60;

void write_data (int arg) {
    switch(arg) {
```

```
case 0:
write7seg(0x7e);
break;

case 1:
write7seg(0x30);

break;
case 2:
write7seg(0x6d);
break;
case 3:
write7seg(0x79);
break;
case 4:
write7seg(0x33);
break;
case 5:
write7seg(0x5b);
break;
case 6:
write7seg(0x1f);
break;
case 7:
write7seg(0x70);
break;
case 8:
write7seg(0x7f);
break;
case 9:
write7seg(0x73);
break;
}
}

void write7seg(unsigned char arg) {
    unsigned char segmen = 0x01;
    unsigned char display1;
    display1 = arg;
```

```

for(int i=0; i<8;i++) {
    if((display1 & segmen) == 0x00)
        digitalWrite(i, LOW);
    else
        digitalWrite(i, HIGH);
    segmen <<=1;
}
}

void refresh (int data1, int data0) {
    int j=40;
    int tiempo_refresco = tiempototal/(2*j);
    int i;

    for(i=0; i<j;i++) {
        delay(tiempo_refresco);
        digitalWrite(dis1, 1);
        digitalWrite(dis2, 0);
        write_data(data1);

        delay(tiempo_refresco);
        digitalWrite(dis1, 0);
        digitalWrite(dis2, 1);
        write_data(data0);
    }
}

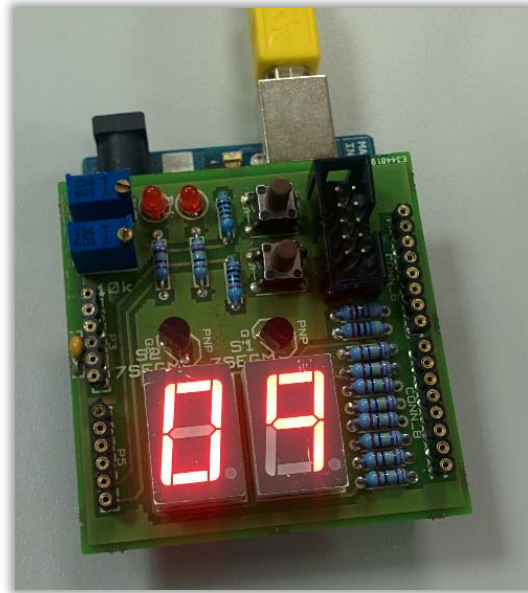
void setup() {
    for (int thisseg = 0; thisseg < 8; thisseg++) {
        pinMode(segPins[thisseg], OUTPUT);
    }
    pinMode(dis1, OUTPUT);
    pinMode(dis2, OUTPUT);
}

void loop() {
    int valor;
    while(1) {
        for (valor = 0; valor < tiempoMax; valor++) {
            dat1 = valor / 10;

```

```
    dat0 = valor % 10;
    refresh(dat1, dat0);
}
}
}
```

Resultado final



Durante la realización de este ejercicio, el profesor nos sugirió otras funcionalidades, las cuales tendríamos que hacer modificaciones en nuestro código. Son las siguientes:

- 1) ¿Qué modificaciones habría que hacerle al código para que los segundos pasasen más rápidos?

Para que los segundos pasen más rápido, es decir, menos de 1 segundo, simplemente hay que aumentar el factor multiplicador de j, por ejemplo  $4*j$  de la siguiente sentencia:

```
int tiempo_refresco = tiempototal/(2*j); //El contador va de 1 segundo en 1 segundo
```

Si quisieras que los segundos pasen más lentos, sólo tendríamos que disminuir el factor multiplicador de j, así el tiempo de refresco disminuirá.

- 2) ¿Qué modificaciones habría que hacerle al código para que el contador comenzase en 90 y terminase en 100?

Dentro de la función loop() encontramos el siguiente for:

```
for (valor = 0; valor < tiempoMax; valor++) {
```

Dentro del for, sólo habría que modificar valor = 0 por valor = 90, e irnos al principio del código y poner la variable tiempoMax = 100.

### 1.13 Conclusión

Durante los ejercicios propuestos para la plataforma de Arduino hemos aprendido una infinidad de cosas muy interesantes, desde utilizar el IDE de Arduino hasta programar pequeños códigos en lenguaje Python. Los ejercicios más interesantes fueron los del giro del motor mediante el potenciómetro, mostrar la temperatura en el display LCD y el contador.